Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Área: Minería de Datos y Machine Learning

Object Recognition in Images
A Deep Learning Approach

Autor: Miguel Andrés Rodríguez Olmos

Tutor: Anna Bosch Rué

Profesor: Jordi Casas Roma

Barcelona, 30 de Diciembre de 2018

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Object Recognition in Images. A Deep Learning Approach.
Nombre del autor:	Miguel Andrés Rodríguez Olmos
Nombre del colaborador/a docente:	Anna Bosch Rué
Nombre del PRA:	Jordi Casas Roma
Fecha de entrega (mm/aaaa):	01/2019
Titulación o programa:	Master en Data Science
Área del Trabajo Final:	Minería de Datos y Machine Learning
Idioma del trabajo:	Inglés
Palabras clave	computer vision, deep learning, transfer learning

Abstract

Abstract
We employ methods from deep learning for image recognition. We use a dataset with +70k images and 73 classes in order to compare the performance of several well known deep network architectures. The approaches used include the full training of these networks and also the techniques of transfer learning and fine tuning with the weights pretrained on the ImageNet set. We show the superiority of the latter approach in our dataset. We also experiment with a reorganization of the labels in our dataset by grouping several classes shown by the confusion matrix to be indistinguishable for the models. In this case we obtain a classification accuracy score higher than 50%.

Keywords: computer vision, convolutional neural network, deep learning, transfer learning.

Resumen
Utilizamos métodos de aprendizaje profundo en el contexto de reconocimiento de imágenes. Utilizamos un conjunto de más de 70 mil imágenes y 73 clases diferentes para comparar el rendimiento de diferentes arquitecturas comunmente empleadas. Los enfoques utilizados en este trabajo incluyen en un entrenamiento completo de estas redes y también estrategias de transferencia del aprendizaje y calibración de modelos con redes preentrenadas en el conjunto ImageNet. Mostramos la superioridad de este ultimo enfoque en nuestro conjunto de imágenes. También experimentamos con una reorganizacón de las categorías de nuestro conjunto, fusionando aquellas categorías que la matriz de confusión muestra que son más comunmente confundidas por nuestros modelos. En este caso obtenemos una precisión superior al 50%.

Palabras clave: computer vision, convolutional neural network, deep learning, transfer learning.

Contents

List of Figures

Chapter 1

Preface

1.1 Introduction

In this work we apply Deep Learning techniques to a recognition problem in Computer Vision. The goal of this project is to develop and train deep network architectures for recognizing objects in a dataset consisting of 73 labelled categories of images commonly found in hygiene and beauty retail stores. We will refer to it as the Beauty dataset. Our approach consists in applying transfer learning and fine tuning to some well known deep neural network architectures. We will also conduct a complete training of some of these models, with full random initialization of their weights. The different methodologies will be evaluated in terms of performance and compared among them.

For the technical side, due to the inherent demanding nature of deep neural networks, it has been necessary to use a GPU. Therefore the whole process has been carried out on a Google Cloud virtual machine with 4 CPU cores, 15GB of RAM memory, 250GB of disk space and a Tesla K80 GPU. We have used the Keras API to the TensorFlow backend and the code has been written in Python using Jupyter notebooks. A big part of the code used in this work has been adapted from or inspired by [1].

This work is organized as follows: The remainder of this chapter is devoted to a high level, self contained explanation of the general framework, usual practices and state of the art about which this project revolves. Chapter 2 introduces the Beauty dataset and shows how to organize its folder structure in such a way that the analysis can be performed on it. In particular, we set up different sets for training, validation and test purposes. Chapter 3 introduces the main concepts and features of the InceptionV3 architecture, which is the main tool used in this work for doing transfer learning and fine tuning with pretrained deep networks. In Chapter 4 we apply transfer learning and fine tuning to the Beauty dataset using a pretrained InceptionV3 network. We do this in several stages. In Chapter 5 we analyze the confusion matrix for the best classification model obtained so far and regroup those classes more commonly mistaken by the algorithm, obtaining a new dataset with the same images but with 42 labels instead of 73. The same training process is then applied to this new dataset. In Chapter 6 we train several other networks from scratch, with full weight initialization and no transfer learning. We also do a comparison of several performance scores across all the different models. As the final step of the learning process, we obtain a real life accuracy estimate on the test set.

Finally, the conclusions of the whole research project are summarized in Chapter 7. The full output of all the code used for the learning and evaluation processes of all the models is collected in the Appendix, for future reference and its potential interest for possible future extensions of our work.

1.2 Preliminaries

Object recognition (OR) is one of the main applications of the field of Computer Vision (CV). The main idea underlying this discipline is to automatically extract from digital 2D images (or video) information about the different objects present in the image, therefore performing a classification task, in which each image is labelled with the object or objects showed, without human intervention. This is part of the interdisciplinary field of CV, that has evolved parallel to developments in the engineering and technological applications to image treatment, as well as (more recently) to developments in Artificial Intelligence, and more importantly Deep Learning (DL).

There are several reasons and advances behind this evolution. Some of the most important ones are:

(i) The availability of larger datasets already labelled and available for public use. These have been intensively used for research in CV and OR and typically used as benchmarks for testing the superiority of different models and algorithms. Some of the most important ones are

   – MNIST dataset [2], consisting on 70000 greyscale images representing handwritten numeric digits. This is also one of the most widely studied datasets on which OR techniques, and most notably DL approaches have superseded human performance.

   – Imagenet [3] a database with over 14 million images from every aspect of the visual world, which aims to be a repository for creating models that can recognize every possible image.

   – Cifar10/100 [4], consisting on 60000 color images divided in 10 or 100 categories. It has been developed at the university of Toronto and is extensively used for prototyping and benchmarking new network architectures.

(ii) More powerful CPU's and the introduction of GPU's and parallel computing on commodity hardware. This allowed to the implementation of previously existing algorithms that were not physically applicable due to the computing limitations of the available technology at the time.

(iii) Theoretical advances in the design of algorithms that can perform automatic image classification. In particular, the introduction of convolutional neural networks, together with their regularization methods and the implementation of backpropagation has allowed to implement in practice previously intractable neural network architectures.

1.3 Traditional Computer Vision

Traditional methods for computer vision (i.e. before the advent of DL) differ fundamentally from the modern machine learning approach for image recognition. Although the subject of computer vision is very wide we can identify the two main differences in which these traditional methods diverge from DL: feature extraction and preprocessing. These are routine tasks in computer vision (including shallow machine learning algorithms like support vector machines) that are inherently performed by human beings on a problem by problem basis essentially converting each image recognition task in a custom project.

Feature extraction is a routine that is part of every classification or regression problem attacked by methods not based in deep learning. These include most machine learning algorithms in classification, regression and unsupervised learning. In the case of computer vision feature extraction consists of manually instructing the software to recognize visual elements like corners, edges, etc. . . in order to later use these elements as tags in each image, which will then be fed to a classification algorithm. Preprocessing is a very general paradigm that consists of feeding the software or algorithm with good quality data.

Preprocessing includes manually assessing the quality of the original data in raw form and performing a series of tasks like rescaling, cleaning, standardizing, as well as some actions of statistical nature (correcting bias in variable distributions, treating outliers and missing values etc. . .). In the field of traditional computer vision preprocessing consists of a series of well defined actions like resampling, noise reduction, contrast enhancement and scale space [5]. Again, these are performed on a case by case basis. As we shall see in the next paragraph, the introduction of DL allows, to a great extent, to reduce the efforts in feature extraction and preprocessing, also providing impressive success in image recognition tasks, which many times go beyond the human recognition ability on the same problem.

1.4 Artificial Neural Networks

The methods employed in this work belong to the the field of DL, which is a particular case of the set of algorithms based on artificial neural networks (ANN). ANN were introduced theoretically in the 1940's as predictive models (regression and classification) loosely inspired by the interconnection of neurons in the human brain, see [6] for a survey. Among the different approaches proposed, the nonlinear perceptron algorithm [7] can be considered as the predecessor of modern ANN and DL in general. The simplest model of perceptron consists of a unique neuron that maps a dataset with m rows (observations) and n columns (variables) to a m-dimensional vector with components taking values in the interval $[0, 1]$. It works by taking a linear combination of all the elements in each row of the dataset, adding a constant term (bias) and then passing the result to a nonlinear real function of one variable g as follows:

Let $\mathbf{x^i} = (x^i_1, \ldots, x^i_n)$ be the i-th observation. The parameters of the perceptron are then b^i (the bias term) and $\mathbf{w^i} = (w^i_1, \ldots, w^i_n)$ (the weights vector). Then, the perceptron performs the composite operation

$$\mathbf{x^i} \mapsto g(b^i + \mathbf{w^i} \cdot \mathbf{x^i}).$$

Notice that the perceptron consists of an affine transformation (which of course can be thought as a linear map on a $n + 1$ linear space) followed by a nonlinearity g. In case g is the sigmoid (logistic) function the nonlinear perceptron is equivalent to the classic logistic regression algorithm. However, the power of this approach is that there could be many neurons (nodes in the following) in the perceptron running the same operations but each with different weights and biases, forming a layer. Also there could be a concatenation of several layers, finally arriving to the contemporary architecture known as the multilayer perceptron.

Notice that a multilayer perceptron is nothing else than a more or less complicated composition of linear and nonlinear operations in which the coefficients, or weights, of the linear maps involved are free parameters of the model. These must be "learned" by the algorithm by an optimization process, typically based on the numeric search of the local minimum of a cost function with low enough total value. This process for getting the optimal parameters is usually known as automatic learning and the system is said to learn from the examples or experience (which is just the dataset fed to the algorithm). These cost functions come in many flavors but they don't differ in principle from those used in other classic optimization contexts, i.e. mean squared error or mean absolute error for regression problems with real outcome, binary cross-entropy for regression problems with binary output (classification) or categorical cross-entropy for linear regression problems with discrete output (multilabel classification).

A great deal of effort has been put in the nature of the nonlinearities used in multilayer perceptrons and, besides sigmoid functions, other activation functions like relu or softmax are widely used at this time. Also, the non-convexity of the cost function in multilayer perceptrons has motivated a lot of research directed towards developing good optimization algorithms for this task. Besides the backpropagation algorithm previously mentioned, optimizers like the stochastic gradient descent or the adam algorithm are ubiquitous in models based in ANN. Collectively, those ANN architectures with more than one layer are referred to as Deep Learning systems.

An important breakthrough happened when it was realized that DNN systems could be specially effective for solving tasks based in the paradigm of "perception problems" instead of being used for traditional function approximation as most traditional ML algorithms do. Perception problems are those for which the solution must incorporate the way a human would interpret the problem, and they cannot be solved in an intrinsic way which ignores the human influence in the process. Typical examples include image and text recognition. These are perception problems because it is essential how the data are organized and perceived by humans in order to be able to attack the problem. For instance, we cannot arbitrarily reorganize the pixels in an image of a house and expect that anyone would still recognize a house in the image. Typically, as is the case for images and text, perception problems involve non-structured data, which cannot be presented in a tabular fashion or at least this is not the optimal way to solve them.

In the field of computer vision one of the main problems is that each observation consists of a lot of data. Let us take for instance a color image of dimension (in pixels) 100x100. This implies that its representation actually contains 100 x 100 x 3 = 30000 variables, each of them representing the intensity of each color channel for each pixel. Notice that for any ML algorithm to have a decent chance of working with this kind of images, the dataset to be used, if represented in tabular form, should have a number of rows $m \gg n = 30000$. In practice, much more are needed since the number of parameters in a DL system grows exponentially with the complexity of the architecture so there is clearly a problem in using the multilayer perceptron for tasks related to images. At this point is where the introduction of convolutional neural networks (CNN) steps in. Without going too far into technical details, a CNN architecture is a particular case of DNN systems with two main properties:

(i) Multiple nodes share *all* of their parameters, resulting in more tractable models which are easier to train and that also require significant less data to converge or to avoid overfitting and underfitting during the optimization process.

(ii) The architectures of CNN models are designed to solve iteratively an image problem in a human fashion way. Specifically, the parameters of the nodes are learned in such a way that each layer is able to recognize geometric and visual objects in the images (corners, curves, faces, etc...) in a translational and rotational (among other transformations) invariant manner. Also, as we go deeper into the successive layers of the system, the partial objects recognized by the network get increasingly more abstract and more oriented to the specific particularities of the set of images under study.

1.5 Succesful Architectures

Due to the trial and error nature of the process of designing good CNN architectures for image recognition, there several models that are considered as specially successful. Most of the time the reason for this is that they have obtained a high classification accuracy in the Imagenet dataset, which is supposed to be a representation of our visual world, and therefore these architectures are closer to the ideal of a universal algorithm for image recognition. We will not enter into technical details of how all these models differ among them but only name a few important ones in chronological order together with their error rate (1-accuracy) on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition. See [8] for more in-depth review of these architectures.

- AlexNet (2012). Error rate: 0.153. This was the first time a CNN architecture won the ILSVRC.
- VGG16 (2015). Error rate: 0.073.
- InceptionV1 (2014). Error rate: 0.067.
- InceptionV2 (2015). Error rate: 0.056.
- InceptionV3 (2015). Error rate: 0.0358.
- ResNet (2015). Error rate: 0.0357.
- InceptionV4 (Inception-ResNet) (2016). Error rate: 0.0308.
- Squeeze-and-excitation (2017). Error rate: 0.0225.

1.6 Transfer Learning

The concept of transfer learning is rooted in one of the main drawbacks of CNN architectures: they are prone to overfitting. Technically, this is a direct consequence of the high capacity of all deep learning models, which is in turn related to the high number of parameters that the network needs to learn, as opposed to more traditional (a.k.a. shallow) ML algorithms. There are ways to mitigate the problem of overfitting in deep learning models for computer vision: image augmentation, dropout layers, minibatch normalization, etc.. However these are primarily regularization techniques that do not address the issue of the high capacity of the model, and mainly control the available ranges for the values of the parameters in the model. The only real way to deal with high capacity CNN architectures is to use an extremely high number of independent images, which is not always possible in many fields of interest.

Transfer learning is a novel and extremely effective way of using high capacity models with small sets of images. The approach is specifically designed to take into account the particular way that CNN are constructed for solving image perception problems. As it has been mentioned in Section 1.4, CNN consist of different layers, each of them having a number of different possible sub-architectures. A common principle to all CNN models, however, is that the initial layers capture very high level geometric structures, and as we go deeper in the network the remaining layers capture more problem-specific elements. For instance, if we train a deep network with the goal of distinguishing among different car models, the first layers may recognize circles, corners, colors, etc... and the last few layers could identify more car oriented objects like brakes, wheels, etc... The key point here being that, following this reasoning, if a deep CNN with very high capacity is trained on a general purpose image set like ImageNet, the weights of the more external layers are believed to be problem independent, and therefore will do a good job in image recognition tasks for other image datasets different than the one on which they have been originally trained. Therefore, one could use one of the models that have been successfully trained on ImageNet and freeze all the parameters for the most external part of the network, therefore effectively reducing immensely the model's capacity, and therefore its natural tendency to overfit. The rest of the layers will be trained on the new dataset under study and in this way we will enjoy an architecture that has the predictive power of a high capacity model on high level visual features, and at the same time this new training (for the most specific-oriented deeper layers) hopefully will deliver a good predictive power on the particular features of the new dataset.

A technical and crucial point on adapting pretrained models for transfer learning purposes is the cardinality of the classification categories set. The subset of the ImageNet dataset used for the ILSVRC competition has about 1000 different categories, and therefore all the architectures pretrained on ImageNet have as the top layer a softmax classifier with this number of nodes. This will be of course incompatible with any other dataset that we would like to study with these models if the number of categories do not match. Therefore it is necessary to substitute this top classifier of the pretrained model with another one with as many nodes as labelled categories exist in the new dataset. In this project for instance, we will be working in a classification problem with an image set consisting of 73 classes. Therefore the 1000 node top classifier of any of the previously mentioned pretrained models would be substituted with a 73 node softmax classifier.

Strictly speaking, we should only refer to transfer learning when the whole pretrained network is frozen and has no free parameters, except for the top classifier that has replaced the original one. Therefore we are simply training a multiclass linear classifier composed with a nonlinear softmax vector-valued function. The logic behind this thinking is that all the inner layers of the CNN provide increasingly better representation of the data, in the sense that they unwrap they way the images show the discriminant elements of the different categories and present them in such a way that a simple softmax classifier can separate these categories, something that was not possible with the original representation of the data. In this way, we can regard the pretrained CNN minus the top classifier as an immutable pipeline performing feature extraction on our image set. Afterwards these features are passed to the top classifier that can then do a good job recognizing the different categories.

Fine tuning, however, is commonly referred to as a further step in this strategy. In this setting, instead of freezing all the pretrained CNN we could declare some of its last (deeper) layers trainable, therefore adding all their weights to those of the top classifier, and they would be jointly trained. In this sense with fine tuning we would be training our final classifier but also modifying, or tuning, those parameters of the most external layers which, as has been noted before, are those responsible for recognizing low level features of our particular image set.

Chapter 2

The Beauty Dataset

2.1 Nature of the Dataset

In this work we will be working with the Beauty dataset. This dataset consists of around 73000 jpg images of different body products for beauty, personal hygiene, etc... There are 73 different categories each of them with 1000 images except for some cases where there are a few less images. The total size is of about 15GB. The folder structure of the Beauty dataset is the following:

```
boxes_Body_AntiCelluliteCream_1186
boxes_Body_BodyExfoliantsScrub_1189
boxes_Body_BodyFirmings_1187
boxes_Body_Bodyemollients_1185
boxes_Body_Bodymoisturizers_1184
boxes_Body_Bodytreatment_1200
boxes_Body_Hairremoversandbodybleaches_1188
boxes_Body_Handstreatment_1192
boxes_Body_SupplementsBody_1193
boxes_Bodyhygiene_Bathcosmetics_1116
boxes_Bodyhygiene_Deodorantsandantiperspiration_1113
boxes_Bodyhygiene_Feethygieniccosmetics_1118
boxes_Bodyhygiene_Liquidsoaps_1115
boxes_Bodyhygiene_SoapsandSyndets_1114
boxes_Bodyhygiene_WipesWetNapkins_1119
boxes_CosmeticGiftwraps_Beautysalonsthermalspa_1202
boxes_CosmeticGiftwraps_Giftwrap_1198
boxes_Cosmeticaccessories_Candles_1164
boxes_Cosmeticaccessories_Combs_1145
boxes_Cosmeticaccessories_Hairbrushes_1147
boxes_Cosmeticaccessories_MakeupBagsKits_1154
boxes_Cosmeticaccessories_Makeupbrushes_1151
```

```
boxes_Cosmeticaccessories_Mirrors_1155
boxes_Cosmeticaccessories_Razor_1160
boxes_Cosmeticaccessories_Sponges_1150
boxes_Eyesmakeup_Eyelinerandeyepencils_1176
boxes_Eyesmakeup_Eyeshadows_1174
boxes_Eyesmakeup_Mascara_1175
boxes_Facemakeup_Allinonefacemakeup_1173
boxes_Facemakeup_Blush_1170
boxes_Facemakeup_Concealer_1201
boxes_Facemakeup_Facedecoration_1172
boxes_Facemakeup_Foundationcream_1167
boxes_Facemakeup_Foundationcreampowdercompact_1169
boxes_Facemakeup_Makeupfashionmentions_1208
boxes_Facemakeup_Powder_1168
boxes_Facemakeup_TintedMoisturizer_1171
boxes_Facetreatment_AcneOilySkinTreatment_1161
boxes_Facetreatment_Antiagecreamfirmingcream_1149
boxes_Facetreatment_Eyestreatment_1143
boxes_Facetreatment_FaceExfoliantsScrub_1159
boxes_Facetreatment_Facecleansersanmakeupremovers_1140
boxes_Facetreatment_Facemasks_1158
boxes_Facetreatment_Faceserum_1243
boxes_Facetreatment_Facetoner_1144
boxes_Facetreatment_Facetreatment_1146
boxes_Facetreatment_Lipstreatment_1163
boxes_Fragrances_Fragrances_1195
boxes_Fragrances_HouseFragrances_1356
boxes_HairScalp_Conditioner_1131
boxes_HairScalp_Hairdyes_1127
boxes_HairScalp_Hairfashionmentions_1207
boxes_HairScalp_Hairspray_1129
boxes_HairScalp_Hairtreatment_1121
boxes_HairScalp_Shampoo_1104
boxes_HairScalp_Stylingserumgelmousse_1105
boxes_HairScalp_SupplementsHair_1138
boxes_Handsmakeup_Nailsdecoration_1183
boxes_Handsmakeup_Nailspolish_1181
boxes_Lipsmakeup_Lipliners_1178
boxes_Lipsmakeup_Lipsticksandgloss_1177
boxes_Manline_Mancreamsandlotions_1132
boxes_Manline_PreandAfterShaveLotions_1130
boxes_Oralhygiene_ElectricToothbrush_1124
boxes_Oralhygiene_Toothbrushes_1123
boxes_Oralhygiene_Toothpaste_1120
```

```
boxes_Packagingmultiproduct_Multicosmeticspackaging_1179
boxes_Setline_Setline_1205
boxes_Suntanlotions_AfterSunLotion_1109
boxes_Suntanlotions_SelftanningLotion_1111
boxes_Suntanlotions_SunScreenBody_1106
boxes_Suntanlotions_SunScreenFace_1107
boxes_Suntanlotions_Suntanlotions_1110
```

We will start by looking at some of the images of the dataset in order to grasp an idea of their nature. The following code will create an array of images, choosing one for each category.

Input:

```
import os, shutil
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure, imshow, axis
from keras.preprocessing import image

size = 100

hSize = size
wSize = size
col = 7

subdirs = list()

for root, dirs, files in os.walk('/home/miguelyogur/datasets/Beauty'):
    subdirs.append(root)

subdirs = subdirs[1:]
images = []
for subdir in subdirs:
    name= os.listdir(subdir)[0]
    img_path =  os.path.join(subdir, name)
    images.append(image.load_img(img_path, target_size=(wSize, hSize)))

fig = figure( figsize=(wSize, hSize))
number_of_files = len(images)
row = number_of_files/col
if (number_of_files%col != 0):
    row += 1
for i in range(number_of_files):
    a=fig.add_subplot(row,col,i+1)
    image = images[i]
    imshow(image)
    axis('off')
```

The images produced by this code are shown in Figure 2.1.

2.2 Organization of the Dataset

With its current folder structure, we cannot yet work with this dataset. The main reason being that it is not feasible to encapsulate all these images in numpy arrays for training, validation and testing since we would run out of memory. For this reason, we will resort to keras generators that will read batches of images from disk. A previous housekeeping step in order to implement this procedure is to reorganize the images in train, validation and test folders, each of them having a subfolder for each category. We will reserve 80% of the images for training and 10% for validation and test, respectively. We will also clean the names of the folders, getting rid of the prefixes and suffixes, which are not informative. The following code will create a copy of the dataset called "dataset_beauty" having all these practical properties.

Input:

```
#define folder paths for train, validation, test
import os, shutil

#folders names
original_dataset_dir = '/home/miguelyogur/datasets/Beauty/'
base_dir = '/home/miguelyogur/pruebas/dataset_beauty'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

#create folders
os.mkdir(base_dir)
os.mkdir(train_dir)
os.mkdir(validation_dir)
os.mkdir(test_dir)

#get list containing all paths for the categories of the Beauty dataset
subdirs = list()

for root, dirs, files in os.walk('/home/miguelyogur/datasets/Beauty'):
    subdirs.append(root)
```

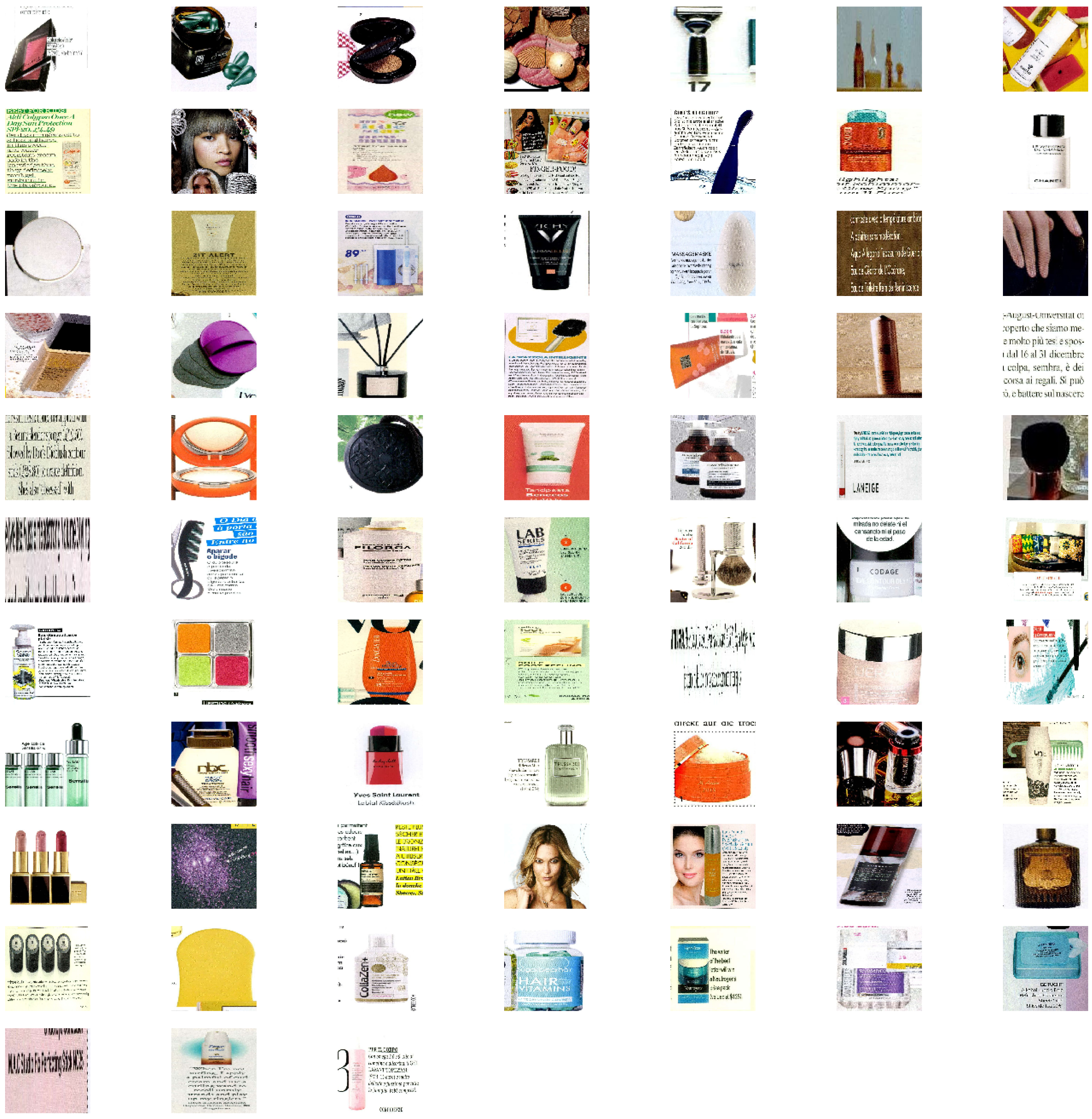


Figure 2.1: Sample images from the Beauty dataset.

```
subdirs = subdirs[1:]

#store categories' names in list
names = list({x.replace('/home/miguelyogur/datasets/Beauty/boxes_', '')[:-5]
        for x in subdirs})

subdirs.sort()
names.sort()

#create subfolders for each category inside the train, validation, test folders
folders_list = [train_dir, validation_dir, test_dir]

for folder in folders_list:
    for name in names:
        new_name = os.path.join(folder, name)
        os.mkdir(new_name)

#copy images from each category to the base directory:
#80% for train, 10% for validation and 10% for test
for i in range(len(names)):
    filenames = list()
    for root, dirs, files in os.walk(subdirs[i]):
        filenames.append(files)

    filenames = filenames[0]
    filenames.sort()

    n_test  = len(filenames)
    n_train = round(0.8*n_test)
    n_val   = round(0.9*n_test)

    for j in range(n_train):
        src = os.path.join(subdirs[i], filenames[j])
        dst = os.path.join(train_dir, names[i], filenames[j])
        shutil.copyfile(src, dst)

    for j in range(n_train,n_val):
        src = os.path.join(subdirs[i], filenames[j])
        dst = os.path.join(validation_dir, names[i], filenames[j])
        shutil.copyfile(src, dst)

    for j in range(n_val, n_test):
        src = os.path.join(subdirs[i], filenames[j])
        dst = os.path.join(test_dir, names[i], filenames[j])
        shutil.copyfile(src, dst)
```

We can check the number of images in each category for the original dataset.

Input:

```
#count images from each category in the original dataset
for root, dirs, files in os.walk(original_dataset_dir):
    print(root,len(os.listdir(root)))
```

Output:

```
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_Blush_1170 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_AntiCelluliteCream_1186 1000
/home/miguelyogur/datasets/Beauty/boxes_Suntanlotions_Suntanlotions_1110 1000
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_Powder_1168 1000
/home/miguelyogur/datasets/Beauty/boxes_Cosmeticaccessories_Razor_1160 1000
/home/miguelyogur/datasets/Beauty/boxes_Packagingmultiproduct_Multicosmeticspackaging_1179 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_Facetoner_1144 1000
/home/miguelyogur/datasets/Beauty/boxes_Suntanlotions_SunScreenFace_1107 1000
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_Makeupfashionmentions_1208 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_Bodyemollients_1185 1000
/home/miguelyogur/datasets/Beauty/boxes_Handsmakeup_Nailsdecoration_1183 1000
/home/miguelyogur/datasets/Beauty/boxes_Oralhygiene_Toothbrushes_1123 1000
/home/miguelyogur/datasets/Beauty/boxes_Suntanlotions_AfterSunLotion_1109 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_FaceExfoliantsScrub_1159 1000
/home/miguelyogur/datasets/Beauty/boxes_Cosmeticaccessories_Mirrors_1155 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_AcneOilySkinTreatment_1161 1000
/home/miguelyogur/datasets/Beauty/boxes_Oralhygiene_ElectricToothbrush_1124 880
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_TintedMoisturizer_1171 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_Facemasks_1158 1000
/home/miguelyogur/datasets/Beauty/boxes_Fragrances_Fragrances_1195 1000
/home/miguelyogur/datasets/Beauty/boxes_Handsmakeup_Nailspolish_1181 1000
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_Foundationcream_1167 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_Hairremoversandbodybleaches_1188 1000
/home/miguelyogur/datasets/Beauty/boxes_Cosmeticaccessories_Candles_1164 1000
/home/miguelyogur/datasets/Beauty/boxes_Cosmeticaccessories_Hairbrushes_1147 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_Handstreatment_1192 1000
/home/miguelyogur/datasets/Beauty/boxes_Lipsmakeup_Lipliners_1178 1000
/home/miguelyogur/datasets/Beauty/boxes_CosmeticGiftwraps_Beautysalonsthermalspa_1202 1000
/home/miguelyogur/datasets/Beauty/boxes_Cosmeticaccessories_Sponges_1150 1000
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_Foundationcreampowdercompact_1169 1000
/home/miguelyogur/datasets/Beauty/boxes_Bodyhygiene_SoapsandSyndets_1114 1000
/home/miguelyogur/datasets/Beauty/boxes_Oralhygiene_Toothpaste_1120 1000
/home/miguelyogur/datasets/Beauty/boxes_Bodyhygiene_Liquidsoaps_1115 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_Lipstreatment_1163 1000
/home/miguelyogur/datasets/Beauty/boxes_Cosmeticaccessories_Makeupbrushes_1151 1000
/home/miguelyogur/datasets/Beauty/boxes_HairScalp_Hairspray_1129 1000
/home/miguelyogur/datasets/Beauty/boxes_Cosmeticaccessories_Combs_1145 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_BodyFirmings_1187 1000
/home/miguelyogur/datasets/Beauty/boxes_Manline_Mancreamsandlotions_1132 1000
/home/miguelyogur/datasets/Beauty/boxes_Setline_Setline_1205 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_Eyestreatment_1143 1000
```

```
/home/miguelyogur/datasets/Beauty/boxes_Cosmeticaccessories_MakeupBagsKits_1154 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_Facecleansersanmakeupremovers_1140 1000
/home/miguelyogur/datasets/Beauty/boxes_Eyesmakeup_Eyeshadows_1174 1000
/home/miguelyogur/datasets/Beauty/boxes_Suntanlotions_SunScreenBody_1106 1000
/home/miguelyogur/datasets/Beauty/boxes_Bodyhygiene_Feethygieniccosmetics_1118 1000
/home/miguelyogur/datasets/Beauty/boxes_HairScalp_Hairdyes_1127 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_Antiagecreamfirmingcream_1149 1000
/home/miguelyogur/datasets/Beauty/boxes_Eyesmakeup_Mascara_1175 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_Facetreatment_1146 1000
/home/miguelyogur/datasets/Beauty/boxes_HairScalp_Hairtreatment_1121 1000
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_Allinonefacemakeup_1173 1000
/home/miguelyogur/datasets/Beauty/boxes_Manline_PreandAfterShaveLotions_1130 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_BodyExfoliantsScrub_1189 1000
/home/miguelyogur/datasets/Beauty/boxes_Eyesmakeup_Eyelinerandeyepencils_1176 1000
/home/miguelyogur/datasets/Beauty/boxes_HairScalp_Conditioner_1131 1000
/home/miguelyogur/datasets/Beauty/boxes_Lipsmakeup_Lipsticksandgloss_1177 1000
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_Facedecoration_1172 1000
/home/miguelyogur/datasets/Beauty/boxes_Bodyhygiene_Deodorantsandantiperspiration_1113 1000
/home/miguelyogur/datasets/Beauty/boxes_HairScalp_Hairfashionmentions_1207 1000
/home/miguelyogur/datasets/Beauty/boxes_Facetreatment_Faceserum_1243 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_Bodytreatment_1200 1000
/home/miguelyogur/datasets/Beauty/boxes_Fragrances_HouseFragrances_1356 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_Bodymoisturizers_1184 1000
/home/miguelyogur/datasets/Beauty/boxes_Suntanlotions_SelftanningLotion_1111 1000
/home/miguelyogur/datasets/Beauty/boxes_Body_SupplementsBody_1193 1000
/home/miguelyogur/datasets/Beauty/boxes_HairScalp_SupplementsHair_1138 1000
/home/miguelyogur/datasets/Beauty/boxes_CosmeticGiftwraps_Giftwrap_1198 1000
/home/miguelyogur/datasets/Beauty/boxes_HairScalp_Shampoo_1104 1000
/home/miguelyogur/datasets/Beauty/boxes_Bodyhygiene_WipesWetNapkins_1119 1000
/home/miguelyogur/datasets/Beauty/boxes_Facemakeup_Concealer_1201 1000
/home/miguelyogur/datasets/Beauty/boxes_HairScalp_Stylingserumgelmousse_1105 1000
/home/miguelyogur/datasets/Beauty/boxes_Bodyhygiene_Bathcosmetics_1116 1000
```

And we can also check the structure and number of images for each category in the new reorganized dataset.

Input:

```python
#count imgages from each category in the train, validation and test datasets
for folder in folders_list:
    filenames = list()
    print('------------')
    print(str(folder))
    print('------------')
    for root, dirs, files in os.walk(folder):
        print(root,len(os.listdir(root)))
```

Output:

```
------------
/home/miguelyogur/pruebas/dataset_beauty/train
------------
```

```
/home/miguelyogur/pruebas/dataset_beauty/train 73
/home/miguelyogur/pruebas/dataset_beauty/train/Bodyhygiene_WipesWetNapkins 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_Facetoner 800
/home/miguelyogur/pruebas/dataset_beauty/train/HairScalp_Stylingserumgelmousse 800
/home/miguelyogur/pruebas/dataset_beauty/train/HairScalp_Hairfashionmentions 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_Foundationcream 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_Faceserum 800
/home/miguelyogur/pruebas/dataset_beauty/train/Suntanlotions_AfterSunLotion 800
/home/miguelyogur/pruebas/dataset_beauty/train/Oralhygiene_Toothbrushes 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_Makeupfashionmentions 800
/home/miguelyogur/pruebas/dataset_beauty/train/Oralhygiene_ElectricToothbrush 704
/home/miguelyogur/pruebas/dataset_beauty/train/Handsmakeup_Nailspolish 800
/home/miguelyogur/pruebas/dataset_beauty/train/Cosmeticaccessories_Combs 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_Antiagecreamfirmingcream 800
/home/miguelyogur/pruebas/dataset_beauty/train/Eyesmakeup_Eyeshadows 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_BodyFirmings 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_Bodyemollients 800
/home/miguelyogur/pruebas/dataset_beauty/train/HairScalp_Conditioner 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_Facedecoration 800
/home/miguelyogur/pruebas/dataset_beauty/train/CosmeticGiftwraps_Beautysalonsthermalspa 800
/home/miguelyogur/pruebas/dataset_beauty/train/Fragrances_Fragrances 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_Foundationcreampowdercompact 800
/home/miguelyogur/pruebas/dataset_beauty/train/Cosmeticaccessories_Candles 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_Facecleansersanmakeupremovers 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_SupplementsBody 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_AcneOilySkinTreatment 800
/home/miguelyogur/pruebas/dataset_beauty/train/Setline_Setline 800
/home/miguelyogur/pruebas/dataset_beauty/train/Lipsmakeup_Lipsticksandgloss 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_Powder 800
/home/miguelyogur/pruebas/dataset_beauty/train/Lipsmakeup_Lipliners 800
/home/miguelyogur/pruebas/dataset_beauty/train/Manline_Mancreamsandlotions 800
/home/miguelyogur/pruebas/dataset_beauty/train/Cosmeticaccessories_Makeupbrushes 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_Concealer 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_Handstreatment 800
/home/miguelyogur/pruebas/dataset_beauty/train/Cosmeticaccessories_Sponges 800
/home/miguelyogur/pruebas/dataset_beauty/train/HairScalp_Shampoo 800
/home/miguelyogur/pruebas/dataset_beauty/train/HairScalp_Hairdyes 800
/home/miguelyogur/pruebas/dataset_beauty/train/HairScalp_Hairtreatment 800
/home/miguelyogur/pruebas/dataset_beauty/train/Suntanlotions_SunScreenBody 800
/home/miguelyogur/pruebas/dataset_beauty/train/Bodyhygiene_Feethygieniccosmetics 800
/home/miguelyogur/pruebas/dataset_beauty/train/Manline_PreandAfterShaveLotions 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_TintedMoisturizer 800
/home/miguelyogur/pruebas/dataset_beauty/train/Fragrances_HouseFragrances 800
/home/miguelyogur/pruebas/dataset_beauty/train/Suntanlotions_Suntanlotions 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_Lipstreatment 800
/home/miguelyogur/pruebas/dataset_beauty/train/Eyesmakeup_Eyelinerandeyepencils 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_Eyestreatment 800
/home/miguelyogur/pruebas/dataset_beauty/train/Bodyhygiene_Deodorantsandantiperspiration 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_Bodymoisturizers 800
/home/miguelyogur/pruebas/dataset_beauty/train/Cosmeticaccessories_MakeupBagsKits 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_FaceExfoliantsScrub 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_AntiCelluliteCream 800
/home/miguelyogur/pruebas/dataset_beauty/train/Suntanlotions_SelftanningLotion 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_Bodytreatment 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_Hairremoversandbodybleaches 800
```

```
/home/miguelyogur/pruebas/dataset_beauty/train/Handsmakeup_Nailsdecoration 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_Allinonefacemakeup 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_Facemasks 800
/home/miguelyogur/pruebas/dataset_beauty/train/Suntanlotions_SunScreenFace 800
/home/miguelyogur/pruebas/dataset_beauty/train/Body_BodyExfoliantsScrub 800
/home/miguelyogur/pruebas/dataset_beauty/train/Eyesmakeup_Mascara 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facemakeup_Blush 800
/home/miguelyogur/pruebas/dataset_beauty/train/Oralhygiene_Toothpaste 800
/home/miguelyogur/pruebas/dataset_beauty/train/Bodyhygiene_SoapsandSyndets 800
/home/miguelyogur/pruebas/dataset_beauty/train/Facetreatment_Facetreatment 800
/home/miguelyogur/pruebas/dataset_beauty/train/Cosmeticaccessories_Hairbrushes 800
/home/miguelyogur/pruebas/dataset_beauty/train/Bodyhygiene_Liquidsoaps 800
/home/miguelyogur/pruebas/dataset_beauty/train/HairScalp_SupplementsHair 800
/home/miguelyogur/pruebas/dataset_beauty/train/Cosmeticaccessories_Razor 800
/home/miguelyogur/pruebas/dataset_beauty/train/Packagingmultiproduct_Multicosmeticspackaging 800
/home/miguelyogur/pruebas/dataset_beauty/train/CosmeticGiftwraps_Giftwrap 800
/home/miguelyogur/pruebas/dataset_beauty/train/HairScalp_Hairspray 800
/home/miguelyogur/pruebas/dataset_beauty/train/Bodyhygiene_Bathcosmetics 800
/home/miguelyogur/pruebas/dataset_beauty/train/Cosmeticaccessories_Mirrors 800
------------
/home/miguelyogur/pruebas/dataset_beauty/validation
------------
/home/miguelyogur/pruebas/dataset_beauty/validation 73
/home/miguelyogur/pruebas/dataset_beauty/validation/Bodyhygiene_WipesWetNapkins 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_Facetoner 100
/home/miguelyogur/pruebas/dataset_beauty/validation/HairScalp_Stylingserumgelmousse 100
/home/miguelyogur/pruebas/dataset_beauty/validation/HairScalp_Hairfashionmentions 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_Foundationcream 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_Faceserum 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Suntanlotions_AfterSunLotion 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Oralhygiene_Toothbrushes 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_Makeupfashionmentions 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Oralhygiene_ElectricToothbrush 88
/home/miguelyogur/pruebas/dataset_beauty/validation/Handsmakeup_Nailspolish 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Cosmeticaccessories_Combs 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_Antiagecreamfirmingcream 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Eyesmakeup_Eyeshadows 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_BodyFirmings 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_Bodyemollients 100
/home/miguelyogur/pruebas/dataset_beauty/validation/HairScalp_Conditioner 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_Facedecoration 100
/home/miguelyogur/pruebas/dataset_beauty/validation/CosmeticGiftwraps_Beautysalonsthermalspa 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Fragrances_Fragrances 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_Foundationcreampowdercompact 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Cosmeticaccessories_Candles 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_Facecleansersanmakeupremovers 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_SupplementsBody 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_AcneOilySkinTreatment 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Setline_Setline 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Lipsmakeup_Lipsticksandgloss 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_Powder 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Lipsmakeup_Lipliners 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Manline_Mancreamsandlotions 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Cosmeticaccessories_Makeupbrushes 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_Concealer 100
```

```
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_Handstreatment 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Cosmeticaccessories_Sponges 100
/home/miguelyogur/pruebas/dataset_beauty/validation/HairScalp_Shampoo 100
/home/miguelyogur/pruebas/dataset_beauty/validation/HairScalp_Hairdyes 100
/home/miguelyogur/pruebas/dataset_beauty/validation/HairScalp_Hairtreatment 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Suntanlotions_SunScreenBody 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Bodyhygiene_Feethygieniccosmetics 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Manline_PreandAfterShaveLotions 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_TintedMoisturizer 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Fragrances_HouseFragrances 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Suntanlotions_Suntanlotions 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_Lipstreatment 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Eyesmakeup_Eyelinerandeyepencils 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_Eyestreatment 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Bodyhygiene_Deodorantsandantiperspiration 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_Bodymoisturizers 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Cosmeticaccessories_MakeupBagsKits 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_FaceExfoliantsScrub 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_AntiCelluliteCream 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Suntanlotions_SelftanningLotion 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_Bodytreatment 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_Hairremoversandbodybleaches 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Handsmakeup_Nailsdecoration 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_Allinonefacemakeup 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_Facemasks 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Suntanlotions_SunScreenFace 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Body_BodyExfoliantsScrub 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Eyesmakeup_Mascara 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facemakeup_Blush 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Oralhygiene_Toothpaste 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Bodyhygiene_SoapsandSyndets 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Facetreatment_Facetreatment 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Cosmeticaccessories_Hairbrushes 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Bodyhygiene_Liquidsoaps 100
/home/miguelyogur/pruebas/dataset_beauty/validation/HairScalp_SupplementsHair 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Cosmeticaccessories_Razor 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Packagingmultiproduct_Multicosmeticspackaging 100
/home/miguelyogur/pruebas/dataset_beauty/validation/CosmeticGiftwraps_Giftwrap 100
/home/miguelyogur/pruebas/dataset_beauty/validation/HairScalp_Hairspray 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Bodyhygiene_Bathcosmetics 100
/home/miguelyogur/pruebas/dataset_beauty/validation/Cosmeticaccessories_Mirrors 100
------------
/home/miguelyogur/pruebas/dataset_beauty/test
------------
/home/miguelyogur/pruebas/dataset_beauty/test 73
/home/miguelyogur/pruebas/dataset_beauty/test/Bodyhygiene_WipesWetNapkins 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_Facetoner 100
/home/miguelyogur/pruebas/dataset_beauty/test/HairScalp_Stylingserumgelmousse 100
/home/miguelyogur/pruebas/dataset_beauty/test/HairScalp_Hairfashionmentions 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_Foundationcream 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_Faceserum 100
/home/miguelyogur/pruebas/dataset_beauty/test/Suntanlotions_AfterSunLotion 100
/home/miguelyogur/pruebas/dataset_beauty/test/Oralhygiene_Toothbrushes 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_Makeupfashionmentions 100
/home/miguelyogur/pruebas/dataset_beauty/test/Oralhygiene_ElectricToothbrush 88
```

```
/home/miguelyogur/pruebas/dataset_beauty/test/Handsmakeup_Nailspolish 100
/home/miguelyogur/pruebas/dataset_beauty/test/Cosmeticaccessories_Combs 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_Antiagecreamfirmingcream 100
/home/miguelyogur/pruebas/dataset_beauty/test/Eyesmakeup_Eyeshadows 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_BodyFirmings 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_Bodyemollients 100
/home/miguelyogur/pruebas/dataset_beauty/test/HairScalp_Conditioner 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_Facedecoration 100
/home/miguelyogur/pruebas/dataset_beauty/test/CosmeticGiftwraps_Beautysalonsthermalspa 100
/home/miguelyogur/pruebas/dataset_beauty/test/Fragrances_Fragrances 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_Foundationcreampowdercompact 100
/home/miguelyogur/pruebas/dataset_beauty/test/Cosmeticaccessories_Candles 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_Facecleansersanmakeupremovers 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_SupplementsBody 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_AcneOilySkinTreatment 100
/home/miguelyogur/pruebas/dataset_beauty/test/Setline_Setline 100
/home/miguelyogur/pruebas/dataset_beauty/test/Lipsmakeup_Lipsticksandgloss 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_Powder 100
/home/miguelyogur/pruebas/dataset_beauty/test/Lipsmakeup_Lipliners 100
/home/miguelyogur/pruebas/dataset_beauty/test/Manline_Mancreamsandlotions 100
/home/miguelyogur/pruebas/dataset_beauty/test/Cosmeticaccessories_Makeupbrushes 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_Concealer 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_Handstreatment 100
/home/miguelyogur/pruebas/dataset_beauty/test/Cosmeticaccessories_Sponges 100
/home/miguelyogur/pruebas/dataset_beauty/test/HairScalp_Shampoo 100
/home/miguelyogur/pruebas/dataset_beauty/test/HairScalp_Hairdyes 100
/home/miguelyogur/pruebas/dataset_beauty/test/HairScalp_Hairtreatment 100
/home/miguelyogur/pruebas/dataset_beauty/test/Suntanlotions_SunScreenBody 100
/home/miguelyogur/pruebas/dataset_beauty/test/Bodyhygiene_Feethygieniccosmetics 100
/home/miguelyogur/pruebas/dataset_beauty/test/Manline_PreandAfterShaveLotions 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_TintedMoisturizer 100
/home/miguelyogur/pruebas/dataset_beauty/test/Fragrances_HouseFragrances 100
/home/miguelyogur/pruebas/dataset_beauty/test/Suntanlotions_Suntanlotions 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_Lipstreatment 100
/home/miguelyogur/pruebas/dataset_beauty/test/Eyesmakeup_Eyelinerandeyepencils 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_Eyestreatment 100
/home/miguelyogur/pruebas/dataset_beauty/test/Bodyhygiene_Deodorantsandantiperspiration 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_Bodymoisturizers 100
/home/miguelyogur/pruebas/dataset_beauty/test/Cosmeticaccessories_MakeupBagsKits 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_FaceExfoliantsScrub 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_AntiCelluliteCream 100
/home/miguelyogur/pruebas/dataset_beauty/test/Suntanlotions_SelftanningLotion 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_Bodytreatment 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_Hairremoversandbodybleaches 100
/home/miguelyogur/pruebas/dataset_beauty/test/Handsmakeup_Nailsdecoration 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_Allinonefacemakeup 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_Facemasks 100
/home/miguelyogur/pruebas/dataset_beauty/test/Suntanlotions_SunScreenFace 100
/home/miguelyogur/pruebas/dataset_beauty/test/Body_BodyExfoliantsScrub 100
/home/miguelyogur/pruebas/dataset_beauty/test/Eyesmakeup_Mascara 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facemakeup_Blush 100
/home/miguelyogur/pruebas/dataset_beauty/test/Oralhygiene_Toothpaste 100
/home/miguelyogur/pruebas/dataset_beauty/test/Bodyhygiene_SoapsandSyndets 100
/home/miguelyogur/pruebas/dataset_beauty/test/Facetreatment_Facetreatment 100
/home/miguelyogur/pruebas/dataset_beauty/test/Cosmeticaccessories_Hairbrushes 100
```

```
/home/miguelyogur/pruebas/dataset_beauty/test/Bodyhygiene_Liquidsoaps 100
/home/miguelyogur/pruebas/dataset_beauty/test/HairScalp_SupplementsHair 100
/home/miguelyogur/pruebas/dataset_beauty/test/Cosmeticaccessories_Razor 100
/home/miguelyogur/pruebas/dataset_beauty/test/Packagingmultiproduct_Multicosmeticspackaging 100
/home/miguelyogur/pruebas/dataset_beauty/test/CosmeticGiftwraps_Giftwrap 100
/home/miguelyogur/pruebas/dataset_beauty/test/HairScalp_Hairspray 100
/home/miguelyogur/pruebas/dataset_beauty/test/Bodyhygiene_Bathcosmetics 100
/home/miguelyogur/pruebas/dataset_beauty/test/Cosmeticaccessories_Mirrors 100
```

With this setup we are already in position of designing deep learning algorithms for the predictive classification of the dataset.

Chapter 3

The InceptionV3 Architecture

The inception architecture for deep convolutional neural networks was originally introduced in [14]. Since then, there have been several versions and improvements (see [9] and [10] for a survey). The main novelty of this family of architectures with respect to other CNN architectures at the time of the introduction of InceptionV1 is the use of the so called inception modules. In this work we have used the InceptionV3 version, introduced in [13]. In this chapter we will provide a very basic introduction to the main ideas behind the inception family of CNN and the specific block diagram structure of the InceptionV3 network, in order to prepare for its use in subsequent chapters.

3.1 Inception Modules

All the inception architectures are organized in blocks, the inception modules (to which we will refer indistinctly as blocks), with some common and very particular design concepts. The main idea behind these blocks is to use several convolution kernels in parallel.

Recall that the kernel in a convolutional layer is directly related to the size of the geometric objects that the network learns to recognize. A recurrent problem in image recognition is that the portion of the image occupied by the object to be recognized can vary a lot, therefore also varying its size. This can happen not only from one dataset to another, but more importantly, among different images in the same dataset. For this reason, in the inception blocks several kernels (of dim 1, 3 and 5) are used at the same level instead of stacked in successive layers in a linear fashion. The outputs of these convolutions are then concatenated and the results are passed to the following block. In this fashion, the network can adapt itself to different sizes of the relevant objects during the training process. A schematic representation of the most basic form of an inception block is shown in Figure 3.1.

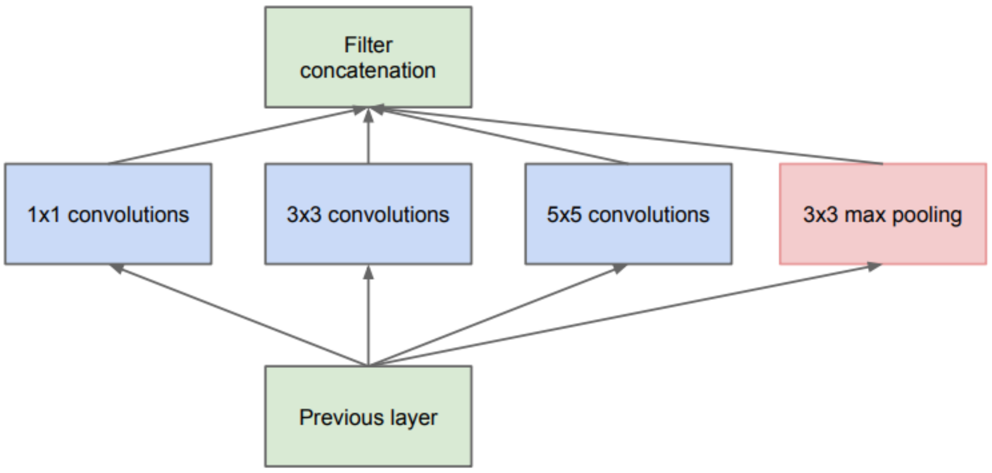


Figure 3.1: An inception block (source [14])

3.2 Block Diagram

The full InceptionV3 network block diagram is shown in Figure 3.2.

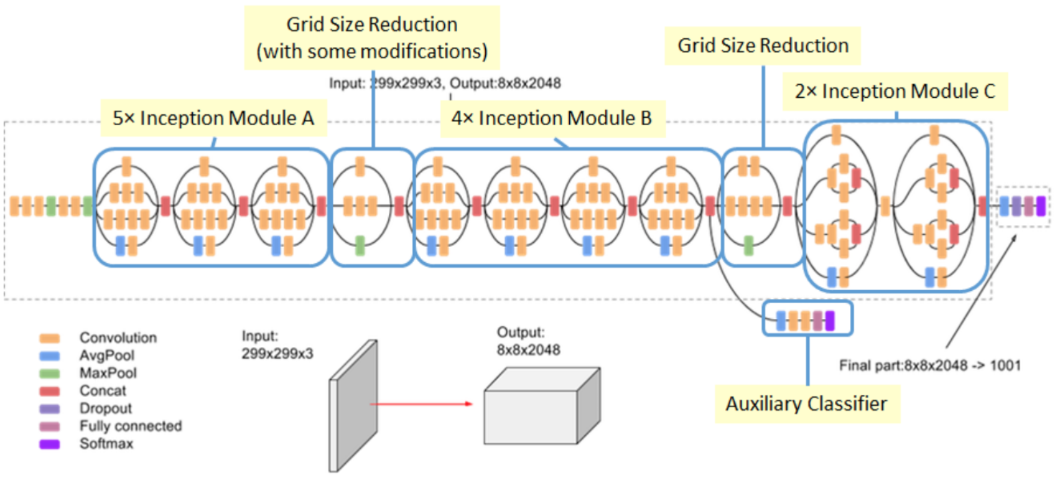


Figure 3.2: InceptionV3 block diagram (source [10])

It consists of 11 inception blocks, an input block and an output block (top classifier). It also contains an auxiliary classifier, with the purpose of helping with the backpropagation process and mitigate the problem of vanishing gradientes, ubiquitous in deep architectures. Notice that the inner structure of the inception blocks is not exactly the same as the one showed in 3.1. However, the basic designing principles are similar.

3.3 Top Classifier

As we can see from Figure 3.2 the top classifier actually consists of 4 layers: an average pooling layer, a dropout layer (for regularization purposes) and two dense layers, the final one being a softmax classifier. In the remaining part of this work we will replicate this structure, replacing the last softmax dense layer with a similar one having the necessary number of nodes for our datasets. This will be in fact the only modification done to the InceptionV3 architecture in Chapters 4 and 5.

Chapter 4

Transfer Learning on the Full Dataset

In this chapter we will use the InceptionV3 network previously trained on the ImageNet dataset. Keras includes this architecture already programmed, including the weights of the different nodes trained on the ImageNet dataset. We will develop a program that will download this network, replace the top classifier with one adapted to the Beauty dataset, initialize the model freezing the blocks that we wish to declare non-trainable, train the model and finally compute some quality scores like validation and loss curves and confusion matrices.

4.1 Building the Model

We will start by loading the necessary packages for the full implementation of the training stage of the project, and also define a function that will build and initialize the model.

Input:

```
import os, shutil
from keras import models
from keras import layers
from keras import optimizers
from keras.applications import InceptionV3
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.models import load_model
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
import itertools
```

```
def init_model():
    conv_base = InceptionV3(weights='imagenet',
                    include_top=False,
                    input_shape=(input_dim, input_dim, 3)
                   )

    print(conv_base.summary())

    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(class_dim, activation = 'relu'))
    model.add(layers.Dense(n_classes, activation='softmax'))
    print(model.summary())

    for layer in model.layers[0].layers:
        layer.trainable = False

    return model
```

The `init_function` function first downloads the inceptionV3 network with the weights pretrained in ImageNet. We have included an option to exclude the top classifier and also the dimensions of the image are passed as a variable that will be initialized later to 299, since the original InceptionV3 architecture expects to be fed $299 \times 299 \times 3$ sized images. This will be the main component of our model, stored under the name `conv_base`. If we examine the summary of this component so far we obtain:

Output:

```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 299, 299, 3)  0
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 149, 149, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 149, 149, 32) 96          conv2d_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 149, 149, 32) 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 147, 147, 32) 9216        activation_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 147, 147, 32) 96          conv2d_2[0][0]
__________________________________________________________________________________________________
```

```
activation_2 (Activation)       (None, 147, 147, 32) 0          batch_normalization_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 147, 147, 64) 18432      activation_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 147, 147, 64) 192        conv2d_3[0][0]
__________________________________________________________________________________________________
activation_3 (Activation)       (None, 147, 147, 64) 0          batch_normalization_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 73, 73, 64)   0          activation_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 73, 73, 80)   5120       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 73, 73, 80)   240        conv2d_4[0][0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 73, 73, 80)   0          batch_normalization_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 71, 71, 192)  138240     activation_4[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 71, 71, 192)  576        conv2d_5[0][0]
__________________________________________________________________________________________________
activation_5 (Activation)       (None, 71, 71, 192)  0          batch_normalization_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 35, 35, 192)  0          activation_5[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 35, 35, 64)   12288      max_pooling2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 35, 35, 64)   192        conv2d_9[0][0]
__________________________________________________________________________________________________
activation_9 (Activation)       (None, 35, 35, 64)   0          batch_normalization_9[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 35, 35, 48)   9216       max_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 35, 35, 96)   55296      activation_9[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 35, 35, 48)   144        conv2d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 35, 35, 96)   288        conv2d_10[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, 35, 35, 48)   0          batch_normalization_7[0][0]
__________________________________________________________________________________________________
activation_10 (Activation)      (None, 35, 35, 96)   0          batch_normalization_10[0][0]
__________________________________________________________________________________________________
average_pooling2d_1 (AveragePoo (None, 35, 35, 192)  0          max_pooling2d_2[0][0]
__________________________________________________________________________________________________
```

```
conv2d_6 (Conv2D)               (None, 35, 35, 64)   12288       max_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 35, 35, 64)   76800       activation_7[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 35, 35, 96)   82944       activation_10[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 35, 35, 32)   6144        average_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 35, 35, 64)   192         conv2d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 35, 35, 64)   192         conv2d_8[0][0]
__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 35, 35, 96)   288         conv2d_11[0][0]
__________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 35, 35, 32)   96          conv2d_12[0][0]
__________________________________________________________________________________________________
activation_6 (Activation)       (None, 35, 35, 64)   0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
activation_8 (Activation)       (None, 35, 35, 64)   0           batch_normalization_8[0][0]
__________________________________________________________________________________________________
activation_11 (Activation)      (None, 35, 35, 96)   0           batch_normalization_11[0][0]
__________________________________________________________________________________________________
activation_12 (Activation)      (None, 35, 35, 32)   0           batch_normalization_12[0][0]
__________________________________________________________________________________________________
mixed0 (Concatenate)            (None, 35, 35, 256)  0           activation_6[0][0]
                                                                 activation_8[0][0]
                                                                 activation_11[0][0]
                                                                 activation_12[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 35, 35, 64)   16384       mixed0[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 35, 35, 64)   192         conv2d_16[0][0]
__________________________________________________________________________________________________
activation_16 (Activation)      (None, 35, 35, 64)   0           batch_normalization_16[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 35, 35, 48)   12288       mixed0[0][0]
__________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 35, 35, 96)   55296       activation_16[0][0]
__________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 35, 35, 48)   144         conv2d_14[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 35, 35, 96)   288         conv2d_17[0][0]
__________________________________________________________________________________________________
activation_14 (Activation)      (None, 35, 35, 48)   0           batch_normalization_14[0][0]
```

```
__________________________________________________________________________________________________
activation_17 (Activation)      (None, 35, 35, 96)   0           batch_normalization_17[0][0]
__________________________________________________________________________________________________
average_pooling2d_2 (AveragePoo (None, 35, 35, 256)  0           mixed0[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 35, 35, 64)   16384       mixed0[0][0]
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 35, 35, 64)   76800       activation_14[0][0]
__________________________________________________________________________________________________
conv2d_18 (Conv2D)              (None, 35, 35, 96)   82944       activation_17[0][0]
__________________________________________________________________________________________________
conv2d_19 (Conv2D)              (None, 35, 35, 64)   16384       average_pooling2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 35, 35, 64)   192         conv2d_13[0][0]
__________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 35, 35, 64)   192         conv2d_15[0][0]
__________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, 35, 35, 96)   288         conv2d_18[0][0]
__________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 35, 35, 64)   192         conv2d_19[0][0]
__________________________________________________________________________________________________
activation_13 (Activation)      (None, 35, 35, 64)   0           batch_normalization_13[0][0]
__________________________________________________________________________________________________
activation_15 (Activation)      (None, 35, 35, 64)   0           batch_normalization_15[0][0]
__________________________________________________________________________________________________
activation_18 (Activation)      (None, 35, 35, 96)   0           batch_normalization_18[0][0]
__________________________________________________________________________________________________
activation_19 (Activation)      (None, 35, 35, 64)   0           batch_normalization_19[0][0]
__________________________________________________________________________________________________
mixed1 (Concatenate)            (None, 35, 35, 288)  0           activation_13[0][0]
                                                                 activation_15[0][0]
                                                                 activation_18[0][0]
                                                                 activation_19[0][0]
__________________________________________________________________________________________________
conv2d_23 (Conv2D)              (None, 35, 35, 64)   18432       mixed1[0][0]
__________________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, 35, 35, 64)   192         conv2d_23[0][0]
__________________________________________________________________________________________________
activation_23 (Activation)      (None, 35, 35, 64)   0           batch_normalization_23[0][0]
__________________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, 35, 35, 48)   13824       mixed1[0][0]
__________________________________________________________________________________________________
conv2d_24 (Conv2D)              (None, 35, 35, 96)   55296       activation_23[0][0]
__________________________________________________________________________________________________
```

```
batch_normalization_21 (BatchNo (None, 35, 35, 48)   144         conv2d_21[0][0]
__________________________________________________________________________________________________
batch_normalization_24 (BatchNo (None, 35, 35, 96)   288         conv2d_24[0][0]
__________________________________________________________________________________________________
activation_21 (Activation)      (None, 35, 35, 48)   0           batch_normalization_21[0][0]
__________________________________________________________________________________________________
activation_24 (Activation)      (None, 35, 35, 96)   0           batch_normalization_24[0][0]
__________________________________________________________________________________________________
average_pooling2d_3 (AveragePoo (None, 35, 35, 288)  0           mixed1[0][0]
__________________________________________________________________________________________________
conv2d_20 (Conv2D)              (None, 35, 35, 64)   18432       mixed1[0][0]
__________________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 35, 35, 64)   76800       activation_21[0][0]
__________________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, 35, 35, 96)   82944       activation_24[0][0]
__________________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, 35, 35, 64)   18432       average_pooling2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 35, 35, 64)   192         conv2d_20[0][0]
__________________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 35, 35, 64)   192         conv2d_22[0][0]
__________________________________________________________________________________________________
batch_normalization_25 (BatchNo (None, 35, 35, 96)   288         conv2d_25[0][0]
__________________________________________________________________________________________________
batch_normalization_26 (BatchNo (None, 35, 35, 64)   192         conv2d_26[0][0]
__________________________________________________________________________________________________
activation_20 (Activation)      (None, 35, 35, 64)   0           batch_normalization_20[0][0]
__________________________________________________________________________________________________
activation_22 (Activation)      (None, 35, 35, 64)   0           batch_normalization_22[0][0]
__________________________________________________________________________________________________
activation_25 (Activation)      (None, 35, 35, 96)   0           batch_normalization_25[0][0]
__________________________________________________________________________________________________
activation_26 (Activation)      (None, 35, 35, 64)   0           batch_normalization_26[0][0]
__________________________________________________________________________________________________
mixed2 (Concatenate)            (None, 35, 35, 288)  0           activation_20[0][0]
                                                                 activation_22[0][0]
                                                                 activation_25[0][0]
                                                                 activation_26[0][0]
__________________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 35, 35, 64)   18432       mixed2[0][0]
__________________________________________________________________________________________________
batch_normalization_28 (BatchNo (None, 35, 35, 64)   192         conv2d_28[0][0]
__________________________________________________________________________________________________
activation_28 (Activation)      (None, 35, 35, 64)   0           batch_normalization_28[0][0]
```

```
__________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 35, 35, 96)   55296       activation_28[0][0]
__________________________________________________________________________________________________
batch_normalization_29 (BatchNo (None, 35, 35, 96)   288         conv2d_29[0][0]
__________________________________________________________________________________________________
activation_29 (Activation)      (None, 35, 35, 96)   0           batch_normalization_29[0][0]
__________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 17, 17, 384)  995328      mixed2[0][0]
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 17, 17, 96)   82944       activation_29[0][0]
__________________________________________________________________________________________________
batch_normalization_27 (BatchNo (None, 17, 17, 384)  1152        conv2d_27[0][0]
__________________________________________________________________________________________________
batch_normalization_30 (BatchNo (None, 17, 17, 96)   288         conv2d_30[0][0]
__________________________________________________________________________________________________
activation_27 (Activation)      (None, 17, 17, 384)  0           batch_normalization_27[0][0]
__________________________________________________________________________________________________
activation_30 (Activation)      (None, 17, 17, 96)   0           batch_normalization_30[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 17, 17, 288)  0           mixed2[0][0]
__________________________________________________________________________________________________
mixed3 (Concatenate)            (None, 17, 17, 768)  0           activation_27[0][0]
                                                                 activation_30[0][0]
                                                                 max_pooling2d_3[0][0]
__________________________________________________________________________________________________
conv2d_35 (Conv2D)              (None, 17, 17, 128)  98304       mixed3[0][0]
__________________________________________________________________________________________________
batch_normalization_35 (BatchNo (None, 17, 17, 128)  384         conv2d_35[0][0]
__________________________________________________________________________________________________
activation_35 (Activation)      (None, 17, 17, 128)  0           batch_normalization_35[0][0]
__________________________________________________________________________________________________
conv2d_36 (Conv2D)              (None, 17, 17, 128)  114688      activation_35[0][0]
__________________________________________________________________________________________________
batch_normalization_36 (BatchNo (None, 17, 17, 128)  384         conv2d_36[0][0]
__________________________________________________________________________________________________
activation_36 (Activation)      (None, 17, 17, 128)  0           batch_normalization_36[0][0]
__________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 17, 17, 128)  98304       mixed3[0][0]
__________________________________________________________________________________________________
conv2d_37 (Conv2D)              (None, 17, 17, 128)  114688      activation_36[0][0]
__________________________________________________________________________________________________
batch_normalization_32 (BatchNo (None, 17, 17, 128)  384         conv2d_32[0][0]
__________________________________________________________________________________________________
batch_normalization_37 (BatchNo (None, 17, 17, 128)  384         conv2d_37[0][0]
__________________________________________________________________________________________________
activation_32 (Activation)      (None, 17, 17, 128)  0           batch_normalization_32[0][0]
__________________________________________________________________________________________________
activation_37 (Activation)      (None, 17, 17, 128)  0           batch_normalization_37[0][0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 17, 17, 128)  114688      activation_32[0][0]
__________________________________________________________________________________________________
conv2d_38 (Conv2D)              (None, 17, 17, 128)  114688      activation_37[0][0]
__________________________________________________________________________________________________
batch_normalization_33 (BatchNo (None, 17, 17, 128)  384         conv2d_33[0][0]
__________________________________________________________________________________________________
batch_normalization_38 (BatchNo (None, 17, 17, 128)  384         conv2d_38[0][0]
__________________________________________________________________________________________________
activation_33 (Activation)      (None, 17, 17, 128)  0           batch_normalization_33[0][0]
__________________________________________________________________________________________________
activation_38 (Activation)      (None, 17, 17, 128)  0           batch_normalization_38[0][0]
__________________________________________________________________________________________________
average_pooling2d_4 (AveragePoo (None, 17, 17, 768)  0           mixed3[0][0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 17, 17, 192)  147456      mixed3[0][0]
__________________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 17, 17, 192)  172032      activation_33[0][0]
__________________________________________________________________________________________________
conv2d_39 (Conv2D)              (None, 17, 17, 192)  172032      activation_38[0][0]
__________________________________________________________________________________________________
conv2d_40 (Conv2D)              (None, 17, 17, 192)  147456      average_pooling2d_4[0][0]
__________________________________________________________________________________________________
batch_normalization_31 (BatchNo (None, 17, 17, 192)  576         conv2d_31[0][0]
__________________________________________________________________________________________________
batch_normalization_34 (BatchNo (None, 17, 17, 192)  576         conv2d_34[0][0]
__________________________________________________________________________________________________
batch_normalization_39 (BatchNo (None, 17, 17, 192)  576         conv2d_39[0][0]
__________________________________________________________________________________________________
batch_normalization_40 (BatchNo (None, 17, 17, 192)  576         conv2d_40[0][0]
__________________________________________________________________________________________________
activation_31 (Activation)      (None, 17, 17, 192)  0           batch_normalization_31[0][0]
__________________________________________________________________________________________________
activation_34 (Activation)      (None, 17, 17, 192)  0           batch_normalization_34[0][0]
__________________________________________________________________________________________________
activation_39 (Activation)      (None, 17, 17, 192)  0           batch_normalization_39[0][0]
__________________________________________________________________________________________________
activation_40 (Activation)      (None, 17, 17, 192)  0           batch_normalization_40[0][0]
__________________________________________________________________________________________________
mixed4 (Concatenate)            (None, 17, 17, 768)  0           activation_31[0][0]
```

```
                                                                activation_34[0][0]
                                                                activation_39[0][0]
                                                                activation_40[0][0]
__________________________________________________________________________________________________
conv2d_45 (Conv2D)              (None, 17, 17, 160)  122880      mixed4[0][0]
__________________________________________________________________________________________________
batch_normalization_45 (BatchNo (None, 17, 17, 160)  480         conv2d_45[0][0]
__________________________________________________________________________________________________
activation_45 (Activation)      (None, 17, 17, 160)  0           batch_normalization_45[0][0]
__________________________________________________________________________________________________
conv2d_46 (Conv2D)              (None, 17, 17, 160)  179200      activation_45[0][0]
__________________________________________________________________________________________________
batch_normalization_46 (BatchNo (None, 17, 17, 160)  480         conv2d_46[0][0]
__________________________________________________________________________________________________
activation_46 (Activation)      (None, 17, 17, 160)  0           batch_normalization_46[0][0]
__________________________________________________________________________________________________
conv2d_42 (Conv2D)              (None, 17, 17, 160)  122880      mixed4[0][0]
__________________________________________________________________________________________________
conv2d_47 (Conv2D)              (None, 17, 17, 160)  179200      activation_46[0][0]
__________________________________________________________________________________________________
batch_normalization_42 (BatchNo (None, 17, 17, 160)  480         conv2d_42[0][0]
__________________________________________________________________________________________________
batch_normalization_47 (BatchNo (None, 17, 17, 160)  480         conv2d_47[0][0]
__________________________________________________________________________________________________
activation_42 (Activation)      (None, 17, 17, 160)  0           batch_normalization_42[0][0]
__________________________________________________________________________________________________
activation_47 (Activation)      (None, 17, 17, 160)  0           batch_normalization_47[0][0]
__________________________________________________________________________________________________
conv2d_43 (Conv2D)              (None, 17, 17, 160)  179200      activation_42[0][0]
__________________________________________________________________________________________________
conv2d_48 (Conv2D)              (None, 17, 17, 160)  179200      activation_47[0][0]
__________________________________________________________________________________________________
batch_normalization_43 (BatchNo (None, 17, 17, 160)  480         conv2d_43[0][0]
__________________________________________________________________________________________________
batch_normalization_48 (BatchNo (None, 17, 17, 160)  480         conv2d_48[0][0]
__________________________________________________________________________________________________
activation_43 (Activation)      (None, 17, 17, 160)  0           batch_normalization_43[0][0]
__________________________________________________________________________________________________
activation_48 (Activation)      (None, 17, 17, 160)  0           batch_normalization_48[0][0]
__________________________________________________________________________________________________
average_pooling2d_5 (AveragePoo (None, 17, 17, 768)  0           mixed4[0][0]
__________________________________________________________________________________________________
conv2d_41 (Conv2D)              (None, 17, 17, 192)  147456      mixed4[0][0]
__________________________________________________________________________________________________
```

```
conv2d_44 (Conv2D)              (None, 17, 17, 192)  215040      activation_43[0][0]
__________________________________________________________________________________________________
conv2d_49 (Conv2D)              (None, 17, 17, 192)  215040      activation_48[0][0]
__________________________________________________________________________________________________
conv2d_50 (Conv2D)              (None, 17, 17, 192)  147456      average_pooling2d_5[0][0]
__________________________________________________________________________________________________
batch_normalization_41 (BatchNo (None, 17, 17, 192)  576         conv2d_41[0][0]
__________________________________________________________________________________________________
batch_normalization_44 (BatchNo (None, 17, 17, 192)  576         conv2d_44[0][0]
__________________________________________________________________________________________________
batch_normalization_49 (BatchNo (None, 17, 17, 192)  576         conv2d_49[0][0]
__________________________________________________________________________________________________
batch_normalization_50 (BatchNo (None, 17, 17, 192)  576         conv2d_50[0][0]
__________________________________________________________________________________________________
activation_41 (Activation)      (None, 17, 17, 192)  0           batch_normalization_41[0][0]
__________________________________________________________________________________________________
activation_44 (Activation)      (None, 17, 17, 192)  0           batch_normalization_44[0][0]
__________________________________________________________________________________________________
activation_49 (Activation)      (None, 17, 17, 192)  0           batch_normalization_49[0][0]
__________________________________________________________________________________________________
activation_50 (Activation)      (None, 17, 17, 192)  0           batch_normalization_50[0][0]
__________________________________________________________________________________________________
mixed5 (Concatenate)            (None, 17, 17, 768)  0           activation_41[0][0]
                                                                 activation_44[0][0]
                                                                 activation_49[0][0]
                                                                 activation_50[0][0]
__________________________________________________________________________________________________
conv2d_55 (Conv2D)              (None, 17, 17, 160)  122880      mixed5[0][0]
__________________________________________________________________________________________________
batch_normalization_55 (BatchNo (None, 17, 17, 160)  480         conv2d_55[0][0]
__________________________________________________________________________________________________
activation_55 (Activation)      (None, 17, 17, 160)  0           batch_normalization_55[0][0]
__________________________________________________________________________________________________
conv2d_56 (Conv2D)              (None, 17, 17, 160)  179200      activation_55[0][0]
__________________________________________________________________________________________________
batch_normalization_56 (BatchNo (None, 17, 17, 160)  480         conv2d_56[0][0]
__________________________________________________________________________________________________
activation_56 (Activation)      (None, 17, 17, 160)  0           batch_normalization_56[0][0]
__________________________________________________________________________________________________
conv2d_52 (Conv2D)              (None, 17, 17, 160)  122880      mixed5[0][0]
__________________________________________________________________________________________________
conv2d_57 (Conv2D)              (None, 17, 17, 160)  179200      activation_56[0][0]
__________________________________________________________________________________________________
batch_normalization_52 (BatchNo (None, 17, 17, 160)  480         conv2d_52[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_57 (BatchNo (None, 17, 17, 160)  480         conv2d_57[0][0]
--------------------------------------------------------------------------------------------------
activation_52 (Activation)      (None, 17, 17, 160)  0           batch_normalization_52[0][0]
--------------------------------------------------------------------------------------------------
activation_57 (Activation)      (None, 17, 17, 160)  0           batch_normalization_57[0][0]
--------------------------------------------------------------------------------------------------
conv2d_53 (Conv2D)              (None, 17, 17, 160)  179200      activation_52[0][0]
--------------------------------------------------------------------------------------------------
conv2d_58 (Conv2D)              (None, 17, 17, 160)  179200      activation_57[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_53 (BatchNo (None, 17, 17, 160)  480         conv2d_53[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_58 (BatchNo (None, 17, 17, 160)  480         conv2d_58[0][0]
--------------------------------------------------------------------------------------------------
activation_53 (Activation)      (None, 17, 17, 160)  0           batch_normalization_53[0][0]
--------------------------------------------------------------------------------------------------
activation_58 (Activation)      (None, 17, 17, 160)  0           batch_normalization_58[0][0]
--------------------------------------------------------------------------------------------------
average_pooling2d_6 (AveragePoo (None, 17, 17, 768)  0           mixed5[0][0]
--------------------------------------------------------------------------------------------------
conv2d_51 (Conv2D)              (None, 17, 17, 192)  147456      mixed5[0][0]
--------------------------------------------------------------------------------------------------
conv2d_54 (Conv2D)              (None, 17, 17, 192)  215040      activation_53[0][0]
--------------------------------------------------------------------------------------------------
conv2d_59 (Conv2D)              (None, 17, 17, 192)  215040      activation_58[0][0]
--------------------------------------------------------------------------------------------------
conv2d_60 (Conv2D)              (None, 17, 17, 192)  147456      average_pooling2d_6[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_51 (BatchNo (None, 17, 17, 192)  576         conv2d_51[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_54 (BatchNo (None, 17, 17, 192)  576         conv2d_54[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_59 (BatchNo (None, 17, 17, 192)  576         conv2d_59[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_60 (BatchNo (None, 17, 17, 192)  576         conv2d_60[0][0]
--------------------------------------------------------------------------------------------------
activation_51 (Activation)      (None, 17, 17, 192)  0           batch_normalization_51[0][0]
--------------------------------------------------------------------------------------------------
activation_54 (Activation)      (None, 17, 17, 192)  0           batch_normalization_54[0][0]
--------------------------------------------------------------------------------------------------
activation_59 (Activation)      (None, 17, 17, 192)  0           batch_normalization_59[0][0]
--------------------------------------------------------------------------------------------------
activation_60 (Activation)      (None, 17, 17, 192)  0           batch_normalization_60[0][0]
```

```
__________________________________________________________________________________________________
mixed6 (Concatenate)            (None, 17, 17, 768)  0           activation_51[0][0]
                                                                 activation_54[0][0]
                                                                 activation_59[0][0]
                                                                 activation_60[0][0]
__________________________________________________________________________________________________
conv2d_65 (Conv2D)              (None, 17, 17, 192)  147456      mixed6[0][0]
__________________________________________________________________________________________________
batch_normalization_65 (BatchNo (None, 17, 17, 192)  576         conv2d_65[0][0]
__________________________________________________________________________________________________
activation_65 (Activation)      (None, 17, 17, 192)  0           batch_normalization_65[0][0]
__________________________________________________________________________________________________
conv2d_66 (Conv2D)              (None, 17, 17, 192)  258048      activation_65[0][0]
__________________________________________________________________________________________________
batch_normalization_66 (BatchNo (None, 17, 17, 192)  576         conv2d_66[0][0]
__________________________________________________________________________________________________
activation_66 (Activation)      (None, 17, 17, 192)  0           batch_normalization_66[0][0]
__________________________________________________________________________________________________
conv2d_62 (Conv2D)              (None, 17, 17, 192)  147456      mixed6[0][0]
__________________________________________________________________________________________________
conv2d_67 (Conv2D)              (None, 17, 17, 192)  258048      activation_66[0][0]
__________________________________________________________________________________________________
batch_normalization_62 (BatchNo (None, 17, 17, 192)  576         conv2d_62[0][0]
__________________________________________________________________________________________________
batch_normalization_67 (BatchNo (None, 17, 17, 192)  576         conv2d_67[0][0]
__________________________________________________________________________________________________
activation_62 (Activation)      (None, 17, 17, 192)  0           batch_normalization_62[0][0]
__________________________________________________________________________________________________
activation_67 (Activation)      (None, 17, 17, 192)  0           batch_normalization_67[0][0]
__________________________________________________________________________________________________
conv2d_63 (Conv2D)              (None, 17, 17, 192)  258048      activation_62[0][0]
__________________________________________________________________________________________________
conv2d_68 (Conv2D)              (None, 17, 17, 192)  258048      activation_67[0][0]
__________________________________________________________________________________________________
batch_normalization_63 (BatchNo (None, 17, 17, 192)  576         conv2d_63[0][0]
__________________________________________________________________________________________________
batch_normalization_68 (BatchNo (None, 17, 17, 192)  576         conv2d_68[0][0]
__________________________________________________________________________________________________
activation_63 (Activation)      (None, 17, 17, 192)  0           batch_normalization_63[0][0]
__________________________________________________________________________________________________
activation_68 (Activation)      (None, 17, 17, 192)  0           batch_normalization_68[0][0]
__________________________________________________________________________________________________
average_pooling2d_7 (AveragePoo (None, 17, 17, 768)  0           mixed6[0][0]
__________________________________________________________________________________________________
```

```
conv2d_61 (Conv2D)              (None, 17, 17, 192)  147456      mixed6[0][0]
__________________________________________________________________________________________________
conv2d_64 (Conv2D)              (None, 17, 17, 192)  258048      activation_63[0][0]
__________________________________________________________________________________________________
conv2d_69 (Conv2D)              (None, 17, 17, 192)  258048      activation_68[0][0]
__________________________________________________________________________________________________
conv2d_70 (Conv2D)              (None, 17, 17, 192)  147456      average_pooling2d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_61 (BatchNo (None, 17, 17, 192)  576         conv2d_61[0][0]
__________________________________________________________________________________________________
batch_normalization_64 (BatchNo (None, 17, 17, 192)  576         conv2d_64[0][0]
__________________________________________________________________________________________________
batch_normalization_69 (BatchNo (None, 17, 17, 192)  576         conv2d_69[0][0]
__________________________________________________________________________________________________
batch_normalization_70 (BatchNo (None, 17, 17, 192)  576         conv2d_70[0][0]
__________________________________________________________________________________________________
activation_61 (Activation)      (None, 17, 17, 192)  0           batch_normalization_61[0][0]
__________________________________________________________________________________________________
activation_64 (Activation)      (None, 17, 17, 192)  0           batch_normalization_64[0][0]
__________________________________________________________________________________________________
activation_69 (Activation)      (None, 17, 17, 192)  0           batch_normalization_69[0][0]
__________________________________________________________________________________________________
activation_70 (Activation)      (None, 17, 17, 192)  0           batch_normalization_70[0][0]
__________________________________________________________________________________________________
mixed7 (Concatenate)            (None, 17, 17, 768)  0           activation_61[0][0]
                                                                 activation_64[0][0]
                                                                 activation_69[0][0]
                                                                 activation_70[0][0]
__________________________________________________________________________________________________
conv2d_73 (Conv2D)              (None, 17, 17, 192)  147456      mixed7[0][0]
__________________________________________________________________________________________________
batch_normalization_73 (BatchNo (None, 17, 17, 192)  576         conv2d_73[0][0]
__________________________________________________________________________________________________
activation_73 (Activation)      (None, 17, 17, 192)  0           batch_normalization_73[0][0]
__________________________________________________________________________________________________
conv2d_74 (Conv2D)              (None, 17, 17, 192)  258048      activation_73[0][0]
__________________________________________________________________________________________________
batch_normalization_74 (BatchNo (None, 17, 17, 192)  576         conv2d_74[0][0]
__________________________________________________________________________________________________
activation_74 (Activation)      (None, 17, 17, 192)  0           batch_normalization_74[0][0]
__________________________________________________________________________________________________
conv2d_71 (Conv2D)              (None, 17, 17, 192)  147456      mixed7[0][0]
__________________________________________________________________________________________________
conv2d_75 (Conv2D)              (None, 17, 17, 192)  258048      activation_74[0][0]
```

```
____________________________________________________________________________________________________
batch_normalization_71 (BatchNo (None, 17, 17, 192)  576         conv2d_71[0][0]
____________________________________________________________________________________________________
batch_normalization_75 (BatchNo (None, 17, 17, 192)  576         conv2d_75[0][0]
____________________________________________________________________________________________________
activation_71 (Activation)      (None, 17, 17, 192)  0           batch_normalization_71[0][0]
____________________________________________________________________________________________________
activation_75 (Activation)      (None, 17, 17, 192)  0           batch_normalization_75[0][0]
____________________________________________________________________________________________________
conv2d_72 (Conv2D)              (None, 8, 8, 320)    552960      activation_71[0][0]
____________________________________________________________________________________________________
conv2d_76 (Conv2D)              (None, 8, 8, 192)    331776      activation_75[0][0]
____________________________________________________________________________________________________
batch_normalization_72 (BatchNo (None, 8, 8, 320)    960         conv2d_72[0][0]
____________________________________________________________________________________________________
batch_normalization_76 (BatchNo (None, 8, 8, 192)    576         conv2d_76[0][0]
____________________________________________________________________________________________________
activation_72 (Activation)      (None, 8, 8, 320)    0           batch_normalization_72[0][0]
____________________________________________________________________________________________________
activation_76 (Activation)      (None, 8, 8, 192)    0           batch_normalization_76[0][0]
____________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 8, 8, 768)    0           mixed7[0][0]
____________________________________________________________________________________________________
mixed8 (Concatenate)            (None, 8, 8, 1280)   0           activation_72[0][0]
                                                                 activation_76[0][0]
                                                                 max_pooling2d_4[0][0]
____________________________________________________________________________________________________
conv2d_81 (Conv2D)              (None, 8, 8, 448)    573440      mixed8[0][0]
____________________________________________________________________________________________________
batch_normalization_81 (BatchNo (None, 8, 8, 448)    1344        conv2d_81[0][0]
____________________________________________________________________________________________________
activation_81 (Activation)      (None, 8, 8, 448)    0           batch_normalization_81[0][0]
____________________________________________________________________________________________________
conv2d_78 (Conv2D)              (None, 8, 8, 384)    491520      mixed8[0][0]
____________________________________________________________________________________________________
conv2d_82 (Conv2D)              (None, 8, 8, 384)    1548288     activation_81[0][0]
____________________________________________________________________________________________________
batch_normalization_78 (BatchNo (None, 8, 8, 384)    1152        conv2d_78[0][0]
____________________________________________________________________________________________________
batch_normalization_82 (BatchNo (None, 8, 8, 384)    1152        conv2d_82[0][0]
____________________________________________________________________________________________________
activation_78 (Activation)      (None, 8, 8, 384)    0           batch_normalization_78[0][0]
____________________________________________________________________________________________________
activation_82 (Activation)      (None, 8, 8, 384)    0           batch_normalization_82[0][0]
```

```
__________________________________________________________________________________________________
conv2d_79 (Conv2D)              (None, 8, 8, 384)    442368      activation_78[0][0]
__________________________________________________________________________________________________
conv2d_80 (Conv2D)              (None, 8, 8, 384)    442368      activation_78[0][0]
__________________________________________________________________________________________________
conv2d_83 (Conv2D)              (None, 8, 8, 384)    442368      activation_82[0][0]
__________________________________________________________________________________________________
conv2d_84 (Conv2D)              (None, 8, 8, 384)    442368      activation_82[0][0]
__________________________________________________________________________________________________
average_pooling2d_8 (AveragePoo (None, 8, 8, 1280)   0           mixed8[0][0]
__________________________________________________________________________________________________
conv2d_77 (Conv2D)              (None, 8, 8, 320)    409600      mixed8[0][0]
__________________________________________________________________________________________________
batch_normalization_79 (BatchNo (None, 8, 8, 384)    1152        conv2d_79[0][0]
__________________________________________________________________________________________________
batch_normalization_80 (BatchNo (None, 8, 8, 384)    1152        conv2d_80[0][0]
__________________________________________________________________________________________________
batch_normalization_83 (BatchNo (None, 8, 8, 384)    1152        conv2d_83[0][0]
__________________________________________________________________________________________________
batch_normalization_84 (BatchNo (None, 8, 8, 384)    1152        conv2d_84[0][0]
__________________________________________________________________________________________________
conv2d_85 (Conv2D)              (None, 8, 8, 192)    245760      average_pooling2d_8[0][0]
__________________________________________________________________________________________________
batch_normalization_77 (BatchNo (None, 8, 8, 320)    960         conv2d_77[0][0]
__________________________________________________________________________________________________
activation_79 (Activation)      (None, 8, 8, 384)    0           batch_normalization_79[0][0]
__________________________________________________________________________________________________
activation_80 (Activation)      (None, 8, 8, 384)    0           batch_normalization_80[0][0]
__________________________________________________________________________________________________
activation_83 (Activation)      (None, 8, 8, 384)    0           batch_normalization_83[0][0]
__________________________________________________________________________________________________
activation_84 (Activation)      (None, 8, 8, 384)    0           batch_normalization_84[0][0]
__________________________________________________________________________________________________
batch_normalization_85 (BatchNo (None, 8, 8, 192)    576         conv2d_85[0][0]
__________________________________________________________________________________________________
activation_77 (Activation)      (None, 8, 8, 320)    0           batch_normalization_77[0][0]
__________________________________________________________________________________________________
mixed9_0 (Concatenate)          (None, 8, 8, 768)    0           activation_79[0][0]
                                                                 activation_80[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 8, 8, 768)    0           activation_83[0][0]
                                                                 activation_84[0][0]
__________________________________________________________________________________________________
activation_85 (Activation)      (None, 8, 8, 192)    0           batch_normalization_85[0][0]
```

```
----------------------------------------------------------------------------------------------------
mixed9 (Concatenate)            (None, 8, 8, 2048)   0           activation_77[0][0]
                                                                 mixed9_0[0][0]
                                                                 concatenate_1[0][0]
                                                                 activation_85[0][0]
----------------------------------------------------------------------------------------------------
conv2d_90 (Conv2D)              (None, 8, 8, 448)    917504      mixed9[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_90 (BatchNo (None, 8, 8, 448)    1344        conv2d_90[0][0]
----------------------------------------------------------------------------------------------------
activation_90 (Activation)      (None, 8, 8, 448)    0           batch_normalization_90[0][0]
----------------------------------------------------------------------------------------------------
conv2d_87 (Conv2D)              (None, 8, 8, 384)    786432      mixed9[0][0]
----------------------------------------------------------------------------------------------------
conv2d_91 (Conv2D)              (None, 8, 8, 384)    1548288     activation_90[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_87 (BatchNo (None, 8, 8, 384)    1152        conv2d_87[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_91 (BatchNo (None, 8, 8, 384)    1152        conv2d_91[0][0]
----------------------------------------------------------------------------------------------------
activation_87 (Activation)      (None, 8, 8, 384)    0           batch_normalization_87[0][0]
----------------------------------------------------------------------------------------------------
activation_91 (Activation)      (None, 8, 8, 384)    0           batch_normalization_91[0][0]
----------------------------------------------------------------------------------------------------
conv2d_88 (Conv2D)              (None, 8, 8, 384)    442368      activation_87[0][0]
----------------------------------------------------------------------------------------------------
conv2d_89 (Conv2D)              (None, 8, 8, 384)    442368      activation_87[0][0]
----------------------------------------------------------------------------------------------------
conv2d_92 (Conv2D)              (None, 8, 8, 384)    442368      activation_91[0][0]
----------------------------------------------------------------------------------------------------
conv2d_93 (Conv2D)              (None, 8, 8, 384)    442368      activation_91[0][0]
----------------------------------------------------------------------------------------------------
average_pooling2d_9 (AveragePoo (None, 8, 8, 2048)   0           mixed9[0][0]
----------------------------------------------------------------------------------------------------
conv2d_86 (Conv2D)              (None, 8, 8, 320)    655360      mixed9[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_88 (BatchNo (None, 8, 8, 384)    1152        conv2d_88[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_89 (BatchNo (None, 8, 8, 384)    1152        conv2d_89[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_92 (BatchNo (None, 8, 8, 384)    1152        conv2d_92[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_93 (BatchNo (None, 8, 8, 384)    1152        conv2d_93[0][0]
----------------------------------------------------------------------------------------------------
```

```
conv2d_94 (Conv2D)              (None, 8, 8, 192)    393216      average_pooling2d_9[0][0]
__________________________________________________________________________________________________
batch_normalization_86 (BatchNo (None, 8, 8, 320)    960         conv2d_86[0][0]
__________________________________________________________________________________________________
activation_88 (Activation)      (None, 8, 8, 384)    0           batch_normalization_88[0][0]
__________________________________________________________________________________________________
activation_89 (Activation)      (None, 8, 8, 384)    0           batch_normalization_89[0][0]
__________________________________________________________________________________________________
activation_92 (Activation)      (None, 8, 8, 384)    0           batch_normalization_92[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, 8, 8, 384)    0           batch_normalization_93[0][0]
__________________________________________________________________________________________________
batch_normalization_94 (BatchNo (None, 8, 8, 192)    576         conv2d_94[0][0]
__________________________________________________________________________________________________
activation_86 (Activation)      (None, 8, 8, 320)    0           batch_normalization_86[0][0]
__________________________________________________________________________________________________
mixed9_1 (Concatenate)          (None, 8, 8, 768)    0           activation_88[0][0]
                                                                 activation_89[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 8, 8, 768)    0           activation_92[0][0]
                                                                 activation_93[0][0]
__________________________________________________________________________________________________
activation_94 (Activation)      (None, 8, 8, 192)    0           batch_normalization_94[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, 8, 8, 2048)   0           activation_86[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_2[0][0]
                                                                 activation_94[0][0]
==================================================================================================
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
```

As we can see, this component replicates the architecture of the InceptionV3 network. Each of the layers `mixed0` to `mixed10` are the concatenation nodes at the end of each convolution block, as shown in Section 3.2. Notice that `mixed10` is the last layer of this component since we have decided not to download the top classifier. Therefore in order to complete the network, again according to the block diagram of Section 3.2, we have added on top of this component the remaining layers, consisting on an Average Pooling, a Dropout, a Dense layer (with relu activation since it is not the final layer) and finally another Dense softmax layer. In the previous code, this is what happens with the sentences:

```
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(class_dim, activation = 'relu'))
model.add(layers.Dense(n_classes, activation='softmax'))
```

The `class_dim` and `n_classes` variables are the dimensions of the last two Dense layers. In the original InceptionV3 network these dimensions are 4096 and 1000 respectively (the ImageNet dataset consists of 1000 classes). In our case, and for the purposes of training on the full dataset, the number of `n_classes` is 73, and therefore, in order to control the complexity of the model, we have decided to limit `class_dim` to 512. We have decided to leave these parameters as variables for added flexibility in case we want to modify them later. For instance, in Chapter 5 we will reorganize the dataset in such a way that `n_classes` will be 42. If we print the summary of the full model the result is:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
inception_v3 (Model)         (None, 8, 8, 2048)        21802784  
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1049088   
_________________________________________________________________
dense_2 (Dense)              (None, 73)                37449     
=================================================================
Total params: 22,889,321
Trainable params: 22,854,889
Non-trainable params: 34,432
_________________________________________________________________
```

This shows the whole pretrained component previously studied as a block called `inception_v3` and after that the top classifier top that we added. The layers of the full model are stored as a list `model.layers`. Therefore, if we want to access the individual layers of the `inception_v3` component, we can do it with the list `model.layers[0].layers`. In order to do transfer learning we have to freeze all the parameters of the InceptionV3 component, which is what the loop of the last two lines of code in the `init_model()` function does.

After running the function, we can check the status of all the layers in the full model and in the pretrained component as follows:

Input:

```
for i, layer in enumerate(model.layers[0].layers):
    print(i, layer.name, ', Trainable: ', layer.trainable)

for i, layer in enumerate(model.layers):
    print(i, layer.name, ', Trainable: ', layer.trainable)
```

Output:

```
0 input_1 , Trainable:  False
1 conv2d_1 , Trainable:  False
2 batch_normalization_1 , Trainable:  False
3 activation_1 , Trainable:  False
4 conv2d_2 , Trainable:  False
5 batch_normalization_2 , Trainable:  False
6 activation_2 , Trainable:  False
7 conv2d_3 , Trainable:  False
8 batch_normalization_3 , Trainable:  False
9 activation_3 , Trainable:  False
10 max_pooling2d_1 , Trainable:  False
11 conv2d_4 , Trainable:  False
12 batch_normalization_4 , Trainable:  False
13 activation_4 , Trainable:  False
14 conv2d_5 , Trainable:  False
15 batch_normalization_5 , Trainable:  False
16 activation_5 , Trainable:  False
17 max_pooling2d_2 , Trainable:  False
18 conv2d_9 , Trainable:  False
19 batch_normalization_9 , Trainable:  False
20 activation_9 , Trainable:  False
21 conv2d_7 , Trainable:  False
22 conv2d_10 , Trainable:  False
23 batch_normalization_7 , Trainable:  False
24 batch_normalization_10 , Trainable:  False
25 activation_7 , Trainable:  False
26 activation_10 , Trainable:  False
27 average_pooling2d_1 , Trainable:  False
28 conv2d_6 , Trainable:  False
29 conv2d_8 , Trainable:  False
30 conv2d_11 , Trainable:  False
31 conv2d_12 , Trainable:  False
32 batch_normalization_6 , Trainable:  False
33 batch_normalization_8 , Trainable:  False
34 batch_normalization_11 , Trainable:  False
35 batch_normalization_12 , Trainable:  False
```

```
36 activation_6 , Trainable:  False
37 activation_8 , Trainable:  False
38 activation_11 , Trainable:  False
39 activation_12 , Trainable:  False
40 mixed0 , Trainable:  False
41 conv2d_16 , Trainable:  False
42 batch_normalization_16 , Trainable:  False
43 activation_16 , Trainable:  False
44 conv2d_14 , Trainable:  False
45 conv2d_17 , Trainable:  False
46 batch_normalization_14 , Trainable:  False
47 batch_normalization_17 , Trainable:  False
48 activation_14 , Trainable:  False
49 activation_17 , Trainable:  False
50 average_pooling2d_2 , Trainable:  False
51 conv2d_13 , Trainable:  False
52 conv2d_15 , Trainable:  False
53 conv2d_18 , Trainable:  False
54 conv2d_19 , Trainable:  False
55 batch_normalization_13 , Trainable:  False
56 batch_normalization_15 , Trainable:  False
57 batch_normalization_18 , Trainable:  False
58 batch_normalization_19 , Trainable:  False
59 activation_13 , Trainable:  False
60 activation_15 , Trainable:  False
61 activation_18 , Trainable:  False
62 activation_19 , Trainable:  False
63 mixed1 , Trainable:  False
64 conv2d_23 , Trainable:  False
65 batch_normalization_23 , Trainable:  False
66 activation_23 , Trainable:  False
67 conv2d_21 , Trainable:  False
68 conv2d_24 , Trainable:  False
69 batch_normalization_21 , Trainable:  False
70 batch_normalization_24 , Trainable:  False
71 activation_21 , Trainable:  False
72 activation_24 , Trainable:  False
73 average_pooling2d_3 , Trainable:  False
74 conv2d_20 , Trainable:  False
75 conv2d_22 , Trainable:  False
76 conv2d_25 , Trainable:  False
77 conv2d_26 , Trainable:  False
78 batch_normalization_20 , Trainable:  False
79 batch_normalization_22 , Trainable:  False
```

```
80 batch_normalization_25 , Trainable:  False
81 batch_normalization_26 , Trainable:  False
82 activation_20 , Trainable:  False
83 activation_22 , Trainable:  False
84 activation_25 , Trainable:  False
85 activation_26 , Trainable:  False
86 mixed2 , Trainable:  False
87 conv2d_28 , Trainable:  False
88 batch_normalization_28 , Trainable:  False
89 activation_28 , Trainable:  False
90 conv2d_29 , Trainable:  False
91 batch_normalization_29 , Trainable:  False
92 activation_29 , Trainable:  False
93 conv2d_27 , Trainable:  False
94 conv2d_30 , Trainable:  False
95 batch_normalization_27 , Trainable:  False
96 batch_normalization_30 , Trainable:  False
97 activation_27 , Trainable:  False
98 activation_30 , Trainable:  False
99 max_pooling2d_3 , Trainable:  False
100 mixed3 , Trainable:  False
101 conv2d_35 , Trainable:  False
102 batch_normalization_35 , Trainable:  False
103 activation_35 , Trainable:  False
104 conv2d_36 , Trainable:  False
105 batch_normalization_36 , Trainable:  False
106 activation_36 , Trainable:  False
107 conv2d_32 , Trainable:  False
108 conv2d_37 , Trainable:  False
109 batch_normalization_32 , Trainable:  False
110 batch_normalization_37 , Trainable:  False
111 activation_32 , Trainable:  False
112 activation_37 , Trainable:  False
113 conv2d_33 , Trainable:  False
114 conv2d_38 , Trainable:  False
115 batch_normalization_33 , Trainable:  False
116 batch_normalization_38 , Trainable:  False
117 activation_33 , Trainable:  False
118 activation_38 , Trainable:  False
119 average_pooling2d_4 , Trainable:  False
120 conv2d_31 , Trainable:  False
121 conv2d_34 , Trainable:  False
122 conv2d_39 , Trainable:  False
123 conv2d_40 , Trainable:  False
```

```
124 batch_normalization_31 , Trainable:  False
125 batch_normalization_34 , Trainable:  False
126 batch_normalization_39 , Trainable:  False
127 batch_normalization_40 , Trainable:  False
128 activation_31 , Trainable:  False
129 activation_34 , Trainable:  False
130 activation_39 , Trainable:  False
131 activation_40 , Trainable:  False
132 mixed4 , Trainable:  False
133 conv2d_45 , Trainable:  False
134 batch_normalization_45 , Trainable:  False
135 activation_45 , Trainable:  False
136 conv2d_46 , Trainable:  False
137 batch_normalization_46 , Trainable:  False
138 activation_46 , Trainable:  False
139 conv2d_42 , Trainable:  False
140 conv2d_47 , Trainable:  False
141 batch_normalization_42 , Trainable:  False
142 batch_normalization_47 , Trainable:  False
143 activation_42 , Trainable:  False
144 activation_47 , Trainable:  False
145 conv2d_43 , Trainable:  False
146 conv2d_48 , Trainable:  False
147 batch_normalization_43 , Trainable:  False
148 batch_normalization_48 , Trainable:  False
149 activation_43 , Trainable:  False
150 activation_48 , Trainable:  False
151 average_pooling2d_5 , Trainable:  False
152 conv2d_41 , Trainable:  False
153 conv2d_44 , Trainable:  False
154 conv2d_49 , Trainable:  False
155 conv2d_50 , Trainable:  False
156 batch_normalization_41 , Trainable:  False
157 batch_normalization_44 , Trainable:  False
158 batch_normalization_49 , Trainable:  False
159 batch_normalization_50 , Trainable:  False
160 activation_41 , Trainable:  False
161 activation_44 , Trainable:  False
162 activation_49 , Trainable:  False
163 activation_50 , Trainable:  False
164 mixed5 , Trainable:  False
165 conv2d_55 , Trainable:  False
166 batch_normalization_55 , Trainable:  False
167 activation_55 , Trainable:  False
```

```
168 conv2d_56 , Trainable:  False
169 batch_normalization_56 , Trainable:  False
170 activation_56 , Trainable:  False
171 conv2d_52 , Trainable:  False
172 conv2d_57 , Trainable:  False
173 batch_normalization_52 , Trainable:  False
174 batch_normalization_57 , Trainable:  False
175 activation_52 , Trainable:  False
176 activation_57 , Trainable:  False
177 conv2d_53 , Trainable:  False
178 conv2d_58 , Trainable:  False
179 batch_normalization_53 , Trainable:  False
180 batch_normalization_58 , Trainable:  False
181 activation_53 , Trainable:  False
182 activation_58 , Trainable:  False
183 average_pooling2d_6 , Trainable:  False
184 conv2d_51 , Trainable:  False
185 conv2d_54 , Trainable:  False
186 conv2d_59 , Trainable:  False
187 conv2d_60 , Trainable:  False
188 batch_normalization_51 , Trainable:  False
189 batch_normalization_54 , Trainable:  False
190 batch_normalization_59 , Trainable:  False
191 batch_normalization_60 , Trainable:  False
192 activation_51 , Trainable:  False
193 activation_54 , Trainable:  False
194 activation_59 , Trainable:  False
195 activation_60 , Trainable:  False
196 mixed6 , Trainable:  False
197 conv2d_65 , Trainable:  False
198 batch_normalization_65 , Trainable:  False
199 activation_65 , Trainable:  False
200 conv2d_66 , Trainable:  False
201 batch_normalization_66 , Trainable:  False
202 activation_66 , Trainable:  False
203 conv2d_62 , Trainable:  False
204 conv2d_67 , Trainable:  False
205 batch_normalization_62 , Trainable:  False
206 batch_normalization_67 , Trainable:  False
207 activation_62 , Trainable:  False
208 activation_67 , Trainable:  False
209 conv2d_63 , Trainable:  False
210 conv2d_68 , Trainable:  False
211 batch_normalization_63 , Trainable:  False
```

```
212 batch_normalization_68 , Trainable:  False
213 activation_63 , Trainable:  False
214 activation_68 , Trainable:  False
215 average_pooling2d_7 , Trainable:  False
216 conv2d_61 , Trainable:  False
217 conv2d_64 , Trainable:  False
218 conv2d_69 , Trainable:  False
219 conv2d_70 , Trainable:  False
220 batch_normalization_61 , Trainable:  False
221 batch_normalization_64 , Trainable:  False
222 batch_normalization_69 , Trainable:  False
223 batch_normalization_70 , Trainable:  False
224 activation_61 , Trainable:  False
225 activation_64 , Trainable:  False
226 activation_69 , Trainable:  False
227 activation_70 , Trainable:  False
228 mixed7 , Trainable:  False
229 conv2d_73 , Trainable:  False
230 batch_normalization_73 , Trainable:  False
231 activation_73 , Trainable:  False
232 conv2d_74 , Trainable:  False
233 batch_normalization_74 , Trainable:  False
234 activation_74 , Trainable:  False
235 conv2d_71 , Trainable:  False
236 conv2d_75 , Trainable:  False
237 batch_normalization_71 , Trainable:  False
238 batch_normalization_75 , Trainable:  False
239 activation_71 , Trainable:  False
240 activation_75 , Trainable:  False
241 conv2d_72 , Trainable:  False
242 conv2d_76 , Trainable:  False
243 batch_normalization_72 , Trainable:  False
244 batch_normalization_76 , Trainable:  False
245 activation_72 , Trainable:  False
246 activation_76 , Trainable:  False
247 max_pooling2d_4 , Trainable:  False
248 mixed8 , Trainable:  False
249 conv2d_81 , Trainable:  False
250 batch_normalization_81 , Trainable:  False
251 activation_81 , Trainable:  False
252 conv2d_78 , Trainable:  False
253 conv2d_82 , Trainable:  False
254 batch_normalization_78 , Trainable:  False
255 batch_normalization_82 , Trainable:  False
```

```
256 activation_78 , Trainable:  False
257 activation_82 , Trainable:  False
258 conv2d_79 , Trainable:  False
259 conv2d_80 , Trainable:  False
260 conv2d_83 , Trainable:  False
261 conv2d_84 , Trainable:  False
262 average_pooling2d_8 , Trainable:  False
263 conv2d_77 , Trainable:  False
264 batch_normalization_79 , Trainable:  False
265 batch_normalization_80 , Trainable:  False
266 batch_normalization_83 , Trainable:  False
267 batch_normalization_84 , Trainable:  False
268 conv2d_85 , Trainable:  False
269 batch_normalization_77 , Trainable:  False
270 activation_79 , Trainable:  False
271 activation_80 , Trainable:  False
272 activation_83 , Trainable:  False
273 activation_84 , Trainable:  False
274 batch_normalization_85 , Trainable:  False
275 activation_77 , Trainable:  False
276 mixed9_0 , Trainable:  False
277 concatenate_1 , Trainable:  False
278 activation_85 , Trainable:  False
279 mixed9 , Trainable:  False
280 conv2d_90 , Trainable:  False
281 batch_normalization_90 , Trainable:  False
282 activation_90 , Trainable:  False
283 conv2d_87 , Trainable:  False
284 conv2d_91 , Trainable:  False
285 batch_normalization_87 , Trainable:  False
286 batch_normalization_91 , Trainable:  False
287 activation_87 , Trainable:  False
288 activation_91 , Trainable:  False
289 conv2d_88 , Trainable:  False
290 conv2d_89 , Trainable:  False
291 conv2d_92 , Trainable:  False
292 conv2d_93 , Trainable:  False
293 average_pooling2d_9 , Trainable:  False
294 conv2d_86 , Trainable:  False
295 batch_normalization_88 , Trainable:  False
296 batch_normalization_89 , Trainable:  False
297 batch_normalization_92 , Trainable:  False
298 batch_normalization_93 , Trainable:  False
299 conv2d_94 , Trainable:  False
```

```
300 batch_normalization_86 , Trainable:  False
301 activation_88 , Trainable:  False
302 activation_89 , Trainable:  False
303 activation_92 , Trainable:  False
304 activation_93 , Trainable:  False
305 batch_normalization_94 , Trainable:  False
306 activation_86 , Trainable:  False
307 mixed9_1 , Trainable:  False
308 concatenate_2 , Trainable:  False
309 activation_94 , Trainable:  False
310 mixed10 , Trainable:  False
0 inception_v3 , Trainable:  True
1 global_average_pooling2d_1 , Trainable:  True
2 dropout_1 , Trainable:  True
3 dense_1 , Trainable:  True
4 dense_2 , Trainable:  True
```

Notice how the first 311 layers correspond to the InceptionV3 network and are all non-trainable. The next part of the output, layers 0 to 4, correspond to the full model, for which layer 0 is the InceptionV3 component seen as a layer in the full model. This layer is declared trainable, however it doesn't have any trainable parameter has we have seen from the previous lines. It is important that the 0 layer of the full model be trainable since in the fine tuning process we will be unfreezing several of its internal layers, which must be seen as trainable for the full model.

For implementing the fine tuning process we create the function `unfreeze_cnn` which will selectively unfreeze layers of the pretrained component of the full model. The code is the following:

Input:

```python
def unfreeze_cnn(first, last):
    if last == 'end':
        for i, layer in enumerate(model.layers[0].layers[first:]):
            layer.trainable = True
    else:
        for i, layer in enumerate(model.layers[0].layers[first:last+1]):
            layer.trainable = True
    for i, layer in enumerate(model.layers[0].layers):
       print(i, layer.name, ', Trainable: ', layer.trainable)
    for i, layer in enumerate(model.layers):
       print(i, layer.name, ', Trainable: ', layer.trainable)
```

For instance, if we want to unfreeze the last inception block we would set the parameters as `first = 280` and `last = 'end'`. An extract of the result of applying the function would be:

Input:

```
unfreeze_cnn(280, 'end')
```

Output:

```
...
268 conv2d_85 , Trainable:  False
269 batch_normalization_77 , Trainable:  False
270 activation_79 , Trainable:  False
271 activation_80 , Trainable:  False
272 activation_83 , Trainable:  False
273 activation_84 , Trainable:  False
274 batch_normalization_85 , Trainable:  False
275 activation_77 , Trainable:  False
276 mixed9_0 , Trainable:  False
277 concatenate_1 , Trainable:  False
278 activation_85 , Trainable:  False
279 mixed9 , Trainable:  False
280 conv2d_90 , Trainable:  True
281 batch_normalization_90 , Trainable:  True
282 activation_90 , Trainable:  True
283 conv2d_87 , Trainable:  True
284 conv2d_91 , Trainable:  True
285 batch_normalization_87 , Trainable:  True
286 batch_normalization_91 , Trainable:  True
287 activation_87 , Trainable:  True
288 activation_91 , Trainable:  True
289 conv2d_88 , Trainable:  True
290 conv2d_89 , Trainable:  True
291 conv2d_92 , Trainable:  True
292 conv2d_93 , Trainable:  True
293 average_pooling2d_9 , Trainable:  True
294 conv2d_86 , Trainable:  True
295 batch_normalization_88 , Trainable:  True
296 batch_normalization_89 , Trainable:  True
297 batch_normalization_92 , Trainable:  True
298 batch_normalization_93 , Trainable:  True
299 conv2d_94 , Trainable:  True
300 batch_normalization_86 , Trainable:  True
301 activation_88 , Trainable:  True
302 activation_89 , Trainable:  True
```

```
303 activation_92 , Trainable:  True
304 activation_93 , Trainable:  True
305 batch_normalization_94 , Trainable:  True
306 activation_86 , Trainable:  True
307 mixed9_1 , Trainable:  True
308 concatenate_2 , Trainable:  True
309 activation_94 , Trainable:  True
310 mixed10 , Trainable:  True
0 inception_v3 , Trainable:  True
1 global_average_pooling2d_1 , Trainable:  True
2 dropout_1 , Trainable:  True
3 dense_1 , Trainable:  True
4 dense_2 , Trainable:  True
```

4.2 Generators and Image Augmentation

As mentioned before, in order to train high capacity deep models we need a lot of images, which make impossible the usual approach of feeding the network with a numpy array storing all the images. For this reason we are going to use the technique of generators. In simple terms, and in our context, a generator will be composed of two elements, a `flow_from_directory` method and a `ImageDataGenerator` object. The method will read images from disk and will pass them to the object, which will then apply several transformations. These two processes will be encapsulated in the generator, which will execute the whole process in batches, producing numpy arrays that will be used for the training, predicting and validating steps.

We will take now a small detour in order to delve deeper into an important feature (and a standard procedure in training deep networks for image recognition) of the generator: image augmentation. Image augmentation is a way to synthetically produce more images than the ones we have available. This works by applying a number of transformations to a given image in a randomized and unlimited way, therefore resulting in an infinite set of images representing the same classification category. These transformations include rotations, stretching, zooming, altering the color scheme, flipping the image around an axis, etc... In this way we are effectively enlarging immensely our training set and therefore we will hopefully be able to deal with very high capacity models without underfitting or overfitting them. We will now provide an example of how data augmentation works in the Beauty dataset. This code will not form part of the main scripts training our model, and is included here only for the purpose of illustrating the kind of new images that we will artificially generate in the training process.

We will choose a single image and load it into an array. Then, a `ImageDataGenerator` will generate an infinite number of modified images within certain transformation parameters. We will stop the process after 9 images.

Input:

```
import os
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
%matplotlib inline

directory = '/home/miguelyogur/tfm/dataset_beauty/train/Body_AntiCelluliteCream'
fnames = [os.path.join(directory, fname) for fname in os.listdir(directory)]
img_path = fnames[8]
img = image.load_img(img_path)

print('Original image')
plt.imshow(img)
plt.axis('off')
plt.show()

datagen = ImageDataGenerator(
      rotation_range=30,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.1,
      horizontal_flip=True,
      fill_mode='nearest')

x = image.img_to_array(img)
x = x.reshape((1,) + x.shape)

print('Augmented images')
fig=plt.figure(figsize = (15, 15))
i = 1
for batch in datagen.flow(x, batch_size=1):
    fig.add_subplot(3,3, i)
    plt.imshow(image.array_to_img(batch[0]))
    plt.axis('off')
    i += 1
    if i > 9:
        break
plt.show()
```

The outputs images of this example are shown in Figure 4.1.

Returning to our main goal, we will now set up two generators, one for training and one for validation and testing. Both generators will first read images from disk, which will be further processed by the `ImageDataGenerator`. This processing will consist on converting them in batches of numpy arrays, min-max normalizing them (in this context this consists of dividing each pixel intensity by the maximum possible value, which is 255 for 8-bit depth images) and then applying image augmentation. Notice that image augmentation will be performed only in the training set, since when evaluating the model we need to do it on real images. The code for creating the generators is:

Input:

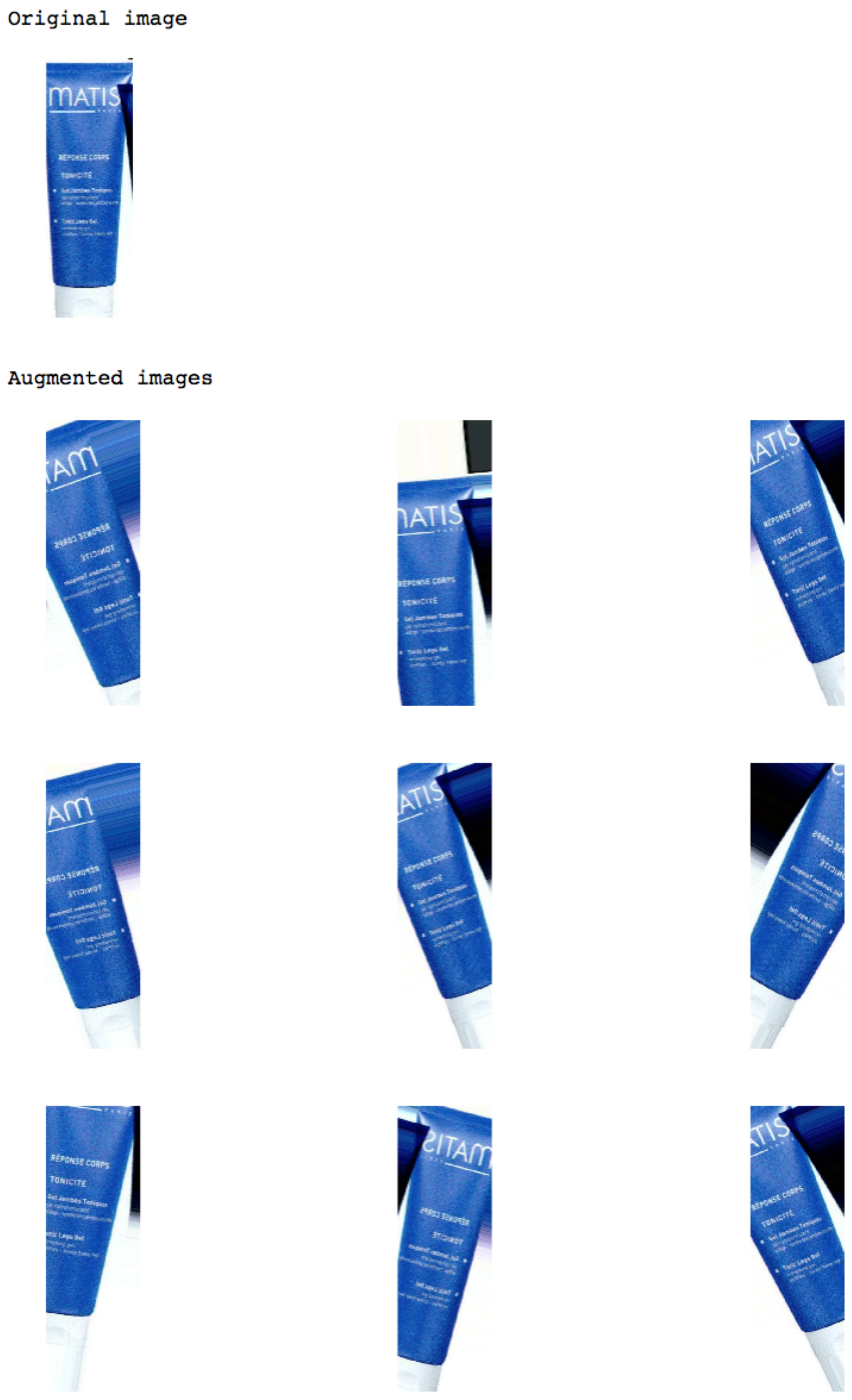
```
train_datagen = ImageDataGenerator(
      rescale=1./255
      ,rotation_range=40
      ,width_shift_range=0.2
      ,height_shift_range=0.2
      ,shear_range=0.2
      ,zoom_range=0.2
      ,horizontal_flip=True
      ,fill_mode='nearest'
      )

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(input_dim, input_dim),
        batch_size=64,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(input_dim, input_dim),
        batch_size=128,
        class_mode='categorical')
```

The generators include the choice of batch size (that will be the number of images included in each array passed to the network when fitting or predicting). Also, they include as parameters the dimension of the images, which has already been used when constructing the full model, and the folders from which to extract the training, validation or test images. These routes will be declared when the final piece of code is assembled. Finally, note that we use the categorical



```
Original image
```

```
Augmented images
```

Figure 4.1: Artificial augmentation of a single image.

class mode since we will fit a classification algorithm with more than two classes.

4.3 Functions for Training and Quality

We now create the function that will take care of the training process. This will consist of compiling the model and then fitting it to the training set by means of the training generator previously constructed. Since we are dealing with a lot of data, each epoch can take a lot of time. Therefore we will define a list of callbacks that essentially will stop the training process if the validation loss does not improve for 3 consecutive epochs, and the best model will be saved. The validation process will be done after each epoch on the validation set via the validation generator.

One important point is that, as has been previously mentioned, the two generators that we introduced will generate endless batches of images. For this reason it is necessary to impose a halt mechanism for both the training of each epoch and the validation process. This will be done in the `steps_per_epoch` and `validation_steps` respectively, which are chosen in such a way that these number of steps, together with the number of images produced by the batch generator for each step, take care of the totality of images in the corresponding set. The code for the training function is:

Input:

```
def train_model(lr=2e-4, n_epochs=100):

    callbacks_list = [
        EarlyStopping(monitor='val_loss', patience=3),
        ModelCheckpoint(filepath='best_model.h5', monitor='val_loss',
                                        save_best_only=True)
                    ]

    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizers.RMSprop(lr=lr),
                  metrics=['acc'])

    history = model.fit_generator(
           train_generator,
           steps_per_epoch= 911,
           epochs=n_epochs,
           callbacks=callbacks_list,
           validation_data=validation_generator,
           validation_steps=57,
        workers = 4)

    validation_loss, validation_acc = model.evaluate_generator(
        generator=validation_generator, steps = 57, workers = 4
        )
    print('last␣model␣validation␣loss:', validation_loss,
        'last␣model␣validation␣acc:', validation_acc)

    # plot acc and loss curves for training and validation

    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(len(acc))

    plt.plot(epochs, acc, 'bo', label='Training␣acc')
    plt.plot(epochs, val_acc, 'b', label='Validation␣acc')
    plt.title('Training␣and␣validation␣accuracy')
    plt.legend()

    plt.figure()

    plt.plot(epochs, loss, 'bo', label='Training␣loss')
    plt.plot(epochs, val_loss, 'b', label='Validation␣loss')
    plt.title('Training␣and␣validation␣loss')
    plt.legend()

    plt.show()
```

Notice that the second part of the function includes a preliminary assessment of the quality of the model, once it has stopped by virtue of the callback clause. This part will produce estimations of the validation loss and accuracy based on the last epoch (which is actually three epochs later than the best model saved) and also a graphic history of the accuracy and loss for both the training and validation sets. In this way it will be easier to get an idea of the bias-variance trade of the algorithm as we add epochs to the training routine.

The last piece of code that we are going to need is a more refined measure of the quality of the model. This will be obtained by a new function that will produce an accuracy indicator on the validation set well as a confusion matrix for the best model saved. This will be useful later for evaluating the accuracy of the model on each category and for grouping those that are mistaken by the algorithm into a single one, therefore reorganizing the images into a new dataset with less categories, which hopefully will improve the model's performance. This will be the topic of Chapter 5. The code for this function is:

Input:

```
val_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(input_dim, input_dim),
        batch_size=1,
        class_mode='categorical')

# Get labels
label_map = list(range(len(val_generator.class_indices)))
for key, value in val_generator.class_indices.items():
    label_map[value] = key

def get_quality_model(model):
    y_true = []
    y_pred = []

    i = 0
    for input_image, input_label in val_generator:
        y_true.append((np.argmax(input_label)))
        y_pred.append((np.argmax(model.predict(input_image))))
        i = i + 1
        if i > len(val_generator): break

    y_true = np.asarray(y_true)
    y_pred = np.asarray(y_pred)

    cf = confusion_matrix(y_true, y_pred)
    cf_pd = pd.DataFrame(cf, index = label_map, columns = label_map)
    cf_pd.to_csv('best_model_confusion_matrix.csv', sep='\t')

    print('MODEL_RESULTS: ')
    print('Validation accuracy: ', np.mean(y_true == y_pred))
```

Before the function we have defined a new generator exclusively for validating the model after the last epoch has finished. It works by feeding images to the trained model one by one (batch size is 1), predicting the class of each image, and then computing the accuracy over all the predictions on the validation set. We have also created a list with all the names of the labels that will be used later for displaying purposes.

The function `get_quality_model` will use the generator we just created to perform an accuracy analysis on the validation set. It will also create a confusion matrix that will be converted to a pandas dataframe. The previously created list of labels will be then used to label the rows and columns of the dataframe for further analysis. This confusion matrix is saved to disk.

4.4 Training the Model

In this section we will present the full script for training and assessing the model with transfer learning and also with fine tuning. The code that will take care of the whole process is the following:

Input:

```
#      MODEL InceptionV3
#      WITH DATA AUGMENTATION
#      FINE TUNING

##############################
##############################
input_dim  = 299 #input dimension of the pretrained network.
                              #the input shape will be (input_dim, input_dim, 3)
class_dim  = 512 #number of nodes of the last Dense layer before the classifier
l_unfreeze = 280 #unfreeze last 1 blocks of inceptionV3 and train
n_classes = 73 #number of classes
base_dir = '/home/miguelro1976/tfm/dataset_beauty' #directory of the dataset
##############################
##############################

import os, shutil
from keras import models
from keras import layers
from keras import optimizers
from keras.applications import InceptionV3
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.models import load_model
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
import itertools
```

```
#folders names

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

'''
Define train data generator with image augmentation and test data generator without
The test generator is used for both validation and test data
'''
train_datagen = ImageDataGenerator(
      rescale=1./255
      ,rotation_range=40
      ,width_shift_range=0.2
      ,height_shift_range=0.2
      ,shear_range=0.2
      ,zoom_range=0.2
      ,horizontal_flip=True
      ,fill_mode='nearest'
      )

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(input_dim, input_dim),
        batch_size=64,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(input_dim, input_dim),
        batch_size=128,
        class_mode='categorical')

'''
Functions for unfreezing layers in the pretrained cnn and for training the model.
The arguments are:

unfreeze_cnn:
- first and last (both included) layers to be unfrozen

train_model:
```

```
- lr: learning rate of the optimizer
- n_epochs: maximum number of epochs to train
'''

def unfreeze_cnn(first, last):
    if last == 'end':
        for i, layer in enumerate(model.layers[0].layers[first:]):
            layer.trainable = True
    else:
        for i, layer in enumerate(model.layers[0].layers[first:last+1]):
            layer.trainable = True
    for i, layer in enumerate(model.layers[0].layers):
       print(i, layer.name, ', Trainable: ', layer.trainable)
    for i, layer in enumerate(model.layers):
       print(i, layer.name, ', Trainable: ', layer.trainable)


def train_model(lr=2e-4, n_epochs=100):

    callbacks_list = [
        EarlyStopping(monitor='val_loss', patience=3),
        ModelCheckpoint(filepath='best_model.h5', monitor='val_loss',
                            save_best_only=True)
                    ]

    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizers.RMSprop(lr=lr),
                  metrics=['acc'])

    history = model.fit_generator(
          train_generator,
          steps_per_epoch= 911,
          epochs=n_epochs,
          callbacks=callbacks_list,
          validation_data=validation_generator,
          validation_steps=57,
          workers = 4)

    validation_loss, validation_acc = model.evaluate_generator(
        generator=validation_generator, steps = 57, workers = 4
        )
    print('last_model_validation_loss:', validation_loss,
            'last_model_validation_acc:', validation_acc)
    # plot acc and loss curves for training and validation

    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(len(acc))

    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.figure()

    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()

'''
Define a generator for validation purposes on the already trained model
We also produce a list of labels for the categories which will be used for naming
rows and colums in the confusion matrix
'''

val_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(input_dim, input_dim),
        batch_size=1,
        class_mode='categorical')

# Get labels
label_map = list(range(len(val_generator.class_indices)))
for key, value in val_generator.class_indices.items():
    label_map[value] = key


'''
Define a function for testing the quality of the best model saved
in one iteration of the training process. It takes as arguments:
```

```
- model: the model name given when loading it from disk
'''


def get_quality_model(model):
    y_true = []
    y_pred = []

    i = 0
    for input_image, input_label in val_generator:
        y_true.append((np.argmax(input_label)))
        y_pred.append((np.argmax(model.predict(input_image))))
        i = i + 1
        if i > len(val_generator): break

    y_true = np.asarray(y_true)
    y_pred = np.asarray(y_pred)

    cf = confusion_matrix(y_true, y_pred)
    cf_pd = pd.DataFrame(cf, index = label_map, columns = label_map)
    cf_pd.to_csv('best_model_confusion_matrix.csv', sep='\t')

    print('MODEL_RESULTS:')
    print('Validation_accuracy:_', np.mean(y_true == y_pred))

'''
We set the model and show its architecture for
future reference when freezing/unfreezing nodes
'''
def init_model():
    conv_base = InceptionV3(weights='imagenet',
                            include_top=False,
                            input_shape=(input_dim, input_dim, 3)
                           )

    #print(conv_base.summary())

    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(class_dim, activation = 'relu'))
    model.add(layers.Dense(n_classes, activation='softmax'))
    #print(model.summary())
    '''
    Start by freezing the pretrained cnn completely and showing the trainability
    status of all layers in the cnn and in the full model
    '''

    for layer in model.layers[0].layers:
        layer.trainable = False

#   for i, layer in enumerate(model.layers[0].layers):
#       print(i, layer.name, ', Trainable: ', layer.trainable)
#
#    for i, layer in enumerate(model.layers):
#       print(i, layer.name, ', Trainable: ', layer.trainable)
    return model

########################
########################
#reset model
model = init_model()
'''
unfreeze layers for fine tuning
this line should be commented for transfer learning, but not for fine tuning
'''
#unfreeze_cnn(l_unfreeze, 'end')
#train model an get quality
train_model()
best_model = load_model('best_model.h5')
cf, cr = get_quality_model(best_model)
############################
############################
```

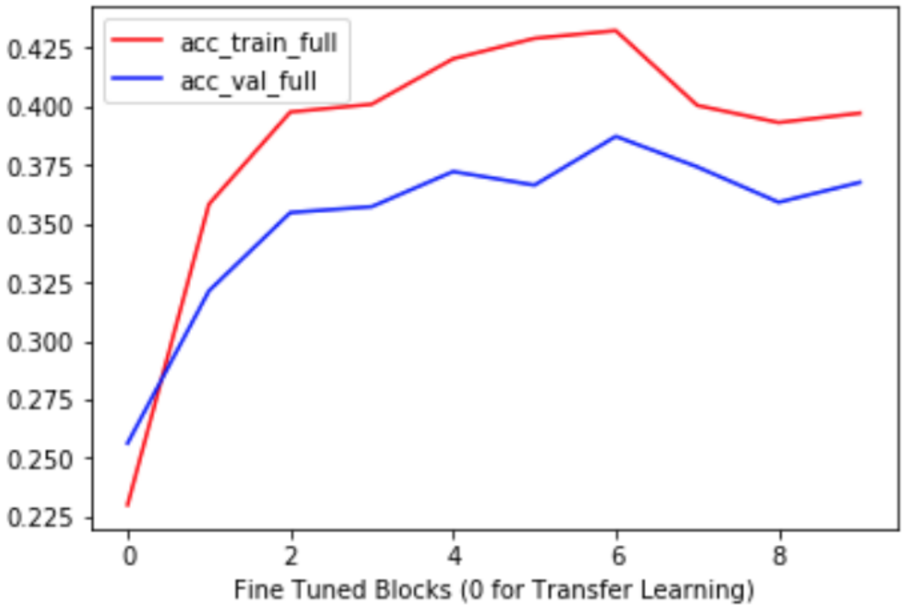
As we can see, all the necessary parameters are introduced at the beginning of the script, then all the functions are defined and finally, at the bottom of the script, the initialization, training, and quality assessment of the model are done. In this part, there is a line invoking the `unfreeze_cnn` function. Notice that this line should be omitted for the transfer learning, and it should be used for fine tuning. According to the output of this function showed in page 47, the values of the `l_unfreeze` parameter are:

- 280 for fine tuning one block,
- 249 for fine tuning two blocks,
- 229 for fine tuning three blocks,

- 197 for fine tuning four blocks,
- 165 for fine tuning five blocks,
- 133 for fine tuning six blocks,
- etc. . .

4.5 Results and Quality of the Model

All the outputs of each transfer learning or fine tuning step are collected in Section 8.1. We have trained and evaluated models up to fine tuning 9 inception blocks. We find an increase in the validation accuracy as we unfreeze more blocks, up to a point where the model cannot learn anymore. Figure 4.2 shows the comparison of accuracy and loss on training and validation sets for each training strategy. As we can see, with this approach, the best result occurs when the last 6 inception blocks are fine tuned, reaching a validation accuracy of 38.73%.



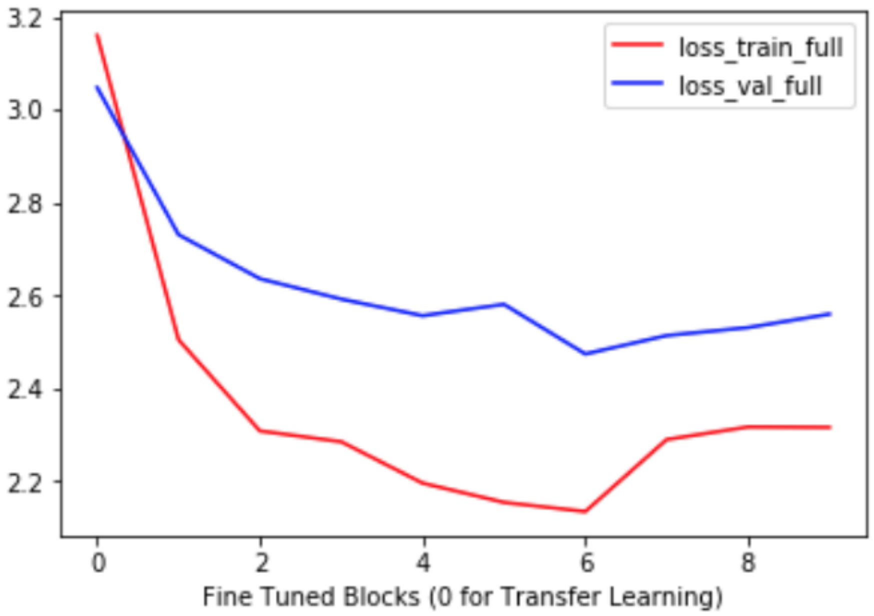


Figure 4.2: Quality of the different fine tuned models for the Beauty dataset using the InceptionV3 network.

Chapter 5

Grouping Categories

In this chapter we will try to improve the accuracy results of our model by reducing the number of categories of our dataset. To this end, we will study the confusion matrix for the best model obtained by the methods of Chapter 4. We will then identify the categories that are most frequently mixed by the algorithm and regroup them, obtaining a dataset of the same size, but with 42 classes instead of 73. Afterwards the same steps as in Chapter 4 for preprocessing, training and assessing the models will be carried out for this grouped dataset. The comparison of the different quality scores for the model applied to the original categories set or the new grouped categories set will be the main focus of Chapter 6.

5.1 The Confusion Matrix

The confusion matrix for the best model obtained in the previous chapter is a 73×73 pandas dataframe that is best visualized with a color scheme as in Figure 5.1. This matrix shows the results of testing the model on the validation set, which has 100 images for each category (except for one category with only 88 images). The legend, based on the number of examples in each case, is as follows:

- red: $n \geq 60$,
- orange $40 \leq n < 60$,
- yellow $20 \leq n < 40$,
- blue $10 \leq n < 20$,
- green $1 \leq n < 10$,
- black $n = 0$.

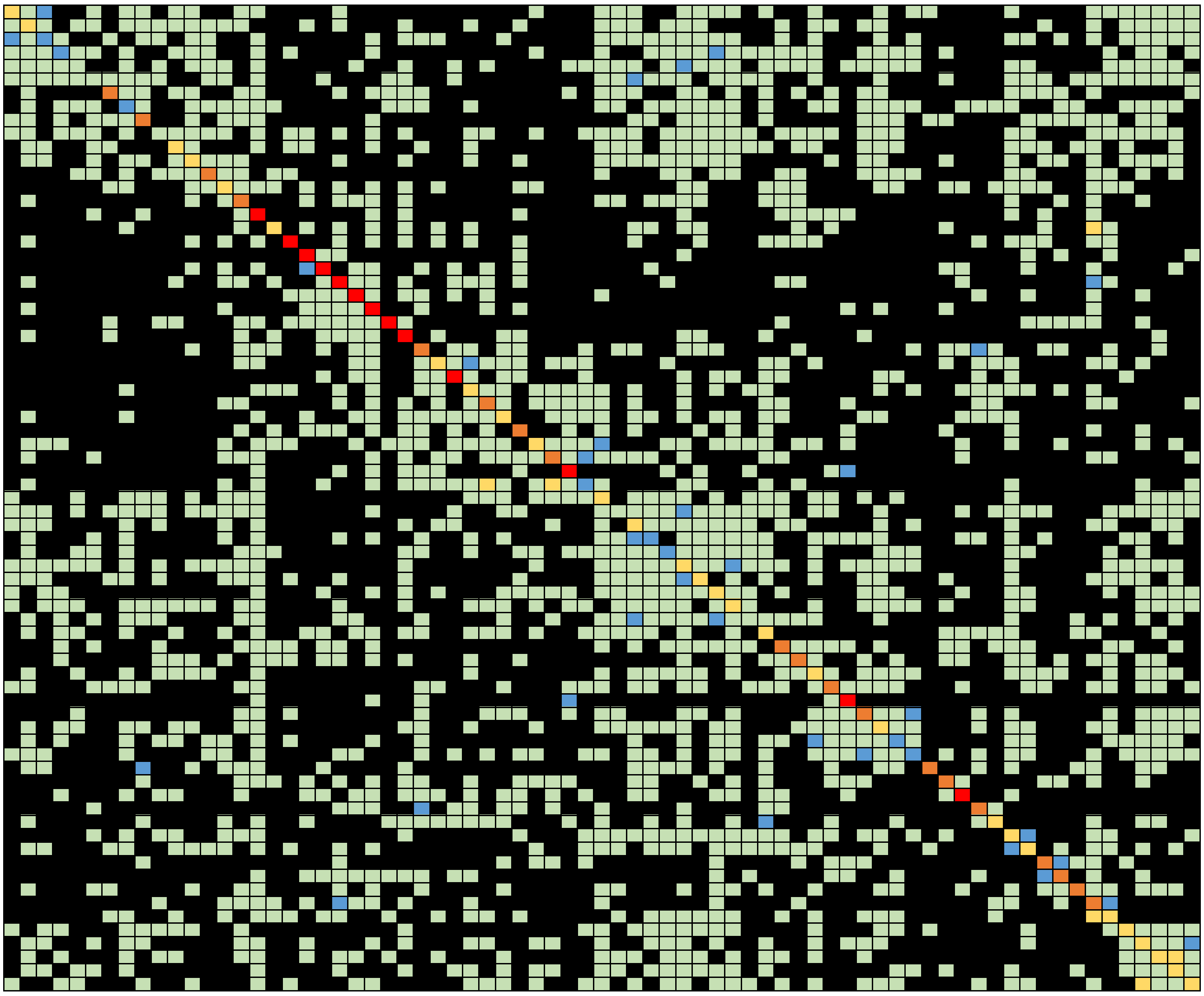


Figure 5.1: Confusion matrix for the Beauty dataset.

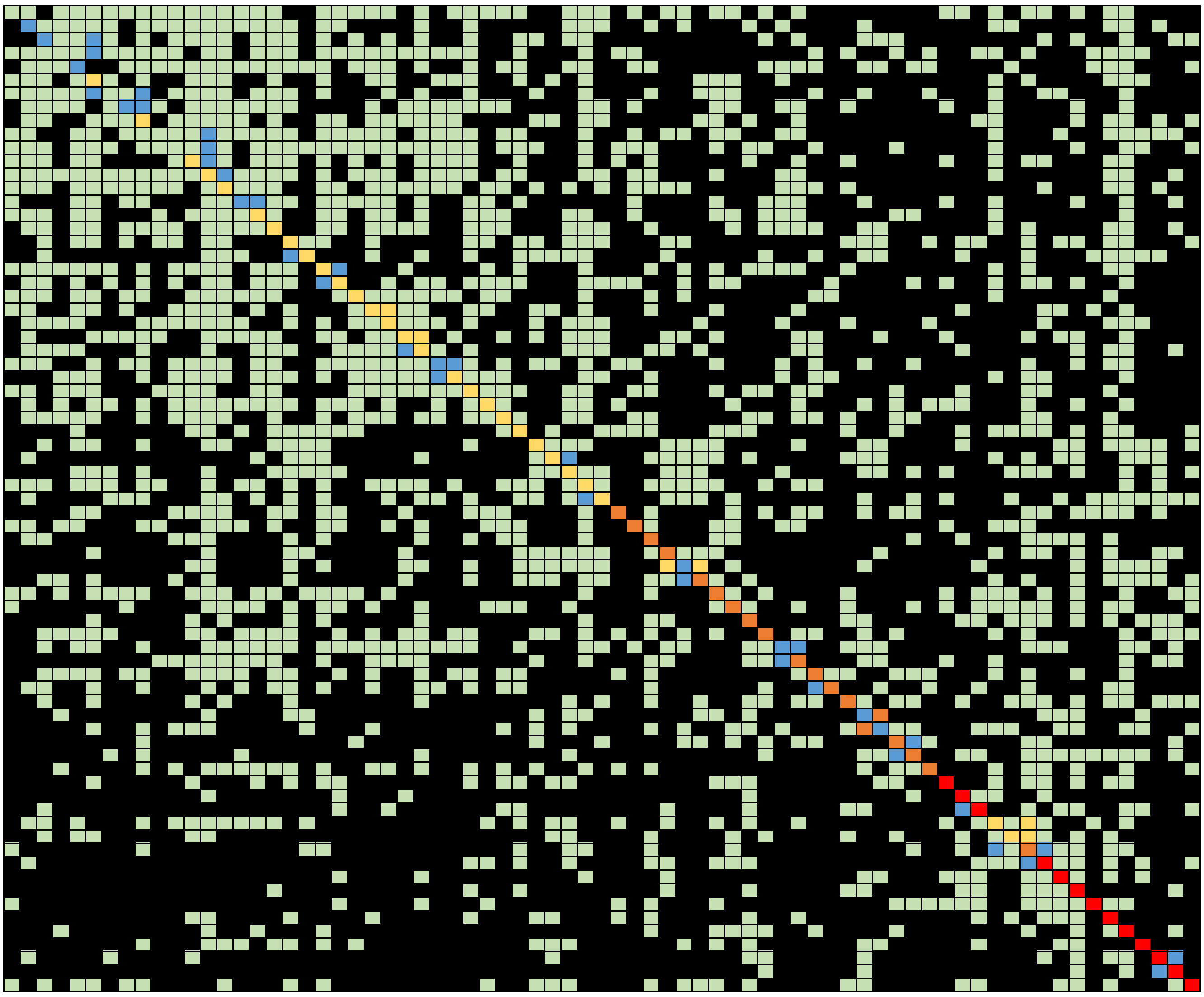


Figure 5.2: Reordering of categories for the Beauty dataset.

As we can see, the best accuracy results are along the diagonal, which is good. However, the network has trouble discerning among several classes. This behavior is more clearly exhibited if we wrangle with the order in which the categories are presented (which corresponds to the process of changing rows and columns in the matrix while preserving the diagonal). This is done in Figure 5.2.

5.2 Reorganization of the Categories

A natural way to group categories into clusters in such a way that the occupancy of the off-diagonal blocks is minimized is shown in Figure 5.3. Here, each red square block is a proposed new clusterization of several previously independent categories.

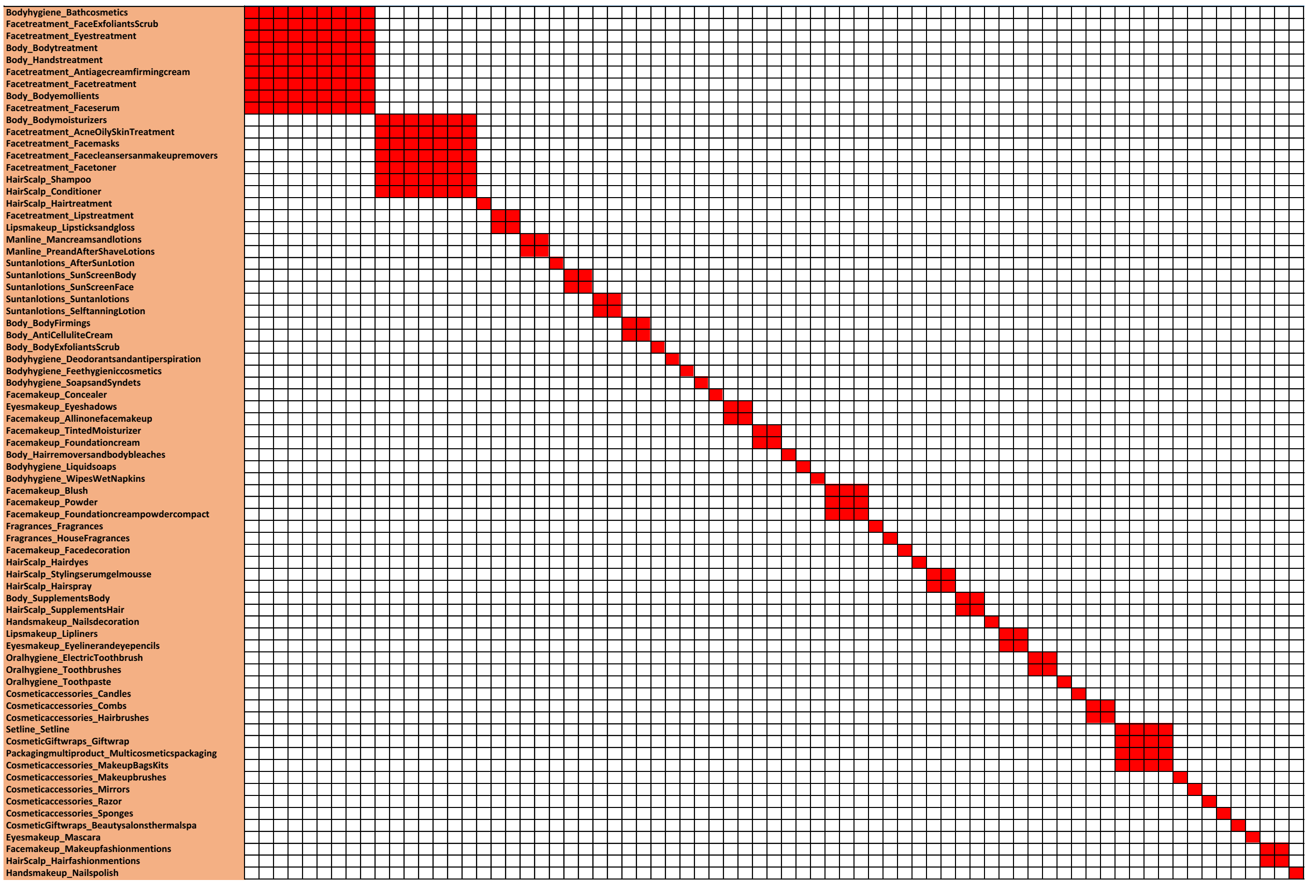

Bodyhygiene_Bathcosmetics
Facetreatment_FaceExfoliantsScrub
Facetreatment_Eyestreatment
Body_Bodytreatment
Body_Handstreatment
Facetreatment_Antiagecreamfirmingcream
Facetreatment_Facetreatment
Body_Bodyemollients
Facetreatment_Faceserum
Body_Bodymoisturizers
Facetreatment_AcneOilySkinTreatment
Facetreatment_Facemasks
Facetreatment_Facecleansersanmakeupremovers
Facetreatment_Facetoner
HairScalp_Shampoo
HairScalp_Conditioner
HairScalp_Hairtreatment
Facetreatment_Lipstreatment
Lipsmakeup_Lipsticksandgloss
Manline_Mancreamsandlotions
Manline_PreandAfterShaveLotions
Suntanlotions_AfterSunLotion
Suntanlotions_SunScreenBody
Suntanlotions_SunScreenFace
Suntanlotions_Suntanlotions
Suntanlotions_SelftanningLotion
Body_BodyFirmings
Body_AntiCelluliteCream
Body_BodyExfoliantsScrub
Bodyhygiene_Deodorantsandantiperspiration
Bodyhygiene_Feethygieniccosmetics
Bodyhygiene_SoapsandSyndets
Facemakeup_Concealer
Eyesmakeup_Eyeshadows
Facemakeup_Allinonefacemakeup
Facemakeup_TintedMoisturizer
Facemakeup_Foundationcream
Body_Hairremoversandbodybleaches
Bodyhygiene_Liquidsoaps
Bodyhygiene_WipesWetNapkins
Facemakeup_Blush
Facemakeup_Powder
Facemakeup_Foundationcreampowdercompact
Fragrances_Fragrances
Fragrances_HouseFragrances
Facemakeup_Facedecoration
HairScalp_Hairdyes
HairScalp_Stylingserumgelmousse
HairScalp_Hairspray
Body_SupplementsBody
HairScalp_SupplementsHair
Handsmakeup_Nailsdecoration
Lipsmakeup_Lipliners
Eyesmakeup_Eyelinerandeyepencils
Oralhygiene_ElectricToothbrush
Oralhygiene_Toothbrushes
Oralhygiene_Toothpaste
Cosmeticaccessories_Candles
Cosmeticaccessories_Combs
Cosmeticaccessories_Hairbrushes
Setline_Setline
CosmeticGiftwraps_Giftwrap
Packagingmultiproduct_Multicosmeticspackaging
Cosmeticaccessories_MakeupBagsKits
Cosmeticaccessories_Makeupbrushes
Cosmeticaccessories_Mirrors
Cosmeticaccessories_Razor
Cosmeticaccessories_Sponges
CosmeticGiftwraps_Beautysalonsthermalspa
Eyesmakeup_Mascara
Facemakeup_Makeupfashionmentions
HairScalp_Hairfashionmentions
Handsmakeup_Nailspolish

Figure 5.3: New categories for the Beauty dataset.

We will refer to this new organization of the images as the Grouped dataset. In order to implement the clusterization, we have to reorganize the folder structure of the Beauty dataset as showed in page 11. We will accomplish this with the following code:

Input:

```
#define folder paths for train, validation y test
import os, shutil

#folders names
original_dataset_dir = '/home/miguelyogur/datasets/Beauty/'
base_dir = '/home/miguelyogur/tfm/Beauty_Grouped_A'

#new groupings
G1 = [
    'Bodyhygiene_Bathcosmetics'
    ,'Facetreatment_FaceExfoliantsScrub'
    ,'Facetreatment_Eyestreatment'
    ,'Body_Bodytreatment'
    ,'Body_Handstreatment'
    ,'Facetreatment_Antiagecreamfirmingcream'
    ,'Facetreatment_Facetreatment'
    ,'Body_Bodyemollients'
    ,'Facetreatment_Faceserum'
]

G2 = [
    'Facetreatment_AcneOilySkinTreatment'
    ,'Facetreatment_Facemasks'
    ,'Facetreatment_Facecleansersanmakeupremovers'
    ,'Facetreatment_Facetoner'
    ,'HairScalp_Shampoo'
    ,'HairScalp_Conditioner'
]

G3 = [
    'Facetreatment_Lipstreatment'
    ,'Lipsmakeup_Lipsticksandgloss'
]

G4 = [
    'Manline_Mancreamsandlotions'
    ,'Manline_PreandAfterShaveLotions'
]

G5 = [
    'Suntanlotions_SunScreenBody'
    ,'Suntanlotions_SunScreenFace'
]

G6 = [
    'Suntanlotions_Suntanlotions'
    ,'Suntanlotions_SelftanningLotion'
]

G7 = [
    'Body_BodyFirmings'
    ,'Body_AntiCelluliteCream'
]
```

```
G8 = [
     'Eyesmakeup_Eyeshadows'
     ,'Facemakeup_Allinonefacemakeup'
]

G9 = [
     'Facemakeup_TintedMoisturizer'
     ,'Facemakeup_Foundationcream'
]

G10 = [
     'Facemakeup_Blush'
     ,'Facemakeup_Powder'
     ,'Facemakeup_Foundationcreampowdercompact'
]

G11 = [
     'HairScalp_Stylingserumgelmousse'
     ,'HairScalp_Hairspray'
]

G12 = [
     'Body_SupplementsBody'
     ,'HairScalp_SupplementsHair'
]

G13 = [
     'Lipsmakeup_Lipliners'
     ,'Eyesmakeup_Eyelinerandeyepencils'
]

G14 = [
     'Oralhygiene_ElectricToothbrush'
     ,'Oralhygiene_Toothbrushes'
]

G15 = [
     'Cosmeticaccessories_Combs'
     ,'Cosmeticaccessories_Hairbrushes'
]

G16 = [
     'Setline_Setline'
     ,'CosmeticGiftwraps_Giftwrap'
```

```
    ,'Packagingmultiproduct_Multicosmeticspackaging'
    ,'Cosmeticaccessories_MakeupBagsKits'
]

G17 = [
    'Facemakeup_Makeupfashionmentions'
    ,'HairScalp_Hairfashionmentions'
]

grouped_cat = [G1,
               G2,
               G3,
               G4,
               G5,
               G6,
               G7,
               G8,
               G9,
               G10,
               G11,
               G12,
               G13,
               G14,
               G15,
               G16,
               G17]
grouped_cat_names = ['G1',
               'G2',
               'G3',
               'G4',
               'G5',
               'G6',
               'G7',
               'G8',
               'G9',
               'G10',
               'G11',
               'G12',
               'G13',
               'G14',
               'G15',
               'G16',
               'G17']
```

```
#get list containing all paths for the categories of the Beauty dataset
subdirs = list()

for root, dirs, files in os.walk('/home/miguelyogur/datasets/Beauty'):
    subdirs.append(root)

subdirs = subdirs[1:]
#store categories' names in list

names = list({x.replace('/home/miguelyogur/datasets/Beauty/boxes_', '')[:-5]
    for x in subdirs})
In [4]:
#create new dataset folder
shutil.rmtree(base_dir)
os.mkdir(base_dir)
In [5]:
#create new groupings and copy the files

for i, group in enumerate(grouped_cat):
    dest_dir = os.path.join(base_dir, grouped_cat_names[i])
    os.mkdir(dest_dir)
    print('Grouping:', grouped_cat_names[i])

    for subdir in subdirs:
        name_subdir = subdir.replace(
            '/home/miguelyogur/datasets/Beauty/boxes_', '')[:-5]
        if name_subdir in group:
            print(name_subdir, 'contained,␣copying␣files')
            filenames = []
            for root, dirs, files in os.walk(subdir):
                filenames.append(files)
                filenames = filenames[0]
                for file in filenames:
                    src = os.path.join(subdir, file)
                    dst = os.path.join(dest_dir, file)
                    shutil.copyfile(src, dst)

#copy the remaining files for the ungrouped categories
categories_grouped = []
for group in grouped_cat:
    for name in group:
        categories_grouped.append(name)

categories_ungrouped = list(set(names) - set(categories_grouped))
```

```
for subdir in subdirs:
    name_subdir = subdir.replace('/home/miguelyogur/datasets/Beauty/boxes_', '')[:-5]
    if name_subdir in categories_ungrouped:
        dest_dir = os.path.join(base_dir, name_subdir)
        os.mkdir(dest_dir)
        print(name_subdir, 'not grouped, copying files')
        filenames = []
        for root, dirs, files in os.walk(subdir):
            filenames.append(files)
            filenames = filenames[0]
            for file in filenames:
                src = os.path.join(subdir, file)
                dst = os.path.join(dest_dir, file)
                shutil.copyfile(src, dst)
```

This script creates the `Beauty_Grouped_A` dataset. At this point, the same process for creating a new folder structure with train, validation and test folders as we did for the original dataset is performed. We can then apply the same approach for fitting and evaluating the model on this new, reorganized version of the Beauty dataset.

5.3 Training on the Grouped Dataset

The training process mimics the one employed for the full dataset. However, since we now have 42 categories instead of 73, and we are also using a different location for the data, we have to modify the header of the script used for training and evaluating. This new header will be as follows:

Input:

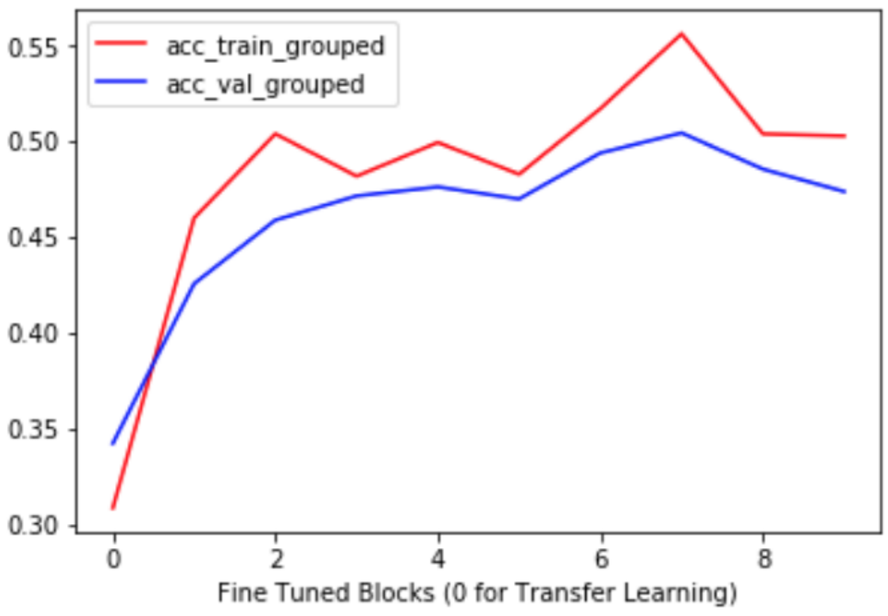
```
#     MODEL InceptionV3
#     WITH DATA AUGMENTATION
#     FINE TUNING

##############################
##############################
input_dim  = 299 #input dimension of the pretrained network.
                              #the input shape will be (input_dim, input_dim, 3)
class_dim  = 512 #number of nodes of the last Dense layer before the classificator
#l_unfreeze = 197 #unfreeze last 4 blocks of inceptionV3 and train
n_classes = 42 #number of classes
base_dir = '/home/miguelro1976/tfm/dataset_beauty_grouped_A'
```

############################
############################

We will save the best model before overfitting for each training round, where for each round a further inception block is fine tuned. The last block being fine tuned is the ninth one, exactly like it was the case for the full dataset. The training and evaluation process is summarized in Figure 5.4.

As we can see, the best accuracy obtained with the transfer learning and fine tuning approaches reaches a maximum of 50.45% with the last 7 inception blocks fine tuned before the model starts degrading. Again, the full results of the training and validating process are collected in Section 8.2.



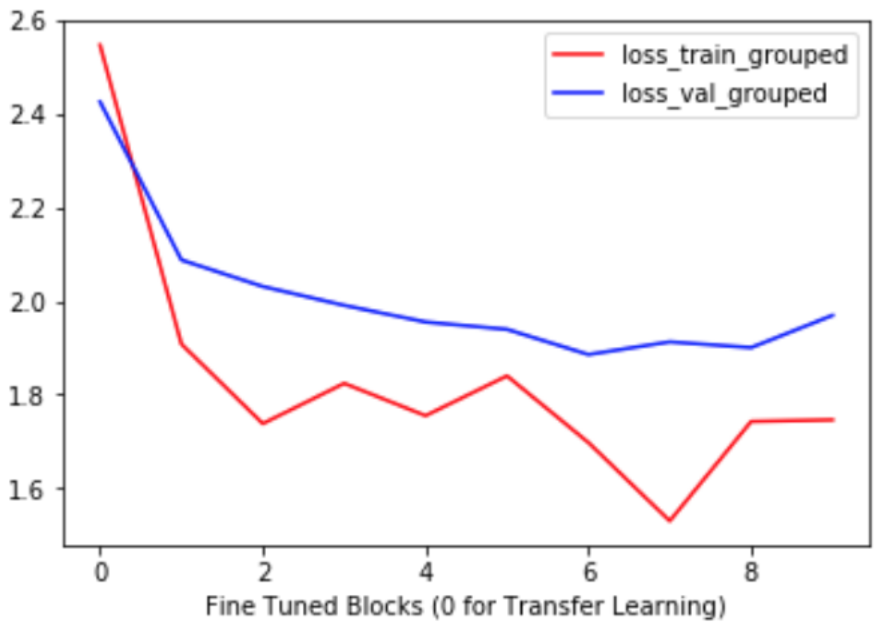


Figure 5.4: Quality of the different fine tuned models for the grouped Beauty dataset using the InceptionV3 network.

Chapter 6

Additional Models and Quality Comparison

6.1 Complete Training of other Network Architectures

In this section, and for the sake of comparison, we will fit several other models with standard state of the art deep network architectures. In these cases, we will not use the transfer learning approach, and we will train the networks from scratch with full random initialization of the parameters. The models considered will be

- InceptionV3, introduced in [13]
- ResNet50, introduced in [12]
- SimpleNet, introduced in [11]

For InceptionV3 and ResNet we will use the implementations included in keras. Notice that we will not use the weights pretrained on ImageNet and that we will include the top classifier layer. In the case of InceptionV3, the keras implementation of this network, although it adapts to the number of classes of any dataset, is slightly different from the original architecture, and therefore, does not correspond exactly to the top classifier block that we have used in the transfer learning and fine tuning processes. The only modification needed to our code is changing the `init_model` function in the following ways, depending on the architecture:

Input:

```
#complete training InceptionV3
def init_model():
    model = InceptionV3(weights=None,
                                include_top=True,
                                input_shape=(input_dim, input_dim, 3),
                                 classes = n_classes)
    return model

#complete training ResNet50
    def init_model():
    model = ResNet50(weights=None,
                                include_top=True,
                                input_shape=(input_dim, input_dim, 3),
                                classes = n_classes)
    return model
```

For the SimpleNet architecture, we will use the keras implementation developed in [15]. The fitting and quality assessment processes are identical to the ones carried out in previous sections of this work. We will only be fitting the models on the full Beauty dataset.

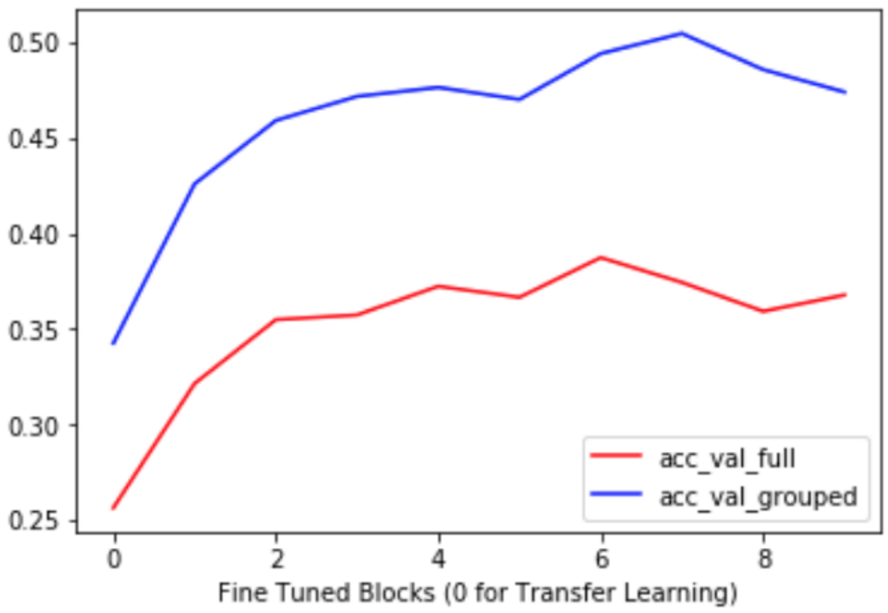
The accuracy scores on the validation set for each model are:

- InceptionV3 0.2846,
- ResNet50 0.1450,
- SimpleNet, 0.1504.

As we can see, a complete training of these networks, even with image augmentation, cannot compete with the results obtained by transfer learning and fine tuning techniques. However, the InceptionV3 network seems to work much better than the two other models. The reason is, most likely, that the synthetic images generated by image augmentation are not completely decorrelated from the original ones. Therefore, in the context of a high capacity model such as the deep networks considered in this section, image augmentation is not enough to avoid overfitting.

6.2 Quality Comparison for the Full and Grouped Beauty Datasets

We will now look at a comparison between the quality scores obtained on the full and grouped Beauty datasets with the methods of Chapters 4 and 5. If we compare the accuracy and loss on the validation set for each model and each step of the transfer learning and fine tuning approach, we find the behavior shown in Figure 6.1. Notice that for each step of the process, the accuracy on the grouped dataset is consistently about 10 percentage points higher. The



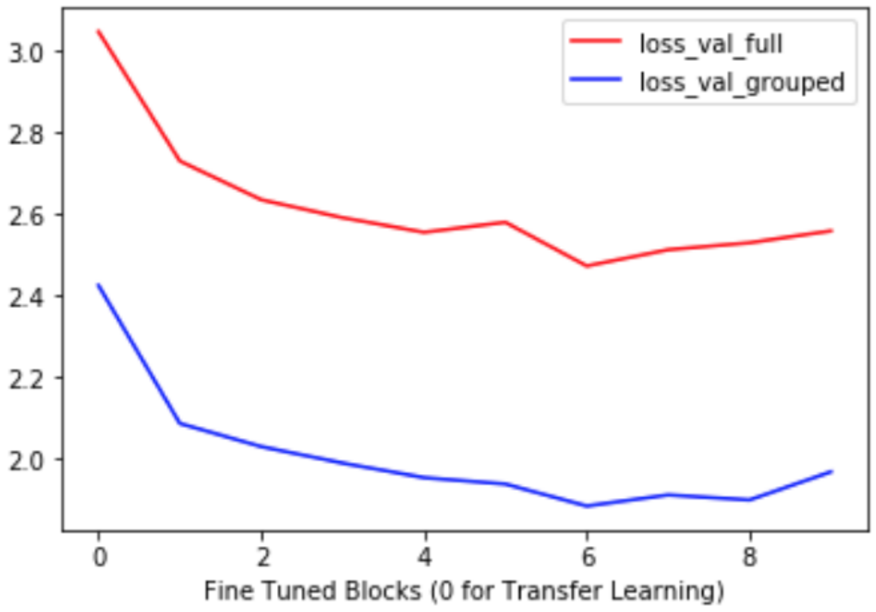


Figure 6.1: Comparison of the performance on the full and grouped datasets using the InceptionV3 network.

best model obtained correspond to fine tune 6 and 7 inception blocks respectively, and the performance on the grouped dataset, for this case, offers a 30 percent relative improvement with respect to the full dataset.

6.3 Test Accuracies

As explained in the previous section, for each of the two datasets the final model will be the one with best validation accuracy . Since these are actually different models on different datasets it makes sense to compute the final test accuracy for both cases. The way of doing this is a straightforward modification of the validation process used by the `get_quality_model`, consisting in changing the origin of the data for the `val_generator` from the validation set to the test set. Recall that, since the test set has not yet been used, this score can be considered a fair estimation of what the performance of each model would be on unseen data. The test accuracies obtained are:

- **0.3733** for the full dataset, and
- **0.5003** for the grouped dataset.

Chapter 7

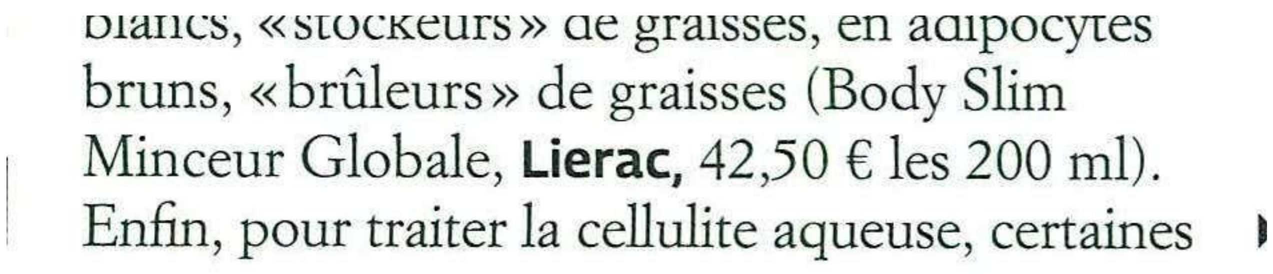
Conclusions

In this project we have used deep learning to solve an image recognition problem. Specifically we have applied transfer learning and fine tuning approaches with the InceptionV3 network pretrained on the ImageNet dataset in order to obtain predictive models for classifying a dataset with 73 categories. We have also trained full models with random weights initialization for several standard deep network architectures.

It has been shown that using the full training approach, InceptionV3 is the best performant model for our dataset. However, the accuracy obtained is not comparable to the one that is possible to achieve with the same architecture combining transfer learning and fine tuning strategies. Furthermore, our results are much improved with a reorganization of the categories of our dataset consisting in grouping together several classes obtaining a coarser categories set with 42 different classification labels.

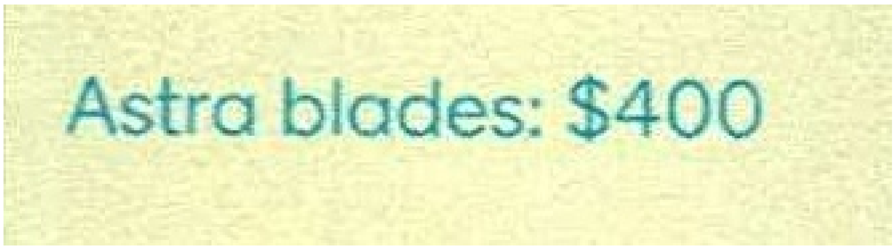
The best accuracy scores obtained within all the strategies used in this work are not stellar. Probably the main reason for this is the small number of images available for each class. But also, some of the images are very difficult to categorize, since many have been obtained from advertisements in printed media and may be visually ambiguous. See some examples of this problem below. These two facts have acted as a bottleneck preventing us from getting better accuracy results. The technique of image augmentation has helped to mitigate this fact to a certain extent, but it has not prevented the model from overfitting eventually.

As future research lines, it is expected that our results could be improved (probably marginally) by an in depth investigation of the hyperparameters used in the fitting process. Specifically, choosing among more optimizers and learning rates, or different batch sizes which would impact the stochastic nature of the optimization process could be plausible improving strategies. However, we do not expect these methods to drastically improve our results.



blancs, «stockeurs» de graisses, en adipocytes
bruns, «brûleurs» de graisses (Body Slim
Minceur Globale, **Lierac,** 42,50 € les 200 ml).
Enfin, pour traiter la cellulite aqueuse, certaines

Figure 7.1: Problematic image in the Body Firmings category.



Astra blades: $400

Figure 7.2: Problematic image in the Cosmetic Accessories. Razor category.

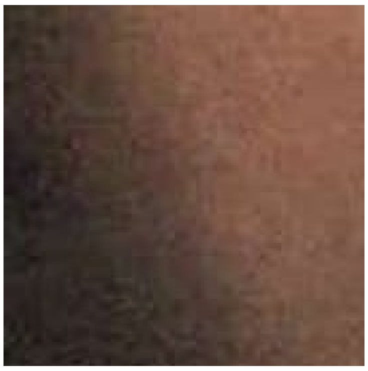


Figure 7.3: Problematic image in the Deodorant and Antiperspiration category.

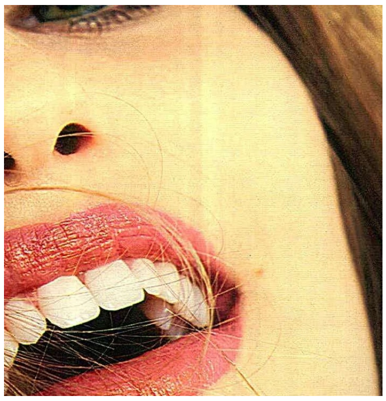


Figure 7.4: Problematic image in the Face Makeup. Powder category.

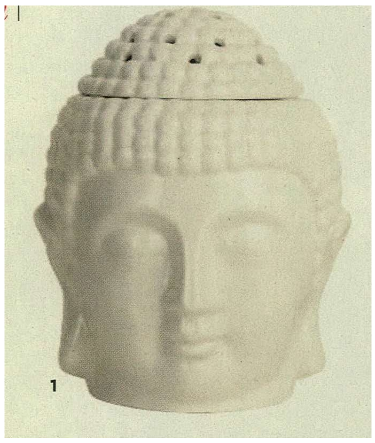


Figure 7.5: Problematic image in the Fragances. House Fragances category.

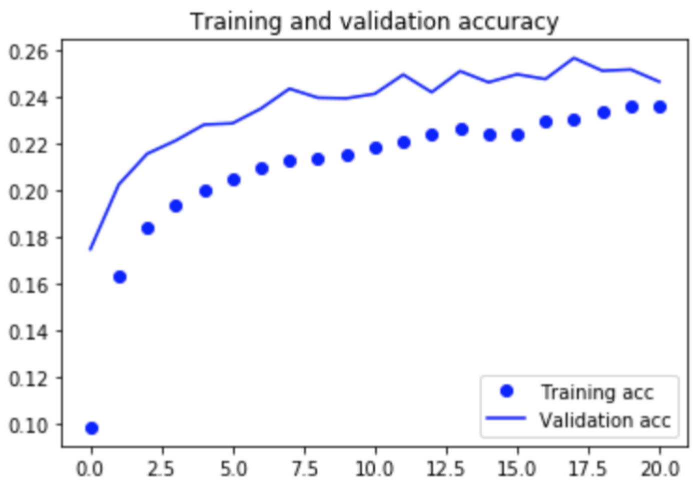
Chapter 8

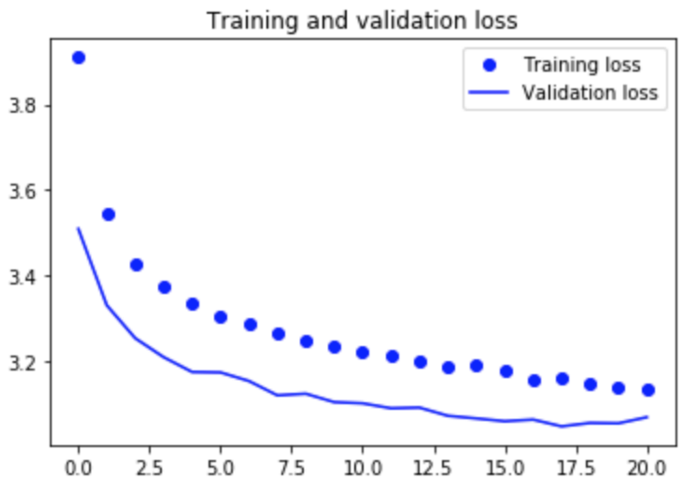
Appendix

In this chapter we collect, for archival purposes, the full output results of the different models fitted during this project. For each one there is a summary of each epoch, as well as training and loss visual comparisons for the training and validation datat.

8.1 Results for the Beauty Dataset with InceptionV3

8.1.1 Transfer Learning



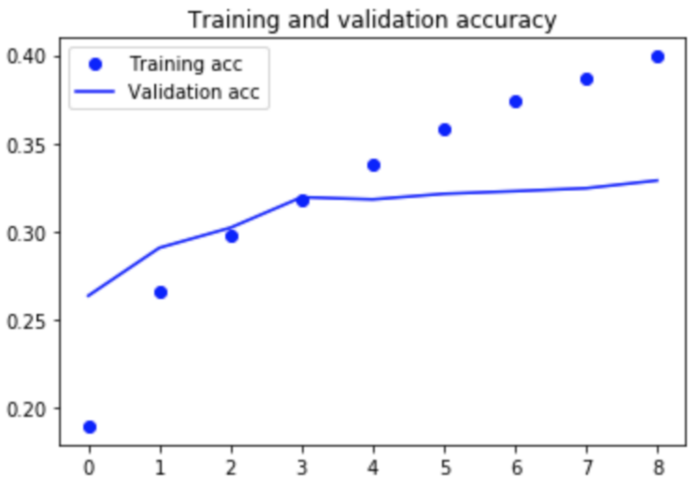


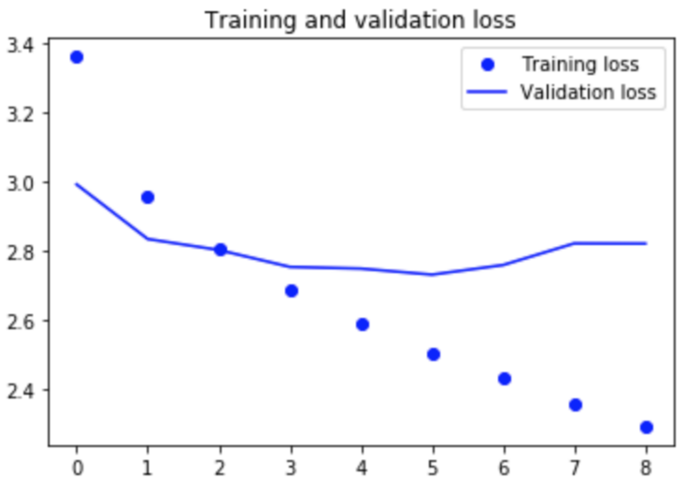
```
Epoch 1/100
911/911 [==============================] - 1586s 2s/step - loss: 3.9106 - acc: 0.0983 - val_loss: 3.5091 - val_acc: 0.1745
Epoch 2/100
911/911 [==============================] - 1442s 2s/step - loss: 3.5444 - acc: 0.1627 - val_loss: 3.3305 - val_acc: 0.2023
Epoch 3/100
911/911 [==============================] - 1450s 2s/step - loss: 3.4270 - acc: 0.1838 - val_loss: 3.2535 - val_acc: 0.2154
Epoch 4/100
911/911 [==============================] - 1441s 2s/step - loss: 3.3737 - acc: 0.1933 - val_loss: 3.2092 - val_acc: 0.2210
Epoch 5/100
911/911 [==============================] - 1457s 2s/step - loss: 3.3333 - acc: 0.1995 - val_loss: 3.1745 - val_acc: 0.2279
Epoch 6/100
911/911 [==============================] - 1437s 2s/step - loss: 3.3032 - acc: 0.2048 - val_loss: 3.1735 - val_acc: 0.2285
Epoch 7/100
911/911 [==============================] - 1447s 2s/step - loss: 3.2855 - acc: 0.2093 - val_loss: 3.1534 - val_acc: 0.2348
Epoch 8/100
```

```
911/911 [==============================] - 1451s 2s/step - loss: 3.2644 - acc: 0.2130 - val_loss: 3.1199 - val_acc: 0.2433
Epoch 9/100
911/911 [==============================] - 1442s 2s/step - loss: 3.2467 - acc: 0.2137 - val_loss: 3.1237 - val_acc: 0.2394
Epoch 10/100
911/911 [==============================] - 1458s 2s/step - loss: 3.2327 - acc: 0.2150 - val_loss: 3.1040 - val_acc: 0.2392
Epoch 11/100
911/911 [==============================] - 1444s 2s/step - loss: 3.2205 - acc: 0.2184 - val_loss: 3.1013 - val_acc: 0.2411
Epoch 12/100
911/911 [==============================] - 1442s 2s/step - loss: 3.2114 - acc: 0.2208 - val_loss: 3.0897 - val_acc: 0.2493
Epoch 13/100
911/911 [==============================] - 1456s 2s/step - loss: 3.2020 - acc: 0.2239 - val_loss: 3.0912 - val_acc: 0.2418
Epoch 14/100
911/911 [==============================] - 1448s 2s/step - loss: 3.1878 - acc: 0.2265 - val_loss: 3.0721 - val_acc: 0.2508
Epoch 15/100
911/911 [==============================] - 1452s 2s/step - loss: 3.1901 - acc: 0.2240 - val_loss: 3.0659 - val_acc: 0.2460
Epoch 16/100
911/911 [==============================] - 1442s 2s/step - loss: 3.1788 - acc: 0.2240 - val_loss: 3.0594 - val_acc: 0.2495
Epoch 17/100
911/911 [==============================] - 1446s 2s/step - loss: 3.1577 - acc: 0.2297 - val_loss: 3.0632 - val_acc: 0.2474
Epoch 18/100
911/911 [==============================] - 1438s 2s/step - loss: 3.1600 - acc: 0.2301 - val_loss: 3.0474 - val_acc: 0.2564
Epoch 19/100
911/911 [==============================] - 1435s 2s/step - loss: 3.1482 - acc: 0.2333 - val_loss: 3.0554 - val_acc: 0.2510
Epoch 20/100
911/911 [==============================] - 1440s 2s/step - loss: 3.1399 - acc: 0.2360 - val_loss: 3.0545 - val_acc: 0.2515
Epoch 21/100
911/911 [==============================] - 1432s 2s/step - loss: 3.1353 - acc: 0.2356 - val_loss: 3.0687 - val_acc: 0.2463
last model validation_loss: 3.0693485325960896 last model validation_acc: 0.2472557627343451


MODEL RESULTS:
Validation accuracy:  0.25641377418027167
```

8.1.2 Fine Tuning 1 block



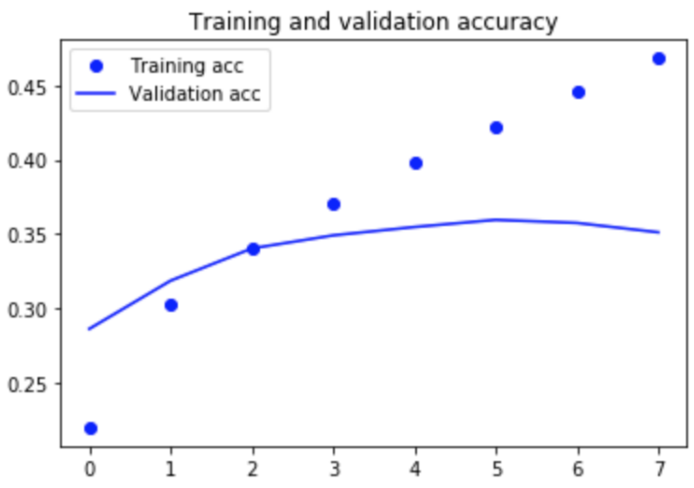


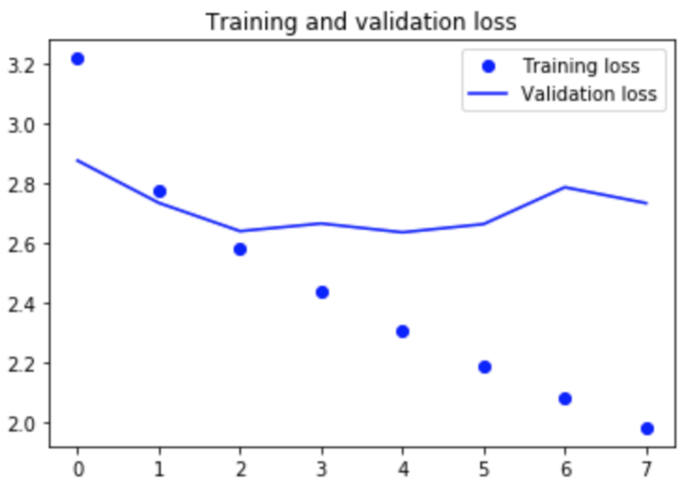
```
Epoch 1/100
911/911 [==============================] - 1606s 2s/step - loss: 3.3622 - acc: 0.1898 - val_loss: 2.9920 - val_acc: 0.2636
Epoch 2/100
911/911 [==============================] - 1455s 2s/step - loss: 2.9578 - acc: 0.2660 - val_loss: 2.8339 - val_acc: 0.2908
Epoch 3/100
911/911 [==============================] - 1457s 2s/step - loss: 2.8058 - acc: 0.2981 - val_loss: 2.8021 - val_acc: 0.3021
Epoch 4/100
911/911 [==============================] - 1468s 2s/step - loss: 2.6865 - acc: 0.3179 - val_loss: 2.7531 - val_acc: 0.3194
Epoch 5/100
911/911 [==============================] - 1463s 2s/step - loss: 2.5866 - acc: 0.3377 - val_loss: 2.7482 - val_acc: 0.3182
Epoch 6/100
911/911 [==============================] - 1455s 2s/step - loss: 2.5044 - acc: 0.3584 - val_loss: 2.7306 - val_acc: 0.3214
Epoch 7/100
911/911 [==============================] - 1462s 2s/step - loss: 2.4324 - acc: 0.3744 - val_loss: 2.7585 - val_acc: 0.3229
Epoch 8/100
911/911 [==============================] - 1452s 2s/step - loss: 2.3579 - acc: 0.3872 - val_loss: 2.8211 - val_acc: 0.3245
Epoch 9/100
911/911 [==============================] - 1453s 2s/step - loss: 2.2915 - acc: 0.3993 - val_loss: 2.8206 - val_acc: 0.3289
```

```
last model validation_loss: 2.8335836886312777 last model validation_acc: 0.32656421511547495


MODEL RESULTS:
Validation accuracy:  0.3213060776512553
```

8.1.3 Fine Tuning 2 blocks

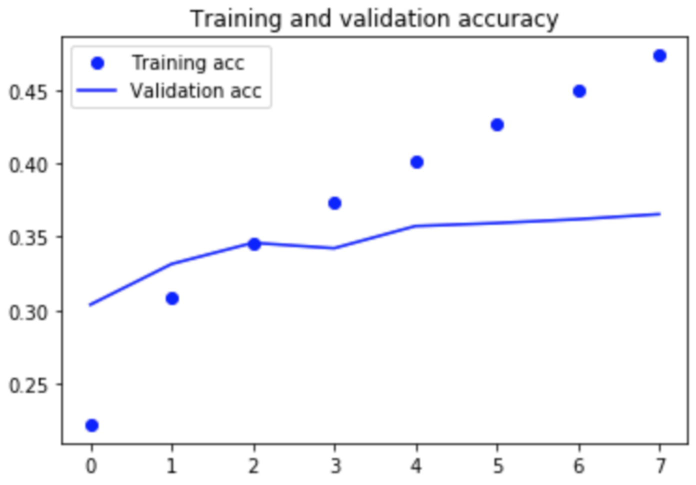


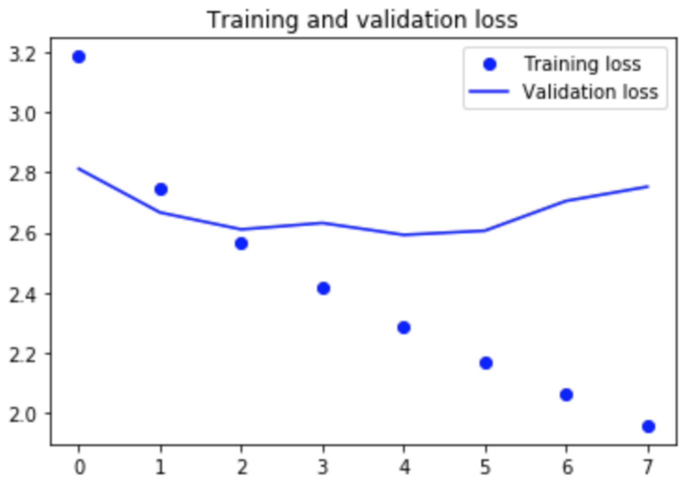


```
Epoch 1/100
911/911 [==============================] - 1532s 2s/step - loss: 3.2165 - acc: 0.2201 - val_loss: 2.8753 - val_acc: 0.2864
Epoch 2/100
911/911 [==============================] - 1476s 2s/step - loss: 2.7748 - acc: 0.3024 - val_loss: 2.7342 - val_acc: 0.3187
Epoch 3/100
911/911 [==============================] - 1479s 2s/step - loss: 2.5797 - acc: 0.3410 - val_loss: 2.6398 - val_acc: 0.3404
Epoch 4/100
911/911 [==============================] - 1479s 2s/step - loss: 2.4376 - acc: 0.3705 - val_loss: 2.6654 - val_acc: 0.3492
Epoch 5/100
911/911 [==============================] - 1474s 2s/step - loss: 2.3079 - acc: 0.3977 - val_loss: 2.6358 - val_acc: 0.3548
Epoch 6/100
911/911 [==============================] - 1473s 2s/step - loss: 2.1918 - acc: 0.4218 - val_loss: 2.6634 - val_acc: 0.3596
Epoch 7/100
911/911 [==============================] - 1472s 2s/step - loss: 2.0845 - acc: 0.4460 - val_loss: 2.7863 - val_acc: 0.3576
Epoch 8/100
911/911 [==============================] - 1474s 2s/step - loss: 1.9841 - acc: 0.4683 - val_loss: 2.7334 - val_acc: 0.3513
last model validation_loss: 2.7249390843409214 last model validation_acc: 0.3507135016138284
```

```
MODEL RESULTS:
Validation accuracy:  0.35491837014679656
```

8.1.4 Fine Tuning 3 blocks

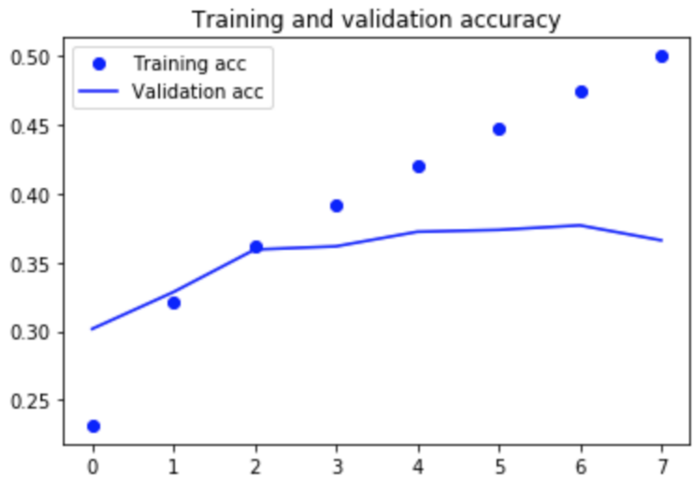


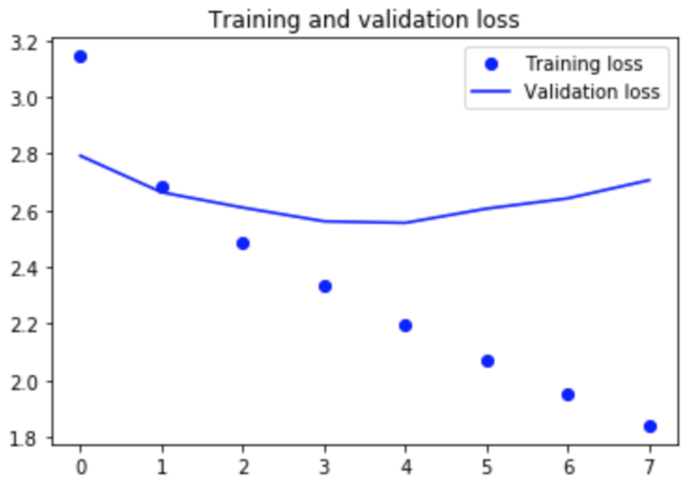


```
Epoch 1/100
911/911 [==============================] - 1541s 2s/step - loss: 3.1863 - acc: 0.2222 - val_loss: 2.8119 - val_acc: 0.3038
Epoch 2/100
911/911 [==============================] - 1482s 2s/step - loss: 2.7458 - acc: 0.3087 - val_loss: 2.6665 - val_acc: 0.3315
Epoch 3/100
911/911 [==============================] - 1483s 2s/step - loss: 2.5630 - acc: 0.3449 - val_loss: 2.6101 - val_acc: 0.3459
Epoch 4/100
911/911 [==============================] - 1487s 2s/step - loss: 2.4195 - acc: 0.3738 - val_loss: 2.6314 - val_acc: 0.3422
Epoch 5/100
911/911 [==============================] - 1468s 2s/step - loss: 2.2847 - acc: 0.4010 - val_loss: 2.5918 - val_acc: 0.3573
Epoch 6/100
911/911 [==============================] - 1484s 2s/step - loss: 2.1687 - acc: 0.4264 - val_loss: 2.6057 - val_acc: 0.3594
Epoch 7/100
911/911 [==============================] - 1472s 2s/step - loss: 2.0628 - acc: 0.4498 - val_loss: 2.7049 - val_acc: 0.3620
Epoch 8/100
911/911 [==============================] - 1470s 2s/step - loss: 1.9572 - acc: 0.4737 - val_loss: 2.7521 - val_acc: 0.3654
last model validation_loss: 2.7343021470288678 last model validation_acc: 0.3669045007778444
```

```
MODEL RESULTS:
Validation accuracy:  0.35725065166689535
```

8.1.5 Fine Tuning 4 blocks

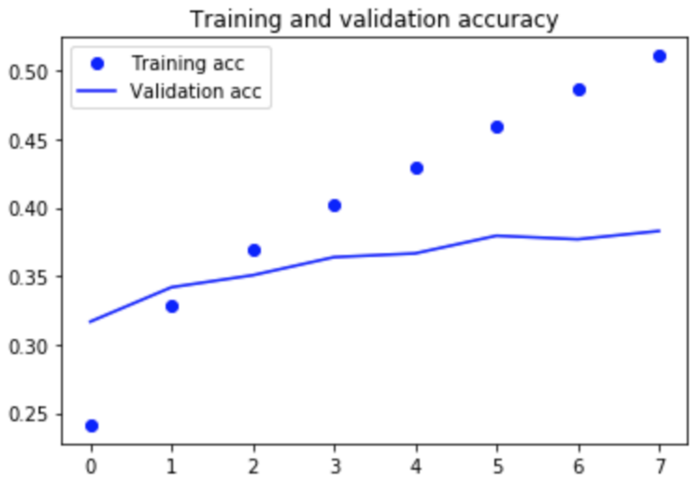


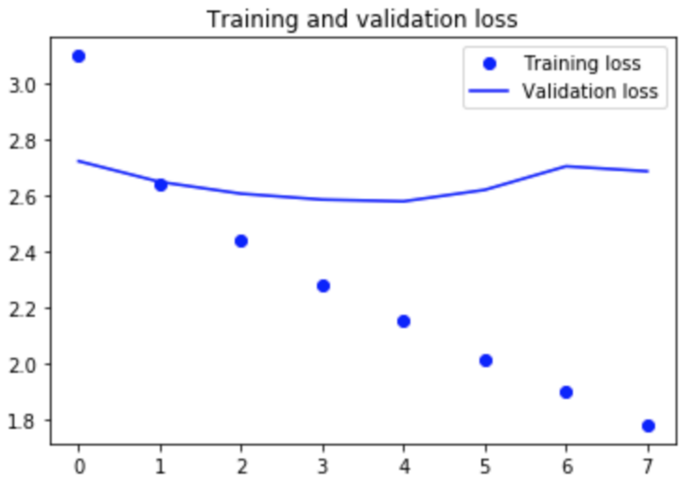


```
Epoch 1/100
911/911 [==============================] - 1541s 2s/step - loss: 3.1444 - acc: 0.2318 - val_loss: 2.7931 - val_acc: 0.3017
Epoch 2/100
911/911 [==============================] - 1505s 2s/step - loss: 2.6824 - acc: 0.3211 - val_loss: 2.6640 - val_acc: 0.3285
Epoch 3/100
911/911 [==============================] - 1507s 2s/step - loss: 2.4853 - acc: 0.3623 - val_loss: 2.6100 - val_acc: 0.3594
Epoch 4/100
911/911 [==============================] - 1510s 2s/step - loss: 2.3336 - acc: 0.3914 - val_loss: 2.5617 - val_acc: 0.3617
Epoch 5/100
911/911 [==============================] - 1501s 2s/step - loss: 2.1955 - acc: 0.4204 - val_loss: 2.5561 - val_acc: 0.3723
Epoch 6/100
911/911 [==============================] - 1500s 2s/step - loss: 2.0745 - acc: 0.4476 - val_loss: 2.6065 - val_acc: 0.3736
Epoch 7/100
911/911 [==============================] - 1507s 2s/step - loss: 1.9558 - acc: 0.4746 - val_loss: 2.6427 - val_acc: 0.3769
Epoch 8/100
911/911 [==============================] - 1504s 2s/step - loss: 1.8412 - acc: 0.4999 - val_loss: 2.7069 - val_acc: 0.3661
last model validation_loss: 2.685029826755712 last model validation_acc: 0.3722557630287698
```

```
MODEL RESULTS:
Validation accuracy:  0.3722046920016463
```

8.1.6 Fine Tuning 5 blocks

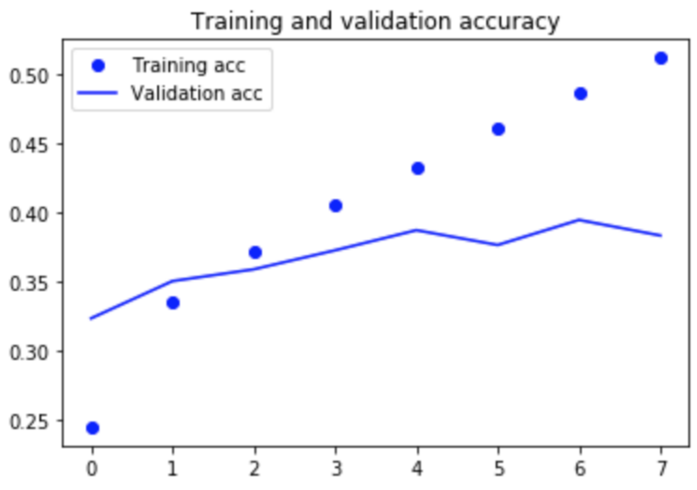


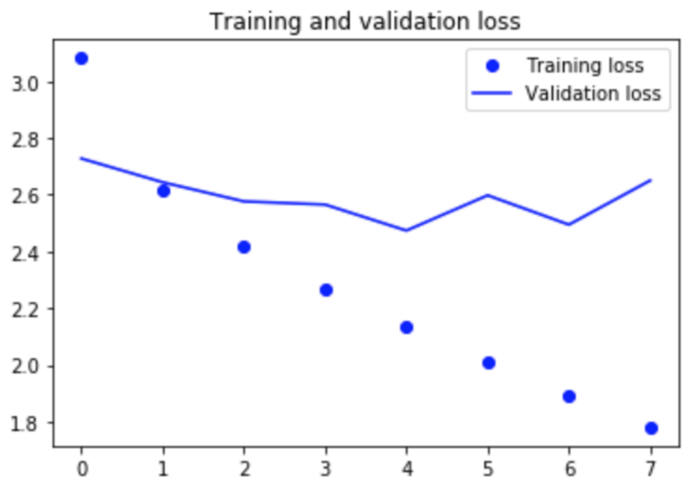


```
Epoch 1/100
911/911 [==============================] - 1586s 2s/step - loss: 3.1000 - acc: 0.2414 - val_loss: 2.7241 - val_acc: 0.3167
Epoch 2/100
911/911 [==============================] - 1531s 2s/step - loss: 2.6439 - acc: 0.3283 - val_loss: 2.6501 - val_acc: 0.3418
Epoch 3/100
911/911 [==============================] - 1531s 2s/step - loss: 2.4395 - acc: 0.3692 - val_loss: 2.6080 - val_acc: 0.3506
Epoch 4/100
911/911 [==============================] - 1521s 2s/step - loss: 2.2824 - acc: 0.4018 - val_loss: 2.5869 - val_acc: 0.3637
Epoch 5/100
911/911 [==============================] - 1517s 2s/step - loss: 2.1541 - acc: 0.4291 - val_loss: 2.5806 - val_acc: 0.3665
Epoch 6/100
911/911 [==============================] - 1516s 2s/step - loss: 2.0186 - acc: 0.4598 - val_loss: 2.6216 - val_acc: 0.3794
Epoch 7/100
911/911 [==============================] - 1507s 2s/step - loss: 1.8998 - acc: 0.4863 - val_loss: 2.7057 - val_acc: 0.3768
Epoch 8/100
911/911 [==============================] - 1510s 2s/step - loss: 1.7833 - acc: 0.5107 - val_loss: 2.6880 - val_acc: 0.3828
last model validation_loss: 2.6809152708880597 last model validation_acc: 0.3866630076184362
```

```
MODEL RESULTS:
Validation accuracy:  0.36657977774729045
```

8.1.7 Fine Tuning 6 blocks

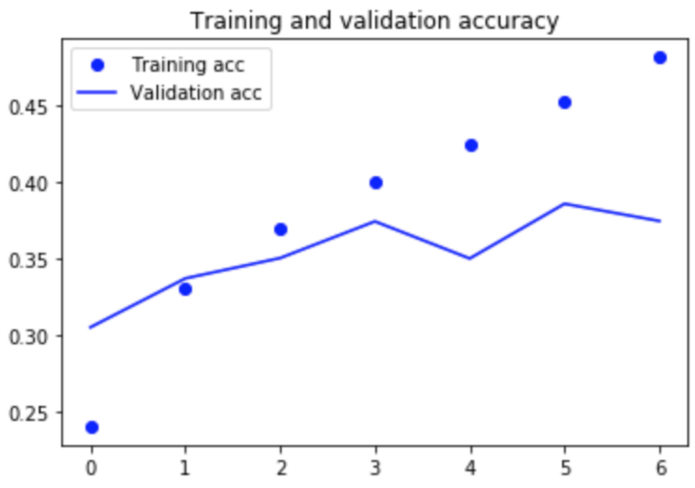


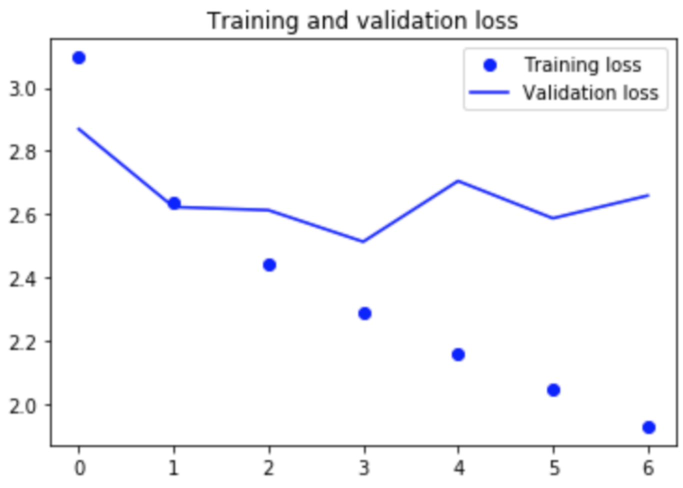


```
Epoch 1/100
911/911 [==============================] - 1662s 2s/step - loss: 3.0822 - acc: 0.2452 - val_loss: 2.7276 - val_acc: 0.3237
Epoch 2/100
911/911 [==============================] - 1541s 2s/step - loss: 2.6179 - acc: 0.3359 - val_loss: 2.6440 - val_acc: 0.3506
Epoch 3/100
911/911 [==============================] - 1534s 2s/step - loss: 2.4164 - acc: 0.3726 - val_loss: 2.5765 - val_acc: 0.3591
Epoch 4/100
911/911 [==============================] - 1562s 2s/step - loss: 2.2658 - acc: 0.4051 - val_loss: 2.5652 - val_acc: 0.3728
Epoch 5/100
911/911 [==============================] - 1547s 2s/step - loss: 2.1343 - acc: 0.4324 - val_loss: 2.4736 - val_acc: 0.3873
Epoch 6/100
911/911 [==============================] - 1536s 2s/step - loss: 2.0101 - acc: 0.4611 - val_loss: 2.5979 - val_acc: 0.3766
Epoch 7/100
911/911 [==============================] - 1519s 2s/step - loss: 1.8918 - acc: 0.4873 - val_loss: 2.4945 - val_acc: 0.3948
Epoch 8/100
911/911 [==============================] - 1528s 2s/step - loss: 1.7803 - acc: 0.5120 - val_loss: 2.6501 - val_acc: 0.3835
last model validation_loss: 2.649222436773005 last model validation_acc: 0.38570252473084776
```

```
MODEL RESULTS:
Validation accuracy:  0.38729592536699137
```

8.1.8 Fine Tuning 7 blocks



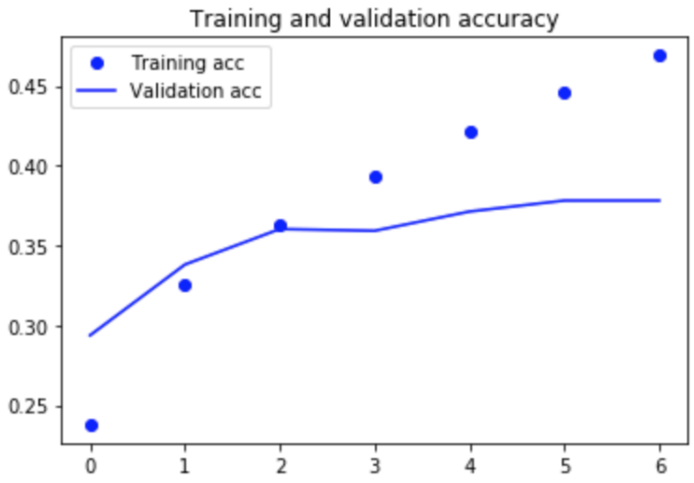


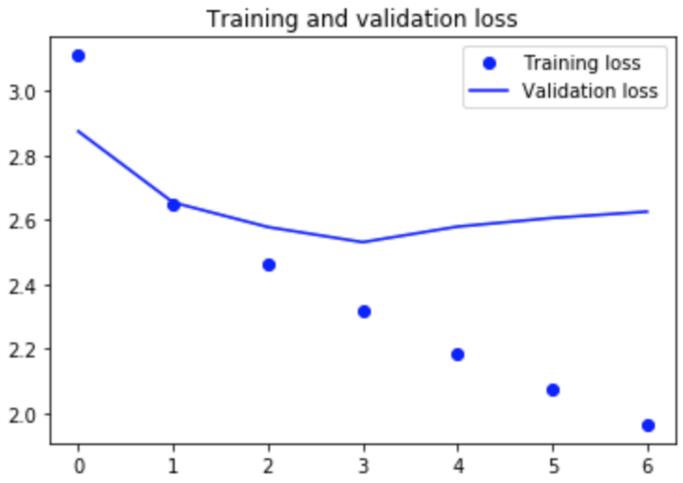
```
Epoch 1/100
911/911 [==============================] - 1681s 2s/step - loss: 3.0966 - acc: 0.2402 - val_loss: 2.8696 - val_acc: 0.3050
Epoch 2/100
911/911 [==============================] - 1549s 2s/step - loss: 2.6363 - acc: 0.3302 - val_loss: 2.6230 - val_acc: 0.3370
Epoch 3/100
911/911 [==============================] - 1551s 2s/step - loss: 2.4421 - acc: 0.3694 - val_loss: 2.6131 - val_acc: 0.3503
Epoch 4/100
911/911 [==============================] - 1547s 2s/step - loss: 2.2895 - acc: 0.4005 - val_loss: 2.5133 - val_acc: 0.3743
Epoch 5/100
911/911 [==============================] - 1510s 2s/step - loss: 2.1599 - acc: 0.4250 - val_loss: 2.7053 - val_acc: 0.3500
Epoch 6/100
911/911 [==============================] - 1513s 2s/step - loss: 2.0449 - acc: 0.4524 - val_loss: 2.5873 - val_acc: 0.3858
Epoch 7/100
911/911 [==============================] - 1531s 2s/step - loss: 1.9294 - acc: 0.4818 - val_loss: 2.6591 - val_acc: 0.3746
last model validation_loss: 2.6565333894003107 last model validation_acc: 0.374176728934802
```

```
MODEL RESULTS:
```

```
Validation accuracy:  0.374262587460557
```

8.1.9 Fine Tuning 8 blocks



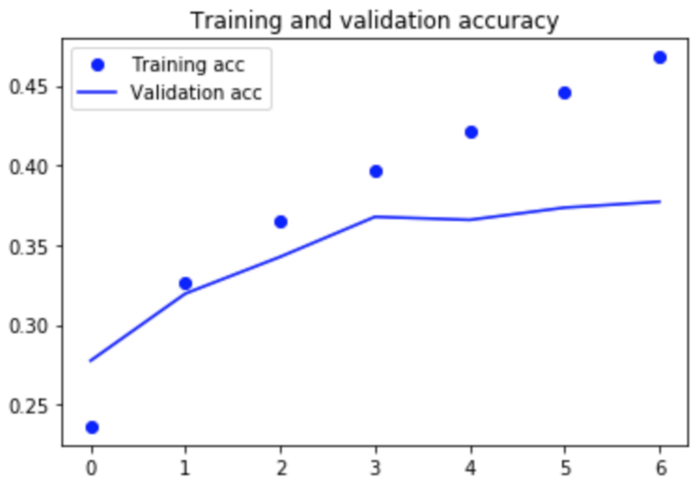


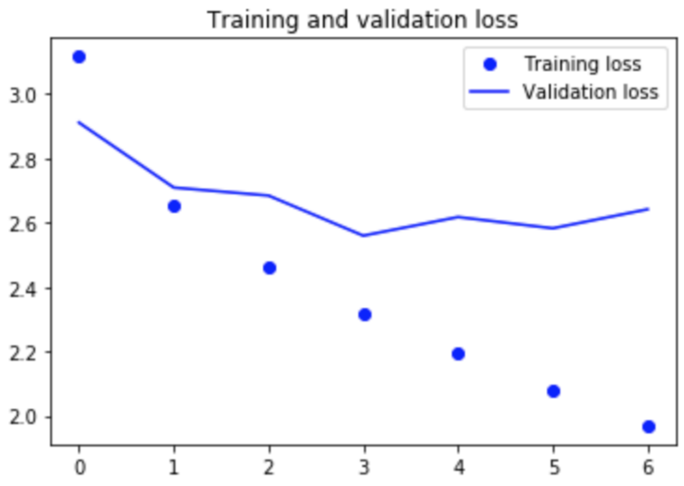
```
Epoch 1/100
911/911 [==============================] - 1703s 2s/step - loss: 3.1090 - acc: 0.2379 - val_loss: 2.8737 - val_acc: 0.2938
Epoch 2/100
911/911 [==============================] - 1554s 2s/step - loss: 2.6506 - acc: 0.3259 - val_loss: 2.6534 - val_acc: 0.3381
Epoch 3/100
911/911 [==============================] - 1556s 2s/step - loss: 2.4628 - acc: 0.3633 - val_loss: 2.5774 - val_acc: 0.3603
Epoch 4/100
911/911 [==============================] - 1555s 2s/step - loss: 2.3165 - acc: 0.3932 - val_loss: 2.5305 - val_acc: 0.3592
Epoch 5/100
911/911 [==============================] - 1550s 2s/step - loss: 2.1871 - acc: 0.4214 - val_loss: 2.5787 - val_acc: 0.3713
Epoch 6/100
911/911 [==============================] - 1542s 2s/step - loss: 2.0735 - acc: 0.4456 - val_loss: 2.6055 - val_acc: 0.3782
Epoch 7/100
911/911 [==============================] - 1555s 2s/step - loss: 1.9660 - acc: 0.4691 - val_loss: 2.6250 - val_acc: 0.3782
last model validation_loss: 2.653221637304999 last model validation_acc: 0.37321624598178593


MODEL RESULTS:
```

```
Validation accuracy:  0.359308547125806
```

8.1.10 Fine Tuning 9 blocks



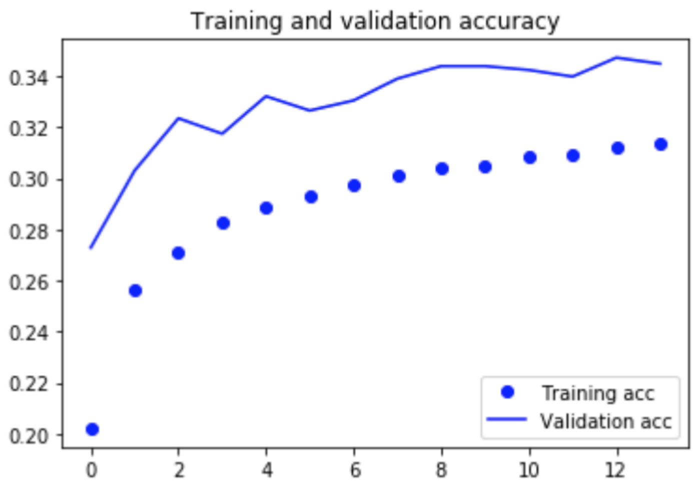


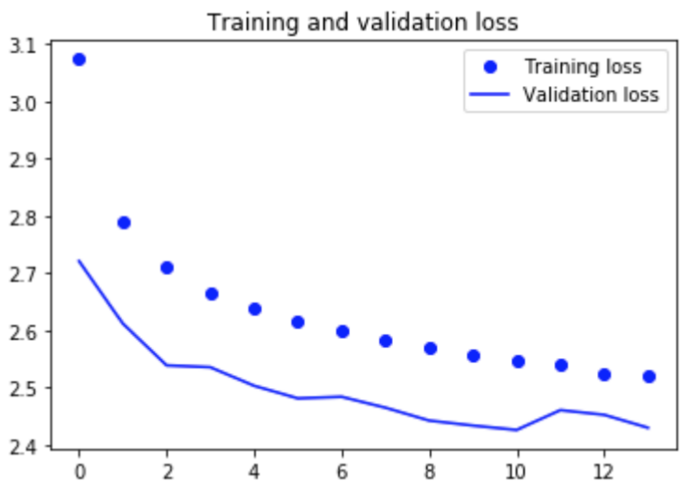
```
Epoch 1/100
911/911 [==============================] - 1693s 2s/step - loss: 3.1164 - acc: 0.2362 - val_loss: 2.9107 - val_acc: 0.2774
Epoch 2/100
911/911 [==============================] - 1572s 2s/step - loss: 2.6539 - acc: 0.3262 - val_loss: 2.7090 - val_acc: 0.3196
Epoch 3/100
911/911 [==============================] - 1572s 2s/step - loss: 2.4627 - acc: 0.3655 - val_loss: 2.6841 - val_acc: 0.3426
Epoch 4/100
911/911 [==============================] - 1568s 2s/step - loss: 2.3156 - acc: 0.3972 - val_loss: 2.5595 - val_acc: 0.3677
Epoch 5/100
911/911 [==============================] - 1560s 2s/step - loss: 2.1958 - acc: 0.4214 - val_loss: 2.6174 - val_acc: 0.3658
Epoch 6/100
911/911 [==============================] - 1561s 2s/step - loss: 2.0800 - acc: 0.4465 - val_loss: 2.5824 - val_acc: 0.3735
Epoch 7/100
911/911 [==============================] - 1556s 2s/step - loss: 1.9694 - acc: 0.4680 - val_loss: 2.6414 - val_acc: 0.3772
last model validation_loss: 2.6295608325271522 last model validation_acc: 0.3840559822733133

MODEL RESULTS:
Validation accuracy:  0.3676773219920428
```

8.2 Results for the Grouped Beauty Dataset with InceptionV3

8.2.1 Transfer Learning



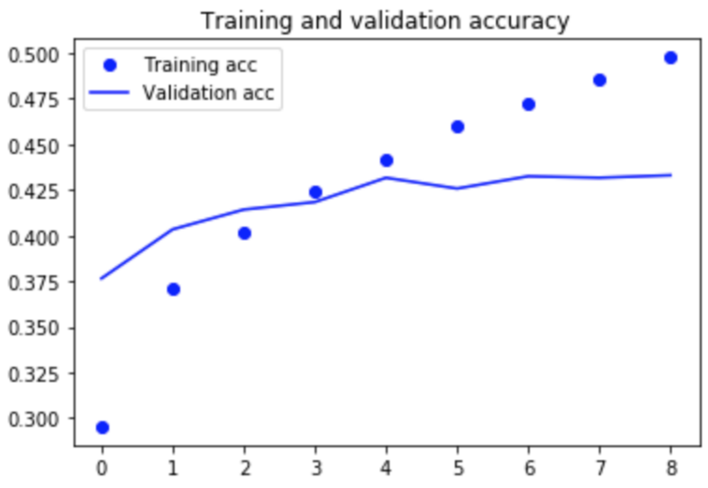


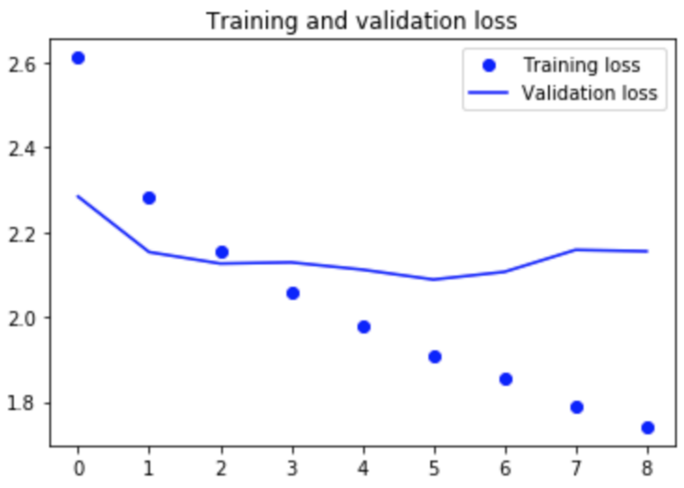
```
Epoch 1/100
911/911 [==============================] - 1454s 2s/step - loss: 3.0743 - acc: 0.2022 - val_loss: 2.7209 - val_acc: 0.2729
Epoch 2/100
911/911 [==============================] - 1447s 2s/step - loss: 2.7890 - acc: 0.2568 - val_loss: 2.6118 - val_acc: 0.3031
Epoch 3/100
911/911 [==============================] - 1471s 2s/step - loss: 2.7095 - acc: 0.2714 - val_loss: 2.5388 - val_acc: 0.3235
Epoch 4/100
911/911 [==============================] - 1481s 2s/step - loss: 2.6660 - acc: 0.2826 - val_loss: 2.5356 - val_acc: 0.3175
Epoch 5/100
911/911 [==============================] - 1470s 2s/step - loss: 2.6391 - acc: 0.2888 - val_loss: 2.5031 - val_acc: 0.3322
Epoch 6/100
911/911 [==============================] - 1467s 2s/step - loss: 2.6158 - acc: 0.2935 - val_loss: 2.4811 - val_acc: 0.3266
Epoch 7/100
```

```
911/911 [==============================] - 1464s 2s/step - loss: 2.5980 - acc: 0.2974 - val_loss: 2.4839 - val_acc: 0.3305
Epoch 8/100
911/911 [==============================] - 1474s 2s/step - loss: 2.5823 - acc: 0.3013 - val_loss: 2.4649 - val_acc: 0.3391
Epoch 9/100
911/911 [==============================] - 1468s 2s/step - loss: 2.5708 - acc: 0.3040 - val_loss: 2.4423 - val_acc: 0.3440
Epoch 10/100
911/911 [==============================] - 1467s 2s/step - loss: 2.5567 - acc: 0.3050 - val_loss: 2.4335 - val_acc: 0.3440
Epoch 11/100
911/911 [==============================] - 1487s 2s/step - loss: 2.5472 - acc: 0.3088 - val_loss: 2.4259 - val_acc: 0.3425
Epoch 12/100
911/911 [==============================] - 1466s 2s/step - loss: 2.5391 - acc: 0.3095 - val_loss: 2.4606 - val_acc: 0.3399
Epoch 13/100
911/911 [==============================] - 1440s 2s/step - loss: 2.5256 - acc: 0.3124 - val_loss: 2.4523 - val_acc: 0.3473
Epoch 14/100
911/911 [==============================] - 1428s 2s/step - loss: 2.5210 - acc: 0.3140 - val_loss: 2.4297 - val_acc: 0.3450
last model validation_loss: 2.41889938041486 last model validation_acc: 0.34824368802566824

MODEL RESULTS:
Validation accuracy:  0.34243380436273835
```

8.2.2 Fine Tuning 1 block



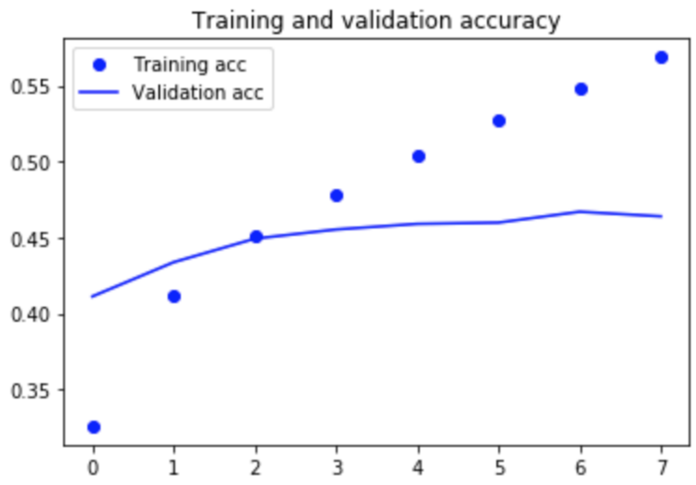


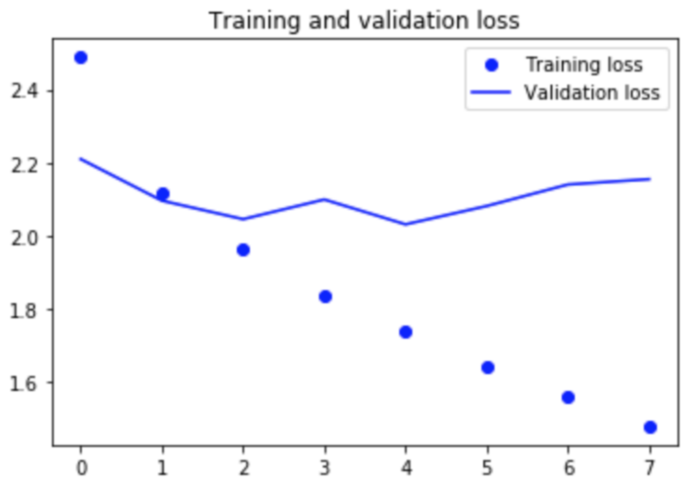
```
Epoch 1/100
911/911 [==============================] - 1557s 2s/step - loss: 2.6113 - acc: 0.2954 - val_loss: 2.2836 - val_acc: 0.3765
Epoch 2/100
911/911 [==============================] - 1435s 2s/step - loss: 2.2800 - acc: 0.3712 - val_loss: 2.1527 - val_acc: 0.4034
Epoch 3/100
911/911 [==============================] - 1446s 2s/step - loss: 2.1529 - acc: 0.4015 - val_loss: 2.1256 - val_acc: 0.4142
Epoch 4/100
911/911 [==============================] - 1431s 2s/step - loss: 2.0573 - acc: 0.4244 - val_loss: 2.1284 - val_acc: 0.4184
Epoch 5/100
911/911 [==============================] - 1432s 2s/step - loss: 1.9774 - acc: 0.4418 - val_loss: 2.1113 - val_acc: 0.4317
Epoch 6/100
911/911 [==============================] - 1435s 2s/step - loss: 1.9082 - acc: 0.4601 - val_loss: 2.0879 - val_acc: 0.4258
Epoch 7/100
911/911 [==============================] - 1428s 2s/step - loss: 1.8534 - acc: 0.4719 - val_loss: 2.1060 - val_acc: 0.4325
Epoch 8/100
911/911 [==============================] - 1435s 2s/step - loss: 1.7884 - acc: 0.4852 - val_loss: 2.1578 - val_acc: 0.4317
Epoch 9/100
911/911 [==============================] - 1432s 2s/step - loss: 1.7411 - acc: 0.4977 - val_loss: 2.1545 - val_acc: 0.4330
```

```
last model validation_loss: 2.147867797627538 last model validation_acc: 0.43139407205529323


MODEL RESULTS:
Validation accuracy:  0.4258471669639182
```

8.2.3 Fine Tuning 2 blocks

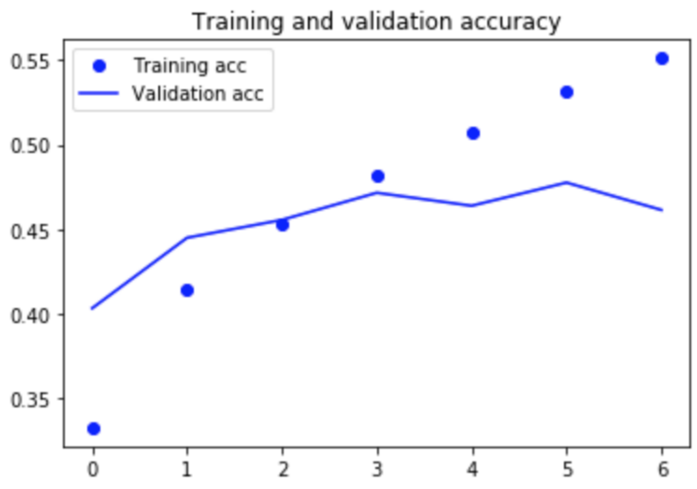


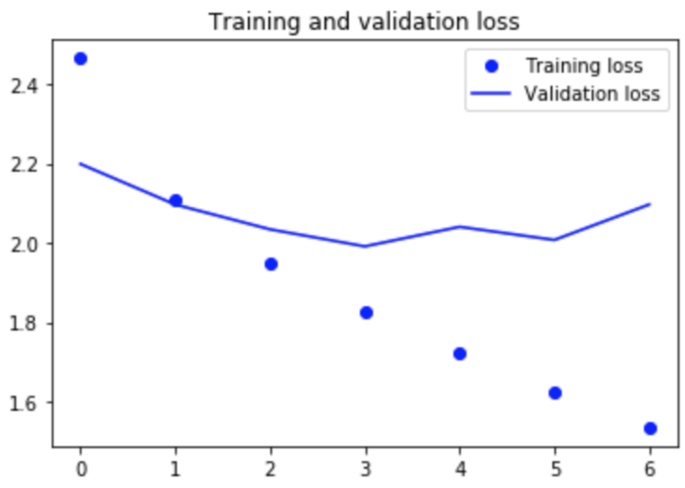


```
Epoch 1/100
911/911 [==============================] - 1499s 2s/step - loss: 2.4884 - acc: 0.3259 - val_loss: 2.2096 - val_acc: 0.4112
Epoch 2/100
911/911 [==============================] - 1449s 2s/step - loss: 2.1177 - acc: 0.4115 - val_loss: 2.0960 - val_acc: 0.4337
Epoch 3/100
911/911 [==============================] - 1474s 2s/step - loss: 1.9617 - acc: 0.4509 - val_loss: 2.0454 - val_acc: 0.4492
Epoch 4/100
911/911 [==============================] - 1470s 2s/step - loss: 1.8365 - acc: 0.4785 - val_loss: 2.0992 - val_acc: 0.4553
Epoch 5/100
911/911 [==============================] - 1454s 2s/step - loss: 1.7381 - acc: 0.5040 - val_loss: 2.0312 - val_acc: 0.4590
Epoch 6/100
911/911 [==============================] - 1462s 2s/step - loss: 1.6426 - acc: 0.5268 - val_loss: 2.0813 - val_acc: 0.4598
Epoch 7/100
911/911 [==============================] - 1442s 2s/step - loss: 1.5588 - acc: 0.5487 - val_loss: 2.1401 - val_acc: 0.4669
Epoch 8/100
911/911 [==============================] - 1454s 2s/step - loss: 1.4796 - acc: 0.5688 - val_loss: 2.1547 - val_acc: 0.4639
last model validation_loss: 2.154631296444672 last model validation_acc: 0.46130625702416206
```

```
MODEL RESULTS:
Validation accuracy:  0.4590478803676773
```

8.2.4 Fine Tuning 3 blocks

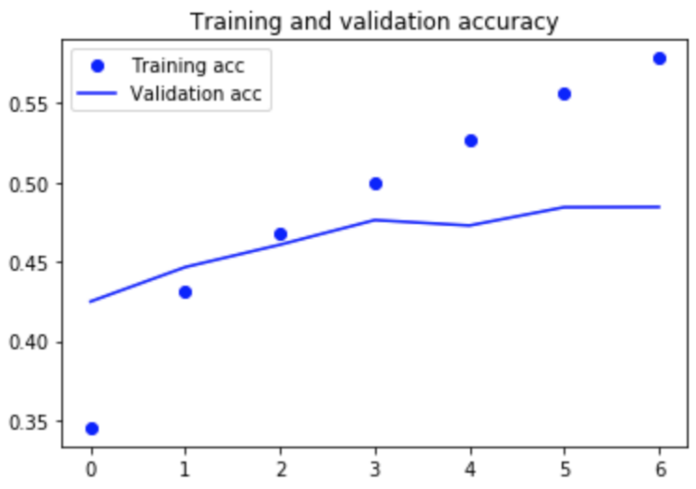


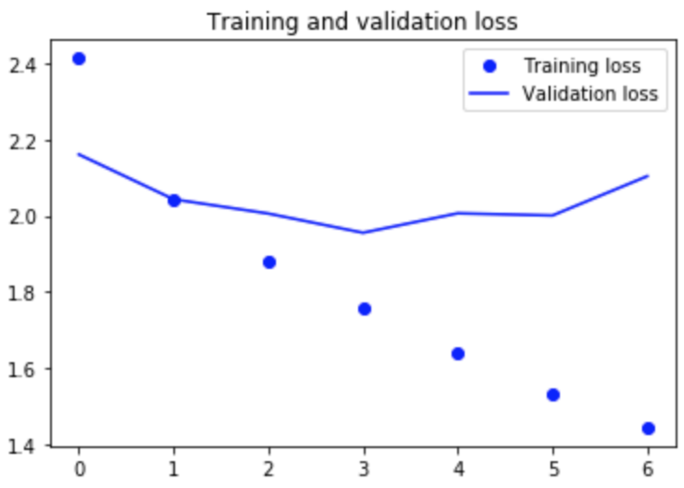


```
Epoch 1/100
911/911 [==============================] - 1556s 2s/step - loss: 2.4655 - acc: 0.3326 - val_loss: 2.1987 - val_acc: 0.4033
Epoch 2/100
911/911 [==============================] - 1489s 2s/step - loss: 2.1077 - acc: 0.4138 - val_loss: 2.0964 - val_acc: 0.4450
Epoch 3/100
911/911 [==============================] - 1492s 2s/step - loss: 1.9480 - acc: 0.4529 - val_loss: 2.0339 - val_acc: 0.4555
Epoch 4/100
911/911 [==============================] - 1482s 2s/step - loss: 1.8242 - acc: 0.4819 - val_loss: 1.9909 - val_acc: 0.4716
Epoch 5/100
911/911 [==============================] - 1472s 2s/step - loss: 1.7206 - acc: 0.5077 - val_loss: 2.0402 - val_acc: 0.4639
Epoch 6/100
911/911 [==============================] - 1457s 2s/step - loss: 1.6247 - acc: 0.5319 - val_loss: 2.0071 - val_acc: 0.4776
Epoch 7/100
911/911 [==============================] - 1472s 2s/step - loss: 1.5343 - acc: 0.5514 - val_loss: 2.0965 - val_acc: 0.4614
last model validation_loss: 2.0849299829956207 last model validation_acc: 0.4636388588553595


MODEL RESULTS:
Validation accuracy:  0.4716696391823295
```

8.2.5 Fine Tuning 4 blocks

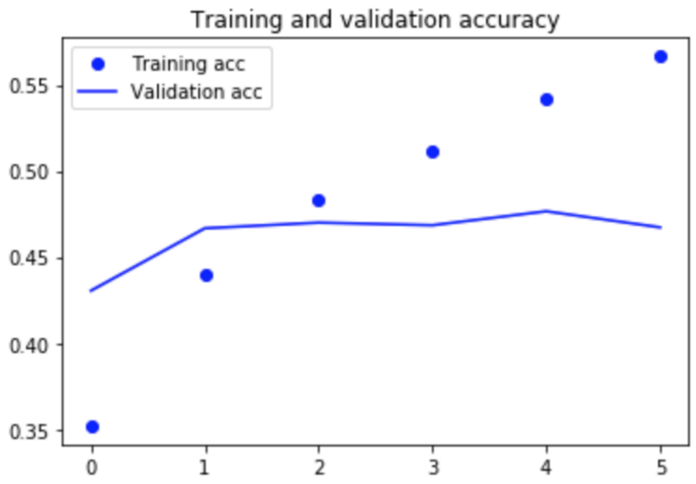


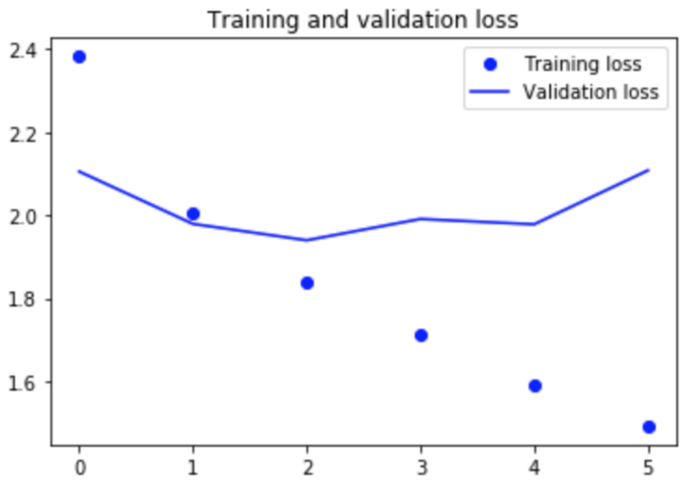


```
Epoch 1/100
911/911 [==============================] - 1557s 2s/step - loss: 2.4141 - acc: 0.3455 - val_loss: 2.1612 - val_acc: 0.4249
Epoch 2/100
911/911 [==============================] - 1498s 2s/step - loss: 2.0444 - acc: 0.4316 - val_loss: 2.0433 - val_acc: 0.4466
Epoch 3/100
911/911 [==============================] - 1512s 2s/step - loss: 1.8815 - acc: 0.4680 - val_loss: 2.0058 - val_acc: 0.4608
Epoch 4/100
911/911 [==============================] - 1498s 2s/step - loss: 1.7551 - acc: 0.4995 - val_loss: 1.9553 - val_acc: 0.4763
Epoch 5/100
911/911 [==============================] - 1499s 2s/step - loss: 1.6397 - acc: 0.5266 - val_loss: 2.0065 - val_acc: 0.4728
Epoch 6/100
911/911 [==============================] - 1492s 2s/step - loss: 1.5337 - acc: 0.5564 - val_loss: 2.0007 - val_acc: 0.4844
Epoch 7/100
911/911 [==============================] - 1494s 2s/step - loss: 1.4436 - acc: 0.5784 - val_loss: 2.1039 - val_acc: 0.4845
last model validation_loss: 2.0905812639828962 last model validation_acc: 0.48970911083446506


MODEL RESULTS:
Validation accuracy:  0.4763342022225271
```

8.2.6 Fine Tuning 5 blocks

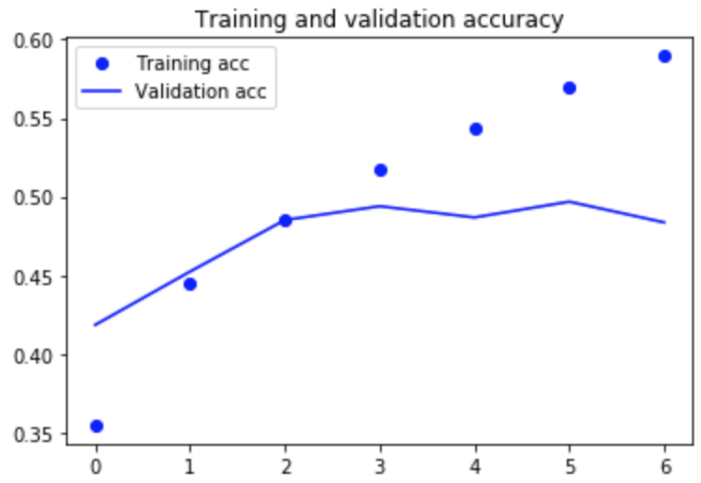


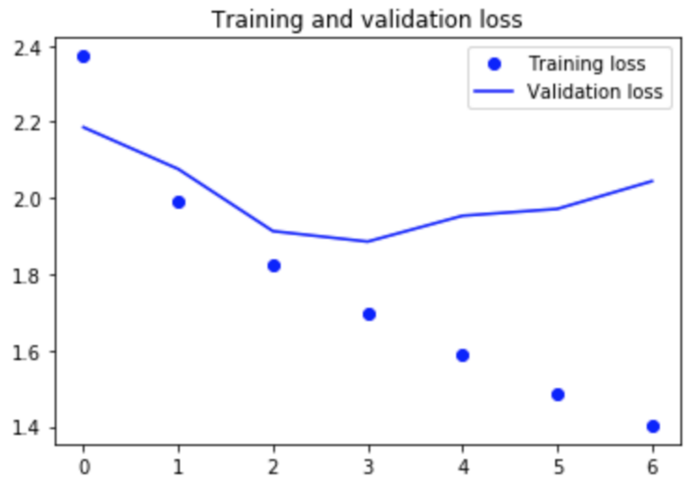


```
Epoch 1/100
911/911 [==============================] - 1560s 2s/step - loss: 2.3819 - acc: 0.3520 - val_loss: 2.1050 - val_acc: 0.4304
Epoch 2/100
911/911 [==============================] - 1512s 2s/step - loss: 2.0045 - acc: 0.4401 - val_loss: 1.9790 - val_acc: 0.4667
Epoch 3/100
911/911 [==============================] - 1520s 2s/step - loss: 1.8402 - acc: 0.4829 - val_loss: 1.9396 - val_acc: 0.4700
Epoch 4/100
911/911 [==============================] - 1512s 2s/step - loss: 1.7146 - acc: 0.5117 - val_loss: 1.9909 - val_acc: 0.4684
Epoch 5/100
911/911 [==============================] - 1509s 2s/step - loss: 1.5926 - acc: 0.5423 - val_loss: 1.9780 - val_acc: 0.4765
Epoch 6/100
911/911 [==============================] - 1508s 2s/step - loss: 1.4930 - acc: 0.5666 - val_loss: 2.1076 - val_acc: 0.4672
last model validation_loss: 2.1003863764384967 last model validation_acc: 0.46857848531197505


MODEL RESULTS:
Validation accuracy:  0.4698861297846069
```

8.2.7 Fine Tuning 6 blocks



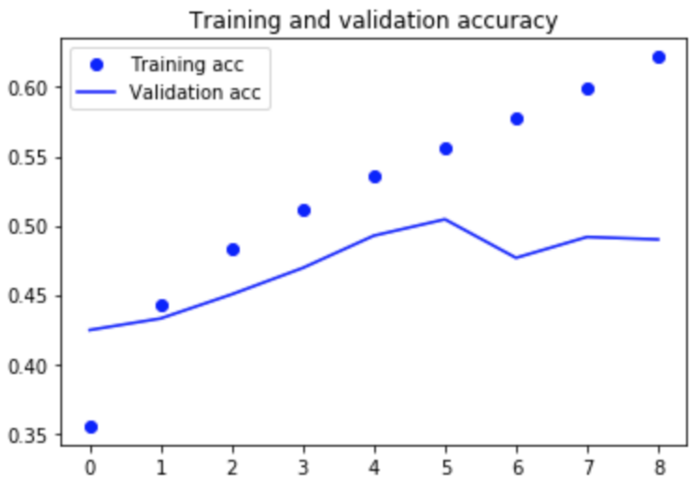


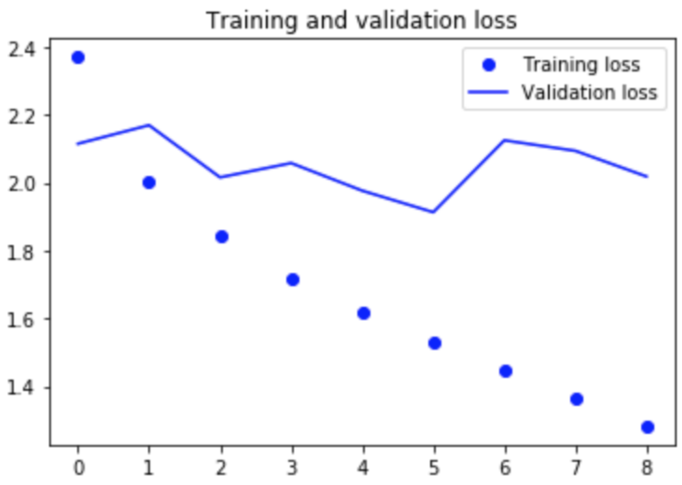
```
Epoch 1/100
911/911 [==============================] - 1666s 2s/step - loss: 2.3729 - acc: 0.3548 - val_loss: 2.1860 - val_acc: 0.4185
Epoch 2/100
911/911 [==============================] - 1520s 2s/step - loss: 1.9925 - acc: 0.4444 - val_loss: 2.0764 - val_acc: 0.4524
Epoch 3/100
911/911 [==============================] - 1525s 2s/step - loss: 1.8250 - acc: 0.4854 - val_loss: 1.9132 - val_acc: 0.4852
Epoch 4/100
911/911 [==============================] - 1518s 2s/step - loss: 1.6978 - acc: 0.5171 - val_loss: 1.8858 - val_acc: 0.4940
Epoch 5/100
911/911 [==============================] - 1533s 2s/step - loss: 1.5895 - acc: 0.5433 - val_loss: 1.9535 - val_acc: 0.4868
Epoch 6/100
911/911 [==============================] - 1525s 2s/step - loss: 1.4878 - acc: 0.5695 - val_loss: 1.9716 - val_acc: 0.4968
Epoch 7/100
911/911 [==============================] - 1527s 2s/step - loss: 1.4032 - acc: 0.5896 - val_loss: 2.0445 - val_acc: 0.4837
last model validation_loss: 2.034902963496982 last model validation_acc: 0.4880625689330661
```

```
MODEL RESULTS:
```

```
Validation accuracy:  0.494032103169159
```

8.2.8 Fine Tuning 7 blocks



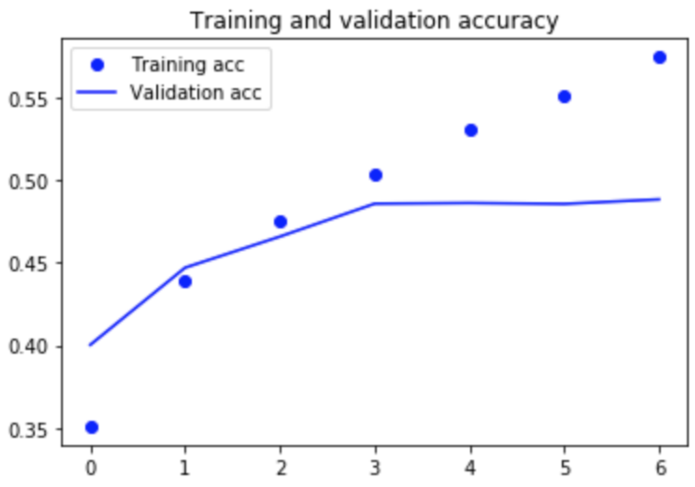


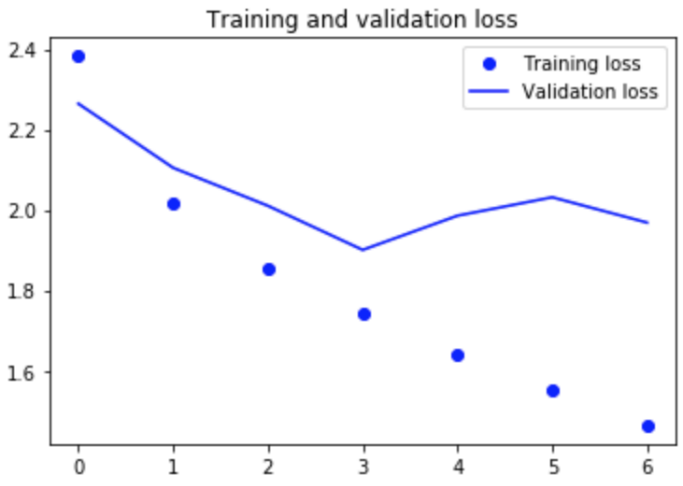
```
Epoch 1/100
911/911 [==============================] - 1667s 2s/step - loss: 2.3707 - acc: 0.3552 - val_loss: 2.1149 - val_acc: 0.4245
Epoch 2/100
911/911 [==============================] - 1529s 2s/step - loss: 2.0037 - acc: 0.4425 - val_loss: 2.1695 - val_acc: 0.4329
Epoch 3/100
911/911 [==============================] - 1521s 2s/step - loss: 1.8422 - acc: 0.4836 - val_loss: 2.0156 - val_acc: 0.4503
Epoch 4/100
911/911 [==============================] - 1525s 2s/step - loss: 1.7183 - acc: 0.5118 - val_loss: 2.0578 - val_acc: 0.4694
Epoch 5/100
911/911 [==============================] - 1512s 2s/step - loss: 1.6193 - acc: 0.5356 - val_loss: 1.9764 - val_acc: 0.4927
Epoch 6/100
911/911 [==============================] - 1524s 2s/step - loss: 1.5301 - acc: 0.5562 - val_loss: 1.9129 - val_acc: 0.5045
Epoch 7/100
911/911 [==============================] - 1529s 2s/step - loss: 1.4453 - acc: 0.5782 - val_loss: 2.1249 - val_acc: 0.4767
Epoch 8/100
911/911 [==============================] - 1514s 2s/step - loss: 1.3645 - acc: 0.5998 - val_loss: 2.0939 - val_acc: 0.4918
Epoch 9/100
911/911 [==============================] - 1522s 2s/step - loss: 1.2822 - acc: 0.6219 - val_loss: 2.0184 - val_acc: 0.4900
```

```
last model validation_loss: 2.012993316226419 last model validation_acc: 0.49053238212866


MODEL RESULTS:
Validation accuracy:  0.5044587734943065
```

8.2.9 Fine Tuning 8 blocks

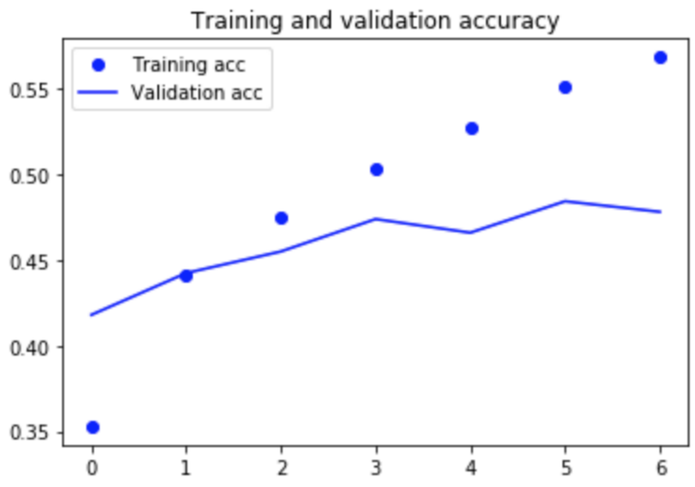


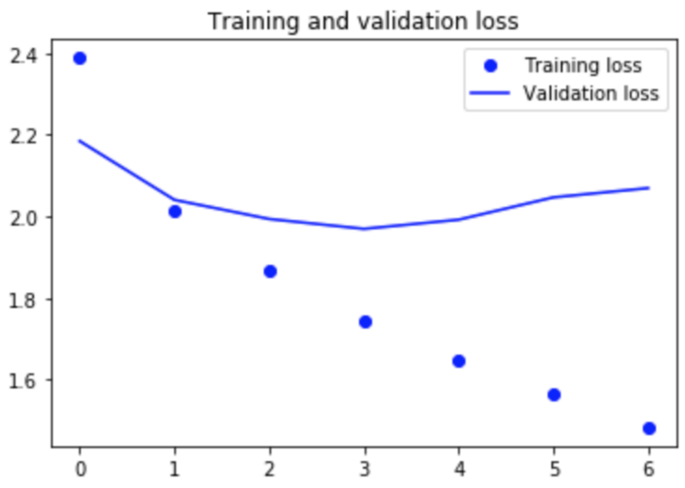


```
Epoch 1/100
911/911 [==============================] - 1689s 2s/step - loss: 2.3831 - acc: 0.3510 - val_loss: 2.2645 - val_acc: 0.4002
Epoch 2/100
911/911 [==============================] - 1543s 2s/step - loss: 2.0148 - acc: 0.4385 - val_loss: 2.1049 - val_acc: 0.4469
Epoch 3/100
911/911 [==============================] - 1548s 2s/step - loss: 1.8562 - acc: 0.4756 - val_loss: 2.0103 - val_acc: 0.4657
Epoch 4/100
911/911 [==============================] - 1543s 2s/step - loss: 1.7428 - acc: 0.5040 - val_loss: 1.9006 - val_acc: 0.4856
Epoch 5/100
911/911 [==============================] - 1528s 2s/step - loss: 1.6419 - acc: 0.5304 - val_loss: 1.9857 - val_acc: 0.4860
Epoch 6/100
911/911 [==============================] - 1536s 2s/step - loss: 1.5534 - acc: 0.5504 - val_loss: 2.0315 - val_acc: 0.4855
Epoch 7/100
911/911 [==============================] - 1543s 2s/step - loss: 1.4640 - acc: 0.5743 - val_loss: 1.9685 - val_acc: 0.4882
last model validation_loss: 1.94711599601218 last model validation_acc: 0.4921789243899114


MODEL RESULTS:
Validation accuracy:  0.4856633283029222
```

8.2.10 Fine Tuning 9 blocks





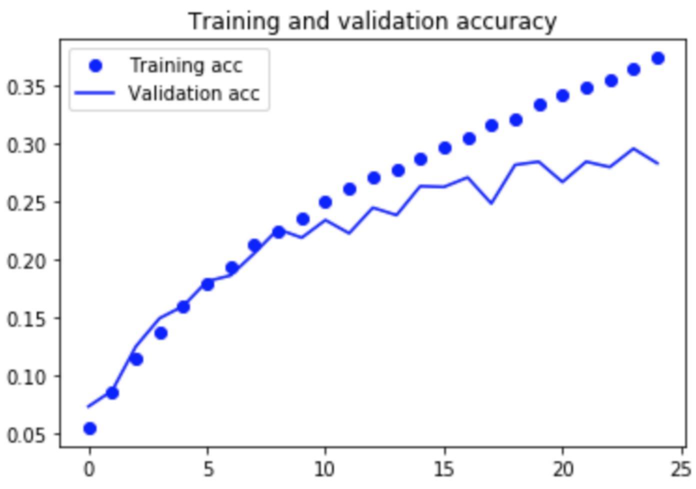
```
Epoch 1/100
911/911 [==============================] - 1675s 2s/step - loss: 2.3884 - acc: 0.3528 - val_loss: 2.1845 - val_acc: 0.4179
Epoch 2/100
911/911 [==============================] - 1557s 2s/step - loss: 2.0148 - acc: 0.4408 - val_loss: 2.0405 - val_acc: 0.4424
Epoch 3/100
911/911 [==============================] - 1562s 2s/step - loss: 1.8671 - acc: 0.4753 - val_loss: 1.9937 - val_acc: 0.4550
Epoch 4/100
911/911 [==============================] - 1561s 2s/step - loss: 1.7462 - acc: 0.5029 - val_loss: 1.9695 - val_acc: 0.4739
Epoch 5/100
911/911 [==============================] - 1552s 2s/step - loss: 1.6470 - acc: 0.5272 - val_loss: 1.9919 - val_acc: 0.4660
Epoch 6/100
911/911 [==============================] - 1555s 2s/step - loss: 1.5659 - acc: 0.5513 - val_loss: 2.0466 - val_acc: 0.4844
Epoch 7/100
911/911 [==============================] - 1552s 2s/step - loss: 1.4833 - acc: 0.5686 - val_loss: 2.0693 - val_acc: 0.4782
last model validation_loss: 2.0921138076175056 last model validation_acc: 0.4704994511525796
```

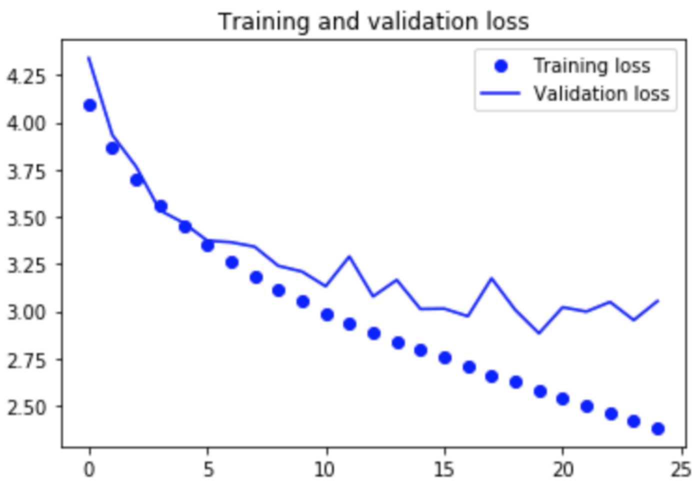
```
MODEL RESULTS:
```

```
Validation accuracy:  0.4740019207024283
```

8.3 Results for Complete Training of Miscellanous Models

8.3.1 InceptionV3



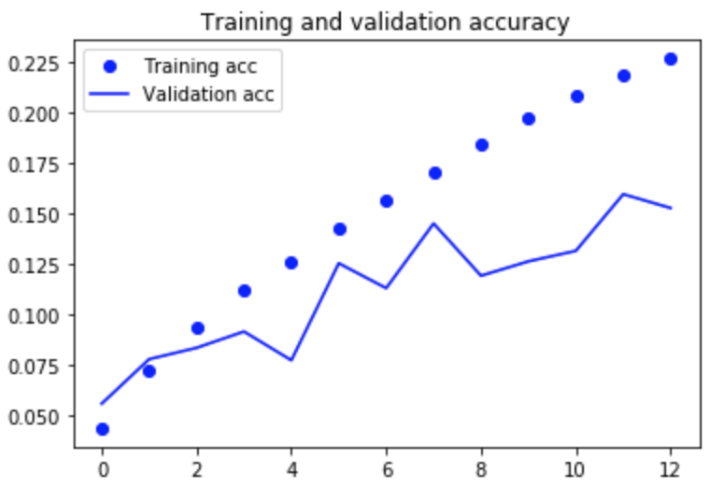


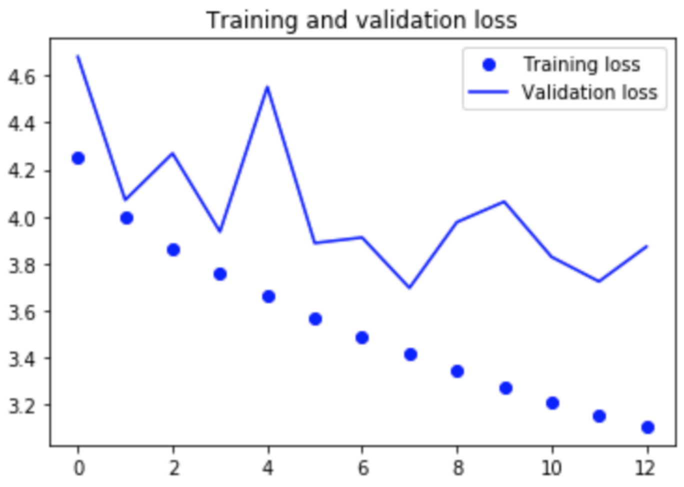
```
Epoch 1/100
911/911 [==============================] - 2151s 2s/step - loss: 4.0908 - acc: 0.0554 - val_loss: 4.3404 - val_acc: 0.0735
Epoch 2/100
911/911 [==============================] - 2078s 2s/step - loss: 3.8675 - acc: 0.0859 - val_loss: 3.9328 - val_acc: 0.0871
Epoch 3/100
911/911 [==============================] - 2091s 2s/step - loss: 3.7022 - acc: 0.1153 - val_loss: 3.7659 - val_acc: 0.1253
Epoch 4/100
911/911 [==============================] - 2103s 2s/step - loss: 3.5646 - acc: 0.1383 - val_loss: 3.5343 - val_acc: 0.1496
Epoch 5/100
911/911 [==============================] - 2106s 2s/step - loss: 3.4514 - acc: 0.1594 - val_loss: 3.4680 - val_acc: 0.1601
Epoch 6/100
911/911 [==============================] - 2094s 2s/step - loss: 3.3508 - acc: 0.1796 - val_loss: 3.3736 - val_acc: 0.1814
Epoch 7/100
```

```
911/911 [==============================] - 2096s 2s/step - loss: 3.2652 - acc: 0.1947 - val_loss: 3.3652 - val_acc: 0.1865
Epoch 8/100
911/911 [==============================] - 2096s 2s/step - loss: 3.1846 - acc: 0.2129 - val_loss: 3.3415 - val_acc: 0.2057
Epoch 9/100
911/911 [==============================] - 2094s 2s/step - loss: 3.1174 - acc: 0.2245 - val_loss: 3.2400 - val_acc: 0.2264
Epoch 10/100
911/911 [==============================] - 2090s 2s/step - loss: 3.0544 - acc: 0.2359 - val_loss: 3.2097 - val_acc: 0.2190
Epoch 11/100
911/911 [==============================] - 2086s 2s/step - loss: 2.9897 - acc: 0.2510 - val_loss: 3.1318 - val_acc: 0.2342
Epoch 12/100
911/911 [==============================] - 2086s 2s/step - loss: 2.9379 - acc: 0.2619 - val_loss: 3.2896 - val_acc: 0.2228
Epoch 13/100
911/911 [==============================] - 2083s 2s/step - loss: 2.8853 - acc: 0.2713 - val_loss: 3.0779 - val_acc: 0.2449
Epoch 14/100
911/911 [==============================] - 2080s 2s/step - loss: 2.8415 - acc: 0.2774 - val_loss: 3.1652 - val_acc: 0.2385
Epoch 15/100
911/911 [==============================] - 2080s 2s/step - loss: 2.7944 - acc: 0.2879 - val_loss: 3.0111 - val_acc: 0.2634
Epoch 16/100
911/911 [==============================] - 2080s 2s/step - loss: 2.7545 - acc: 0.2974 - val_loss: 3.0132 - val_acc: 0.2628
Epoch 17/100
911/911 [==============================] - 2083s 2s/step - loss: 2.7057 - acc: 0.3052 - val_loss: 2.9730 - val_acc: 0.2710
Epoch 18/100
911/911 [==============================] - 2084s 2s/step - loss: 2.6612 - acc: 0.3164 - val_loss: 3.1738 - val_acc: 0.2485
Epoch 19/100
911/911 [==============================] - 2083s 2s/step - loss: 2.6256 - acc: 0.3209 - val_loss: 3.0065 - val_acc: 0.2818
Epoch 20/100
911/911 [==============================] - 2086s 2s/step - loss: 2.5795 - acc: 0.3339 - val_loss: 2.8823 - val_acc: 0.2846
Epoch 21/100
911/911 [==============================] - 2078s 2s/step - loss: 2.5411 - acc: 0.3421 - val_loss: 3.0206 - val_acc: 0.2670
Epoch 22/100
911/911 [==============================] - 2086s 2s/step - loss: 2.4986 - acc: 0.3490 - val_loss: 2.9976 - val_acc: 0.2846
Epoch 23/100
911/911 [==============================] - 2088s 2s/step - loss: 2.4615 - acc: 0.3557 - val_loss: 3.0488 - val_acc: 0.2799
Epoch 24/100
911/911 [==============================] - 2088s 2s/step - loss: 2.4227 - acc: 0.3649 - val_loss: 2.9523 - val_acc: 0.2958
Epoch 25/100
911/911 [==============================] - 2088s 2s/step - loss: 2.3822 - acc: 0.3741 - val_loss: 3.0533 - val_acc: 0.2832
last model validation_loss: 3.05653485116006 last model validation_acc: 0.2826564214493912


MODEL RESULTS:
Validation accuracy:  0.28453834545205103
```

8.3.2 ResNet



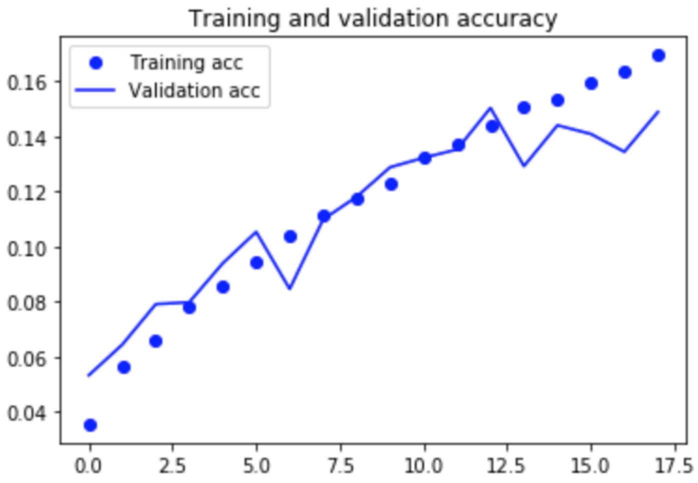


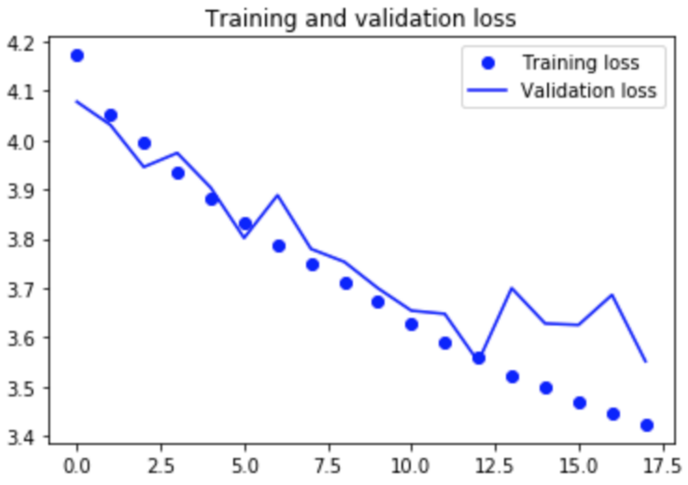
```
Epoch 1/100
911/911 [==============================] - 1578s 2s/step - loss: 4.2503 - acc: 0.0434 - val_loss: 4.6785 - val_acc: 0.0556
Epoch 2/100
911/911 [==============================] - 1556s 2s/step - loss: 3.9958 - acc: 0.0722 - val_loss: 4.0702 - val_acc: 0.0777
Epoch 3/100
911/911 [==============================] - 1554s 2s/step - loss: 3.8661 - acc: 0.0932 - val_loss: 4.2681 - val_acc: 0.0833
Epoch 4/100
911/911 [==============================] - 1552s 2s/step - loss: 3.7572 - acc: 0.1117 - val_loss: 3.9355 - val_acc: 0.0914
Epoch 5/100
911/911 [==============================] - 1552s 2s/step - loss: 3.6630 - acc: 0.1264 - val_loss: 4.5507 - val_acc: 0.0771
Epoch 6/100
911/911 [==============================] - 1551s 2s/step - loss: 3.5706 - acc: 0.1429 - val_loss: 3.8873 - val_acc: 0.1253
Epoch 7/100
911/911 [==============================] - 1553s 2s/step - loss: 3.4887 - acc: 0.1561 - val_loss: 3.9109 - val_acc: 0.1129
Epoch 8/100
911/911 [==============================] - 1547s 2s/step - loss: 3.4150 - acc: 0.1709 - val_loss: 3.6968 - val_acc: 0.1450
Epoch 9/100
```

```
911/911 [==============================] - 1551s 2s/step - loss: 3.3461 - acc: 0.1841 - val_loss: 3.9762 - val_acc: 0.1191
Epoch 10/100
911/911 [==============================] - 1553s 2s/step - loss: 3.2763 - acc: 0.1976 - val_loss: 4.0630 - val_acc: 0.1262
Epoch 11/100
911/911 [==============================] - 1553s 2s/step - loss: 3.2125 - acc: 0.2085 - val_loss: 3.8284 - val_acc: 0.1314
Epoch 12/100
911/911 [==============================] - 1551s 2s/step - loss: 3.1584 - acc: 0.2189 - val_loss: 3.7241 - val_acc: 0.1596
Epoch 13/100
911/911 [==============================] - 1553s 2s/step - loss: 3.1090 - acc: 0.2269 - val_loss: 3.8720 - val_acc: 0.1527
last model validation_loss: 3.865097592874102 last model validation_acc: 0.15230515913303808


MODEL RESULTS:
Validation accuracy:  0.14501303333790644
```

8.3.3 SimpleNet





```
Epoch 1/100
911/911 [==============================] - 1275s 1s/step - loss: 4.1725 - acc: 0.0356 - val_loss: 4.0778 - val_acc: 0.0534
Epoch 2/100
911/911 [==============================] - 1244s 1s/step - loss: 4.0537 - acc: 0.0563 - val_loss: 4.0311 - val_acc: 0.0645
Epoch 3/100
911/911 [==============================] - 1242s 1s/step - loss: 3.9967 - acc: 0.0658 - val_loss: 3.9456 - val_acc: 0.0792
Epoch 4/100
911/911 [==============================] - 1246s 1s/step - loss: 3.9362 - acc: 0.0786 - val_loss: 3.9740 - val_acc: 0.0799
Epoch 5/100
911/911 [==============================] - 1243s 1s/step - loss: 3.8819 - acc: 0.0855 - val_loss: 3.9039 - val_acc: 0.0940
Epoch 6/100
911/911 [==============================] - 1248s 1s/step - loss: 3.8328 - acc: 0.0946 - val_loss: 3.8016 - val_acc: 0.1054
Epoch 7/100
911/911 [==============================] - 1247s 1s/step - loss: 3.7885 - acc: 0.1037 - val_loss: 3.8885 - val_acc: 0.0847
Epoch 8/100
911/911 [==============================] - 1247s 1s/step - loss: 3.7485 - acc: 0.1114 - val_loss: 3.7796 - val_acc: 0.1099
Epoch 9/100
911/911 [==============================] - 1243s 1s/step - loss: 3.7101 - acc: 0.1175 - val_loss: 3.7532 - val_acc: 0.1181
```

```
Epoch 10/100
911/911 [==============================] - 1247s 1s/step - loss: 3.6719 - acc: 0.1232 - val_loss: 3.7000 - val_acc: 0.1288
Epoch 11/100
911/911 [==============================] - 1243s 1s/step - loss: 3.6283 - acc: 0.1323 - val_loss: 3.6545 - val_acc: 0.1323
Epoch 12/100
911/911 [==============================] - 1245s 1s/step - loss: 3.5922 - acc: 0.1371 - val_loss: 3.6479 - val_acc: 0.1353
Epoch 13/100
911/911 [==============================] - 1245s 1s/step - loss: 3.5604 - acc: 0.1438 - val_loss: 3.5512 - val_acc: 0.1504
Epoch 14/100
911/911 [==============================] - 1245s 1s/step - loss: 3.5213 - acc: 0.1505 - val_loss: 3.7001 - val_acc: 0.1293
Epoch 15/100
911/911 [==============================] - 1241s 1s/step - loss: 3.5000 - acc: 0.1532 - val_loss: 3.6284 - val_acc: 0.1441
Epoch 16/100
911/911 [==============================] - 1247s 1s/step - loss: 3.4688 - acc: 0.1597 - val_loss: 3.6255 - val_acc: 0.1409
Epoch 17/100
911/911 [==============================] - 1242s 1s/step - loss: 3.4463 - acc: 0.1640 - val_loss: 3.6864 - val_acc: 0.1345
Epoch 18/100
911/911 [==============================] - 1243s 1s/step - loss: 3.4242 - acc: 0.1696 - val_loss: 3.5518 - val_acc: 0.1489
last model validation_loss: 3.549648682450882 last model validation_acc: 0.1488748627881449


MODEL RESULTS:
Validation accuracy:  0.15036356153107422
```

Bibliography

[1] Chollet, F. (2017) *Deep Learning with Python*. Manning Publications Co.

[2] http://yann.lecun.com/exdb/mnist/

[3] http://www.image-net.org/

[4] https://www.cs.toronto.edu/ kriz/cifar.html

[5] https://en.wikipedia.org/wiki/Computer_vision

[6] https://en.wikipedia.org/wiki/Artificial_neural_network

[7] https://en.wikipedia.org/wiki/Perceptron

[8] https://medium.com/comet-app/review-of-deep-learning-algorithms-for-image-classification-5fdbca4a05e2

[9] https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

[10] https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

[11] Hasanpour, S., Rouhani, M., Fayyaz, M. and Sabokrou, M. (2016). Lets keep it simple: using simple architectures to outperform deeper and more complex architectures. arXiv:1608.06037

[12] He K., Zhang X., Ren S. and Sun J. (2015) Deep Residual Learning for Image Recognition. arXiv:1512.03385

[13] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2015) Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567

[14] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2014) Going Deeper with Convolutions. arXiv:1409.4842

[15] https://github.com/EricAlcaide/SimpleNet-Keras/blob/master/simplenet.py