Machine Learning based scratches on printed paper detection, in high-speed printing systems

Universitat Oberta de Catalunya
Màster Universitari en Enginyeria Informàtica
Treball Final de Màster - Intel·ligència Artificial

Professor responsable de l'assignatura: Carles Ventura Royo
Consultor: Antonio Burguera Burguera
Alumne: Jordi Falcés i Valls

Idioma: Anglès

Desembre de 2021

Final work sheet

Title:	Machine Learning based scratches on printed paper detection, in high-speed printing systems
Author:	Jordi Falcés i Valls
Consultant:	Antonio Burguera Burguera
Accountable professor:	Carles Ventura Royo
Delivery date:	12/2021
Academic program:	Màster Universitari en Enginyeria Informàtica Universitat Oberta de Catalunya
Area of knowledge:	Artificial Intelligence
Language:	English
Keywords:	Machine Learning, scratches detection, printed paper, high-speed, printing systems, dataset creation, data augmentation

Abstract:

Printing industry rapidly is adopting digital technologies and the requirements in terms of speed and print quality are also becoming more demanding. The is a wide range of possible quality defects in printed paper. This makes it impossible to have humans inspect the printed paper for such a big amount of possible quality defects at the high-speeds the printouts are produced.

Printing industry is not taking advantage of the Artificial Intelligence to detect defects in printed paper at speed without human intervention. It is possible to generate millions of images (captures) with printed content from a printing system every day. Most of these images will not have any defect but some other will and can be used to generate a data set to be used in a machine learning system.

The intention of this research work is to find ways artificial intelligence can help on automatically detecting defects on printed paper in a printing system and classifying them, without human intervention. Focusing on scratches, I've explored what are the actual proposals and solutions, and how machine learning can help improving them by using datasets with different techniques, implementing possible solutions and comparing the obtained results.

Index

Final work sheet 3

Index 4

List of figures and tables 7

Master's Final Work Report 8

1. Introduction 9

1.1. Context and justification 9

1.2. Objectives 11

1.3. Approach and methodology 12

1.4. Planning 13

1.5. Summary of obtained products 14

1.6. Brief description of the next sections 14

2. Research of current approaches/methods 15

2.1. High-speed Defect Detection Method for Color Printer Matter 15

2.2. Color-Defect Classification for Printed-Matter Visual Inspection System 16

2.3. An automated defect classification algorithm for printed documents 16

2.4. An Automation System for High-Speed Detection of Printed Matter and Defect Recognition 17

2.5. A real-time print-defect detection system for web offset printing 18

2.6. Learning from Imbalanced Data 19

2.7. Automatic visual inspection and defect detection on Variable Data Prints 20

2.8. Learning from small datasets containing nominal attributes 22

2.9. A Deep Learning Based Printing Defect Classification Method with Imbalanced Samples 23

2.10. Fabric Defect Detection System Using Stacked Convolutional Denoising Auto-Encoders Trained with Synthetic Defect Data 23
2.11. Surface Defect Detection: Dataset & Papers 24
2.12. Image Anomaly Detection Using Normal Data Only by Latent Space Resampling 26
2.13. Partial conclusions of research 27
3. Datasets 29
3.1. Color 64x64px tiles 32
3.2. Color 320x320px tiles 33
3.3. Converting tiles to grayscale 35
3.4. Human selection of “scratched” and “not scratched” tiles 38
3.5. The numbers 39
3.6. Balancing the dataset 40
3.7. Creating the final datasets 41
3.8. Augmentation 42
5. Experiments 44
5.1. Experiment 1: 320x320px Color Balanced without Augmentation 46
5.2. Experiment 2: 320x320px Color Balanced with Augmentation 47
5.3. Experiment 3: 320x320px Color Imbalanced without Augmentation 49
5.4. Experiment 4: 320x320px Color Imbalanced with Augmentation 51
5.5. Experiment 5: 320x320px Grayscale Balanced without Augmentation 53
5.6. Experiment 6: 320x320px Grayscale Balanced with Augmentation 55
5.7. Experiment 7: 320x320px Grayscale Imbalanced without Augmentation 57
5.8. Experiment 8: 320x320px Grayscale Imbalanced with Augmentation 58
5.9. Experiment 9: 64x64px Color Balanced without Augmentation 61
5.10. Experiment 10: 64x64px Color Balanced with Augmentation 62
5.11. Experiment 11: 64x64px Color Imbalanced without Augmentation 64

5.12. Experiment 12: 64x64px Color Imbalanced with Augmentation 66
5.13. Experiment 13: 64x64px Grayscale Balanced without Augmentation 68
5.14. Experiment 14: 64x64px Grayscale Balanced with Augmentation 69
5.15. Experiment 15: 64x64px Grayscale Imbalanced without Augmentation 71
5.16. Experiment 16: 64x64px Grayscale Imbalanced with Augmentation 73
5.17. Comparison of experiment results 75
5.20. Experiment conclusions 78
6. Conclusions 79
7. Glossary 84
Bibliography and consulted references 86

Appendix 1. Sample of the dataset with and without scratches 89
Appendix 2. Code to balance the datasets 97
Appendix 3. Code to create the datasets and the CSV files for Scratched and NotScratched 102
Appendix 4. Samples of augmented datasets 110
Appendix 5. Hardware used during the experiments 115
Appendix 6. Code to prepare the datasets and for chars of Experiments 118
Appendix 7. Full outputs of Experiments 144

List of figures and tables

Figure 1 Master's Final Work general planning 13
Figure 2 Master's Final Work Gantt chart 14
Figure 3 HP PageWide Web Press T250HD 29
Figure 4 BMP RGB images captured by the Vision System 30
Figure 5 Structure of the final dataset 41
Figure 6 Comparison summary of experiment charts 76
Figure 7 Tiles with subtle scratch (LEFT) vs. images without scratch (RIGHT) 80

Table 1 Master's Final Work general milestones and deliverables 13
Table 2 Sizes of Scratches partial datasets 39
Table 3 Sizes of Scratches balanced datasets 40
Table 4 Experiments to be executed in Phase 2 44
Table 5 Summary of experiments 77

Master's Final Work Report

The additional files related to this Master's Final Work report can be downloaded from:

Google Drive

(The link will only work for members of the "Fundació per a la Universitat Oberta de Catalunya")

The content of the folder is:

datasets (folder): Original BMP files captured by the Vision System.

datasetsPhase1.zip: Contains the final datasets used in the experiments phase.

Google Colab Notebook.ipynb: The Google Colab Notebook with the Python code to create the Datasets and run the Experiments

The next sections contain the development of the Master's Final Work Report. It has been divided into five different parts or phases.

- **Introduction:** Technical aspects of the Master's Final Work. This is the formal part of the report, detailing the context and justification, objectives, approach and methodology, planning, etc.
- **Research of current approaches/methods:** A state of the art on defect detection and classification for printing systems to aid the design of experiments.
- **Creation of the datasets:** This is half of the work in this Master's Final Work, as it was not found any valid dataset to experiment with and train machine learning systems.
- **Experiments:** The design of experiments and the experiments themselves, together with partial conclusions.
- **Conclusions:** Final conclusions of the Master's Final Work.

1. Introduction

1.1. Context and justification

I have been working at HP for more than 15 years. I started working for the Large Format division and I'm currently developing my work activity in the PageWide Web Press Industrial division, where I support HP PageWide Web Presses as a Customer Assurance Master Engineer.

The HP PageWide Web Presses are high-speed presses that are intended to print on paper reels (both coated and uncoated) with very high-quality, at very high-speeds that can actually reach up to 1,000 feet per minute.

As a Customer Assurance Engineer, I'm quite concerned about Print Quality defects in the printouts, but not only about the defects produced by the ink on the paper. There are other defects that can be observed in the paper, caused by the fact that a press passes the paper through the printing zones of the press, dryers, and other modules that are intended to re-moisture overdried paper, apply silicone to avoid frictions at the finishing equipment, etc.

Our current products have a Vision System that captures some of the printed content (we call it frames) and, based on special marks that we print on the paper, can report some of the defects (mostly streaking). A more recent implementation in that Vision System, compares the ripped image (PDF file separated into color planes) with the actual printed content, but it is currently only capable of highlighting streaking.

Wouldn't it be interesting to have a system that could learn from the differences between what should have been printed and what is really printed? Wouldn't it be interesting that this same system could classify the defects so it could advice the operator on what to do next to resolve them?

While printing industry is adopting digital technologies, the requirements in terms of speed and print quality are also becoming more demanding.

Quality defects in printed paper can include, but not only, missing content, streaking, color plane misalignment, spray, decap, picking, coalescence, offsetting, banding, ink transfer, ink dripping, ink

starvation, scratching, ink smearing, sharpness, wrinkling, etc. Some of these defects are print defects, and some other are side effects of the printing process. This makes it impossible to have humans inspect the printed paper for such a big amount of possible quality defects at the high-speeds the printouts are produced.

In digital environments, it's relatively easy to compare what should have been printed versus what was really printed, maybe using a computer vision system. This doesn't require learning, but to detect these defects based on rules and comparisons.

The printing industry is not taking advantage of the Artificial Intelligence to detect defects in printed paper at speed without human intervention.

It should be possible to generate millions of images (captures) with printed content from a printing system every day. Most of these images will not have any defect but some other will and can be used to generate a data set to be used in a machine learning system.

Automation would not only avoid human errors but also a tedious and repetitive job.

Once the defects are classified, they can be associated to corrective actions or guidance, so the operator (or even the printer itself) can resolve them.

There are a lot of possible defects associated to the fact a paper is printed by a printing system. When printing at low speeds and when there is low number of printouts to inspect a person can look at them and report and even resolve the root cause of the problem, sometimes while printing.

This is more challenging when the printing system prints at high-speeds, like in a web press. The printed paper is printed at very high-speeds and it's rolled to a winder or even processed through a finishing equipment (like a book binder). In this case, the chances an operator can find a defect are very limited unless they are assisted by a vision system.

A vision system in a printing system uses to compare what should have been printed with what has really been printed. If the file image doesn't match the captured image, then it may highlight that difference.

What these systems do is to sample the printed content, so it may happen that a defect is not caught because its area has not been captured by the vision system to be inspected.

There are several papers that relate artificial intelligence methods to defects on substrates or surfaces of different materials and properties. Only a few are directly related to defects on printed paper because of the printing, or on the side effects of that printing.

There is even less literature related to defects associated to printing systems that print at high-speeds.

The intention of this research work is to find ways artificial intelligence can help on automatically detecting defects on printed paper in a printing system and classifying them, without human intervention. I will explore what are the actual proposals and solutions, and how machine learning can help improving them by using datasets with different techniques, implementing possible solutions and comparing the obtained results.

The title of this Master's Final Work is:

> *Machine Learning based scratches on printed paper detection, in high-speed printing systems*

The title expresses:

- A Machine Learning uses a set of given algorithms to obtain data, learn from that data, and decide based on these learnings.
- An Automated Visual Detection System using Machine Learning can help prescinding from human intervention to visually detect defects (scratches) on printed paper.
- We are looking for a valid solution that would work in high-speed printing systems.

1.2. Objectives

The initial objectives of this Master's Final Work are:

- Find actual artificial intelligence-based solutions or methods to detect and classify defects on printed content, in printing systems running at high-speed, or actual solutions or methods in other kind of surfaces that could be leveraged.
- Find areas of improvement or specificity for printing systems running at high-speed.
- Choose datasets and techniques to implement a proposal of solution or method and compare the obtained results.
- Propose a solution or method to be implemented in high-speed printing systems.
- Determine limitations to a viable solution.

What it is, and what it is not?

- It is intended to find actual situation of the matter and propose improvements.

- It is intended to provide an approach of a solution or a method that could be leveraged in an implementation project.

- It is not intended to provide a full implementation for a real-world printing system.

1.3. Approach and methodology

This Final Master's Work consists in three main phases that define the approach and used methodology:

- **Research of current approaches/methods:** A state of the art on defect detection and classification for printing systems to aid the design of experiments.
- **Creation of the datasets:** This is half of the work in this Master's Final Work, as it was not found any valid dataset to experiment with and train machine learning systems.
- **Experiments:** Use the new dataset to design a model that combines different techniques and dataset modifications (color vs. grayscale dataset, different image size in dataset, imbalanced vs. balanced dataset, enhanced datasets using augmentation) together with TensorFlow Keras machine learning and high-level neural network libraries to detect scratches in printed paper with high-accuracy enough.

After every phase, the conclusions will help defining the next phase, depending on the findings and results.

Having good valid datasets is key for this work. This will require high dedication to create is from zero, as no good enough dataset has been found in the field.

The experimentation phase will use the newly created datasets, with different modifications, to train a model with the purpose of finding the best dataset that would produce better accuracy on scratch detection, considering the cost (resources and time) of creating that dataset and the cost (resources and time) to train the model for that accuracy.

This approach and methodology will help responding questions like:

- Will the dataset and method be good enough to be used in a high-speed printing system?
- What are the limitations and constrains?

1.4. Planning

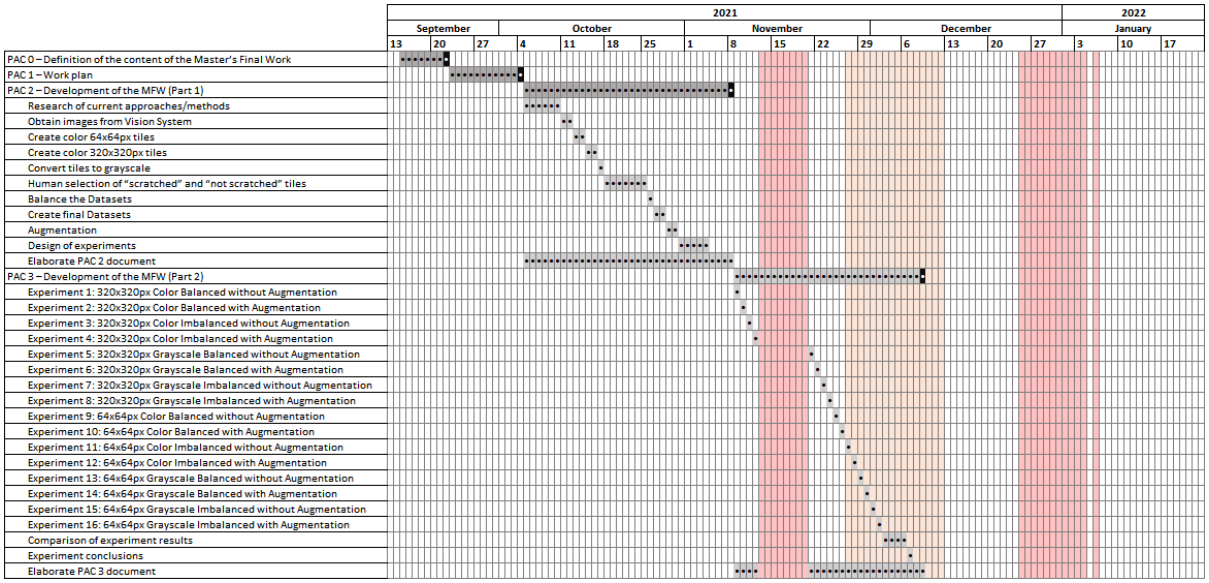
This Master's Final Work has been divided into 6 parts, all of them with a deliverable.

Figure 1 Master's Final Work general planning

Milestone/Deliverable	Date
PAC 0 – Definition of the content of the Master's Final Work	22/09/2021 (DONE)
PAC 1 – Work plan	04/10/2021 (DONE)
PAC 2 – Development of the MFW (Part 1)	08/11/2021 (DONE)
PAC 3 – Development of the MFW (Part 2)	09/12/2021 (DONE)
PAC 4 – Writing of the MFW report	24/12/2021 (DONE)
PAC 5a – Creation of presentation	03/01/2022[1]
PAC 5b – Public presentation	21/01/2022[2]

Table 1 Master's Final Work general milestones and deliverables

I've created a Gantt chart of the entire project, with the planification of each task, important milestones, and identified dates that can put in risk the project, according to the previous data.



(Continues in the next page)

[1] The creation of the presentation is the next deliverable. It's not yet done at the time this report has been published but it's work in process.

[2] The Public presentation has not yet been scheduled but the date in this report is the last day to be done.

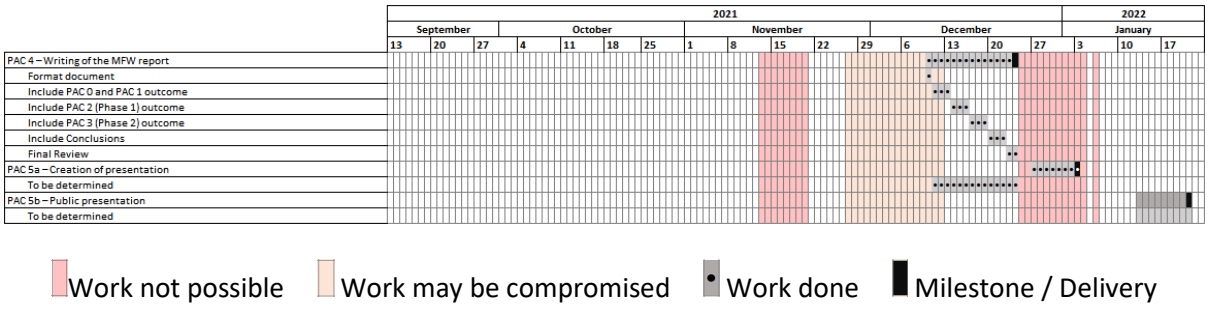


Figure 2 Master's Final Work Gantt chart

1.5. Summary of obtained products

The outcome of the Master's Final Work in hand is this report itself, with the research, creation of the datasets, the experiments and the conclusions, together with the Google Colab Notebook, the images to create the datasets and the final datasets, so the Google Colab Notebook can be executed.

1.6. Brief description of the next sections

The next sections focus on the development of the work itself.

- **2. Research of current approaches/methods:** A state of the art on defect detection and classification for printing systems to aid the design of experiments.
- **3. Creation of the datasets:** This is half of the work in this Master's Final Work, as it was not found any valid dataset to experiment with and train machine learning systems.
- **4. Experiments:** The design of experiments and the experiments themselves, together with partial conclusions.
- **5. Conclusions:** Final conclusions of the Master's Final Work.

2. Research of current approaches/methods

This sub-section is the result of a research of current (previous) approaches and methods to detect and classify printing defects. Some of them are related to machine learning, but some other sources propose other ways to automatically detect and classify defects (maybe not directly related to the printing industry) but that will help shape this work.

NOTE: The next papers have been sorted by date of publication. The full details of each source have been included to the section *Bibliography and consulted references*.

2.1. High-speed Defect Detection Method for Color Printer Matter

Month and year of publication: November 1990

Author/s: Katsuyuki Tanimizu, Shin'ichi Meguro, Akira Ishii

Information of interest for my research:

- Defects produce **customer complaints** and may require complete **re-printing**. Products generally **visually inspected by humans** (accuracy fluctuates, defect are overlooked, speed is limited). Strong **demand for automatic inspection** machine.
- The paper presents a defect detection method based on **image processing technologies for automatic inspection of color printed matter, at high-speed using human-like judgment criteria**. Classified into shape-defects (this paper) and chromatic-defects. Targets pre-paid cards as a substrate to be inspected (small surface).
- It's not Artificial Intelligence related.

2.2. Color-Defect Classification for Printed-Matter Visual Inspection System

Month and year of publication: June 2002

Author/s: Ichirou Ishimaru, Seiji Hata, Masayoshi Hirokari

Information of interest for my research:

- The paper establishes the automatic-identification method of printing-troubles without re-inspection by worker for taking appropriate countermeasures on the fly. Proposed a classification method that recognizes **shape-defects** and discriminates **color-difference**. Method for **offset printing**.
- It's not Artificial Intelligence related.
- If images are **black and white**, then color cannot be assessed for defects.
- Identifies **color-defects** by calculating color difference between defect region and surrounding region.
- Vary small dataset used.
- **Shape-defect** classification **accuracy** is **93%**.
- **Color-defect** classification **accuracy** is **90.5%**.

2.3. An automated defect classification algorithm for printed documents

Month and year of publication: January 2006

Author/s: Onome Augustine Ugbeme, Eli Saber, Wencheng Wu

Information of interest for my research:

- **Print engines** are required to produce **consistent and stable image quality** as measured by various **metrics** and ultimately **evaluated by customers**. Market demands the best image quality from suppliers at competitive costs with minimum downtime. However, these systems possess a variety of image quality defects (from faults and degradations) that present in a variety of **shapes** and **sizes** that can occur in different **locations** of the printout.
- **Operator intervention** may be required to perform corrective actions to eliminate or reduce the occurrences of defects.

- It is vital to **automate the defect detection process** to minimize downtime. This will produce higher profit margins.
- The paper proposes a **method for automatically identifying image quality defects** on printed documents that accepts scanned images containing the defect and employs a 3-stage identification approach.
- It's not Artificial Intelligence related.
- The proposed **algorithm** consists in pre-processing, global thresholding, localized thresholding and classification.
- Vary small dataset used.
- Total **accuracy** is claimed to be **80.5%**.

2.4. An Automation System for High-Speed Detection of Printed Matter and Defect Recognition

Month and year of publication: March 2007

Author/s: Yang Ou, Hu Tao, Guo Xuan, Guo Baoping

Information of interest for my research:

- Automated visual inspection of industrial goods for quality control plays an ever-increasing role in production process as the **global market pressures put higher and higher demand on quality**.
- The **quality inspection** through visual inspection is still **carried out by humans**.
- **Human visual inspection** has **accuracy fluctuations**, **overlooked defects**, l**imited inspection speed**, etc.
- **Defects cause waste in print material**.
- Strong demand for an automated inspection machine.
- Comparing reference and printed image. If the difference between the images exceed a given limit, it's flagged as having a defect.
- The design of the system has to consider the **speed of printing**, the **area to be inspected**, the **inspection accuracy**, the **characteristics of the CPU**, the **kind of printed matter**, etc.

- The paper proposes an **automation system for high-speed detection of printer matter and defect recognition**, such as smudges, doctor streaks, pin holes, character misprints, foreign matters, hazing, wrinkles, etc.
- It's not Artificial Intelligence related.
- It uses CCD cameras to collect the image of the printed matter, then the images are processed by grabbing cards with FPGA/DSP, and the computer compares the reference image with the printed image and classifies the defects of printed matter.
- The system uses a multi-camera and multi-computer structure that are synchronized and do parallel processing to different regions, data management and statistics.
- **Every pixel of the reference image is compared to the corresponding pixel of the captured image**. The printed matter material is slightly distorted or rotated, not perfectly matching the reference image. The algorithm eliminates defects at the edge of the patterns in the difference image with the reference image and the inspection image. Flaws are all found and defects are eliminated.

2.5. A real-time print-defect detection system for web offset printing

Month and year of publication: November 2008

Author/s: N. G. Shankar, N. Ravi, S. W. Zhong

Information of interest for my research:

- **Quality and reliability** are **essential** requisites in modern printing machine systems.
- Printing houses don't want to **waste print materials**.
- **Not much research** has been conducted on the on-line defect detection of web offset printing using computer vision techniques.
- The paper presents a description of a **vision system** developed to **detect and locate the non-uniformities** (structural faults, color variations, missing characters, ink splashes, streaks, etc.) that appear on the web of offset printing machines. This includes hardware modules and image processing algorithm to detect edges and boundaries using linear and non-linear filters of dynamic size, threshold and transformation for further analysis.
- It's not Artificial Intelligence related.

- A **real-time print-defect detection system** require high-resolution progressive scan CCD cameras to capture print images that are digitalized using a digital frame grabber for analysis in PC, and work by comparing two images (golden master or reference image vs. actual print image) during the identification of faults. These images have to be in the same position during comparison. The reference image is stored and then compared with the actual print image acquired on regular basis by the cameras.

2.6. Learning from Imbalanced Data

Month and year of publication: September 2009

Author/s: Haibo He, Edwardo A. Garcia

Information of interest for my research:

- Concern with the performance of learning algorithms in the presence of **underrepresented data and severe class distribution skews** that **significantly compromises the performance of learning algorithms** that assume or expect balanced class distributions or equal misclassification costs. Algorithms fail to properly represent the distributive characteristics of the data and resultantly provide unfavorable accuracies across the classes of the data.
- The problem is even bigger when, in addition to **imbalanced data**, the **sample size is small**. Learning algorithms often fail to generalize inductive rules over the sample space. The induction rules that describe the minority concepts are often fewer and weaker than the majority concepts since the minority class is often both outnumbered and underrepresented.
- The paper offers a comprehensive review of the development of research in learning from imbalanced data.
- **Oversampling** appends replicated data to the original dataset. Multiple instances of certain examples can cause overfitting because classifiers produce multiple clauses in a rule for multiple copies of the same example that causes the rule to become too specific. Even the training accuracy will be high in this scenario, the classification performance on the unseen testing data is generally far worse.

- **Undersampling** removes data from the original dataset. To randomly select a subset of the majority class examples and remove these samples from the original dataset. Method to adjust the balance of the original dataset. Removing examples from the majority class may cause the classifier to miss important concepts pertaining to the majority class.
- **Synthetic Sampling with Data Generation** is presented as a powerful method that has shown great success. It creates artificial data based on the feature space similarities between existing minority examples.

2.7. Automatic visual inspection and defect detection on Variable Data Prints

Month and year of publication: June 2010

Author/s: Marie Vans, Sagi Schein, Carl Staelin, Pavel Kisilev, Steven Simske, Ram Dagan, Shlomo Harush

Information of interest for my research:

- Having **operators to inspect printouts** off the press implies having them **occupied in single machine**.
- Some customers require **100% visual inspection**.
- **Human intervention is about 80% effective**, and only when the inspection process is highly structured and repeatable.
- **Offline inspection** cause **paper waste**.
- **Inspection systems for variable print data do not exist**.
- Defects in textiles fall into well-defined categories, which make it less difficult to develop defect-specific detection algorithms.
- This paper proposes a system for **automatic, on-line visual inspection and print defect detection for Variable Data Printing (VDP)**, so it has the ability to change the reference on every print. The system can automatically stop de printing process and alert the operator of problems.
- It's not Artificial Intelligence related.
- Uses an **on-line scanner that captures the entire print** and sends the image data through a specialized image processing board and then onto the main defect detection system where it is compared with the reference image.

- The **scanner** may introduce **artificial differences** between the images.
- Stretching, skew, paper, blanket, writing heads, vibrations on the press got gain, illumination and reflection, etc. can **contribute to difficult the detection of defects**.
- Both the reference and the scanned **images need to be pre-processed** (noise removal, blurring, etc.).
- The requirement is to be able to perform defect detection as the press prints at a speed of 2 meters per second (120 meters per minute / ~394 feet per minute). At that speed, the system must acquire both the reference and printed image, perform various image pre-processing activities, and determine whether a defect exists (in real time). **Objectives** are **high detection of most of the visible defects** and **low false alarm rates**.
- **Image reference approach:** A reference image exists. Inspection of the 100% of the potentially defective unit. Much more reliable approach.
- **Design-Rule approach:** As little as 10% of a product can be inspected before generating the appropriate statistics and determining whether a defect exists.

- **OF SPECIAL INTEREST:**[3]
    - **Scratches** are sometimes **difficult to detect with general purpose detection methods** because of the typical low-contrast nature of these type of defects. Have very low signal-to-noise ratio.
    - Missed 27% of the defects present in 454 prints and **the majority of** these **missed defects were scratches**. Suggestion to use a dedicated scratch detector that performs better than a general-purpose defect detector.
    - **Specific Scratch Detector** still **detected** the **67% of the scratches** only and reported a 0.036% of false alarms.
    - The majority of scratch defects have **characteristics in common**:
        - **Very thin**.
        - **Very light contrast with the surrounding background**, which can be textured and noisy.
        - The **direction** of the scratch is usually **known in advance**.

[3] Highlighted by the student

2.8. Learning from small datasets containing nominal attributes

Month and year of publication: February 2018

Author/s: Der-Chiang Li, Hung-Yu Chen, Qi-Shi Shi

Information of interest for my research:

- **Machine learning algorithms** are extensively applied **to extract knowledge**.
- Most **machine learning algorithms** are **built on the assumption that each value has enough representatives in the training set**. If population information contained in training sets is insufficient, the knowledge extracted by algorithms is thus limited or less precise.
- **Obtaining more samples** until algorithms can carry out sufficient learning could be an intuitive approach to the issue of small-data-learning problems, but this is sometimes **challenging** and/or **costly** in certain domains.
- Algorithms can still sometimes make precise models with a few samples, though the information they contain may be limited.
- **Gaps** between a pair of close samples should be **filled with samples** in a complete dataset. These samples to fill the gaps may not be available even they may contain **important properties**.
- The paper develops a **sample generating procedure** to tackle the learning issue caused by datasets that don't fully represent the properties of the population, by **data preprocessing**.
- **Virtual Sample Generation (VSG)** are considered on their use of data preprocessing.
- **Bootstrap Procedure (BP)** draws samples from original datasets with replacement to form bootstrapping sets, to **reduce** the influence of **overfitting** issue in training sets. This is an effective approach to handle small datasets.
- VSG is proposed to **create samples for the minority class when working with imbalanced data**.
- **Fill information gaps with virtual values**.

2.9. A Deep Learning Based Printing Defect Classification Method with Imbalanced Samples

Month and year of publication: November 2019

Author/s: Erhu Zhang, Bo Li, Peilin Li, Yajun Chen

Information of interest for my research:

- Surface defect detection and classification in printing industry, with the **increasing speed** of modern printing machines and the **demand of product quality** makes **human visual inspection impossible**. It is required an **automated defect detection and classification system**.
- **Deep learning** has been successfully applied to classification tasks in many fields due to its good performance in learning discriminative features but, **the application to printing defect classification is very rare**. There is almost no research on the classification of printing defects with imbalanced samples.
- The paper presents a **deep convolutional neural network model to extract deep features directly from printed image defects in an unbalanced number of deferent kinds of defects**.
- Experiments achieved a 96.86% classification accuracy.

2.10. Fabric Defect Detection System Using Stacked Convolutional Denoising Auto-Encoders Trained with Synthetic Defect Data

Month and year of publication: April 2020

Author/s: Young-Joo Han, Ha-Jin Yu

Information of interest for my research:

- Detecting and classifying defects using image segmentation, image detection and image classification require a large number of actual **defect data** which **is difficult to obtain in industrial areas**.
- **Defect detection systems using deep learning show better performance** than the conventional feature-based defect detection system in complex patterns. They are divided

into “supervised learning” (image segmentation, classification and detection) and “unsupervised learning” (usage of autoencoders).

- **Classification networks** are trained in a supervised manner with labeled datasets and judges an image as a defect. The **segmentation networks** are trained in an unsupervised manner and locates the defect.
- The paper proposes a **method for defect detection using stacked convolutional autoencoders**, trained by **using only non-defect data and synthetic defect data** generated by using the characteristics of defect based on knowledge of the experts. The performance is comparable to the systems trained using real defect data, with an **80% accuracy** in fabric datasets.
- The images from the camera are divide into small patches and processed one-by-one, then the defects are generated, randomly located and scaled to random size, using characteristics of the defects (as described by the experts) and added to the non-defect training data. Augmentation methods are applied to the synthetic data. The training dataset was made of 64,000 images of 256x256px resized to 128x128px for training. The validation dataset was made of 6,000 captured images of real defected fabric samples. Trained the model with RED30 and Adam optimizer, Xavier initialization, learning rate of 0.0001, a weight decay of 0.0005 and a noise level of 0.25. The number of filters was 64 and the filter size of the convolution and deconvolution layers was 3x3.
- The system has a **limitation** of the **low detection-rate** of unknown defects that is shared with defect detection system based on real defect data.

2.11. Surface Defect Detection: Dataset & Papers

Month and year of publication: September 2020

Author/s: Wei Zhang

Information of interest for my research:

- Surface defect equipment based on machine vision has widely replaced artificial visual inspection in various industrial fields.

- In a real industrial environment, there are **too few industrial defect samples that can be provided**. The most critical problem faced in surface defect detection is the **small sample problem**.
- The defect detection methods based on deep learning include three main links in industrial applications: data annotation, model training and model inference[4]. Most defect detection methods are focused on the accuracy of classification or recognition. Little attention is paid to the efficiency of model inference.
- The paper proposes some methods to **overcome** these the "**small sample problem**" and the "**real-time problem**".
- **Small Sample Problem** proposals:
    - Use **data amplification** (mirroring, rotation, translation, distortion, filtering, contrast adjustment) on the original defect samples to obtain more samples.
    - **Synthetize** by fusing and superimposing individual defects on normal (non-defective) samples from defective samples.
    - **Pre-train networks** and **transfer learning** to avoid overfitting caused by small samples.
    - Use a **reasonable network structure design** so the need for samples can be reduced.
    - Use **unsupervised** or semi-supervised **models**, so only normal (non-defective) samples are needed to train the model.
- **Real-time Problem** proposals:
    - Use **model weighting** and **model pruning** techniques.
    - Use **FPGA** (Field Programable Gate Array) instead of **GPU** (Graphic Processing Unit).

[4] Inference: a process whereby a conclusion is drawn without complete certainty, but with some degree of probability relative to the evidence on which it is based.

2.12. Image Anomaly Detection Using Normal Data Only by Latent Space Resampling

Month and year of publication: December 2020

Author/s: Lu Wang, Dongki Zhang, Jiahao Guo, Yuexing Han

Information of interest for my research:

- Autoencoder has been used in image anomaly detection without using anomalous images during training. This often leads to unwanted reconstructions of the anomalous parts. This image anomaly detection problem looks for patterns that are different from normal images.
- **Anomalous instances are generally rare**, whereas **normal instances account for a significant proportion**. Distinct and unknown types can be exhibited in anomalies such as varied sizes, shapes, locations or textures. It is **almost impossible to capture large number of abnormal data containing all anomaly types**.
- **AutoEncoders** can **map the input data nonlinearly into a low-dimensional latent space and reconstruct back into the data space**.
- The paper proposes a **method based on the AutoEncoder**, with **strong reconstruction performance** that **reduces false detections**, where **the latent space of the autoencoder is estimated using a discrete probability model**.
- **Feature Extraction Based Model:** Maps images to the appropriate feature space and detect anomalies based on distance.
- **Probability Based Model:** Assumes that anomalies occur i low probability regions of the normal data.
- **Reconstruction Based Method:** Normal images can be reconstructed from latent space better than anomalous images.
- The **time required** for resampling and inference is related to the dimension of the latent space, so this may **constrain the real-time performance of the method**.

2.13. Partial conclusions of research

After doing in-deep research and reading of the previous papers, next are my general conclusions of the state-of-the-art in regards of using machine learning to detect defects in printed paper at high-speeds.

- Print quality and reliability are more and more demanding over time.
- Defects in printed matter may cause customer complaints.
- Defects in printed matter may require complete reprint.
- Print shops want to avoid printing material waste and look for increased profit margin.
- Human inspection requires dedicated operators per printer.
- Human inspection accuracy fluctuates, defects are overlooked and speed is limited.
- Some applications may require 100% inspection rate, which is not possible at high-speeds.
- Automation is a must for defect detection and classification.
- Deep learning has been successfully applied to classification tasks in many fields due to its good performance in learning discriminative features but the application to printing defect classification is very rare.
- Before developing a method or system to detect defects in printing systems at high-speeds, there are some aspects that have to be considered:
    - Printing speed.
    - Kind of printed matter.
    - Kind of defect.
    - Area to be inspected.
    - Requirement of inspection accuracy.
    - Characteristics of the Processing Unit.
    - Network speed.
- Pre-processing may be required to remove noise, remove scanning or camera artifacts, blurring, etc.
- Black and white captures would not allow assess color defects.
- The kind of defect has to be considered for its own characteristics:
    - SCRATCHES are difficult to detect with general purpose methods. Specific Scratch Detector may be required.

- Small and imbalanced datasets is a problem. Anomalous instances are generally rare, whereas normal instances account for a significant proportion. It is almost impossible to capture large number of abnormal data containing all anomaly types. Aids to overcome:
    - Augmentation.
    - Oversampling.
    - Undersampling.
    - Synthetic Sampling with Data Generation.
    - Pre-train networks and transfer learning to avoid overfitting.
- Real-time (due to high printing speed) is a problem. Aids to overcome:
    - Use model weighting and model pruning techniques.
    - Using GPU (or FPGA) instead of CPU can help with real-time (or very fast) requirements.

I've not found any paper disclosing a method or system that uses machine learning to detect and classify defects in printed paper, using digital (variable content) high-speed printers.

3. Datasets

One of the big problems of machine learning for defect detection is the lack of good datasets to train the model. I've done extensive research and I've been unable to find datasets about defects on printed paper.

I then decided to create my own dataset with printed defects but, as found in the literature too, it's not easy. The number of images with defects is very limited and the cost of creating them is quite high.

In the next lines I'm summarizing the process I've used to create the scratched/not-scratches dataset. I'm then developing it in detail in the next sub-sections.

Based on the previous evidence, I decided to start with a dataset of a single defect, relatively easy to reproduce. From the huge list of possible defects in printed paper (picking, coalescence, wicking, bleeding, condensation, offsetting, smearing, wrinkling, streaking, ink starvation, etc.) I ran a job (old edition of a newspaper) in an HP PageWide Web Press T250HD [5]press and used a tube of rolled paper to scratch the printout after the print arches, before the dryer. I then forced the Vision System (which is a module included in the press system) to capture periodic images from both sides of the paper and save these images as BMP files.

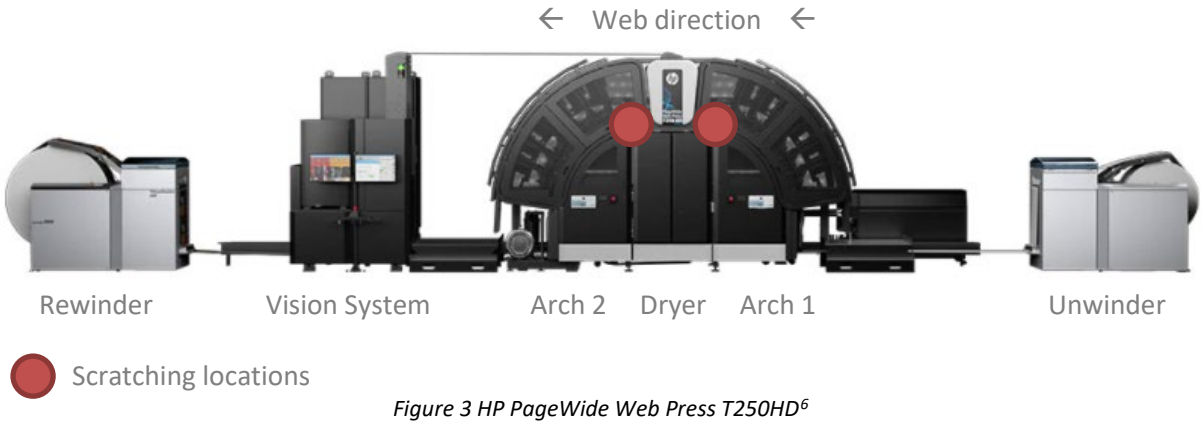


Figure 3 HP PageWide Web Press T250HD[6]

[5] HP PageWide Web Press T250HD: https://www.hp.com/us-en/commercial-industrial-printing/pagewide/t250-hd-web-presses.html
[6] Source: https://www.hp.com/es-es/commercial-printers/pagewide-industrial/t200-HD-series.html

I obtained 16 BMP RGB 2904x2166px to 2904x2369px images of random pages of the newspaper. The images contain text, lines, color pictures, faces, grayscale images, diagrams, drawings, banners, etc. It's quite representative of what a printer shop would do.

Figure 4 BMP RGB images captured by the Vision System

I then created 64x64px and 320x320px tiles from these images, had them converted to grayscale (so I had both color and grayscale datasets) and removed the tiles that ended up by not being 64x64px or 320x320px.

I had a very big amount of tiles now, to be classified as "scratched" or as "not-scratched" so they could be used to train the model. I did this classification manually, by inspecting every single image, one by one. Took me several days.

This resulted in a big amount of tiles without defect and a, compared, small amount of tiles with defect[7]. The dataset was imbalanced.

I will have to balance the dataset and I've considered two options:

- Augmentation with Data Generators
- Undersampling

In the next subsections, I've developed all the steps in detail, I have included the code and thumbnails as an example of the generated subset.

I've used Google Colab[8] to create the datasets, and Google Drive[9] to store the resulting files.

Before start creating the datasets, I've had to prepare the folders that will contain the image files:

```
# Prepares the environment to create the dataset

!rm -rf datasets
!mkdir datasets
!mkdir datasets/scratches
!mkdir datasets/scratches/defects
!mkdir datasets/scratches/defects/tiles64
!mkdir datasets/scratches/defects/tiles64/color
!mkdir datasets/scratches/defects/tiles64/grayscale
!mkdir datasets/scratches/defects/tiles320
!mkdir datasets/scratches/defects/tiles320/color
!mkdir datasets/scratches/defects/tiles320/grayscale

!cp drive/MyDrive/Transfer/UOC/TFM/datasets/scratches/defects/*.bmp datasets/scratches/d
efects/.

# Generates dataset from original captured images

import glob

import cv2
import math

from PIL import Image

datasetspath = '/content/datasets/'
datasetfolder = 'scratches/'
```

[7] Numbers are detailed in a subsection 3.5. below.
[8] Google Colab: https://colab.research.google.com/
[9] Google Drive: https://drive.google.com/

3.1. Color 64x64px tiles

In this sub-section I'm creating the 64x64px tiles from the original BMP files captured by the Vision System. All BMP files have scratches on them, even only is some of the regions. This means that some tiles will have scratches and some other will not.

Some tiles will be smaller than 64x64px because it may happen that the size of the BMP images is not multiple of 64 in one of the two dimensions, or even in both (far right and bottom corners).

```
# Generates 64x64px tiles from all available BMP images containing defects
# 8s - 26266 tiles

tile_size = (64, 64)
offset = (64, 64)
counter = 0

for image in glob.iglob(datasetspath + datasetfolder + 'defects/*.bmp'):
  img = cv2.imread(image)
  img_shape = img.shape
  for i in range(int(math.ceil(img_shape[0] / (offset[1] * 1.0)))):
    for j in range(int(math.ceil(img_shape[1] / (offset[0] * 1.0)))):
      cropped_img = img[offset[1] * i:min(offset[1] * i + tile_size[1],
                                          img_shape[0]), offset[0] *
                        j:min(offset[0] * j + tile_size[0], img_shape[1])]
      cv2.imwrite(datasetspath + datasetfolder + 'defects/tiles64/color/' +
                  'def' + image.lstrip(datasetspath + datasetfolder +
                                       'defects/') + '_' + str(i) + '_' +
                  str(j) + '.png', cropped_img)
      counter = counter + 1

print('Tiles = ' + str(counter))
```

Next, there is a random sample of the generated images:

```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles64/color/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
```

```
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs)

plt.show()
```

Output:

Note that there are images that are not 64x64px. This is addressed in a sub-section below.

3.2. Color 320x320px tiles

In this sub-section I'm creating the 320x320px tiles from the original BMP files captured by the Vision System. All BMP files have scratches on them, even only is some of the regions. This means that some tiles will have scratches and some other will not.

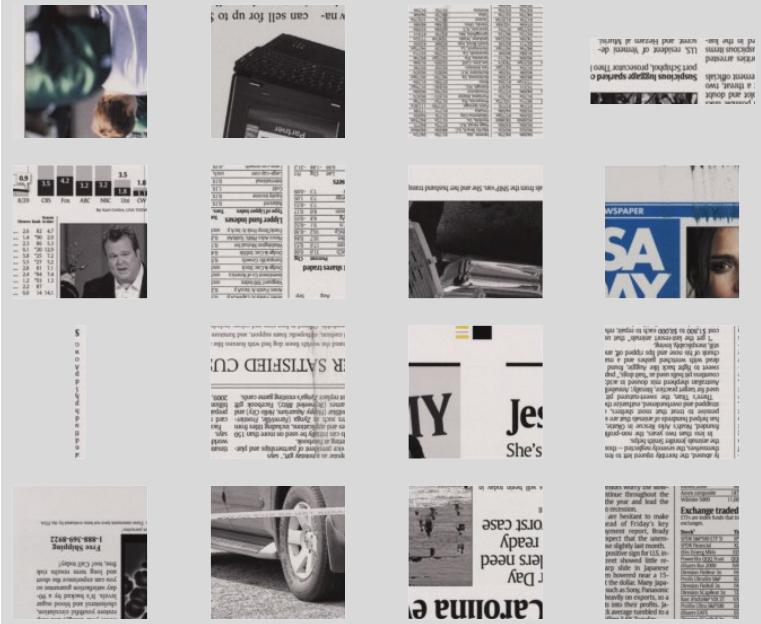
Some tiles will be smaller than 320x320px because it may happen that the size of the BMP images is not multiple of 320 in one of the two dimensions, or even in both (far right and bottom corners).

```
# Generates 320x320px tiles from all available BMP images containing defects
# 6s - 1200 tiles

tile_size = (320, 320)
offset = (320, 320)

counter = 0

for image in glob.iglob(datasetspath + datasetfolder + 'defects/*.bmp'):
  img = cv2.imread(image)
  img_shape = img.shape
  for i in range(int(math.ceil(img_shape[0] / (offset[1] * 1.0)))):
    for j in range(int(math.ceil(img_shape[1] / (offset[0] * 1.0)))):
      cropped_img = img[offset[1] * i:min(offset[1] * i + tile_size[1],
                                          img_shape[0]), offset[0] *
                        j:min(offset[0] * j + tile_size[0], img_shape[1])]
      cv2.imwrite(datasetspath + datasetfolder + 'defects/tiles320/color/' +
                  'def' + image.lstrip(datasetspath + datasetfolder +
                                       'defects/') + '_' + str(i) + '_' +
                  str(j) + '.png', cropped_img)
      counter = counter + 1

print('Tiles = ' + str(counter))
```

Next, there is a random sample of the generated images:

```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles320/color/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs)

plt.show()
```

Output:



Note that there are images that are not 64x64px. This is addressed in a sub-section below.

3.3. Converting tiles to grayscale

In this sub-section, I'm converting the color tiles to grayscale, by only keeping the lightness (L).

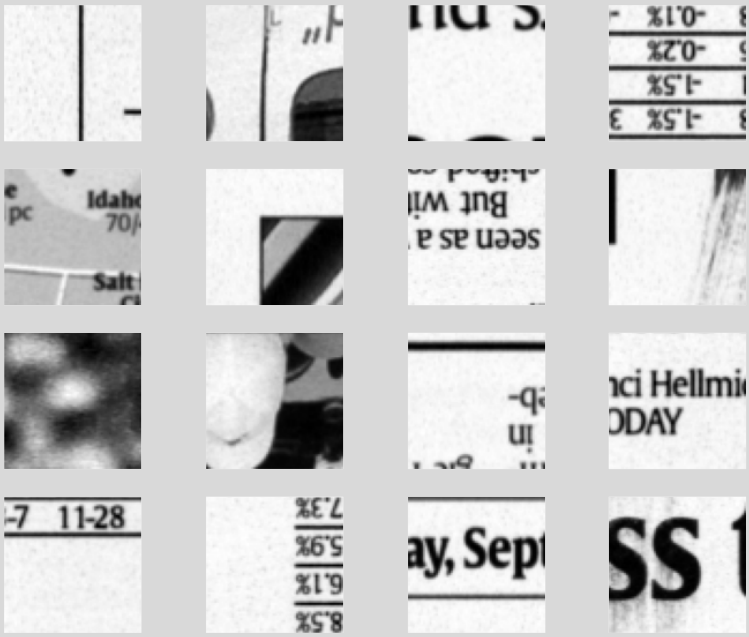
```
# Converts 64x64px Color tiles to Grayscale
# 24s - 26266 images

counter = 0

for imagecolor in glob.iglob(datasetspath + datasetfolder +
                             'defects/tiles64/color/*.png'):
    img = Image.open(imagecolor).convert('L')
    img.save(datasetspath + datasetfolder + 'defects/tiles64/grayscale/' +
             'def' + imagecolor.lstrip(datasetspath + datasetfolder +
                                       'defects/tiles64/color/'))
    counter = counter + 1

print('Images = ' + str(counter))
```

Next, there is a random sample of the generated images:

```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt
```

```
folder = "/content/datasets/scratches/defects/tiles64/grayscale/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs, cmap = 'gray')

plt.show()
```

Output:

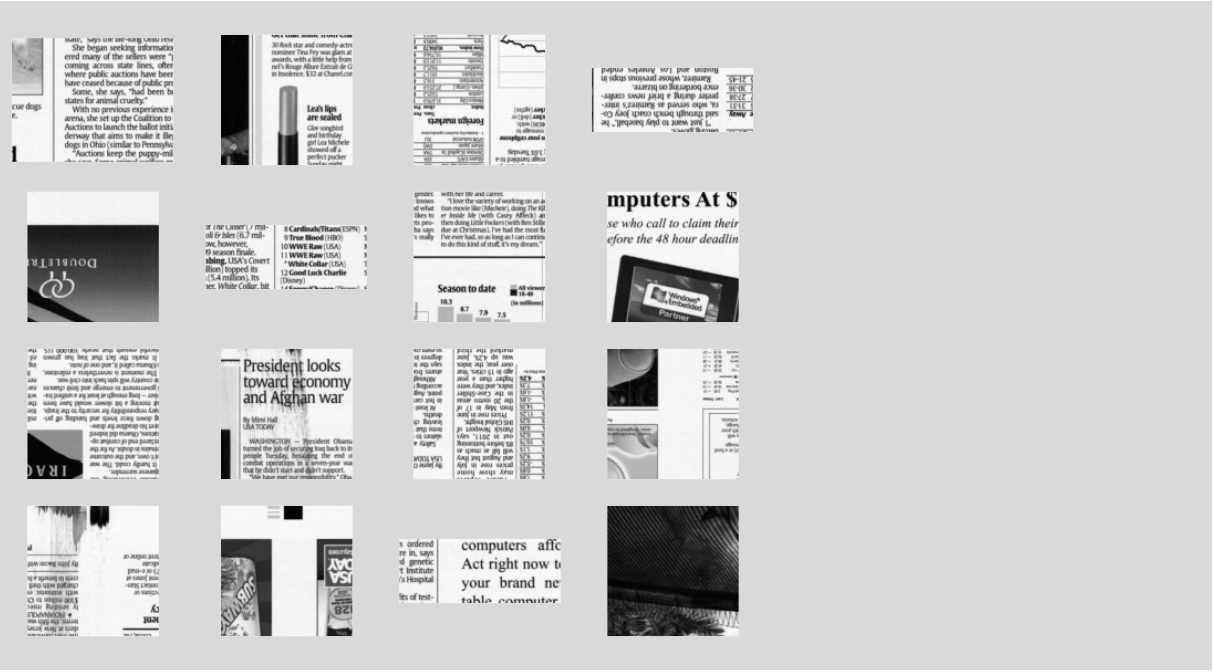


```
# Converts 320x320px Color tiles to Grayscale
# 21s - 1200 images

counter = 0

for imagecolor in glob.iglob(datasetspath + datasetfolder +
                             'defects/tiles320/color/*.png'):
    img = Image.open(imagecolor).convert('L')
    img.save(datasetspath + datasetfolder + 'defects/tiles320/grayscale/' +
             'def' + imagecolor.lstrip(datasetspath + datasetfolder +
                                       'defects/tiles320/color/'))
    counter = counter + 1

print('Images = ' + str(counter))
```

Next, there is a random sample of the generated images:

```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles320/grayscale/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs, cmap = 'gray')

plt.show()
```

Output:



3.4. Human selection of “scratched” and “not scratched” tiles

After 64x64px and 320x320px tiles have been created in both color and grayscale versions, I had to manually select what tiles had scratches and what tiles did not. This has been very tedious task as the partial datasets I’ve obtained are quite big. See subsection 3.5. The numbers for more details on dataset sizes.

I did the selection of the “scratched” and “not-scratched” tiles on my laptop, so I had to move the partial datasets from Google Colab to Google Drive:

```
# Zips the datasets folder
# 140s

!zip -r datasets.zip datasets


# Copies the datasets.zip file to Google Drive
# 8s

!cp datasets.zip drive/MyDrive/Transfer/UOC/TFM/.
```

First I did was to remove the tiles smaller than 64x64px and 320x320px from the partial datasets, as explained before, because the original captured BMP images may not be multiple of 64x64px or 320x320px (at the far right edge and far bottom edge of the image).

Then I did visually inspect every single of the color tiles, and moved the scratched ones to a folder, and the non-scratched ones to another folder.

As both the color and grayscale tiles have the same names but they are in different folders, I only had to visually inspect and select the color tiles and make sure, using a script, that the grayscale files were sorted in the same way. This simplified the task a lot, as only half of the full partial dataset had to be inspected.

Let's prepare the environment for the next. I'm first downloading the ZIP file that contains the unbalanced datasets (after being human selected):

```
# Downloads the sorted dataset to the local environment
# 60s

!mkdir datasets
!mkdir datasets/scratches

!cp drive/MyDrive/Transfer/UOC/TFM/datasets/scratches/defects.zip datasets/scratches/.

!unzip datasets/scratches/defects.zip -d datasets/scratches/
```

At this point, to see samples of the dataset with and without scratches, see Appendix 1.

3.5. The numbers

Let's look at the numbers of the partial datasets so I can analyze them and propose next steps.

At this time, the scratches dataset is as follows:

Scratches							
64x64px				320x320px			
Grayscale		Color		Grayscale		Color	
Scratched	Not-Scratched	Scratched	Not-Scratched	Scratched	Not-Scratched	Scratched	Not-Scratched
1,222	23,754	1,222	23,754	114	822	114	822
24,976		24,976		936		936	
49,952				1,872			
81,824							

Table 2 Sizes of Scratches partial datasets

For every 64x64px and 320x320px and Grayscale/Color combination, we can see that the datasets are imbalanced:

- In the case of **64x64px datasets**, the number of tiles with scratches is 1,222 and the number of tiles without scratches is 23,754. It's an **imbalanced dataset**.
- In the case of **320x320px datasets**, the number of tiles with scratches is 114 and the number of tiles without scratches is 822. It's an **imbalanced and probably small dataset**.

I **propose** to **balance the datasets by doing Random Undersampling**. This means, that I will keep all the tiles from the Scratched dataset, and randomly select the same number of tiles from the Non-Scratched dataset.

I also **propose** to **complement the datasets by doing Augmentation using Data Generators**. This is explained in deep detail in sub-section 3.8. Augmentation.

3.6. Balancing the dataset

In this sub-section I'm balancing the datasets by doing Random Undersampling. This means, that I will keep all the tiles from the Scratched dataset, and randomly select the same number of tiles from the Non-Scratched dataset.

I've divided the balancing of the datasets in two steps. The first one is to keep all the scratched files in the dataset, in a folder. The second step is to randomly select the same number of tiles and keep them in a separate folder.

I do it for 64x64px grayscale, 64x64px color, 320x320px grayscale and 320x320px color datasets:

The code to perform this balancing can be found in Appendix 2.

Now, the datasets have been balanced. Next are the new numbers of the balanced datasets:

Scratches							
64x64px				320x320px			
Grayscale		Color		Grayscale		Color	
Scratched	Not-Scratched	Scratched	Not-Scratched	Scratched	Not-Scratched	Scratched	Not-Scratched
1,222	1,222	1,222	1,222	114	114	114	114
2,444		2,444		228		228	
4,888				456			
5,344							

Table 3 Sizes of Scratches balanced datasets

3.7. Creating the final datasets

In this sub-section, I'm preparing the datasets so I can then separate between Training and Validation datasets, Augmentation, etc. I'll create a CSV file with the name of the files together with their classification (1 for "Scratched" and "0" for "Not-Scratched"). The code can be found in Appendix 3.

I now have the datasets ready for the next phases. The structure of the dataset tree is as follows:

```
datasets                                                    All datasets
└──scratches                                                Scratches dataset
    └──defects                                              Scratched
        ├──*.bmp                                            Original BMPs from the Vision System, with scratches
        ├──tiles360                                         320x320px tiles
        │   ├──color                                        320x320px color tiles
        │   │   ├──balanced                                 320x320px color balanced tiles
        │   │   │   ├──scratched                            320x320px color balanced scratched tiles
        │   │   │   │   └──*.png                            320x320px color balanced scratched PNGs
        │   │   │   └──notscratched                         320x320px color balanced notscratched tiles
        │   │   │       └──*.png                            320x320px color balanced scratched PNGs
        │   │   ├──balancedall                             320x320px color balanced all tiles
        │   │   │   └──*.png                                320x320px color balanced all PNGs
        │   │   ├──imbalancedall                           320x320px color imbalanced all tiles
        │   │   │   └──*.png                                320x320px color imbalanced all PNGs
        │   │   ├──scratched                                320x320px color imbalanced scratched tiles
        │   │   │   └──*.png                                320x320px color imbalanced scratched PNGs
        │   │   ├──notscratched                             320x320px color imbalanced notscratched tiles
        │   │   │   └──*.png                                320x320px color imbalanced notscratched PNGs
        │   │   ├──imbalanced_scratched_notscratched.csv    320x320px color imbalanced all CSV classification
        │   │   └──scratched_notscratched.csv               320x320px color balanced all CSV classification
        │   └──grayscale                                    320x320px grayscale tiles
        │       ├──balanced                                 320x320px grayscale balanced tiles
        │       │   ├──scratched                            320x320px grayscale balanced scratched tiles
        │       │   │   └──*.png                            320x320px grayscale balanced scratched PNGs
        │       │   └──notscratched                         320x320px grayscale balanced notscratched tiles
        │       │       └──*.png                            320x320px grayscale balanced scratched PNGs
        │       ├──balancedall                             320x320px grayscale balanced all tiles
        │       │   └──*.png                                320x320px grayscale balanced all PNGs
        │       ├──scratched                                320x320px grayscale imbalanced scratched tiles
        │       │   └──*.png                                320x320px grayscale imbalanced scratched PNGs
        │       ├──notscratched                             320x320px grayscale imbalanced notscratched tiles
        │       │   └──*.png                                320x320px grayscale imbalanced notscratched PNGs
        │       ├──imbalanced_scratched_notscratched.csv    320x320px grayscale imbalanced all CSV classification
        │       └──scratched_notscratched.csv               320x320px grayscale balanced all CSV classification
        └──tiles64                                          64x64px tiles
            ├──color                                        64x64px color tiles
            │   ├──balanced                                 64x64px color balanced tiles
            │   │   ├──scratched                            64x64px color balanced scratched tiles
            │   │   │   └──*.png                            64x64px color balanced scratched PNGs
            │   │   └──notscratched                         64x64px color balanced notscratched tiles
            │   │       └──*.png                            64x64px color balanced scratched PNGs
            │   ├──balancedall                             64x64px color balanced all tiles
            │   │   └──*.png                                64x64px color balanced all PNGs
            │   ├──imbalancedall                           64x64px color imbalanced all tiles
            │   │   └──*.png                                64x64px color imbalanced all PNGs
            │   ├──scratched                                64x64px color imbalanced scratched tiles
            │   │   └──*.png                                64x64px color imbalanced scratched PNGs
            │   ├──notscratched                             64x64px color imbalanced notscratched tiles
            │   │   └──*.png                                64x64px color imbalanced notscratched PNGs
            │   ├──imbalanced_scratched_notscratched.csv    64x64px color imbalanced all CSV classification
            │   └──scratched_notscratched.csv               64x64px color balanced all CSV classification
            └──grayscale                                    64x64px grayscale tiles
                ├──balanced                                 64x64px grayscale balanced tiles
                │   ├──scratched                            64x64px grayscale balanced scratched tiles
                │   │   └──*.png                            64x64px grayscale balanced scratched PNGs
                │   └──notscratched                         64x64px grayscale balanced notscratched tiles
                │       └──*.png                            64x64px grayscale balanced scratched PNGs
                ├──balancedall                             64x64px grayscale balanced all tiles
                │   └──*.png                                64x64px grayscale balanced all PNGs
                ├──imbalancedall                           64x64px grayscale imbalanced all tiles
                │   └──*.png                                64x64px grayscale imbalanced all PNGs
                ├──scratched                                64x64px grayscale imbalanced scratched tiles
                │   └──*.png                                64x64px grayscale imbalanced scratched PNGs
                ├──notscratched                             64x64px grayscale imbalanced notscratched tiles
                │   └──*.png                                64x64px grayscale imbalanced notscratched PNGs
                ├──imbalanced_scratched_notscratched.csv    64x64px grayscale imbalanced all CSV classification
                └──scratched_notscratched.csv               64x64px grayscale balanced all CSV classification
```

Figure 5 Structure of the final dataset

3.8. Augmentation

One of the outcomes from the research of the state-of-the-art in regards of datasets to be used in defect detection on printer paper is that the available (and maybe ad-hoc generated datasets) are small and imbalanced.

During the generation of the datasets in the previous sub-sections of this work, I have been able to generate a 64x64px dataset that, even may be enough in terms of samples, it's imbalanced. After deciding to apply a Random Undersampling technique to balance the dataset, it may now happen that, because I have removed samples of the "non-scratched" category, we are missing interesting properties of the images that may cause the model to classify the images with less accuracy.

Augmentation is a technique that will use the original images in the dataset to apply transformations on them so the dataset can be increased with transformed copies of the original images. The transformations that Augmentation techniques can do are rotations, horizontal and vertical flips, horizontal and vertical shifts, change in brightness and zooms. These transformations can be defined using parameters that include angles, displacements, filling modes, shifting ranges, brightness thresholds, etc.

Some of the transformations may not apply to a given dataset. For example, we may not want to vertically flip images of persons captured by a camera because people don't walk upside down, or we may not want to rotate content that is supposed to always be in a given angle.

One of the advantages of the Augmentation using an Image Data Generator is that we may not have to create a new dataset with the transformed images. The method that I will use in the next sections will transform the images to augment the dataset on-the fly, while the model is being trained.

While doing the experiments, I noticed that rotation, horizontal and vertical flips were creating undesired results, so I've decided to avoid rotations and shifts, as resulting empty spaces were being filled by populating the nearest pixels of the image.

The next subset of images is as an example of populated pixels when shifting and rotating.

The final definition for Augmentation used in this work is the next:

```
# Creates and defines the iterator to augment the dataset

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        #rotation_range = 10, fill_mode = 'nearest', # Rotation
        #width_shift_range = 0.2,                    # Horizontal shift
        #height_shift_range = 0.2,                   # Vertical shift
        horizontal_flip = True,                    # Horizontal flip
        vertical_flip = True,                      # Vertical flip
        zoom_range = 0.2,                          # Zoom
        brightness_range = [0.2, 1.2])             # Brightness
```

Samples of augmented datasets can be found in Appendix 4.

5. Experiments

As described in previous sections, the goal of the experiments is to find a dataset-model combination that could work enough to detect scratches in printouts. This is to find the dataset-model combination that provides the higher accuracy and the lowest loss.

After having the imbalanced and balanced scratches datasets created, in several different formats (tiles of 64x64px and 320x320px, in both color and grayscale versions), and having defined and Image Data Generator that can augment the data with transformations, I will now develop and execute the experiments to respond the questions defined in the objectives of this Master's Final Work.

I will train a model using the next different combinations, and I'll will look at execution's times accuracies and loss, to determine what dataset may work better to detect scratches from a high-speed printing system.

Experiment #	Tiles	Color/Grayscale	Balanced/Imbalanced	Augmentation
1	320x320px	Color	Balanced	No
2	320x320px	Color	Balanced	Yes
3	320x320px	Color	Imbalanced	No
4	320x320px	Color	Imbalanced	Yes
5	320x320px	Grayscale	Balanced	No
6	320x320px	Grayscale	Balanced	Yes
7	320x320px	Grayscale	Imbalanced	No
8	320x320px	Grayscale	Imbalanced	Yes
9	64x64px	Color	Balanced	No
10	64x64px	Color	Balanced	Yes
11	64x64px	Color	Imbalanced	No
12	64x64px	Color	Imbalanced	Yes
13	64x64px	Grayscale	Balanced	No
14	64x64px	Grayscale	Balanced	Yes
15	64x64px	Grayscale	Imbalanced	No
16	64x64px	Grayscale	Imbalanced	Yes

Table 4 Experiments to be executed in Phase 2

Note that the experiments have been sorted alphabetically by Tiles, Color/Grayscale, Balanced/Imbalanced and Augmentation. The order of the experiments doesn't respond to any expectation or preconception.

As I created the datasets in the previous phase, I don't have to create them again, but to download only. The next code is intended to download, unzip the dataset:

```
# Downloads the Phase1 dataset to the local environment
# 120s

!rm -rf datasets

!cp drive/MyDrive/Transfer/UOC/TFM/datasetsPhase1.zip .

!unzip datasetsPhase1.zip -d .
```

Output:

```
Streaming output truncated to the last 5000 lines.
 extracting:
./datasets/scratches/defects/tiles64/color/balancedall/def_20211001_113111_icf_compare_A
ny_White_69.PSA.VSLeft.bmp_7_42.png
 extracting:
./datasets/scratches/defects/tiles64/color/balancedall/def_20211001_113111_icf_compare_A
ny_White_69.PSA.VSLeft.bmp_6_41.png

[...]

  inflating: ./datasets/scratches/defects/tiles320/color/scratched_notscratched.csv
  inflating:
./datasets/scratches/defects/def_20211001_113111_icf_compare_Any_White_64.PSA.VSLeft.bmp
  inflating:
./datasets/scratches/defects/def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp
```

Also, and before proceeding with the experiments, it's important to highlight that the execution times reported in this document are based on executions in Google Colab using a specific hardware. Details on the hardware used can be found in Appendix 5.

The code to prepare the datasets to be used in the experiments can be consulted in the Appendix 6, for each experiment. In this section only the code related to the model definition is included.

Before proceeding, I need some imports to use TensorFlow Keras.

```
# Preparing to run models
# 1s

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense, Conv2D
from tensorflow.keras.layers import MaxPooling2D
import tensorflow.keras
```

5.1. Experiment 1: 320x320px Color Balanced without Augmentation

Next is the code to define and train the model without Augmentation. The code to prepare the datasets can be found in Appendix 6.

```
# Configures and runs the model WITHOUT Augmentation
# GPU 37s

# Model architecture
model_exp1 = Sequential()

model_exp1.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (320, 320, 3)))
model_exp1.add(MaxPooling2D(pool_size = (2, 2)))

model_exp1.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp1.add(MaxPooling2D(pool_size = (2, 2)))

model_exp1.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp1.add(MaxPooling2D(pool_size = (2, 2)))

model_exp1.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp1.add(MaxPooling2D(pool_size = (2, 2)))

model_exp1.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp1.add(MaxPooling2D(pool_size = (2, 2)))

model_exp1.add(Flatten())
model_exp1.add(Dense(64, activation = 'relu'))
model_exp1.add(Dropout(0.24))
model_exp1.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp1.summary()

# Compiles the model
model_exp1.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

# Trains the model
history_exp1 = model_exp1.fit(x_train, y_train,
                              epochs = 35,
                              validation_data = (x_valid, y_valid),
                              verbose = 1)
```

Output:

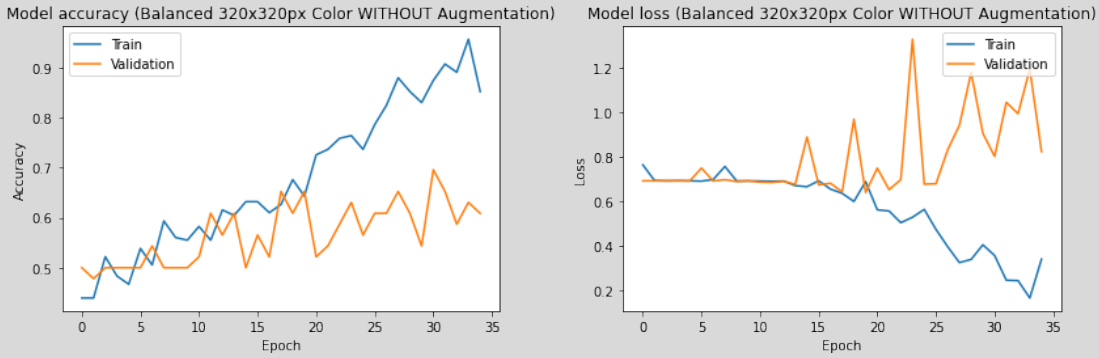
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 318, 318, 32)      896
 max_pooling2d (MaxPooling2D)  (None, 159, 159, 32)    0
 conv2d_1 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_1 (MaxPooling2D)  (None, 78, 78, 32)    0
 conv2d_2 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_2 (MaxPooling2D)  (None, 38, 38, 32)    0
 conv2d_3 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_3 (MaxPooling2D)  (None, 18, 18, 64)    0
 conv2d_4 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_4 (MaxPooling2D)  (None, 8, 8, 64)      0
 flatten (Flatten)           (None, 4096)              0
 dense (Dense)               (None, 64)                262208
 dropout (Dropout)           (None, 64)                0
 dense_1 (Dense)             (None, 2)                 130
```

```
=================================================================
Total params: 337,154
Trainable params: 337,154
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 20, the accuracy of the training keeps growing while the accuracy of the validation stays around 52-65%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 65%.

5.2. Experiment 2: 320x320px Color Balanced with Augmentation

I'm leveraging the Training and Validation sub-datasets from the training without Augmentation, and using the ImageDataGenerator defined in section 3.8. Augmentation.

> **NOTE:** This experiment is leveraging the code from the previous experiment, so the datasets are not re-created and the declaration of the TensorFlow Keras is not duplicated herein.

Next is the code to define and train the model with Augmentation:

```
# Configures and runs the model WITH Augmentation
# GPU 10m

# Model architecture
model_exp2 = Sequential()

model_exp2.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (320, 320, 3)))
model_exp2.add(MaxPooling2D(pool_size = (2, 2)))

model_exp2.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp2.add(MaxPooling2D(pool_size = (2, 2)))

model_exp2.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp2.add(MaxPooling2D(pool_size = (2, 2)))

model_exp2.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp2.add(MaxPooling2D(pool_size = (2, 2)))

model_exp2.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp2.add(MaxPooling2D(pool_size = (2, 2)))

model_exp2.add(Flatten())
model_exp2.add(Dense(64, activation = 'relu'))
model_exp2.add(Dropout(0.24))
model_exp2.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp2.summary()

# Compiles the model
model_exp2.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

# Trains the model
history_exp2 = model_exp2.fit(datagen.flow(x_train, y_train,
                                           seed = 2021,
                                           shuffle = True),
                              epochs = 100,
                              validation_data = datagen.flow(x_valid, y_valid,
                                                             seed = 2021,
                                                             shuffle = True),
                              verbose = 1)
```

Output:

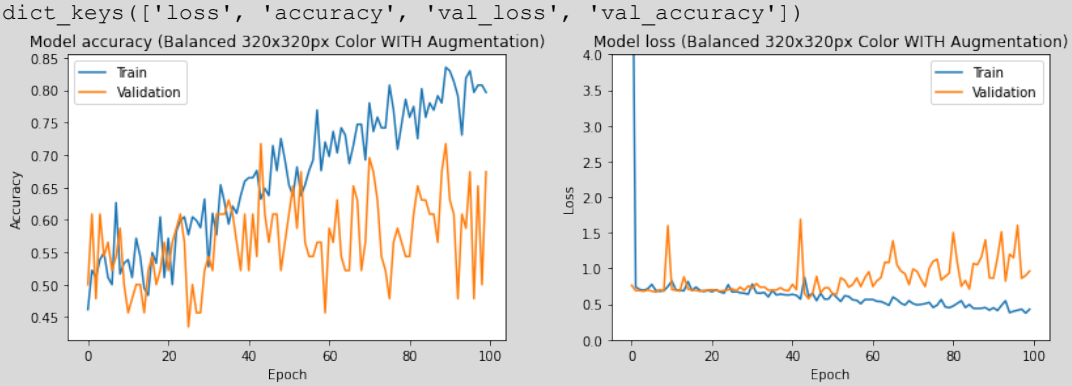
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 318, 318, 32)      896
 max_pooling2d (MaxPooling2D) (None, 159, 159, 32)      0
 conv2d_1 (Conv2D)            (None, 157, 157, 32)      9248
 max_pooling2d_1 (MaxPooling2D) (None, 78, 78, 32)      0
 conv2d_2 (Conv2D)            (None, 76, 76, 32)        9248
 max_pooling2d_2 (MaxPooling2D) (None, 38, 38, 32)      0
 conv2d_3 (Conv2D)            (None, 36, 36, 64)        18496
 max_pooling2d_3 (MaxPooling2D) (None, 18, 18, 64)      0
 conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928
 max_pooling2d_4 (MaxPooling2D) (None, 8, 8, 64)        0
 flatten (Flatten)            (None, 4096)              0
 dense (Dense)                (None, 64)                262208
 dropout (Dropout)            (None, 64)                0
 dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 337,154
Trainable params: 337,154
Non-trainable params: 0
_________________________________________________________________
```

```
[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 50, the accuracy of the training keeps growing while the accuracy of the validation stays around 50-70%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 65%.

5.3. Experiment 3: 320x320px Color Imbalanced without Augmentation

Next is the code to define and train the model without Augmentation. The code to prepare the datasets can be found in Appendix 6.

```
# Configures and runs the model WITHOUT Augmentation
# GPU 33s

# Model architecture
model_exp3 = Sequential()

model_exp3.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (320, 320, 3)))
model_exp3.add(MaxPooling2D(pool_size = (2, 2)))

model_exp3.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp3.add(MaxPooling2D(pool_size = (2, 2)))

model_exp3.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp3.add(MaxPooling2D(pool_size = (2, 2)))

model_exp3.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp3.add(MaxPooling2D(pool_size = (2, 2)))
```

```
model_exp3.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp3.add(MaxPooling2D(pool_size = (2, 2)))

model_exp3.add(Flatten())
model_exp3.add(Dense(64, activation = 'relu'))
model_exp3.add(Dropout(0.24))
model_exp3.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp3.summary()

# Compiles the model
model_exp3.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

# Trains the model
history_exp3 = model_exp3.fit(x_train, y_train,
                              epochs = 35,
                              validation_data = (x_valid, y_valid),
                              verbose = 1)
```

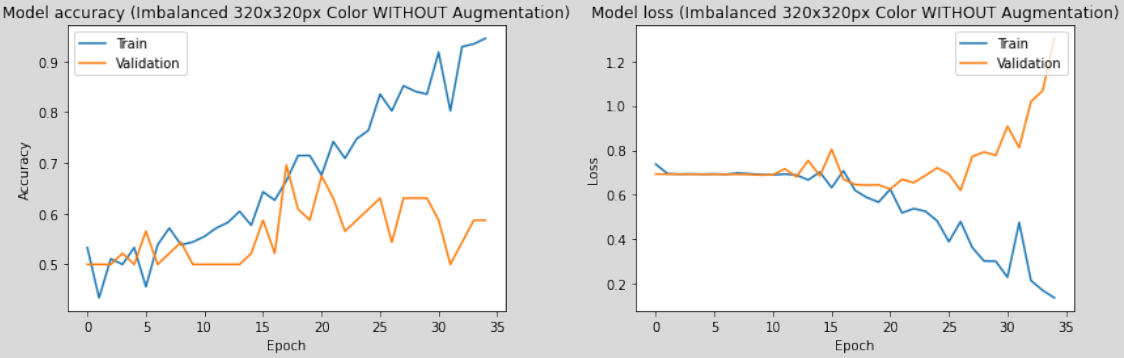
Output:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)           (None, 318, 318, 32)      896
 max_pooling2d_10 (MaxPooling2D) (None, 159, 159, 32)   0
 conv2d_11 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_11 (MaxPooling2D) (None, 78, 78, 32)     0
 conv2d_12 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_12 (MaxPooling2D) (None, 38, 38, 32)     0
 conv2d_13 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_13 (MaxPooling2D) (None, 18, 18, 64)     0
 conv2d_14 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_14 (MaxPooling2D) (None, 8, 8, 64)       0
 flatten_2 (Flatten)          (None, 4096)              0
 dense_4 (Dense)              (None, 64)                262208
 dropout_2 (Dropout)          (None, 64)                0
 dense_5 (Dense)              (None, 2)                 130
=================================================================
Total params: 337,154
Trainable params: 337,154
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 18, the accuracy of the training keeps growing while the accuracy of the validation stays around 70%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 70%.

5.4. Experiment 4: 320x320px Color Imbalanced with Augmentation

I'm leveraging the Training and Validation sub-datasets from the training without Augmentation, and using the ImageDataGenerator defined in section 3.8. Augmentation.

> **NOTE:** This experiment is leveraging the code from the previous experiment, so the datasets are not re-created and the declaration of the TensorFlow Keras is not duplicated herein.

Next is the code to define and train the model with Augmentation:

```
# Configures and runs the model WITH Augmentation
# GPU 10m

# Model architecture
model_exp4 = Sequential()

model_exp4.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (320, 320, 3)))
model_exp4.add(MaxPooling2D(pool_size = (2, 2)))

model_exp4.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp4.add(MaxPooling2D(pool_size = (2, 2)))

model_exp4.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp4.add(MaxPooling2D(pool_size = (2, 2)))

model_exp4.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp4.add(MaxPooling2D(pool_size = (2, 2)))

model_exp4.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp4.add(MaxPooling2D(pool_size = (2, 2)))
```

```
model_exp4.add(Flatten())
model_exp4.add(Dense(64, activation = 'relu'))
model_exp4.add(Dropout(0.24))
model_exp4.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp4.summary()

# Compiles the model
model_exp4.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

# Trains the model
history_exp4 = model_exp4.fit(datagen.flow(x_train, y_train,
                                           seed = 2021,
                                           shuffle = True),
                              epochs = 100,
                              validation_data = datagen.flow(x_valid, y_valid,
                                                             seed = 2021,
                                                             shuffle = True),
                              verbose = 1)
```

Output:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)           (None, 318, 318, 32)      896
 max_pooling2d_15 (MaxPooling2D) (None, 159, 159, 32)   0
 conv2d_16 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_16 (MaxPooling2D) (None, 78, 78, 32)     0
 conv2d_17 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_17 (MaxPooling2D) (None, 38, 38, 32)     0
 conv2d_18 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_18 (MaxPooling2D) (None, 18, 18, 64)     0
 conv2d_19 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_19 (MaxPooling2D) (None, 8, 8, 64)       0
 flatten_3 (Flatten)          (None, 4096)              0
 dense_6 (Dense)              (None, 64)                262208
 dropout_3 (Dropout)          (None, 64)                0
 dense_7 (Dense)              (None, 2)                 130
=================================================================
Total params: 337,154
Trainable params: 337,154
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

The training of the model shows that, around the Epoch 57, the accuracy of the training keeps growing while the accuracy of the validation stays around 72%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 72%.

5.5. Experiment 5: 320x320px Grayscale Balanced without Augmentation

Next is the code to define and train the model without Augmentation. The code to prepare the datasets can be found in Appendix 6.

```
# Configures and runs the model WITHOUT Augmentation
# GPU 42s

# Model architecture
model_exp5 = Sequential()

model_exp5.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (320, 320, 1)))
model_exp5.add(MaxPooling2D(pool_size = (2, 2)))

model_exp5.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp5.add(MaxPooling2D(pool_size = (2, 2)))

model_exp5.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp5.add(MaxPooling2D(pool_size = (2, 2)))

model_exp5.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp5.add(MaxPooling2D(pool_size = (2, 2)))

model_exp5.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp5.add(MaxPooling2D(pool_size = (2, 2)))

model_exp5.add(Flatten())
model_exp5.add(Dense(64, activation = 'relu'))
model_exp5.add(Dropout(0.24))
model_exp5.add(Dense(2, activation = 'softmax'))
```

```
# Shows model summary
model_exp5.summary()

# Compiles the model
model_exp5.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

# Trains the model
history_exp5 = model_exp5.fit(x_train, y_train,
                              epochs = 35,
                              validation_data = (x_valid, y_valid),
                              verbose = 1)
```

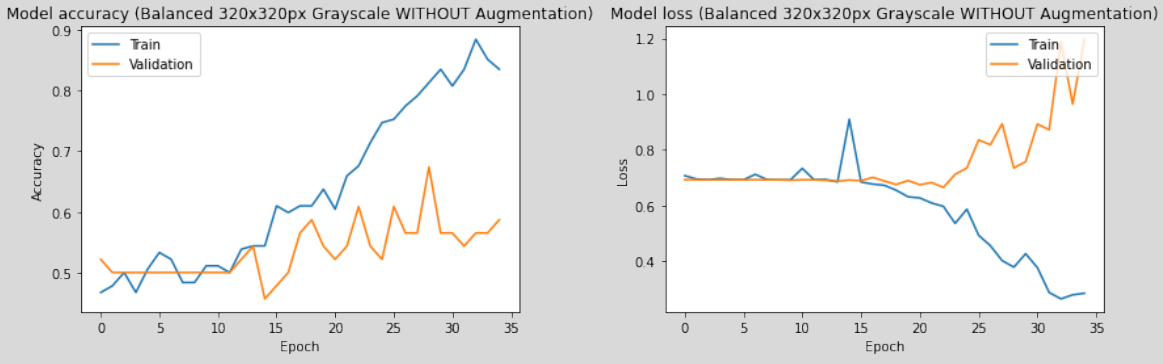
Output:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)           (None, 318, 318, 32)      320
 max_pooling2d_20 (MaxPooling2D) (None, 159, 159, 32)   0
 conv2d_21 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_21 (MaxPooling2D) (None, 78, 78, 32)     0
 conv2d_22 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_22 (MaxPooling2D) (None, 38, 38, 32)     0
 conv2d_23 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_23 (MaxPooling2D) (None, 18, 18, 64)     0
 conv2d_24 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_24 (MaxPooling2D) (None, 8, 8, 64)       0
 flatten_4 (Flatten)          (None, 4096)              0
 dense_8 (Dense)              (None, 64)                262208
 dropout_4 (Dropout)          (None, 64)                0
 dense_9 (Dense)              (None, 2)                 130
=================================================================
Total params: 336,578
Trainable params: 336,578
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 29, the accuracy of the training keeps growing while the accuracy of the validation stays around 52-67%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 67%.

5.6. Experiment 6: 320x320px Grayscale Balanced with Augmentation

I'm leveraging the Training and Validation sub-datasets from the training without Augmentation, and using the ImageDataGenerator defined in section 3.8. Augmentation.

> **NOTE:** This experiment is leveraging the code from the previous experiment, so the datasets are not re-created and the declaration of the TensorFlow Keras is not duplicated herein.

Next is the code to define and train the model with Augmentation:

```
# Configures and runs the model WITH Augmentation
# GPU 3m

# Model architecture
model_exp6 = Sequential()

model_exp6.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (320, 320, 1)))
model_exp6.add(MaxPooling2D(pool_size = (2, 2)))

model_exp6.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp6.add(MaxPooling2D(pool_size = (2, 2)))

model_exp6.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp6.add(MaxPooling2D(pool_size = (2, 2)))

model_exp6.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp6.add(MaxPooling2D(pool_size = (2, 2)))

model_exp6.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp6.add(MaxPooling2D(pool_size = (2, 2)))

model_exp6.add(Flatten())
model_exp6.add(Dense(64, activation = 'relu'))
model_exp6.add(Dropout(0.24))
model_exp6.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp6.summary()

# Compiles the model
model_exp6.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])
```

```
# Trains the model
history_exp6 = model_exp6.fit(datagen.flow(x_train, y_train,
                                           seed = 2021,
                                           shuffle = True),
                              epochs = 100,
                              validation_data = datagen.flow(x_valid, y_valid,
                                                             seed = 2021,
                                                             shuffle = True),
                              verbose = 1)
```

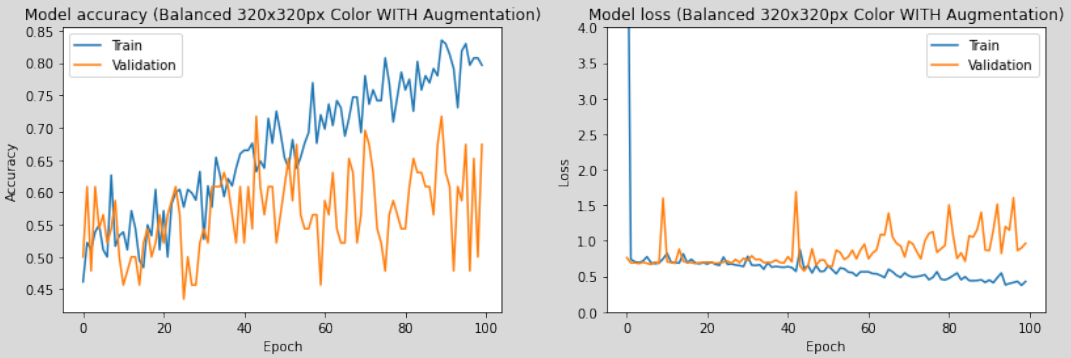
Output:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)           (None, 318, 318, 32)      320
 max_pooling2d_25 (MaxPooling2D) (None, 159, 159, 32)   0
 conv2d_26 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_26 (MaxPooling2D) (None, 78, 78, 32)     0
 conv2d_27 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_27 (MaxPooling2D) (None, 38, 38, 32)     0
 conv2d_28 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_28 (MaxPooling2D) (None, 18, 18, 64)     0
 conv2d_29 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_29 (MaxPooling2D) (None, 8, 8, 64)       0
 flatten_5 (Flatten)          (None, 4096)              0
 dense_10 (Dense)             (None, 64)                262208
 dropout_5 (Dropout)          (None, 64)                0
 dense_11 (Dense)             (None, 2)                 130
=================================================================
Total params: 336,578
Trainable params: 336,578
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 44, the accuracy of the training keeps growing while the accuracy of the validation stays around 90-72%, and the loss of the validation slightly increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 72%.

5.7. Experiment 7: 320x320px Grayscale Imbalanced without Augmentation

Next is the code to define and train the model without Augmentation. The code to prepare the datasets can be found in Appendix 6.

```
# Configures and runs the model WITHOUT Augmentation
# GPU 42s

# Model architecture
model_exp7 = Sequential()

model_exp7.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (320, 320, 1)))
model_exp7.add(MaxPooling2D(pool_size = (2, 2)))

model_exp7.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp7.add(MaxPooling2D(pool_size = (2, 2)))

model_exp7.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp7.add(MaxPooling2D(pool_size = (2, 2)))

model_exp7.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp7.add(MaxPooling2D(pool_size = (2, 2)))

model_exp7.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp7.add(MaxPooling2D(pool_size = (2, 2)))

model_exp7.add(Flatten())
model_exp7.add(Dense(64, activation = 'relu'))
model_exp7.add(Dropout(0.24))
model_exp7.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp7.summary()

# Compiles the model
model_exp7.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

# Trains the model
history_exp7 = model_exp7.fit(x_train, y_train,
                              epochs = 35,
                              validation_data = (x_valid, y_valid),
                              verbose = 1)
```

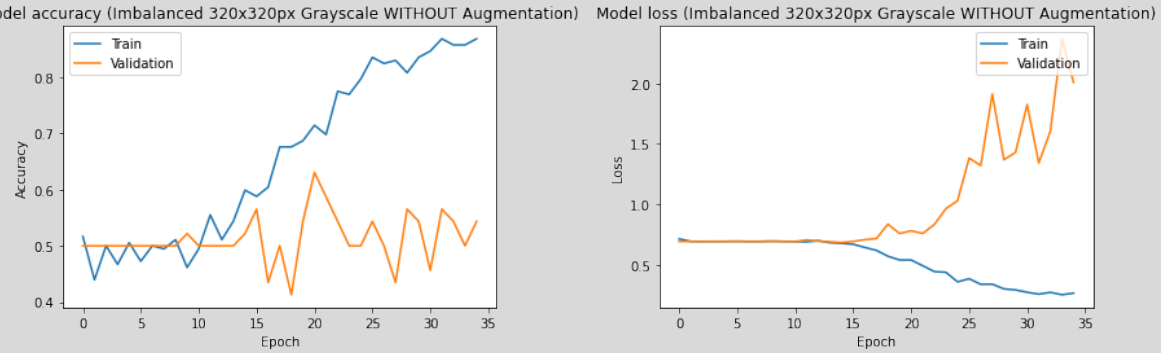
Output:

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_30 (Conv2D)           (None, 318, 318, 32)      320
 max_pooling2d_30 (MaxPooling2D) (None, 159, 159, 32)      0
 conv2d_31 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_31 (MaxPooling2D) (None, 78, 78, 32)        0
 conv2d_32 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_32 (MaxPooling2D) (None, 38, 38, 32)        0
 conv2d_33 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_33 (MaxPooling2D) (None, 18, 18, 64)        0
 conv2d_34 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_34 (MaxPooling2D) (None, 8, 8, 64)          0
 flatten_6 (Flatten)          (None, 4096)              0
 dense_12 (Dense)             (None, 64)                262208
 dropout_6 (Dropout)          (None, 64)                0
 dense_13 (Dense)             (None, 2)                 130
=================================================================
Total params: 336,578
Trainable params: 336,578Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 16, the accuracy of the training keeps growing while the accuracy of the validation stays around 41-63%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 56%.

5.8. Experiment 8: 320x320px Grayscale Imbalanced with Augmentation

I'm leveraging the Training and Validation sub-datasets from the training without Augmentation, and using the ImageDataGenerator defined in section 3.8. Augmentation.

NOTE: This experiment is leveraging the code from the previous experiment, so the datasets are not re-created and the declaration of the TensorFlow Keras is not duplicated herein.

Next is the code to define and train the model with Augmentation:

```
# Configures and runs the model WITH Augmentation
# GPU 3.5m

# Model architecture
model_exp8 = Sequential()

model_exp8.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (320, 320, 1)))
model_exp8.add(MaxPooling2D(pool_size = (2, 2)))

model_exp8.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp8.add(MaxPooling2D(pool_size = (2, 2)))

model_exp8.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp8.add(MaxPooling2D(pool_size = (2, 2)))

model_exp8.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp8.add(MaxPooling2D(pool_size = (2, 2)))

model_exp8.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp8.add(MaxPooling2D(pool_size = (2, 2)))

model_exp8.add(Flatten())
model_exp8.add(Dense(64, activation = 'relu'))
model_exp8.add(Dropout(0.24))
model_exp8.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp8.summary()

# Compiles the model
model_exp8.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

# Trains the model
history_exp8 = model_exp8.fit(datagen.flow(x_train, y_train,
                                           seed = 2021,
                                           shuffle = True),
                              epochs = 100,
                              validation_data = datagen.flow(x_valid, y_valid,
                                                             seed = 2021,
                                                             shuffle = True),
                              verbose = 1)
```

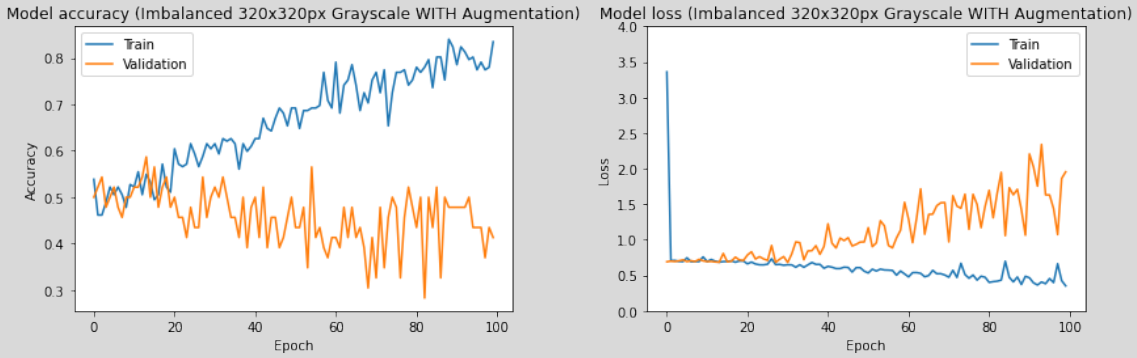
Output:

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_35 (Conv2D)           (None, 318, 318, 32)      320
 max_pooling2d_35 (MaxPooling2D) (None, 159, 159, 32)     0
 conv2d_36 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_36 (MaxPooling2D) (None, 78, 78, 32)       0
 conv2d_37 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_37 (MaxPooling2D) (None, 38, 38, 32)       0
 conv2d_38 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_38 (MaxPooling2D) (None, 18, 18, 64)       0
 conv2d_39 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_39 (MaxPooling2D) (None, 8, 8, 64)         0
 flatten_7 (Flatten)          (None, 4096)              0
 dense_14 (Dense)             (None, 64)                262208
 dropout_7 (Dropout)          (None, 64)                0
 dense_15 (Dense)             (None, 2)                 130
=================================================================
Total params: 336,578
Trainable params: 336,578
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 14, the accuracy of the training keeps growing while the accuracy of the validation lowers oscillating between 30% and 56%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 59%.

5.9. Experiment 9: 64x64px Color Balanced without Augmentation

Next is the code to define and train the model without Augmentation. The code to prepare the datasets can be found in Appendix 6.

```
# Configures and runs the model WITHOUT Augmentation
# GPU 33s

# Model architecture
model_exp9 = Sequential()

model_exp9.add(Conv2D(32, (3, 3), activation = 'relu',
                      input_shape = (64, 64, 3)))
model_exp9.add(MaxPooling2D(pool_size = (2, 2)))

model_exp9.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp9.add(MaxPooling2D(pool_size = (2, 2)))

model_exp9.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp9.add(MaxPooling2D(pool_size = (2, 2)))

model_exp9.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp9.add(MaxPooling2D(pool_size = (2, 2)))

model_exp9.add(Flatten())
model_exp9.add(Dense(64, activation = 'relu'))
model_exp9.add(Dropout(0.24))
model_exp9.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp9.summary()

# Compiles the model
model_exp9.compile(loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

# Trains the model
history_exp9 = model_exp9.fit(x_train, y_train,
                              epochs = 35,
                              validation_data = (x_valid, y_valid),
                              verbose = 1)
```

Output:

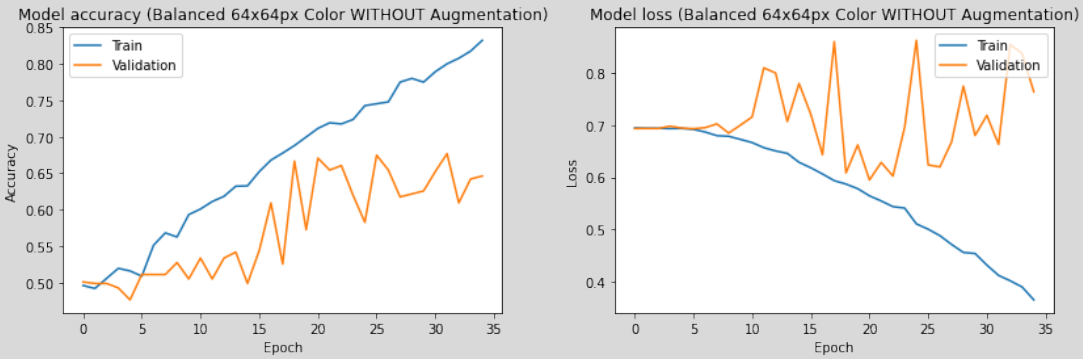
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_40 (Conv2D)           (None, 62, 62, 32)        896
 max_pooling2d_40 (MaxPooling2D) (None, 31, 31, 32)     0
 conv2d_41 (Conv2D)           (None, 29, 29, 32)        9248
 max_pooling2d_41 (MaxPooling2D) (None, 14, 14, 32)     0
 conv2d_42 (Conv2D)           (None, 12, 12, 32)        9248
 max_pooling2d_42 (MaxPooling2D) (None, 6, 6, 32)       0
 conv2d_43 (Conv2D)           (None, 4, 4, 64)          18496
 max_pooling2d_43 (MaxPooling2D) (None, 2, 2, 64)       0
 flatten_8 (Flatten)          (None, 256)               0
 dense_16 (Dense)             (None, 64)                16448
 dropout_8 (Dropout)          (None, 64)                0
 dense_17 (Dense)             (None, 2)                 130
=================================================================
Total params: 54,466
Trainable params: 54,466
Non-trainable params: 0
_________________________________________________________________
```

```
[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 5, the accuracy of the training keeps growing while the accuracy of the validation stays around 48-68%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 48%.

5.10. Experiment 10: 64x64px Color Balanced with Augmentation

I'm leveraging the Training and Validation sub-datasets from the training without Augmentation, and using the ImageDataGenerator defined in section 3.8. Augmentation.

> **NOTE:** This experiment is leveraging the code from the previous experiment, so the datasets are not re-created and the declaration of the TensorFlow Keras is not duplicated herein.

Next is the code to define and train the model with Augmentation:

```
# Configures and runs the model WITH Augmentation
# GPU 7m

# Model architecture
model_exp10 = Sequential()

model_exp10.add(Conv2D(32, (3, 3), activation = 'relu',
                       input_shape = (64, 64, 3)))
model_exp10.add(MaxPooling2D(pool_size = (2, 2)))
```

```
model_exp10.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp10.add(MaxPooling2D(pool_size = (2, 2)))

model_exp10.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp10.add(MaxPooling2D(pool_size = (2, 2)))

model_exp10.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp10.add(MaxPooling2D(pool_size = (2, 2)))

model_exp10.add(Flatten())
model_exp10.add(Dense(64, activation = 'relu'))
model_exp10.add(Dropout(0.24))
model_exp10.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp10.summary()

# Compiles the model
model_exp10.compile(loss = 'binary_crossentropy',
                    metrics = ['accuracy'])


# Trains the model
history_exp10 = model_exp10.fit(datagen.flow(x_train, y_train,
                                             seed = 2021,
                                             shuffle = True),
                                epochs = 100,
                                validation_data = datagen.flow(x_valid, y_valid,
                                                               seed = 2021,
                                                               shuffle = True),
                                verbose = 1)
```

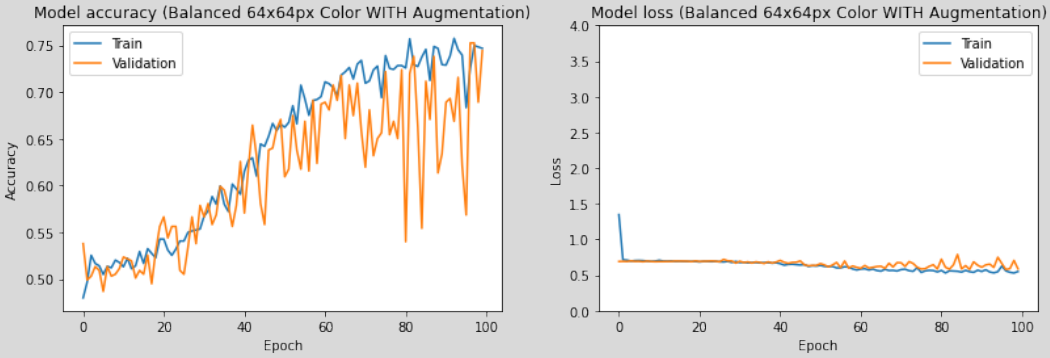
Output:

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
conv2d_44 (Conv2D)           (None, 62, 62, 32)        896
 max_pooling2d_44 (MaxPooling2D) (None, 31, 31, 32)       0
 conv2d_45 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_45 (MaxPooling2D) (None, 14, 14, 32)       0
 conv2d_46 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_46 (MaxPooling2D) (None, 6, 6, 32)         0
 conv2d_47 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_47 (MaxPooling2D) (None, 2, 2, 64)         0
 flatten_9 (Flatten)         (None, 256)               0
 dense_18 (Dense)            (None, 64)                16448
 dropout_9 (Dropout)         (None, 64)                0
 dense_19 (Dense)            (None, 2)                 130
=================================================================
Total params: 54,466
Trainable params: 54,466
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that both training and validation accuracies keep growing up to 72%, while both loses trend to be lower. This is true after 65 Epochs. It's worth to mention that, after 65 Epochs, the accuracy trends to flatten, and I don't see a real benefit of running it for more Epochs. The accuracy of this model with this dataset is of 72%.

5.11. Experiment 11: 64x64px Color Imbalanced without Augmentation

Next is the code to define and train the model without Augmentation. The code to prepare the datasets can be found in Appendix 6.

```
# Configures and runs the model WITHOUT Augmentation
# GPU 42s

# Model architecture
model_exp11 = Sequential()

model_exp11.add(Conv2D(32, (3, 3), activation = 'relu',
                       input_shape = (64, 64, 3),))
model_exp11.add(MaxPooling2D(pool_size = (2, 2)))

model_exp11.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp11.add(MaxPooling2D(pool_size = (2, 2)))

model_exp11.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp11.add(MaxPooling2D(pool_size = (2, 2)))

model_exp11.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp11.add(MaxPooling2D(pool_size = (2, 2)))

model_exp11.add(Flatten())
model_exp11.add(Dense(64, activation = 'relu'))
model_exp11.add(Dropout(0.24))
model_exp11.add(Dense(2, activation = 'softmax'))
```

```
# Shows model summary
model_exp11.summary()

# Compiles the model
model_exp11.compile(loss = 'binary_crossentropy',
                    metrics = ['accuracy'])

# Trains the model
history_exp11 = model_exp11.fit(x_train, y_train,
                                epochs = 35,
                                validation_data = (x_valid, y_valid),
                                verbose = 1)
```

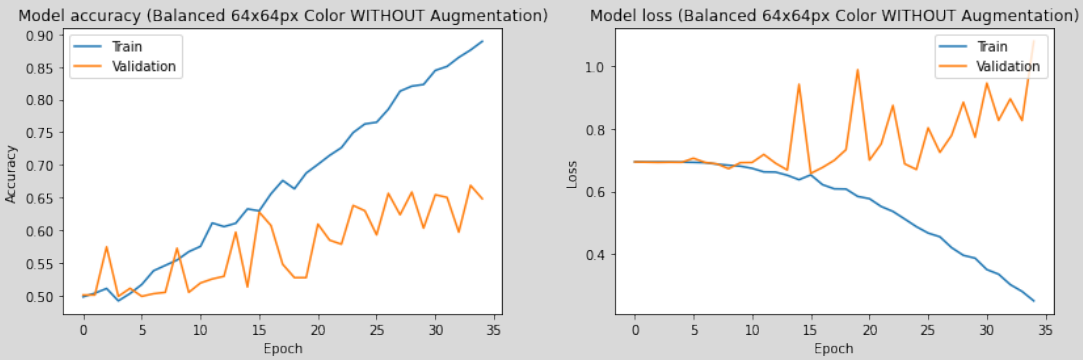
Output:

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_48 (Conv2D)          (None, 62, 62, 32)        896
 max_pooling2d_48 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_49 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_49 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_50 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_50 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_51 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_51 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_10 (Flatten)        (None, 256)               0
 dense_20 (Dense)            (None, 64)                16448
 dropout_10 (Dropout)        (None, 64)                0
 dense_21 (Dense)            (None, 2)                 130
=================================================================
Total params: 54,466
Trainable params: 54,466
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 16, the accuracy of the training keeps growing while the accuracy of the validation stays around 53-67%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 63%.

5.12. Experiment 12: 64x64px Color Imbalanced with Augmentation

I'm leveraging the Training and Validation sub-datasets from the training without Augmentation, and using the ImageDataGenerator defined in section 3.8. Augmentation.

> **NOTE:** This experiment is leveraging the code from the previous experiment, so the datasets are not re-created and the declaration of the TensorFlow Keras is not duplicated herein.

Next is the code to define and train the model with Augmentation:

```
# Configures and runs the model WITH Augmentation
# GPU 15m

# Model architecture
model_exp12 = Sequential()

model_exp12.add(Conv2D(32, (3, 3), activation = 'relu',
                       input_shape = (64, 64, 3)))
model_exp12.add(MaxPooling2D(pool_size = (2, 2)))

model_exp12.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp12.add(MaxPooling2D(pool_size = (2, 2)))

model_exp12.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp12.add(MaxPooling2D(pool_size = (2, 2)))

model_exp12.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp12.add(MaxPooling2D(pool_size = (2, 2)))

model_exp12.add(Flatten())
model_exp12.add(Dense(64, activation = 'relu'))
model_exp12.add(Dropout(0.24))
model_exp12.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp12.summary()

# Compiles the model
model_exp12.compile(loss = 'binary_crossentropy',
                    metrics = ['accuracy'])
```

```
# Trains the model
history_exp12 = model_exp12.fit(datagen.flow(x_train, y_train,
                                             seed = 2021,
                                             shuffle = True),
                                epochs = 200,
                                validation_data = datagen.flow(x_valid, y_valid,
                                                               seed = 2021,
                                                               shuffle = True),
                                verbose = 1)
```

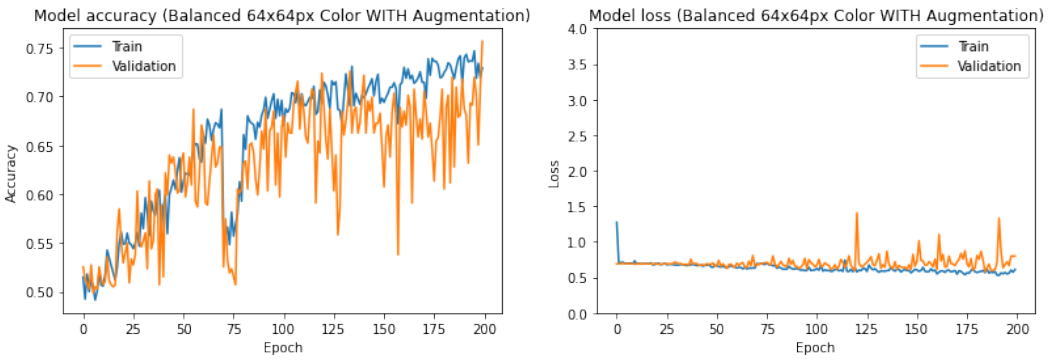
Output:

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_52 (Conv2D)           (None, 62, 62, 32)        896
 max_pooling2d_52 (MaxPooling2D) (None, 31, 31, 32)     0
 conv2d_53 (Conv2D)           (None, 29, 29, 32)        9248
 max_pooling2d_53 (MaxPooling2D) (None, 14, 14, 32)     0
 conv2d_54 (Conv2D)           (None, 12, 12, 32)        9248
 max_pooling2d_54 (MaxPooling2D) (None, 6, 6, 32)       0
 conv2d_55 (Conv2D)           (None, 4, 4, 64)          18496
 max_pooling2d_55 (MaxPooling2D) (None, 2, 2, 64)       0
 flatten_11 (Flatten)         (None, 256)               0
 dense_22 (Dense)             (None, 64)                16448
 dropout_11 (Dropout)         (None, 64)                0
 dense_23 (Dense)             (None, 2)                 130
=================================================================
Total params: 54,466
Trainable params: 54,466
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that both training and validation accuracies keep growing up to 76%, while both loses trend to be lower. This is true after 200 Epochs. It's worth to mention that, after 100 Epochs, the accuracy trends to flatten, and I don't see a real benefit of running it for more Epochs. The accuracy of this model with this dataset is of 76%.

5.13. Experiment 13: 64x64px Grayscale Balanced without Augmentation

Next is the code to define and train the model without Augmentation. The code to prepare the datasets can be found in Appendix 6.

```
# Configures and runs the model WITHOUT Augmentation
# GPU 34s

# Model architecture
model_exp13 = Sequential()

model_exp13.add(Conv2D(32, (3, 3), activation = 'relu',
                       input_shape = (64, 64, 1),))
model_exp13.add(MaxPooling2D(pool_size = (2, 2)))

model_exp13.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp13.add(MaxPooling2D(pool_size = (2, 2)))

model_exp13.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp13.add(MaxPooling2D(pool_size = (2, 2)))

model_exp13.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp13.add(MaxPooling2D(pool_size = (2, 2)))

model_exp13.add(Flatten())
model_exp13.add(Dense(64, activation = 'relu'))
model_exp13.add(Dropout(0.24))
model_exp13.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp13.summary()

# Compiles the model
model_exp13.compile(loss = 'binary_crossentropy',
                    metrics = ['accuracy'])

# Trains the model
history_exp13 = model_exp13.fit(x_train, y_train,
                                epochs = 35,
                                validation_data = (x_valid, y_valid),
                                verbose = 1)
```

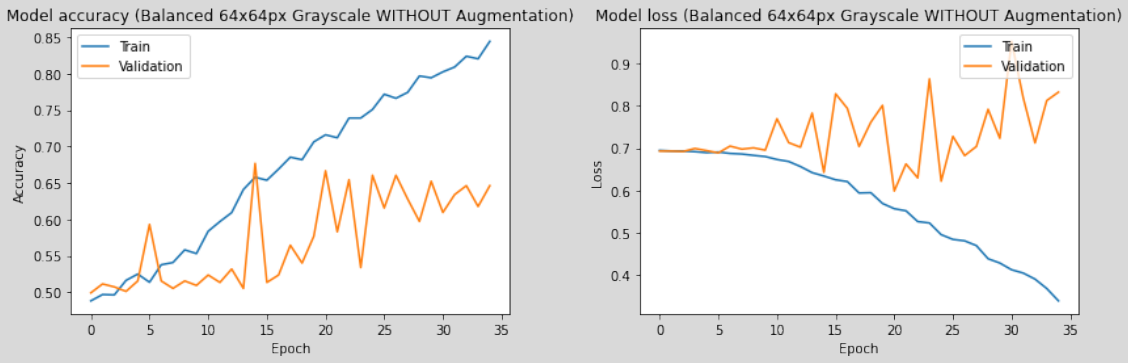
Output:

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_56 (Conv2D)          (None, 62, 62, 32)        320
 max_pooling2d_56 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_57 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_57 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_58 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_58 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_59 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_59 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_12 (Flatten)        (None, 256)               0
 dense_24 (Dense)            (None, 64)                16448
 dropout_12 (Dropout)        (None, 64)                0
 dense_25 (Dense)            (None, 2)                 130
=================================================================
Total params: 53,890
Trainable params: 53,890
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that, around the Epoch 6, the accuracy of the training keeps growing while the accuracy of the validation stays around 51-66%, and the loss of the validation increases. This is symptom of overtraining, so the maximum accuracy of this model with this dataset would be of 59%.

5.14. Experiment 14: 64x64px Grayscale Balanced with Augmentation

I'm leveraging the Training and Validation sub-datasets from the training without Augmentation, and using the ImageDataGenerator defined in section 3.8. Augmentation.

NOTE: This experiment is leveraging the code from the previous experiment, so the datasets are not re-created and the declaration of the TensorFlow Keras is not duplicated herein.

Next is the code to define and train the model with Augmentation:

```
# Configures and runs the model WITH Augmentation
# GPU 4m

# Model architecture
model_exp14 = Sequential()

model_exp14.add(Conv2D(32, (3, 3), activation = 'relu',
                       input_shape = (64, 64, 1)))
model_exp14.add(MaxPooling2D(pool_size = (2, 2)))

model_exp14.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp14.add(MaxPooling2D(pool_size = (2, 2)))

model_exp14.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp14.add(MaxPooling2D(pool_size = (2, 2)))

model_exp14.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp14.add(MaxPooling2D(pool_size = (2, 2)))

model_exp14.add(Flatten())
model_exp14.add(Dense(64, activation = 'relu'))
model_exp14.add(Dropout(0.24))
model_exp14.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp14.summary()

# Compiles the model
model_exp14.compile(loss = 'binary_crossentropy',
                    metrics = ['accuracy'])

# Trains the model
history_exp14 = model_exp14.fit(datagen.flow(x_train, y_train,
                                             seed = 2021,
                                             shuffle = True),
                                epochs = 400,
                                validation_data = datagen.flow(x_valid, y_valid,
                                                               seed = 2021,
                                                               shuffle = True),
                                verbose = 1)
```

Output:

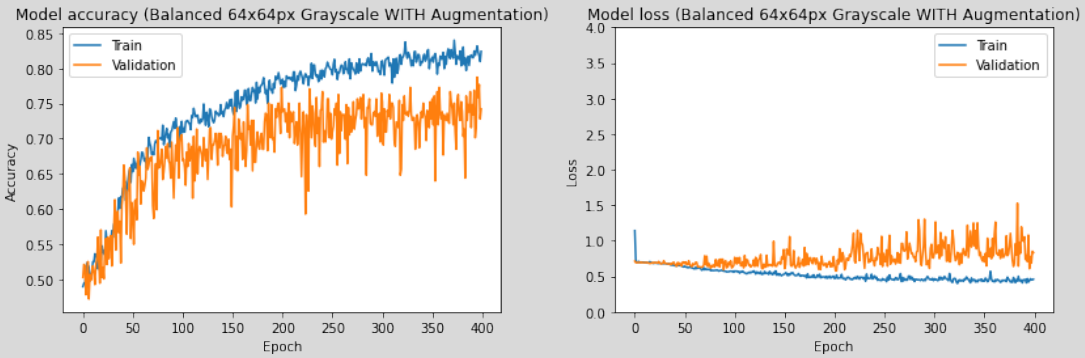
```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_60 (Conv2D)          (None, 62, 62, 32)        320
 max_pooling2d_60 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_61 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_61 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_62 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_62 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_63 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_63 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_13 (Flatten)        (None, 256)               0
 dense_26 (Dense)            (None, 64)                16448
 dropout_13 (Dropout)        (None, 64)                0
 dense_27 (Dense)            (None, 2)                 130
=================================================================
```

```
Total params: 53,890
Trainable params: 53,890
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that both training and validation accuracies keep growing up to 79%, while both loses trend to be lower. This is true after 400 Epochs, even an overtraining can be appreciated after Epoch 100.

The accuracy of this model with this dataset is of 79%, but I would only run it with 100 Epochs because there is less overtraining, it takes only 4 minutes (compared to 18 minutes for 400 Epochs) and the accuracy still stays at 75%.

5.15. Experiment 15: 64x64px Grayscale Imbalanced without Augmentation

Next is the code to define and train the model without Augmentation. The code to prepare the datasets can be found in Appendix 6.

```
# Configures and runs the model WITHOUT Augmentation
# GPU 32s

# Model architecture
model_exp15 = Sequential()

model_exp15.add(Conv2D(32, (3, 3), activation = 'relu',
                       input_shape = (64, 64, 1),))
model_exp15.add(MaxPooling2D(pool_size = (2, 2)))
```

```
model_exp15.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp15.add(MaxPooling2D(pool_size = (2, 2)))

model_exp15.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp15.add(MaxPooling2D(pool_size = (2, 2)))

model_exp15.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp15.add(MaxPooling2D(pool_size = (2, 2)))

model_exp15.add(Flatten())
model_exp15.add(Dense(64, activation = 'relu'))
model_exp15.add(Dropout(0.24))
model_exp15.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp15.summary()

# Compiles the model
model_exp15.compile(loss = 'binary_crossentropy',
                    metrics = ['accuracy'])

# Trains the model
history_exp15 = model_exp15.fit(x_train, y_train,
                                epochs = 35,
                                validation_data = (x_valid, y_valid),
                                verbose = 1)
```

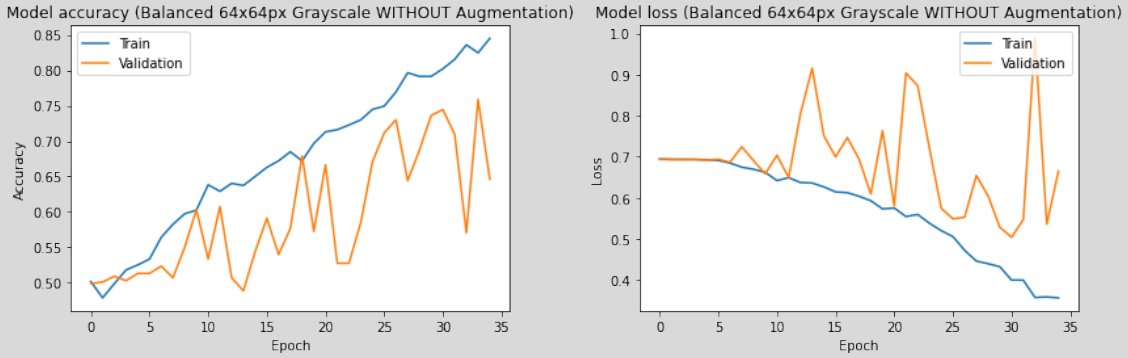
Output:

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_64 (Conv2D)          (None, 62, 62, 32)        320
 max_pooling2d_64 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_65 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_65 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_66 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_66 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_67 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_67 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_14 (Flatten)        (None, 256)               0
 dense_28 (Dense)            (None, 64)                16448
 dropout_14 (Dropout)        (None, 64)                0
 dense_29 (Dense)            (None, 2)                 130
=================================================================
Total params: 53,890
Trainable params: 53,890
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



The training of the model shows that both during train and validation, the accuracy trends to grow, even during training the trend is quite diagonal and during validation it oscillates between 40% and 75%. The trend for the loss is to lower during both train and validation, even it oscillates quite a lot between 0.5 and 1 during validation. The maximum accuracy of this model with this dataset would be of 75% after 35 Epochs but, given the oscillations in the accuracy during validation, it doesn't seem to be a good model-dataset combination.

5.16. Experiment 16: 64x64px Grayscale Imbalanced with Augmentation

I'm leveraging the Training and Validation sub-datasets from the training without Augmentation, and using the ImageDataGenerator defined in section 3.8. Augmentation.

> **NOTE:** This experiment is leveraging the code from the previous experiment, so the datasets are not re-created and the declaration of the TensorFlow Keras is not duplicated herein.

Next is the code to define and train the model with Augmentation:

```
# Configures and runs the model WITH Augmentation
# GPU 4m

# Model architecture
model_exp16 = Sequential()

model_exp16.add(Conv2D(32, (3, 3), activation = 'relu',
                       input_shape = (64, 64, 1)))
model_exp16.add(MaxPooling2D(pool_size = (2, 2)))

model_exp16.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp16.add(MaxPooling2D(pool_size = (2, 2)))
```

```
model_exp16.add(Conv2D(32, (3, 3), activation = 'relu'))
model_exp16.add(MaxPooling2D(pool_size = (2, 2)))

model_exp16.add(Conv2D(64, (3, 3), activation = 'relu'))
model_exp16.add(MaxPooling2D(pool_size = (2, 2)))

model_exp16.add(Flatten())
model_exp16.add(Dense(64, activation = 'relu'))
model_exp16.add(Dropout(0.24))
model_exp16.add(Dense(2, activation = 'softmax'))

# Shows model summary
model_exp16.summary()

# Compiles the model
model_exp16.compile(loss = 'binary_crossentropy',
                    metrics = ['accuracy'])

# Trains the model
history_exp16 = model_exp16.fit(datagen.flow(x_train, y_train,
                                             seed = 2021,
                                             shuffle = True),
                                epochs = 100,
                                validation_data = datagen.flow(x_valid, y_valid,
                                                               seed = 2021,
                                                               shuffle = True),
                                verbose = 1)
```

Output:

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_68 (Conv2D)          (None, 62, 62, 32)        320
 max_pooling2d_68 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_69 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_69 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_70 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_70 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_71 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_71 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_15 (Flatten)        (None, 256)               0
 dense_30 (Dense)            (None, 64)                16448
 dropout_15 (Dropout)        (None, 64)                0
 dense_31 (Dense)            (None, 2)                 130
=================================================================
Total params: 53,890
Trainable params: 53,890
Non-trainable params: 0
_________________________________________________________________

[...] (Full output in Appendix 7)
```

Next are the charts that graphically represent the results of the previous model training. The code to generate these charts can be found in Appendix 6.

Output:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

The training of the model shows accuracy growth during training to 74% and during validation to 70%, while both loses trend to be lower. This is true after 100 Epochs, even both curves trend to relax after Epoch 60. The accuracy of this model with this dataset is of 70%.

5.17. Comparison of experiment results

At the end of the previous sub-sections, one for each experiment, I've already explained the results of the training and validation of the different models with the different datasets combinations. Now it's time to compare the different experiment results.

The Figure 3 Comparison summary of experiment charts is a graphic representation that summarizes the results. The title of each individual chart shows the name of the dataset used in the experiment. Blue lines are the results of the training during the training of the method, while the orange lines are the results of the validation.

(Continues in the next page)

Comparison summary of experiment charts

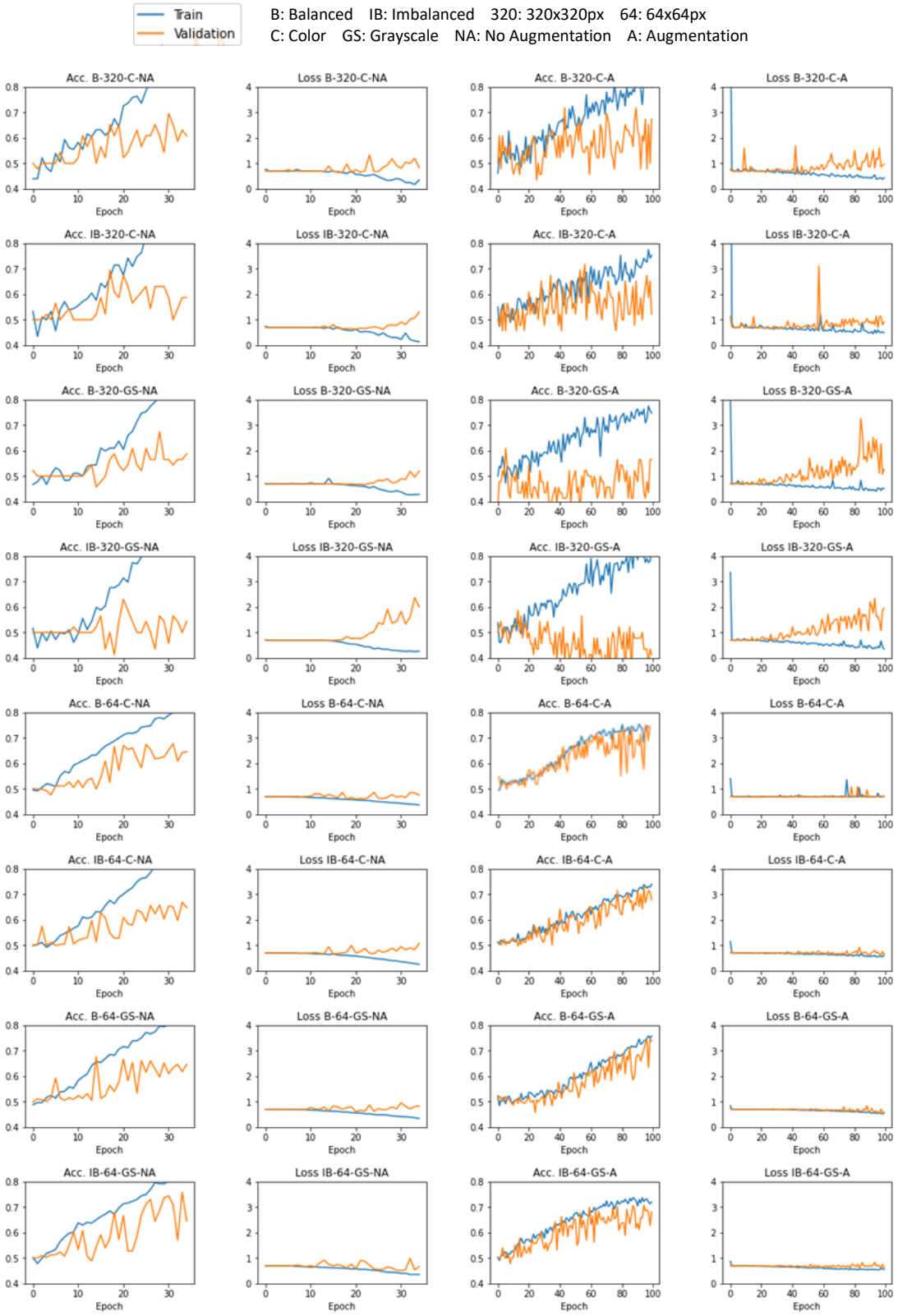


Figure 6 Comparison summary of experiment charts

The next table shows some of the data related to the previous charts, so it can be even more easy to compare with numbers:

Experiment	Dataset	Epochs at max. accuracy	Max. accuracy (%)	Execution time (s)
1	B-320-C-NA	20	65	37
2	B-320-C-A	50	65	600
3	IB-320-C-NA	18	70	33
4	IB-320-C-A	57	72	600
5	B-320-GS-NA	29	67	42
6	B-320-GS-A	44	72	180
7	IB-320-GS-NA	16	56	42
8	IB-320-GS-A	14	59	210
9	B-64-C-NA	5	48	33
10	B-64-C-A	65	72	420
11	IB-64-C-NA	16	63	42
12	IB-64-C-A	200	73	900
13	B-64-GS-NA	6	59	34
14	B-64-GS-A	100	75	240
15	IB-64-GS-NA	35	75	32
16	IB-64-GS-A	100	70	240

B: Balanced IB: Imbalanced 320: 320x320px 64: 64x64px C: Color GS: Grayscale NA: No Augmentation A: Augmentation

Table 5 Summary of experiments

NOTE: The heatmap in the previous table is only support data and doesn't conclude about any topic. I'm developing the experiment conclusions in the next sub-section.

Some observations I can made looking at both the charts and the table are:

- The model requires more development for datasets without augmentation. Most of them show a divergence between the accuracy during training and the accuracy during validation, around Epochs 10 to 20.
- There is not an evident difference in the accuracy of the models between color and grayscale datasets.
- When using datasets with augmentation, the model behaves much better on 64x64px tiles than in 320x320px tiles. In this case, both color and grayscale datasets behave quite similar.
- The validation loss diverges more from the training loss when using 320x320px datasets, and this is true for both datasets without augmentation and for datasets with augmentation.
- There is not a real impact between the balanced and imbalanced datasets. The accuracies and the loses are similar.

5.20. Experiment conclusions

After reviewing the results of the trainings and validations of the models in combination with all the datasets, next are the experiment conclusions:

- There is not a significant difference between the results when using imbalanced or balanced datasets.
- There is not a significant difference between the results when using color or grayscale datasets.
- There is a significant difference between the results when using 320x320px and 64x64px datasets, positive for 64x64px tiles. The model doesn't do a good job when training using 320x320px tiles.
- Data augmentation has a positive impact in the training of the models.
- The best-found dataset-training combination is the model of the Experiment 14 with a balanced dataset having grayscale 64x64px, with an accuracy of 79% after 400 Epochs, that requires 18 minutes of execution time using a Tesla K80 GPU in Google Colab.
- Alternatively, the next best-found dataset-training combination is the model of the Experiment 14 with a balanced dataset having grayscale 64x64px, with an accuracy of 75% after 100 Epochs, that requires 4 minutes of execution time using a Tesla K80 GPU in Google Colab.

6. Conclusions

In this section I'll go thru the final conclusions, first separated by state-of-the-art conclusions, dataset creation related conclusions and experiment results conclusions.

In regards of the **state-of-the-art conclusions**, after reading several papers related to the usage of machine learning to detect defects in printed paper at high-speeds, next are the most important findings:

- Deep learning has been successfully applied to classification tasks in many fields due to its good performance in learning discriminative features but the application to printing defect classification is very rare (Zhang, Li, Li, & Chen, 2019). I've not found any paper disclosing a method or system that uses machine learning to detect and classify defects in printed paper, using digital (variable content) high-speed printers.
- Black and white captures would not allow assess color defects (Ishimaru, Hata, & Hirokari, 2002). As we will see later on in this same section, grayscale datasets are good enough to detect scratches.
- SCRATCHES are difficult to detect with general purpose methods. Scratches have special characteristics (very thin, very light contrast vs. background) that may require Specific Scratch Detector systems. Previous studies detected only 67% of the scratches using specific scratch detection systems. (Vans, et al., 2010)
- Small and imbalanced datasets is a problem. Anomalous instances are generally rare, whereas normal instances account for a significant proportion. It is almost impossible to capture large number of abnormal data containing all anomaly types. (Wang, Zhang, Guo, & Han, 2020)
- Real-time (due to high printing speed) is a problem. (Zhang W. , 2020) and (Wang, Zhang, Guo, & Han, 2020)

The previous findings in previous literature match quite good with the results of the experimentation, as I will detail below.

In regards of the **datasets**, next are my **conclusions**:

- It is not easy to create a good dataset from scratch. This is in terms of obtaining images that would be representative enough of the real world, image quality, quantity of elements, and balanced enough so it can be used in a machine learning system.
- Creating a good dataset is time consuming and, even part of the process can be automated (tiling, conversion to grayscale, etc.), there is still a classification that needs to be done by expert human eyes. In the case of the scratches, I had to inspect tile by tile to determine if a tile had (or not) a scratch on it.
- The difference between a tile with or without scratch can be very subtle and can often be confused with noise in the image:

Figure 7 Tiles with subtle scratch (LEFT) vs. images without scratch (RIGHT)

  This may also be hard for the machine learning system to detect.

- Using Undersampling to create balanced datasets is a valid method but may remove important data that could potentially create a better dataset. Undersampling is omitting information.
- Data Augmentation resulted to be a valid method to increase the number of samples in the dataset. The technique has to be designed accurately so no noise is introduced into the dataset. This means, for example, that filling gaps because of a rotation may create a pattern that the machine learning algorithm may understand as the defect.

In regards of the **experiment conclusions**, next are my findings and observations:

- The model requires more development for datasets without augmentation. Most of them show a divergence between the accuracy during training and the accuracy during validation, around Epochs 10 to 20.
- There is not an evident difference in the accuracy of the models between color and grayscale datasets, but grayscale datasets have 1/3 of the data so can be processed faster. If defect detection related to color is not required, grayscale datasets are enough for scratch detection.
- When using datasets with augmentation, the model behaves much better on 64x64px tiles than in 320x320px tiles. In this case, both color and grayscale datasets behave quite similar.

- Data augmentation has a positive impact in the training of the models. The accuracy grows much more over epochs, even it requires more epochs to reach better accuracy.
- The validation loss diverges more from the training loss when using 320x320px datasets, and this is true for both datasets without augmentation and for datasets with augmentation.
- Looking at the data, there is not a real impact between the balanced and imbalanced datasets. The accuracies and the loses are similar. Even this, it is well known that imbalanced datasets trend to overtrain.
- The best-found dataset-training combination is the model of the Experiment 14 with a balanced dataset having grayscale 64x64px, with an accuracy of 79% after 400 Epochs, that requires 18 minutes of execution time using a Tesla K80 GPU in Google Colab.
- Alternatively, the next best-found dataset-training combination is the model of the Experiment 14 with a balanced dataset having grayscale 64x64px, with an accuracy of 75% after 100 Epochs, that requires 4 minutes of execution time using a Tesla K80 GPU in Google Colab.

Now that we have concluded about the different parts of the research, let's try to provide response to the objectives of this Master's Final Work:

- **Find actual artificial intelligence-based solutions or methods to detect and classify defects on printed content, in printing systems running at high-speed, or actual solutions or methods in other kind of surfaces that could be leveraged.**
  This has been done, by finding previous literature about defect detection and classification in printing systems running at high-speed, and summarizing the findings and conclusions in this report.
- **Find areas of improvement or specificity for printing systems running at high-speed.**
  Machine learning has been tested as a solution to detect scratches in printed content, without needing to compare the printout with the original image. This required a dataset created in-purpose and a model to be trained. The accuracy has been found to be up to 75% to 79%, which is higher than the accuracy of 67% reported in previous studies using specific scratch detection systems (Vans, et al., 2010).
- **Choose datasets and techniques to implement a proposal of solution or method and compare the obtained results.**

This has been done and I have compared the accuracy and the loss for every dataset-training combination. Balanced grayscale 64x64px datasets have reported an accuracy of 75% to 79% when being trained with the proposed model.

- **Propose a solution or method to be implemented in high-speed printing systems.**

  The proposed solution to be implemented in high-speed printing systems to detect scratches in printed content is to train a machine learning model with a good dataset, so images captured by a Vision System can be tested with the model to determine if there are scratches on it.

  By tiling the captured images to smaller size tiles, can help locating the scratch in the original image.

  This would require training the model before starting printing but, as printing systems require some time to initialize, it can be done in parallel, so it's not really a waste of time.

- **Determine limitations to a viable solution.**

  There are several limitations to the viability of the proposed solution.

  - The accuracy of 75% to 79% may not be enough for systems requiring high-precision.
  - Machine Learning works as a black box. It's almost impossible to troubleshoot what rules have been applied to determine if a tile has or has not a scratch on it.
  - Printing at very-high-speeds (up to 1000fpm in HP PageWide Web Presses) makes it unrealistic to capture every single printed frame (page), have it converted to grayscale, tiled, and verified by the trained machine learning model fast enough to report findings. If the application accepts sampling (analyze only a subset of captures), that would be enough but, for applications requiring high level of inspections, the solution may not work because of technology limitations (network bandwidth, processor, display, etc.

Next suggested steps, out of the scope of this work, are:

- Apply filters to the images to reduce noise.
- Do experiments with more in-deep models to avoid overtraining.
- Implement a full system to train the model, and automatically accept images from the vision system, so scratch detection results can be feedbacked to the operator of a high-speed printing system.

- Extend the scope so the system can not only detect scratches but also other kind of defects, and classify them.

Finally, in regards of the formal part of this Master's Final Work:

- The first idea for this Master's Final Work was to be able to detect and classify different defects in printed content. Given the difficulty of having good valid datasets, that are not in the field and would have to be created, I had to focus in one of the defects. Scratches was found to be an easy defect to be generated but a challenging one to detect.
- After redefining the scope of the Master's Final Work, I think all the objectives have been completed, even I would like to have more time and resources (hardware to be integrated to a real press) to better develop the model and be able to create an automated workflow so it could be tested in a real world system.
- I think the planning of this Master's Final Work has been very realistic, I have been able to follow it up without major difficulties, so I have delivered all the deliverables with the full expected content on-time. The risk assessment has been a key part of the planning phase, so I have been able to determine contingency plans for possible risks.

7. Glossary

Accuracy: Number of true positives and true negatives divided by the number of true positives, true negatives, false positives and false negatives.

Artificial Intelligence: A wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence.

Augmentation: Technique to artificially create new training data from existing training data. Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, etc.

Balanced dataset: Dataset where each output class (or target class) is represented by the same number of input samples.

Batch size: The number of examples in a subset of the dataset that will be used for the training, at the same time.

Convolutional Neural Network (CNN): A Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other.

Color dataset: A group if images that contain 3 channels, one for Red (R), one for Green (G) and one for Blue (B).

Dataset: A collection of data pieces that can be treated by a computer as a single unit for analytic and prediction purposes.

Deep learning: Considered an evolution of machine learning. It uses a programmable neural network that enables machines to make accurate decisions without help from humans.

Epochs: The number of passes of the entire training dataset the machine learning algorithm has completed.

Grayscale dataset: A group of images that contain 1 channel only, that represents the lightness of the gray.

ImageDataGenerator: From TensorFlow Keras, augments the images from a dataset in real-time while the model is still training. Can apply any random transformations on each training image as it is passed to the model.

Imbalanced dataset: Dataset where target class has an uneven distribution of observations.

Keras: A high-level neural networks library running on top of TensorFlow. Using Keras in deep learning allows for easy and fast prototyping as well as running seamlessly on CPU and GPU. This framework is written in Python code which is easy to debug and allows ease for extensibility.

Loss: The penalty for bad prediction. The number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero.

Machine Learning: An application of artificial intelligence that includes algorithms that parse data, learn from that data, and then apply what they've learned to make informed decisions.

Step: A training step is one gradient update. During one step an amount of "batch size" examples are processed.

TensorFlow: End-to-end open-source platform for machine learning. A comprehensive and flexible ecosystem of tools, libraries and other resources that provide workflows with high-level APIs. The framework offers various levels of concepts to build and deploy machine learning models.

Tile: A part resulting of dividing a full image.

Training phase: In machine learning, the phase on which the algorithm is trained to find the rules that will create the right output.

Validation phase: In machine learning, the phase on which the algorithm is tested to estimate how well the mode has been trained.

Bibliography and consulted references

Amidi, A., & Amidi, S. (n.d.). *A detailed example of how to use data generators with Keras*. Retrieved November 15, 2021, from Shervine Amidi's Blog: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

Bhandari, A. (2020, August 11). *Image Augmentation on the fly using Keras ImageDataGenerator!* Retrieved October 29, 2021, from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/

Brownlee, J. (2019, October 3). *Display Deep Learning Model Training History in Keras*. Retrieved November 15, 2021, from Machine Learning Mastery: https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

Brownlee, J. (2019, July 5). *How to Configure Image Data Augmentation in Keras*. Retrieved November 15, 2021, from Machine Learning Mastery: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

Chollet, F. (2020, April 28). *Image classification from scratch*. Retrieved November 15, 2021, from Keras: https://keras.io/examples/vision/image_classification_from_scratch/

DanB. (2020, October 15). *Data Augmentation*, 23. Retrieved October 29, 2021, from Kaggle: https://www.kaggle.com/dansbecker/data-augmentation

Han, Y.-J., & Yu, H.-J. (2020, April 6). Fabric Defect Detection System Using Stacked Convolutional Denoising Auto-Encoders Trained with Synthetic Defect Data. *Applied Sciences, 10*(7), 1-10. doi:10.3390/app10072511

He, H., & Garcia, E. A. (2009, June 29). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering, 21*(9), 1263-1284. doi:10.1109/TKDE.2008.239

Ishimaru, I., Hata, S., & Hirokari, M. (2002). Color-defect classification for printed-matter visual inspection system. *4th World Congress on Intelligent Control and Automation* (pp. 3261-3265). Shanghai, China: IEEE. doi:10.1109/WCICA.2002.1020137

Li, D.-C., Chen, H.-Y., & Shi, Q.-S. (2018, May 24). Learning from small datasets containing nominal attributes. *Neurocomputing, 291*, 226-236. doi:10.1016/j.neucom.2018.02.069

Ou, Y., Tao, H., Xuan, G., & Baoping, G. (2017). An Automation System for High-Speed Detection of Printed Matter and Defect Recognition. *2007 IEEE International Conference on Integration Technology* (pp. 213-217). Shenzhen, China: IEEE. doi:10.1109/ICITECHNOLOGY.2007.4290463

Shankar, N., Ravi, N., & Zhong, Z. (2008, November 12). A real-time print-defect detection system for web offset printing. *Measurement, 42*(5), 645-652. doi:10.1016/j.measurement.2008.10.012

Tanimizu, K., Meguro, S., & Ishii, A. (1990). High-speed Defect Detection Method for Color Printer Matter. *16th Annual Conference of IEEE Industrial Electronics Society. 1*, pp. 653-658. Pacific Grove, CA, USA: IEEE. doi:10.1109/IECON.1990.149219

TensorFlow. (2021, November 5). *tf.keras.utils.image_dataset_from_directory*, 2.7.0. Retrieved November 15, 2021, from TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory

Ugbeme, O., Wu, W., & Sabel, E. (2006). An automated defect classification algorithm for printed documents. *ICIS'06 International Congress of Imaging Science* (pp. 317-320). Society for Imaging Science and Technology. Retrieved October 5, 2021, from https://www.imaging.org/site/PDFS/Papers/2006/ICIS-0-736/33759.pdf

unutbu. (2012, August 30). *How can I convert an RGB image into grayscale in Python?* Retrieved October 17, 2021, from Stack Overflow: https://stackoverflow.com/a/12201744

Vans, M., Schein, S., Staelin, C., Kisilev, P., Simske, S., Dagan, R., & Harush, S. (2010, June 21). Automatic visual inspection and defect detection on Variable Data Prints. (H. Laboratories, Ed.) *Journal of Electronic Imaging 20(1)*. doi:10.1117/1.3537837

Wang, L., Zhang, D., Guo, J., & Han, Y. (2020, December 3). Image Anomaly Detection Using Normal Data Only by Latent Space Resampling. *Applied Sciences, 10*(23), 1-19. doi:10.3390/app10238660

What is a Learning Curve in Machine Learning? (2020, July 30). Retrieved November 15, 2021, from Baeldung: https://www.baeldung.com/cs/learning-curve-ml

ZdaR. (2017, August 30). *Creating image tiles (m*n) of original image using Python and Numpy*. Retrieved October 13, 2021, from Stack Overflow: https://stackoverflow.com/a/45952399

Zhang, E., Li, B., Li, P., & Chen, Y. (2019, November 22). A Deep Learning Based Printing Defect Classification Method with Imbalanced Samples. *Symmetry, 11*(12), 1440-. doi:10.3390/sym11121440

Zhang, W. (2020, December 11). *Surface Defect Detection: Dataset & Papers*, 1.0. Retrieved September 22, 2021, from GitHub: https://github.com/Charmve/Surface-Defect-Detection

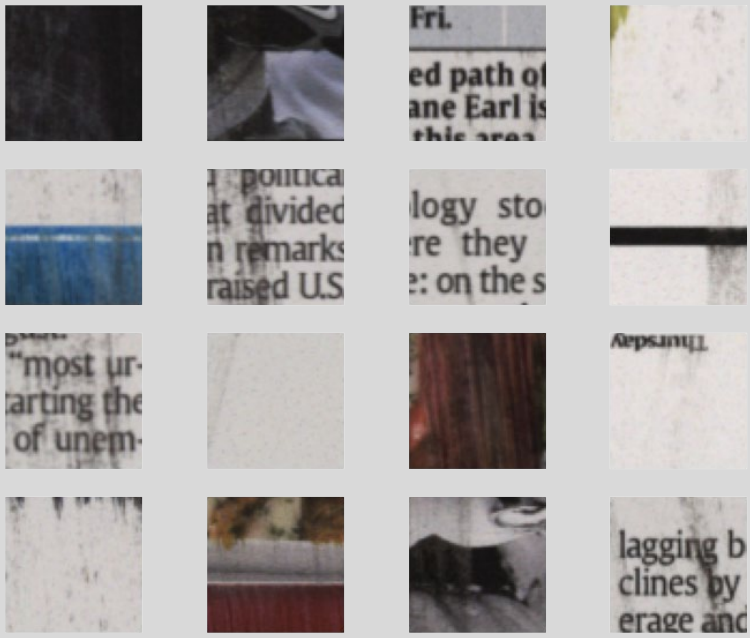
Appendix 1. Sample of the dataset with and without scratches

```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles64/grayscale/scratched/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs, cmap = 'gray')

plt.show()
```

Output:

```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles64/color/scratched/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs)

plt.show()
```

Output:

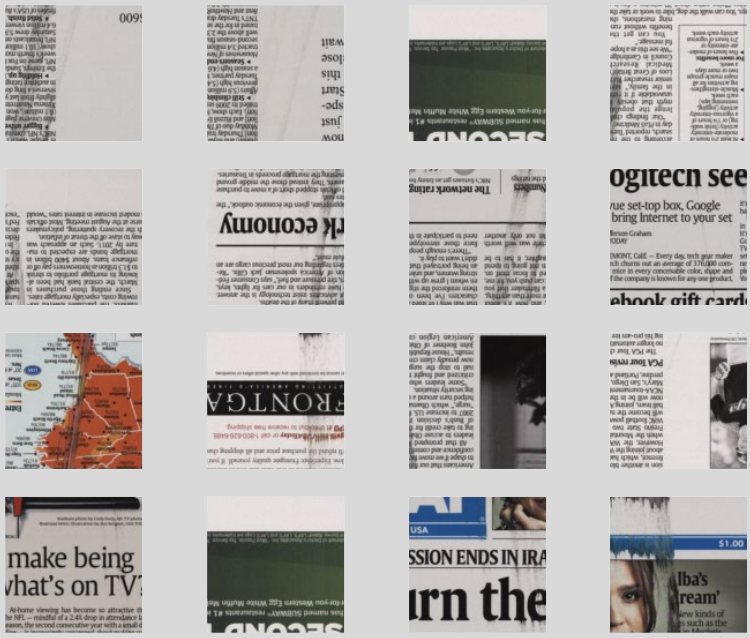


```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles320/grayscale/scratched/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs, cmap = 'gray')

plt.show()
```

Output:

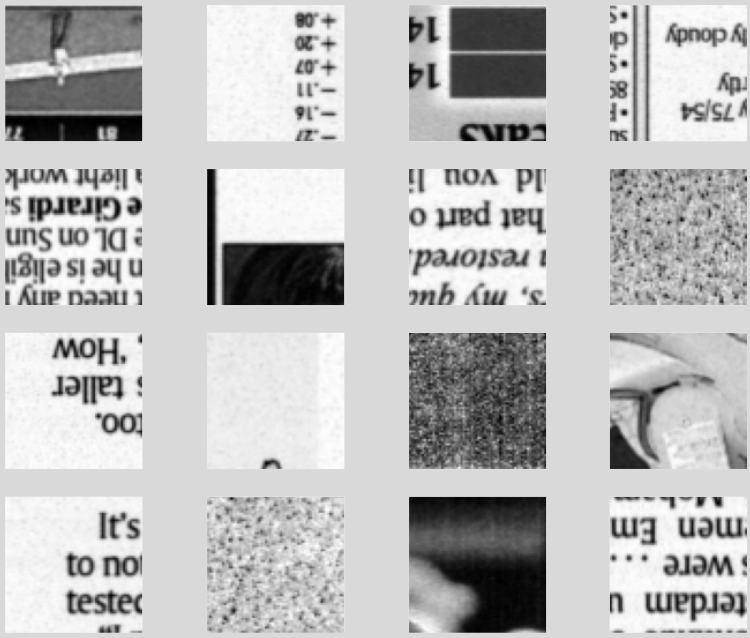
```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles320/color/scratched/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs)

plt.show()
```

Output:



```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles64/grayscale/notscratched/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs, cmap = 'gray')

plt.show()
```

Output:



```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles64/color/notscratched/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs)

plt.show()
```

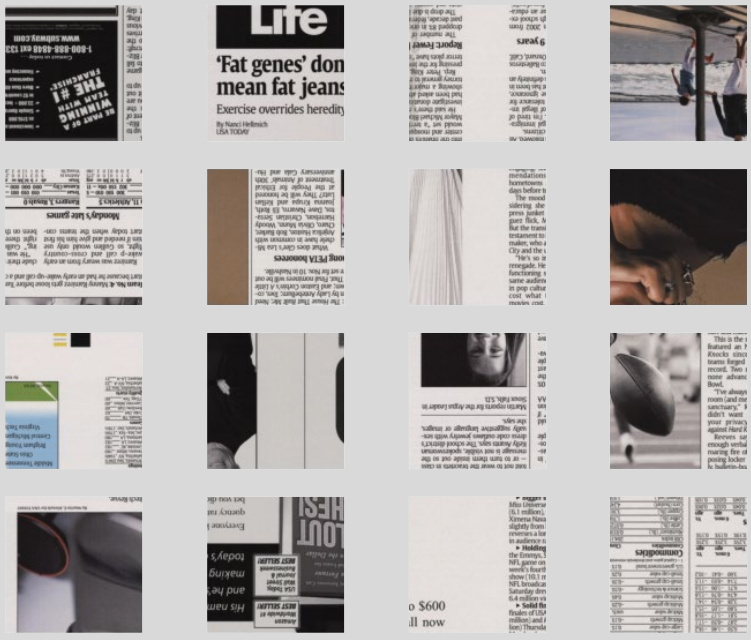
Output:

```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles320/grayscale/notscratched/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs, cmap = 'gray')

plt.show()
```

Output:

```
import random
import glob, os
from PIL import Image
from skimage.io import imread

import matplotlib.pyplot as plt

folder = "/content/datasets/scratches/defects/tiles320/color/notscratched/"
filetype = "*.png"

images = []

os.chdir(folder)
for file in glob.glob(filetype):
  images.append(os.path.join(folder, file))

plt.figure(1, figsize = (12, 10))
plt.axis('off')
n = 0
for i in range(16):
  n += 1
  randomimg = random.choice(images)
  imgs = imread(randomimg)
  plt.subplot(4, 4, n)
  plt.axis('off')
  plt.imshow(imgs)

plt.show()
```

Output:



Appendix 2. Code to balance the datasets

```
# Creates a balanced 64x64px grayscale dataset - Step 1
# 1s

!rm -rf datasets/scratches/defects/tiles64/grayscale/balanced

!mkdir datasets/scratches/defects/tiles64/grayscale/balanced
!mkdir datasets/scratches/defects/tiles64/grayscale/balanced/scratched
!mkdir datasets/scratches/defects/tiles64/grayscale/balanced/notscratched

print('Files in datasets/scratches/defects/tiles64/grayscale/scratched/:')
!ls datasets/scratches/defects/tiles64/grayscale/scratched | wc -l
print('Files in datasets/scratches/defects/tiles64/grayscale/notscratched/:')
!ls datasets/scratches/defects/tiles64/grayscale/notscratched | wc -l

!cp datasets/scratches/defects/tiles64/grayscale/scratched/*.png datasets/scratches/defe
cts/tiles64/grayscale/balanced/scratched/.
```

Output:

```
Files in datasets/scratches/defects/tiles64/grayscale/scratched/:
1222
Files in datasets/scratches/defects/tiles64/grayscale/notscratched/:
23754
```

```
# Creates a balanced 64x64px grayscale dataset - Step 2
# 1s

import os
import shutil
import random

files = [file for file in
         os.listdir('datasets/scratches/defects/tiles64/grayscale/notscratched/')
         if os.path.isfile(os.path.join('datasets/scratches/defects/tiles64/grayscale/no
tscratched/',
                                        file))]

random_amount = 1222
```

```
for x in range(random_amount):
  selection = random.randint(0, len(files)-1)
  file = files.pop(selection)
  shutil.copyfile(os.path.join('datasets/scratches/defects/tiles64/grayscale/notscratche
d',
                               file), os.path.join('datasets/scratches/defects/tiles64/g
rayscale/balanced/notscratched',
                                                   file))

print('Files in datasets/scratches/defects/tiles64/grayscale/balanced/scratched/:')
!ls datasets/scratches/defects/tiles64/grayscale/balanced/scratched | wc -l
print('Files in datasets/scratches/defects/tiles64/grayscale/balanced/notscratched/:')
!ls datasets/scratches/defects/tiles64/grayscale/balanced/notscratched | wc -l
```

Output:

```
Files in datasets/scratches/defects/tiles64/grayscale/balanced/scratched/:
1222
Files in datasets/scratches/defects/tiles64/grayscale/balanced/notscratched/:
1222
```

```
# Creates a balanced 320x320px grayscale dataset - Step 1
# ls

!rm -rf datasets/scratches/defects/tiles320/grayscale/balanced

!mkdir datasets/scratches/defects/tiles320/grayscale/balanced
!mkdir datasets/scratches/defects/tiles320/grayscale/balanced/scratched
!mkdir datasets/scratches/defects/tiles320/grayscale/balanced/notscratched

print('Files in datasets/scratches/defects/tiles320/grayscale/scratched/:')
!ls datasets/scratches/defects/tiles320/grayscale/scratched | wc -l
print('Files in datasets/scratches/defects/tiles320/grayscale/notscratched/:')
!ls datasets/scratches/defects/tiles320/grayscale/notscratched | wc -l

!cp datasets/scratches/defects/tiles320/grayscale/scratched/*.png datasets/scratches/def
ects/tiles320/grayscale/balanced/scratched/.
```

Output:

```
Files in datasets/scratches/defects/tiles320/grayscale/scratched/:
114
Files in datasets/scratches/defects/tiles320/grayscale/notscratched/:
822
```

```
# Creates a balanced 320x320px grayscale dataset - Step 2
# 1s

import os
import shutil
import random

files = [file for file in
         os.listdir('datasets/scratches/defects/tiles320/grayscale/notscratched/')
         if os.path.isfile(os.path.join('datasets/scratches/defects/tiles320/grayscale/notscratched/',
                                        file))]

random_amount = 114
```

```
for x in range(random_amount):
  selection = random.randint(0, len(files)-1)
  file = files.pop(selection)
  shutil.copyfile(os.path.join('datasets/scratches/defects/tiles320/grayscale/notscratched',
                               file), os.path.join('datasets/scratches/defects/tiles320/grayscale/balanced/notscratched',
                                                   file))

print('Files in datasets/scratches/defects/tiles320/grayscale/balanced/scratched/:')
!ls datasets/scratches/defects/tiles320/grayscale/balanced/scratched | wc -l
print('Files in datasets/scratches/defects/tiles320/grayscale/balanced/notscratched/:')
!ls datasets/scratches/defects/tiles320/grayscale/balanced/notscratched | wc -l
```

Output:

```
Files in datasets/scratches/defects/tiles320/grayscale/balanced/scratched/:
114
Files in datasets/scratches/defects/tiles320/grayscale/balanced/notscratched/:
114
```

```
# Creates a balanced 64x6px color dataset - Step 1
# 1s

!rm -rf datasets/scratches/defects/tiles64/color/balanced

!mkdir datasets/scratches/defects/tiles64/color/balanced
!mkdir datasets/scratches/defects/tiles64/color/balanced/scratched
!mkdir datasets/scratches/defects/tiles64/color/balanced/notscratched

print('Files in datasets/scratches/defects/tiles64/color/scratched/:')
!ls datasets/scratches/defects/tiles64/color/scratched | wc -l
print('Files in datasets/scratches/defects/tiles64/color/notscratched/:')
!ls datasets/scratches/defects/tiles64/color/notscratched | wc -l

!cp datasets/scratches/defects/tiles64/color/scratched/*.png datasets/scratches/defects/tiles64/color/balanced/scratched/.
```

Output:

```
Files in datasets/scratches/defects/tiles64/color/scratched/:
1222
Files in datasets/scratches/defects/tiles64/color/notscratched/:
23753
```

```
# Creates a balanced 64x64px color dataset - Step 2
# 1s

import os
import shutil
import random

files = [file for file in \
         os.listdir('datasets/scratches/defects/tiles64/color/notscratched/')
         if os.path.isfile(os.path.join('datasets/scratches/defects/tiles64/color/notscr
atched/',
                                        file))]

random_amount = 1222

for x in range(random_amount):
  selection = random.randint(0, len(files)-1)
  file = files.pop(selection)
  shutil.copyfile(os.path.join('datasets/scratches/defects/tiles64/color/notscratched',
                               file), os.path.join('datasets/scratches/defects/tiles64/c
olor/balanced/notscratched',
                                                   file))

print('Files in datasets/scratches/defects/tiles64/color/balanced/scratched/:')
!ls datasets/scratches/defects/tiles64/color/balanced/scratched | wc -l
print('Files in datasets/scratches/defects/tiles64/color/balanced/notscratched/:')
!ls datasets/scratches/defects/tiles64/color/balanced/notscratched | wc -l
```

Output:

```
Files in datasets/scratches/defects/tiles64/color/balanced/scratched/:
1222
Files in datasets/scratches/defects/tiles64/color/balanced/notscratched/:
1222
```

```
# Creates a balanced 320x320px color dataset - Step 1
# 1s

!rm -rf datasets/scratches/defects/tiles320/color/balanced

!mkdir datasets/scratches/defects/tiles320/color/balanced
!mkdir datasets/scratches/defects/tiles320/color/balanced/scratched
!mkdir datasets/scratches/defects/tiles320/color/balanced/notscratched

print('Files in datasets/scratches/defects/tiles320/color/scratched/:')
!ls datasets/scratches/defects/tiles320/color/scratched | wc -l
print('Files in datasets/scratches/defects/tiles320/color/notscratched/:')
!ls datasets/scratches/defects/tiles320/color/notscratched | wc -l
```

```
!cp datasets/scratches/defects/tiles320/color/scratched/*.png datasets/scratches/defects
/tiles320/color/balanced/scratched/.
```

Output:

```
Files in datasets/scratches/defects/tiles320/color/scratched/:
114
Files in datasets/scratches/defects/tiles320/color/notscratched/:
822
```

```
# Creates a balanced 320x320px color dataset - Step 2
# 1s

import os
import shutil
import random

files = [file for file in
         os.listdir('datasets/scratches/defects/tiles320/color/notscratched/')
         if os.path.isfile(os.path.join('datasets/scratches/defects/tiles320/color/notsc
ratched/',
                                        file))]

random_amount = 114

for x in range(random_amount):
  selection = random.randint(0, len(files)-1)
  file = files.pop(selection)
  shutil.copyfile(os.path.join('datasets/scratches/defects/tiles320/color/notscratched',
                               file), os.path.join('datasets/scratches/defects/tiles320/
color/balanced/notscratched',
                                                   file))

print('Files in datasets/scratches/defects/tiles320/color/balanced/scratched/:')
!ls datasets/scratches/defects/tiles320/color/balanced/scratched | wc -l
print('Files in datasets/scratches/defects/tiles320/color/balanced/notscratched/:')
!ls datasets/scratches/defects/tiles320/color/balanced/notscratched | wc -l
```

Output:

```
Files in datasets/scratches/defects/tiles64/color/balanced/scratched/:
114
Files in datasets/scratches/defects/tiles64/color/balanced/notscratched/:
114
```

Appendix 3. Code to create the datasets and the CSV files for Scratched and NotScratched

```
# Creates the Datasets and the CSV file for Scratched and NotScratched
# Balanced 64x64 grayscale
# 1s

!rm -rf datasets/scratches/defects/tiles64/grayscale/balancedall
!mkdir datasets/scratches/defects/tiles64/grayscale/balancedall

!ls datasets/scratches/defects/tiles64/grayscale/balanced/scratched > scratched.txt
!ls datasets/scratches/defects/tiles64/grayscale/balanced/notscratched > notscratched.tx
t

!sed -i 's/.png/.png,1/g' scratched.txt
!sed -i 's/.png/.png,0/g' notscratched.txt

!echo "image_names,scratched_notscratched" > datasets/scratches/defects/tiles64/grayscal
e/scratched_notscratched.csv
!cat scratched.txt >> datasets/scratches/defects/tiles64/grayscale/scratched_notscratche
d.csv
!cat notscratched.txt >> datasets/scratches/defects/tiles64/grayscale/scratched_notscrat
ched.csv

!cp datasets/scratches/defects/tiles64/grayscale/balanced/scratched/* datasets/scratches
/defects/tiles64/grayscale/balancedall/.
!cp datasets/scratches/defects/tiles64/grayscale/balanced/notscratched/* datasets/scratc
hes/defects/tiles64/grayscale/balancedall/.

!cat datasets/scratches/defects/tiles64/grayscale/scratched_notscratched.csv
```

Output:

```
image_names,scratched_notscratched
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_40.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_41.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_10_41.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_11_40.png,1

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_9_22.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_19.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_27.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_31.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_10_0.png,0

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_9_29.png,0
```

```
# Creates the Datasets and the CSV file for Scratched and NotScratched
# Imbalanced 64x64 grayscale
# 80s

!rm -rf datasets/scratches/defects/tiles64/grayscale/imbalancedall
!mkdir datasets/scratches/defects/tiles64/grayscale/imbalancedall

!ls datasets/scratches/defects/tiles64/grayscale/scratched > scratched.txt
!ls datasets/scratches/defects/tiles64/grayscale/notscratched > notscratched.txt

!sed -i 's/.png/.png,1/g' scratched.txt
!sed -i 's/.png/.png,0/g' notscratched.txt

!echo "image_name,scratched_notscratched" > datasets/scratches/defects/tiles64/grayscale
/imbalanced_scratched_notscratched.csv
!cat scratched.txt >> datasets/scratches/defects/tiles64/grayscale/imbalanced_scratched_
notscratched.csv
!cat notscratched.txt >> datasets/scratches/defects/tiles64/grayscale/imbalanced_scratch
ed_notscratched.csv

!for i in datasets/scratches/defects/tiles64/grayscale/scratched/*; do cp "$i" datasets/
scratches/defects/tiles64/grayscale/imbalancedall/.; done
!for i in datasets/scratches/defects/tiles64/grayscale/notscratched/*; do cp "$i" datase
ts/scratches/defects/tiles64/grayscale/imbalancedall/.; done

!cat datasets/scratches/defects/tiles64/grayscale/imbalanced_scratched_notscratched.csv
```

Output:

```
image_name,scratched_notscratched
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_40.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_41.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_10_41.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_11_40.png,1

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_9_22.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_0.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_10.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_11.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_12.png,0

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_9_9.png,0
```

```
# Creates the Datasets and the CSV file for Scratched and NotScratched
# Balanced 320x320 grayscale
# 1s

!rm -rf datasets/scratches/defects/tiles320/grayscale/balancedall
!mkdir datasets/scratches/defects/tiles320/grayscale/balancedall

!ls datasets/scratches/defects/tiles320/grayscale/balanced/scratched > scratched.txt
!ls datasets/scratches/defects/tiles320/grayscale/balanced/notscratched > notscratched.t
xt

!sed -i 's/.png/.png,1/g' scratched.txt
!sed -i 's/.png/.png,0/g' notscratched.txt

!echo "image_names,scratched_notscratched" > datasets/scratches/defects/tiles320/graysca
le/scratched_notscratched.csv
!cat scratched.txt >> datasets/scratches/defects/tiles320/grayscale/scratched_notscratch
ed.csv
!cat notscratched.txt >> datasets/scratches/defects/tiles320/grayscale/scratched_notscra
tched.csv

!cp datasets/scratches/defects/tiles320/grayscale/balanced/scratched/* datasets/scratche
s/defects/tiles320/grayscale/balancedall/.
!cp datasets/scratches/defects/tiles320/grayscale/balanced/notscratched/* datasets/scrat
ches/defects/tiles320/grayscale/balancedall/.

!cat datasets/scratches/defects/tiles320/grayscale/scratched_notscratched.csv
```

Output:

```
image_names,scratched_notscratched
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_1_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_2_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_3_8.png,1

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_5_3.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_6.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_7.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_1_0.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_2_4.png,0
[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_4_1.png,0
```

```
# Creates the Datasets and the CSV file for Scratched and NotScratched
# Imbalanced 320x320 grayscale
# 1s

!rm -rf datasets/scratches/defects/tiles320/grayscale/imbalancedall
!mkdir datasets/scratches/defects/tiles320/grayscale/imbalancedall

!ls datasets/scratches/defects/tiles320/grayscale/scratched > scratched.txt
!ls datasets/scratches/defects/tiles320/grayscale/notscratched > notscratched.txt

!sed -i 's/.png/.png,1/g' scratched.txt
!sed -i 's/.png/.png,0/g' notscratched.txt

!echo "image_name,scratched_notscratched" > datasets/scratches/defects/tiles320/grayscal
e/imbalanced_scratched_notscratched.csv
!cat scratched.txt >> datasets/scratches/defects/tiles320/grayscale/imbalanced_scratched
_notscratched.csv
!cat notscratched.txt >> datasets/scratches/defects/tiles320/grayscale/imbalanced_scratc
hed_notscratched.csv

!cp datasets/scratches/defects/tiles320/grayscale/scratched/* datasets/scratches/defects
/tiles320/grayscale/imbalancedall/.
!cp datasets/scratches/defects/tiles320/grayscale/notscratched/* datasets/scratches/defe
cts/tiles320/grayscale/imbalancedall/.

!cat datasets/scratches/defects/tiles320/grayscale/imbalanced_scratched_notscratched.csv
```

Output:

```
image_name,scratched_notscratched
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_1_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_2_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_3_8.png,1

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_5_3.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_0.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_1.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_2.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_3.png,0

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_5_8.png,0
```

```
# Creates the Datasets and the CSV file for Scratched and NotScratched
# Balanced 64x64 color
# 1s

!rm -rf datasets/scratches/defects/tiles64/color/balancedall
!mkdir datasets/scratches/defects/tiles64/color/balancedall

!ls datasets/scratches/defects/tiles64/color/balanced/scratched > scratched.txt
!ls datasets/scratches/defects/tiles64/color/balanced/notscratched > notscratched.txt

!sed -i 's/.png/.png,1/g' scratched.txt
!sed -i 's/.png/.png,0/g' notscratched.txt

!echo "image_names,scratched_notscratched" > datasets/scratches/defects/tiles64/color/sc
ratched_notscratched.csv
!cat scratched.txt >> datasets/scratches/defects/tiles64/color/scratched_notscratched.cs
v
!cat notscratched.txt >> datasets/scratches/defects/tiles64/color/scratched_notscratched
.csv

!cp datasets/scratches/defects/tiles64/color/balanced/scratched/* datasets/scratches/def
ects/tiles64/color/balancedall/.
!cp datasets/scratches/defects/tiles64/color/balanced/notscratched/* datasets/scratches/
defects/tiles64/color/balancedall/.

!cat datasets/scratches/defects/tiles64/color/scratched_notscratched.csv
```

Output:

```
image_names,scratched_notscratched
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_40.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_41.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_10_41.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_11_40.png,1

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_9_22.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_23.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_4.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_9.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_10_0.png,0

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_9_23.png,0
```

```
# Creates the Datasets and the CSV file for Scratched and NotScratched
# Imbalanced 64x64 color
# 80s

!rm -rf datasets/scratches/defects/tiles64/color/imbalancedall
!mkdir datasets/scratches/defects/tiles64/color/imbalancedall

!ls datasets/scratches/defects/tiles64/color/scratched > scratched.txt
!ls datasets/scratches/defects/tiles64/color/notscratched > notscratched.txt

!sed -i 's/.png/.png,1/g' scratched.txt
!sed -i 's/.png/.png,0/g' notscratched.txt

!echo "image_name,scratched_notscratched" > datasets/scratches/defects/tiles64/color/imb
alanced_scratched_notscratched.csv
!cat scratched.txt >> datasets/scratches/defects/tiles64/color/imbalanced_scratched_nots
cratched.csv
!cat notscratched.txt >> datasets/scratches/defects/tiles64/color/imbalanced_scratched_n
otscratched.csv

!for i in datasets/scratches/defects/tiles64/color/scratched/*; do cp "$i" datasets/scra
tches/defects/tiles64/color/imbalancedall/.; done
!for i in datasets/scratches/defects/tiles64/color/notscratched/*; do cp "$i" datasets/s
cratches/defects/tiles64/color/imbalancedall/.; done

!cat datasets/scratches/defects/tiles64/color/imbalanced_scratched_notscratched.csv
```

Output:

```
image_name,scratched_notscratched
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_40.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_41.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_10_41.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_11_40.png,1

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_9_22.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_0.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_10.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_11.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_12.png,0

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_9_9.png,0
```

```
# Creates the Datasets and the CSV file for Scratched and NotScratched
# Balanced 320x320 color
# 1s

!rm -rf datasets/scratches/defects/tiles320/color/balancedall
!mkdir datasets/scratches/defects/tiles320/color/balancedall

!ls datasets/scratches/defects/tiles320/color/balanced/scratched > scratched.txt
!ls datasets/scratches/defects/tiles320/color/balanced/notscratched > notscratched.txt

!sed -i 's/.png/.png,1/g' scratched.txt
!sed -i 's/.png/.png,0/g' notscratched.txt

!echo "image_names,scratched_notscratched" > datasets/scratches/defects/tiles320/color/s
cratched_notscratched.csv
!cat scratched.txt >> datasets/scratches/defects/tiles320/color/scratched_notscratched.c
sv
!cat notscratched.txt >> datasets/scratches/defects/tiles320/clor/scratched_notscratched
.csv

!cp datasets/scratches/defects/tiles320/color/balanced/scratched/* datasets/scratches/de
fects/tiles320/color/balancedall/.
!cp datasets/scratches/defects/tiles320/color/balanced/notscratched/* datasets/scratches
/defects/tiles320/color/balancedall/.

!cat datasets/scratches/defects/tiles320/color/scratched_notscratched.csv
```

Output:

```
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_1_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_2_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_3_8.png,1

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_5_3.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_2_6.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_4_5.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_5_2.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_5_6.png,0

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_5_1.png,0
```

```
# Creates the Datasets and the CSV file for Scratched and NotScratched
# Imbalanced 320x320 color
# 1s

!rm -rf datasets/scratches/defects/tiles320/color/imbalancedall
!mkdir datasets/scratches/defects/tiles320/color/imbalancedall

!ls datasets/scratches/defects/tiles320/color/scratched > scratched.txt
!ls datasets/scratches/defects/tiles320/color/notscratched > notscratched.txt

!sed -i 's/.png/.png,1/g' scratched.txt
!sed -i 's/.png/.png,0/g' notscratched.txt

!echo "image_name,scratched_notscratched" > datasets/scratches/defects/tiles320/color/im
balanced_scratched_notscratched.csv
!cat scratched.txt >> datasets/scratches/defects/tiles320/color/imbalanced_scratched_not
scratched.csv
!cat notscratched.txt >> datasets/scratches/defects/tiles320/color/imbalanced_scratched_
notscratched.csv

!cp datasets/scratches/defects/tiles320/color/scratched/* datasets/scratches/defects/til
es320/color/imbalancedall/.
!cp datasets/scratches/defects/tiles320/color/notscratched/* datasets/scratches/defects/
tiles320/color/imbalancedall/.

!cat datasets/scratches/defects/tiles64/color/imbalanced_scratched_notscratched.csv
```

Output:

```
image_name,scratched_notscratched
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_1_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_2_8.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_3_8.png,1

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_5_3.png,1
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_0.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_1.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_2.png,0
def_20211001_113111_icf_compare_Any_White_100.PSB.VSLeft.bmp_0_3.png,0

[...]

def_20211001_113111_icf_compare_Any_White_99.PSA.VSLeft.bmp_5_8.png,0
```

Appendix 4. Samples of augmented datasets

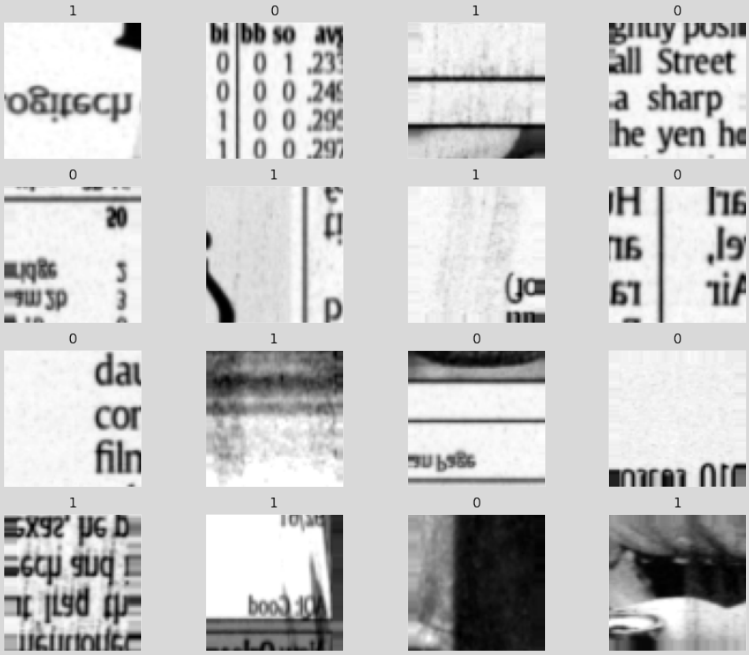
```
# Creates and defines the iterator to augment the images
# 2s

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        #rotation_range = 10, fill_mode = 'nearest', # Rotation
        #width_shift_range = 0.2,                    # Horizontal shift
        #height_shift_range = 0.2,                   # Vertical shift
        horizontal_flip = True,                     # Horizontal flip
        vertical_flip = True,                       # Vertical flip
        zoom_range = 0.2,                           # Zoom
        brightness_range = [0.2, 1.2])              # Brightness
```

Let's now use the ImgeDataGenerator:

```
# Shows samples of the resulting dataset, after augmentation
# 1s

import matplotlib.pyplot as plt

generator = datagen.flow_from_directory(
    directory = 'datasets/scratches/defects/tiles64/grayscale/balanced',
    target_size = (64, 64), # Resize to this size
    color_mode = "grayscale", # For color grayscale
    batch_size = 1, # Number of images to extract from folder for every batch
    class_mode = "binary", # Classes to predict
    seed = 2021 # To make the result reproducible
    )

plt.figure(figsize = (12, 10))

for i in range(16):
    image, label = generator.next()
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(image[0].squeeze(), cmap='gray')
    plt.title(int(label[0]))
    plt.axis("off")

plt.show()
```

Output:

```
Found 2444 images belonging to 2 classes.
```



As it can be observed, the images provided by the ImageDataGenerator have been transformed based on the configured parameters.

This can be, of course, done for the rest of the datasets.

```
# Shows samples of the resulting dataset, after augmentation
# 1s

import matplotlib.pyplot as plt
import numpy as np

generator = datagen.flow_from_directory(
    directory = 'datasets/scratches/defects/tiles64/color/balanced',
    target_size = (64, 64), # Resize to this size
    color_mode = "rgb", # For color images
    batch_size = 1, # Number of images to extract from folder for every batch
    class_mode = "binary", # Classes to predict
    seed = 2021 # To make the result reproducible
    )

plt.figure(figsize = (12, 10))

for i in range(16):
    image, label = generator.next()
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow((image[0]).astype(np.uint8))
    plt.title(int(label[0]))
    plt.axis("off")

plt.show()
```

Output:

```
Found 2444 images belonging to 2 classes.
```


```
# Shows samples of the resulting dataset, after augmentation
# 1s

import matplotlib.pyplot as plt

generator = datagen.flow_from_directory(
    directory = 'datasets/scratches/defects/tiles320/grayscale/balanced',
    target_size = (320, 320), # Resize to this size
    color_mode = "grayscale", # For color grayscale
    batch_size = 1, # Number of images to extract from folder for every batch
    class_mode = "binary", # Classes to predict
    seed = 2021 # To make the result reproducible
    )

plt.figure(figsize = (12, 10))

for i in range(16):
    image, label = generator.next()
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(image[0].squeeze(), cmap='gray')
    plt.title(int(label[0]))
    plt.axis("off")

plt.show()
```

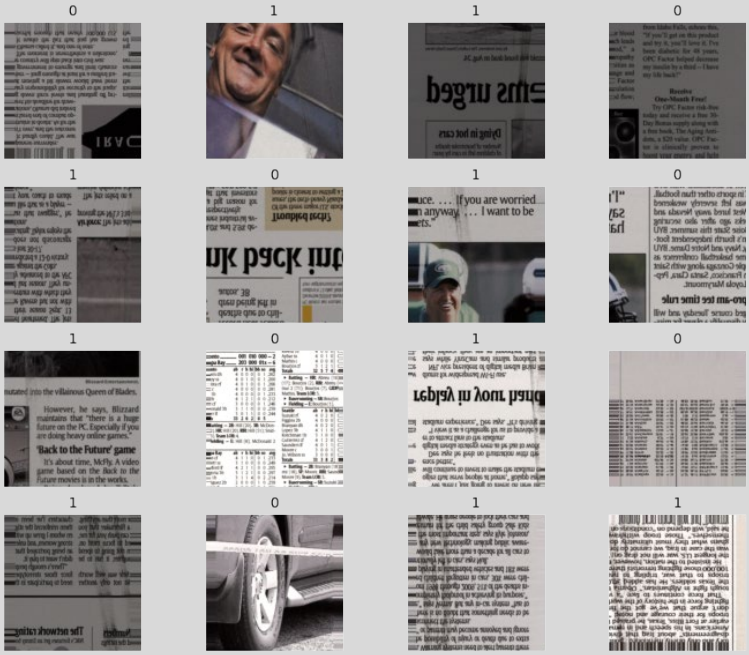
Output:

```
Found 228 images belonging to 2 classes.
```


```
# Shows samples of the resulting dataset, after augmentation
# 1s

import matplotlib.pyplot as plt
import numpy as np

generator = datagen.flow_from_directory(
    directory = 'datasets/scratches/defects/tiles320/color/balanced',
    target_size = (320, 320), # Resize to this size
    color_mode = "rgb", # For color images
    batch_size = 1, # Number of images to extract from folder for every batch
    class_mode = "binary", # Classes to predict
    seed = 2021 # To make the result reproducible
    )

plt.figure(figsize = (12, 10))

for i in range(16):
    image, label = generator.next()
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow((image[0]).astype(np.uint8))
    plt.title(int(label[0]))
    plt.axis("off")

plt.show()
```

Output:

```
Found 228 images belonging to 2 classes.
```



Appendix 5. Hardware used during the experiments

```
import tensorflow as tf
from tensorflow.python.client import device_lib
import subprocess
import platform

print(tf.test.gpu_device_name())
print(device_lib.list_local_devices())
!cat /proc/meminfo
print((subprocess.check_output("lscpu", shell = True).strip()).decode())
print(platform.platform())
!nvidia-smi
```

Output:

```
/device:GPU:0
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 2403921563350414642
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 11321147392
locality {
  bus_id: 1
  links {
  }
}
incarnation: 2260562214069817586

physical_device_desc: "device: 0, name: Tesla K80, pci bus id: 0000:00:04.0, compute
capability: 3.7"
xla_global_id: 416903419
]

MemTotal:       13302924 kB
MemFree:          216300 kB
MemAvailable:    9186412 kB
Buffers:          150184 kB
Cached:          4344164 kB
SwapCached:            0 kB
Active:          5381980 kB
Inactive:        7155216 kB
Active(anon):    3512712 kB
Inactive(anon):     5580 kB
Active(file):    1869268 kB
Inactive(file):  7149636 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
```

```
Dirty:               300 kB
Writeback:             0 kB
AnonPages:       8042740 kB
Mapped:          1660960 kB
Shmem:              6332 kB
KReclaimable:     277156 kB
Slab:             338240 kB
SReclaimable:     277156 kB
SUnreclaim:        61084 kB
KernelStack:        6688 kB
PageTables:        53720 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     6651460 kB
Committed_AS:   24381504 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       48540 kB
VmallocChunk:          0 kB
Percpu:             1440 kB
AnonHugePages:   1148928 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:         0 kB
FilePmdMapped:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      291648 kB
DirectMap2M:    10190848 kB
DirectMap1G:     5242880 kB

Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  2
Core(s) per socket:  1
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               63
Model name:          Intel(R) Xeon(R) CPU @ 2.30GHz
Stepping:            0
CPU MHz:             2299.998
BogoMIPS:            4599.99
Hypervisor vendor:   KVM
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            46080K
NUMA node0 CPU(s):   0,1
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good
nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1
sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
invpcid_single ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid
xsaveopt arat md_clear arch_capabilities
```

```
Linux-5.4.104+-x86_64-with-Ubuntu-18.04-bionic

Sat Dec  4 22:16:22 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   73C    P0    75W / 149W |   4380MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Appendix 6. Code to prepare the datasets and for chars of Experiments

Experiment 1

Next is the code to create the Training and Validation sub-datasets for this experiment:

```
# Creates the Training and Validation subsets of the dataset from CSV
# randomly, in folders
# No augmentation aplied yet
# 1s

!rm -rf datasets/scratches/defects/tiles320/color/balancedall/train
!rm -rf datasets/scratches/defects/tiles320/color/balancedall/valid

import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split

# Folder containing the tiles and CSV
home_path = r'datasets/scratches/defects/tiles320/color/balancedall'

# Creates train and validation folders
train_path = os.path.join(home_path, 'train')
os.mkdir(train_path)
val_path = os.path.join(home_path, 'valid')
os.mkdir(val_path)

# Creates sub-folders for scratched and notscratched tiles in train and
# validation folders
scratched_train_path = os.path.join(home_path + r'/train', 'scratched')
os.mkdir(scratched_train_path)

notscratched_train_path = os.path.join(home_path + r'/train', 'notscratched')
os.mkdir(notscratched_train_path)

scratched_val_path = os.path.join(home_path + r'/valid', 'scratched')
os.mkdir(scratched_val_path)

notscratched_val_path = os.path.join(home_path + r'/valid', 'notscratched')
os.mkdir(notscratched_val_path)

# Original DataFrame with the data from the CSV
df = pd.read_csv('datasets/scratches/defects/tiles320/color/scratched_notscratched.csv')

# Images and Categories
X = df.loc[:,'image_name']
y = df.loc[:,'scratched_notscratched']

# Splits data into Training and Validation
train_x, val_x, train_y, val_y = train_test_split(X, y,
                                                  test_size = 0.2,
                                                  random_state = 2021,
                                                  stratify = y)
```

```
# Train DataFrame
df_train = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_train['image_name'] = train_x
df_train['scratched_notscratched'] = train_y

# Validation DataFrame
df_valid = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_valid['image_name'] = val_x
df_valid['scratched_notscratched'] = val_y

# Resets indexes
df_train.reset_index(drop = True, inplace = True)
df_valid.reset_index(drop = True, inplace = True)

# Copy train images to sub-folder
for i in range(len(df_train)):

    image = df_train.loc[i, 'image_name']

    if df_train.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_train_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_train_path)

# Copy validation images to sub-folder
for i in range(len(df_valid)):

    image = df_valid.loc[i,'image_name']

    if df_valid.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_val_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_val_path)

# Loads the tiles into the Train and Validation arrays
# 1s

import cv2
from tensorflow import keras
import numpy as np

# Images
train_images = df_train.loc[:,'image_name']
train_labels = df_train.loc[:,'scratched_notscratched']

valid_images = df_valid.loc[:,'image_name']
valid_labels = df_valid.loc[:,'scratched_notscratched']

# Train images
x_train = []
for i in train_images:
    image = home_path + '/' + i
    img = cv2.imread(image) # Color
    x_train.append(img.squeeze())

# Train labels
y_train = keras.utils.to_categorical(train_labels)

# Validation images
x_valid = []
for i in valid_images:
    image = home_path + '/' +i
    img = cv2.imread(image) # Color
    x_valid.append(img.squeeze())
```

```
# Validation labels
y_valid = keras.utils.to_categorical(valid_labels)

# Normalize images
x_train = np.array(x_train, dtype = "float") / 255.0
x_valid = np.array(x_valid, dtype = "float") / 255.0
```

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way
# 1s

import matplotlib.pyplot as plt

# List all data in history
print(history_exp1.history.keys())

# Summarize history for accuracy
plt.plot(history_exp1.history['accuracy'])
plt.plot(history_exp1.history['val_accuracy'])
plt.title('Model accuracy (Balanced 320x320px Color WITHOUT Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp1.history['loss'])
plt.plot(history_exp1.history['val_loss'])
plt.title('Model loss (Balanced 320x320px Color WITHOUT Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```

Experiment 2

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp2.history.keys())
```

```
# Summarize history for accuracy
plt.plot(history_exp2.history['accuracy'])
plt.plot(history_exp2.history['val_accuracy'])
plt.title('Model accuracy (Balanced 320x320px Color WITH Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp2.history['loss'])
plt.plot(history_exp2.history['val_loss'])
plt.title('Model loss (Balanced 320x320px Color WITH Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.ylim(0, 4)
plt.show()
```

Experiment 3

Next is the code to create the Training and Validation sub-datasets for this experiment:

```
# Creates the Training and Validation subsets of the dataset from CSV
# randomly, in folders
# No augmentation aplied yet
# ls

!rm -rf datasets/scratches/defects/tiles320/color/imbalancedall/train
!rm -rf datasets/scratches/defects/tiles320/color/imbalancedall/valid

import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split

# Folder containing the tiles and CSV
home_path = r'datasets/scratches/defects/tiles320/color/imbalancedall'

# Creates train and validation folders
train_path = os.path.join(home_path, 'train')
os.mkdir(train_path)
val_path = os.path.join(home_path, 'valid')
os.mkdir(val_path)

# Creates sub-folders for scratched and notscratched tiles in train and
# validation folders
scratched_train_path = os.path.join(home_path + r'/train', 'scratched')
os.mkdir(scratched_train_path)

notscratched_train_path = os.path.join(home_path + r'/train', 'notscratched')
os.mkdir(notscratched_train_path)

scratched_val_path = os.path.join(home_path + r'/valid', 'scratched')
os.mkdir(scratched_val_path)

notscratched_val_path = os.path.join(home_path + r'/valid', 'notscratched')
os.mkdir(notscratched_val_path)
```

```
# Original DataFrame with the data from the CSV
df = pd.read_csv('datasets/scratches/defects/tiles320/color/scratched_notscratched.csv')

# Images and Categories
X = df.loc[:,'image_name']
y = df.loc[:,'scratched_notscratched']

# Splits data into Training and Validation
train_x, val_x, train_y, val_y = train_test_split(X, y,
                                                  test_size = 0.2,
                                                  random_state = 2021,
                                                  stratify = y)

# Train DataFrame
df_train = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_train['image_name'] = train_x
df_train['scratched_notscratched'] = train_y

# Validation DataFrame
df_valid = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_valid['image_name'] = val_x
df_valid['scratched_notscratched'] = val_y

# Resets indexes
df_train.reset_index(drop = True, inplace = True)
df_valid.reset_index(drop = True, inplace = True)

# Copy train images to sub-folder
for i in range(len(df_train)):

    image = df_train.loc[i, 'image_name']

    if df_train.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_train_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_train_path)

# Copy validation images to sub-folder
for i in range(len(df_valid)):

    image = df_valid.loc[i,'image_name']

    if df_valid.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_val_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_val_path)
# Loads the tiles into the Train and Validation arrays

import cv2
from tensorflow import keras
import numpy as np

# Images
train_images = df_train.loc[:,'image_name']
train_labels = df_train.loc[:,'scratched_notscratched']

valid_images = df_valid.loc[:,'image_name']
valid_labels = df_valid.loc[:,'scratched_notscratched']
```

```
# Train images
x_train = []
for i in train_images:
    image = home_path + '/' + i
    img = cv2.imread(image) # Color
    x_train.append(img.squeeze())

# Train labels
y_train = keras.utils.to_categorical(train_labels)

# Validation images
x_valid = []
for i in valid_images:
    image = home_path + '/' +i
    img = cv2.imread(image) # Color
    x_valid.append(img.squeeze())

# Validation labels
y_valid = keras.utils.to_categorical(valid_labels)

# Normalize images
x_train = np.array(x_train, dtype = "float") / 255.0
x_valid = np.array(x_valid, dtype = "float") / 255.0
```

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp3.history.keys())

# Summarize history for accuracy
plt.plot(history_exp3.history['accuracy'])
plt.plot(history_exp3.history['val_accuracy'])
plt.title('Model accuracy (Imbalanced 320x320px Color WITHOUT Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp3.history['loss'])
plt.plot(history_exp3.history['val_loss'])
plt.title('Model loss (Imbalanced 320x320px Color WITHOUT Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```

Experiment 4

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp4.history.keys())

# Summarize history for accuracy
plt.plot(history_exp4.history['accuracy'])
plt.plot(history_exp4.history['val_accuracy'])
plt.title('Model accuracy (Imbalanced 320x320px Color WITH Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp4.history['loss'])
plt.plot(history_exp4.history['val_loss'])
plt.title('Model loss (Imbalanced 320x320px Color WITH Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.ylim(0, 4)
plt.show()
```

Experiment 5

Next is the code to create the Training and Validation sub-datasets for this experiment:

```
# Creates the Training and Validation subsets of the dataset from CSV
# randomly, in folders
# No augmentation aplied yet
# 1s

!rm -rf datasets/scratches/defects/tiles320/grayscale/balancedall/train
!rm -rf datasets/scratches/defects/tiles320/grayscale/balancedall/valid

import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split

# Folder containing the tiles and CSV
home_path = r'datasets/scratches/defects/tiles320/grayscale/balancedall'

# Creates train and validation folders
train_path = os.path.join(home_path, 'train')
os.mkdir(train_path)
val_path = os.path.join(home_path, 'valid')
os.mkdir(val_path)

# Creates sub-folders for scratched and notscratched tiles in train and
# validation folders
scratched_train_path = os.path.join(home_path + r'/train', 'scratched')
os.mkdir(scratched_train_path)

notscratched_train_path = os.path.join(home_path + r'/train', 'notscratched')
os.mkdir(notscratched_train_path)
```

```
scratched_val_path = os.path.join(home_path + r'/valid', 'scratched')
os.mkdir(scratched_val_path)

notscratched_val_path = os.path.join(home_path + r'/valid', 'notscratched')
os.mkdir(notscratched_val_path)

# Original DataFrame with the data from the CSV
df = pd.read_csv('datasets/scratches/defects/tiles320/grayscale/scratched_notscratched.c
sv')

# Images and Categories
X = df.loc[:,'image_name']
y = df.loc[:,'scratched_notscratched']

# Splits data into Training and Validation
train_x, val_x, train_y, val_y = train_test_split(X, y,
                                                  test_size = 0.2,
                                                  random_state = 2021,
                                                  stratify = y)

# Train DataFrame
df_train = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_train['image_name'] = train_x
df_train['scratched_notscratched'] = train_y

# Validation DataFrame
df_valid = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_valid['image_name'] = val_x
df_valid['scratched_notscratched'] = val_y


# Resets indexes
df_train.reset_index(drop = True, inplace = True)
df_valid.reset_index(drop = True, inplace = True)

# Copy train images to sub-folder
for i in range(len(df_train)):

    image = df_train.loc[i, 'image_name']

    if df_train.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_train_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_train_path)

# Copy validation images to sub-folder
for i in range(len(df_valid)):

    image = df_valid.loc[i,'image_name']

    if df_valid.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_val_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_val_path)
# Loads the tiles into the Train and Validation arrays

import cv2
from tensorflow import keras
import numpy as np

# Images
train_images = df_train.loc[:,'image_name']
train_labels = df_train.loc[:,'scratched_notscratched']

valid_images = df_valid.loc[:,'image_name']
valid_labels = df_valid.loc[:,'scratched_notscratched']
```

```
# Train images
x_train = []
for i in train_images:
    image = home_path + '/' + i
    img = cv2.imread(image, 0) # Grayscale
    x_train.append(img.squeeze())

# Train labels
y_train = keras.utils.to_categorical(train_labels)

# Validation images
x_valid = []
for i in valid_images:
    image = home_path + '/' +i
    img = cv2.imread(image, 0) # Grayscale
    x_valid.append(img.squeeze())

# Validation labels
y_valid = keras.utils.to_categorical(valid_labels)

# Normalize images
x_train = np.array(x_train, dtype = "float") / 255.0
x_valid = np.array(x_valid, dtype = "float") / 255.0

x_train = x_train.reshape(182, 320, 320, 1)
x_valid = x_valid.reshape(46, 320, 320, 1)
```

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp5.history.keys())

# Summarize history for accuracy
plt.plot(history_exp5.history['accuracy'])
plt.plot(history_exp5.history['val_accuracy'])
plt.title('Model accuracy (Balanced 320x320px Grayscale WITHOUT Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp5.history['loss'])
plt.plot(history_exp5.history['val_loss'])
plt.title('Model loss (Balanced 320x320px Grayscale WITHOUT Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```

Experiment 6

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp2.history.keys())

# Summarize history for accuracy
plt.plot(history_exp2.history['accuracy'])
plt.plot(history_exp2.history['val_accuracy'])
plt.title('Model accuracy (Balanced 320x320px Color WITH Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp2.history['loss'])
plt.plot(history_exp2.history['val_loss'])
plt.title('Model loss (Balanced 320x320px Color WITH Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.ylim(0, 4)
plt.show()
```

Experiment 7

Next is the code to create the Training and Validation sub-datasets for this experiment:

```
# Creates the Training and Validation subsets of the dataset from CSV
# randomly, in folders
# No augmentation aplied yet
# 1s

!rm -rf datasets/scratches/defects/tiles320/grayscale/imbalancedall/train
!rm -rf datasets/scratches/defects/tiles320/grayscale/imbalancedall/valid

import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split

# Folder containing the tiles and CSV
home_path = r'datasets/scratches/defects/tiles320/grayscale/imbalancedall'

# Creates train and validation folders
train_path = os.path.join(home_path, 'train')
os.mkdir(train_path)
val_path = os.path.join(home_path, 'valid')
os.mkdir(val_path)
```

```
# Creates sub-folders for scratched and notscratched tiles in train and
# validation folders
scratched_train_path = os.path.join(home_path + r'/train', 'scratched')
os.mkdir(scratched_train_path)

notscratched_train_path = os.path.join(home_path + r'/train', 'notscratched')
os.mkdir(notscratched_train_path)

scratched_val_path = os.path.join(home_path + r'/valid', 'scratched')
os.mkdir(scratched_val_path)

notscratched_val_path = os.path.join(home_path + r'/valid', 'notscratched')
os.mkdir(notscratched_val_path)

# Original DataFrame with the data from the CSV
df = pd.read_csv('datasets/scratches/defects/tiles320/grayscale/scratched_notscratched.c
sv')

# Images and Categories
X = df.loc[:,'image_name']
y = df.loc[:,'scratched_notscratched']

# Splits data into Training and Validation
train_x, val_x, train_y, val_y = train_test_split(X, y,
                                                  test_size = 0.2,
                                                  random_state = 2021,
                                                  stratify = y)

# Train DataFrame
df_train = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_train['image_name'] = train_x
df_train['scratched_notscratched'] = train_y

# Validation DataFrame
df_valid = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_valid['image_name'] = val_x
df_valid['scratched_notscratched'] = val_y

# Resets indexes
df_train.reset_index(drop = True, inplace = True)
df_valid.reset_index(drop = True, inplace = True)

# Copy train images to sub-folder
for i in range(len(df_train)):

    image = df_train.loc[i, 'image_name']

    if df_train.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_train_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_train_path)

# Copy validation images to sub-folder
for i in range(len(df_valid)):

    image = df_valid.loc[i,'image_name']

    if df_valid.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_val_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_val_path)
```

```
# Loads the tiles into the Train and Validation arrays

import cv2
from tensorflow import keras
import numpy as np

# Images
train_images = df_train.loc[:,'image_name']
train_labels = df_train.loc[:,'scratched_notscratched']

valid_images = df_valid.loc[:,'image_name']
valid_labels = df_valid.loc[:,'scratched_notscratched']

# Train images
x_train = []
for i in train_images:
    image = home_path + '/' + i
    img = cv2.imread(image, 0) # Grayscale
    x_train.append(img.squeeze())

# Train labels
y_train = keras.utils.to_categorical(train_labels)

# Validation images
x_valid = []
for i in valid_images:
    image = home_path + '/' +i
    img = cv2.imread(image, 0) # Grayscale
    x_valid.append(img.squeeze())

# Validation labels
y_valid = keras.utils.to_categorical(valid_labels)

# Normalize images
x_train = np.array(x_train, dtype = "float") / 255.0
x_valid = np.array(x_valid, dtype = "float") / 255.0

x_train = x_train.reshape(182, 320, 320, 1)
x_valid = x_valid.reshape(46, 320, 320, 1)
```

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp7.history.keys())

# Summarize history for accuracy
plt.plot(history_exp7.history['accuracy'])
plt.plot(history_exp7.history['val_accuracy'])
plt.title('Model accuracy (Imbalanced 320x320px Grayscale WITHOUT Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```

```
# Summarize history for loss
plt.plot(history_exp7.history['loss'])
plt.plot(history_exp7.history['val_loss'])
plt.title('Model loss (Imbalanced 320x320px Grayscale WITHOUT Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```

Experiment 8

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp8.history.keys())

# Summarize history for accuracy
plt.plot(history_exp8.history['accuracy'])
plt.plot(history_exp8.history['val_accuracy'])
plt.title('Model accuracy (Imbalanced 320x320px Grayscale WITH Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp8.history['loss'])
plt.plot(history_exp8.history['val_loss'])
plt.title('Model loss (Imbalanced 320x320px Grayscale WITH Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.ylim(0, 4)
plt.show()
```

Experiment 9

Next is the code to create the Training and Validation sub-datasets for this experiment:

```
# Creates the Training and Validation subsets of the dataset from CSV
# randomly, in folders
# No augmentation aplied yet
# 1s

!rm -rf datasets/scratches/defects/tiles64/color/balancedall/train
!rm -rf datasets/scratches/defects/tiles64/color/balancedall/valid

import pandas as pd
```

```
import os
import shutil
from sklearn.model_selection import train_test_split

# Folder containing the tiles and CSV
home_path = r'datasets/scratches/defects/tiles64/color/balancedall'

# Creates train and validation folders
train_path = os.path.join(home_path, 'train')
os.mkdir(train_path)
val_path = os.path.join(home_path, 'valid')
os.mkdir(val_path)

# Creates sub-folders for scratched and notscratched tiles in train and
# validation folders
scratched_train_path = os.path.join(home_path + r'/train', 'scratched')
os.mkdir(scratched_train_path)

notscratched_train_path = os.path.join(home_path + r'/train', 'notscratched')
os.mkdir(notscratched_train_path)

scratched_val_path = os.path.join(home_path + r'/valid', 'scratched')
os.mkdir(scratched_val_path)

notscratched_val_path = os.path.join(home_path + r'/valid', 'notscratched')
os.mkdir(notscratched_val_path)

# Original DataFrame with the data from the CSV
df = pd.read_csv('datasets/scratches/defects/tiles64/color/scratched_notscratched.csv')

# Images and Categories
X = df.loc[:,'image_name']
y = df.loc[:,'scratched_notscratched']

# Splits data into Training and Validation
train_x, val_x, train_y, val_y = train_test_split(X, y,
                                                  test_size = 0.2,
                                                  random_state = 2021,
                                                  stratify = y)

# Train DataFrame
df_train = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_train['image_name'] = train_x
df_train['scratched_notscratched'] = train_y

# Validation DataFrame
df_valid = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_valid['image_name'] = val_x
df_valid['scratched_notscratched'] = val_y

# Resets indexes
df_train.reset_index(drop = True, inplace = True)
df_valid.reset_index(drop = True, inplace = True)

# Copy train images to sub-folder
for i in range(len(df_train)):

    image = df_train.loc[i, 'image_name']

    if df_train.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_train_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_train_path)
```

```
# Copy validation images to sub-folder
for i in range(len(df_valid)):

    image = df_valid.loc[i,'image_name']

    if df_valid.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_val_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_val_path)

# Loads the tiles into the Train and Validation arrays

import cv2
from tensorflow import keras
import numpy as np

# Images
train_images = df_train.loc[:,'image_name']
train_labels = df_train.loc[:,'scratched_notscratched']

valid_images = df_valid.loc[:,'image_name']
valid_labels = df_valid.loc[:,'scratched_notscratched']

# Train images
x_train = []
for i in train_images:
    image = home_path + '/' + i
    img = cv2.imread(image) # Color
    x_train.append(img.squeeze())

# Train labels
y_train = keras.utils.to_categorical(train_labels)

# Validation images
x_valid = []
for i in valid_images:
    image = home_path + '/' +i
    img = cv2.imread(image) # Color
    x_valid.append(img.squeeze())

# Validation labels
y_valid = keras.utils.to_categorical(valid_labels)

# Normalize images
x_train = np.array(x_train, dtype = "float") / 255.0
x_valid = np.array(x_valid, dtype = "float") / 255.0
```

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp9.history.keys())
```

```
# Summarize history for accuracy
plt.plot(history_exp9.history['accuracy'])
plt.plot(history_exp9.history['val_accuracy'])
plt.title('Model accuracy (Balanced 64x64px Color WITHOUT Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp9.history['loss'])
plt.plot(history_exp9.history['val_loss'])
plt.title('Model loss (Balanced 64x64px Color WITHOUT Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```

Experiment 10

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp10.history.keys())

# Summarize history for accuracy
plt.plot(history_exp10.history['accuracy'])
plt.plot(history_exp10.history['val_accuracy'])
plt.title('Model accuracy (Balanced 64x64px Color WITH Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp10.history['loss'])
plt.plot(history_exp10.history['val_loss'])
plt.title('Model loss (Balanced 64x64px Color WITH Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.ylim(0, 4)
plt.show()
```

Experiment 11

Next is the code to create the Training and Validation sub-datasets for this experiment:

```
# Creates the Training and Validation subsets of the dataset from CSV
# randomly, in folders
# No augmentation aplied yet
# 1s

!rm -rf datasets/scratches/defects/tiles64/color/imbalancedall/train
!rm -rf datasets/scratches/defects/tiles64/color/imbalancedall/valid

import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split

# Folder containing the tiles and CSV
home_path = r'datasets/scratches/defects/tiles64/color/imbalancedall'

# Creates train and validation folders
train_path = os.path.join(home_path, 'train')
os.mkdir(train_path)
val_path = os.path.join(home_path, 'valid')
os.mkdir(val_path)

# Creates sub-folders for scratched and notscratched tiles in train and
# validation folders
scratched_train_path = os.path.join(home_path + r'/train', 'scratched')
os.mkdir(scratched_train_path)

notscratched_train_path = os.path.join(home_path + r'/train', 'notscratched')
os.mkdir(notscratched_train_path)

scratched_val_path = os.path.join(home_path + r'/valid', 'scratched')
os.mkdir(scratched_val_path)

notscratched_val_path = os.path.join(home_path + r'/valid', 'notscratched')
os.mkdir(notscratched_val_path)

# Original DataFrame with the data from the CSV
df = pd.read_csv('datasets/scratches/defects/tiles64/color/scratched_notscratched.csv')

# Images and Categories
X = df.loc[:,'image_name']
y = df.loc[:,'scratched_notscratched']

# Splits data into Training and Validation
train_x, val_x, train_y, val_y = train_test_split(X, y,
                                                  test_size = 0.2,
                                                  random_state = 2021,
                                                  stratify = y)

# Train DataFrame
df_train = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_train['image_name'] = train_x
df_train['scratched_notscratched'] = train_y

# Validation DataFrame
df_valid = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_valid['image_name'] = val_x
df_valid['scratched_notscratched'] = val_y
```

```
# Resets indexes
df_train.reset_index(drop = True, inplace = True)
df_valid.reset_index(drop = True, inplace = True)

# Copy train images to sub-folder
for i in range(len(df_train)):

    image = df_train.loc[i, 'image_name']

    if df_train.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_train_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_train_path)

# Copy validation images to sub-folder
for i in range(len(df_valid)):

    image = df_valid.loc[i,'image_name']

    if df_valid.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_val_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_val_path)

# Loads the tiles into the Train and Validation arrays

import cv2
from tensorflow import keras
import numpy as np

# Images
train_images = df_train.loc[:,'image_name']
train_labels = df_train.loc[:,'scratched_notscratched']

valid_images = df_valid.loc[:,'image_name']
valid_labels = df_valid.loc[:,'scratched_notscratched']

# Train images
x_train = []
for i in train_images:
    image = home_path + '/' + i
    img = cv2.imread(image) # Color
    x_train.append(img.squeeze())

# Train labels
y_train = keras.utils.to_categorical(train_labels)

# Validation images
x_valid = []
for i in valid_images:
    image = home_path + '/' +i
    img = cv2.imread(image) # Color
    x_valid.append(img.squeeze())

# Validation labels
y_valid = keras.utils.to_categorical(valid_labels)

# Normalize images
x_train = np.array(x_train, dtype = "float") / 255.0
x_valid = np.array(x_valid, dtype = "float") / 255.0
```

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp11.history.keys())

# Summarize history for accuracy
plt.plot(history_exp11.history['accuracy'])
plt.plot(history_exp11.history['val_accuracy'])
plt.title('Model accuracy (Balanced 64x64px Color WITHOUT Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp11.history['loss'])
plt.plot(history_exp11.history['val_loss'])
plt.title('Model loss (Balanced 64x64px Color WITHOUT Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```

Experiment 12

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp12.history.keys())

# Summarize history for accuracy
plt.plot(history_exp12.history['accuracy'])
plt.plot(history_exp12.history['val_accuracy'])
plt.title('Model accuracy (Balanced 64x64px Color WITH Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```

```
# Summarize history for loss
plt.plot(history_exp12.history['loss'])
plt.plot(history_exp12.history['val_loss'])
plt.title('Model loss (Balanced 64x64px Color WITH Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.ylim(0, 4)
plt.show()
```

Experiment 13

Next is the code to create the Training and Validation sub-datasets for this experiment:

```
# Creates the Training and Validation subsets of the dataset from CSV
# randomly, in folders
# No augmentation aplied yet
# 1s

!rm -rf datasets/scratches/defects/tiles64/grayscale/balancedall/train
!rm -rf datasets/scratches/defects/tiles64/grayscale/balancedall/valid

import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split

# Folder containing the tiles and CSV
home_path = r'datasets/scratches/defects/tiles64/grayscale/balancedall'

# Creates train and validation folders
train_path = os.path.join(home_path, 'train')
os.mkdir(train_path)
val_path = os.path.join(home_path, 'valid')
os.mkdir(val_path)

# Creates sub-folders for scratched and notscratched tiles in train and
# validation folders
scratched_train_path = os.path.join(home_path + r'/train', 'scratched')
os.mkdir(scratched_train_path)

notscratched_train_path = os.path.join(home_path + r'/train', 'notscratched')
os.mkdir(notscratched_train_path)

scratched_val_path = os.path.join(home_path + r'/valid', 'scratched')
os.mkdir(scratched_val_path)

notscratched_val_path = os.path.join(home_path + r'/valid', 'notscratched')
os.mkdir(notscratched_val_path)

# Original DataFrame with the data from the CSV
df = pd.read_csv('datasets/scratches/defects/tiles64/grayscale/scratched_notscratched.cs
v')

# Images and Categories
X = df.loc[:,'image_name']
y = df.loc[:,'scratched_notscratched']
```

```
# Splits data into Training and Validation
train_x, val_x, train_y, val_y = train_test_split(X, y,
                                                  test_size = 0.2,
                                                  random_state = 2021,
                                                  stratify = y)

# Train DataFrame
df_train = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_train['image_name'] = train_x
df_train['scratched_notscratched'] = train_y

# Validation DataFrame
df_valid = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_valid['image_name'] = val_x
df_valid['scratched_notscratched'] = val_y

# Resets indexes
df_train.reset_index(drop = True, inplace = True)
df_valid.reset_index(drop = True, inplace = True)

# Copy train images to sub-folder
for i in range(len(df_train)):

    image = df_train.loc[i, 'image_name']

    if df_train.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_train_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_train_path)

# Copy validation images to sub-folder
for i in range(len(df_valid)):

    image = df_valid.loc[i,'image_name']

    if df_valid.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_val_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_val_path)

# Loads the tiles into the Train and Validation arrays

import cv2
from tensorflow import keras
import numpy as np

# Images
train_images = df_train.loc[:,'image_name']
train_labels = df_train.loc[:,'scratched_notscratched']

valid_images = df_valid.loc[:,'image_name']
valid_labels = df_valid.loc[:,'scratched_notscratched']

# Train images
x_train = []
for i in train_images:
    image = home_path + '/' + i
    img = cv2.imread(image, 0) # Grayscale
    x_train.append(img.squeeze())

# Train labels
y_train = keras.utils.to_categorical(train_labels)
```

```
# Validation images
x_valid = []
for i in valid_images:
    image = home_path + '/' +i
    img = cv2.imread(image, 0) # Grayscale
    x_valid.append(img.squeeze())

# Validation labels
y_valid = keras.utils.to_categorical(valid_labels)

# Normalize images
x_train = np.array(x_train, dtype = "float") / 255.0
x_valid = np.array(x_valid, dtype = "float") / 255.0

x_train = x_train.reshape(1955, 64, 64, 1)
x_valid = x_valid.reshape(489, 64, 64, 1)
```

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp13.history.keys())

# Summarize history for accuracy
plt.plot(history_exp13.history['accuracy'])
plt.plot(history_exp13.history['val_accuracy'])
plt.title('Model accuracy (Balanced 64x64px Grayscale WITHOUT Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp13.history['loss'])
plt.plot(history_exp13.history['val_loss'])
plt.title('Model loss (Balanced 64x64px Grayscale WITHOUT Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```

Experiment 14

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp14.history.keys())

# Summarize history for accuracy
plt.plot(history_exp14.history['accuracy'])
plt.plot(history_exp14.history['val_accuracy'])
plt.title('Model accuracy (Balanced 64x64px Grayscale WITH Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp14.history['loss'])
plt.plot(history_exp14.history['val_loss'])
plt.title('Model loss (Balanced 64x64px Grayscale WITH Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.ylim(0, 4)
plt.show()
```

Experiment 15

Next is the code to create the Training and Validation sub-datasets for this experiment:

```
# Creates the Training and Validation subsets of the dataset from CSV
# randomly, in folders
# No augmentation aplied yet
# 1s

!rm -rf datasets/scratches/defects/tiles64/grayscale/imbalancedall/train
!rm -rf datasets/scratches/defects/tiles64/grayscale/imbalancedall/valid

import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split

# Folder containing the tiles and CSV
home_path = r'datasets/scratches/defects/tiles64/grayscale/imbalancedall'

# Creates train and validation folders
train_path = os.path.join(home_path, 'train')
os.mkdir(train_path)
val_path = os.path.join(home_path, 'valid')
os.mkdir(val_path)
```

```
# Creates sub-folders for scratched and notscratched tiles in train and
# validation folders
scratched_train_path = os.path.join(home_path + r'/train', 'scratched')
os.mkdir(scratched_train_path)

notscratched_train_path = os.path.join(home_path + r'/train', 'notscratched')
os.mkdir(notscratched_train_path)

scratched_val_path = os.path.join(home_path + r'/valid', 'scratched')
os.mkdir(scratched_val_path)

notscratched_val_path = os.path.join(home_path + r'/valid', 'notscratched')
os.mkdir(notscratched_val_path)

# Original DataFrame with the data from the CSV
df = pd.read_csv('datasets/scratches/defects/tiles64/grayscale/scratched_notscratched.cs
v')

# Images and Categories
X = df.loc[:,'image_name']
y = df.loc[:,'scratched_notscratched']

# Splits data into Training and Validation
train_x, val_x, train_y, val_y = train_test_split(X, y,
                                                  test_size = 0.2,
                                                  random_state = 2021,
                                                  stratify = y)

# Train DataFrame
df_train = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_train['image_name'] = train_x
df_train['scratched_notscratched'] = train_y

# Validation DataFrame
df_valid = pd.DataFrame(columns = ['image_name', 'scratched_notscratched'])
df_valid['image_name'] = val_x
df_valid['scratched_notscratched'] = val_y


# Resets indexes
df_train.reset_index(drop = True, inplace = True)
df_valid.reset_index(drop = True, inplace = True)

# Copy train images to sub-folder
for i in range(len(df_train)):

    image = df_train.loc[i, 'image_name']

    if df_train.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_train_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_train_path)

# Copy validation images to sub-folder
for i in range(len(df_valid)):

    image = df_valid.loc[i,'image_name']

    if df_valid.loc[i, 'scratched_notscratched'] == 0:
        shutil.copy(home_path + r'/' + image, notscratched_val_path)
    else:
        shutil.copy(home_path + r'/' + image, scratched_val_path)

# Loads the tiles into the Train and Validation arrays
```

```
import cv2
from tensorflow import keras
import numpy as np

# Images
train_images = df_train.loc[:,'image_name']
train_labels = df_train.loc[:,'scratched_notscratched']

valid_images = df_valid.loc[:,'image_name']
valid_labels = df_valid.loc[:,'scratched_notscratched']

# Train images
x_train = []
for i in train_images:
    image = home_path + '/' + i
    img = cv2.imread(image, 0) # Grayscale
    x_train.append(img.squeeze())

# Train labels
y_train = keras.utils.to_categorical(train_labels)

# Validation images
x_valid = []
for i in valid_images:
    image = home_path + '/' +i
    img = cv2.imread(image, 0) # Grayscale
    x_valid.append(img.squeeze())

# Validation labels
y_valid = keras.utils.to_categorical(valid_labels)

# Normalize images
x_train = np.array(x_train, dtype = "float") / 255.0
x_valid = np.array(x_valid, dtype = "float") / 255.0

x_train = x_train.reshape(1955, 64, 64, 1)
x_valid = x_valid.reshape(489, 64, 64, 1)
```

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp15.history.keys())

# Summarize history for accuracy
plt.plot(history_exp15.history['accuracy'])
plt.plot(history_exp15.history['val_accuracy'])
plt.title('Model accuracy (Balanced 64x64px Grayscale WITHOUT Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```

```
# Summarize history for loss
plt.plot(history_exp15.history['loss'])
plt.plot(history_exp15.history['val_loss'])
plt.title('Model loss (Balanced 64x64px Grayscale WITHOUT Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```

Experiment 16

Next is the code to generate the charts that graphically represent the results of the model training for this experiment:

```
# Shows training results in a graphical way

import matplotlib.pyplot as plt

# List all data in history
print(history_exp16.history.keys())

# Summarize history for accuracy
plt.plot(history_exp16.history['accuracy'])
plt.plot(history_exp16.history['val_accuracy'])
plt.title('Model accuracy (Balanced 64x64px Grayscale WITH Augmentation)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

# Summarize history for loss
plt.plot(history_exp16.history['loss'])
plt.plot(history_exp16.history['val_loss'])
plt.title('Model loss (Balanced 64x64px Grayscale WITH Augmentation)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.ylim(0, 4)
plt.show()
```

Appendix 7. Full outputs of Experiments

Experiment 1

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 318, 318, 32)      896
 max_pooling2d (MaxPooling2D)  (None, 159, 159, 32)    0
 conv2d_1 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_1 (MaxPooling2D)  (None, 78, 78, 32)    0
 conv2d_2 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_2 (MaxPooling2D)  (None, 38, 38, 32)    0
 conv2d_3 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_3 (MaxPooling2D)  (None, 18, 18, 64)    0
 conv2d_4 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_4 (MaxPooling2D)  (None, 8, 8, 64)      0
 flatten (Flatten)           (None, 4096)              0
 dense (Dense)               (None, 64)                262208
 dropout (Dropout)           (None, 64)                0
 dense_1 (Dense)             (None, 2)                 130
=================================================================
Total params: 337,154
Trainable params: 337,154
Non-trainable params: 0
_________________________________________________________________
Epoch 1/35
6/6 [====] - 6s 452ms/step - loss: 0.7648 - accuracy: 0.4396 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 2/35
6/6 [====] - 1s 155ms/step - loss: 0.6945 - accuracy: 0.4396 - val_loss: 0.6930 - val_accuracy: 0.4783
Epoch 3/35
6/6 [====] - 1s 154ms/step - loss: 0.6931 - accuracy: 0.5220 - val_loss: 0.6927 - val_accuracy: 0.5000
Epoch 4/35
6/6 [====] - 1s 152ms/step - loss: 0.6938 - accuracy: 0.4835 - val_loss: 0.6928 - val_accuracy: 0.5000
Epoch 5/35
6/6 [====] - 1s 153ms/step - loss: 0.6933 - accuracy: 0.4670 - val_loss: 0.6923 - val_accuracy: 0.5000
Epoch 6/35
6/6 [====] - 1s 153ms/step - loss: 0.6917 - accuracy: 0.5385 - val_loss: 0.7503 - val_accuracy: 0.5000
Epoch 7/35
6/6 [====] - 1s 153ms/step - loss: 0.6989 - accuracy: 0.5055 - val_loss: 0.6923 - val_accuracy: 0.5435
Epoch 8/35
6/6 [====] - 1s 153ms/step - loss: 0.7577 - accuracy: 0.5934 - val_loss: 0.6981 - val_accuracy: 0.5000
Epoch 9/35
6/6 [====] - 1s 150ms/step - loss: 0.6927 - accuracy: 0.5604 - val_loss: 0.6900 - val_accuracy: 0.5000
Epoch 10/35
6/6 [====] - 1s 153ms/step - loss: 0.6928 - accuracy: 0.5549 - val_loss: 0.6927 - val_accuracy: 0.5000
Epoch 11/35
6/6 [====] - 1s 152ms/step - loss: 0.6923 - accuracy: 0.5824 - val_loss: 0.6873 - val_accuracy: 0.5217
Epoch 12/35
6/6 [====] - 1s 151ms/step - loss: 0.6912 - accuracy: 0.5549 - val_loss: 0.6860 - val_accuracy: 0.6087
Epoch 13/35
6/6 [====] - 1s 152ms/step - loss: 0.6923 - accuracy: 0.6154 - val_loss: 0.6909 - val_accuracy: 0.5652
Epoch 14/35
6/6 [====] - 1s 154ms/step - loss: 0.6716 - accuracy: 0.6044 - val_loss: 0.6775 - val_accuracy: 0.6087
Epoch 15/35
6/6 [====] - 1s 152ms/step - loss: 0.6667 - accuracy: 0.6319 - val_loss: 0.8898 - val_accuracy: 0.5000
Epoch 16/35
6/6 [====] - 1s 152ms/step - loss: 0.6934 - accuracy: 0.6319 - val_loss: 0.6749 - val_accuracy: 0.5652
Epoch 17/35
6/6 [====] - 1s 152ms/step - loss: 0.6559 - accuracy: 0.6099 - val_loss: 0.6821 - val_accuracy: 0.5217
Epoch 18/35
6/6 [====] - 1s 152ms/step - loss: 0.6370 - accuracy: 0.6264 - val_loss: 0.6439 - val_accuracy: 0.6522
Epoch 19/35
6/6 [====] - 1s 149ms/step - loss: 0.6006 - accuracy: 0.6758 - val_loss: 0.9697 - val_accuracy: 0.6087
Epoch 20/35
```

```
6/6 [====] - 1s 152ms/step - loss: 0.6894 - accuracy: 0.6429 - val_loss: 0.6409 - val_accuracy: 0.6522
Epoch 21/35
6/6 [====] - 1s 154ms/step - loss: 0.5631 - accuracy: 0.7253 - val_loss: 0.7500 - val_accuracy: 0.5217
Epoch 22/35
6/6 [====] - 1s 153ms/step - loss: 0.5577 - accuracy: 0.7363 - val_loss: 0.6535 - val_accuracy: 0.5435
Epoch 23/35
6/6 [====] - 1s 154ms/step - loss: 0.5053 - accuracy: 0.7582 - val_loss: 0.6982 - val_accuracy: 0.5870
Epoch 24/35
6/6 [====] - 1s 153ms/step - loss: 0.5298 - accuracy: 0.7637 - val_loss: 1.3285 - val_accuracy: 0.6304
Epoch 25/35
6/6 [====] - 1s 153ms/step - loss: 0.5650 - accuracy: 0.7363 - val_loss: 0.6780 - val_accuracy: 0.5652
Epoch 26/35
6/6 [====] - 1s 153ms/step - loss: 0.4750 - accuracy: 0.7857 - val_loss: 0.6801 - val_accuracy: 0.6087
Epoch 27/35
6/6 [====] - 1s 156ms/step - loss: 0.3974 - accuracy: 0.8242 - val_loss: 0.8326 - val_accuracy: 0.6087
Epoch 28/35
6/6 [====] - 1s 153ms/step - loss: 0.3262 - accuracy: 0.8791 - val_loss: 0.9425 - val_accuracy: 0.6522
Epoch 29/35
6/6 [====] - 1s 153ms/step - loss: 0.3405 - accuracy: 0.8516 - val_loss: 1.1796 - val_accuracy: 0.6087
Epoch 30/35
6/6 [====] - 1s 152ms/step - loss: 0.4061 - accuracy: 0.8297 - val_loss: 0.9064 - val_accuracy: 0.5435
Epoch 31/35
6/6 [====] - 1s 154ms/step - loss: 0.3580 - accuracy: 0.8736 - val_loss: 0.8026 - val_accuracy: 0.6957
Epoch 32/35
6/6 [====] - 1s 154ms/step - loss: 0.2472 - accuracy: 0.9066 - val_loss: 1.0455 - val_accuracy: 0.6522
Epoch 33/35
6/6 [====] - 1s 156ms/step - loss: 0.2454 - accuracy: 0.8901 - val_loss: 0.9941 - val_accuracy: 0.5870
Epoch 34/35
6/6 [====] - 1s 155ms/step - loss: 0.1672 - accuracy: 0.9560 - val_loss: 1.1947 - val_accuracy: 0.6304
Epoch 35/35
6/6 [====] - 1s 151ms/step - loss: 0.3417 - accuracy: 0.8516 - val_loss: 0.8244 - val_accuracy: 0.6087
```

Experiment 2

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 318, 318, 32)      896
 max_pooling2d (MaxPooling2D)  (None, 159, 159, 32)    0
 conv2d_1 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_1 (MaxPooling2D)  (None, 78, 78, 32)    0
 conv2d_2 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_2 (MaxPooling2D)  (None, 38, 38, 32)    0
 conv2d_3 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_3 (MaxPooling2D)  (None, 18, 18, 64)    0
 conv2d_4 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_4 (MaxPooling2D)  (None, 8, 8, 64)      0
 flatten (Flatten)           (None, 4096)              0
 dense (Dense)               (None, 64)                262208
 dropout (Dropout)           (None, 64)                0
 dense_1 (Dense)             (None, 2)                 130
=================================================================
Total params: 337,154
Trainable params: 337,154
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
6/6 [====] - 8s 1s/step - loss: 7.2763 - accuracy: 0.4615 - val_loss: 0.7636 - val_accuracy: 0.5000
Epoch 2/100
6/6 [====] - 6s 1s/step - loss: 0.7396 - accuracy: 0.5220 - val_loss: 0.6890 - val_accuracy: 0.6087
Epoch 3/100
6/6 [====] - 6s 1s/step - loss: 0.7058 - accuracy: 0.5110 - val_loss: 0.6907 - val_accuracy: 0.4783
Epoch 4/100
6/6 [====] - 6s 1s/step - loss: 0.6945 - accuracy: 0.5385 - val_loss: 0.6806 - val_accuracy: 0.6087
Epoch 5/100
6/6 [====] - 6s 1s/step - loss: 0.7153 - accuracy: 0.5495 - val_loss: 0.6988 - val_accuracy: 0.5435
Epoch 6/100
6/6 [====] - 6s 1s/step - loss: 0.7761 - accuracy: 0.5110 - val_loss: 0.6829 - val_accuracy: 0.5652
Epoch 7/100
6/6 [====] - 6s 1s/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6690 - val_accuracy: 0.5217
Epoch 8/100
6/6 [====] - 6s 1s/step - loss: 0.6792 - accuracy: 0.6264 - val_loss: 0.7014 - val_accuracy: 0.5435
Epoch 9/100
```

```
6/6 [====] - 6s 1s/step - loss: 0.6886 - accuracy: 0.5165 - val_loss: 0.6851 - val_accuracy: 0.5870
Epoch 10/100
6/6 [====] - 6s 1s/step - loss: 0.7467 - accuracy: 0.5330 - val_loss: 1.5992 - val_accuracy: 0.5000
Epoch 11/100
6/6 [====] - 6s 1s/step - loss: 0.8250 - accuracy: 0.5385 - val_loss: 0.7048 - val_accuracy: 0.4565
Epoch 12/100
6/6 [====] - 6s 1s/step - loss: 0.6988 - accuracy: 0.5110 - val_loss: 0.6968 - val_accuracy: 0.4783
Epoch 13/100
6/6 [====] - 6s 1s/step - loss: 0.6926 - accuracy: 0.5714 - val_loss: 0.6830 - val_accuracy: 0.5000
Epoch 14/100
6/6 [====] - 6s 1s/step - loss: 0.6876 - accuracy: 0.5440 - val_loss: 0.8824 - val_accuracy: 0.5000
Epoch 15/100
6/6 [====] - 6s 1s/step - loss: 0.8174 - accuracy: 0.4945 - val_loss: 0.7123 - val_accuracy: 0.4565
Epoch 16/100
6/6 [====] - 6s 1s/step - loss: 0.6908 - accuracy: 0.4835 - val_loss: 0.6955 - val_accuracy: 0.5217
Epoch 17/100
6/6 [====] - 6s 1s/step - loss: 0.7397 - accuracy: 0.5495 - val_loss: 0.6940 - val_accuracy: 0.5435
Epoch 18/100
6/6 [====] - 6s 1s/step - loss: 0.6910 - accuracy: 0.5330 - val_loss: 0.6802 - val_accuracy: 0.5000
Epoch 19/100
6/6 [====] - 6s 1s/step - loss: 0.6784 - accuracy: 0.6044 - val_loss: 0.6931 - val_accuracy: 0.5217
Epoch 20/100
6/6 [====] - 6s 1s/step - loss: 0.6919 - accuracy: 0.5110 - val_loss: 0.6953 - val_accuracy: 0.5652
Epoch 21/100
6/6 [====] - 6s 1s/step - loss: 0.6738 - accuracy: 0.5714 - val_loss: 0.7018 - val_accuracy: 0.5217
Epoch 22/100
6/6 [====] - 6s 1s/step - loss: 0.7015 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5652
Epoch 23/100
6/6 [====] - 6s 1s/step - loss: 0.6696 - accuracy: 0.5824 - val_loss: 0.6846 - val_accuracy: 0.5870
Epoch 24/100
6/6 [====] - 6s 1s/step - loss: 0.6556 - accuracy: 0.5989 - val_loss: 0.6896 - val_accuracy: 0.6087
Epoch 25/100
6/6 [====] - 6s 1s/step - loss: 0.7716 - accuracy: 0.6044 - val_loss: 0.7004 - val_accuracy: 0.5652
Epoch 26/100
6/6 [====] - 6s 1s/step - loss: 0.6700 - accuracy: 0.5769 - val_loss: 0.7152 - val_accuracy: 0.4348
Epoch 27/100
6/6 [====] - 6s 1s/step - loss: 0.6752 - accuracy: 0.6044 - val_loss: 0.6861 - val_accuracy: 0.5000
Epoch 28/100
6/6 [====] - 6s 1s/step - loss: 0.6600 - accuracy: 0.5989 - val_loss: 0.7382 - val_accuracy: 0.4565
Epoch 29/100
6/6 [====] - 6s 1s/step - loss: 0.6527 - accuracy: 0.5879 - val_loss: 0.6934 - val_accuracy: 0.4565
Epoch 30/100
6/6 [====] - 6s 1s/step - loss: 0.6390 - accuracy: 0.6319 - val_loss: 0.7552 - val_accuracy: 0.5217
Epoch 31/100
6/6 [====] - 6s 1s/step - loss: 0.7816 - accuracy: 0.5275 - val_loss: 0.7222 - val_accuracy: 0.5435
Epoch 32/100
6/6 [====] - 6s 1s/step - loss: 0.6588 - accuracy: 0.6099 - val_loss: 0.7887 - val_accuracy: 0.5217
Epoch 33/100
6/6 [====] - 6s 1s/step - loss: 0.6540 - accuracy: 0.5769 - val_loss: 0.7357 - val_accuracy: 0.6087
Epoch 34/100
6/6 [====] - 6s 1s/step - loss: 0.6634 - accuracy: 0.6538 - val_loss: 0.7422 - val_accuracy: 0.6087
Epoch 35/100
6/6 [====] - 6s 1s/step - loss: 0.6014 - accuracy: 0.6264 - val_loss: 0.6954 - val_accuracy: 0.6087
Epoch 36/100
6/6 [====] - 6s 1s/step - loss: 0.6923 - accuracy: 0.5934 - val_loss: 0.6953 - val_accuracy: 0.6304
Epoch 37/100
6/6 [====] - 6s 1s/step - loss: 0.6291 - accuracy: 0.6209 - val_loss: 0.6983 - val_accuracy: 0.6087
Epoch 38/100
6/6 [====] - 6s 1s/step - loss: 0.6407 - accuracy: 0.6099 - val_loss: 0.7300 - val_accuracy: 0.5652
Epoch 39/100
6/6 [====] - 6s 1s/step - loss: 0.6317 - accuracy: 0.6374 - val_loss: 0.6967 - val_accuracy: 0.5217
Epoch 40/100
6/6 [====] - 6s 1s/step - loss: 0.6279 - accuracy: 0.6593 - val_loss: 0.6875 - val_accuracy: 0.6087
Epoch 41/100
6/6 [====] - 6s 1s/step - loss: 0.6368 - accuracy: 0.6648 - val_loss: 0.7770 - val_accuracy: 0.5217
Epoch 42/100
6/6 [====] - 6s 1s/step - loss: 0.6234 - accuracy: 0.6648 - val_loss: 0.7051 - val_accuracy: 0.6087
Epoch 43/100
6/6 [====] - 6s 1s/step - loss: 0.5704 - accuracy: 0.6758 - val_loss: 1.6890 - val_accuracy: 0.5435
Epoch 44/100
6/6 [====] - 6s 1s/step - loss: 0.8757 - accuracy: 0.6319 - val_loss: 0.6486 - val_accuracy: 0.7174
Epoch 45/100
6/6 [====] - 6s 1s/step - loss: 0.6140 - accuracy: 0.6484 - val_loss: 0.5760 - val_accuracy: 0.6087
Epoch 46/100
6/6 [====] - 6s 1s/step - loss: 0.6504 - accuracy: 0.6374 - val_loss: 0.6492 - val_accuracy: 0.5652
Epoch 47/100
6/6 [====] - 6s 1s/step - loss: 0.5489 - accuracy: 0.7143 - val_loss: 0.8881 - val_accuracy: 0.6087
Epoch 48/100
6/6 [====] - 6s 1s/step - loss: 0.6537 - accuracy: 0.6758 - val_loss: 0.6574 - val_accuracy: 0.6087
Epoch 49/100
6/6 [====] - 6s 1s/step - loss: 0.5693 - accuracy: 0.7253 - val_loss: 0.7288 - val_accuracy: 0.5217
Epoch 50/100
```

```
6/6 [====] - 6s 1s/step - loss: 0.5741 - accuracy: 0.6923 - val_loss: 0.7300 - val_accuracy: 0.5652
Epoch 51/100
6/6 [====] - 6s 1s/step - loss: 0.6509 - accuracy: 0.6538 - val_loss: 0.6371 - val_accuracy: 0.6087
Epoch 52/100
6/6 [====] - 6s 1s/step - loss: 0.5945 - accuracy: 0.6374 - val_loss: 0.6411 - val_accuracy: 0.6522
Epoch 53/100
6/6 [====] - 6s 1s/step - loss: 0.5379 - accuracy: 0.6813 - val_loss: 0.8692 - val_accuracy: 0.5870
Epoch 54/100
6/6 [====] - 6s 1s/step - loss: 0.6179 - accuracy: 0.6374 - val_loss: 0.8341 - val_accuracy: 0.6739
Epoch 55/100
6/6 [====] - 6s 1s/step - loss: 0.6070 - accuracy: 0.6538 - val_loss: 0.7372 - val_accuracy: 0.5652
Epoch 56/100
6/6 [====] - 6s 1s/step - loss: 0.5603 - accuracy: 0.6758 - val_loss: 0.7760 - val_accuracy: 0.5435
Epoch 57/100
6/6 [====] - 6s 1s/step - loss: 0.5534 - accuracy: 0.6923 - val_loss: 0.8674 - val_accuracy: 0.5435
Epoch 58/100
6/6 [====] - 6s 1s/step - loss: 0.5060 - accuracy: 0.7692 - val_loss: 0.7469 - val_accuracy: 0.5652
Epoch 59/100
6/6 [====] - 6s 1s/step - loss: 0.5645 - accuracy: 0.6758 - val_loss: 0.8693 - val_accuracy: 0.5652
Epoch 60/100
6/6 [====] - 6s 1s/step - loss: 0.5649 - accuracy: 0.7198 - val_loss: 0.9531 - val_accuracy: 0.4565
Epoch 61/100
6/6 [====] - 6s 1s/step - loss: 0.5644 - accuracy: 0.6978 - val_loss: 0.7474 - val_accuracy: 0.5870
Epoch 62/100
6/6 [====] - 6s 1s/step - loss: 0.5404 - accuracy: 0.7363 - val_loss: 0.8272 - val_accuracy: 0.5652
Epoch 63/100
6/6 [====] - 6s 1s/step - loss: 0.5367 - accuracy: 0.7033 - val_loss: 0.8719 - val_accuracy: 0.6304
Epoch 64/100
6/6 [====] - 6s 1s/step - loss: 0.5131 - accuracy: 0.7418 - val_loss: 1.0886 - val_accuracy: 0.5435
Epoch 65/100
6/6 [====] - 6s 1s/step - loss: 0.4840 - accuracy: 0.7308 - val_loss: 1.0825 - val_accuracy: 0.5217
Epoch 66/100
6/6 [====] - 6s 1s/step - loss: 0.5998 - accuracy: 0.6868 - val_loss: 1.3885 - val_accuracy: 0.5217
Epoch 67/100
6/6 [====] - 6s 1s/step - loss: 0.5641 - accuracy: 0.7143 - val_loss: 1.0499 - val_accuracy: 0.6522
Epoch 68/100
6/6 [====] - 6s 1s/step - loss: 0.5129 - accuracy: 0.7473 - val_loss: 0.9628 - val_accuracy: 0.6304
Epoch 69/100
6/6 [====] - 6s 1s/step - loss: 0.4873 - accuracy: 0.7473 - val_loss: 0.9262 - val_accuracy: 0.5217
Epoch 70/100
6/6 [====] - 6s 1s/step - loss: 0.5484 - accuracy: 0.6923 - val_loss: 0.7734 - val_accuracy: 0.5652
Epoch 71/100
6/6 [====] - 6s 1s/step - loss: 0.5076 - accuracy: 0.7802 - val_loss: 0.9924 - val_accuracy: 0.6957
Epoch 72/100
6/6 [====] - 6s 1s/step - loss: 0.4903 - accuracy: 0.7363 - val_loss: 0.9577 - val_accuracy: 0.6739
Epoch 73/100
6/6 [====] - 6s 1s/step - loss: 0.4970 - accuracy: 0.7582 - val_loss: 0.8465 - val_accuracy: 0.6304
Epoch 74/100
6/6 [====] - 6s 1s/step - loss: 0.5066 - accuracy: 0.7418 - val_loss: 0.7473 - val_accuracy: 0.5435
Epoch 75/100
6/6 [====] - 6s 1s/step - loss: 0.5246 - accuracy: 0.7418 - val_loss: 0.9996 - val_accuracy: 0.5217
Epoch 76/100
6/6 [====] - 6s 1s/step - loss: 0.4539 - accuracy: 0.8077 - val_loss: 1.1016 - val_accuracy: 0.4783
Epoch 77/100
6/6 [====] - 6s 1s/step - loss: 0.4867 - accuracy: 0.7692 - val_loss: 1.1279 - val_accuracy: 0.5652
Epoch 78/100
6/6 [====] - 6s 1s/step - loss: 0.5651 - accuracy: 0.7088 - val_loss: 0.8382 - val_accuracy: 0.5870
Epoch 79/100
6/6 [====] - 6s 1s/step - loss: 0.4619 - accuracy: 0.7473 - val_loss: 0.8870 - val_accuracy: 0.5652
Epoch 80/100
6/6 [====] - 6s 1s/step - loss: 0.4501 - accuracy: 0.7857 - val_loss: 0.9344 - val_accuracy: 0.5435
Epoch 81/100
6/6 [====] - 6s 1s/step - loss: 0.4747 - accuracy: 0.7582 - val_loss: 1.5038 - val_accuracy: 0.5435
Epoch 82/100
6/6 [====] - 6s 1s/step - loss: 0.5105 - accuracy: 0.7747 - val_loss: 1.0824 - val_accuracy: 0.6087
Epoch 83/100
6/6 [====] - 6s 1s/step - loss: 0.5480 - accuracy: 0.7253 - val_loss: 0.7521 - val_accuracy: 0.6522
Epoch 84/100
6/6 [====] - 6s 1s/step - loss: 0.4527 - accuracy: 0.8022 - val_loss: 0.8311 - val_accuracy: 0.6304
Epoch 85/100
6/6 [====] - 6s 1s/step - loss: 0.4948 - accuracy: 0.7582 - val_loss: 0.7125 - val_accuracy: 0.6304
Epoch 86/100
6/6 [====] - 6s 1s/step - loss: 0.4420 - accuracy: 0.7802 - val_loss: 1.0692 - val_accuracy: 0.6087
Epoch 87/100
6/6 [====] - 6s 1s/step - loss: 0.4393 - accuracy: 0.7692 - val_loss: 1.0531 - val_accuracy: 0.6087
Epoch 88/100
6/6 [====] - 6s 1s/step - loss: 0.4429 - accuracy: 0.7912 - val_loss: 1.1580 - val_accuracy: 0.5652
Epoch 89/100
6/6 [====] - 6s 1s/step - loss: 0.4521 - accuracy: 0.7802 - val_loss: 1.4009 - val_accuracy: 0.6739
Epoch 90/100
6/6 [====] - 6s 1s/step - loss: 0.4159 - accuracy: 0.8352 - val_loss: 0.8698 - val_accuracy: 0.7174
Epoch 91/100
```

```
6/6 [====] - 6s 1s/step - loss: 0.4493 - accuracy: 0.8297 - val_loss: 0.8641 - val_accuracy: 0.6304
Epoch 92/100
6/6 [====] - 6s 1s/step - loss: 0.4111 - accuracy: 0.8132 - val_loss: 1.1529 - val_accuracy: 0.6087
Epoch 93/100
6/6 [====] - 6s 1s/step - loss: 0.4830 - accuracy: 0.7912 - val_loss: 1.5141 - val_accuracy: 0.4783
Epoch 94/100
6/6 [====] - 6s 1s/step - loss: 0.5473 - accuracy: 0.7308 - val_loss: 0.8234 - val_accuracy: 0.6087
Epoch 95/100
6/6 [====] - 6s 1s/step - loss: 0.3811 - accuracy: 0.8187 - val_loss: 1.1994 - val_accuracy: 0.5870
Epoch 96/100
6/6 [====] - 6s 1s/step - loss: 0.4003 - accuracy: 0.8297 - val_loss: 1.1504 - val_accuracy: 0.6739
Epoch 97/100
6/6 [====] - 6s 1s/step - loss: 0.4143 - accuracy: 0.7967 - val_loss: 1.6051 - val_accuracy: 0.4783
Epoch 98/100
6/6 [====] - 6s 1s/step - loss: 0.4304 - accuracy: 0.8077 - val_loss: 0.8624 - val_accuracy: 0.6522
Epoch 99/100
6/6 [====] - 6s 1s/step - loss: 0.3747 - accuracy: 0.8077 - val_loss: 0.9013 - val_accuracy: 0.5000
Epoch 100/100
6/6 [====] - 6s 1s/step - loss: 0.4284 - accuracy: 0.7967 - val_loss: 0.9641 - val_accuracy: 0.6739
```

Experiment 3

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)           (None, 318, 318, 32)      896
 max_pooling2d_10 (MaxPooling2D) (None, 159, 159, 32)      0
 conv2d_11 (Conv2D)           (None, 157, 157, 32)      9248
 max_pooling2d_11 (MaxPooling2D) (None, 78, 78, 32)        0
 conv2d_12 (Conv2D)           (None, 76, 76, 32)        9248
 max_pooling2d_12 (MaxPooling2D) (None, 38, 38, 32)        0
 conv2d_13 (Conv2D)           (None, 36, 36, 64)        18496
 max_pooling2d_13 (MaxPooling2D) (None, 18, 18, 64)        0
 conv2d_14 (Conv2D)           (None, 16, 16, 64)        36928
 max_pooling2d_14 (MaxPooling2D) (None, 8, 8, 64)          0
 flatten_2 (Flatten)          (None, 4096)              0
 dense_4 (Dense)              (None, 64)                262208
 dropout_2 (Dropout)          (None, 64)                0
 dense_5 (Dense)              (None, 2)                 130
=================================================================
Total params: 337,154
Trainable params: 337,154
Non-trainable params: 0
_________________________________________________________________
Epoch 1/35
6/6 [====] - 2s 220ms/step - loss: 0.7376 - accuracy: 0.5330 - val_loss: 0.6933 - val_accuracy: 0.5000
Epoch 2/35
6/6 [====] - 1s 151ms/step - loss: 0.6948 - accuracy: 0.4341 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 3/35
6/6 [====] - 1s 155ms/step - loss: 0.6930 - accuracy: 0.5110 - val_loss: 0.6927 - val_accuracy: 0.5000
Epoch 4/35
6/6 [====] - 1s 156ms/step - loss: 0.6940 - accuracy: 0.5000 - val_loss: 0.6926 - val_accuracy: 0.5217
Epoch 5/35
6/6 [====] - 1s 151ms/step - loss: 0.6928 - accuracy: 0.5330 - val_loss: 0.6921 - val_accuracy: 0.5000
Epoch 6/35
6/6 [====] - 1s 152ms/step - loss: 0.6940 - accuracy: 0.4560 - val_loss: 0.6923 - val_accuracy: 0.5652
Epoch 7/35
6/6 [====] - 1s 153ms/step - loss: 0.6922 - accuracy: 0.5385 - val_loss: 0.6921 - val_accuracy: 0.5000
Epoch 8/35
6/6 [====] - 1s 152ms/step - loss: 0.6978 - accuracy: 0.5714 - val_loss: 0.6918 - val_accuracy: 0.5217
Epoch 9/35
6/6 [====] - 1s 150ms/step - loss: 0.6946 - accuracy: 0.5385 - val_loss: 0.6914 - val_accuracy: 0.5435
Epoch 10/35
6/6 [====] - 1s 153ms/step - loss: 0.6906 - accuracy: 0.5440 - val_loss: 0.6884 - val_accuracy: 0.5000
Epoch 11/35
6/6 [====] - 1s 153ms/step - loss: 0.6892 - accuracy: 0.5549 - val_loss: 0.6902 - val_accuracy: 0.5000
Epoch 12/35
6/6 [====] - 1s 151ms/step - loss: 0.6930 - accuracy: 0.5714 - val_loss: 0.7166 - val_accuracy: 0.5000
Epoch 13/35
6/6 [====] - 1s 166ms/step - loss: 0.6888 - accuracy: 0.5824 - val_loss: 0.6811 - val_accuracy: 0.5000
Epoch 14/35
6/6 [====] - 1s 153ms/step - loss: 0.6667 - accuracy: 0.6044 - val_loss: 0.7542 - val_accuracy: 0.5000
Epoch 15/35
```

```
6/6 [====] - 1s 153ms/step - loss: 0.7024 - accuracy: 0.5769 - val_loss: 0.6850 - val_accuracy: 0.5217
Epoch 16/35
6/6 [====] - 1s 154ms/step - loss: 0.6320 - accuracy: 0.6429 - val_loss: 0.8044 - val_accuracy: 0.5870
Epoch 17/35
6/6 [====] - 1s 154ms/step - loss: 0.7082 - accuracy: 0.6264 - val_loss: 0.6708 - val_accuracy: 0.5217
Epoch 18/35
6/6 [====] - 1s 152ms/step - loss: 0.6196 - accuracy: 0.6648 - val_loss: 0.6460 - val_accuracy: 0.6957
Epoch 19/35
6/6 [====] - 1s 155ms/step - loss: 0.5881 - accuracy: 0.7143 - val_loss: 0.6434 - val_accuracy: 0.6087
Epoch 20/35
6/6 [====] - 1s 154ms/step - loss: 0.5669 - accuracy: 0.7143 - val_loss: 0.6450 - val_accuracy: 0.5870
Epoch 21/35
6/6 [====] - 1s 151ms/step - loss: 0.6249 - accuracy: 0.6758 - val_loss: 0.6255 - val_accuracy: 0.6739
Epoch 22/35
6/6 [====] - 1s 154ms/step - loss: 0.5187 - accuracy: 0.7418 - val_loss: 0.6693 - val_accuracy: 0.6304
Epoch 23/35
6/6 [====] - 1s 154ms/step - loss: 0.5379 - accuracy: 0.7088 - val_loss: 0.6548 - val_accuracy: 0.5652
Epoch 24/35
6/6 [====] - 1s 154ms/step - loss: 0.5259 - accuracy: 0.7473 - val_loss: 0.6868 - val_accuracy: 0.5870
Epoch 25/35
6/6 [====] - 1s 153ms/step - loss: 0.4828 - accuracy: 0.7637 - val_loss: 0.7213 - val_accuracy: 0.6087
Epoch 26/35
6/6 [====] - 1s 158ms/step - loss: 0.3891 - accuracy: 0.8352 - val_loss: 0.6933 - val_accuracy: 0.6304
Epoch 27/35
6/6 [====] - 1s 155ms/step - loss: 0.4796 - accuracy: 0.8022 - val_loss: 0.6198 - val_accuracy: 0.5435
Epoch 28/35
6/6 [====] - 1s 154ms/step - loss: 0.3630 - accuracy: 0.8516 - val_loss: 0.7722 - val_accuracy: 0.6304
Epoch 29/35
6/6 [====] - 1s 156ms/step - loss: 0.3026 - accuracy: 0.8407 - val_loss: 0.7919 - val_accuracy: 0.6304
Epoch 30/35
6/6 [====] - 1s 158ms/step - loss: 0.3013 - accuracy: 0.8352 - val_loss: 0.7777 - val_accuracy: 0.6304
Epoch 31/35
6/6 [====] - 1s 153ms/step - loss: 0.2295 - accuracy: 0.9176 - val_loss: 0.9081 - val_accuracy: 0.5870
Epoch 32/35
6/6 [====] - 1s 154ms/step - loss: 0.4755 - accuracy: 0.8022 - val_loss: 0.8119 - val_accuracy: 0.5000
Epoch 33/35
6/6 [====] - 1s 155ms/step - loss: 0.2146 - accuracy: 0.9286 - val_loss: 1.0185 - val_accuracy: 0.5435
Epoch 34/35
6/6 [====] - 1s 153ms/step - loss: 0.1709 - accuracy: 0.9341 - val_loss: 1.0679 - val_accuracy: 0.5870
Epoch 35/35
6/6 [====] - 1s 152ms/step - loss: 0.1368 - accuracy: 0.9451 - val_loss: 1.3032 - val_accuracy: 0.5870
```

Experiment 4

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 318, 318, 32)      896
 max_pooling2d_15 (MaxPooling2D) (None, 159, 159, 32)      0
 conv2d_16 (Conv2D)          (None, 157, 157, 32)      9248
 max_pooling2d_16 (MaxPooling2D) (None, 78, 78, 32)        0
 conv2d_17 (Conv2D)          (None, 76, 76, 32)        9248
 max_pooling2d_17 (MaxPooling2D) (None, 38, 38, 32)        0
 conv2d_18 (Conv2D)          (None, 36, 36, 64)        18496
 max_pooling2d_18 (MaxPooling2D) (None, 18, 18, 64)        0
 conv2d_19 (Conv2D)          (None, 16, 16, 64)        36928
 max_pooling2d_19 (MaxPooling2D) (None, 8, 8, 64)          0
 flatten_3 (Flatten)         (None, 4096)              0
 dense_6 (Dense)             (None, 64)                262208
 dropout_3 (Dropout)         (None, 64)                0
 dense_7 (Dense)             (None, 2)                 130
=================================================================
Total params: 337,154
Trainable params: 337,154
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
6/6 [====] - 8s 1s/step - loss: 14.8047 - accuracy: 0.5495 - val_loss: 1.1373 - val_accuracy: 0.5000
Epoch 2/100
6/6 [====] - 6s 1s/step - loss: 0.8846 - accuracy: 0.4725 - val_loss: 0.7221 - val_accuracy: 0.4783
Epoch 3/100
6/6 [====] - 6s 1s/step - loss: 0.7120 - accuracy: 0.5165 - val_loss: 0.6822 - val_accuracy: 0.5435
Epoch 4/100
```

```
6/6 [====] - 6s 1s/step - loss: 0.6950 - accuracy: 0.5604 - val_loss: 0.7092 - val_accuracy: 0.4565
Epoch 5/100
6/6 [====] - 6s 1s/step - loss: 0.6965 - accuracy: 0.5330 - val_loss: 0.6947 - val_accuracy: 0.5652
Epoch 6/100
6/6 [====] - 6s 1s/step - loss: 0.7052 - accuracy: 0.5220 - val_loss: 0.7217 - val_accuracy: 0.4565
Epoch 7/100
6/6 [====] - 6s 1s/step - loss: 0.7032 - accuracy: 0.5110 - val_loss: 0.6812 - val_accuracy: 0.5652
Epoch 8/100
6/6 [====] - 6s 1s/step - loss: 0.7953 - accuracy: 0.5220 - val_loss: 0.8898 - val_accuracy: 0.5435
Epoch 9/100
6/6 [====] - 6s 1s/step - loss: 0.8241 - accuracy: 0.4890 - val_loss: 0.7644 - val_accuracy: 0.5000
Epoch 10/100
6/6 [====] - 6s 1s/step - loss: 0.6810 - accuracy: 0.5440 - val_loss: 0.6938 - val_accuracy: 0.5000
Epoch 11/100
6/6 [====] - 6s 1s/step - loss: 0.8220 - accuracy: 0.5000 - val_loss: 0.7050 - val_accuracy: 0.5435
Epoch 12/100
6/6 [====] - 6s 1s/step - loss: 0.7055 - accuracy: 0.5165 - val_loss: 0.6911 - val_accuracy: 0.5000
Epoch 13/100
6/6 [====] - 6s 1s/step - loss: 0.6852 - accuracy: 0.5385 - val_loss: 0.6954 - val_accuracy: 0.4565
Epoch 14/100
6/6 [====] - 6s 1s/step - loss: 0.6873 - accuracy: 0.5824 - val_loss: 0.7310 - val_accuracy: 0.5435
Epoch 15/100
6/6 [====] - 6s 1s/step - loss: 0.6849 - accuracy: 0.5824 - val_loss: 0.7260 - val_accuracy: 0.4783
Epoch 16/100
6/6 [====] - 6s 1s/step - loss: 0.6705 - accuracy: 0.5659 - val_loss: 1.1297 - val_accuracy: 0.4783
Epoch 17/100
6/6 [====] - 6s 1s/step - loss: 0.7477 - accuracy: 0.5385 - val_loss: 0.6933 - val_accuracy: 0.5435
Epoch 18/100
6/6 [====] - 6s 1s/step - loss: 0.6774 - accuracy: 0.5659 - val_loss: 0.7034 - val_accuracy: 0.4565
Epoch 19/100
6/6 [====] - 6s 1s/step - loss: 0.6883 - accuracy: 0.5385 - val_loss: 0.7100 - val_accuracy: 0.5217
Epoch 20/100
6/6 [====] - 6s 1s/step - loss: 0.7810 - accuracy: 0.5769 - val_loss: 0.6859 - val_accuracy: 0.5217
Epoch 21/100
6/6 [====] - 6s 1s/step - loss: 0.7576 - accuracy: 0.5549 - val_loss: 0.6815 - val_accuracy: 0.6087
Epoch 22/100
6/6 [====] - 6s 1s/step - loss: 0.6864 - accuracy: 0.5879 - val_loss: 0.6711 - val_accuracy: 0.5217
Epoch 23/100
6/6 [====] - 6s 1s/step - loss: 0.6749 - accuracy: 0.5385 - val_loss: 0.6667 - val_accuracy: 0.5652
Epoch 24/100
6/6 [====] - 6s 1s/step - loss: 0.6700 - accuracy: 0.5659 - val_loss: 0.6955 - val_accuracy: 0.5217
Epoch 25/100
6/6 [====] - 6s 1s/step - loss: 0.7270 - accuracy: 0.5220 - val_loss: 0.6763 - val_accuracy: 0.5435
Epoch 26/100
6/6 [====] - 6s 1s/step - loss: 0.6704 - accuracy: 0.5549 - val_loss: 0.6607 - val_accuracy: 0.5870
Epoch 27/100
6/6 [====] - 6s 1s/step - loss: 0.6593 - accuracy: 0.5824 - val_loss: 0.6821 - val_accuracy: 0.5000
Epoch 28/100
6/6 [====] - 6s 1s/step - loss: 0.6793 - accuracy: 0.5824 - val_loss: 0.6586 - val_accuracy: 0.5652
Epoch 29/100
6/6 [====] - 6s 1s/step - loss: 0.6314 - accuracy: 0.6264 - val_loss: 0.7563 - val_accuracy: 0.6522
Epoch 30/100
6/6 [====] - 6s 1s/step - loss: 0.6872 - accuracy: 0.6154 - val_loss: 0.7025 - val_accuracy: 0.5217
Epoch 31/100
6/6 [====] - 6s 1s/step - loss: 0.6593 - accuracy: 0.5714 - val_loss: 0.7227 - val_accuracy: 0.4783
Epoch 32/100
6/6 [====] - 6s 1s/step - loss: 0.6820 - accuracy: 0.5934 - val_loss: 0.6775 - val_accuracy: 0.5435
Epoch 33/100
6/6 [====] - 6s 1s/step - loss: 0.6516 - accuracy: 0.5879 - val_loss: 0.6844 - val_accuracy: 0.5217
Epoch 34/100
6/6 [====] - 6s 1s/step - loss: 0.7179 - accuracy: 0.5879 - val_loss: 0.6914 - val_accuracy: 0.5870
Epoch 35/100
6/6 [====] - 6s 1s/step - loss: 0.6330 - accuracy: 0.6044 - val_loss: 0.6517 - val_accuracy: 0.6739
Epoch 36/100
6/6 [====] - 6s 1s/step - loss: 0.6950 - accuracy: 0.6099 - val_loss: 0.7105 - val_accuracy: 0.5435
Epoch 37/100
6/6 [====] - 6s 1s/step - loss: 0.6208 - accuracy: 0.6429 - val_loss: 0.7468 - val_accuracy: 0.5217
Epoch 38/100
6/6 [====] - 6s 1s/step - loss: 0.6486 - accuracy: 0.6264 - val_loss: 0.9465 - val_accuracy: 0.5435
Epoch 39/100
6/6 [====] - 6s 1s/step - loss: 0.6508 - accuracy: 0.6154 - val_loss: 0.7402 - val_accuracy: 0.6522
Epoch 40/100
6/6 [====] - 6s 1s/step - loss: 0.6728 - accuracy: 0.6593 - val_loss: 0.6990 - val_accuracy: 0.4783
Epoch 41/100
6/6 [====] - 6s 1s/step - loss: 0.6074 - accuracy: 0.6538 - val_loss: 0.6931 - val_accuracy: 0.5217
Epoch 42/100
6/6 [====] - 6s 1s/step - loss: 0.6536 - accuracy: 0.6209 - val_loss: 0.8686 - val_accuracy: 0.4565
Epoch 43/100
6/6 [====] - 6s 1s/step - loss: 0.6724 - accuracy: 0.5659 - val_loss: 0.6751 - val_accuracy: 0.6304
Epoch 44/100
6/6 [====] - 6s 1s/step - loss: 0.6370 - accuracy: 0.5549 - val_loss: 0.7590 - val_accuracy: 0.5652
Epoch 45/100
```

```
6/6 [====] - 6s 1s/step - loss: 0.6104 - accuracy: 0.6429 - val_loss: 0.7566 - val_accuracy: 0.5217
Epoch 46/100
6/6 [====] - 6s 1s/step - loss: 0.6555 - accuracy: 0.6209 - val_loss: 0.6857 - val_accuracy: 0.5870
Epoch 47/100
6/6 [====] - 6s 1s/step - loss: 0.6074 - accuracy: 0.6099 - val_loss: 0.7181 - val_accuracy: 0.5870
Epoch 48/100
6/6 [====] - 6s 1s/step - loss: 0.6139 - accuracy: 0.5934 - val_loss: 0.6942 - val_accuracy: 0.6522
Epoch 49/100
6/6 [====] - 6s 1s/step - loss: 0.6025 - accuracy: 0.6538 - val_loss: 0.7663 - val_accuracy: 0.5435
Epoch 50/100
6/6 [====] - 6s 1s/step - loss: 0.6058 - accuracy: 0.6374 - val_loss: 0.7597 - val_accuracy: 0.5652
Epoch 51/100
6/6 [====] - 6s 1s/step - loss: 0.7931 - accuracy: 0.5659 - val_loss: 0.7202 - val_accuracy: 0.5217
Epoch 52/100
6/6 [====] - 6s 1s/step - loss: 0.5801 - accuracy: 0.6429 - val_loss: 0.7686 - val_accuracy: 0.5870
Epoch 53/100
6/6 [====] - 6s 1s/step - loss: 0.5859 - accuracy: 0.6648 - val_loss: 0.7280 - val_accuracy: 0.6304
Epoch 54/100
6/6 [====] - 6s 1s/step - loss: 0.6664 - accuracy: 0.5879 - val_loss: 0.7603 - val_accuracy: 0.6957
Epoch 55/100
6/6 [====] - 6s 1s/step - loss: 0.5627 - accuracy: 0.6648 - val_loss: 0.8452 - val_accuracy: 0.5652
Epoch 56/100
6/6 [====] - 6s 1s/step - loss: 0.6215 - accuracy: 0.6484 - val_loss: 0.6180 - val_accuracy: 0.6739
Epoch 57/100
6/6 [====] - 6s 1s/step - loss: 0.6315 - accuracy: 0.6648 - val_loss: 0.6379 - val_accuracy: 0.7174
Epoch 58/100
6/6 [====] - 6s 1s/step - loss: 0.5681 - accuracy: 0.6868 - val_loss: 3.1064 - val_accuracy: 0.5000
Epoch 59/100
6/6 [====] - 6s 1s/step - loss: 1.2411 - accuracy: 0.5659 - val_loss: 0.7932 - val_accuracy: 0.5870
Epoch 60/100
6/6 [====] - 6s 1s/step - loss: 0.6564 - accuracy: 0.6703 - val_loss: 0.7878 - val_accuracy: 0.5652
Epoch 61/100
6/6 [====] - 6s 1s/step - loss: 0.5504 - accuracy: 0.7088 - val_loss: 0.6451 - val_accuracy: 0.6522
Epoch 62/100
6/6 [====] - 6s 1s/step - loss: 0.6041 - accuracy: 0.6484 - val_loss: 0.8662 - val_accuracy: 0.6087
Epoch 63/100
6/6 [====] - 6s 1s/step - loss: 0.5665 - accuracy: 0.6978 - val_loss: 0.9791 - val_accuracy: 0.6087
Epoch 64/100
6/6 [====] - 6s 1s/step - loss: 0.5737 - accuracy: 0.6374 - val_loss: 0.8073 - val_accuracy: 0.5870
Epoch 65/100
6/6 [====] - 6s 1s/step - loss: 0.6158 - accuracy: 0.6429 - val_loss: 0.7272 - val_accuracy: 0.5217
Epoch 66/100
6/6 [====] - 6s 1s/step - loss: 0.5592 - accuracy: 0.6923 - val_loss: 0.8768 - val_accuracy: 0.6087
Epoch 67/100
6/6 [====] - 6s 1s/step - loss: 0.7991 - accuracy: 0.6209 - val_loss: 0.8574 - val_accuracy: 0.5435
Epoch 68/100
6/6 [====] - 6s 1s/step - loss: 0.5935 - accuracy: 0.6593 - val_loss: 0.7227 - val_accuracy: 0.5000
Epoch 69/100
6/6 [====] - 6s 1s/step - loss: 0.5158 - accuracy: 0.6758 - val_loss: 0.8457 - val_accuracy: 0.5652
Epoch 70/100
6/6 [====] - 6s 1s/step - loss: 0.5343 - accuracy: 0.7088 - val_loss: 0.7238 - val_accuracy: 0.5870
Epoch 71/100
6/6 [====] - 6s 1s/step - loss: 0.5546 - accuracy: 0.7033 - val_loss: 0.7970 - val_accuracy: 0.5435
Epoch 72/100
6/6 [====] - 6s 1s/step - loss: 0.5973 - accuracy: 0.6703 - val_loss: 0.7141 - val_accuracy: 0.6522
Epoch 73/100
6/6 [====] - 6s 1s/step - loss: 0.5199 - accuracy: 0.7088 - val_loss: 1.0159 - val_accuracy: 0.6087
Epoch 74/100
6/6 [====] - 6s 1s/step - loss: 0.6363 - accuracy: 0.7033 - val_loss: 0.7190 - val_accuracy: 0.5870
Epoch 75/100
6/6 [====] - 6s 1s/step - loss: 0.5216 - accuracy: 0.6703 - val_loss: 1.0206 - val_accuracy: 0.5870
Epoch 76/100
6/6 [====] - 6s 1s/step - loss: 0.5522 - accuracy: 0.6758 - val_loss: 0.8448 - val_accuracy: 0.5000
Epoch 77/100
6/6 [====] - 6s 1s/step - loss: 0.5346 - accuracy: 0.6374 - val_loss: 0.8777 - val_accuracy: 0.4565
Epoch 78/100
6/6 [====] - 6s 1s/step - loss: 0.5447 - accuracy: 0.6538 - val_loss: 1.0212 - val_accuracy: 0.5217
Epoch 79/100
6/6 [====] - 6s 1s/step - loss: 0.5299 - accuracy: 0.6648 - val_loss: 0.7494 - val_accuracy: 0.6087
Epoch 80/100
6/6 [====] - 6s 1s/step - loss: 0.4966 - accuracy: 0.7527 - val_loss: 1.0558 - val_accuracy: 0.5652
Epoch 81/100
6/6 [====] - 6s 1s/step - loss: 0.6284 - accuracy: 0.6538 - val_loss: 0.8930 - val_accuracy: 0.5000
Epoch 82/100
6/6 [====] - 6s 1s/step - loss: 0.5795 - accuracy: 0.6923 - val_loss: 0.9677 - val_accuracy: 0.5000
Epoch 83/100
6/6 [====] - 6s 1s/step - loss: 0.5361 - accuracy: 0.7253 - val_loss: 0.9629 - val_accuracy: 0.6522
Epoch 84/100
6/6 [====] - 6s 1s/step - loss: 0.5377 - accuracy: 0.6813 - val_loss: 0.7953 - val_accuracy: 0.5870
Epoch 85/100
6/6 [====] - 6s 1s/step - loss: 0.5312 - accuracy: 0.6813 - val_loss: 0.8079 - val_accuracy: 0.5217
Epoch 86/100
```

```
6/6 [====] - 6s 1s/step - loss: 0.9020 - accuracy: 0.6484 - val_loss: 0.8927 - val_accuracy: 0.5217
Epoch 87/100
6/6 [====] - 6s 1s/step - loss: 0.5744 - accuracy: 0.6978 - val_loss: 0.8677 - val_accuracy: 0.5652
Epoch 88/100
6/6 [====] - 6s 1s/step - loss: 0.5692 - accuracy: 0.6648 - val_loss: 0.9281 - val_accuracy: 0.6304
Epoch 89/100
6/6 [====] - 6s 1s/step - loss: 0.5253 - accuracy: 0.6813 - val_loss: 1.0473 - val_accuracy: 0.4783
Epoch 90/100
6/6 [====] - 6s 1s/step - loss: 0.4847 - accuracy: 0.7363 - val_loss: 0.8815 - val_accuracy: 0.6522
Epoch 91/100
6/6 [====] - 6s 1s/step - loss: 0.4877 - accuracy: 0.7143 - val_loss: 0.9406 - val_accuracy: 0.5217
Epoch 92/100
6/6 [====] - 6s 1s/step - loss: 0.5273 - accuracy: 0.7088 - val_loss: 0.8369 - val_accuracy: 0.5217
Epoch 93/100
6/6 [====] - 6s 1s/step - loss: 0.4450 - accuracy: 0.7033 - val_loss: 0.9344 - val_accuracy: 0.5870
Epoch 94/100
6/6 [====] - 6s 1s/step - loss: 0.5940 - accuracy: 0.7198 - val_loss: 0.8037 - val_accuracy: 0.5000
Epoch 95/100
6/6 [====] - 6s 1s/step - loss: 0.4763 - accuracy: 0.7418 - val_loss: 1.1492 - val_accuracy: 0.5870
Epoch 96/100
6/6 [====] - 6s 1s/step - loss: 0.5991 - accuracy: 0.7308 - val_loss: 0.6966 - val_accuracy: 0.6304
Epoch 97/100
6/6 [====] - 6s 1s/step - loss: 0.5002 - accuracy: 0.7253 - val_loss: 1.1218 - val_accuracy: 0.6739
Epoch 98/100
6/6 [====] - 6s 1s/step - loss: 0.4959 - accuracy: 0.7747 - val_loss: 1.1329 - val_accuracy: 0.5870
Epoch 99/100
6/6 [====] - 6s 1s/step - loss: 0.5221 - accuracy: 0.7418 - val_loss: 0.7961 - val_accuracy: 0.6522
Epoch 100/100
6/6 [====] - 6s 1s/step - loss: 0.4852 - accuracy: 0.7527 - val_loss: 0.9091 - val_accuracy: 0.5217
```

Experiment 5

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 318, 318, 32)      320
 max_pooling2d_20 (MaxPooling2D) (None, 159, 159, 32)      0
 conv2d_21 (Conv2D)          (None, 157, 157, 32)      9248
 max_pooling2d_21 (MaxPooling2D) (None, 78, 78, 32)        0
 conv2d_22 (Conv2D)          (None, 76, 76, 32)        9248
 max_pooling2d_22 (MaxPooling2D) (None, 38, 38, 32)        0
 conv2d_23 (Conv2D)          (None, 36, 36, 64)        18496
 max_pooling2d_23 (MaxPooling2D) (None, 18, 18, 64)        0
 conv2d_24 (Conv2D)          (None, 16, 16, 64)        36928
 max_pooling2d_24 (MaxPooling2D) (None, 8, 8, 64)          0
 flatten_4 (Flatten)         (None, 4096)              0
 dense_8 (Dense)             (None, 64)                262208
 dropout_4 (Dropout)         (None, 64)                0
 dense_9 (Dense)             (None, 2)                 130
=================================================================
Total params: 336,578
Trainable params: 336,578
Non-trainable params: 0
_________________________________________________________________
Epoch 1/35
6/6 [====] - 3s 237ms/step - loss: 0.7076 - accuracy: 0.4670 - val_loss: 0.6931 - val_accuracy: 0.5217
Epoch 2/35
6/6 [====] - 1s 153ms/step - loss: 0.6957 - accuracy: 0.4780 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3/35
6/6 [====] - 1s 153ms/step - loss: 0.6933 - accuracy: 0.5000 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 4/35
6/6 [====] - 1s 149ms/step - loss: 0.6979 - accuracy: 0.4670 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5/35
6/6 [====] - 1s 150ms/step - loss: 0.6934 - accuracy: 0.5055 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6/35
6/6 [====] - 1s 151ms/step - loss: 0.6930 - accuracy: 0.5330 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 7/35
6/6 [====] - 1s 152ms/step - loss: 0.7124 - accuracy: 0.5220 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 8/35
6/6 [====] - 1s 152ms/step - loss: 0.6935 - accuracy: 0.4835 - val_loss: 0.6928 - val_accuracy: 0.5000
Epoch 9/35
6/6 [====] - 1s 151ms/step - loss: 0.6925 - accuracy: 0.4835 - val_loss: 0.6927 - val_accuracy: 0.5000
Epoch 10/35
```

```
6/6 [====] - 1s 151ms/step - loss: 0.6929 - accuracy: 0.5110 - val_loss: 0.6911 - val_accuracy: 0.5000
Epoch 11/35
6/6 [====] - 1s 155ms/step - loss: 0.7338 - accuracy: 0.5110 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 12/35
6/6 [====] - 1s 149ms/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6927 - val_accuracy: 0.5000
Epoch 13/35
6/6 [====] - 1s 151ms/step - loss: 0.6941 - accuracy: 0.5385 - val_loss: 0.6907 - val_accuracy: 0.5217
Epoch 14/35
6/6 [====] - 1s 153ms/step - loss: 0.6853 - accuracy: 0.5440 - val_loss: 0.6878 - val_accuracy: 0.5435
Epoch 15/35
6/6 [====] - 1s 151ms/step - loss: 0.9101 - accuracy: 0.5440 - val_loss: 0.6919 - val_accuracy: 0.4565
Epoch 16/35
6/6 [====] - 1s 152ms/step - loss: 0.6848 - accuracy: 0.6099 - val_loss: 0.6894 - val_accuracy: 0.4783
Epoch 17/35
6/6 [====] - 1s 153ms/step - loss: 0.6774 - accuracy: 0.5989 - val_loss: 0.7013 - val_accuracy: 0.5000
Epoch 18/35
6/6 [====] - 1s 152ms/step - loss: 0.6729 - accuracy: 0.6099 - val_loss: 0.6883 - val_accuracy: 0.5652
Epoch 19/35
6/6 [====] - 1s 151ms/step - loss: 0.6551 - accuracy: 0.6099 - val_loss: 0.6760 - val_accuracy: 0.5870
Epoch 20/35
6/6 [====] - 1s 150ms/step - loss: 0.6317 - accuracy: 0.6374 - val_loss: 0.6902 - val_accuracy: 0.5435
Epoch 21/35
6/6 [====] - 1s 151ms/step - loss: 0.6276 - accuracy: 0.6044 - val_loss: 0.6750 - val_accuracy: 0.5217
Epoch 22/35
6/6 [====] - 1s 153ms/step - loss: 0.6095 - accuracy: 0.6593 - val_loss: 0.6831 - val_accuracy: 0.5435
Epoch 23/35
6/6 [====] - 1s 152ms/step - loss: 0.5976 - accuracy: 0.6758 - val_loss: 0.6653 - val_accuracy: 0.6087
Epoch 24/35
6/6 [====] - 1s 151ms/step - loss: 0.5363 - accuracy: 0.7143 - val_loss: 0.7125 - val_accuracy: 0.5435
Epoch 25/35
6/6 [====] - 1s 155ms/step - loss: 0.5873 - accuracy: 0.7473 - val_loss: 0.7352 - val_accuracy: 0.5217
Epoch 26/35
6/6 [====] - 1s 152ms/step - loss: 0.4936 - accuracy: 0.7527 - val_loss: 0.8359 - val_accuracy: 0.6087
Epoch 27/35
6/6 [====] - 1s 151ms/step - loss: 0.4563 - accuracy: 0.7747 - val_loss: 0.8187 - val_accuracy: 0.5652
Epoch 28/35
6/6 [====] - 1s 152ms/step - loss: 0.4027 - accuracy: 0.7912 - val_loss: 0.8938 - val_accuracy: 0.5652
Epoch 29/35
6/6 [====] - 1s 149ms/step - loss: 0.3792 - accuracy: 0.8132 - val_loss: 0.7352 - val_accuracy: 0.6739
Epoch 30/35
6/6 [====] - 1s 152ms/step - loss: 0.4275 - accuracy: 0.8352 - val_loss: 0.7580 - val_accuracy: 0.5652
Epoch 31/35
6/6 [====] - 1s 152ms/step - loss: 0.3774 - accuracy: 0.8077 - val_loss: 0.8928 - val_accuracy: 0.5652
Epoch 32/35
6/6 [====] - 1s 150ms/step - loss: 0.2882 - accuracy: 0.8352 - val_loss: 0.8724 - val_accuracy: 0.5435
Epoch 33/35
6/6 [====] - 1s 151ms/step - loss: 0.2654 - accuracy: 0.8846 - val_loss: 1.1875 - val_accuracy: 0.5652
Epoch 34/35
6/6 [====] - 1s 150ms/step - loss: 0.2798 - accuracy: 0.8516 - val_loss: 0.9643 - val_accuracy: 0.5652
Epoch 35/35
6/6 [====] - 1s 151ms/step - loss: 0.2856 - accuracy: 0.8352 - val_loss: 1.1971 - val_accuracy: 0.5870
```

Experiment 6

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 318, 318, 32)      320
 max_pooling2d_25 (MaxPooling2D) (None, 159, 159, 32)      0
 conv2d_26 (Conv2D)          (None, 157, 157, 32)      9248
 max_pooling2d_26 (MaxPooling2D) (None, 78, 78, 32)        0
 conv2d_27 (Conv2D)          (None, 76, 76, 32)        9248
 max_pooling2d_27 (MaxPooling2D) (None, 38, 38, 32)        0
 conv2d_28 (Conv2D)          (None, 36, 36, 64)        18496
 max_pooling2d_28 (MaxPooling2D) (None, 18, 18, 64)        0
 conv2d_29 (Conv2D)          (None, 16, 16, 64)        36928
 max_pooling2d_29 (MaxPooling2D) (None, 8, 8, 64)          0
 flatten_5 (Flatten)         (None, 4096)              0
 dense_10 (Dense)            (None, 64)                262208
 dropout_5 (Dropout)         (None, 64)                0
 dense_11 (Dense)            (None, 2)                 130
=================================================================
Total params: 336,578
```

```
Trainable params: 336,578
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
6/6 [====] - 8s 1s/step - loss: 7.2763 - accuracy: 0.4615 - val_loss: 0.7636 - val_accuracy: 0.5000
Epoch 2/100
6/6 [====] - 6s 1s/step - loss: 0.7396 - accuracy: 0.5220 - val_loss: 0.6890 - val_accuracy: 0.6087
Epoch 3/100
6/6 [====] - 6s 1s/step - loss: 0.7058 - accuracy: 0.5110 - val_loss: 0.6907 - val_accuracy: 0.4783
Epoch 4/100
6/6 [====] - 6s 1s/step - loss: 0.6945 - accuracy: 0.5385 - val_loss: 0.6806 - val_accuracy: 0.6087
Epoch 5/100
6/6 [====] - 6s 1s/step - loss: 0.7153 - accuracy: 0.5495 - val_loss: 0.6988 - val_accuracy: 0.5435
Epoch 6/100
6/6 [====] - 6s 1s/step - loss: 0.7761 - accuracy: 0.5110 - val_loss: 0.6829 - val_accuracy: 0.5652
Epoch 7/100
6/6 [====] - 6s 1s/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6690 - val_accuracy: 0.5217
Epoch 8/100
6/6 [====] - 6s 1s/step - loss: 0.6792 - accuracy: 0.6264 - val_loss: 0.7014 - val_accuracy: 0.5435
Epoch 9/100
6/6 [====] - 6s 1s/step - loss: 0.6886 - accuracy: 0.5165 - val_loss: 0.6851 - val_accuracy: 0.5870
Epoch 10/100
6/6 [====] - 6s 1s/step - loss: 0.7467 - accuracy: 0.5330 - val_loss: 1.5992 - val_accuracy: 0.5000
Epoch 11/100
6/6 [====] - 6s 1s/step - loss: 0.8250 - accuracy: 0.5385 - val_loss: 0.7048 - val_accuracy: 0.4565
Epoch 12/100
6/6 [====] - 6s 1s/step - loss: 0.6988 - accuracy: 0.5110 - val_loss: 0.6968 - val_accuracy: 0.4783
Epoch 13/100
6/6 [====] - 6s 1s/step - loss: 0.6926 - accuracy: 0.5714 - val_loss: 0.6830 - val_accuracy: 0.5000
Epoch 14/100
6/6 [====] - 6s 1s/step - loss: 0.6876 - accuracy: 0.5440 - val_loss: 0.8824 - val_accuracy: 0.5000
Epoch 15/100
6/6 [====] - 6s 1s/step - loss: 0.8174 - accuracy: 0.4945 - val_loss: 0.7123 - val_accuracy: 0.4565
Epoch 16/100
6/6 [====] - 6s 1s/step - loss: 0.6908 - accuracy: 0.4835 - val_loss: 0.6955 - val_accuracy: 0.5217
Epoch 17/100
6/6 [====] - 6s 1s/step - loss: 0.7397 - accuracy: 0.5495 - val_loss: 0.6940 - val_accuracy: 0.5435
Epoch 18/100
6/6 [====] - 6s 1s/step - loss: 0.6910 - accuracy: 0.5330 - val_loss: 0.6802 - val_accuracy: 0.5000
Epoch 19/100
6/6 [====] - 6s 1s/step - loss: 0.6784 - accuracy: 0.6044 - val_loss: 0.6931 - val_accuracy: 0.5217
Epoch 20/100
6/6 [====] - 6s 1s/step - loss: 0.6919 - accuracy: 0.5110 - val_loss: 0.6953 - val_accuracy: 0.5652
Epoch 21/100
6/6 [====] - 6s 1s/step - loss: 0.6738 - accuracy: 0.5714 - val_loss: 0.7018 - val_accuracy: 0.5217
Epoch 22/100
6/6 [====] - 6s 1s/step - loss: 0.7015 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5652
Epoch 23/100
6/6 [====] - 6s 1s/step - loss: 0.6696 - accuracy: 0.5824 - val_loss: 0.6846 - val_accuracy: 0.5870
Epoch 24/100
6/6 [====] - 6s 1s/step - loss: 0.6556 - accuracy: 0.5989 - val_loss: 0.6896 - val_accuracy: 0.6087
Epoch 25/100
6/6 [====] - 6s 1s/step - loss: 0.7716 - accuracy: 0.6044 - val_loss: 0.7004 - val_accuracy: 0.5652
Epoch 26/100
6/6 [====] - 6s 1s/step - loss: 0.6700 - accuracy: 0.5769 - val_loss: 0.7152 - val_accuracy: 0.4348
Epoch 27/100
6/6 [====] - 6s 1s/step - loss: 0.6752 - accuracy: 0.6044 - val_loss: 0.6861 - val_accuracy: 0.5000
Epoch 28/100
6/6 [====] - 6s 1s/step - loss: 0.6600 - accuracy: 0.5989 - val_loss: 0.7382 - val_accuracy: 0.4565
Epoch 29/100
6/6 [====] - 6s 1s/step - loss: 0.6527 - accuracy: 0.5879 - val_loss: 0.6934 - val_accuracy: 0.4565
Epoch 30/100
6/6 [====] - 6s 1s/step - loss: 0.6390 - accuracy: 0.6319 - val_loss: 0.7552 - val_accuracy: 0.5217
Epoch 31/100
6/6 [====] - 6s 1s/step - loss: 0.7816 - accuracy: 0.5275 - val_loss: 0.7222 - val_accuracy: 0.5435
Epoch 32/100
6/6 [====] - 6s 1s/step - loss: 0.6588 - accuracy: 0.6099 - val_loss: 0.7887 - val_accuracy: 0.5217
Epoch 33/100
6/6 [====] - 6s 1s/step - loss: 0.6540 - accuracy: 0.5769 - val_loss: 0.7357 - val_accuracy: 0.6087
Epoch 34/100
6/6 [====] - 6s 1s/step - loss: 0.6634 - accuracy: 0.6538 - val_loss: 0.7422 - val_accuracy: 0.6087
Epoch 35/100
6/6 [====] - 6s 1s/step - loss: 0.6014 - accuracy: 0.6264 - val_loss: 0.6954 - val_accuracy: 0.6087
Epoch 36/100
6/6 [====] - 6s 1s/step - loss: 0.6923 - accuracy: 0.5934 - val_loss: 0.6953 - val_accuracy: 0.6304
Epoch 37/100
6/6 [====] - 6s 1s/step - loss: 0.6291 - accuracy: 0.6209 - val_loss: 0.6983 - val_accuracy: 0.6087
Epoch 38/100
6/6 [====] - 6s 1s/step - loss: 0.6407 - accuracy: 0.6099 - val_loss: 0.7300 - val_accuracy: 0.5652
Epoch 39/100
6/6 [====] - 6s 1s/step - loss: 0.6317 - accuracy: 0.6374 - val_loss: 0.6967 - val_accuracy: 0.5217
Epoch 40/100
```

```
6/6 [====] - 6s 1s/step - loss: 0.6279 - accuracy: 0.6593 - val_loss: 0.6875 - val_accuracy: 0.6087
Epoch 41/100
6/6 [====] - 6s 1s/step - loss: 0.6368 - accuracy: 0.6648 - val_loss: 0.7770 - val_accuracy: 0.5217
Epoch 42/100
6/6 [====] - 6s 1s/step - loss: 0.6234 - accuracy: 0.6648 - val_loss: 0.7051 - val_accuracy: 0.6087
Epoch 43/100
6/6 [====] - 6s 1s/step - loss: 0.5704 - accuracy: 0.6758 - val_loss: 1.6890 - val_accuracy: 0.5435
Epoch 44/100
6/6 [====] - 6s 1s/step - loss: 0.8757 - accuracy: 0.6319 - val_loss: 0.6486 - val_accuracy: 0.7174
Epoch 45/100
6/6 [====] - 6s 1s/step - loss: 0.6140 - accuracy: 0.6484 - val_loss: 0.5760 - val_accuracy: 0.6087
Epoch 46/100
6/6 [====] - 6s 1s/step - loss: 0.6504 - accuracy: 0.6374 - val_loss: 0.6492 - val_accuracy: 0.5652
Epoch 47/100
6/6 [====] - 6s 1s/step - loss: 0.5489 - accuracy: 0.7143 - val_loss: 0.8881 - val_accuracy: 0.6087
Epoch 48/100
6/6 [====] - 6s 1s/step - loss: 0.6537 - accuracy: 0.6758 - val_loss: 0.6574 - val_accuracy: 0.6087
Epoch 49/100
6/6 [====] - 6s 1s/step - loss: 0.5693 - accuracy: 0.7253 - val_loss: 0.7288 - val_accuracy: 0.5217
Epoch 50/100
6/6 [====] - 6s 1s/step - loss: 0.5741 - accuracy: 0.6923 - val_loss: 0.7300 - val_accuracy: 0.5652
Epoch 51/100
6/6 [====] - 6s 1s/step - loss: 0.6509 - accuracy: 0.6538 - val_loss: 0.6371 - val_accuracy: 0.6087
Epoch 52/100
6/6 [====] - 6s 1s/step - loss: 0.5945 - accuracy: 0.6374 - val_loss: 0.6411 - val_accuracy: 0.6522
Epoch 53/100
6/6 [====] - 6s 1s/step - loss: 0.5379 - accuracy: 0.6813 - val_loss: 0.8692 - val_accuracy: 0.5870
Epoch 54/100
6/6 [====] - 6s 1s/step - loss: 0.6179 - accuracy: 0.6374 - val_loss: 0.8341 - val_accuracy: 0.6739
Epoch 55/100
6/6 [====] - 6s 1s/step - loss: 0.6070 - accuracy: 0.6538 - val_loss: 0.7372 - val_accuracy: 0.5652
Epoch 56/100
6/6 [====] - 6s 1s/step - loss: 0.5603 - accuracy: 0.6758 - val_loss: 0.7760 - val_accuracy: 0.5435
Epoch 57/100
6/6 [====] - 6s 1s/step - loss: 0.5534 - accuracy: 0.6923 - val_loss: 0.8674 - val_accuracy: 0.5435
Epoch 58/100
6/6 [====] - 6s 1s/step - loss: 0.5060 - accuracy: 0.7692 - val_loss: 0.7469 - val_accuracy: 0.5652
Epoch 59/100
6/6 [====] - 6s 1s/step - loss: 0.5645 - accuracy: 0.6758 - val_loss: 0.8693 - val_accuracy: 0.5652
Epoch 60/100
6/6 [====] - 6s 1s/step - loss: 0.5649 - accuracy: 0.7198 - val_loss: 0.9531 - val_accuracy: 0.4565
Epoch 61/100
6/6 [====] - 6s 1s/step - loss: 0.5644 - accuracy: 0.6978 - val_loss: 0.7474 - val_accuracy: 0.5870
Epoch 62/100
6/6 [====] - 6s 1s/step - loss: 0.5404 - accuracy: 0.7363 - val_loss: 0.8272 - val_accuracy: 0.5652
Epoch 63/100
6/6 [====] - 6s 1s/step - loss: 0.5367 - accuracy: 0.7033 - val_loss: 0.8719 - val_accuracy: 0.6304
Epoch 64/100
6/6 [====] - 6s 1s/step - loss: 0.5131 - accuracy: 0.7418 - val_loss: 1.0886 - val_accuracy: 0.5435
Epoch 65/100
6/6 [====] - 6s 1s/step - loss: 0.4840 - accuracy: 0.7308 - val_loss: 1.0825 - val_accuracy: 0.5217
Epoch 66/100
6/6 [====] - 6s 1s/step - loss: 0.5998 - accuracy: 0.6868 - val_loss: 1.3885 - val_accuracy: 0.5217
Epoch 67/100
6/6 [====] - 6s 1s/step - loss: 0.5641 - accuracy: 0.7143 - val_loss: 1.0499 - val_accuracy: 0.6522
Epoch 68/100
6/6 [====] - 6s 1s/step - loss: 0.5129 - accuracy: 0.7473 - val_loss: 0.9628 - val_accuracy: 0.6304
Epoch 69/100
6/6 [====] - 6s 1s/step - loss: 0.4873 - accuracy: 0.7473 - val_loss: 0.9262 - val_accuracy: 0.5217
Epoch 70/100
6/6 [====] - 6s 1s/step - loss: 0.5484 - accuracy: 0.6923 - val_loss: 0.7734 - val_accuracy: 0.5652
Epoch 71/100
6/6 [====] - 6s 1s/step - loss: 0.5076 - accuracy: 0.7802 - val_loss: 0.9924 - val_accuracy: 0.6957
Epoch 72/100
6/6 [====] - 6s 1s/step - loss: 0.4903 - accuracy: 0.7363 - val_loss: 0.9577 - val_accuracy: 0.6739
Epoch 73/100
6/6 [====] - 6s 1s/step - loss: 0.4970 - accuracy: 0.7582 - val_loss: 0.8465 - val_accuracy: 0.6304
Epoch 74/100
6/6 [====] - 6s 1s/step - loss: 0.5066 - accuracy: 0.7418 - val_loss: 0.7473 - val_accuracy: 0.5435
Epoch 75/100
6/6 [====] - 6s 1s/step - loss: 0.5246 - accuracy: 0.7418 - val_loss: 0.9996 - val_accuracy: 0.5217
Epoch 76/100
6/6 [====] - 6s 1s/step - loss: 0.4539 - accuracy: 0.8077 - val_loss: 1.1016 - val_accuracy: 0.4783
Epoch 77/100
6/6 [====] - 6s 1s/step - loss: 0.4867 - accuracy: 0.7692 - val_loss: 1.1279 - val_accuracy: 0.5652
Epoch 78/100
6/6 [====] - 6s 1s/step - loss: 0.5651 - accuracy: 0.7088 - val_loss: 0.8382 - val_accuracy: 0.5870
Epoch 79/100
6/6 [====] - 6s 1s/step - loss: 0.4619 - accuracy: 0.7473 - val_loss: 0.8870 - val_accuracy: 0.5652
Epoch 80/100
6/6 [====] - 6s 1s/step - loss: 0.4501 - accuracy: 0.7857 - val_loss: 0.9344 - val_accuracy: 0.5435
Epoch 81/100
```

```
6/6 [====] - 6s 1s/step - loss: 0.4747 - accuracy: 0.7582 - val_loss: 1.5038 - val_accuracy: 0.5435
Epoch 82/100
6/6 [====] - 6s 1s/step - loss: 0.5105 - accuracy: 0.7747 - val_loss: 1.0824 - val_accuracy: 0.6087
Epoch 83/100
6/6 [====] - 6s 1s/step - loss: 0.5480 - accuracy: 0.7253 - val_loss: 0.7521 - val_accuracy: 0.6522
Epoch 84/100
6/6 [====] - 6s 1s/step - loss: 0.4527 - accuracy: 0.8022 - val_loss: 0.8311 - val_accuracy: 0.6304
Epoch 85/100
6/6 [====] - 6s 1s/step - loss: 0.4948 - accuracy: 0.7582 - val_loss: 0.7125 - val_accuracy: 0.6304
Epoch 86/100
6/6 [====] - 6s 1s/step - loss: 0.4420 - accuracy: 0.7802 - val_loss: 1.0692 - val_accuracy: 0.6087
Epoch 87/100
6/6 [====] - 6s 1s/step - loss: 0.4393 - accuracy: 0.7692 - val_loss: 1.0531 - val_accuracy: 0.6087
Epoch 88/100
6/6 [====] - 6s 1s/step - loss: 0.4429 - accuracy: 0.7912 - val_loss: 1.1580 - val_accuracy: 0.5652
Epoch 89/100
6/6 [====] - 6s 1s/step - loss: 0.4521 - accuracy: 0.7802 - val_loss: 1.4009 - val_accuracy: 0.6739
Epoch 90/100
6/6 [====] - 6s 1s/step - loss: 0.4159 - accuracy: 0.8352 - val_loss: 0.8698 - val_accuracy: 0.7174
Epoch 91/100
6/6 [====] - 6s 1s/step - loss: 0.4493 - accuracy: 0.8297 - val_loss: 0.8641 - val_accuracy: 0.6304
Epoch 92/100
6/6 [====] - 6s 1s/step - loss: 0.4111 - accuracy: 0.8132 - val_loss: 1.1529 - val_accuracy: 0.6087
Epoch 93/100
6/6 [====] - 6s 1s/step - loss: 0.4830 - accuracy: 0.7912 - val_loss: 1.5141 - val_accuracy: 0.4783
Epoch 94/100
6/6 [====] - 6s 1s/step - loss: 0.5473 - accuracy: 0.7308 - val_loss: 0.8234 - val_accuracy: 0.6087
Epoch 95/100
6/6 [====] - 6s 1s/step - loss: 0.3811 - accuracy: 0.8187 - val_loss: 1.1994 - val_accuracy: 0.5870
Epoch 96/100
6/6 [====] - 6s 1s/step - loss: 0.4003 - accuracy: 0.8297 - val_loss: 1.1504 - val_accuracy: 0.6739
Epoch 97/100
6/6 [====] - 6s 1s/step - loss: 0.4143 - accuracy: 0.7967 - val_loss: 1.6051 - val_accuracy: 0.4783
Epoch 98/100
6/6 [====] - 6s 1s/step - loss: 0.4304 - accuracy: 0.8077 - val_loss: 0.8624 - val_accuracy: 0.6522
Epoch 99/100
6/6 [====] - 6s 1s/step - loss: 0.3747 - accuracy: 0.8077 - val_loss: 0.9013 - val_accuracy: 0.5000
Epoch 100/100
6/6 [====] - 6s 1s/step - loss: 0.4284 - accuracy: 0.7967 - val_loss: 0.9641 - val_accuracy: 0.6739
```

Experiment 7

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_30 (Conv2D)          (None, 318, 318, 32)      320
 max_pooling2d_30 (MaxPooling2D) (None, 159, 159, 32)      0
 conv2d_31 (Conv2D)          (None, 157, 157, 32)      9248
 max_pooling2d_31 (MaxPooling2D) (None, 78, 78, 32)        0
 conv2d_32 (Conv2D)          (None, 76, 76, 32)        9248
 max_pooling2d_32 (MaxPooling2D) (None, 38, 38, 32)        0
 conv2d_33 (Conv2D)          (None, 36, 36, 64)        18496
 max_pooling2d_33 (MaxPooling2D) (None, 18, 18, 64)        0
 conv2d_34 (Conv2D)          (None, 16, 16, 64)        36928
 max_pooling2d_34 (MaxPooling2D) (None, 8, 8, 64)          0
 flatten_6 (Flatten)         (None, 4096)              0
 dense_12 (Dense)            (None, 64)                262208
 dropout_6 (Dropout)         (None, 64)                0
 dense_13 (Dense)            (None, 2)                 130
=================================================================
Total params: 336,578
Trainable params: 336,578Non-trainable params: 0
_________________________________________________________________
Epoch 1/35
6/6 [====] - 2s 194ms/step - loss: 0.7140 - accuracy: 0.5165 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 2/35
6/6 [====] - 1s 152ms/step - loss: 0.6941 - accuracy: 0.4396 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3/35
6/6 [====] - 1s 149ms/step - loss: 0.6933 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4/35
6/6 [====] - 1s 149ms/step - loss: 0.6933 - accuracy: 0.4670 - val_loss: 0.6942 - val_accuracy: 0.5000
Epoch 5/35
6/6 [====] - 1s 149ms/step - loss: 0.6941 - accuracy: 0.5055 - val_loss: 0.6932 - val_accuracy: 0.5000
```

```
Epoch 6/35
6/6 [====] - 1s 151ms/step - loss: 0.6954 - accuracy: 0.4725 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 7/35
6/6 [====] - 1s 151ms/step - loss: 0.6935 - accuracy: 0.5000 - val_loss: 0.6928 - val_accuracy: 0.5000
Epoch 8/35
6/6 [====] - 1s 152ms/step - loss: 0.6938 - accuracy: 0.4945 - val_loss: 0.6926 - val_accuracy: 0.5000
Epoch 9/35
6/6 [====] - 1s 153ms/step - loss: 0.6964 - accuracy: 0.5110 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 10/35
6/6 [====] - 1s 151ms/step - loss: 0.6940 - accuracy: 0.4615 - val_loss: 0.6931 - val_accuracy: 0.5217
Epoch 11/35
6/6 [====] - 1s 150ms/step - loss: 0.6928 - accuracy: 0.4945 - val_loss: 0.6926 - val_accuracy: 0.5000
Epoch 12/35
6/6 [====] - 1s 152ms/step - loss: 0.6896 - accuracy: 0.5549 - val_loss: 0.7057 - val_accuracy: 0.5000
Epoch 13/35
6/6 [====] - 1s 163ms/step - loss: 0.6997 - accuracy: 0.5110 - val_loss: 0.6954 - val_accuracy: 0.5000
Epoch 14/35
6/6 [====] - 1s 154ms/step - loss: 0.6827 - accuracy: 0.5440 - val_loss: 0.6899 - val_accuracy: 0.5000
Epoch 15/35
6/6 [====] - 1s 152ms/step - loss: 0.6777 - accuracy: 0.5989 - val_loss: 0.6841 - val_accuracy: 0.5217
Epoch 16/35
6/6 [====] - 1s 152ms/step - loss: 0.6700 - accuracy: 0.5879 - val_loss: 0.6933 - val_accuracy: 0.5652
Epoch 17/35
6/6 [====] - 1s 153ms/step - loss: 0.6435 - accuracy: 0.6044 - val_loss: 0.7073 - val_accuracy: 0.4348
Epoch 18/35
6/6 [====] - 1s 152ms/step - loss: 0.6194 - accuracy: 0.6758 - val_loss: 0.7168 - val_accuracy: 0.5000
Epoch 19/35
6/6 [====] - 1s 164ms/step - loss: 0.5709 - accuracy: 0.6758 - val_loss: 0.8360 - val_accuracy: 0.4130
Epoch 20/35
6/6 [====] - 1s 150ms/step - loss: 0.5396 - accuracy: 0.6868 - val_loss: 0.7585 - val_accuracy: 0.5435
Epoch 21/35
6/6 [====] - 1s 152ms/step - loss: 0.5399 - accuracy: 0.7143 - val_loss: 0.7811 - val_accuracy: 0.6304
Epoch 22/35
6/6 [====] - 1s 164ms/step - loss: 0.4919 - accuracy: 0.6978 - val_loss: 0.7589 - val_accuracy: 0.5870
Epoch 23/35
6/6 [====] - 1s 163ms/step - loss: 0.4438 - accuracy: 0.7747 - val_loss: 0.8339 - val_accuracy: 0.5435
Epoch 24/35
6/6 [====] - 1s 151ms/step - loss: 0.4376 - accuracy: 0.7692 - val_loss: 0.9644 - val_accuracy: 0.5000
Epoch 25/35
6/6 [====] - 1s 151ms/step - loss: 0.3587 - accuracy: 0.7967 - val_loss: 1.0303 - val_accuracy: 0.5000
Epoch 26/35
6/6 [====] - 1s 152ms/step - loss: 0.3844 - accuracy: 0.8352 - val_loss: 1.3831 - val_accuracy: 0.5435
Epoch 27/35
6/6 [====] - 1s 153ms/step - loss: 0.3382 - accuracy: 0.8242 - val_loss: 1.3215 - val_accuracy: 0.5000
Epoch 28/35
6/6 [====] - 1s 152ms/step - loss: 0.3393 - accuracy: 0.8297 - val_loss: 1.9129 - val_accuracy: 0.4348
Epoch 29/35
6/6 [====] - 1s 154ms/step - loss: 0.3003 - accuracy: 0.8077 - val_loss: 1.3688 - val_accuracy: 0.5652
Epoch 30/35
6/6 [====] - 1s 165ms/step - loss: 0.2918 - accuracy: 0.8352 - val_loss: 1.4308 - val_accuracy: 0.5435
Epoch 31/35
6/6 [====] - 1s 164ms/step - loss: 0.2723 - accuracy: 0.8462 - val_loss: 1.8264 - val_accuracy: 0.4565
Epoch 32/35
6/6 [====] - 1s 152ms/step - loss: 0.2570 - accuracy: 0.8681 - val_loss: 1.3423 - val_accuracy: 0.5652
Epoch 33/35
6/6 [====] - 1s 149ms/step - loss: 0.2716 - accuracy: 0.8571 - val_loss: 1.6043 - val_accuracy: 0.5435
Epoch 34/35
6/6 [====] - 1s 153ms/step - loss: 0.2520 - accuracy: 0.8571 - val_loss: 2.3708 - val_accuracy: 0.5000
Epoch 35/35
6/6 [====] - 1s 151ms/step - loss: 0.2651 - accuracy: 0.8681 - val_loss: 2.0084 - val_accuracy: 0.5435
```

Experiment 8

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
conv2d_35 (Conv2D)          (None, 318, 318, 32)      320
 max_pooling2d_35 (MaxPooling2D) (None, 159, 159, 32)      0
 conv2d_36 (Conv2D)          (None, 157, 157, 32)      9248
 max_pooling2d_36 (MaxPooling2D) (None, 78, 78, 32)        0
 conv2d_37 (Conv2D)          (None, 76, 76, 32)        9248
 max_pooling2d_37 (MaxPooling2D) (None, 38, 38, 32)        0
 conv2d_38 (Conv2D)          (None, 36, 36, 64)        18496
 max_pooling2d_38 (MaxPooling2D) (None, 18, 18, 64)        0
 conv2d_39 (Conv2D)          (None, 16, 16, 64)        36928
 max_pooling2d_39 (MaxPooling2D) (None, 8, 8, 64)          0
 flatten_7 (Flatten)         (None, 4096)              0
 dense_14 (Dense)            (None, 64)                262208
 dropout_7 (Dropout)         (None, 64)                0
 dense_15 (Dense)            (None, 2)                 130
=================================================================
Total params: 336,578
Trainable params: 336,578
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
6/6 [====] - 4s 473ms/step - loss: 3.3589 - accuracy: 0.5385 - val_loss: 0.6917 - val_accuracy: 0.5000
Epoch 2/100
6/6 [====] - 2s 386ms/step - loss: 0.7073 - accuracy: 0.4615 - val_loss: 0.7009 - val_accuracy: 0.5217
Epoch 3/100
6/6 [====] - 2s 380ms/step - loss: 0.7097 - accuracy: 0.4615 - val_loss: 0.6955 - val_accuracy: 0.5435
Epoch 4/100
6/6 [====] - 2s 387ms/step - loss: 0.6972 - accuracy: 0.4890 - val_loss: 0.7050 - val_accuracy: 0.4783
Epoch 5/100
6/6 [====] - 2s 375ms/step - loss: 0.6941 - accuracy: 0.5220 - val_loss: 0.7177 - val_accuracy: 0.5000
Epoch 6/100
6/6 [====] - 2s 375ms/step - loss: 0.7471 - accuracy: 0.5055 - val_loss: 0.6943 - val_accuracy: 0.5217
Epoch 7/100
6/6 [====] - 2s 382ms/step - loss: 0.6927 - accuracy: 0.5220 - val_loss: 0.7065 - val_accuracy: 0.4783
Epoch 8/100
6/6 [====] - 2s 381ms/step - loss: 0.6968 - accuracy: 0.5055 - val_loss: 0.6953 - val_accuracy: 0.4565
Epoch 9/100
6/6 [====] - 2s 377ms/step - loss: 0.6940 - accuracy: 0.4780 - val_loss: 0.7255 - val_accuracy: 0.5000
Epoch 10/100
6/6 [====] - 2s 402ms/step - loss: 0.7604 - accuracy: 0.5275 - val_loss: 0.7046 - val_accuracy: 0.5000
Epoch 11/100
6/6 [====] - 2s 385ms/step - loss: 0.6949 - accuracy: 0.5220 - val_loss: 0.6951 - val_accuracy: 0.5217
Epoch 12/100
6/6 [====] - 2s 378ms/step - loss: 0.7237 - accuracy: 0.5549 - val_loss: 0.6967 - val_accuracy: 0.5217
Epoch 13/100
6/6 [====] - 2s 374ms/step - loss: 0.6997 - accuracy: 0.5055 - val_loss: 0.6894 - val_accuracy: 0.5435
Epoch 14/100
6/6 [====] - 2s 373ms/step - loss: 0.6862 - accuracy: 0.5495 - val_loss: 0.6849 - val_accuracy: 0.5870
Epoch 15/100
6/6 [====] - 2s 373ms/step - loss: 0.6941 - accuracy: 0.5330 - val_loss: 0.8086 - val_accuracy: 0.5000
Epoch 16/100
6/6 [====] - 2s 377ms/step - loss: 0.6954 - accuracy: 0.4945 - val_loss: 0.6919 - val_accuracy: 0.5652
Epoch 17/100
6/6 [====] - 2s 400ms/step - loss: 0.7049 - accuracy: 0.5055 - val_loss: 0.6984 - val_accuracy: 0.4783
Epoch 18/100
6/6 [====] - 2s 380ms/step - loss: 0.6851 - accuracy: 0.5714 - val_loss: 0.7565 - val_accuracy: 0.5217
Epoch 19/100
6/6 [====] - 2s 402ms/step - loss: 0.7024 - accuracy: 0.5220 - val_loss: 0.7171 - val_accuracy: 0.5435
Epoch 20/100
6/6 [====] - 2s 379ms/step - loss: 0.7023 - accuracy: 0.5110 - val_loss: 0.7089 - val_accuracy: 0.4783
Epoch 21/100
6/6 [====] - 2s 382ms/step - loss: 0.6624 - accuracy: 0.6044 - val_loss: 0.7863 - val_accuracy: 0.5000
Epoch 22/100
6/6 [====] - 2s 406ms/step - loss: 0.6877 - accuracy: 0.5714 - val_loss: 0.8314 - val_accuracy: 0.4565
Epoch 23/100
6/6 [====] - 2s 373ms/step - loss: 0.6602 - accuracy: 0.5659 - val_loss: 0.7301 - val_accuracy: 0.4565
Epoch 24/100
6/6 [====] - 2s 406ms/step - loss: 0.6492 - accuracy: 0.5714 - val_loss: 0.7633 - val_accuracy: 0.4130
Epoch 25/100
6/6 [====] - 2s 374ms/step - loss: 0.6483 - accuracy: 0.6154 - val_loss: 0.7320 - val_accuracy: 0.4783
Epoch 26/100
6/6 [====] - 2s 369ms/step - loss: 0.6576 - accuracy: 0.5934 - val_loss: 0.7103 - val_accuracy: 0.4348
Epoch 27/100
6/6 [====] - 2s 398ms/step - loss: 0.7330 - accuracy: 0.5659 - val_loss: 0.9205 - val_accuracy: 0.4348
```

```
Epoch 28/100
6/6 [====] - 2s 405ms/step - loss: 0.6513 - accuracy: 0.5879 - val_loss: 0.6831 - val_accuracy: 0.5435
Epoch 29/100
6/6 [====] - 2s 378ms/step - loss: 0.6549 - accuracy: 0.6154 - val_loss: 0.7275 - val_accuracy: 0.4565
Epoch 30/100
6/6 [====] - 2s 398ms/step - loss: 0.6428 - accuracy: 0.6044 - val_loss: 0.7643 - val_accuracy: 0.5000
Epoch 31/100
6/6 [====] - 2s 374ms/step - loss: 0.6497 - accuracy: 0.6154 - val_loss: 0.6816 - val_accuracy: 0.5217
Epoch 32/100
6/6 [====] - 2s 379ms/step - loss: 0.6478 - accuracy: 0.5934 - val_loss: 0.8017 - val_accuracy: 0.5000
Epoch 33/100
6/6 [====] - 2s 402ms/step - loss: 0.6144 - accuracy: 0.6264 - val_loss: 0.9710 - val_accuracy: 0.5435
Epoch 34/100
6/6 [====] - 2s 377ms/step - loss: 0.6498 - accuracy: 0.6209 - val_loss: 0.9597 - val_accuracy: 0.5000
Epoch 35/100
6/6 [====] - 2s 372ms/step - loss: 0.6138 - accuracy: 0.6264 - val_loss: 0.7192 - val_accuracy: 0.4565
Epoch 36/100
6/6 [====] - 2s 373ms/step - loss: 0.6478 - accuracy: 0.6154 - val_loss: 0.8435 - val_accuracy: 0.4565
Epoch 37/100
6/6 [====] - 2s 380ms/step - loss: 0.6814 - accuracy: 0.5604 - val_loss: 0.8473 - val_accuracy: 0.4130
Epoch 38/100
6/6 [====] - 2s 375ms/step - loss: 0.6547 - accuracy: 0.6154 - val_loss: 0.9174 - val_accuracy: 0.5000
Epoch 39/100
6/6 [====] - 2s 371ms/step - loss: 0.6549 - accuracy: 0.5989 - val_loss: 0.7958 - val_accuracy: 0.3913
Epoch 40/100
6/6 [====] - 2s 376ms/step - loss: 0.6004 - accuracy: 0.6099 - val_loss: 0.9059 - val_accuracy: 0.4783
Epoch 41/100
6/6 [====] - 2s 371ms/step - loss: 0.6273 - accuracy: 0.6264 - val_loss: 1.2249 - val_accuracy: 0.5000
Epoch 42/100
6/6 [====] - 2s 367ms/step - loss: 0.6150 - accuracy: 0.6264 - val_loss: 0.9515 - val_accuracy: 0.4130
Epoch 43/100
6/6 [====] - 2s 372ms/step - loss: 0.5958 - accuracy: 0.6703 - val_loss: 0.8849 - val_accuracy: 0.5217
Epoch 44/100
6/6 [====] - 2s 371ms/step - loss: 0.5960 - accuracy: 0.6484 - val_loss: 1.0241 - val_accuracy: 0.3913
Epoch 45/100
6/6 [====] - 2s 395ms/step - loss: 0.6154 - accuracy: 0.6429 - val_loss: 0.9890 - val_accuracy: 0.4565
Epoch 46/100
6/6 [====] - 2s 395ms/step - loss: 0.6113 - accuracy: 0.6703 - val_loss: 1.0289 - val_accuracy: 0.4565
Epoch 47/100
6/6 [====] - 2s 367ms/step - loss: 0.5472 - accuracy: 0.6923 - val_loss: 0.9136 - val_accuracy: 0.3913
Epoch 48/100
6/6 [====] - 2s 396ms/step - loss: 0.6091 - accuracy: 0.6813 - val_loss: 0.9362 - val_accuracy: 0.4130
Epoch 49/100
6/6 [====] - 2s 370ms/step - loss: 0.6081 - accuracy: 0.6538 - val_loss: 0.9687 - val_accuracy: 0.4565
Epoch 50/100
6/6 [====] - 2s 371ms/step - loss: 0.5586 - accuracy: 0.6923 - val_loss: 0.9707 - val_accuracy: 0.5000
Epoch 51/100
6/6 [====] - 2s 369ms/step - loss: 0.5344 - accuracy: 0.6923 - val_loss: 1.1707 - val_accuracy: 0.4348
Epoch 52/100
6/6 [====] - 2s 390ms/step - loss: 0.5870 - accuracy: 0.6484 - val_loss: 0.9028 - val_accuracy: 0.4348
Epoch 53/100
6/6 [====] - 2s 400ms/step - loss: 0.5618 - accuracy: 0.6868 - val_loss: 0.9571 - val_accuracy: 0.4783
Epoch 54/100
6/6 [====] - 2s 376ms/step - loss: 0.5876 - accuracy: 0.6868 - val_loss: 1.2676 - val_accuracy: 0.3478
Epoch 55/100
6/6 [====] - 2s 378ms/step - loss: 0.5764 - accuracy: 0.6923 - val_loss: 1.1968 - val_accuracy: 0.5652
Epoch 56/100
6/6 [====] - 2s 374ms/step - loss: 0.5741 - accuracy: 0.6923 - val_loss: 0.9204 - val_accuracy: 0.4130
Epoch 57/100
6/6 [====] - 2s 366ms/step - loss: 0.5698 - accuracy: 0.6978 - val_loss: 0.8865 - val_accuracy: 0.4348
Epoch 58/100
6/6 [====] - 2s 372ms/step - loss: 0.5031 - accuracy: 0.7692 - val_loss: 1.0259 - val_accuracy: 0.3913
Epoch 59/100
6/6 [====] - 2s 373ms/step - loss: 0.5618 - accuracy: 0.7088 - val_loss: 1.1343 - val_accuracy: 0.3696
Epoch 60/100
6/6 [====] - 2s 372ms/step - loss: 0.5210 - accuracy: 0.6923 - val_loss: 1.5271 - val_accuracy: 0.4130
Epoch 61/100
6/6 [====] - 2s 368ms/step - loss: 0.4789 - accuracy: 0.7912 - val_loss: 1.2900 - val_accuracy: 0.4130
Epoch 62/100
6/6 [====] - 2s 377ms/step - loss: 0.5386 - accuracy: 0.6813 - val_loss: 0.9595 - val_accuracy: 0.3913
Epoch 63/100
6/6 [====] - 2s 374ms/step - loss: 0.5398 - accuracy: 0.7418 - val_loss: 1.3064 - val_accuracy: 0.4783
Epoch 64/100
6/6 [====] - 2s 377ms/step - loss: 0.5285 - accuracy: 0.7527 - val_loss: 1.7168 - val_accuracy: 0.4130
Epoch 65/100
6/6 [====] - 2s 381ms/step - loss: 0.4803 - accuracy: 0.7857 - val_loss: 1.0782 - val_accuracy: 0.4783
Epoch 66/100
6/6 [====] - 2s 371ms/step - loss: 0.4982 - accuracy: 0.7418 - val_loss: 1.3565 - val_accuracy: 0.4130
Epoch 67/100
6/6 [====] - 2s 373ms/step - loss: 0.5708 - accuracy: 0.6868 - val_loss: 1.3604 - val_accuracy: 0.4348
Epoch 68/100
6/6 [====] - 2s 372ms/step - loss: 0.5235 - accuracy: 0.7253 - val_loss: 1.4790 - val_accuracy: 0.3913
```

```
Epoch 69/100
6/6 [====] - 2s 368ms/step - loss: 0.5240 - accuracy: 0.7033 - val_loss: 1.5203 - val_accuracy: 0.3043
Epoch 70/100
6/6 [====] - 2s 371ms/step - loss: 0.5048 - accuracy: 0.7527 - val_loss: 1.5230 - val_accuracy: 0.4130
Epoch 71/100
6/6 [====] - 2s 368ms/step - loss: 0.4746 - accuracy: 0.7692 - val_loss: 0.9708 - val_accuracy: 0.3261
Epoch 72/100
6/6 [====] - 2s 373ms/step - loss: 0.5719 - accuracy: 0.7253 - val_loss: 1.6191 - val_accuracy: 0.4783
Epoch 73/100
6/6 [====] - 2s 377ms/step - loss: 0.4709 - accuracy: 0.7747 - val_loss: 1.4729 - val_accuracy: 0.3478
Epoch 74/100
6/6 [====] - 2s 377ms/step - loss: 0.6697 - accuracy: 0.6538 - val_loss: 1.4434 - val_accuracy: 0.4565
Epoch 75/100
6/6 [====] - 2s 372ms/step - loss: 0.5120 - accuracy: 0.7253 - val_loss: 1.6413 - val_accuracy: 0.5000
Epoch 76/100
6/6 [====] - 2s 375ms/step - loss: 0.4596 - accuracy: 0.7692 - val_loss: 1.1472 - val_accuracy: 0.4783
Epoch 77/100
6/6 [====] - 2s 373ms/step - loss: 0.5058 - accuracy: 0.7692 - val_loss: 1.6396 - val_accuracy: 0.3261
Epoch 78/100
6/6 [====] - 2s 373ms/step - loss: 0.4339 - accuracy: 0.7747 - val_loss: 1.4979 - val_accuracy: 0.4565
Epoch 79/100
6/6 [====] - 2s 372ms/step - loss: 0.4886 - accuracy: 0.7418 - val_loss: 1.1710 - val_accuracy: 0.5217
Epoch 80/100
6/6 [====] - 2s 419ms/step - loss: 0.4735 - accuracy: 0.7527 - val_loss: 1.4712 - val_accuracy: 0.4783
Epoch 81/100
6/6 [====] - 2s 378ms/step - loss: 0.4021 - accuracy: 0.7802 - val_loss: 1.6956 - val_accuracy: 0.4348
Epoch 82/100
6/6 [====] - 2s 402ms/step - loss: 0.4125 - accuracy: 0.7692 - val_loss: 1.3094 - val_accuracy: 0.5000
Epoch 83/100
6/6 [====] - 2s 371ms/step - loss: 0.4184 - accuracy: 0.7802 - val_loss: 1.6563 - val_accuracy: 0.2826
Epoch 84/100
6/6 [====] - 2s 372ms/step - loss: 0.4344 - accuracy: 0.7967 - val_loss: 1.9476 - val_accuracy: 0.5000
Epoch 85/100
6/6 [====] - 2s 399ms/step - loss: 0.6992 - accuracy: 0.7363 - val_loss: 1.0580 - val_accuracy: 0.4348
Epoch 86/100
6/6 [====] - 2s 371ms/step - loss: 0.4732 - accuracy: 0.8022 - val_loss: 1.7314 - val_accuracy: 0.5217
Epoch 87/100
6/6 [====] - 2s 370ms/step - loss: 0.4092 - accuracy: 0.8022 - val_loss: 1.6320 - val_accuracy: 0.3261
Epoch 88/100
6/6 [====] - 2s 393ms/step - loss: 0.4774 - accuracy: 0.7527 - val_loss: 1.7093 - val_accuracy: 0.5000
Epoch 89/100
6/6 [====] - 2s 373ms/step - loss: 0.3723 - accuracy: 0.8407 - val_loss: 1.4244 - val_accuracy: 0.4783
Epoch 90/100
6/6 [====] - 2s 375ms/step - loss: 0.4873 - accuracy: 0.8242 - val_loss: 1.0649 - val_accuracy: 0.4783
Epoch 91/100
6/6 [====] - 2s 380ms/step - loss: 0.4662 - accuracy: 0.7857 - val_loss: 2.2062 - val_accuracy: 0.4783
Epoch 92/100
6/6 [====] - 2s 395ms/step - loss: 0.3956 - accuracy: 0.8242 - val_loss: 2.0248 - val_accuracy: 0.4783
Epoch 93/100
6/6 [====] - 2s 374ms/step - loss: 0.3655 - accuracy: 0.8132 - val_loss: 1.7527 - val_accuracy: 0.4783
Epoch 94/100
6/6 [====] - 2s 400ms/step - loss: 0.4074 - accuracy: 0.7967 - val_loss: 2.3401 - val_accuracy: 0.5000
Epoch 95/100
6/6 [====] - 2s 371ms/step - loss: 0.3833 - accuracy: 0.8022 - val_loss: 1.6314 - val_accuracy: 0.4348
Epoch 96/100
6/6 [====] - 2s 374ms/step - loss: 0.4557 - accuracy: 0.7747 - val_loss: 1.6300 - val_accuracy: 0.4348
Epoch 97/100
6/6 [====] - 2s 368ms/step - loss: 0.3983 - accuracy: 0.7912 - val_loss: 1.4365 - val_accuracy: 0.4348
Epoch 98/100
6/6 [====] - 2s 374ms/step - loss: 0.6638 - accuracy: 0.7747 - val_loss: 1.0734 - val_accuracy: 0.3696
Epoch 99/100
6/6 [====] - 2s 379ms/step - loss: 0.4263 - accuracy: 0.7802 - val_loss: 1.8616 - val_accuracy: 0.4348
Epoch 100/100
6/6 [====] - 2s 379ms/step - loss: 0.3525 - accuracy: 0.8352 - val_loss: 1.9552 - val_accuracy: 0.4130
```

Experiment 9

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_40 (Conv2D)          (None, 62, 62, 32)        896
 max_pooling2d_40 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_41 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_41 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_42 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_42 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_43 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_43 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_8 (Flatten)         (None, 256)               0
 dense_16 (Dense)            (None, 64)                16448
 dropout_8 (Dropout)         (None, 64)                0
 dense_17 (Dense)            (None, 2)                 130
=================================================================
Total params: 54,466
Trainable params: 54,466
Non-trainable params: 0
_________________________________________________________________
Epoch 1/35
62/62 [====] - 3s 21ms/step - loss: 0.6943 - accuracy: 0.4962 - val_loss: 0.6932 - val_accuracy: 0.5010
Epoch 2/35
62/62 [====] - 1s 15ms/step - loss: 0.6937 - accuracy: 0.4921 - val_loss: 0.6933 - val_accuracy: 0.4990
Epoch 3/35
62/62 [====] - 1s 13ms/step - loss: 0.6936 - accuracy: 0.5059 - val_loss: 0.6932 - val_accuracy: 0.4990
Epoch 4/35
62/62 [====] - 1s 14ms/step - loss: 0.6931 - accuracy: 0.5197 - val_loss: 0.6977 - val_accuracy: 0.4928
Epoch 5/35
62/62 [====] - 1s 13ms/step - loss: 0.6934 - accuracy: 0.5161 - val_loss: 0.6941 - val_accuracy: 0.4765
Epoch 6/35
62/62 [====] - 1s 14ms/step - loss: 0.6916 - accuracy: 0.5090 - val_loss: 0.6929 - val_accuracy: 0.5112
Epoch 7/35
62/62 [====] - 1s 14ms/step - loss: 0.6866 - accuracy: 0.5514 - val_loss: 0.6948 - val_accuracy: 0.5112
Epoch 8/35
62/62 [====] - 1s 15ms/step - loss: 0.6796 - accuracy: 0.5683 - val_loss: 0.7020 - val_accuracy: 0.5112
Epoch 9/35
62/62 [====] - 1s 13ms/step - loss: 0.6785 - accuracy: 0.5627 - val_loss: 0.6845 - val_accuracy: 0.5276
Epoch 10/35
62/62 [====] - 1s 14ms/step - loss: 0.6724 - accuracy: 0.5934 - val_loss: 0.6996 - val_accuracy: 0.5051
Epoch 11/35
62/62 [====] - 1s 15ms/step - loss: 0.6664 - accuracy: 0.6010 - val_loss: 0.7152 - val_accuracy: 0.5337
Epoch 12/35
62/62 [====] - 1s 13ms/step - loss: 0.6566 - accuracy: 0.6113 - val_loss: 0.8091 - val_accuracy: 0.5051
Epoch 13/35
62/62 [====] - 1s 14ms/step - loss: 0.6504 - accuracy: 0.6184 - val_loss: 0.7993 - val_accuracy: 0.5337
Epoch 14/35
62/62 [====] - 1s 15ms/step - loss: 0.6457 - accuracy: 0.6322 - val_loss: 0.7066 - val_accuracy: 0.5419
Epoch 15/35
62/62 [====] - 1s 14ms/step - loss: 0.6287 - accuracy: 0.6327 - val_loss: 0.7792 - val_accuracy: 0.4990
Epoch 16/35
62/62 [====] - 1s 15ms/step - loss: 0.6182 - accuracy: 0.6522 - val_loss: 0.7206 - val_accuracy: 0.5440
Epoch 17/35
62/62 [====] - 1s 15ms/step - loss: 0.6063 - accuracy: 0.6680 - val_loss: 0.6431 - val_accuracy: 0.6094
Epoch 18/35
62/62 [====] - 1s 15ms/step - loss: 0.5936 - accuracy: 0.6777 - val_loss: 0.8593 - val_accuracy: 0.5256
Epoch 19/35
62/62 [====] - 1s 15ms/step - loss: 0.5871 - accuracy: 0.6880 - val_loss: 0.6084 - val_accuracy: 0.6667
Epoch 20/35
62/62 [====] - 1s 13ms/step - loss: 0.5783 - accuracy: 0.6997 - val_loss: 0.6620 - val_accuracy: 0.5726
Epoch 21/35
62/62 [====] - 1s 13ms/step - loss: 0.5646 - accuracy: 0.7115 - val_loss: 0.5949 - val_accuracy: 0.6708
Epoch 22/35
62/62 [====] - 1s 14ms/step - loss: 0.5548 - accuracy: 0.7192 - val_loss: 0.6283 - val_accuracy: 0.6544
Epoch 23/35
62/62 [====] - 1s 14ms/step - loss: 0.5440 - accuracy: 0.7176 - val_loss: 0.6024 - val_accuracy: 0.6605
Epoch 24/35
62/62 [====] - 1s 15ms/step - loss: 0.5412 - accuracy: 0.7238 - val_loss: 0.6957 - val_accuracy: 0.6196
Epoch 25/35
62/62 [====] - 1s 15ms/step - loss: 0.5110 - accuracy: 0.7427 - val_loss: 0.8617 - val_accuracy: 0.5828
Epoch 26/35
62/62 [====] - 1s 15ms/step - loss: 0.5009 - accuracy: 0.7453 - val_loss: 0.6237 - val_accuracy: 0.6748
Epoch 27/35
62/62 [====] - 1s 15ms/step - loss: 0.4886 - accuracy: 0.7478 - val_loss: 0.6198 - val_accuracy: 0.6544
Epoch 28/35
62/62 [====] - 1s 13ms/step - loss: 0.4716 - accuracy: 0.7749 - val_loss: 0.6678 - val_accuracy: 0.6176
```

```
Epoch 29/35
62/62 [====] - 1s 13ms/step - loss: 0.4565 - accuracy: 0.7801 - val_loss: 0.7742 - val_accuracy: 0.6217
Epoch 30/35
62/62 [====] - 1s 14ms/step - loss: 0.4544 - accuracy: 0.7749 - val_loss: 0.6799 - val_accuracy: 0.6258
Epoch 31/35
62/62 [====] - 1s 14ms/step - loss: 0.4323 - accuracy: 0.7893 - val_loss: 0.7183 - val_accuracy: 0.6524
Epoch 32/35
62/62 [====] - 1s 14ms/step - loss: 0.4126 - accuracy: 0.8000 - val_loss: 0.6629 - val_accuracy: 0.6769
Epoch 33/35
62/62 [====] - 1s 14ms/step - loss: 0.4022 - accuracy: 0.8077 - val_loss: 0.8535 - val_accuracy: 0.6094
Epoch 34/35
62/62 [====] - 1s 15ms/step - loss: 0.3905 - accuracy: 0.8174 - val_loss: 0.8367 - val_accuracy: 0.6421
Epoch 35/35
62/62 [====] - 1s 15ms/step - loss: 0.3658 - accuracy: 0.8322 - val_loss: 0.7634 - val_accuracy: 0.6462
```

Experiment 10

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
conv2d_44 (Conv2D)           (None, 62, 62, 32)        896
 max_pooling2d_44 (MaxPooling2D) (None, 31, 31, 32)       0
 conv2d_45 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_45 (MaxPooling2D) (None, 14, 14, 32)       0
 conv2d_46 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_46 (MaxPooling2D) (None, 6, 6, 32)         0
 conv2d_47 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_47 (MaxPooling2D) (None, 2, 2, 64)         0
 flatten_9 (Flatten)         (None, 256)               0
 dense_18 (Dense)            (None, 64)                16448
 dropout_9 (Dropout)         (None, 64)                0
 dense_19 (Dense)            (None, 2)                 130
=================================================================
Total params: 54,466
Trainable params: 54,466
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
62/62 [====] - 6s 79ms/step - loss: 1.3481 - accuracy: 0.4798 - val_loss: 0.6917 - val_accuracy: 0.5378
Epoch 2/100
62/62 [====] - 5s 76ms/step - loss: 0.7152 - accuracy: 0.4967 - val_loss: 0.6937 - val_accuracy: 0.4990
Epoch 3/100
62/62 [====] - 5s 75ms/step - loss: 0.7105 - accuracy: 0.5253 - val_loss: 0.6917 - val_accuracy: 0.5031
Epoch 4/100
62/62 [====] - 5s 75ms/step - loss: 0.6972 - accuracy: 0.5166 - val_loss: 0.6998 - val_accuracy: 0.5133
Epoch 5/100
62/62 [====] - 5s 75ms/step - loss: 0.7042 - accuracy: 0.5141 - val_loss: 0.6931 - val_accuracy: 0.5092
Epoch 6/100
62/62 [====] - 5s 75ms/step - loss: 0.7058 - accuracy: 0.5049 - val_loss: 0.6938 - val_accuracy: 0.4867
Epoch 7/100
62/62 [====] - 5s 76ms/step - loss: 0.7033 - accuracy: 0.5136 - val_loss: 0.6888 - val_accuracy: 0.5133
Epoch 8/100
62/62 [====] - 5s 75ms/step - loss: 0.6967 - accuracy: 0.5120 - val_loss: 0.6974 - val_accuracy: 0.5031
Epoch 9/100
62/62 [====] - 5s 75ms/step - loss: 0.6931 - accuracy: 0.5202 - val_loss: 0.6914 - val_accuracy: 0.5051
Epoch 10/100
62/62 [====] - 5s 75ms/step - loss: 0.6965 - accuracy: 0.5176 - val_loss: 0.6894 - val_accuracy: 0.5112
Epoch 11/100
62/62 [====] - 5s 75ms/step - loss: 0.7087 - accuracy: 0.5130 - val_loss: 0.6906 - val_accuracy: 0.5235
Epoch 12/100
62/62 [====] - 5s 75ms/step - loss: 0.6969 - accuracy: 0.5223 - val_loss: 0.6905 - val_accuracy: 0.5215
Epoch 13/100
62/62 [====] - 5s 75ms/step - loss: 0.6960 - accuracy: 0.5110 - val_loss: 0.6923 - val_accuracy: 0.5194
Epoch 14/100
62/62 [====] - 5s 75ms/step - loss: 0.6980 - accuracy: 0.5141 - val_loss: 0.6927 - val_accuracy: 0.5010
Epoch 15/100
62/62 [====] - 5s 76ms/step - loss: 0.6927 - accuracy: 0.5294 - val_loss: 0.6971 - val_accuracy: 0.5092
Epoch 16/100
62/62 [====] - 5s 75ms/step - loss: 0.6943 - accuracy: 0.5166 - val_loss: 0.6962 - val_accuracy: 0.5051
Epoch 17/100
62/62 [====] - 5s 75ms/step - loss: 0.6926 - accuracy: 0.5325 - val_loss: 0.6962 - val_accuracy: 0.5256
Epoch 18/100
62/62 [====] - 5s 75ms/step - loss: 0.6909 - accuracy: 0.5274 - val_loss: 0.6953 - val_accuracy: 0.4949
```

```
Epoch 19/100
62/62 [====] - 5s 76ms/step - loss: 0.6911 - accuracy: 0.5228 - val_loss: 0.6902 - val_accuracy: 0.5297
Epoch 20/100
62/62 [====] - 5s 76ms/step - loss: 0.6923 - accuracy: 0.5427 - val_loss: 0.6993 - val_accuracy: 0.5562
Epoch 21/100
62/62 [====] - 5s 76ms/step - loss: 0.6860 - accuracy: 0.5427 - val_loss: 0.6922 - val_accuracy: 0.5665
Epoch 22/100
62/62 [====] - 5s 77ms/step - loss: 0.6927 - accuracy: 0.5309 - val_loss: 0.6948 - val_accuracy: 0.5440
Epoch 23/100
62/62 [====] - 5s 77ms/step - loss: 0.6916 - accuracy: 0.5253 - val_loss: 0.6955 - val_accuracy: 0.5562
Epoch 24/100
62/62 [====] - 5s 76ms/step - loss: 0.6901 - accuracy: 0.5320 - val_loss: 0.6986 - val_accuracy: 0.5562
Epoch 25/100
62/62 [====] - 5s 76ms/step - loss: 0.6927 - accuracy: 0.5407 - val_loss: 0.6952 - val_accuracy: 0.5092
Epoch 26/100
62/62 [====] - 5s 76ms/step - loss: 0.6876 - accuracy: 0.5407 - val_loss: 0.6885 - val_accuracy: 0.5051
Epoch 27/100
62/62 [====] - 5s 76ms/step - loss: 0.6853 - accuracy: 0.5499 - val_loss: 0.7211 - val_accuracy: 0.5337
Epoch 28/100
62/62 [====] - 5s 76ms/step - loss: 0.6932 - accuracy: 0.5514 - val_loss: 0.7005 - val_accuracy: 0.5665
Epoch 29/100
62/62 [====] - 5s 76ms/step - loss: 0.6786 - accuracy: 0.5524 - val_loss: 0.6988 - val_accuracy: 0.5378
Epoch 30/100
62/62 [====] - 5s 77ms/step - loss: 0.6922 - accuracy: 0.5535 - val_loss: 0.6686 - val_accuracy: 0.5787
Epoch 31/100
62/62 [====] - 5s 76ms/step - loss: 0.6767 - accuracy: 0.5673 - val_loss: 0.6953 - val_accuracy: 0.5665
Epoch 32/100
62/62 [====] - 5s 76ms/step - loss: 0.6812 - accuracy: 0.5729 - val_loss: 0.6770 - val_accuracy: 0.5808
Epoch 33/100
62/62 [====] - 5s 77ms/step - loss: 0.6726 - accuracy: 0.5882 - val_loss: 0.6864 - val_accuracy: 0.5583
Epoch 34/100
62/62 [====] - 5s 76ms/step - loss: 0.6755 - accuracy: 0.5801 - val_loss: 0.6694 - val_accuracy: 0.5685
Epoch 35/100
62/62 [====] - 5s 77ms/step - loss: 0.6777 - accuracy: 0.5995 - val_loss: 0.6869 - val_accuracy: 0.5992
Epoch 36/100
62/62 [====] - 5s 77ms/step - loss: 0.6712 - accuracy: 0.5801 - val_loss: 0.6776 - val_accuracy: 0.5951
Epoch 37/100
62/62 [====] - 5s 76ms/step - loss: 0.6752 - accuracy: 0.5724 - val_loss: 0.6638 - val_accuracy: 0.5787
Epoch 38/100
62/62 [====] - 5s 76ms/step - loss: 0.6757 - accuracy: 0.6015 - val_loss: 0.6867 - val_accuracy: 0.5562
Epoch 39/100
62/62 [====] - 5s 75ms/step - loss: 0.6689 - accuracy: 0.5964 - val_loss: 0.6700 - val_accuracy: 0.5787
Epoch 40/100
62/62 [====] - 5s 75ms/step - loss: 0.6729 - accuracy: 0.5908 - val_loss: 0.6833 - val_accuracy: 0.6258
Epoch 41/100
62/62 [====] - 5s 75ms/step - loss: 0.6617 - accuracy: 0.6153 - val_loss: 0.7069 - val_accuracy: 0.5706
Epoch 42/100
62/62 [====] - 5s 76ms/step - loss: 0.6400 - accuracy: 0.6271 - val_loss: 0.6803 - val_accuracy: 0.6196
Epoch 43/100
62/62 [====] - 5s 76ms/step - loss: 0.6507 - accuracy: 0.6297 - val_loss: 0.6656 - val_accuracy: 0.6646
Epoch 44/100
62/62 [====] - 5s 76ms/step - loss: 0.6536 - accuracy: 0.6102 - val_loss: 0.6784 - val_accuracy: 0.6319
Epoch 45/100
62/62 [====] - 5s 76ms/step - loss: 0.6478 - accuracy: 0.6445 - val_loss: 0.6811 - val_accuracy: 0.5808
Epoch 46/100
62/62 [====] - 5s 75ms/step - loss: 0.6426 - accuracy: 0.6419 - val_loss: 0.6991 - val_accuracy: 0.5583
Epoch 47/100
62/62 [====] - 5s 76ms/step - loss: 0.6476 - accuracy: 0.6527 - val_loss: 0.6454 - val_accuracy: 0.6380
Epoch 48/100
62/62 [====] - 5s 76ms/step - loss: 0.6228 - accuracy: 0.6665 - val_loss: 0.6316 - val_accuracy: 0.6401
Epoch 49/100
62/62 [====] - 5s 76ms/step - loss: 0.6315 - accuracy: 0.6588 - val_loss: 0.6364 - val_accuracy: 0.6605
Epoch 50/100
62/62 [====] - 5s 76ms/step - loss: 0.6280 - accuracy: 0.6655 - val_loss: 0.6369 - val_accuracy: 0.6708
Epoch 51/100
62/62 [====] - 5s 76ms/step - loss: 0.6373 - accuracy: 0.6624 - val_loss: 0.6618 - val_accuracy: 0.6094
Epoch 52/100
62/62 [====] - 5s 76ms/step - loss: 0.6234 - accuracy: 0.6675 - val_loss: 0.6428 - val_accuracy: 0.6176
Epoch 53/100
62/62 [====] - 5s 75ms/step - loss: 0.6224 - accuracy: 0.6854 - val_loss: 0.6165 - val_accuracy: 0.6748
Epoch 54/100
62/62 [====] - 5s 76ms/step - loss: 0.6221 - accuracy: 0.6660 - val_loss: 0.6429 - val_accuracy: 0.6380
Epoch 55/100
62/62 [====] - 5s 78ms/step - loss: 0.6022 - accuracy: 0.7074 - val_loss: 0.6779 - val_accuracy: 0.6176
Epoch 56/100
62/62 [====] - 5s 76ms/step - loss: 0.6025 - accuracy: 0.6926 - val_loss: 0.6047 - val_accuracy: 0.6687
Epoch 57/100
62/62 [====] - 5s 83ms/step - loss: 0.6174 - accuracy: 0.6752 - val_loss: 0.6986 - val_accuracy: 0.6155
Epoch 58/100
62/62 [====] - 5s 76ms/step - loss: 0.6127 - accuracy: 0.6910 - val_loss: 0.5992 - val_accuracy: 0.6912
Epoch 59/100
62/62 [====] - 5s 76ms/step - loss: 0.5868 - accuracy: 0.6921 - val_loss: 0.6286 - val_accuracy: 0.6237
```

```
Epoch 60/100
62/62 [====] - 5s 77ms/step - loss: 0.5727 - accuracy: 0.6951 - val_loss: 0.6091 - val_accuracy: 0.6871
Epoch 61/100
62/62 [====] - 5s 76ms/step - loss: 0.5853 - accuracy: 0.7110 - val_loss: 0.5959 - val_accuracy: 0.6892
Epoch 62/100
62/62 [====] - 5s 76ms/step - loss: 0.5895 - accuracy: 0.7090 - val_loss: 0.6354 - val_accuracy: 0.6810
Epoch 63/100
62/62 [====] - 5s 76ms/step - loss: 0.5726 - accuracy: 0.7049 - val_loss: 0.6006 - val_accuracy: 0.7076
Epoch 64/100
62/62 [====] - 5s 76ms/step - loss: 0.5854 - accuracy: 0.6957 - val_loss: 0.6102 - val_accuracy: 0.6912
Epoch 65/100
62/62 [====] - 5s 76ms/step - loss: 0.5648 - accuracy: 0.7182 - val_loss: 0.6207 - val_accuracy: 0.7178
Epoch 66/100
62/62 [====] - 5s 76ms/step - loss: 0.5564 - accuracy: 0.7217 - val_loss: 0.6252 - val_accuracy: 0.6503
Epoch 67/100
62/62 [====] - 5s 76ms/step - loss: 0.5815 - accuracy: 0.7263 - val_loss: 0.5947 - val_accuracy: 0.7076
Epoch 68/100
62/62 [====] - 5s 76ms/step - loss: 0.5651 - accuracy: 0.7141 - val_loss: 0.6664 - val_accuracy: 0.6748
Epoch 69/100
62/62 [====] - 5s 75ms/step - loss: 0.5670 - accuracy: 0.7299 - val_loss: 0.6150 - val_accuracy: 0.7096
Epoch 70/100
62/62 [====] - 5s 76ms/step - loss: 0.5583 - accuracy: 0.7340 - val_loss: 0.6754 - val_accuracy: 0.6585
Epoch 71/100
62/62 [====] - 5s 76ms/step - loss: 0.5785 - accuracy: 0.7095 - val_loss: 0.6735 - val_accuracy: 0.6196
Epoch 72/100
62/62 [====] - 5s 76ms/step - loss: 0.5837 - accuracy: 0.7120 - val_loss: 0.6220 - val_accuracy: 0.6810
Epoch 73/100
62/62 [====] - 5s 76ms/step - loss: 0.5621 - accuracy: 0.7238 - val_loss: 0.6900 - val_accuracy: 0.6319
Epoch 74/100
62/62 [====] - 5s 75ms/step - loss: 0.5544 - accuracy: 0.7279 - val_loss: 0.6628 - val_accuracy: 0.6503
Epoch 75/100
62/62 [====] - 5s 76ms/step - loss: 0.6038 - accuracy: 0.6941 - val_loss: 0.6095 - val_accuracy: 0.6564
Epoch 76/100
62/62 [====] - 5s 76ms/step - loss: 0.5376 - accuracy: 0.7391 - val_loss: 0.5864 - val_accuracy: 0.7219
Epoch 77/100
62/62 [====] - 5s 76ms/step - loss: 0.5606 - accuracy: 0.7253 - val_loss: 0.5914 - val_accuracy: 0.6544
Epoch 78/100
62/62 [====] - 5s 77ms/step - loss: 0.5672 - accuracy: 0.7243 - val_loss: 0.6242 - val_accuracy: 0.6687
Epoch 79/100
62/62 [====] - 5s 76ms/step - loss: 0.5641 - accuracy: 0.7284 - val_loss: 0.6456 - val_accuracy: 0.6503
Epoch 80/100
62/62 [====] - 5s 76ms/step - loss: 0.5454 - accuracy: 0.7284 - val_loss: 0.5855 - val_accuracy: 0.7239
Epoch 81/100
62/62 [====] - 5s 76ms/step - loss: 0.5664 - accuracy: 0.7258 - val_loss: 0.7238 - val_accuracy: 0.5399
Epoch 82/100
62/62 [====] - 5s 75ms/step - loss: 0.5308 - accuracy: 0.7570 - val_loss: 0.6109 - val_accuracy: 0.7198
Epoch 83/100
62/62 [====] - 5s 76ms/step - loss: 0.5610 - accuracy: 0.7294 - val_loss: 0.5841 - val_accuracy: 0.7382
Epoch 84/100
62/62 [====] - 5s 83ms/step - loss: 0.5578 - accuracy: 0.7274 - val_loss: 0.6459 - val_accuracy: 0.6626
Epoch 85/100
62/62 [====] - 5s 76ms/step - loss: 0.5543 - accuracy: 0.7376 - val_loss: 0.7899 - val_accuracy: 0.5542
Epoch 86/100
62/62 [====] - 5s 76ms/step - loss: 0.5430 - accuracy: 0.7458 - val_loss: 0.5870 - val_accuracy: 0.7117
Epoch 87/100
62/62 [====] - 5s 76ms/step - loss: 0.5672 - accuracy: 0.7125 - val_loss: 0.6337 - val_accuracy: 0.6708
Epoch 88/100
62/62 [====] - 5s 76ms/step - loss: 0.5489 - accuracy: 0.7488 - val_loss: 0.5803 - val_accuracy: 0.7382
Epoch 89/100
62/62 [====] - 5s 76ms/step - loss: 0.5394 - accuracy: 0.7468 - val_loss: 0.6869 - val_accuracy: 0.6135
Epoch 90/100
62/62 [====] - 5s 76ms/step - loss: 0.5691 - accuracy: 0.7294 - val_loss: 0.6394 - val_accuracy: 0.6339
Epoch 91/100
62/62 [====] - 5s 76ms/step - loss: 0.5520 - accuracy: 0.7289 - val_loss: 0.6108 - val_accuracy: 0.6892
Epoch 92/100
62/62 [====] - 5s 77ms/step - loss: 0.5757 - accuracy: 0.7386 - val_loss: 0.6434 - val_accuracy: 0.6933
Epoch 93/100
62/62 [====] - 5s 77ms/step - loss: 0.5444 - accuracy: 0.7575 - val_loss: 0.6546 - val_accuracy: 0.6687
Epoch 94/100
62/62 [====] - 5s 78ms/step - loss: 0.5325 - accuracy: 0.7453 - val_loss: 0.6080 - val_accuracy: 0.7157
Epoch 95/100
62/62 [====] - 5s 77ms/step - loss: 0.5486 - accuracy: 0.7396 - val_loss: 0.7506 - val_accuracy: 0.6217
Epoch 96/100
62/62 [====] - 5s 77ms/step - loss: 0.6270 - accuracy: 0.6834 - val_loss: 0.6622 - val_accuracy: 0.5685
Epoch 97/100
62/62 [====] - 5s 78ms/step - loss: 0.5597 - accuracy: 0.7253 - val_loss: 0.5789 - val_accuracy: 0.7526
Epoch 98/100
62/62 [====] - 5s 78ms/step - loss: 0.5402 - accuracy: 0.7499 - val_loss: 0.5920 - val_accuracy: 0.7526
Epoch 99/100
62/62 [====] - 5s 77ms/step - loss: 0.5302 - accuracy: 0.7483 - val_loss: 0.7064 - val_accuracy: 0.6892
Epoch 100/100
62/62 [====] - 5s 77ms/step - loss: 0.5513 - accuracy: 0.7468 - val_loss: 0.5881 - val_accuracy: 0.7444
```

Experiment 11

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_48 (Conv2D)          (None, 62, 62, 32)        896
 max_pooling2d_48 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_49 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_49 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_50 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_50 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_51 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_51 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_10 (Flatten)        (None, 256)               0
 dense_20 (Dense)            (None, 64)                16448
 dropout_10 (Dropout)        (None, 64)                0
 dense_21 (Dense)            (None, 2)                 130
=================================================================
Total params: 54,466
Trainable params: 54,466
Non-trainable params: 0
_________________________________________________________________
Epoch 1/35
62/62 [====] - 2s 20ms/step - loss: 0.6944 - accuracy: 0.4982 - val_loss: 0.6940 - val_accuracy: 0.5010
Epoch 2/35
62/62 [====] - 1s 17ms/step - loss: 0.6940 - accuracy: 0.5038 - val_loss: 0.6932 - val_accuracy: 0.5010
Epoch 3/35
62/62 [====] - 1s 16ms/step - loss: 0.6941 - accuracy: 0.5110 - val_loss: 0.6925 - val_accuracy: 0.5746
Epoch 4/35
62/62 [====] - 1s 15ms/step - loss: 0.6938 - accuracy: 0.4921 - val_loss: 0.6931 - val_accuracy: 0.4990
Epoch 5/35
62/62 [====] - 1s 15ms/step - loss: 0.6934 - accuracy: 0.5033 - val_loss: 0.6928 - val_accuracy: 0.5112
Epoch 6/35
62/62 [====] - 1s 16ms/step - loss: 0.6927 - accuracy: 0.5171 - val_loss: 0.7058 - val_accuracy: 0.4990
Epoch 7/35
62/62 [====] - 1s 15ms/step - loss: 0.6915 - accuracy: 0.5381 - val_loss: 0.6924 - val_accuracy: 0.5031
Epoch 8/35
62/62 [====] - 1s 15ms/step - loss: 0.6869 - accuracy: 0.5463 - val_loss: 0.6878 - val_accuracy: 0.5051
Epoch 9/35
62/62 [====] - 1s 15ms/step - loss: 0.6834 - accuracy: 0.5545 - val_loss: 0.6724 - val_accuracy: 0.5726
Epoch 10/35
62/62 [====] - 1s 16ms/step - loss: 0.6805 - accuracy: 0.5673 - val_loss: 0.6917 - val_accuracy: 0.5051
Epoch 11/35
62/62 [====] - 1s 15ms/step - loss: 0.6736 - accuracy: 0.5754 - val_loss: 0.6923 - val_accuracy: 0.5194
Epoch 12/35
62/62 [====] - 1s 15ms/step - loss: 0.6623 - accuracy: 0.6113 - val_loss: 0.7179 - val_accuracy: 0.5256
Epoch 13/35
62/62 [====] - 1s 15ms/step - loss: 0.6616 - accuracy: 0.6056 - val_loss: 0.6899 - val_accuracy: 0.5297
Epoch 14/35
62/62 [====] - 1s 15ms/step - loss: 0.6516 - accuracy: 0.6107 - val_loss: 0.6682 - val_accuracy: 0.5971
Epoch 15/35
62/62 [====] - 1s 15ms/step - loss: 0.6372 - accuracy: 0.6327 - val_loss: 0.9416 - val_accuracy: 0.5133
Epoch 16/35
62/62 [====] - 1s 14ms/step - loss: 0.6528 - accuracy: 0.6297 - val_loss: 0.6574 - val_accuracy: 0.6278
Epoch 17/35
62/62 [====] - 1s 14ms/step - loss: 0.6216 - accuracy: 0.6558 - val_loss: 0.6764 - val_accuracy: 0.6074
Epoch 18/35
62/62 [====] - 1s 15ms/step - loss: 0.6083 - accuracy: 0.6762 - val_loss: 0.6990 - val_accuracy: 0.5481
Epoch 19/35
62/62 [====] - 1s 15ms/step - loss: 0.6076 - accuracy: 0.6634 - val_loss: 0.7331 - val_accuracy: 0.5276
Epoch 20/35
62/62 [====] - 1s 15ms/step - loss: 0.5841 - accuracy: 0.6875 - val_loss: 0.9879 - val_accuracy: 0.5276
Epoch 21/35
62/62 [====] - 1s 15ms/step - loss: 0.5768 - accuracy: 0.7008 - val_loss: 0.6998 - val_accuracy: 0.6094
Epoch 22/35
62/62 [====] - 1s 15ms/step - loss: 0.5519 - accuracy: 0.7146 - val_loss: 0.7514 - val_accuracy: 0.5849
Epoch 23/35
62/62 [====] - 1s 15ms/step - loss: 0.5363 - accuracy: 0.7263 - val_loss: 0.8742 - val_accuracy: 0.5787
Epoch 24/35
62/62 [====] - 1s 15ms/step - loss: 0.5122 - accuracy: 0.7494 - val_loss: 0.6879 - val_accuracy: 0.6380
Epoch 25/35
62/62 [====] - 1s 14ms/step - loss: 0.4877 - accuracy: 0.7627 - val_loss: 0.6699 - val_accuracy: 0.6299
Epoch 26/35
62/62 [====] - 1s 14ms/step - loss: 0.4672 - accuracy: 0.7652 - val_loss: 0.8026 - val_accuracy: 0.5930
Epoch 27/35
```

```
62/62 [====] - 1s 14ms/step - loss: 0.4554 - accuracy: 0.7852 - val_loss: 0.7244 - val_accuracy: 0.6564
Epoch 28/35
62/62 [====] - 1s 14ms/step - loss: 0.4208 - accuracy: 0.8128 - val_loss: 0.7788 - val_accuracy: 0.6237
Epoch 29/35
62/62 [====] - 1s 15ms/step - loss: 0.3964 - accuracy: 0.8205 - val_loss: 0.8843 - val_accuracy: 0.6585
Epoch 30/35
62/62 [====] - 1s 14ms/step - loss: 0.3872 - accuracy: 0.8230 - val_loss: 0.7729 - val_accuracy: 0.6033
Epoch 31/35
62/62 [====] - 1s 14ms/step - loss: 0.3510 - accuracy: 0.8445 - val_loss: 0.9451 - val_accuracy: 0.6544
Epoch 32/35
62/62 [====] - 1s 14ms/step - loss: 0.3362 - accuracy: 0.8506 - val_loss: 0.8264 - val_accuracy: 0.6503
Epoch 33/35
62/62 [====] - 1s 14ms/step - loss: 0.3030 - accuracy: 0.8645 - val_loss: 0.8952 - val_accuracy: 0.5971
Epoch 34/35
62/62 [====] - 1s 14ms/step - loss: 0.2813 - accuracy: 0.8757 - val_loss: 0.8260 - val_accuracy: 0.6687
Epoch 35/35
62/62 [====] - 1s 14ms/step - loss: 0.2512 - accuracy: 0.8890 - val_loss: 1.0784 - val_accuracy: 0.6483
```

Experiment 12

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_52 (Conv2D)          (None, 62, 62, 32)        896
 max_pooling2d_52 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_53 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_53 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_54 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_54 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_55 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_55 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_11 (Flatten)        (None, 256)               0
 dense_22 (Dense)            (None, 64)                16448
 dropout_11 (Dropout)        (None, 64)                0
 dense_23 (Dense)            (None, 2)                 130
=================================================================
Total params: 54,466
Trainable params: 54,466
Non-trainable params: 0
_________________________________________________________________
Epoch 1/200
62/62 [====] - 19s 81ms/step - loss: 1.2729 - accuracy: 0.5151 - val_loss: 0.6903 - val_accuracy: 0.5256
Epoch 2/200
62/62 [====] - 5s 75ms/step - loss: 0.7078 - accuracy: 0.4926 - val_loss: 0.6910 - val_accuracy: 0.5133
Epoch 3/200
62/62 [====] - 5s 73ms/step - loss: 0.7065 - accuracy: 0.5182 - val_loss: 0.6926 - val_accuracy: 0.5092
Epoch 4/200
62/62 [====] - 5s 75ms/step - loss: 0.7140 - accuracy: 0.5003 - val_loss: 0.6973 - val_accuracy: 0.5031
Epoch 5/200
62/62 [====] - 5s 74ms/step - loss: 0.6968 - accuracy: 0.5125 - val_loss: 0.6928 - val_accuracy: 0.5276
Epoch 6/200
62/62 [====] - 5s 76ms/step - loss: 0.6988 - accuracy: 0.5049 - val_loss: 0.6957 - val_accuracy: 0.4990
Epoch 7/200
62/62 [====] - 5s 74ms/step - loss: 0.6976 - accuracy: 0.4916 - val_loss: 0.6943 - val_accuracy: 0.5051
Epoch 8/200
62/62 [====] - 5s 75ms/step - loss: 0.7018 - accuracy: 0.5008 - val_loss: 0.6930 - val_accuracy: 0.5031
Epoch 9/200
62/62 [====] - 5s 75ms/step - loss: 0.6929 - accuracy: 0.5202 - val_loss: 0.6913 - val_accuracy: 0.5256
Epoch 10/200
62/62 [====] - 5s 75ms/step - loss: 0.7323 - accuracy: 0.5069 - val_loss: 0.6852 - val_accuracy: 0.5133
Epoch 11/200
62/62 [====] - 5s 75ms/step - loss: 0.6878 - accuracy: 0.5059 - val_loss: 0.6943 - val_accuracy: 0.5072
Epoch 12/200
62/62 [====] - 5s 74ms/step - loss: 0.6948 - accuracy: 0.5182 - val_loss: 0.6916 - val_accuracy: 0.5112
Epoch 13/200
62/62 [====] - 5s 75ms/step - loss: 0.6916 - accuracy: 0.5427 - val_loss: 0.6866 - val_accuracy: 0.5358
Epoch 14/200
62/62 [====] - 5s 74ms/step - loss: 0.6893 - accuracy: 0.5345 - val_loss: 0.6934 - val_accuracy: 0.5112
Epoch 15/200
62/62 [====] - 5s 75ms/step - loss: 0.6907 - accuracy: 0.5269 - val_loss: 0.6921 - val_accuracy: 0.5072
Epoch 16/200
62/62 [====] - 5s 75ms/step - loss: 0.6952 - accuracy: 0.5197 - val_loss: 0.6905 - val_accuracy: 0.5051
Epoch 17/200
```

```
62/62 [====] - 5s 75ms/step - loss: 0.6880 - accuracy: 0.5120 - val_loss: 0.6927 - val_accuracy: 0.5072
Epoch 18/200
62/62 [====] - 5s 74ms/step - loss: 0.7038 - accuracy: 0.5253 - val_loss: 0.6882 - val_accuracy: 0.5624
Epoch 19/200
62/62 [====] - 5s 74ms/step - loss: 0.6786 - accuracy: 0.5499 - val_loss: 0.6844 - val_accuracy: 0.5849
Epoch 20/200
62/62 [====] - 5s 74ms/step - loss: 0.6809 - accuracy: 0.5611 - val_loss: 0.6996 - val_accuracy: 0.5501
Epoch 21/200
62/62 [====] - 5s 73ms/step - loss: 0.6919 - accuracy: 0.5488 - val_loss: 0.6889 - val_accuracy: 0.5297
Epoch 22/200
62/62 [====] - 5s 76ms/step - loss: 0.6979 - accuracy: 0.5494 - val_loss: 0.6830 - val_accuracy: 0.5399
Epoch 23/200
62/62 [====] - 5s 75ms/step - loss: 0.6825 - accuracy: 0.5601 - val_loss: 0.6829 - val_accuracy: 0.5501
Epoch 24/200
62/62 [====] - 5s 75ms/step - loss: 0.6845 - accuracy: 0.5499 - val_loss: 0.6925 - val_accuracy: 0.5092
Epoch 25/200
62/62 [====] - 5s 74ms/step - loss: 0.6977 - accuracy: 0.5483 - val_loss: 0.6916 - val_accuracy: 0.5337
Epoch 26/200
62/62 [====] - 5s 75ms/step - loss: 0.6780 - accuracy: 0.5442 - val_loss: 0.6935 - val_accuracy: 0.5276
Epoch 27/200
62/62 [====] - 5s 73ms/step - loss: 0.6846 - accuracy: 0.5499 - val_loss: 0.6868 - val_accuracy: 0.5378
Epoch 28/200
62/62 [====] - 5s 74ms/step - loss: 0.6838 - accuracy: 0.5611 - val_loss: 0.6925 - val_accuracy: 0.6033
Epoch 29/200
62/62 [====] - 5s 75ms/step - loss: 0.6800 - accuracy: 0.5468 - val_loss: 0.6908 - val_accuracy: 0.5521
Epoch 30/200
62/62 [====] - 5s 75ms/step - loss: 0.6758 - accuracy: 0.5806 - val_loss: 0.7148 - val_accuracy: 0.5460
Epoch 31/200
62/62 [====] - 5s 75ms/step - loss: 0.6728 - accuracy: 0.5647 - val_loss: 0.7065 - val_accuracy: 0.5521
Epoch 32/200
62/62 [====] - 5s 75ms/step - loss: 0.6751 - accuracy: 0.5964 - val_loss: 0.6890 - val_accuracy: 0.5603
Epoch 33/200
62/62 [====] - 5s 75ms/step - loss: 0.6819 - accuracy: 0.5806 - val_loss: 0.6954 - val_accuracy: 0.5235
Epoch 34/200
62/62 [====] - 5s 74ms/step - loss: 0.6808 - accuracy: 0.5575 - val_loss: 0.6812 - val_accuracy: 0.6135
Epoch 35/200
62/62 [====] - 5s 74ms/step - loss: 0.6723 - accuracy: 0.5928 - val_loss: 0.6788 - val_accuracy: 0.5440
Epoch 36/200
62/62 [====] - 5s 75ms/step - loss: 0.6674 - accuracy: 0.5826 - val_loss: 0.6850 - val_accuracy: 0.5521
Epoch 37/200
62/62 [====] - 5s 75ms/step - loss: 0.6708 - accuracy: 0.5785 - val_loss: 0.6703 - val_accuracy: 0.5992
Epoch 38/200
62/62 [====] - 5s 74ms/step - loss: 0.6784 - accuracy: 0.5964 - val_loss: 0.7557 - val_accuracy: 0.6053
Epoch 39/200
62/62 [====] - 5s 74ms/step - loss: 0.6815 - accuracy: 0.6041 - val_loss: 0.6894 - val_accuracy: 0.5072
Epoch 40/200
62/62 [====] - 5s 74ms/step - loss: 0.6761 - accuracy: 0.5642 - val_loss: 0.6971 - val_accuracy: 0.5890
Epoch 41/200
62/62 [====] - 5s 74ms/step - loss: 0.6682 - accuracy: 0.5877 - val_loss: 0.6947 - val_accuracy: 0.5153
Epoch 42/200
62/62 [====] - 5s 74ms/step - loss: 0.6674 - accuracy: 0.5990 - val_loss: 0.6685 - val_accuracy: 0.6217
Epoch 43/200
62/62 [====] - 5s 75ms/step - loss: 0.6754 - accuracy: 0.5596 - val_loss: 0.6713 - val_accuracy: 0.5992
Epoch 44/200
62/62 [====] - 5s 76ms/step - loss: 0.6572 - accuracy: 0.5995 - val_loss: 0.6787 - val_accuracy: 0.6401
Epoch 45/200
62/62 [====] - 5s 75ms/step - loss: 0.6694 - accuracy: 0.6061 - val_loss: 0.6691 - val_accuracy: 0.6319
Epoch 46/200
62/62 [====] - 5s 74ms/step - loss: 0.6716 - accuracy: 0.6143 - val_loss: 0.6934 - val_accuracy: 0.6380
Epoch 47/200
62/62 [====] - 5s 74ms/step - loss: 0.6829 - accuracy: 0.6051 - val_loss: 0.6841 - val_accuracy: 0.6237
Epoch 48/200
62/62 [====] - 5s 75ms/step - loss: 0.6497 - accuracy: 0.6240 - val_loss: 0.6807 - val_accuracy: 0.6012
Epoch 49/200
62/62 [====] - 5s 74ms/step - loss: 0.6412 - accuracy: 0.6373 - val_loss: 0.7040 - val_accuracy: 0.6033
Epoch 50/200
62/62 [====] - 5s 75ms/step - loss: 0.6736 - accuracy: 0.6020 - val_loss: 0.6564 - val_accuracy: 0.6339
Epoch 51/200
62/62 [====] - 5s 76ms/step - loss: 0.6641 - accuracy: 0.6133 - val_loss: 0.6677 - val_accuracy: 0.6421
Epoch 52/200
62/62 [====] - 5s 75ms/step - loss: 0.6518 - accuracy: 0.6215 - val_loss: 0.7388 - val_accuracy: 0.5971
Epoch 53/200
62/62 [====] - 5s 74ms/step - loss: 0.6456 - accuracy: 0.6205 - val_loss: 0.6569 - val_accuracy: 0.6115
Epoch 54/200
62/62 [====] - 5s 74ms/step - loss: 0.6572 - accuracy: 0.6194 - val_loss: 0.6909 - val_accuracy: 0.6380
Epoch 55/200
62/62 [====] - 5s 74ms/step - loss: 0.6423 - accuracy: 0.6312 - val_loss: 0.6636 - val_accuracy: 0.6094
Epoch 56/200
62/62 [====] - 5s 74ms/step - loss: 0.6382 - accuracy: 0.6465 - val_loss: 0.6247 - val_accuracy: 0.6871
Epoch 57/200
62/62 [====] - 5s 74ms/step - loss: 0.6489 - accuracy: 0.6517 - val_loss: 0.6802 - val_accuracy: 0.5930
Epoch 58/200
```

```
62/62 [====] - 5s 74ms/step - loss: 0.6658 - accuracy: 0.6512 - val_loss: 0.6853 - val_accuracy: 0.5869
Epoch 59/200
62/62 [====] - 5s 75ms/step - loss: 0.6500 - accuracy: 0.6358 - val_loss: 0.6582 - val_accuracy: 0.6217
Epoch 60/200
62/62 [====] - 5s 75ms/step - loss: 0.6474 - accuracy: 0.6327 - val_loss: 0.6412 - val_accuracy: 0.6708
Epoch 61/200
62/62 [====] - 5s 74ms/step - loss: 0.6316 - accuracy: 0.6598 - val_loss: 0.7016 - val_accuracy: 0.6605
Epoch 62/200
62/62 [====] - 5s 74ms/step - loss: 0.6402 - accuracy: 0.6522 - val_loss: 0.6687 - val_accuracy: 0.5910
Epoch 63/200
62/62 [====] - 5s 74ms/step - loss: 0.6386 - accuracy: 0.6767 - val_loss: 0.6950 - val_accuracy: 0.5890
Epoch 64/200
62/62 [====] - 5s 74ms/step - loss: 0.6203 - accuracy: 0.6706 - val_loss: 0.7093 - val_accuracy: 0.6135
Epoch 65/200
62/62 [====] - 5s 75ms/step - loss: 0.6396 - accuracy: 0.6552 - val_loss: 0.6436 - val_accuracy: 0.6360
Epoch 66/200
62/62 [====] - 5s 75ms/step - loss: 0.6183 - accuracy: 0.6660 - val_loss: 0.6402 - val_accuracy: 0.6605
Epoch 67/200
62/62 [====] - 5s 73ms/step - loss: 0.6309 - accuracy: 0.6731 - val_loss: 0.7270 - val_accuracy: 0.6278
Epoch 68/200
62/62 [====] - 5s 74ms/step - loss: 0.6281 - accuracy: 0.6716 - val_loss: 0.6607 - val_accuracy: 0.6319
Epoch 69/200
62/62 [====] - 5s 74ms/step - loss: 0.6337 - accuracy: 0.6680 - val_loss: 0.8089 - val_accuracy: 0.6483
Epoch 70/200
62/62 [====] - 5s 75ms/step - loss: 0.6294 - accuracy: 0.6870 - val_loss: 0.6605 - val_accuracy: 0.6483
Epoch 71/200
62/62 [====] - 5s 74ms/step - loss: 0.6958 - accuracy: 0.5627 - val_loss: 0.6866 - val_accuracy: 0.5256
Epoch 72/200
62/62 [====] - 5s 75ms/step - loss: 0.6873 - accuracy: 0.5570 - val_loss: 0.6803 - val_accuracy: 0.5746
Epoch 73/200
62/62 [====] - 5s 74ms/step - loss: 0.6964 - accuracy: 0.5662 - val_loss: 0.6911 - val_accuracy: 0.5317
Epoch 74/200
62/62 [====] - 5s 74ms/step - loss: 0.6860 - accuracy: 0.5483 - val_loss: 0.7001 - val_accuracy: 0.5194
Epoch 75/200
62/62 [====] - 5s 74ms/step - loss: 0.6821 - accuracy: 0.5816 - val_loss: 0.7065 - val_accuracy: 0.5235
Epoch 76/200
62/62 [====] - 5s 74ms/step - loss: 0.7030 - accuracy: 0.5570 - val_loss: 0.7100 - val_accuracy: 0.5153
Epoch 77/200
62/62 [====] - 5s 73ms/step - loss: 0.6763 - accuracy: 0.5693 - val_loss: 0.6862 - val_accuracy: 0.5072
Epoch 78/200
62/62 [====] - 5s 74ms/step - loss: 0.6814 - accuracy: 0.5795 - val_loss: 0.6784 - val_accuracy: 0.6053
Epoch 79/200
62/62 [====] - 5s 74ms/step - loss: 0.6612 - accuracy: 0.6128 - val_loss: 0.8017 - val_accuracy: 0.6012
Epoch 80/200
62/62 [====] - 5s 74ms/step - loss: 0.6732 - accuracy: 0.5928 - val_loss: 0.6933 - val_accuracy: 0.6053
Epoch 81/200
62/62 [====] - 5s 74ms/step - loss: 0.6279 - accuracy: 0.6609 - val_loss: 0.6472 - val_accuracy: 0.6319
Epoch 82/200
62/62 [====] - 5s 74ms/step - loss: 0.6427 - accuracy: 0.6460 - val_loss: 0.6760 - val_accuracy: 0.6339
Epoch 83/200
62/62 [====] - 5s 74ms/step - loss: 0.6333 - accuracy: 0.6803 - val_loss: 0.7242 - val_accuracy: 0.6053
Epoch 84/200
62/62 [====] - 5s 74ms/step - loss: 0.6142 - accuracy: 0.6747 - val_loss: 0.6779 - val_accuracy: 0.6503
Epoch 85/200
62/62 [====] - 5s 74ms/step - loss: 0.6160 - accuracy: 0.6726 - val_loss: 0.6651 - val_accuracy: 0.6524
Epoch 86/200
62/62 [====] - 5s 74ms/step - loss: 0.6154 - accuracy: 0.6711 - val_loss: 0.6451 - val_accuracy: 0.6421
Epoch 87/200
62/62 [====] - 5s 74ms/step - loss: 0.6563 - accuracy: 0.6563 - val_loss: 0.7676 - val_accuracy: 0.6155
Epoch 88/200
62/62 [====] - 5s 75ms/step - loss: 0.6290 - accuracy: 0.6737 - val_loss: 0.6842 - val_accuracy: 0.5992
Epoch 89/200
62/62 [====] - 5s 74ms/step - loss: 0.6182 - accuracy: 0.6685 - val_loss: 0.7365 - val_accuracy: 0.6258
Epoch 90/200
62/62 [====] - 5s 74ms/step - loss: 0.6189 - accuracy: 0.6808 - val_loss: 0.6574 - val_accuracy: 0.6646
Epoch 91/200
62/62 [====] - 5s 74ms/step - loss: 0.6243 - accuracy: 0.6864 - val_loss: 0.6564 - val_accuracy: 0.6462
Epoch 92/200
62/62 [====] - 5s 75ms/step - loss: 0.6137 - accuracy: 0.6992 - val_loss: 0.6229 - val_accuracy: 0.6871
Epoch 93/200
62/62 [====] - 5s 75ms/step - loss: 0.6012 - accuracy: 0.6777 - val_loss: 0.6456 - val_accuracy: 0.6033
Epoch 94/200
62/62 [====] - 5s 75ms/step - loss: 0.6046 - accuracy: 0.6870 - val_loss: 0.6785 - val_accuracy: 0.6646
Epoch 95/200
62/62 [====] - 5s 74ms/step - loss: 0.5951 - accuracy: 0.6957 - val_loss: 0.7675 - val_accuracy: 0.6687
Epoch 96/200
62/62 [====] - 5s 74ms/step - loss: 0.5988 - accuracy: 0.7028 - val_loss: 0.6199 - val_accuracy: 0.6851
Epoch 97/200
62/62 [====] - 5s 74ms/step - loss: 0.6430 - accuracy: 0.6777 - val_loss: 0.7498 - val_accuracy: 0.6094
Epoch 98/200
62/62 [====] - 5s 75ms/step - loss: 0.5963 - accuracy: 0.6972 - val_loss: 0.7083 - val_accuracy: 0.6626
Epoch 99/200
```

```
62/62 [====] - 5s 74ms/step - loss: 0.6007 - accuracy: 0.6803 - val_loss: 0.6854 - val_accuracy: 0.5971
Epoch 100/200
62/62 [====] - 5s 75ms/step - loss: 0.6046 - accuracy: 0.6962 - val_loss: 0.6493 - val_accuracy: 0.6687
Epoch 101/200
62/62 [====] - 5s 75ms/step - loss: 0.6246 - accuracy: 0.6706 - val_loss: 0.6690 - val_accuracy: 0.6810
Epoch 102/200
62/62 [====] - 5s 75ms/step - loss: 0.6012 - accuracy: 0.6875 - val_loss: 0.7524 - val_accuracy: 0.6380
Epoch 103/200
62/62 [====] - 5s 75ms/step - loss: 0.6688 - accuracy: 0.6839 - val_loss: 0.6517 - val_accuracy: 0.6728
Epoch 104/200
62/62 [====] - 5s 73ms/step - loss: 0.5886 - accuracy: 0.6880 - val_loss: 0.6592 - val_accuracy: 0.6626
Epoch 105/200
62/62 [====] - 5s 74ms/step - loss: 0.6072 - accuracy: 0.7038 - val_loss: 0.7060 - val_accuracy: 0.6626
Epoch 106/200
62/62 [====] - 5s 74ms/step - loss: 0.5949 - accuracy: 0.7028 - val_loss: 0.6794 - val_accuracy: 0.6933
Epoch 107/200
62/62 [====] - 5s 75ms/step - loss: 0.6004 - accuracy: 0.6936 - val_loss: 0.6683 - val_accuracy: 0.7055
Epoch 108/200
62/62 [====] - 5s 75ms/step - loss: 0.5838 - accuracy: 0.7069 - val_loss: 0.6106 - val_accuracy: 0.7157
Epoch 109/200
62/62 [====] - 5s 76ms/step - loss: 0.5908 - accuracy: 0.6905 - val_loss: 0.6320 - val_accuracy: 0.6667
Epoch 110/200
62/62 [====] - 5s 75ms/step - loss: 0.6112 - accuracy: 0.7028 - val_loss: 0.6787 - val_accuracy: 0.7014
Epoch 111/200
62/62 [====] - 5s 75ms/step - loss: 0.6117 - accuracy: 0.6921 - val_loss: 0.6198 - val_accuracy: 0.6851
Epoch 112/200
62/62 [====] - 5s 75ms/step - loss: 0.5928 - accuracy: 0.6905 - val_loss: 0.6544 - val_accuracy: 0.6585
Epoch 113/200
62/62 [====] - 5s 75ms/step - loss: 0.6033 - accuracy: 0.6946 - val_loss: 0.7539 - val_accuracy: 0.6524
Epoch 114/200
62/62 [====] - 5s 75ms/step - loss: 0.5844 - accuracy: 0.6992 - val_loss: 0.7110 - val_accuracy: 0.6851
Epoch 115/200
62/62 [====] - 5s 75ms/step - loss: 0.7432 - accuracy: 0.6977 - val_loss: 0.6405 - val_accuracy: 0.7076
Epoch 116/200
62/62 [====] - 5s 76ms/step - loss: 0.5872 - accuracy: 0.7105 - val_loss: 0.6932 - val_accuracy: 0.7035
Epoch 117/200
62/62 [====] - 5s 75ms/step - loss: 0.5828 - accuracy: 0.6818 - val_loss: 0.8023 - val_accuracy: 0.5910
Epoch 118/200
62/62 [====] - 5s 75ms/step - loss: 0.6133 - accuracy: 0.6844 - val_loss: 0.6682 - val_accuracy: 0.6544
Epoch 119/200
62/62 [====] - 5s 75ms/step - loss: 0.5801 - accuracy: 0.7069 - val_loss: 0.6278 - val_accuracy: 0.6421
Epoch 120/200
62/62 [====] - 5s 75ms/step - loss: 0.6327 - accuracy: 0.6926 - val_loss: 0.5956 - val_accuracy: 0.7239
Epoch 121/200
62/62 [====] - 5s 75ms/step - loss: 0.5759 - accuracy: 0.7146 - val_loss: 1.4072 - val_accuracy: 0.6912
Epoch 122/200
62/62 [====] - 5s 77ms/step - loss: 0.6030 - accuracy: 0.7110 - val_loss: 0.7011 - val_accuracy: 0.6564
Epoch 123/200
62/62 [====] - 5s 75ms/step - loss: 0.5947 - accuracy: 0.7013 - val_loss: 0.6557 - val_accuracy: 0.6360
Epoch 124/200
62/62 [====] - 5s 76ms/step - loss: 0.6259 - accuracy: 0.6880 - val_loss: 0.6563 - val_accuracy: 0.6871
Epoch 125/200
62/62 [====] - 5s 74ms/step - loss: 0.6247 - accuracy: 0.7161 - val_loss: 0.6842 - val_accuracy: 0.6442
Epoch 126/200
62/62 [====] - 5s 75ms/step - loss: 0.5838 - accuracy: 0.7120 - val_loss: 0.7124 - val_accuracy: 0.6033
Epoch 127/200
62/62 [====] - 5s 74ms/step - loss: 0.6740 - accuracy: 0.7151 - val_loss: 0.7412 - val_accuracy: 0.6401
Epoch 128/200
62/62 [====] - 5s 74ms/step - loss: 0.6292 - accuracy: 0.6870 - val_loss: 0.7914 - val_accuracy: 0.5583
Epoch 129/200
62/62 [====] - 5s 75ms/step - loss: 0.6072 - accuracy: 0.6859 - val_loss: 0.6753 - val_accuracy: 0.5849
Epoch 130/200
62/62 [====] - 5s 75ms/step - loss: 0.6226 - accuracy: 0.6716 - val_loss: 0.6652 - val_accuracy: 0.6851
Epoch 131/200
62/62 [====] - 5s 74ms/step - loss: 0.5946 - accuracy: 0.6987 - val_loss: 0.7404 - val_accuracy: 0.6605
Epoch 132/200
62/62 [====] - 5s 73ms/step - loss: 0.5727 - accuracy: 0.7233 - val_loss: 0.8313 - val_accuracy: 0.6748
Epoch 133/200
62/62 [====] - 5s 75ms/step - loss: 0.5910 - accuracy: 0.7064 - val_loss: 0.6269 - val_accuracy: 0.6953
Epoch 134/200
62/62 [====] - 5s 73ms/step - loss: 0.5881 - accuracy: 0.6905 - val_loss: 0.6774 - val_accuracy: 0.7260
Epoch 135/200
62/62 [====] - 5s 75ms/step - loss: 0.5862 - accuracy: 0.7309 - val_loss: 0.6430 - val_accuracy: 0.6626
Epoch 136/200
62/62 [====] - 5s 76ms/step - loss: 0.5968 - accuracy: 0.6905 - val_loss: 0.8684 - val_accuracy: 0.6851
Epoch 137/200
62/62 [====] - 5s 75ms/step - loss: 0.5961 - accuracy: 0.7033 - val_loss: 0.6822 - val_accuracy: 0.6892
Epoch 138/200
62/62 [====] - 5s 75ms/step - loss: 0.6411 - accuracy: 0.6977 - val_loss: 0.6408 - val_accuracy: 0.6626
Epoch 139/200
62/62 [====] - 5s 75ms/step - loss: 0.6059 - accuracy: 0.6921 - val_loss: 0.6251 - val_accuracy: 0.6728
Epoch 140/200
```

```
62/62 [====] - 5s 75ms/step - loss: 0.5887 - accuracy: 0.7013 - val_loss: 0.7197 - val_accuracy: 0.6994
Epoch 141/200
62/62 [====] - 5s 75ms/step - loss: 0.6147 - accuracy: 0.6987 - val_loss: 0.6162 - val_accuracy: 0.7219
Epoch 142/200
62/62 [====] - 5s 75ms/step - loss: 0.6126 - accuracy: 0.7028 - val_loss: 0.6658 - val_accuracy: 0.6626
Epoch 143/200
62/62 [====] - 5s 75ms/step - loss: 0.5760 - accuracy: 0.7084 - val_loss: 0.6468 - val_accuracy: 0.6953
Epoch 144/200
62/62 [====] - 5s 75ms/step - loss: 0.5988 - accuracy: 0.7130 - val_loss: 0.6363 - val_accuracy: 0.6851
Epoch 145/200
62/62 [====] - 5s 75ms/step - loss: 0.6094 - accuracy: 0.7182 - val_loss: 0.6436 - val_accuracy: 0.6994
Epoch 146/200
62/62 [====] - 5s 74ms/step - loss: 0.5947 - accuracy: 0.7008 - val_loss: 0.6112 - val_accuracy: 0.6626
Epoch 147/200
62/62 [====] - 5s 74ms/step - loss: 0.5997 - accuracy: 0.7156 - val_loss: 0.6782 - val_accuracy: 0.6728
Epoch 148/200
62/62 [====] - 5s 82ms/step - loss: 0.5639 - accuracy: 0.7228 - val_loss: 0.6132 - val_accuracy: 0.6728
Epoch 149/200
62/62 [====] - 5s 74ms/step - loss: 0.5895 - accuracy: 0.6931 - val_loss: 0.8241 - val_accuracy: 0.6830
Epoch 150/200
62/62 [====] - 5s 76ms/step - loss: 0.6128 - accuracy: 0.6982 - val_loss: 0.6568 - val_accuracy: 0.6483
Epoch 151/200
62/62 [====] - 5s 76ms/step - loss: 0.6058 - accuracy: 0.6941 - val_loss: 0.7276 - val_accuracy: 0.6074
Epoch 152/200
62/62 [====] - 5s 75ms/step - loss: 0.5846 - accuracy: 0.6992 - val_loss: 1.0135 - val_accuracy: 0.6585
Epoch 153/200
62/62 [====] - 5s 75ms/step - loss: 0.5970 - accuracy: 0.7033 - val_loss: 0.7378 - val_accuracy: 0.6708
Epoch 154/200
62/62 [====] - 5s 76ms/step - loss: 0.6415 - accuracy: 0.7095 - val_loss: 0.7312 - val_accuracy: 0.6380
Epoch 155/200
62/62 [====] - 5s 74ms/step - loss: 0.5855 - accuracy: 0.7095 - val_loss: 0.6729 - val_accuracy: 0.6830
Epoch 156/200
62/62 [====] - 5s 75ms/step - loss: 0.5786 - accuracy: 0.7141 - val_loss: 0.6728 - val_accuracy: 0.7035
Epoch 157/200
62/62 [====] - 5s 76ms/step - loss: 0.5825 - accuracy: 0.7079 - val_loss: 0.7212 - val_accuracy: 0.6769
Epoch 158/200
62/62 [====] - 5s 77ms/step - loss: 0.6201 - accuracy: 0.6721 - val_loss: 0.6906 - val_accuracy: 0.5378
Epoch 159/200
62/62 [====] - 5s 75ms/step - loss: 0.5906 - accuracy: 0.7120 - val_loss: 0.6488 - val_accuracy: 0.6892
Epoch 160/200
62/62 [====] - 5s 75ms/step - loss: 0.6026 - accuracy: 0.7146 - val_loss: 0.7567 - val_accuracy: 0.6687
Epoch 161/200
62/62 [====] - 5s 75ms/step - loss: 0.5497 - accuracy: 0.7299 - val_loss: 0.6108 - val_accuracy: 0.6851
Epoch 162/200
62/62 [====] - 5s 75ms/step - loss: 0.5746 - accuracy: 0.7187 - val_loss: 1.1033 - val_accuracy: 0.6708
Epoch 163/200
62/62 [====] - 5s 74ms/step - loss: 0.5917 - accuracy: 0.7284 - val_loss: 0.7315 - val_accuracy: 0.7014
Epoch 164/200
62/62 [====] - 5s 76ms/step - loss: 0.5898 - accuracy: 0.7182 - val_loss: 0.8233 - val_accuracy: 0.6892
Epoch 165/200
62/62 [====] - 5s 75ms/step - loss: 0.5701 - accuracy: 0.7217 - val_loss: 0.6459 - val_accuracy: 0.5910
Epoch 166/200
62/62 [====] - 5s 75ms/step - loss: 0.5994 - accuracy: 0.7136 - val_loss: 0.6524 - val_accuracy: 0.7076
Epoch 167/200
62/62 [====] - 5s 76ms/step - loss: 0.6020 - accuracy: 0.7156 - val_loss: 0.7996 - val_accuracy: 0.6810
Epoch 168/200
62/62 [====] - 5s 75ms/step - loss: 0.5762 - accuracy: 0.7192 - val_loss: 0.6951 - val_accuracy: 0.6585
Epoch 169/200
62/62 [====] - 5s 75ms/step - loss: 0.5856 - accuracy: 0.7253 - val_loss: 0.6894 - val_accuracy: 0.6769
Epoch 170/200
62/62 [====] - 5s 75ms/step - loss: 0.5956 - accuracy: 0.7146 - val_loss: 0.6658 - val_accuracy: 0.6564
Epoch 171/200
62/62 [====] - 5s 76ms/step - loss: 0.5455 - accuracy: 0.7146 - val_loss: 0.7179 - val_accuracy: 0.7055
Epoch 172/200
62/62 [====] - 5s 75ms/step - loss: 0.5932 - accuracy: 0.6982 - val_loss: 0.7521 - val_accuracy: 0.6851
Epoch 173/200
62/62 [====] - 5s 77ms/step - loss: 0.5851 - accuracy: 0.7386 - val_loss: 0.8371 - val_accuracy: 0.6585
Epoch 174/200
62/62 [====] - 5s 76ms/step - loss: 0.5590 - accuracy: 0.7217 - val_loss: 0.7563 - val_accuracy: 0.6728
Epoch 175/200
62/62 [====] - 5s 75ms/step - loss: 0.5377 - accuracy: 0.7391 - val_loss: 0.8743 - val_accuracy: 0.6462
Epoch 176/200
62/62 [====] - 5s 74ms/step - loss: 0.5650 - accuracy: 0.7361 - val_loss: 0.6887 - val_accuracy: 0.6135
Epoch 177/200
62/62 [====] - 5s 75ms/step - loss: 0.5580 - accuracy: 0.7361 - val_loss: 0.6483 - val_accuracy: 0.6544
Epoch 178/200
62/62 [====] - 5s 77ms/step - loss: 0.5978 - accuracy: 0.7320 - val_loss: 0.7644 - val_accuracy: 0.6585
Epoch 179/200
62/62 [====] - 5s 74ms/step - loss: 0.6168 - accuracy: 0.7192 - val_loss: 0.6017 - val_accuracy: 0.6789
Epoch 180/200
62/62 [====] - 5s 75ms/step - loss: 0.5783 - accuracy: 0.7202 - val_loss: 0.6333 - val_accuracy: 0.7076
Epoch 181/200
```

```
62/62 [====] - 5s 74ms/step - loss: 0.5696 - accuracy: 0.7248 - val_loss: 0.8090 - val_accuracy: 0.6053
Epoch 182/200
62/62 [====] - 5s 75ms/step - loss: 0.5787 - accuracy: 0.7279 - val_loss: 0.7552 - val_accuracy: 0.6892
Epoch 183/200
62/62 [====] - 5s 74ms/step - loss: 0.5852 - accuracy: 0.7345 - val_loss: 0.8669 - val_accuracy: 0.7014
Epoch 184/200
62/62 [====] - 5s 75ms/step - loss: 0.5965 - accuracy: 0.7325 - val_loss: 0.7436 - val_accuracy: 0.6115
Epoch 185/200
62/62 [====] - 5s 76ms/step - loss: 0.5672 - accuracy: 0.7253 - val_loss: 0.5794 - val_accuracy: 0.7198
Epoch 186/200
62/62 [====] - 5s 75ms/step - loss: 0.6123 - accuracy: 0.7182 - val_loss: 0.6369 - val_accuracy: 0.6278
Epoch 187/200
62/62 [====] - 5s 75ms/step - loss: 0.5606 - accuracy: 0.7294 - val_loss: 0.6850 - val_accuracy: 0.7096
Epoch 188/200
62/62 [====] - 5s 76ms/step - loss: 0.5774 - accuracy: 0.7386 - val_loss: 0.5928 - val_accuracy: 0.6789
Epoch 189/200
62/62 [====] - 5s 75ms/step - loss: 0.5668 - accuracy: 0.7417 - val_loss: 0.5989 - val_accuracy: 0.6810
Epoch 190/200
62/62 [====] - 5s 75ms/step - loss: 0.5825 - accuracy: 0.7130 - val_loss: 0.6117 - val_accuracy: 0.7178
Epoch 191/200
62/62 [====] - 5s 76ms/step - loss: 0.5332 - accuracy: 0.7396 - val_loss: 0.7109 - val_accuracy: 0.6871
Epoch 192/200
62/62 [====] - 5s 76ms/step - loss: 0.5287 - accuracy: 0.7432 - val_loss: 1.3313 - val_accuracy: 0.6810
Epoch 193/200
62/62 [====] - 5s 76ms/step - loss: 0.5629 - accuracy: 0.7355 - val_loss: 0.8629 - val_accuracy: 0.6319
Epoch 194/200
62/62 [====] - 5s 76ms/step - loss: 0.5516 - accuracy: 0.7366 - val_loss: 0.6346 - val_accuracy: 0.6933
Epoch 195/200
62/62 [====] - 5s 76ms/step - loss: 0.5686 - accuracy: 0.7361 - val_loss: 0.6801 - val_accuracy: 0.6912
Epoch 196/200
62/62 [====] - 5s 75ms/step - loss: 0.5483 - accuracy: 0.7468 - val_loss: 0.7198 - val_accuracy: 0.7198
Epoch 197/200
62/62 [====] - 5s 76ms/step - loss: 0.5598 - accuracy: 0.7187 - val_loss: 0.6654 - val_accuracy: 0.7014
Epoch 198/200
62/62 [====] - 5s 76ms/step - loss: 0.6022 - accuracy: 0.7340 - val_loss: 0.8004 - val_accuracy: 0.6503
Epoch 199/200
62/62 [====] - 5s 78ms/step - loss: 0.5730 - accuracy: 0.7192 - val_loss: 0.7973 - val_accuracy: 0.7096
Epoch 200/200
62/62 [====] - 5s 76ms/step - loss: 0.6111 - accuracy: 0.7294 - val_loss: 0.7991 - val_accuracy: 0.7566
```

Experiment 13

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_56 (Conv2D)          (None, 62, 62, 32)        320
 max_pooling2d_56 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_57 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_57 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_58 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_58 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_59 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_59 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_12 (Flatten)        (None, 256)               0
 dense_24 (Dense)            (None, 64)                16448
 dropout_12 (Dropout)        (None, 64)                0
 dense_25 (Dense)            (None, 2)                 130
=================================================================
Total params: 53,890
Trainable params: 53,890
Non-trainable params: 0
_________________________________________________________________
Epoch 1/35
62/62 [====] - 2s 19ms/step - loss: 0.6946 - accuracy: 0.4880 - val_loss: 0.6931 - val_accuracy: 0.4990
Epoch 2/35
62/62 [====] - 1s 15ms/step - loss: 0.6933 - accuracy: 0.4967 - val_loss: 0.6930 - val_accuracy: 0.5112
Epoch 3/35
62/62 [====] - 1s 16ms/step - loss: 0.6931 - accuracy: 0.4962 - val_loss: 0.6922 - val_accuracy: 0.5072
Epoch 4/35
62/62 [====] - 1s 15ms/step - loss: 0.6918 - accuracy: 0.5161 - val_loss: 0.6993 - val_accuracy: 0.5010
Epoch 5/35
62/62 [====] - 1s 16ms/step - loss: 0.6894 - accuracy: 0.5248 - val_loss: 0.6943 - val_accuracy: 0.5153
Epoch 6/35
```

```
62/62 [====] - 1s 16ms/step - loss: 0.6909 - accuracy: 0.5136 - val_loss: 0.6889 - val_accuracy: 0.5930
Epoch 7/35
62/62 [====] - 1s 16ms/step - loss: 0.6876 - accuracy: 0.5376 - val_loss: 0.7049 - val_accuracy: 0.5153
Epoch 8/35
62/62 [====] - 1s 16ms/step - loss: 0.6862 - accuracy: 0.5407 - val_loss: 0.6977 - val_accuracy: 0.5051
Epoch 9/35
62/62 [====] - 1s 15ms/step - loss: 0.6830 - accuracy: 0.5581 - val_loss: 0.7008 - val_accuracy: 0.5153
Epoch 10/35
62/62 [====] - 1s 15ms/step - loss: 0.6802 - accuracy: 0.5529 - val_loss: 0.6953 - val_accuracy: 0.5092
Epoch 11/35
62/62 [====] - 1s 14ms/step - loss: 0.6733 - accuracy: 0.5836 - val_loss: 0.7695 - val_accuracy: 0.5235
Epoch 12/35
62/62 [====] - 1s 16ms/step - loss: 0.6685 - accuracy: 0.5969 - val_loss: 0.7131 - val_accuracy: 0.5133
Epoch 13/35
62/62 [====] - 1s 16ms/step - loss: 0.6567 - accuracy: 0.6092 - val_loss: 0.7022 - val_accuracy: 0.5317
Epoch 14/35
62/62 [====] - 1s 16ms/step - loss: 0.6422 - accuracy: 0.6409 - val_loss: 0.7829 - val_accuracy: 0.5051
Epoch 15/35
62/62 [====] - 1s 16ms/step - loss: 0.6342 - accuracy: 0.6578 - val_loss: 0.6429 - val_accuracy: 0.6769
Epoch 16/35
62/62 [====] - 1s 15ms/step - loss: 0.6254 - accuracy: 0.6537 - val_loss: 0.8283 - val_accuracy: 0.5133
Epoch 17/35
62/62 [====] - 1s 16ms/step - loss: 0.6211 - accuracy: 0.6691 - val_loss: 0.7939 - val_accuracy: 0.5235
Epoch 18/35
62/62 [====] - 1s 14ms/step - loss: 0.5942 - accuracy: 0.6854 - val_loss: 0.7040 - val_accuracy: 0.5644
Epoch 19/35
62/62 [====] - 1s 15ms/step - loss: 0.5950 - accuracy: 0.6818 - val_loss: 0.7612 - val_accuracy: 0.5399
Epoch 20/35
62/62 [====] - 1s 15ms/step - loss: 0.5694 - accuracy: 0.7064 - val_loss: 0.8011 - val_accuracy: 0.5767
Epoch 21/35
62/62 [====] - 1s 14ms/step - loss: 0.5571 - accuracy: 0.7161 - val_loss: 0.5988 - val_accuracy: 0.6667
Epoch 22/35
62/62 [====] - 1s 14ms/step - loss: 0.5521 - accuracy: 0.7120 - val_loss: 0.6628 - val_accuracy: 0.5828
Epoch 23/35
62/62 [====] - 1s 15ms/step - loss: 0.5267 - accuracy: 0.7391 - val_loss: 0.6297 - val_accuracy: 0.6544
Epoch 24/35
62/62 [====] - 1s 15ms/step - loss: 0.5234 - accuracy: 0.7391 - val_loss: 0.8636 - val_accuracy: 0.5337
Epoch 25/35
62/62 [====] - 1s 14ms/step - loss: 0.4958 - accuracy: 0.7509 - val_loss: 0.6220 - val_accuracy: 0.6605
Epoch 26/35
62/62 [====] - 1s 15ms/step - loss: 0.4846 - accuracy: 0.7719 - val_loss: 0.7279 - val_accuracy: 0.6155
Epoch 27/35
62/62 [====] - 1s 15ms/step - loss: 0.4812 - accuracy: 0.7662 - val_loss: 0.6825 - val_accuracy: 0.6605
Epoch 28/35
62/62 [====] - 1s 16ms/step - loss: 0.4699 - accuracy: 0.7744 - val_loss: 0.7040 - val_accuracy: 0.6278
Epoch 29/35
62/62 [====] - 1s 15ms/step - loss: 0.4389 - accuracy: 0.7969 - val_loss: 0.7918 - val_accuracy: 0.5971
Epoch 30/35
62/62 [====] - 1s 14ms/step - loss: 0.4288 - accuracy: 0.7944 - val_loss: 0.7232 - val_accuracy: 0.6524
Epoch 31/35
62/62 [====] - 1s 15ms/step - loss: 0.4130 - accuracy: 0.8026 - val_loss: 0.9523 - val_accuracy: 0.6094
Epoch 32/35
62/62 [====] - 1s 16ms/step - loss: 0.4051 - accuracy: 0.8092 - val_loss: 0.8183 - val_accuracy: 0.6339
Epoch 33/35
62/62 [====] - 1s 15ms/step - loss: 0.3908 - accuracy: 0.8240 - val_loss: 0.7124 - val_accuracy: 0.6462
Epoch 34/35
62/62 [====] - 1s 16ms/step - loss: 0.3686 - accuracy: 0.8205 - val_loss: 0.8124 - val_accuracy: 0.6176
Epoch 35/35
62/62 [====] - 1s 16ms/step - loss: 0.3395 - accuracy: 0.8445 - val_loss: 0.8323 - val_accuracy: 0.6462
```

Experiment 14

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_60 (Conv2D)          (None, 62, 62, 32)        320
 max_pooling2d_60 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_61 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_61 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_62 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_62 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_63 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_63 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_13 (Flatten)        (None, 256)               0
 dense_26 (Dense)            (None, 64)                16448
 dropout_13 (Dropout)        (None, 64)                0
 dense_27 (Dense)            (None, 2)                 130
=================================================================
Total params: 53,890
Trainable params: 53,890
Non-trainable params: 0
_________________________________________________________________
Epoch 1/400
62/62 [====] - 4s 44ms/step - loss: 1.1421 - accuracy: 0.4900 - val_loss: 0.7134 - val_accuracy: 0.5031
Epoch 2/400
62/62 [====] - 3s 41ms/step - loss: 0.7251 - accuracy: 0.4946 - val_loss: 0.6939 - val_accuracy: 0.5215
Epoch 3/400
62/62 [====] - 3s 41ms/step - loss: 0.7037 - accuracy: 0.4946 - val_loss: 0.6917 - val_accuracy: 0.5194
Epoch 4/400
62/62 [====] - 3s 41ms/step - loss: 0.6973 - accuracy: 0.5064 - val_loss: 0.6966 - val_accuracy: 0.4785
Epoch 5/400
62/62 [====] - 3s 41ms/step - loss: 0.7104 - accuracy: 0.5238 - val_loss: 0.6945 - val_accuracy: 0.4990
Epoch 6/400
62/62 [====] - 3s 41ms/step - loss: 0.6961 - accuracy: 0.5207 - val_loss: 0.6940 - val_accuracy: 0.5256
Epoch 7/400
62/62 [====] - 3s 40ms/step - loss: 0.7006 - accuracy: 0.5182 - val_loss: 0.7014 - val_accuracy: 0.4724
Epoch 8/400
62/62 [====] - 3s 41ms/step - loss: 0.7006 - accuracy: 0.4982 - val_loss: 0.6926 - val_accuracy: 0.5072
Epoch 9/400
62/62 [====] - 3s 45ms/step - loss: 0.6987 - accuracy: 0.5054 - val_loss: 0.6950 - val_accuracy: 0.4990
Epoch 10/400
62/62 [====] - 3s 41ms/step - loss: 0.6995 - accuracy: 0.5110 - val_loss: 0.6939 - val_accuracy: 0.5112
Epoch 11/400
62/62 [====] - 3s 41ms/step - loss: 0.6925 - accuracy: 0.5253 - val_loss: 0.6939 - val_accuracy: 0.5194
Epoch 12/400
62/62 [====] - 3s 41ms/step - loss: 0.6961 - accuracy: 0.5248 - val_loss: 0.6929 - val_accuracy: 0.5092
Epoch 13/400
62/62 [====] - 3s 41ms/step - loss: 0.6900 - accuracy: 0.5366 - val_loss: 0.6974 - val_accuracy: 0.4928
Epoch 14/400
62/62 [====] - 3s 42ms/step - loss: 0.6892 - accuracy: 0.5340 - val_loss: 0.6903 - val_accuracy: 0.5215
Epoch 15/400
62/62 [====] - 3s 41ms/step - loss: 0.6919 - accuracy: 0.5304 - val_loss: 0.6893 - val_accuracy: 0.5297
Epoch 16/400
62/62 [====] - 3s 41ms/step - loss: 0.7077 - accuracy: 0.5151 - val_loss: 0.6804 - val_accuracy: 0.5603
Epoch 17/400
62/62 [====] - 3s 41ms/step - loss: 0.6896 - accuracy: 0.5289 - val_loss: 0.6860 - val_accuracy: 0.5235
Epoch 18/400
62/62 [====] - 3s 40ms/step - loss: 0.6920 - accuracy: 0.5274 - val_loss: 0.6918 - val_accuracy: 0.4949
Epoch 19/400
62/62 [====] - 3s 41ms/step - loss: 0.6869 - accuracy: 0.5274 - val_loss: 0.6853 - val_accuracy: 0.5706
Epoch 20/400
62/62 [====] - 3s 41ms/step - loss: 0.6843 - accuracy: 0.5478 - val_loss: 0.6970 - val_accuracy: 0.5031
Epoch 21/400
62/62 [====] - 3s 41ms/step - loss: 0.6892 - accuracy: 0.5381 - val_loss: 0.6917 - val_accuracy: 0.5174
Epoch 22/400
62/62 [====] - 3s 40ms/step - loss: 0.6878 - accuracy: 0.5407 - val_loss: 0.6950 - val_accuracy: 0.5010
Epoch 23/400
62/62 [====] - 3s 41ms/step - loss: 0.6893 - accuracy: 0.5350 - val_loss: 0.6873 - val_accuracy: 0.5440
Epoch 24/400
62/62 [====] - 3s 41ms/step - loss: 0.6824 - accuracy: 0.5693 - val_loss: 0.6919 - val_accuracy: 0.5583
Epoch 25/400
62/62 [====] - 3s 41ms/step - loss: 0.6828 - accuracy: 0.5555 - val_loss: 0.7094 - val_accuracy: 0.5215
Epoch 26/400
62/62 [====] - 3s 40ms/step - loss: 0.6897 - accuracy: 0.5402 - val_loss: 0.6873 - val_accuracy: 0.5603
Epoch 27/400
62/62 [====] - 3s 41ms/step - loss: 0.6827 - accuracy: 0.5560 - val_loss: 0.6755 - val_accuracy: 0.5297
Epoch 28/400
62/62 [====] - 3s 42ms/step - loss: 0.6839 - accuracy: 0.5514 - val_loss: 0.6791 - val_accuracy: 0.5562
```

```
Epoch 29/400
62/62 [====] - 3s 41ms/step - loss: 0.6805 - accuracy: 0.5535 - val_loss: 0.6842 - val_accuracy: 0.5624
Epoch 30/400
62/62 [====] - 3s 41ms/step - loss: 0.6787 - accuracy: 0.5693 - val_loss: 0.7199 - val_accuracy: 0.5194
Epoch 31/400
62/62 [====] - 3s 41ms/step - loss: 0.6885 - accuracy: 0.5596 - val_loss: 0.6820 - val_accuracy: 0.5603
Epoch 32/400
62/62 [====] - 3s 41ms/step - loss: 0.6793 - accuracy: 0.5780 - val_loss: 0.6817 - val_accuracy: 0.5726
Epoch 33/400
62/62 [====] - 3s 41ms/step - loss: 0.6768 - accuracy: 0.5821 - val_loss: 0.6813 - val_accuracy: 0.6135
Epoch 34/400
62/62 [====] - 3s 45ms/step - loss: 0.6735 - accuracy: 0.5801 - val_loss: 0.6758 - val_accuracy: 0.5419
Epoch 35/400
62/62 [====] - 3s 41ms/step - loss: 0.6814 - accuracy: 0.5780 - val_loss: 0.6752 - val_accuracy: 0.5951
Epoch 36/400
62/62 [====] - 3s 41ms/step - loss: 0.6731 - accuracy: 0.5974 - val_loss: 0.6679 - val_accuracy: 0.5971
Epoch 37/400
62/62 [====] - 3s 41ms/step - loss: 0.6599 - accuracy: 0.6159 - val_loss: 0.6795 - val_accuracy: 0.5665
Epoch 38/400
62/62 [====] - 3s 41ms/step - loss: 0.6689 - accuracy: 0.6010 - val_loss: 0.6677 - val_accuracy: 0.5828
Epoch 39/400
62/62 [====] - 3s 41ms/step - loss: 0.6564 - accuracy: 0.6174 - val_loss: 0.6919 - val_accuracy: 0.5235
Epoch 40/400
62/62 [====] - 3s 41ms/step - loss: 0.6599 - accuracy: 0.6169 - val_loss: 0.6994 - val_accuracy: 0.6053
Epoch 41/400
62/62 [====] - 3s 42ms/step - loss: 0.6510 - accuracy: 0.6297 - val_loss: 0.6713 - val_accuracy: 0.6196
Epoch 42/400
62/62 [====] - 3s 41ms/step - loss: 0.6480 - accuracy: 0.6312 - val_loss: 0.6572 - val_accuracy: 0.6626
Epoch 43/400
62/62 [====] - 3s 41ms/step - loss: 0.6506 - accuracy: 0.6399 - val_loss: 0.6425 - val_accuracy: 0.6442
Epoch 44/400
62/62 [====] - 3s 41ms/step - loss: 0.6532 - accuracy: 0.6123 - val_loss: 0.6961 - val_accuracy: 0.6094
Epoch 45/400
62/62 [====] - 3s 41ms/step - loss: 0.6550 - accuracy: 0.6297 - val_loss: 0.6640 - val_accuracy: 0.5644
Epoch 46/400
62/62 [====] - 3s 41ms/step - loss: 0.6512 - accuracy: 0.6389 - val_loss: 0.6887 - val_accuracy: 0.6258
Epoch 47/400
62/62 [====] - 3s 41ms/step - loss: 0.6524 - accuracy: 0.6501 - val_loss: 0.6609 - val_accuracy: 0.6278
Epoch 48/400
62/62 [====] - 3s 41ms/step - loss: 0.6465 - accuracy: 0.6558 - val_loss: 0.6476 - val_accuracy: 0.6585
Epoch 49/400
62/62 [====] - 3s 41ms/step - loss: 0.6365 - accuracy: 0.6379 - val_loss: 0.7035 - val_accuracy: 0.5583
Epoch 50/400
62/62 [====] - 3s 41ms/step - loss: 0.6328 - accuracy: 0.6634 - val_loss: 0.6990 - val_accuracy: 0.6094
Epoch 51/400
62/62 [====] - 3s 41ms/step - loss: 0.6435 - accuracy: 0.6532 - val_loss: 0.6801 - val_accuracy: 0.5951
Epoch 52/400
62/62 [====] - 3s 41ms/step - loss: 0.6316 - accuracy: 0.6721 - val_loss: 0.7024 - val_accuracy: 0.5501
Epoch 53/400
62/62 [====] - 3s 42ms/step - loss: 0.6305 - accuracy: 0.6650 - val_loss: 0.6647 - val_accuracy: 0.6115
Epoch 54/400
62/62 [====] - 3s 41ms/step - loss: 0.6221 - accuracy: 0.6578 - val_loss: 0.6678 - val_accuracy: 0.6135
Epoch 55/400
62/62 [====] - 3s 42ms/step - loss: 0.6255 - accuracy: 0.6639 - val_loss: 0.7192 - val_accuracy: 0.5849
Epoch 56/400
62/62 [====] - 3s 40ms/step - loss: 0.6259 - accuracy: 0.6685 - val_loss: 0.6611 - val_accuracy: 0.6810
Epoch 57/400
62/62 [====] - 3s 41ms/step - loss: 0.6452 - accuracy: 0.6547 - val_loss: 0.6510 - val_accuracy: 0.6483
Epoch 58/400
62/62 [====] - 3s 41ms/step - loss: 0.6182 - accuracy: 0.6629 - val_loss: 0.6361 - val_accuracy: 0.6605
Epoch 59/400
62/62 [====] - 3s 41ms/step - loss: 0.6166 - accuracy: 0.6701 - val_loss: 0.6797 - val_accuracy: 0.6074
Epoch 60/400
62/62 [====] - 3s 42ms/step - loss: 0.6091 - accuracy: 0.6808 - val_loss: 0.6604 - val_accuracy: 0.6401
Epoch 61/400
62/62 [====] - 3s 42ms/step - loss: 0.6060 - accuracy: 0.6803 - val_loss: 0.6709 - val_accuracy: 0.6380
Epoch 62/400
62/62 [====] - 3s 41ms/step - loss: 0.6291 - accuracy: 0.6716 - val_loss: 0.6987 - val_accuracy: 0.6258
Epoch 63/400
62/62 [====] - 3s 41ms/step - loss: 0.6086 - accuracy: 0.6680 - val_loss: 0.6957 - val_accuracy: 0.6339
Epoch 64/400
62/62 [====] - 3s 42ms/step - loss: 0.6202 - accuracy: 0.6726 - val_loss: 0.6320 - val_accuracy: 0.6442
Epoch 65/400
62/62 [====] - 3s 41ms/step - loss: 0.6228 - accuracy: 0.6578 - val_loss: 0.6392 - val_accuracy: 0.6871
Epoch 66/400
62/62 [====] - 3s 41ms/step - loss: 0.6174 - accuracy: 0.6762 - val_loss: 0.6320 - val_accuracy: 0.6564
Epoch 67/400
62/62 [====] - 3s 42ms/step - loss: 0.6077 - accuracy: 0.7018 - val_loss: 0.6314 - val_accuracy: 0.6626
Epoch 68/400
62/62 [====] - 3s 41ms/step - loss: 0.5874 - accuracy: 0.6936 - val_loss: 0.6275 - val_accuracy: 0.6728
Epoch 69/400
62/62 [====] - 3s 41ms/step - loss: 0.5946 - accuracy: 0.6890 - val_loss: 0.6364 - val_accuracy: 0.6687
```

```
Epoch 70/400
62/62 [====] - 3s 41ms/step - loss: 0.6232 - accuracy: 0.6777 - val_loss: 0.6595 - val_accuracy: 0.6748
Epoch 71/400
62/62 [====] - 3s 41ms/step - loss: 0.5915 - accuracy: 0.6808 - val_loss: 0.7892 - val_accuracy: 0.6053
Epoch 72/400
62/62 [====] - 3s 41ms/step - loss: 0.6050 - accuracy: 0.6772 - val_loss: 0.6648 - val_accuracy: 0.5869
Epoch 73/400
62/62 [====] - 3s 41ms/step - loss: 0.5941 - accuracy: 0.6926 - val_loss: 0.7833 - val_accuracy: 0.6360
Epoch 74/400
62/62 [====] - 3s 41ms/step - loss: 0.6012 - accuracy: 0.6972 - val_loss: 0.6372 - val_accuracy: 0.6646
Epoch 75/400
62/62 [====] - 3s 41ms/step - loss: 0.5948 - accuracy: 0.6951 - val_loss: 0.6858 - val_accuracy: 0.5992
Epoch 76/400
62/62 [====] - 3s 41ms/step - loss: 0.6008 - accuracy: 0.6987 - val_loss: 0.6496 - val_accuracy: 0.7117
Epoch 77/400
62/62 [====] - 3s 41ms/step - loss: 0.5800 - accuracy: 0.7013 - val_loss: 0.6148 - val_accuracy: 0.6585
Epoch 78/400
62/62 [====] - 3s 42ms/step - loss: 0.5878 - accuracy: 0.7003 - val_loss: 0.6552 - val_accuracy: 0.6605
Epoch 79/400
62/62 [====] - 3s 41ms/step - loss: 0.5801 - accuracy: 0.7013 - val_loss: 0.6404 - val_accuracy: 0.6708
Epoch 80/400
62/62 [====] - 3s 41ms/step - loss: 0.5839 - accuracy: 0.6987 - val_loss: 0.6620 - val_accuracy: 0.6851
Epoch 81/400
62/62 [====] - 3s 41ms/step - loss: 0.5842 - accuracy: 0.7192 - val_loss: 0.6317 - val_accuracy: 0.6421
Epoch 82/400
62/62 [====] - 3s 41ms/step - loss: 0.5844 - accuracy: 0.6916 - val_loss: 0.7364 - val_accuracy: 0.6524
Epoch 83/400
62/62 [====] - 3s 41ms/step - loss: 0.5903 - accuracy: 0.6982 - val_loss: 0.6558 - val_accuracy: 0.6687
Epoch 84/400
62/62 [====] - 3s 41ms/step - loss: 0.5853 - accuracy: 0.7084 - val_loss: 0.6471 - val_accuracy: 0.6892
Epoch 85/400
62/62 [====] - 3s 41ms/step - loss: 0.5739 - accuracy: 0.7146 - val_loss: 0.7340 - val_accuracy: 0.6728
Epoch 86/400
62/62 [====] - 3s 41ms/step - loss: 0.5793 - accuracy: 0.7013 - val_loss: 0.6324 - val_accuracy: 0.6933
Epoch 87/400
62/62 [====] - 3s 41ms/step - loss: 0.5824 - accuracy: 0.7064 - val_loss: 0.7432 - val_accuracy: 0.6687
Epoch 88/400
62/62 [====] - 3s 42ms/step - loss: 0.5873 - accuracy: 0.6880 - val_loss: 0.6921 - val_accuracy: 0.6646
Epoch 89/400
62/62 [====] - 3s 41ms/step - loss: 0.5786 - accuracy: 0.7100 - val_loss: 0.7417 - val_accuracy: 0.6442
Epoch 90/400
62/62 [====] - 3s 42ms/step - loss: 0.5745 - accuracy: 0.7110 - val_loss: 0.5936 - val_accuracy: 0.6953
Epoch 91/400
62/62 [====] - 3s 42ms/step - loss: 0.5721 - accuracy: 0.7182 - val_loss: 0.7060 - val_accuracy: 0.6564
Epoch 92/400
62/62 [====] - 3s 41ms/step - loss: 0.5732 - accuracy: 0.7243 - val_loss: 0.7189 - val_accuracy: 0.6155
Epoch 93/400
62/62 [====] - 3s 41ms/step - loss: 0.5623 - accuracy: 0.7146 - val_loss: 0.6215 - val_accuracy: 0.6605
Epoch 94/400
62/62 [====] - 3s 42ms/step - loss: 0.5658 - accuracy: 0.7294 - val_loss: 0.6346 - val_accuracy: 0.6912
Epoch 95/400
62/62 [====] - 3s 41ms/step - loss: 0.5644 - accuracy: 0.7161 - val_loss: 0.6975 - val_accuracy: 0.6687
Epoch 96/400
62/62 [====] - 3s 41ms/step - loss: 0.5733 - accuracy: 0.7136 - val_loss: 0.6630 - val_accuracy: 0.7157
Epoch 97/400
62/62 [====] - 3s 43ms/step - loss: 0.5730 - accuracy: 0.6972 - val_loss: 0.7182 - val_accuracy: 0.6442
Epoch 98/400
62/62 [====] - 3s 42ms/step - loss: 0.5701 - accuracy: 0.7171 - val_loss: 0.6198 - val_accuracy: 0.6953
Epoch 99/400
62/62 [====] - 3s 41ms/step - loss: 0.5742 - accuracy: 0.7013 - val_loss: 0.6334 - val_accuracy: 0.7035
Epoch 100/400
62/62 [====] - 3s 41ms/step - loss: 0.5582 - accuracy: 0.7248 - val_loss: 0.6533 - val_accuracy: 0.7055
Epoch 101/400
62/62 [====] - 3s 42ms/step - loss: 0.5759 - accuracy: 0.7090 - val_loss: 0.7669 - val_accuracy: 0.6687
Epoch 102/400
62/62 [====] - 3s 42ms/step - loss: 0.5718 - accuracy: 0.7151 - val_loss: 0.7359 - val_accuracy: 0.6708
Epoch 103/400
62/62 [====] - 3s 42ms/step - loss: 0.5757 - accuracy: 0.7095 - val_loss: 0.6741 - val_accuracy: 0.6585
Epoch 104/400
62/62 [====] - 3s 42ms/step - loss: 0.5639 - accuracy: 0.7243 - val_loss: 0.7627 - val_accuracy: 0.6339
Epoch 105/400
62/62 [====] - 3s 42ms/step - loss: 0.5697 - accuracy: 0.7253 - val_loss: 0.6875 - val_accuracy: 0.6810
Epoch 106/400
62/62 [====] - 3s 41ms/step - loss: 0.5452 - accuracy: 0.7258 - val_loss: 0.7651 - val_accuracy: 0.6605
Epoch 107/400
62/62 [====] - 3s 41ms/step - loss: 0.5662 - accuracy: 0.7223 - val_loss: 0.6504 - val_accuracy: 0.6933
Epoch 108/400
62/62 [====] - 3s 41ms/step - loss: 0.5647 - accuracy: 0.7217 - val_loss: 0.6522 - val_accuracy: 0.6687
Epoch 109/400
62/62 [====] - 3s 41ms/step - loss: 0.5750 - accuracy: 0.7130 - val_loss: 0.7067 - val_accuracy: 0.6810
Epoch 110/400
62/62 [====] - 3s 41ms/step - loss: 0.5563 - accuracy: 0.7330 - val_loss: 0.6202 - val_accuracy: 0.6892
```

```
Epoch 111/400
62/62 [====] - 3s 42ms/step - loss: 0.5508 - accuracy: 0.7304 - val_loss: 0.7939 - val_accuracy: 0.6155
Epoch 112/400
62/62 [====] - 3s 42ms/step - loss: 0.5424 - accuracy: 0.7432 - val_loss: 0.6485 - val_accuracy: 0.7035
Epoch 113/400
62/62 [====] - 3s 41ms/step - loss: 0.5360 - accuracy: 0.7366 - val_loss: 0.7347 - val_accuracy: 0.6401
Epoch 114/400
62/62 [====] - 3s 43ms/step - loss: 0.5492 - accuracy: 0.7315 - val_loss: 0.7281 - val_accuracy: 0.6605
Epoch 115/400
62/62 [====] - 3s 41ms/step - loss: 0.5497 - accuracy: 0.7315 - val_loss: 0.6593 - val_accuracy: 0.7137
Epoch 116/400
62/62 [====] - 3s 42ms/step - loss: 0.5615 - accuracy: 0.7335 - val_loss: 0.6667 - val_accuracy: 0.6503
Epoch 117/400
62/62 [====] - 3s 42ms/step - loss: 0.5485 - accuracy: 0.7228 - val_loss: 0.6319 - val_accuracy: 0.6830
Epoch 118/400
62/62 [====] - 3s 43ms/step - loss: 0.5631 - accuracy: 0.7217 - val_loss: 0.6513 - val_accuracy: 0.6892
Epoch 119/400
62/62 [====] - 3s 42ms/step - loss: 0.5613 - accuracy: 0.7279 - val_loss: 0.6579 - val_accuracy: 0.6830
Epoch 120/400
62/62 [====] - 3s 42ms/step - loss: 0.5779 - accuracy: 0.7243 - val_loss: 0.7316 - val_accuracy: 0.6769
Epoch 121/400
62/62 [====] - 3s 42ms/step - loss: 0.5638 - accuracy: 0.7284 - val_loss: 0.6398 - val_accuracy: 0.6605
Epoch 122/400
62/62 [====] - 3s 42ms/step - loss: 0.5337 - accuracy: 0.7494 - val_loss: 0.7681 - val_accuracy: 0.7014
Epoch 123/400
62/62 [====] - 3s 42ms/step - loss: 0.5615 - accuracy: 0.7299 - val_loss: 0.7224 - val_accuracy: 0.6810
Epoch 124/400
62/62 [====] - 3s 42ms/step - loss: 0.5578 - accuracy: 0.7192 - val_loss: 0.6428 - val_accuracy: 0.6687
Epoch 125/400
62/62 [====] - 3s 46ms/step - loss: 0.5414 - accuracy: 0.7376 - val_loss: 0.7362 - val_accuracy: 0.6851
Epoch 126/400
62/62 [====] - 3s 42ms/step - loss: 0.5412 - accuracy: 0.7309 - val_loss: 0.7505 - val_accuracy: 0.6421
Epoch 127/400
62/62 [====] - 3s 42ms/step - loss: 0.5653 - accuracy: 0.7171 - val_loss: 0.8074 - val_accuracy: 0.6769
Epoch 128/400
62/62 [====] - 3s 43ms/step - loss: 0.5277 - accuracy: 0.7422 - val_loss: 0.6165 - val_accuracy: 0.7137
Epoch 129/400
62/62 [====] - 3s 43ms/step - loss: 0.5398 - accuracy: 0.7412 - val_loss: 0.6065 - val_accuracy: 0.7157
Epoch 130/400
62/62 [====] - 3s 42ms/step - loss: 0.5352 - accuracy: 0.7315 - val_loss: 0.6737 - val_accuracy: 0.6708
Epoch 131/400
62/62 [====] - 3s 42ms/step - loss: 0.5690 - accuracy: 0.7258 - val_loss: 0.6278 - val_accuracy: 0.6912
Epoch 132/400
62/62 [====] - 3s 43ms/step - loss: 0.5469 - accuracy: 0.7228 - val_loss: 0.6333 - val_accuracy: 0.6912
Epoch 133/400
62/62 [====] - 3s 43ms/step - loss: 0.5404 - accuracy: 0.7269 - val_loss: 0.7959 - val_accuracy: 0.6973
Epoch 134/400
62/62 [====] - 3s 42ms/step - loss: 0.5473 - accuracy: 0.7391 - val_loss: 0.6411 - val_accuracy: 0.7076
Epoch 135/400
62/62 [====] - 3s 43ms/step - loss: 0.5290 - accuracy: 0.7361 - val_loss: 0.6623 - val_accuracy: 0.6933
Epoch 136/400
62/62 [====] - 3s 42ms/step - loss: 0.5712 - accuracy: 0.7361 - val_loss: 0.6576 - val_accuracy: 0.6626
Epoch 137/400
62/62 [====] - 3s 42ms/step - loss: 0.5386 - accuracy: 0.7325 - val_loss: 0.6665 - val_accuracy: 0.7076
Epoch 138/400
62/62 [====] - 3s 42ms/step - loss: 0.5398 - accuracy: 0.7335 - val_loss: 0.6505 - val_accuracy: 0.6871
Epoch 139/400
62/62 [====] - 3s 42ms/step - loss: 0.5236 - accuracy: 0.7483 - val_loss: 0.6443 - val_accuracy: 0.7076
Epoch 140/400
62/62 [====] - 3s 42ms/step - loss: 0.5583 - accuracy: 0.7355 - val_loss: 0.9932 - val_accuracy: 0.7035
Epoch 141/400
62/62 [====] - 3s 46ms/step - loss: 0.5358 - accuracy: 0.7381 - val_loss: 0.6574 - val_accuracy: 0.6748
Epoch 142/400
62/62 [====] - 3s 43ms/step - loss: 0.5774 - accuracy: 0.7315 - val_loss: 0.6426 - val_accuracy: 0.6892
Epoch 143/400
62/62 [====] - 3s 42ms/step - loss: 0.5177 - accuracy: 0.7412 - val_loss: 0.6444 - val_accuracy: 0.6912
Epoch 144/400
62/62 [====] - 3s 42ms/step - loss: 0.5412 - accuracy: 0.7432 - val_loss: 0.6908 - val_accuracy: 0.7035
Epoch 145/400
62/62 [====] - 3s 42ms/step - loss: 0.5157 - accuracy: 0.7514 - val_loss: 0.6829 - val_accuracy: 0.6830
Epoch 146/400
62/62 [====] - 3s 43ms/step - loss: 0.5382 - accuracy: 0.7432 - val_loss: 0.7015 - val_accuracy: 0.6789
Epoch 147/400
62/62 [====] - 3s 42ms/step - loss: 0.5174 - accuracy: 0.7437 - val_loss: 0.6726 - val_accuracy: 0.6994
Epoch 148/400
62/62 [====] - 3s 42ms/step - loss: 0.5094 - accuracy: 0.7550 - val_loss: 0.7689 - val_accuracy: 0.6708
Epoch 149/400
62/62 [====] - 3s 42ms/step - loss: 0.5239 - accuracy: 0.7468 - val_loss: 0.7434 - val_accuracy: 0.6851
Epoch 150/400
62/62 [====] - 3s 42ms/step - loss: 0.5285 - accuracy: 0.7560 - val_loss: 0.8770 - val_accuracy: 0.6033
Epoch 151/400
62/62 [====] - 3s 42ms/step - loss: 0.5327 - accuracy: 0.7402 - val_loss: 0.6097 - val_accuracy: 0.7035
```

```
Epoch 152/400
62/62 [====] - 3s 43ms/step - loss: 0.5532 - accuracy: 0.7345 - val_loss: 0.6664 - val_accuracy: 0.7178
Epoch 153/400
62/62 [====] - 3s 42ms/step - loss: 0.5431 - accuracy: 0.7448 - val_loss: 0.8822 - val_accuracy: 0.7444
Epoch 154/400
62/62 [====] - 3s 42ms/step - loss: 0.5108 - accuracy: 0.7668 - val_loss: 0.8325 - val_accuracy: 0.6564
Epoch 155/400
62/62 [====] - 3s 42ms/step - loss: 0.5368 - accuracy: 0.7386 - val_loss: 0.8035 - val_accuracy: 0.7035
Epoch 156/400
62/62 [====] - 3s 42ms/step - loss: 0.5163 - accuracy: 0.7688 - val_loss: 1.0595 - val_accuracy: 0.6544
Epoch 157/400
62/62 [====] - 3s 43ms/step - loss: 0.5452 - accuracy: 0.7535 - val_loss: 0.6583 - val_accuracy: 0.7280
Epoch 158/400
62/62 [====] - 3s 43ms/step - loss: 0.5382 - accuracy: 0.7432 - val_loss: 0.6420 - val_accuracy: 0.7178
Epoch 159/400
62/62 [====] - 3s 42ms/step - loss: 0.5538 - accuracy: 0.7463 - val_loss: 0.6675 - val_accuracy: 0.6994
Epoch 160/400
62/62 [====] - 3s 42ms/step - loss: 0.5317 - accuracy: 0.7596 - val_loss: 0.8630 - val_accuracy: 0.7076
Epoch 161/400
62/62 [====] - 3s 41ms/step - loss: 0.5170 - accuracy: 0.7550 - val_loss: 0.6455 - val_accuracy: 0.6708
Epoch 162/400
62/62 [====] - 3s 43ms/step - loss: 0.5105 - accuracy: 0.7560 - val_loss: 0.6171 - val_accuracy: 0.7239
Epoch 163/400
62/62 [====] - 3s 42ms/step - loss: 0.5129 - accuracy: 0.7688 - val_loss: 0.6468 - val_accuracy: 0.6871
Epoch 164/400
62/62 [====] - 3s 43ms/step - loss: 0.5239 - accuracy: 0.7688 - val_loss: 0.6753 - val_accuracy: 0.6605
Epoch 165/400
62/62 [====] - 3s 42ms/step - loss: 0.5404 - accuracy: 0.7488 - val_loss: 0.7675 - val_accuracy: 0.6830
Epoch 166/400
62/62 [====] - 3s 42ms/step - loss: 0.5086 - accuracy: 0.7514 - val_loss: 0.6267 - val_accuracy: 0.7485
Epoch 167/400
62/62 [====] - 3s 42ms/step - loss: 0.5024 - accuracy: 0.7714 - val_loss: 0.6757 - val_accuracy: 0.7137
Epoch 168/400
62/62 [====] - 3s 42ms/step - loss: 0.5160 - accuracy: 0.7550 - val_loss: 0.6910 - val_accuracy: 0.7178
Epoch 169/400
62/62 [====] - 3s 42ms/step - loss: 0.5100 - accuracy: 0.7540 - val_loss: 0.7387 - val_accuracy: 0.6912
Epoch 170/400
62/62 [====] - 3s 42ms/step - loss: 0.5130 - accuracy: 0.7642 - val_loss: 0.6446 - val_accuracy: 0.7260
Epoch 171/400
62/62 [====] - 3s 42ms/step - loss: 0.5000 - accuracy: 0.7729 - val_loss: 0.9046 - val_accuracy: 0.6544
Epoch 172/400
62/62 [====] - 3s 42ms/step - loss: 0.5110 - accuracy: 0.7668 - val_loss: 0.6564 - val_accuracy: 0.7260
Epoch 173/400
62/62 [====] - 3s 42ms/step - loss: 0.4798 - accuracy: 0.7693 - val_loss: 0.8235 - val_accuracy: 0.6667
Epoch 174/400
62/62 [====] - 3s 42ms/step - loss: 0.5352 - accuracy: 0.7509 - val_loss: 0.7532 - val_accuracy: 0.6667
Epoch 175/400
62/62 [====] - 3s 42ms/step - loss: 0.4898 - accuracy: 0.7801 - val_loss: 0.6650 - val_accuracy: 0.6953
Epoch 176/400
62/62 [====] - 3s 42ms/step - loss: 0.4848 - accuracy: 0.7565 - val_loss: 0.7618 - val_accuracy: 0.7096
Epoch 177/400
62/62 [====] - 3s 42ms/step - loss: 0.4893 - accuracy: 0.7729 - val_loss: 0.7609 - val_accuracy: 0.6892
Epoch 178/400
62/62 [====] - 3s 42ms/step - loss: 0.5075 - accuracy: 0.7657 - val_loss: 0.7048 - val_accuracy: 0.6851
Epoch 179/400
62/62 [====] - 3s 42ms/step - loss: 0.4879 - accuracy: 0.7749 - val_loss: 0.6787 - val_accuracy: 0.7035
Epoch 180/400
62/62 [====] - 3s 41ms/step - loss: 0.5226 - accuracy: 0.7760 - val_loss: 0.7600 - val_accuracy: 0.7076
Epoch 181/400
62/62 [====] - 3s 41ms/step - loss: 0.4855 - accuracy: 0.7795 - val_loss: 0.7516 - val_accuracy: 0.6851
Epoch 182/400
62/62 [====] - 3s 41ms/step - loss: 0.4805 - accuracy: 0.7867 - val_loss: 0.6751 - val_accuracy: 0.6851
Epoch 183/400
62/62 [====] - 3s 41ms/step - loss: 0.4920 - accuracy: 0.7744 - val_loss: 0.7540 - val_accuracy: 0.7239
Epoch 184/400
62/62 [====] - 3s 41ms/step - loss: 0.5213 - accuracy: 0.7642 - val_loss: 0.7054 - val_accuracy: 0.7362
Epoch 185/400
62/62 [====] - 3s 41ms/step - loss: 0.4942 - accuracy: 0.7693 - val_loss: 0.6005 - val_accuracy: 0.7342
Epoch 186/400
62/62 [====] - 3s 41ms/step - loss: 0.5294 - accuracy: 0.7504 - val_loss: 0.6926 - val_accuracy: 0.6830
Epoch 187/400
62/62 [====] - 3s 42ms/step - loss: 0.4812 - accuracy: 0.7744 - val_loss: 0.5962 - val_accuracy: 0.7505
Epoch 188/400
62/62 [====] - 3s 41ms/step - loss: 0.4861 - accuracy: 0.7831 - val_loss: 0.7846 - val_accuracy: 0.7035
Epoch 189/400
62/62 [====] - 3s 42ms/step - loss: 0.4950 - accuracy: 0.7698 - val_loss: 0.6899 - val_accuracy: 0.7014
Epoch 190/400
62/62 [====] - 3s 42ms/step - loss: 0.4920 - accuracy: 0.7693 - val_loss: 0.6552 - val_accuracy: 0.7485
Epoch 191/400
62/62 [====] - 3s 41ms/step - loss: 0.4937 - accuracy: 0.7647 - val_loss: 0.6630 - val_accuracy: 0.6994
Epoch 192/400
62/62 [====] - 3s 42ms/step - loss: 0.4768 - accuracy: 0.7785 - val_loss: 0.6841 - val_accuracy: 0.7014
```

```
Epoch 193/400
62/62 [====] - 3s 41ms/step - loss: 0.5013 - accuracy: 0.7719 - val_loss: 0.6267 - val_accuracy: 0.6810
Epoch 194/400
62/62 [====] - 3s 41ms/step - loss: 0.4737 - accuracy: 0.7887 - val_loss: 0.6754 - val_accuracy: 0.7423
Epoch 195/400
62/62 [====] - 3s 41ms/step - loss: 0.4894 - accuracy: 0.7841 - val_loss: 0.6670 - val_accuracy: 0.7301
Epoch 196/400
62/62 [====] - 3s 42ms/step - loss: 0.5052 - accuracy: 0.7857 - val_loss: 0.7114 - val_accuracy: 0.7464
Epoch 197/400
62/62 [====] - 3s 41ms/step - loss: 0.4761 - accuracy: 0.7939 - val_loss: 0.6337 - val_accuracy: 0.7505
Epoch 198/400
62/62 [====] - 3s 41ms/step - loss: 0.4861 - accuracy: 0.7857 - val_loss: 0.6824 - val_accuracy: 0.7321
Epoch 199/400
62/62 [====] - 3s 41ms/step - loss: 0.5011 - accuracy: 0.7775 - val_loss: 0.6401 - val_accuracy: 0.7076
Epoch 200/400
62/62 [====] - 3s 40ms/step - loss: 0.5019 - accuracy: 0.7790 - val_loss: 0.7780 - val_accuracy: 0.7730
Epoch 201/400
62/62 [====] - 3s 41ms/step - loss: 0.4687 - accuracy: 0.7847 - val_loss: 0.6599 - val_accuracy: 0.7382
Epoch 202/400
62/62 [====] - 3s 42ms/step - loss: 0.4790 - accuracy: 0.7729 - val_loss: 0.5776 - val_accuracy: 0.7403
Epoch 203/400
62/62 [====] - 3s 42ms/step - loss: 0.5195 - accuracy: 0.7816 - val_loss: 0.5792 - val_accuracy: 0.7423
Epoch 204/400
62/62 [====] - 3s 41ms/step - loss: 0.4736 - accuracy: 0.7739 - val_loss: 0.7309 - val_accuracy: 0.7239
Epoch 205/400
62/62 [====] - 3s 41ms/step - loss: 0.4765 - accuracy: 0.7893 - val_loss: 0.6744 - val_accuracy: 0.7362
Epoch 206/400
62/62 [====] - 3s 41ms/step - loss: 0.4783 - accuracy: 0.7867 - val_loss: 0.8894 - val_accuracy: 0.7096
Epoch 207/400
62/62 [====] - 3s 41ms/step - loss: 0.4814 - accuracy: 0.7852 - val_loss: 0.7126 - val_accuracy: 0.7505
Epoch 208/400
62/62 [====] - 3s 41ms/step - loss: 0.5284 - accuracy: 0.7872 - val_loss: 0.7106 - val_accuracy: 0.7014
Epoch 209/400
62/62 [====] - 3s 45ms/step - loss: 0.4726 - accuracy: 0.7841 - val_loss: 0.8343 - val_accuracy: 0.7382
Epoch 210/400
62/62 [====] - 3s 40ms/step - loss: 0.4658 - accuracy: 0.7867 - val_loss: 0.7796 - val_accuracy: 0.7239
Epoch 211/400
62/62 [====] - 3s 41ms/step - loss: 0.4738 - accuracy: 0.7816 - val_loss: 0.5946 - val_accuracy: 0.7342
Epoch 212/400
62/62 [====] - 3s 41ms/step - loss: 0.4504 - accuracy: 0.7949 - val_loss: 0.6400 - val_accuracy: 0.7382
Epoch 213/400
62/62 [====] - 3s 42ms/step - loss: 0.4733 - accuracy: 0.7949 - val_loss: 0.7550 - val_accuracy: 0.6933
Epoch 214/400
62/62 [====] - 3s 42ms/step - loss: 0.4894 - accuracy: 0.7683 - val_loss: 0.7456 - val_accuracy: 0.7117
Epoch 215/400
62/62 [====] - 3s 41ms/step - loss: 0.4879 - accuracy: 0.7760 - val_loss: 0.7259 - val_accuracy: 0.7239
Epoch 216/400
62/62 [====] - 3s 42ms/step - loss: 0.4646 - accuracy: 0.7893 - val_loss: 0.8053 - val_accuracy: 0.6912
Epoch 217/400
62/62 [====] - 3s 41ms/step - loss: 0.4678 - accuracy: 0.7831 - val_loss: 0.7781 - val_accuracy: 0.7137
Epoch 218/400
62/62 [====] - 3s 41ms/step - loss: 0.4558 - accuracy: 0.7928 - val_loss: 0.8494 - val_accuracy: 0.7505
Epoch 219/400
62/62 [====] - 3s 41ms/step - loss: 0.4557 - accuracy: 0.7959 - val_loss: 0.9486 - val_accuracy: 0.7076
Epoch 220/400
62/62 [====] - 3s 42ms/step - loss: 0.4770 - accuracy: 0.7908 - val_loss: 1.1070 - val_accuracy: 0.6462
Epoch 221/400
62/62 [====] - 3s 42ms/step - loss: 0.4735 - accuracy: 0.7857 - val_loss: 0.7159 - val_accuracy: 0.7076
Epoch 222/400
62/62 [====] - 3s 41ms/step - loss: 0.4893 - accuracy: 0.7780 - val_loss: 0.6895 - val_accuracy: 0.7382
Epoch 223/400
62/62 [====] - 3s 41ms/step - loss: 0.4673 - accuracy: 0.7882 - val_loss: 0.9451 - val_accuracy: 0.7117
Epoch 224/400
62/62 [====] - 3s 42ms/step - loss: 0.5013 - accuracy: 0.7923 - val_loss: 1.1470 - val_accuracy: 0.5930
Epoch 225/400
62/62 [====] - 3s 41ms/step - loss: 0.4828 - accuracy: 0.7877 - val_loss: 0.8999 - val_accuracy: 0.7342
Epoch 226/400
62/62 [====] - 3s 41ms/step - loss: 0.4614 - accuracy: 0.7882 - val_loss: 0.7219 - val_accuracy: 0.7076
Epoch 227/400
62/62 [====] - 3s 41ms/step - loss: 0.4910 - accuracy: 0.7754 - val_loss: 1.1044 - val_accuracy: 0.6258
Epoch 228/400
62/62 [====] - 3s 41ms/step - loss: 0.4846 - accuracy: 0.7887 - val_loss: 0.9704 - val_accuracy: 0.7710
Epoch 229/400
62/62 [====] - 3s 41ms/step - loss: 0.4873 - accuracy: 0.7872 - val_loss: 0.9457 - val_accuracy: 0.7464
Epoch 230/400
62/62 [====] - 3s 42ms/step - loss: 0.4650 - accuracy: 0.8072 - val_loss: 0.7766 - val_accuracy: 0.7014
Epoch 231/400
62/62 [====] - 3s 40ms/step - loss: 0.4884 - accuracy: 0.7857 - val_loss: 0.6141 - val_accuracy: 0.7464
Epoch 232/400
62/62 [====] - 3s 41ms/step - loss: 0.4329 - accuracy: 0.8046 - val_loss: 0.8193 - val_accuracy: 0.7362
Epoch 233/400
62/62 [====] - 3s 41ms/step - loss: 0.5330 - accuracy: 0.8046 - val_loss: 0.8251 - val_accuracy: 0.6994
```

```
Epoch 234/400
62/62 [====] - 3s 41ms/step - loss: 0.4697 - accuracy: 0.7908 - val_loss: 0.8371 - val_accuracy: 0.6892
Epoch 235/400
62/62 [====] - 3s 42ms/step - loss: 0.4728 - accuracy: 0.7882 - val_loss: 0.6935 - val_accuracy: 0.7587
Epoch 236/400
62/62 [====] - 3s 41ms/step - loss: 0.4690 - accuracy: 0.7908 - val_loss: 0.7304 - val_accuracy: 0.7076
Epoch 237/400
62/62 [====] - 3s 41ms/step - loss: 0.4769 - accuracy: 0.7811 - val_loss: 0.7090 - val_accuracy: 0.7301
Epoch 238/400
62/62 [====] - 3s 42ms/step - loss: 0.5309 - accuracy: 0.7847 - val_loss: 0.8858 - val_accuracy: 0.6605
Epoch 239/400
62/62 [====] - 3s 42ms/step - loss: 0.4643 - accuracy: 0.7969 - val_loss: 0.7275 - val_accuracy: 0.7403
Epoch 240/400
62/62 [====] - 3s 42ms/step - loss: 0.4803 - accuracy: 0.7847 - val_loss: 0.7154 - val_accuracy: 0.7423
Epoch 241/400
62/62 [====] - 3s 42ms/step - loss: 0.5068 - accuracy: 0.7974 - val_loss: 0.6486 - val_accuracy: 0.7669
Epoch 242/400
62/62 [====] - 3s 41ms/step - loss: 0.4752 - accuracy: 0.7913 - val_loss: 0.7627 - val_accuracy: 0.6953
Epoch 243/400
62/62 [====] - 3s 41ms/step - loss: 0.4886 - accuracy: 0.7877 - val_loss: 0.7291 - val_accuracy: 0.7117
Epoch 244/400
62/62 [====] - 3s 42ms/step - loss: 0.4431 - accuracy: 0.7995 - val_loss: 0.9053 - val_accuracy: 0.6953
Epoch 245/400
62/62 [====] - 3s 42ms/step - loss: 0.4575 - accuracy: 0.8015 - val_loss: 0.8181 - val_accuracy: 0.7096
Epoch 246/400
62/62 [====] - 3s 42ms/step - loss: 0.4458 - accuracy: 0.8087 - val_loss: 0.7963 - val_accuracy: 0.7566
Epoch 247/400
62/62 [====] - 3s 42ms/step - loss: 0.4482 - accuracy: 0.8041 - val_loss: 0.6695 - val_accuracy: 0.7566
Epoch 248/400
62/62 [====] - 3s 42ms/step - loss: 0.4406 - accuracy: 0.8015 - val_loss: 1.0636 - val_accuracy: 0.7014
Epoch 249/400
62/62 [====] - 3s 43ms/step - loss: 0.4694 - accuracy: 0.7985 - val_loss: 0.6400 - val_accuracy: 0.7362
Epoch 250/400
62/62 [====] - 3s 42ms/step - loss: 0.4550 - accuracy: 0.7980 - val_loss: 0.8229 - val_accuracy: 0.6994
Epoch 251/400
62/62 [====] - 3s 42ms/step - loss: 0.4491 - accuracy: 0.7908 - val_loss: 0.7160 - val_accuracy: 0.7096
Epoch 252/400
62/62 [====] - 3s 46ms/step - loss: 0.4531 - accuracy: 0.8061 - val_loss: 0.7067 - val_accuracy: 0.7178
Epoch 253/400
62/62 [====] - 3s 42ms/step - loss: 0.4265 - accuracy: 0.8128 - val_loss: 0.8725 - val_accuracy: 0.7423
Epoch 254/400
62/62 [====] - 3s 42ms/step - loss: 0.4932 - accuracy: 0.7949 - val_loss: 0.6435 - val_accuracy: 0.7342
Epoch 255/400
62/62 [====] - 3s 42ms/step - loss: 0.4495 - accuracy: 0.7980 - val_loss: 0.8828 - val_accuracy: 0.7505
Epoch 256/400
62/62 [====] - 3s 42ms/step - loss: 0.5202 - accuracy: 0.7841 - val_loss: 1.0775 - val_accuracy: 0.6626
Epoch 257/400
62/62 [====] - 3s 41ms/step - loss: 0.4674 - accuracy: 0.7944 - val_loss: 0.6722 - val_accuracy: 0.7628
Epoch 258/400
62/62 [====] - 3s 42ms/step - loss: 0.4606 - accuracy: 0.7939 - val_loss: 0.7883 - val_accuracy: 0.6789
Epoch 259/400
62/62 [====] - 3s 41ms/step - loss: 0.4525 - accuracy: 0.8010 - val_loss: 0.7259 - val_accuracy: 0.7382
Epoch 260/400
62/62 [====] - 3s 42ms/step - loss: 0.4574 - accuracy: 0.8046 - val_loss: 0.7075 - val_accuracy: 0.7117
Epoch 261/400
62/62 [====] - 3s 42ms/step - loss: 0.4462 - accuracy: 0.8036 - val_loss: 0.7429 - val_accuracy: 0.7342
Epoch 262/400
62/62 [====] - 3s 42ms/step - loss: 0.4820 - accuracy: 0.7923 - val_loss: 0.6750 - val_accuracy: 0.7342
Epoch 263/400
62/62 [====] - 3s 42ms/step - loss: 0.4771 - accuracy: 0.7959 - val_loss: 0.7136 - val_accuracy: 0.7342
Epoch 264/400
62/62 [====] - 3s 43ms/step - loss: 0.4521 - accuracy: 0.8092 - val_loss: 0.8326 - val_accuracy: 0.6994
Epoch 265/400
62/62 [====] - 3s 42ms/step - loss: 0.4590 - accuracy: 0.7990 - val_loss: 0.6944 - val_accuracy: 0.7485
Epoch 266/400
62/62 [====] - 3s 42ms/step - loss: 0.4900 - accuracy: 0.7985 - val_loss: 0.8136 - val_accuracy: 0.6646
Epoch 267/400
62/62 [====] - 3s 42ms/step - loss: 0.5391 - accuracy: 0.7923 - val_loss: 0.6848 - val_accuracy: 0.7444
Epoch 268/400
62/62 [====] - 3s 42ms/step - loss: 0.4285 - accuracy: 0.8133 - val_loss: 0.9637 - val_accuracy: 0.7055
Epoch 269/400
62/62 [====] - 3s 43ms/step - loss: 0.4843 - accuracy: 0.7852 - val_loss: 0.9608 - val_accuracy: 0.7321
Epoch 270/400
62/62 [====] - 3s 42ms/step - loss: 0.4360 - accuracy: 0.8077 - val_loss: 0.8636 - val_accuracy: 0.7096
Epoch 271/400
62/62 [====] - 3s 42ms/step - loss: 0.4717 - accuracy: 0.8015 - val_loss: 0.8551 - val_accuracy: 0.7219
Epoch 272/400
62/62 [====] - 3s 42ms/step - loss: 0.4607 - accuracy: 0.7985 - val_loss: 0.8732 - val_accuracy: 0.7526
Epoch 273/400
62/62 [====] - 3s 42ms/step - loss: 0.4447 - accuracy: 0.7964 - val_loss: 1.1244 - val_accuracy: 0.7137
Epoch 274/400
62/62 [====] - 3s 42ms/step - loss: 0.4505 - accuracy: 0.7918 - val_loss: 0.7435 - val_accuracy: 0.7260
```

```
Epoch 275/400
62/62 [====] - 3s 43ms/step - loss: 0.5034 - accuracy: 0.8077 - val_loss: 0.8406 - val_accuracy: 0.7342
Epoch 276/400
62/62 [====] - 3s 43ms/step - loss: 0.5050 - accuracy: 0.8046 - val_loss: 0.9671 - val_accuracy: 0.7566
Epoch 277/400
62/62 [====] - 3s 42ms/step - loss: 0.4581 - accuracy: 0.7934 - val_loss: 0.8970 - val_accuracy: 0.7423
Epoch 278/400
62/62 [====] - 3s 42ms/step - loss: 0.4932 - accuracy: 0.8133 - val_loss: 0.8198 - val_accuracy: 0.7444
Epoch 279/400
62/62 [====] - 3s 42ms/step - loss: 0.4553 - accuracy: 0.7959 - val_loss: 0.8219 - val_accuracy: 0.7587
Epoch 280/400
62/62 [====] - 3s 42ms/step - loss: 0.4663 - accuracy: 0.8041 - val_loss: 0.7693 - val_accuracy: 0.7280
Epoch 281/400
62/62 [====] - 3s 42ms/step - loss: 0.4285 - accuracy: 0.8066 - val_loss: 0.7721 - val_accuracy: 0.7546
Epoch 282/400
62/62 [====] - 3s 42ms/step - loss: 0.4591 - accuracy: 0.8056 - val_loss: 0.7062 - val_accuracy: 0.7628
Epoch 283/400
62/62 [====] - 3s 42ms/step - loss: 0.4584 - accuracy: 0.8051 - val_loss: 0.8424 - val_accuracy: 0.7157
Epoch 284/400
62/62 [====] - 3s 42ms/step - loss: 0.4502 - accuracy: 0.8066 - val_loss: 0.8525 - val_accuracy: 0.7301
Epoch 285/400
62/62 [====] - 3s 42ms/step - loss: 0.4557 - accuracy: 0.8036 - val_loss: 1.2967 - val_accuracy: 0.6483
Epoch 286/400
62/62 [====] - 3s 42ms/step - loss: 0.4487 - accuracy: 0.8128 - val_loss: 1.1427 - val_accuracy: 0.7444
Epoch 287/400
62/62 [====] - 3s 43ms/step - loss: 0.4421 - accuracy: 0.8148 - val_loss: 0.7093 - val_accuracy: 0.7444
Epoch 288/400
62/62 [====] - 3s 42ms/step - loss: 0.4620 - accuracy: 0.8046 - val_loss: 0.6531 - val_accuracy: 0.7566
Epoch 289/400
62/62 [====] - 3s 42ms/step - loss: 0.4495 - accuracy: 0.8041 - val_loss: 0.8021 - val_accuracy: 0.7219
Epoch 290/400
62/62 [====] - 3s 42ms/step - loss: 0.4385 - accuracy: 0.8210 - val_loss: 0.9311 - val_accuracy: 0.7198
Epoch 291/400
62/62 [====] - 3s 42ms/step - loss: 0.4591 - accuracy: 0.7959 - val_loss: 1.3036 - val_accuracy: 0.7198
Epoch 292/400
62/62 [====] - 3s 42ms/step - loss: 0.4413 - accuracy: 0.8092 - val_loss: 0.8500 - val_accuracy: 0.7362
Epoch 293/400
62/62 [====] - 3s 42ms/step - loss: 0.4784 - accuracy: 0.8077 - val_loss: 0.6631 - val_accuracy: 0.7382
Epoch 294/400
62/62 [====] - 3s 43ms/step - loss: 0.4994 - accuracy: 0.8036 - val_loss: 0.6841 - val_accuracy: 0.7321
Epoch 295/400
62/62 [====] - 3s 42ms/step - loss: 0.4777 - accuracy: 0.8092 - val_loss: 0.7261 - val_accuracy: 0.7566
Epoch 296/400
62/62 [====] - 3s 43ms/step - loss: 0.4568 - accuracy: 0.8041 - val_loss: 0.6127 - val_accuracy: 0.7035
Epoch 297/400
62/62 [====] - 3s 43ms/step - loss: 0.4568 - accuracy: 0.7949 - val_loss: 0.8936 - val_accuracy: 0.7301
Epoch 298/400
62/62 [====] - 3s 42ms/step - loss: 0.4513 - accuracy: 0.7959 - val_loss: 0.9518 - val_accuracy: 0.7362
Epoch 299/400
62/62 [====] - 3s 42ms/step - loss: 0.4727 - accuracy: 0.7913 - val_loss: 0.7656 - val_accuracy: 0.6953
Epoch 300/400
62/62 [====] - 3s 43ms/step - loss: 0.4413 - accuracy: 0.8092 - val_loss: 0.7586 - val_accuracy: 0.7648
Epoch 301/400
62/62 [====] - 3s 42ms/step - loss: 0.4667 - accuracy: 0.8107 - val_loss: 0.8426 - val_accuracy: 0.7526
Epoch 302/400
62/62 [====] - 3s 43ms/step - loss: 0.4541 - accuracy: 0.7995 - val_loss: 1.2659 - val_accuracy: 0.6912
Epoch 303/400
62/62 [====] - 3s 43ms/step - loss: 0.4579 - accuracy: 0.7903 - val_loss: 0.8221 - val_accuracy: 0.7423
Epoch 304/400
62/62 [====] - 3s 42ms/step - loss: 0.4412 - accuracy: 0.8118 - val_loss: 0.8619 - val_accuracy: 0.7096
Epoch 305/400
62/62 [====] - 3s 42ms/step - loss: 0.4446 - accuracy: 0.8133 - val_loss: 0.9387 - val_accuracy: 0.7096
Epoch 306/400
62/62 [====] - 3s 42ms/step - loss: 0.4924 - accuracy: 0.8143 - val_loss: 0.7491 - val_accuracy: 0.7403
Epoch 307/400
62/62 [====] - 3s 43ms/step - loss: 0.4560 - accuracy: 0.8159 - val_loss: 1.1176 - val_accuracy: 0.7321
Epoch 308/400
62/62 [====] - 3s 43ms/step - loss: 0.4855 - accuracy: 0.8031 - val_loss: 0.7872 - val_accuracy: 0.7628
Epoch 309/400
62/62 [====] - 3s 43ms/step - loss: 0.4536 - accuracy: 0.8077 - val_loss: 1.0092 - val_accuracy: 0.7280
Epoch 310/400
62/62 [====] - 3s 42ms/step - loss: 0.4664 - accuracy: 0.8036 - val_loss: 0.9516 - val_accuracy: 0.7505
Epoch 311/400
62/62 [====] - 3s 44ms/step - loss: 0.4996 - accuracy: 0.7790 - val_loss: 0.8947 - val_accuracy: 0.7505
Epoch 312/400
62/62 [====] - 3s 42ms/step - loss: 0.4743 - accuracy: 0.8133 - val_loss: 1.0574 - val_accuracy: 0.7444
Epoch 313/400
62/62 [====] - 3s 42ms/step - loss: 0.4675 - accuracy: 0.7872 - val_loss: 0.7133 - val_accuracy: 0.6973
Epoch 314/400
62/62 [====] - 3s 42ms/step - loss: 0.4542 - accuracy: 0.8026 - val_loss: 0.7187 - val_accuracy: 0.6892
Epoch 315/400
62/62 [====] - 3s 42ms/step - loss: 0.4400 - accuracy: 0.8143 - val_loss: 0.9450 - val_accuracy: 0.7239
```

```
Epoch 316/400
62/62 [====] - 3s 42ms/step - loss: 0.5406 - accuracy: 0.7928 - val_loss: 0.7051 - val_accuracy: 0.7198
Epoch 317/400
62/62 [====] - 3s 43ms/step - loss: 0.4568 - accuracy: 0.7990 - val_loss: 0.7066 - val_accuracy: 0.7157
Epoch 318/400
62/62 [====] - 3s 43ms/step - loss: 0.4267 - accuracy: 0.8164 - val_loss: 0.9265 - val_accuracy: 0.7546
Epoch 319/400
62/62 [====] - 3s 43ms/step - loss: 0.4139 - accuracy: 0.8194 - val_loss: 1.1452 - val_accuracy: 0.6483
Epoch 320/400
62/62 [====] - 3s 43ms/step - loss: 0.4668 - accuracy: 0.8056 - val_loss: 1.0174 - val_accuracy: 0.6564
Epoch 321/400
62/62 [====] - 3s 43ms/step - loss: 0.4714 - accuracy: 0.8159 - val_loss: 0.8132 - val_accuracy: 0.7485
Epoch 322/400
62/62 [====] - 3s 42ms/step - loss: 0.4675 - accuracy: 0.8164 - val_loss: 0.7204 - val_accuracy: 0.7260
Epoch 323/400
62/62 [====] - 3s 42ms/step - loss: 0.4184 - accuracy: 0.8220 - val_loss: 0.7399 - val_accuracy: 0.7607
Epoch 324/400
62/62 [====] - 3s 42ms/step - loss: 0.4003 - accuracy: 0.8373 - val_loss: 0.8378 - val_accuracy: 0.7362
Epoch 325/400
62/62 [====] - 3s 42ms/step - loss: 0.4392 - accuracy: 0.8205 - val_loss: 0.8106 - val_accuracy: 0.7342
Epoch 326/400
62/62 [====] - 3s 42ms/step - loss: 0.4527 - accuracy: 0.8143 - val_loss: 0.6643 - val_accuracy: 0.7382
Epoch 327/400
62/62 [====] - 3s 43ms/step - loss: 0.4336 - accuracy: 0.8225 - val_loss: 0.8256 - val_accuracy: 0.7321
Epoch 328/400
62/62 [====] - 3s 43ms/step - loss: 0.4483 - accuracy: 0.8194 - val_loss: 0.6855 - val_accuracy: 0.7403
Epoch 329/400
62/62 [====] - 3s 43ms/step - loss: 0.4441 - accuracy: 0.8133 - val_loss: 0.8090 - val_accuracy: 0.7730
Epoch 330/400
62/62 [====] - 3s 43ms/step - loss: 0.4300 - accuracy: 0.8036 - val_loss: 0.8907 - val_accuracy: 0.7342
Epoch 331/400
62/62 [====] - 3s 42ms/step - loss: 0.4661 - accuracy: 0.8113 - val_loss: 0.8227 - val_accuracy: 0.7260
Epoch 332/400
62/62 [====] - 3s 42ms/step - loss: 0.4579 - accuracy: 0.8036 - val_loss: 0.7926 - val_accuracy: 0.7301
Epoch 333/400
62/62 [====] - 3s 42ms/step - loss: 0.4921 - accuracy: 0.8235 - val_loss: 0.8791 - val_accuracy: 0.7301
Epoch 334/400
62/62 [====] - 3s 43ms/step - loss: 0.4660 - accuracy: 0.8240 - val_loss: 1.2190 - val_accuracy: 0.6851
Epoch 335/400
62/62 [====] - 3s 42ms/step - loss: 0.4578 - accuracy: 0.8087 - val_loss: 0.7666 - val_accuracy: 0.7178
Epoch 336/400
62/62 [====] - 3s 42ms/step - loss: 0.4315 - accuracy: 0.8133 - val_loss: 0.9471 - val_accuracy: 0.7342
Epoch 337/400
62/62 [====] - 3s 43ms/step - loss: 0.4104 - accuracy: 0.8199 - val_loss: 1.1381 - val_accuracy: 0.7239
Epoch 338/400
62/62 [====] - 3s 42ms/step - loss: 0.4776 - accuracy: 0.8184 - val_loss: 0.9135 - val_accuracy: 0.6871
Epoch 339/400
62/62 [====] - 3s 43ms/step - loss: 0.4390 - accuracy: 0.8041 - val_loss: 0.9639 - val_accuracy: 0.7219
Epoch 340/400
62/62 [====] - 3s 43ms/step - loss: 0.4473 - accuracy: 0.8092 - val_loss: 1.2497 - val_accuracy: 0.7382
Epoch 341/400
62/62 [====] - 3s 43ms/step - loss: 0.4568 - accuracy: 0.8082 - val_loss: 0.9312 - val_accuracy: 0.7301
Epoch 342/400
62/62 [====] - 3s 42ms/step - loss: 0.4224 - accuracy: 0.8164 - val_loss: 1.1426 - val_accuracy: 0.7505
Epoch 343/400
62/62 [====] - 3s 43ms/step - loss: 0.4375 - accuracy: 0.8199 - val_loss: 0.8865 - val_accuracy: 0.7362
Epoch 344/400
62/62 [====] - 3s 42ms/step - loss: 0.4221 - accuracy: 0.8102 - val_loss: 1.2509 - val_accuracy: 0.7301
Epoch 345/400
62/62 [====] - 3s 42ms/step - loss: 0.4519 - accuracy: 0.8128 - val_loss: 0.9276 - val_accuracy: 0.6973
Epoch 346/400
62/62 [====] - 3s 43ms/step - loss: 0.4497 - accuracy: 0.8246 - val_loss: 0.7915 - val_accuracy: 0.7403
Epoch 347/400
62/62 [====] - 3s 43ms/step - loss: 0.4395 - accuracy: 0.8266 - val_loss: 0.8395 - val_accuracy: 0.7301
Epoch 348/400
62/62 [====] - 3s 43ms/step - loss: 0.4636 - accuracy: 0.7995 - val_loss: 0.7491 - val_accuracy: 0.7157
Epoch 349/400
62/62 [====] - 3s 43ms/step - loss: 0.4310 - accuracy: 0.8118 - val_loss: 0.7163 - val_accuracy: 0.7526
Epoch 350/400
62/62 [====] - 3s 43ms/step - loss: 0.4455 - accuracy: 0.8118 - val_loss: 0.7755 - val_accuracy: 0.7280
Epoch 351/400
62/62 [====] - 3s 44ms/step - loss: 0.4321 - accuracy: 0.8159 - val_loss: 0.7025 - val_accuracy: 0.7566
Epoch 352/400
62/62 [====] - 3s 43ms/step - loss: 0.4765 - accuracy: 0.8087 - val_loss: 0.8232 - val_accuracy: 0.7198
Epoch 353/400
62/62 [====] - 3s 43ms/step - loss: 0.4592 - accuracy: 0.8179 - val_loss: 0.6478 - val_accuracy: 0.7607
Epoch 354/400
62/62 [====] - 3s 43ms/step - loss: 0.4251 - accuracy: 0.8210 - val_loss: 1.0367 - val_accuracy: 0.6401
Epoch 355/400
62/62 [====] - 3s 43ms/step - loss: 0.4388 - accuracy: 0.8118 - val_loss: 0.8267 - val_accuracy: 0.7444
Epoch 356/400
62/62 [====] - 3s 43ms/step - loss: 0.4718 - accuracy: 0.7918 - val_loss: 0.7724 - val_accuracy: 0.7669
```

```
Epoch 357/400
62/62 [====] - 3s 44ms/step - loss: 0.5713 - accuracy: 0.8066 - val_loss: 0.8973 - val_accuracy: 0.7198
Epoch 358/400
62/62 [====] - 3s 43ms/step - loss: 0.4622 - accuracy: 0.8041 - val_loss: 0.7485 - val_accuracy: 0.7730
Epoch 359/400
62/62 [====] - 3s 42ms/step - loss: 0.4320 - accuracy: 0.8174 - val_loss: 1.0312 - val_accuracy: 0.7362
Epoch 360/400
62/62 [====] - 3s 43ms/step - loss: 0.4342 - accuracy: 0.8133 - val_loss: 0.7880 - val_accuracy: 0.7280
Epoch 361/400
62/62 [====] - 3s 44ms/step - loss: 0.4220 - accuracy: 0.8133 - val_loss: 1.0859 - val_accuracy: 0.7382
Epoch 362/400
62/62 [====] - 3s 42ms/step - loss: 0.4545 - accuracy: 0.8225 - val_loss: 1.2637 - val_accuracy: 0.6769
Epoch 363/400
62/62 [====] - 3s 42ms/step - loss: 0.4369 - accuracy: 0.7995 - val_loss: 0.8208 - val_accuracy: 0.7342
Epoch 364/400
62/62 [====] - 3s 44ms/step - loss: 0.4573 - accuracy: 0.8169 - val_loss: 0.8154 - val_accuracy: 0.7301
Epoch 365/400
62/62 [====] - 3s 43ms/step - loss: 0.4458 - accuracy: 0.8056 - val_loss: 0.7971 - val_accuracy: 0.7260
Epoch 366/400
62/62 [====] - 3s 43ms/step - loss: 0.4400 - accuracy: 0.8215 - val_loss: 0.8720 - val_accuracy: 0.7362
Epoch 367/400
62/62 [====] - 3s 42ms/step - loss: 0.4163 - accuracy: 0.8276 - val_loss: 0.7517 - val_accuracy: 0.7260
Epoch 368/400
62/62 [====] - 3s 43ms/step - loss: 0.4281 - accuracy: 0.8169 - val_loss: 0.8094 - val_accuracy: 0.7219
Epoch 369/400
62/62 [====] - 3s 42ms/step - loss: 0.4480 - accuracy: 0.8138 - val_loss: 0.8843 - val_accuracy: 0.7669
Epoch 370/400
62/62 [====] - 3s 43ms/step - loss: 0.4409 - accuracy: 0.8266 - val_loss: 0.7660 - val_accuracy: 0.7526
Epoch 371/400
62/62 [====] - 3s 43ms/step - loss: 0.4438 - accuracy: 0.8184 - val_loss: 0.7904 - val_accuracy: 0.7423
Epoch 372/400
62/62 [====] - 3s 43ms/step - loss: 0.4495 - accuracy: 0.8143 - val_loss: 0.8173 - val_accuracy: 0.7198
Epoch 373/400
62/62 [====] - 3s 43ms/step - loss: 0.4266 - accuracy: 0.8399 - val_loss: 1.0694 - val_accuracy: 0.7505
Epoch 374/400
62/62 [====] - 3s 43ms/step - loss: 0.4888 - accuracy: 0.8230 - val_loss: 0.7911 - val_accuracy: 0.7096
Epoch 375/400
62/62 [====] - 3s 44ms/step - loss: 0.4601 - accuracy: 0.8056 - val_loss: 0.8111 - val_accuracy: 0.6851
Epoch 376/400
62/62 [====] - 3s 43ms/step - loss: 0.4498 - accuracy: 0.8179 - val_loss: 0.9202 - val_accuracy: 0.7546
Epoch 377/400
62/62 [====] - 3s 43ms/step - loss: 0.4203 - accuracy: 0.8153 - val_loss: 1.1050 - val_accuracy: 0.7628
Epoch 378/400
62/62 [====] - 3s 43ms/step - loss: 0.4329 - accuracy: 0.8128 - val_loss: 0.9038 - val_accuracy: 0.7342
Epoch 379/400
62/62 [====] - 3s 42ms/step - loss: 0.4161 - accuracy: 0.8246 - val_loss: 0.7294 - val_accuracy: 0.7260
Epoch 380/400
62/62 [====] - 3s 43ms/step - loss: 0.4840 - accuracy: 0.8036 - val_loss: 0.6400 - val_accuracy: 0.7526
Epoch 381/400
62/62 [====] - 3s 43ms/step - loss: 0.4567 - accuracy: 0.8097 - val_loss: 0.6827 - val_accuracy: 0.7382
Epoch 382/400
62/62 [====] - 3s 43ms/step - loss: 0.4127 - accuracy: 0.8302 - val_loss: 0.9576 - val_accuracy: 0.7035
Epoch 383/400
62/62 [====] - 3s 43ms/step - loss: 0.4545 - accuracy: 0.8051 - val_loss: 0.8034 - val_accuracy: 0.7137
Epoch 384/400
62/62 [====] - 3s 44ms/step - loss: 0.4361 - accuracy: 0.8148 - val_loss: 1.5285 - val_accuracy: 0.6442
Epoch 385/400
62/62 [====] - 3s 44ms/step - loss: 0.5014 - accuracy: 0.7969 - val_loss: 0.6443 - val_accuracy: 0.7607
Epoch 386/400
62/62 [====] - 3s 43ms/step - loss: 0.4340 - accuracy: 0.8123 - val_loss: 0.9794 - val_accuracy: 0.7260
Epoch 387/400
62/62 [====] - 3s 42ms/step - loss: 0.4470 - accuracy: 0.8169 - val_loss: 0.8243 - val_accuracy: 0.7485
Epoch 388/400
62/62 [====] - 3s 43ms/step - loss: 0.4486 - accuracy: 0.8041 - val_loss: 1.1956 - val_accuracy: 0.7382
Epoch 389/400
62/62 [====] - 3s 43ms/step - loss: 0.4095 - accuracy: 0.8169 - val_loss: 1.0077 - val_accuracy: 0.7669
Epoch 390/400
62/62 [====] - 3s 42ms/step - loss: 0.4531 - accuracy: 0.8189 - val_loss: 0.9953 - val_accuracy: 0.7157
Epoch 391/400
62/62 [====] - 3s 43ms/step - loss: 0.4492 - accuracy: 0.8159 - val_loss: 1.0562 - val_accuracy: 0.7485
Epoch 392/400
62/62 [====] - 3s 43ms/step - loss: 0.4561 - accuracy: 0.8148 - val_loss: 0.7373 - val_accuracy: 0.7342
Epoch 393/400
62/62 [====] - 3s 42ms/step - loss: 0.4094 - accuracy: 0.8246 - val_loss: 0.7967 - val_accuracy: 0.7689
Epoch 394/400
62/62 [====] - 3s 43ms/step - loss: 0.4684 - accuracy: 0.8184 - val_loss: 0.7083 - val_accuracy: 0.7014
Epoch 395/400
62/62 [====] - 3s 44ms/step - loss: 0.4205 - accuracy: 0.8240 - val_loss: 1.0806 - val_accuracy: 0.7157
Epoch 396/400
62/62 [====] - 3s 46ms/step - loss: 0.4383 - accuracy: 0.8317 - val_loss: 0.6082 - val_accuracy: 0.7873
Epoch 397/400
62/62 [====] - 3s 44ms/step - loss: 0.4676 - accuracy: 0.8230 - val_loss: 0.7795 - val_accuracy: 0.7587
```

```
Epoch 398/400
62/62 [====] - 3s 43ms/step - loss: 0.4530 - accuracy: 0.8179 - val_loss: 0.6868 - val_accuracy: 0.7771
Epoch 399/400
62/62 [====] - 3s 43ms/step - loss: 0.4510 - accuracy: 0.8097 - val_loss: 0.8522 - val_accuracy: 0.7280
Epoch 400/400
62/62 [====] - 3s 42ms/step - loss: 0.4602 - accuracy: 0.8240 - val_loss: 0.8338 - val_accuracy: 0.7423
```

Experiment 15

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_64 (Conv2D)          (None, 62, 62, 32)        320
 max_pooling2d_64 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_65 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_65 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_66 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_66 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_67 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_67 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_14 (Flatten)        (None, 256)               0
 dense_28 (Dense)            (None, 64)                16448
 dropout_14 (Dropout)        (None, 64)                0
 dense_29 (Dense)            (None, 2)                 130
=================================================================
Total params: 53,890
Trainable params: 53,890
Non-trainable params: 0
_________________________________________________________________
Epoch 1/35
62/62 [====] - 2s 20ms/step - loss: 0.6944 - accuracy: 0.5018 - val_loss: 0.6933 - val_accuracy: 0.4990
Epoch 2/35
62/62 [====] - 1s 15ms/step - loss: 0.6936 - accuracy: 0.4788 - val_loss: 0.6931 - val_accuracy: 0.5010
Epoch 3/35
62/62 [====] - 1s 14ms/step - loss: 0.6935 - accuracy: 0.4987 - val_loss: 0.6928 - val_accuracy: 0.5092
Epoch 4/35
62/62 [====] - 1s 14ms/step - loss: 0.6932 - accuracy: 0.5182 - val_loss: 0.6935 - val_accuracy: 0.5031
Epoch 5/35
62/62 [====] - 1s 15ms/step - loss: 0.6926 - accuracy: 0.5253 - val_loss: 0.6917 - val_accuracy: 0.5133
Epoch 6/35
62/62 [====] - 1s 15ms/step - loss: 0.6908 - accuracy: 0.5335 - val_loss: 0.6937 - val_accuracy: 0.5133
Epoch 7/35
62/62 [====] - 1s 14ms/step - loss: 0.6845 - accuracy: 0.5642 - val_loss: 0.6864 - val_accuracy: 0.5235
Epoch 8/35
62/62 [====] - 1s 14ms/step - loss: 0.6745 - accuracy: 0.5826 - val_loss: 0.7239 - val_accuracy: 0.5072
Epoch 9/35
62/62 [====] - 1s 14ms/step - loss: 0.6694 - accuracy: 0.5974 - val_loss: 0.6905 - val_accuracy: 0.5501
Epoch 10/35
62/62 [====] - 1s 15ms/step - loss: 0.6619 - accuracy: 0.6026 - val_loss: 0.6583 - val_accuracy: 0.6033
Epoch 11/35
62/62 [====] - 1s 14ms/step - loss: 0.6418 - accuracy: 0.6384 - val_loss: 0.7036 - val_accuracy: 0.5337
Epoch 12/35
62/62 [====] - 1s 15ms/step - loss: 0.6489 - accuracy: 0.6292 - val_loss: 0.6488 - val_accuracy: 0.6074
Epoch 13/35
62/62 [====] - 1s 15ms/step - loss: 0.6371 - accuracy: 0.6404 - val_loss: 0.8046 - val_accuracy: 0.5072
Epoch 14/35
62/62 [====] - 1s 15ms/step - loss: 0.6359 - accuracy: 0.6373 - val_loss: 0.9157 - val_accuracy: 0.4888
Epoch 15/35
62/62 [====] - 1s 14ms/step - loss: 0.6263 - accuracy: 0.6501 - val_loss: 0.7507 - val_accuracy: 0.5440
Epoch 16/35
62/62 [====] - 1s 15ms/step - loss: 0.6142 - accuracy: 0.6629 - val_loss: 0.6995 - val_accuracy: 0.5910
Epoch 17/35
62/62 [====] - 1s 14ms/step - loss: 0.6124 - accuracy: 0.6721 - val_loss: 0.7466 - val_accuracy: 0.5399
Epoch 18/35
62/62 [====] - 1s 14ms/step - loss: 0.6036 - accuracy: 0.6849 - val_loss: 0.6938 - val_accuracy: 0.5767
Epoch 19/35
62/62 [====] - 1s 15ms/step - loss: 0.5927 - accuracy: 0.6721 - val_loss: 0.6091 - val_accuracy: 0.6789
Epoch 20/35
62/62 [====] - 1s 15ms/step - loss: 0.5729 - accuracy: 0.6967 - val_loss: 0.7631 - val_accuracy: 0.5726
Epoch 21/35
62/62 [====] - 1s 14ms/step - loss: 0.5749 - accuracy: 0.7130 - val_loss: 0.5802 - val_accuracy: 0.6667
Epoch 22/35
62/62 [====] - 1s 15ms/step - loss: 0.5542 - accuracy: 0.7161 - val_loss: 0.9039 - val_accuracy: 0.5276
```

```
Epoch 23/35
62/62 [====] - 1s 15ms/step - loss: 0.5593 - accuracy: 0.7228 - val_loss: 0.8728 - val_accuracy: 0.5276
Epoch 24/35
62/62 [====] - 1s 14ms/step - loss: 0.5378 - accuracy: 0.7299 - val_loss: 0.7197 - val_accuracy: 0.5849
Epoch 25/35
62/62 [====] - 1s 14ms/step - loss: 0.5199 - accuracy: 0.7448 - val_loss: 0.5746 - val_accuracy: 0.6708
Epoch 26/35
62/62 [====] - 1s 14ms/step - loss: 0.5054 - accuracy: 0.7494 - val_loss: 0.5486 - val_accuracy: 0.7117
Epoch 27/35
62/62 [====] - 1s 13ms/step - loss: 0.4720 - accuracy: 0.7693 - val_loss: 0.5527 - val_accuracy: 0.7301
Epoch 28/35
62/62 [====] - 1s 13ms/step - loss: 0.4457 - accuracy: 0.7964 - val_loss: 0.6540 - val_accuracy: 0.6442
Epoch 29/35
62/62 [====] - 1s 15ms/step - loss: 0.4393 - accuracy: 0.7913 - val_loss: 0.6044 - val_accuracy: 0.6871
Epoch 30/35
62/62 [====] - 1s 13ms/step - loss: 0.4319 - accuracy: 0.7913 - val_loss: 0.5282 - val_accuracy: 0.7362
Epoch 31/35
62/62 [====] - 1s 14ms/step - loss: 0.3995 - accuracy: 0.8020 - val_loss: 0.5037 - val_accuracy: 0.7444
Epoch 32/35
62/62 [====] - 1s 15ms/step - loss: 0.3993 - accuracy: 0.8153 - val_loss: 0.5467 - val_accuracy: 0.7096
Epoch 33/35
62/62 [====] - 1s 13ms/step - loss: 0.3571 - accuracy: 0.8358 - val_loss: 0.9884 - val_accuracy: 0.5706
Epoch 34/35
62/62 [====] - 1s 15ms/step - loss: 0.3584 - accuracy: 0.8246 - val_loss: 0.5358 - val_accuracy: 0.7587
Epoch 35/35
62/62 [====] - 1s 13ms/step - loss: 0.3561 - accuracy: 0.8450 - val_loss: 0.6648 - val_accuracy: 0.6462
```

Experiment 16

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_68 (Conv2D)          (None, 62, 62, 32)        320
 max_pooling2d_68 (MaxPooling2D) (None, 31, 31, 32)    0
 conv2d_69 (Conv2D)          (None, 29, 29, 32)        9248
 max_pooling2d_69 (MaxPooling2D) (None, 14, 14, 32)    0
 conv2d_70 (Conv2D)          (None, 12, 12, 32)        9248
 max_pooling2d_70 (MaxPooling2D) (None, 6, 6, 32)      0
 conv2d_71 (Conv2D)          (None, 4, 4, 64)          18496
 max_pooling2d_71 (MaxPooling2D) (None, 2, 2, 64)      0
 flatten_15 (Flatten)        (None, 256)               0
 dense_30 (Dense)            (None, 64)                16448
 dropout_15 (Dropout)        (None, 64)                0
 dense_31 (Dense)            (None, 2)                 130
=================================================================
Total params: 53,890
Trainable params: 53,890
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
62/62 [====] - 4s 48ms/step - loss: 0.8730 - accuracy: 0.5033 - val_loss: 0.7012 - val_accuracy: 0.4990
Epoch 2/100
62/62 [====] - 3s 41ms/step - loss: 0.6994 - accuracy: 0.4905 - val_loss: 0.6924 - val_accuracy: 0.4969
Epoch 3/100
62/62 [====] - 2s 40ms/step - loss: 0.6965 - accuracy: 0.5090 - val_loss: 0.6950 - val_accuracy: 0.5051
Epoch 4/100
62/62 [====] - 2s 40ms/step - loss: 0.6975 - accuracy: 0.5105 - val_loss: 0.6916 - val_accuracy: 0.5399
Epoch 5/100
62/62 [====] - 2s 40ms/step - loss: 0.6960 - accuracy: 0.5238 - val_loss: 0.6974 - val_accuracy: 0.5092
Epoch 6/100
62/62 [====] - 2s 40ms/step - loss: 0.6974 - accuracy: 0.5233 - val_loss: 0.6959 - val_accuracy: 0.4990
Epoch 7/100
62/62 [====] - 2s 40ms/step - loss: 0.6963 - accuracy: 0.5008 - val_loss: 0.6916 - val_accuracy: 0.5215
Epoch 8/100
62/62 [====] - 2s 40ms/step - loss: 0.6928 - accuracy: 0.5166 - val_loss: 0.6956 - val_accuracy: 0.5092
Epoch 9/100
62/62 [====] - 2s 40ms/step - loss: 0.6962 - accuracy: 0.5166 - val_loss: 0.6953 - val_accuracy: 0.4785
Epoch 10/100
62/62 [====] - 2s 40ms/step - loss: 0.6936 - accuracy: 0.5279 - val_loss: 0.6933 - val_accuracy: 0.5010
Epoch 11/100
62/62 [====] - 2s 40ms/step - loss: 0.6935 - accuracy: 0.5258 - val_loss: 0.6917 - val_accuracy: 0.5010
Epoch 12/100
62/62 [====] - 3s 41ms/step - loss: 0.6914 - accuracy: 0.5325 - val_loss: 0.6947 - val_accuracy: 0.5501
```

```
Epoch 13/100
62/62 [====] - 3s 40ms/step - loss: 0.6922 - accuracy: 0.5396 - val_loss: 0.6912 - val_accuracy: 0.5378
Epoch 14/100
62/62 [====] - 2s 40ms/step - loss: 0.6894 - accuracy: 0.5565 - val_loss: 0.6913 - val_accuracy: 0.5031
Epoch 15/100
62/62 [====] - 3s 41ms/step - loss: 0.6948 - accuracy: 0.5611 - val_loss: 0.6915 - val_accuracy: 0.5767
Epoch 16/100
62/62 [====] - 2s 40ms/step - loss: 0.6849 - accuracy: 0.5785 - val_loss: 0.6963 - val_accuracy: 0.5031
Epoch 17/100
62/62 [====] - 2s 40ms/step - loss: 0.6850 - accuracy: 0.5621 - val_loss: 0.6870 - val_accuracy: 0.5215
Epoch 18/100
62/62 [====] - 3s 41ms/step - loss: 0.6864 - accuracy: 0.5570 - val_loss: 0.6848 - val_accuracy: 0.5481
Epoch 19/100
62/62 [====] - 3s 41ms/step - loss: 0.6809 - accuracy: 0.5714 - val_loss: 0.6860 - val_accuracy: 0.5215
Epoch 20/100
62/62 [====] - 3s 40ms/step - loss: 0.6844 - accuracy: 0.5734 - val_loss: 0.6930 - val_accuracy: 0.5215
Epoch 21/100
62/62 [====] - 2s 40ms/step - loss: 0.6806 - accuracy: 0.5719 - val_loss: 0.6836 - val_accuracy: 0.5440
Epoch 22/100
62/62 [====] - 2s 40ms/step - loss: 0.6774 - accuracy: 0.5637 - val_loss: 0.6999 - val_accuracy: 0.5194
Epoch 23/100
62/62 [====] - 2s 39ms/step - loss: 0.6782 - accuracy: 0.5734 - val_loss: 0.6788 - val_accuracy: 0.5726
Epoch 24/100
62/62 [====] - 2s 40ms/step - loss: 0.6677 - accuracy: 0.5944 - val_loss: 0.6971 - val_accuracy: 0.5583
Epoch 25/100
62/62 [====] - 3s 40ms/step - loss: 0.6796 - accuracy: 0.5770 - val_loss: 0.6832 - val_accuracy: 0.5562
Epoch 26/100
62/62 [====] - 3s 41ms/step - loss: 0.6756 - accuracy: 0.5867 - val_loss: 0.6853 - val_accuracy: 0.5603
Epoch 27/100
62/62 [====] - 3s 41ms/step - loss: 0.6692 - accuracy: 0.5923 - val_loss: 0.6842 - val_accuracy: 0.5787
Epoch 28/100
62/62 [====] - 2s 40ms/step - loss: 0.6698 - accuracy: 0.5893 - val_loss: 0.6759 - val_accuracy: 0.5787
Epoch 29/100
62/62 [====] - 3s 41ms/step - loss: 0.6639 - accuracy: 0.6041 - val_loss: 0.6798 - val_accuracy: 0.6012
Epoch 30/100
62/62 [====] - 3s 41ms/step - loss: 0.6650 - accuracy: 0.5980 - val_loss: 0.6874 - val_accuracy: 0.5521
Epoch 31/100
62/62 [====] - 3s 40ms/step - loss: 0.6726 - accuracy: 0.5780 - val_loss: 0.6992 - val_accuracy: 0.5358
Epoch 32/100
62/62 [====] - 3s 40ms/step - loss: 0.6579 - accuracy: 0.6148 - val_loss: 0.6933 - val_accuracy: 0.5603
Epoch 33/100
62/62 [====] - 2s 40ms/step - loss: 0.6650 - accuracy: 0.5964 - val_loss: 0.7083 - val_accuracy: 0.5297
Epoch 34/100
62/62 [====] - 2s 40ms/step - loss: 0.6552 - accuracy: 0.6138 - val_loss: 0.6974 - val_accuracy: 0.5542
Epoch 35/100
62/62 [====] - 3s 40ms/step - loss: 0.6573 - accuracy: 0.6210 - val_loss: 0.6915 - val_accuracy: 0.5603
Epoch 36/100
62/62 [====] - 2s 40ms/step - loss: 0.6531 - accuracy: 0.6072 - val_loss: 0.6775 - val_accuracy: 0.5746
Epoch 37/100
62/62 [====] - 3s 41ms/step - loss: 0.6619 - accuracy: 0.6235 - val_loss: 0.6766 - val_accuracy: 0.6196
Epoch 38/100
62/62 [====] - 2s 40ms/step - loss: 0.6607 - accuracy: 0.6199 - val_loss: 0.6571 - val_accuracy: 0.6421
Epoch 39/100
62/62 [====] - 2s 40ms/step - loss: 0.6564 - accuracy: 0.6322 - val_loss: 0.7102 - val_accuracy: 0.5317
Epoch 40/100
62/62 [====] - 2s 40ms/step - loss: 0.6494 - accuracy: 0.6297 - val_loss: 0.6978 - val_accuracy: 0.6155
Epoch 41/100
62/62 [====] - 3s 41ms/step - loss: 0.6476 - accuracy: 0.6358 - val_loss: 0.6746 - val_accuracy: 0.6012
Epoch 42/100
62/62 [====] - 3s 41ms/step - loss: 0.6495 - accuracy: 0.6322 - val_loss: 0.6616 - val_accuracy: 0.6626
Epoch 43/100
62/62 [====] - 3s 40ms/step - loss: 0.6398 - accuracy: 0.6450 - val_loss: 0.6672 - val_accuracy: 0.5869
Epoch 44/100
62/62 [====] - 3s 41ms/step - loss: 0.6437 - accuracy: 0.6394 - val_loss: 0.6920 - val_accuracy: 0.6401
Epoch 45/100
62/62 [====] - 3s 40ms/step - loss: 0.6221 - accuracy: 0.6578 - val_loss: 0.6780 - val_accuracy: 0.6421
Epoch 46/100
62/62 [====] - 3s 40ms/step - loss: 0.6442 - accuracy: 0.6527 - val_loss: 0.6694 - val_accuracy: 0.5849
Epoch 47/100
62/62 [====] - 3s 40ms/step - loss: 0.6358 - accuracy: 0.6440 - val_loss: 0.6625 - val_accuracy: 0.6237
Epoch 48/100
62/62 [====] - 3s 40ms/step - loss: 0.6304 - accuracy: 0.6598 - val_loss: 0.6562 - val_accuracy: 0.6278
Epoch 49/100
62/62 [====] - 3s 40ms/step - loss: 0.6242 - accuracy: 0.6532 - val_loss: 0.6900 - val_accuracy: 0.6135
Epoch 50/100
62/62 [====] - 3s 41ms/step - loss: 0.6213 - accuracy: 0.6706 - val_loss: 0.6487 - val_accuracy: 0.6524
Epoch 51/100
62/62 [====] - 3s 41ms/step - loss: 0.6159 - accuracy: 0.6772 - val_loss: 0.6421 - val_accuracy: 0.6503
Epoch 52/100
62/62 [====] - 3s 40ms/step - loss: 0.6106 - accuracy: 0.6783 - val_loss: 0.7026 - val_accuracy: 0.5501
Epoch 53/100
62/62 [====] - 3s 41ms/step - loss: 0.6148 - accuracy: 0.6645 - val_loss: 0.7469 - val_accuracy: 0.5665
```

```
Epoch 54/100
62/62 [====] - 3s 41ms/step - loss: 0.6205 - accuracy: 0.6721 - val_loss: 0.6239 - val_accuracy: 0.6769
Epoch 55/100
62/62 [====] - 3s 41ms/step - loss: 0.6190 - accuracy: 0.6573 - val_loss: 0.6982 - val_accuracy: 0.6196
Epoch 56/100
62/62 [====] - 3s 41ms/step - loss: 0.6079 - accuracy: 0.6849 - val_loss: 0.7014 - val_accuracy: 0.6483
Epoch 57/100
62/62 [====] - 3s 41ms/step - loss: 0.6151 - accuracy: 0.6731 - val_loss: 0.6733 - val_accuracy: 0.6258
Epoch 58/100
62/62 [====] - 3s 40ms/step - loss: 0.6182 - accuracy: 0.6701 - val_loss: 0.6941 - val_accuracy: 0.5706
Epoch 59/100
62/62 [====] - 3s 41ms/step - loss: 0.6246 - accuracy: 0.6665 - val_loss: 0.6384 - val_accuracy: 0.6503
Epoch 60/100
62/62 [====] - 3s 41ms/step - loss: 0.6056 - accuracy: 0.6660 - val_loss: 0.6229 - val_accuracy: 0.6994
Epoch 61/100
62/62 [====] - 3s 41ms/step - loss: 0.5994 - accuracy: 0.6849 - val_loss: 0.6758 - val_accuracy: 0.6319
Epoch 62/100
62/62 [====] - 3s 41ms/step - loss: 0.5924 - accuracy: 0.6813 - val_loss: 0.6549 - val_accuracy: 0.6646
Epoch 63/100
62/62 [====] - 3s 41ms/step - loss: 0.5832 - accuracy: 0.6982 - val_loss: 0.6968 - val_accuracy: 0.5624
Epoch 64/100
62/62 [====] - 3s 41ms/step - loss: 0.5934 - accuracy: 0.6880 - val_loss: 0.6331 - val_accuracy: 0.6851
Epoch 65/100
62/62 [====] - 3s 40ms/step - loss: 0.5893 - accuracy: 0.6946 - val_loss: 0.6998 - val_accuracy: 0.6319
Epoch 66/100
62/62 [====] - 3s 42ms/step - loss: 0.5816 - accuracy: 0.6957 - val_loss: 0.6381 - val_accuracy: 0.6830
Epoch 67/100
62/62 [====] - 3s 41ms/step - loss: 0.5778 - accuracy: 0.7028 - val_loss: 0.7121 - val_accuracy: 0.6564
Epoch 68/100
62/62 [====] - 3s 41ms/step - loss: 0.5802 - accuracy: 0.7003 - val_loss: 0.6683 - val_accuracy: 0.6483
Epoch 69/100
62/62 [====] - 3s 41ms/step - loss: 0.5633 - accuracy: 0.7120 - val_loss: 0.6736 - val_accuracy: 0.6605
Epoch 70/100
62/62 [====] - 2s 40ms/step - loss: 0.5847 - accuracy: 0.7028 - val_loss: 0.6798 - val_accuracy: 0.6380
Epoch 71/100
62/62 [====] - 3s 41ms/step - loss: 0.5576 - accuracy: 0.7243 - val_loss: 0.7126 - val_accuracy: 0.6380
Epoch 72/100
62/62 [====] - 3s 41ms/step - loss: 0.5805 - accuracy: 0.7028 - val_loss: 0.6550 - val_accuracy: 0.6626
Epoch 73/100
62/62 [====] - 3s 41ms/step - loss: 0.5714 - accuracy: 0.7054 - val_loss: 0.6769 - val_accuracy: 0.6462
Epoch 74/100
62/62 [====] - 2s 40ms/step - loss: 0.5740 - accuracy: 0.7207 - val_loss: 0.6925 - val_accuracy: 0.6462
Epoch 75/100
62/62 [====] - 3s 42ms/step - loss: 0.5713 - accuracy: 0.7115 - val_loss: 0.6500 - val_accuracy: 0.6973
Epoch 76/100
62/62 [====] - 3s 40ms/step - loss: 0.5846 - accuracy: 0.6916 - val_loss: 0.6771 - val_accuracy: 0.6544
Epoch 77/100
62/62 [====] - 3s 41ms/step - loss: 0.5623 - accuracy: 0.7074 - val_loss: 0.6944 - val_accuracy: 0.6585
Epoch 78/100
62/62 [====] - 3s 41ms/step - loss: 0.5506 - accuracy: 0.7248 - val_loss: 0.6135 - val_accuracy: 0.7014
Epoch 79/100
62/62 [====] - 3s 40ms/step - loss: 0.5793 - accuracy: 0.7120 - val_loss: 0.7432 - val_accuracy: 0.6094
Epoch 80/100
62/62 [====] - 3s 40ms/step - loss: 0.5582 - accuracy: 0.7171 - val_loss: 0.7580 - val_accuracy: 0.6299
Epoch 81/100
62/62 [====] - 3s 41ms/step - loss: 0.5545 - accuracy: 0.7238 - val_loss: 0.6914 - val_accuracy: 0.6380
Epoch 82/100
62/62 [====] - 3s 41ms/step - loss: 0.5555 - accuracy: 0.7197 - val_loss: 0.6674 - val_accuracy: 0.6380
Epoch 83/100
62/62 [====] - 3s 41ms/step - loss: 0.5427 - accuracy: 0.7309 - val_loss: 0.7585 - val_accuracy: 0.5890
Epoch 84/100
62/62 [====] - 3s 41ms/step - loss: 0.5565 - accuracy: 0.7202 - val_loss: 0.6137 - val_accuracy: 0.6912
Epoch 85/100
62/62 [====] - 3s 40ms/step - loss: 0.5711 - accuracy: 0.7136 - val_loss: 0.7093 - val_accuracy: 0.6626
Epoch 86/100
62/62 [====] - 3s 41ms/step - loss: 0.5539 - accuracy: 0.7340 - val_loss: 0.8257 - val_accuracy: 0.6339
Epoch 87/100
62/62 [====] - 3s 40ms/step - loss: 0.5621 - accuracy: 0.7248 - val_loss: 0.6783 - val_accuracy: 0.6687
Epoch 88/100
62/62 [====] - 3s 41ms/step - loss: 0.5414 - accuracy: 0.7366 - val_loss: 0.6995 - val_accuracy: 0.7055
Epoch 89/100
62/62 [====] - 3s 41ms/step - loss: 0.5507 - accuracy: 0.7330 - val_loss: 0.6606 - val_accuracy: 0.6953
Epoch 90/100
62/62 [====] - 3s 41ms/step - loss: 0.5385 - accuracy: 0.7248 - val_loss: 0.6821 - val_accuracy: 0.6176
Epoch 91/100
62/62 [====] - 3s 40ms/step - loss: 0.5611 - accuracy: 0.7043 - val_loss: 0.7045 - val_accuracy: 0.6360
Epoch 92/100
62/62 [====] - 3s 40ms/step - loss: 0.5458 - accuracy: 0.7299 - val_loss: 0.6757 - val_accuracy: 0.6115
Epoch 93/100
62/62 [====] - 3s 40ms/step - loss: 0.5607 - accuracy: 0.7166 - val_loss: 0.7157 - val_accuracy: 0.6585
Epoch 94/100
62/62 [====] - 2s 40ms/step - loss: 0.5461 - accuracy: 0.7361 - val_loss: 0.8324 - val_accuracy: 0.6585
```

```
Epoch 95/100
62/62 [====] - 2s 40ms/step - loss: 0.5440 - accuracy: 0.7217 - val_loss: 0.6885 - val_accuracy: 0.7076
Epoch 96/100
62/62 [====] - 3s 41ms/step - loss: 0.5390 - accuracy: 0.7304 - val_loss: 0.7243 - val_accuracy: 0.6851
Epoch 97/100
62/62 [====] - 2s 40ms/step - loss: 0.5376 - accuracy: 0.7320 - val_loss: 0.6791 - val_accuracy: 0.6892
Epoch 98/100
62/62 [====] - 3s 40ms/step - loss: 0.5905 - accuracy: 0.7192 - val_loss: 0.6260 - val_accuracy: 0.6789
Epoch 99/100
62/62 [====] - 3s 40ms/step - loss: 0.5699 - accuracy: 0.7130 - val_loss: 0.7518 - val_accuracy: 0.6278
Epoch 100/100
62/62 [====] - 3s 41ms/step - loss: 0.5599 - accuracy: 0.7202 - val_loss: 0.6163 - val_accuracy: 0.6810
```