Universitat Oberta de Catalunya (UOC)

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

MASTER'S THESIS

Área: MEDICINE

Designing an unsupervised learning architecture to reconstruct brain magnetic resonance images

---

Author: Enric Pou Robert

Supervisor: Baris Kanber

Profesor: Ferran Prados Carrasco

Barcelona, January 3, 2021

Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Designing an unsupervised learning architecture to reconstruct brain magnetic resonance images
Nombre del autor:	Enric Pou Robert
Nombre del colaborador/a docente:	Baris Kanber
Nombre del PRA:	Ferran Prados Carrasco
Fecha de entrega (mm/aaaa):	01/2021
Titulación o programa:	Máster en Ciència de Datos
Área del Trabajo Final:	Medicina
Idioma del trabajo:	Inglés
Palabras clave	autoencoders, image reconstruction, brain magnetic resonance imaging

Dedication

To my family and friends, for all their support during the hard moments and to encourage me towards what makes me happy.

Thank you all.

Acknowledgments

First of all, I would like to thank Dr. Baris Kanber for his expert advice and encouragement throughout this project and for his support in this thesis process. You supported me greatly and were always willing to help me. I would like to thank all the training received by the UOC during these years that have made me grow as a person and make me continue loving my passion for technology.

Abstract

Brain magnetic resonance imaging (MRI) is a well-known medical imaging modality used by physicians to diagnose neurological disorders. A professional with great knowledge in the field can analyze the obtained 3D volume to detect abnormalities.

The main goal of this project is to design and develop a neural network model capable of reconstructing brain MRIs by using an unsupervised approach. In essence, it has to give as result the healthy version of the brain used as input. Therefore, the model has to learn the intrinsic patterns from the healthy brains in order to transfer them to the output. For that reason, T1-weighted images from the IXI Dataset have been used to train our models only with pathology-free images. To ensure the model was learning the desired patterns, not only copying the input directly into the output, the data has been corrupted by applying different types of noise and by masking-out regions.

Motivated by the autoencoders simplicity, and with the intention of generating the resulting images with better quality, some experiments have been conducted using different loss functions and trying with different types of skipped connections. The use of both residual blocks and shortcuts to skip long connections has been the combination that has led to the best results.

Keywords: unsupervised learning, image reconstruction, autoencoders, brain MRI.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The brain is still a great unknown. Although the functioning of neurons and the brain, in general, has been studied for more than 100 years, we still know very little about their functioning, in relation to what we know about other vital organs of animals and humans. Like any other organ, it can develop diseases, which their monitoring and screening can save lives if detected on time. Given the importance of the brain, the pathologies it can develop can be fatal or cause permanent and severe effects due to invasive solutions are delicate surgeries.

Diagnostic tests and procedures are vital tools that help physicians confirm or rule out a neurological disorder. A century ago, the only way to make a definite diagnosis for many neurological disorders was to perform an autopsy after someone had died. Today, new instruments and techniques allow scientists to assess the living brain and monitor nervous system activity as it occurs. Doctors now have powerful and accurate tools to better diagnose disease and to test how well a particular therapy may be working.

Perhaps the most significant changes during the past 10 years have occurred in diagnostic imaging. Improved imaging techniques provide high-resolution images that allow physicians to view the structure of the brain. Thanks to these specialized imaging methods one can visualize changes in brain activity or even the amounts of particular brain chemicals. Scientists continue to improve these methods to provide more detailed diagnostic information.

The most common modality for non-invasive diagnoses is magnetic resonance imaging (MRI), which uses both radio waves and computer vision to obtain detailed pictures from the inside of the brain. So, a bunch of images is obtained, each corresponding to a slice of the subject's brain.

The first thing to take into consideration is the quality of the resulting images. MRI is still a field in development in terms of noise or artifacts which affect the final quality. There are several reasons that MRI is sensitive to these undesirable errors, the main being the hardware-induced ones, such as the sudden appearance of undesired magnetic fields. Artifacts are caused by various reasons such as external errors and inappropriate spatial encoding, being the aliasing induced by resampling the most common one [1].

Another aspect it needs to be taken into account is the diagnosis of neurological pathologies. Manual diagnosis is based on the radiologist's viewing, which is time consuming due to the large amount of MRI slice to be analysed. As it requires great knowledge in the field, all the tasks performed to detect abnormalities consume a vast amount of time and might end up with a wrong diagnosis. Reading and interpreting MR scans in an intricate process. It is estimated that in 5-10% of scans, relevant pathology is missed [2]. It would be therefore a real need the fact of automatizing these tasks, not for replacing the role of the radiologist, but to help them detect abnormalities and thus making better decisions.

Being known that convolutional neural networks (CNN) work well with image-related problems, it is undoubtedly a good chance for the deep learning field to provide help with the matter. Although the system to automatize the full process chain is challenging and innovative, the first step would be to develop a neural network capable of modeling and learning how a healthy brain MRI is, in addition to being able to remove any sort of noise or artifacts present in the image. Having that, then it could be used to reconstruct non-healthy brain MRIs and obtain its pathology-free version, being simple then to determine if the image used as input corresponds to a healthy brain or not, based on the differences with the resulting image. Other systems could be developed based on this response, such as a pathology detector or a classifier, among others.

1.2 Goals

This project aims at advancing in the field of brain MRI reconstruction (both noise and artifact removal) and automatically generating the healthy version of the MRI used as input by learning the patterns that identifies the pathology-free subjects. Such an abstract goal is decomposed into various, as follows:

1. Getting a solid understanding of the problem and review the state-of-art solutions.

2. Getting practical knowledge on the technologies required to solve this problem.
3. Applying data augmentation to the dataset.
4. Designing neural network architecture candidates.
5. Training and testing the networks developed.
6. Choosing a network that is resilient to noise and can reconstruct masked regions of the image and obtain the pathology-free version of it, ensuring that the network is learning the patterns right and not copying the input directly into the output.
7. Choosing a metric to evaluate the results.
8. Documenting the whole procedure and the results.

1.3 Methodology

This project is research-oriented and it includes both an extensive gathering of state-of-art articles and the development of a new model to solve the target problem using real data. Due to these reasons, it can benefit from the *Cross-Industry Standard Process for Data Mining* (**CRISP-DM**) methodology.

CRISP-DM is an open standard process model that describes common approaches used by data mining projects and is based on iteration to achieve the objective. The sequence of the phases is not strict and moving back and forth between different phases as it is always required. The arrows in the process diagram (as shown in figure 1.1) indicate the most important and frequent dependencies between phases. The outer circle in the diagram symbolizes the cyclic nature of data mining itself. A data mining process continues after a solution has been deployed. The lessons learned during the process can trigger new, often more focused business questions, and subsequent data mining processes will benefit from the experiences of previous ones.

The process model consist of six major phases:

- **Business Understanding**: includes a depth analysis of the business. Goals and requirements are set.
- **Data Understanding**: data is captured, explored and analysed to become familiar with it and potential problems are identified. Verify the data quality and format.

- **Data Preparation**: data is selected, cleaned, constructed and formatted to fit the model needs.
- **Modeling**: develop several models and fine-tune them. This phase is usually conducted in multiple iterations.
- **Evaluation**: the previous models are evaluated and checked if they fit the goals and requirements set in the first phase.
- **Publication**: the final information is gathered and has to be presented in a structured and organized manner.

Figure 1.1: Process diagram showing the relationship between the different phases of CRISP-DM. Image by Kenneth Jensen, used under CC BY-SA 3.0

Finally, the whole code will be stored into a version control repository in order to provide a log of changes and commitments. *Git* workflow will be used to keep the project well organized: branch naming model, pull requests, *readme* files, etc.

The code will be written mainly in python (using data-science focused frameworks and libraries) and the code will be structured and written following the best practice: project structure, clean code, comments and instruction files. Due to the model will be developed using the *tensorflow* and *keras* frameworks, will organize the code accordingly. Last, *jupyter notebook* will be used to interact with the code dynamically thanks to the user interface provided.

1.4 Planning

The planning summary is as follows.

Phase	Start date	End date	Description
1	16/09/2020	27/09/2020	Definition and planning
2	28/09/2020	18/10/2020	State of the art
3	19/10/2020	20/12/2020	Development
4	21/12/2020	03/01/2021	Write the report
5	04/01/2021	10/01/2021	Presentation

Table 1.1: Planning summary.

The detailed decomposition of the phases is as follows:

Phase 1 is devoted to understand the project problem and gather information related to data mining project's planning. Tasks to be done:

- Define the subject of the work.
- Justify interest and relevance.
- Define the main objectives.
- Document the plan proposed for the project.

Phase 2 is mainly focused on having a knowledge of state-of-art projects and document oneself to become an expert in the subject. Tasks to be done:

- Search and review relevant bibliography for the research to be developed.
- Redefine the partial objectives defined in the previous activity.
- Identify data generation techniques to be used.
- Organize information consistently.
- Become familiar with the dataset.

Phase 3 is intended to design and develop a neural network architecture to solve the problem. Tasks to be done:

- Prepare the dataset.
- Define a grown-truth solution to compare our results with.
- Processing the dataset.
- Apply data augmentation techniques to our dataset: add noise, masks regions.
- Build one or various architectures based on autoencoders (GAN can also be considered) from scratch or by transfer learning.
- Find optimal hyperparameters by applying grid search techniques.
- Define a metric to evaluate the results. Evaluate the models and compare the results with the ground-truth.
- Keep the code versioned in the repository.

Phase 4 is conceived for completing this report. Tasks to be done:

- Document the whole procedure and the results in this report. Must follow and organized and clean structure and must explain in detail all the steps followed to build the solution.

Phase 5 is focused on publishing the results and presenting them. Tasks to be done:

- Gather all the documentation (this report and the code files) and upload it to UOC servers.
- Expose and defend the project publicly.

Chapter 2

State of the art

Magnetic resonance imaging (MRI) enables the in vivo investigation of the morphological features of the human brain. There is much hope that this tool will help elucidate the neuroanatomical correlates of neuropsychiatric disease, leading to improved detection and treatment [3, 4]. However, despite the very large number of scientific publications in this area over the past two decades, the use of MRI in real-world clinical decision-making remains very limited [5]. One of the reasons is that the vast majority of existing studies have used traditional mass-univariate analytical methods which are sensitive to gross and localized differences in the brain. These techniques are not optimal for detecting neuroanatomical alterations in neuropsychiatric disorders which tend to be subtle and spatially distributed [6, 7].

Machine learning (ML), the field that studies algorithms which improve automatically through experience, provides an alternative analytical approach for tackling that kind of problems. By learning from data provided, these algorithms can carry out certain tasks: classification, clustering, image processing, etc. These approaches are traditionally divided into three broad categories, depending on the nature of the feedback provided to the learning system:

- Supervised learning: The computer is presented with example inputs and their desired outputs, being the goal to learn a general rule that maps inputs to outputs.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input.
- Reinforcement learning: As it navigates its problem space, the program is provided feedback that's analogous to rewards, which it tries to maximize

One important application ML has caused a big impact on is in computer vision (VS), being image processing one of the most active in the research community.

Anomaly detection (AD) in images has been associated to numerous fields, from quality control to medical image analysis [8]. In general, AD can be developed with: no prior knowledge of the data, modelling both normality and anomalies, or modelling only normality. In the medical imaging circumstances, the third approach (which is an unsupervised learning case) is the best suited because the gamut of all possible anomalies is not available in most cases, as some forms of pathology are very infrequent, although, the lack of a priori target information could provoke high false alarm rates [2, 9].

For that reasons, the research community has started facing the problem from an unsupervised point of view. That is, because the supervised one carries some disadvantages [10]:

1. Their training calls for large and diverse annotated datasets, which are scarce and costly to obtain.

2. The resulting models are limited to the discovery of lesions which are similar to those in the training data.

Besides, the unsupervised methodology is more similar to how radiologists read MRI and do not require data with pixel-level annotations and have the potential to detect arbitrary anomalies without a-priori knowing about their appearances [10].

2.1 The role of deep learning

In the last decade, there has been an upsurge of interest in the field of deep learning (DL) and its capabilities to understand and solve a wide variety of problems. Is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech [11].

Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years. It has turned out to be very good at discovering intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government [11]. It rose to its prominent position in computer vision when neural networks started outperforming other methods on several high-profile image analysis benchmarks. Specially has beaten record in image recognition problems such as [12, 13, 14, 15].

2.1.1 Convolutional neural networks

The major advance in the field of image processing in DL was the introduction of convolutional neural networks (CNN) back to 1998 [16]. In 2012, a large deep CNN [12] was used to win (by an incredible wide margin), the 2012 ImageNet Large-Scale Visual Recognition Challenge. From that turning point, the field has attracted the attention of researchers from various fields and gained popularity. Although CNNs trained by backpropagation had been around for decades, and GPU implementations of NNs for years, including CNNs, fast implementations of CNNs on GPUs were needed to progress on computer vision.

CNN are designed to process data that come in the form of multiple arrays, for example a colour image composed of three 2D arrays containing pixel intensities in the three colour channels. The architecture of a typical CNN (see Figure 2.1) is structured as a series of stages. The first few stages are composed of two types of layers: convolutional layers and pooling layers.

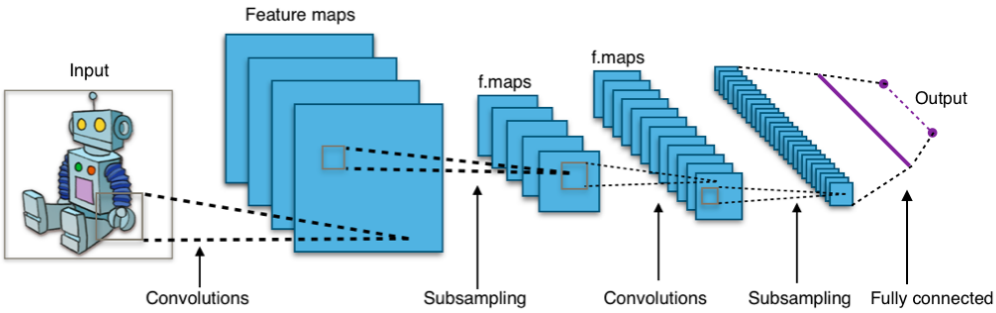


Figure 2.1: Full convolutional neural network schema. Image by Aphex34 (Own work), used under CC BY-SA 4.0

Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linearity such as a ReLU [17]. All units in a feature map share the same filter bank. Different feature maps

in a layer use different filter banks. The reason for this architecture is twofold. First, in array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected. Second, the local statistics of images and other signals are invariant to location. In other words, if a motif can appear in one part of the image, it could appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array [11].

Recent CNN architectures have 10 to 20 layers of ReLUs, hundreds of millions of weights, and billions of connections between units. Whereas training such large networks could have taken weeks only two years ago, progress in hardware, software and algorithm parallelization have reduced training times to a few hours [11]. This success came from the efficient use of GPUs, ReLUs [18], a new regularization technique called dropout [19], and techniques to generate more training examples by deforming the existing ones.

Some of the most common used CNN are (sorted by year of publication):

1. **AlexNet** (2012) [12]: it consist of 8 layers (5 convolutional and 3 fully-connected) that has 60M parameters. AlexNet just stacked a few more layers onto LeNet-5 [16].

2. **VGG-16** (2014) [20]: it has 13 convolutional and 3 fully-connected layers, carrying with them the ReLU tradition from AlexNet. This network stacks more layers onto AlexNet, and use smaller size filters (2x2 and 3x3). It consists of 138M parameters and takes up about 500MB of storage space. The VGG group also designed a deeper variant, VGG-19.

3. **Inception-v1** (2014) [15]: having 22 layers with 5M parameters. Introduces the idea of parallel towers of convolutions with different filters, 1x1 convolutions for dimensionality reduction and two auxiliary classifiers to encourage discrimination in the lower stages of the classifier.

4. **Inception-v3** (2015) [21]: successor to Inception-v1, with 24M parameters.

5. **ResNet-50** (2015) [22]: The basic building block for ResNets are the convolutional and identity blocks. It has 26M parameters. Uses skip connections to deal with the accuracy saturation.

6. **Xception** (2016) [23]: Xception is an adaptation from Inception, where the Inception modules have been replaced with depthwise separable convolutions. It has also roughly the same number of parameters as Inception-v1 (23M). Performs 1x1 to every channel then performs 3x3 to each output.

A summary of the models that are made available alongside pre-trained weights from Keras[1] can be seen in the Table 2.1.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-

Table 2.1: Pre-trained weights which are available in Keras for the architectures that previously exposed. Adapted from a table in the Keras documentation. The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

2.2 Task definition

Since our main goal is to obtain the healthy version of the brain MRI used as input, it is reasonable to search for literature based on image reconstruction. That is, taking into account those models capable of understanding the hidden patterns that make healthy brains unique.

So that, models which work well for the following tasks might be interesting:

- Noise and artifact removal.
- Image impainting.
- Unsupervised anomaly detection (UAD).

The first two scenarios can be applied to any sort of images while the last one is more focused on MRI.

MSE, PSNR, and SSIM are widely used to evaluate the image quality, particularly for noise reduction networks [1]. Sørensen-Dice similarity coefficient for image segmentation is used too.

[1]Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. See https://keras.io.

Our research has been mainly focused on Deep Learning (DL) approaches as they set the current state of the art for all the tasks beforehand mentioned. Despite that, a review on more classical approaches (such as statistical modeling [24], content-based retrieval [25, 26], component analysis [27], clustering or outlier-detection) with focus on brain CT imaging is given in [8] and a full review of classical image impainting in [28].

2.3 Deep learning approaches

2.3.1 Autoencoders

An autoencoder (AE) is a neural network based on an encoder-decoder paradigm that learns to, basically, copy its input into its output. It is constituted by two networks: the encoder and the decoder. First the image is encoded into a lower-dimensional representation of it (an embedding). Then, the decoder will work from this embedding to reconstruct the original image. The two networks are trained jointly to minimize the difference between input and output. This architecture forces the encoder to try to encode as much information as possible into the embedding. Since this is a small vector (a bottleneck), the encoder has to learn high-level, smarter features to compress the input with minimum loss. Those high-level features are hence more robust to noise and changes than raw pixels. Unfortunately, autoencoders may lose important information such as edge or fine structure, if the input image is not a redundant representation [1]. A schema of a basic autoencoder can be seen in Figure 2.2.

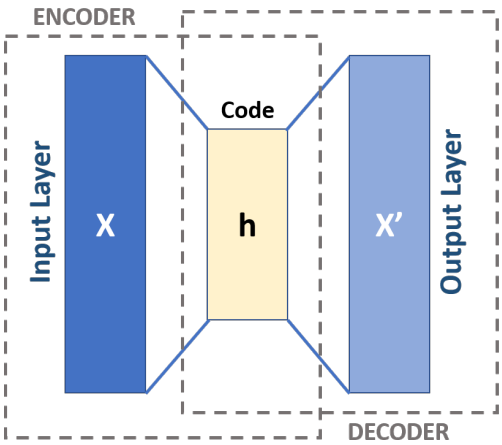


Figure 2.2: Schema of a basic autoencoder. Image by Michela Massi (Own work), used under CC BY-SA 4.0

Applied to our use case, the core concept is the modeling of healthy anatomy with unsu-

pervised deep representation learning. Therefore, the methods leverage a set of healthy MRI scans and learn to project it to and recover it from a lower dimensional distribution (figure 2.3). The rationale behind this is the assumption that an AE trained on only healthy samples cannot properly reconstruct anomalies in pathological data. This approach has been successfully applied to anomaly segmentation in brain MRI [29, 30].

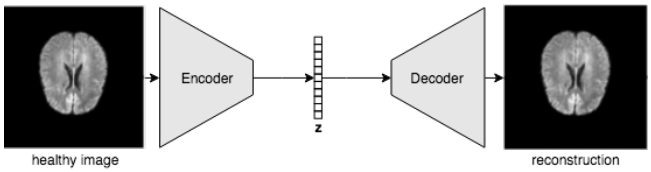


Figure 2.3: Schema of a basic autoencoder applied to MRI. Source [10]

Some networks focus on denoising for brain MR imaging. Noise removal problems aim to extract noise from images in order to remove it from the original one. They use single-scale CNN (SCNN) which is efficient in improving noisy images [31, 32]. SCNN consists of more than one convolution layer without a down- or up-sampling structure, such as pooling and stride. In most cases, residual components such as noise or artifact images are used as the output because it is well-known that the manifold of the residual components has a simpler structure compared to that of the clean image components [33]. Denoising convolutional neural networks (DnCNN) is also a thriving network used for image denoising through a process of learning residual components of images, which is noise. ResNet is commonly used to achieve a complex and deeper structure of the network to overcome the vanishing gradient problem [1]. An example of it can be seen in Figure 2.4.

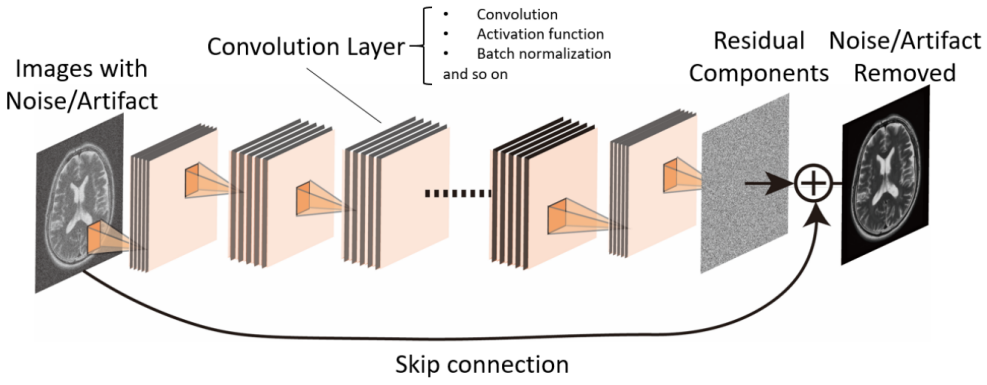


Figure 2.4: Example of a single-scale CNN structure consisting of multiple convolution layers. Noise or artifact reduced images are calculated by subtracting the output of the network from input images. Source [1]

The same approach has been used for segmentation by using the U-net shape [34], which has been used for this type of tasks on, for example Noise2Noise [35]. The model proposed in

[36] uses skip connection for T1-weighted (T1w) imaging of the brain. The developed network, trained with 528 T1w images, significantly improved the image quality based on PSNR analysis in comparison with DnCNN.

Another case that uses this architecture is found in [5]. In this article, they used MRI (T1w) data from 1113 healthy people. The images were previously normalized. The autoencoder was built using a configuration of 104 → 100 → 75 → 100 → 104 (input data → h1 layer → z-layer → h2 layer → reconstruction). To improve the generalization of the model and avoid overfitting, an L2 regularization (regularization parameter = 1×10^{-3}) was applied that penalized high values in the network's weights and facilitated diffuse weight vectors as solutions. To mitigate the network's internal covariate shift, the hidden layers were formed using scaled exponential linear units (SELUs), which provide a faster and more robust training, that is, less training epochs to reach convergence and a strong regularization scheme [37]. The model was trained with 2000 epochs.

A slightly different approach was made in [38], where the reconstruction problem was turned into an inpaiting task using a Context Autoencoder (figure 2.5). In that case, the model is trained to recover missing sections in healthy training images.

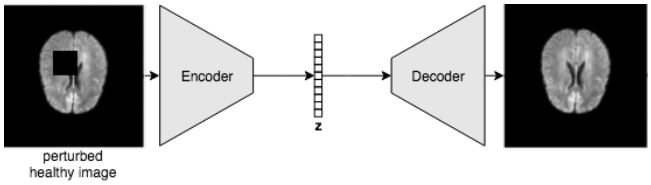


Figure 2.5: Schema of a context autoencoder applied to MRI. Source [10]

The natural choice for the shape of z, here also referred to as latent space, bottleneck or manifold, is a 1D vector. However, it has been shown that spatial AEs with a tensor-shaped bottleneck can be beneficial for high-resolution brain MRI as they preserve spatial context and can generate higher quality reconstructions [29].

Following this line, there is also the spatial alternative (see Figure 2.6) as presented in [29] which uses the idea from [39, 40] to make the architecture fully convolutional in order to ensure that spatial information is not lost in the bottleneck of the model. Notably, this heavily increases the dimensionality of z ($16 \times 16 \times 64$). They first trained normal convolutional AE with a dense latent space of dimensionality 512 and found that, besides not being capable of reconstructing brain lesions, they also lack the capability to reconstruct fine details such as the brain convolutions. By utilizing spatial AEs with sufficient bottleneck resolution, this problem

is mitigated. As postprocessing, they apply a $5 \times 5 \times 5$ median filter to each reconstructed subvolumes to filter out small residuals, usually belonging to brain convolutions.

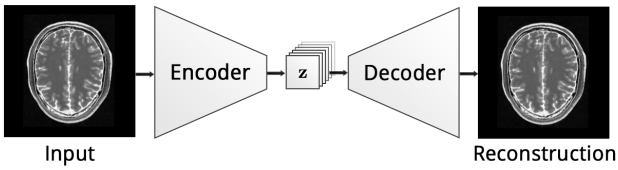


Figure 2.6: Schema of a spatial autoencoder applied to MRI. Source [29]

2.3.2 Generative models

In the last few years, deep learning based generative models have gained more and more interest due to (and implying) some amazing improvements in the field. Relying on huge amount of data, well-designed networks architectures and smart training techniques, deep generative models have shown an incredible ability to produce highly realistic pieces of content of various kind, such as images, texts and sounds. Among these deep generative models, two major families stand out and deserve a special attention: Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs).

2.3.2.1 Variatonal autoencoders

Variational autoencoders [41] (VAE) are basically autoencoders whose encodings distribution are regularised during the training in order to ensure that their latent space has good properties, allowing to generate some new data. Moreover, the term "variational" comes from the close relation there is between the regularisation and the variational inference method in statistics.

Just as a standard autoencoder, a variational autoencoder is an architecture composed of both an encoder and a decoder. Both are trained to minimise the reconstruction error between the encoded-decoded data and the initial data. However, in order to introduce some regularisation of the latent space, it is proceeded to a slight modification of the encoding-decoding process: instead of encoding an input as a single point, we encode it as a distribution over the latent space. So instead of finding z, we are finding $q(z)$ which tells us the PDF of z. In practice, the VAE projects input data onto a learned mean μ and variance σ, from which a sample is drawn and then reconstructed [10]. A simple architecture of a variational autoencoders is found on Figure 2.7.

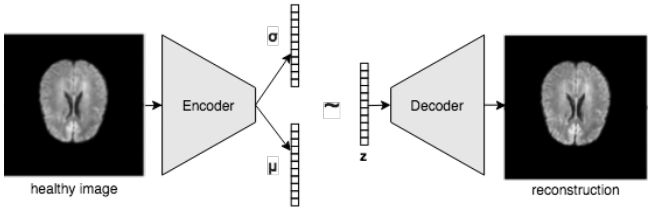


Figure 2.7: Schema of a variational autoencoder applied to MRI. Source [10]

Due to their ability to model the underlying distribution of high dimensional data, these frameworks are naturally suited for modeling the desired normative distribution. Further, their probabilistic nature facilitates the development of principled density-based anomaly detection methods. Consequently, they have been widely employed for outlier-based anomaly detection: VAEs were used in brain MRI for MS lesion [29], or for tumor detection in head CT [42] from aggregate means of Monte-Carlo reconstructions.

A good approach of using VAE for MRI reconstruction can be found in [29]. In that article, the author used a z latent representation of length 512. It introduces the spatial VAE (see Figure 2.8) with a latent space dimension of $16 \times 16 \times 64$. In both cases, images of 256×256 pixels are used as input and the model is trained during 150 epoch in minibatches of size 8, using a learning rate of 0.001. The spatial version, again, achieved much better results.

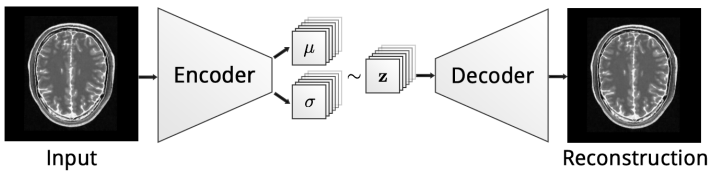


Figure 2.8: Schema of a spatial variational autoencoder applied to MRI. Source [29]

In [43] a 3D VAE is used too, and conditioned (CVAE) by adding an extra prior semantic information about the data (e.g., a label). This way the z-space distribution is learned per condition and more control over the learned space and reconstruction of images is possible.

2.3.2.2 Generative Adversial Networks

Generative adversarial networks (GAN) [44] approach is a promising learning technique for removing noise and artifacts, image inpainting and image reconstruction. GAN consists of two separate networks: generator and discriminator. Generator is the network that creates the output images (in our case would be the pathology free brain MRI versions) whereas discriminator acts as a classifier to determine whether the generated output is real or fake.

That is, the purpose of the generator is to create realistic images, while the discrimintaor plays an adversial role, discriminating between the images generated from the generator, and the real image sampled from the data distribution. Feedback from the discriminator contributes to updating the network of the generator to create convincing fake images to fool the discriminator. GANs don't work with any explicit density function. Instead, they take the game-theoretic approach: learning to generate from training distribution through 2-player game. A schema of a GAN model is found on Figure 2.9.

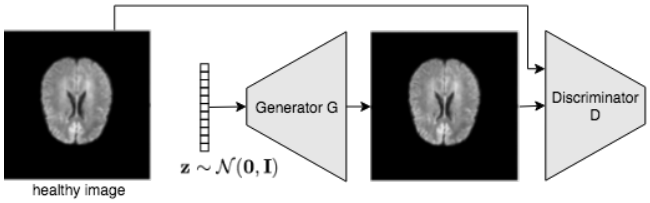


Figure 2.9: Schema of a GAN applied to MRI. Source [10]

With some user interaction, GANs have been applied in interactive image editing [45]. However, GANs can not be directly applied to the inpainting task, because they produce an entirely unrelated image with high probability, unless constrained by the provided corrupted image [46].

GAN has inspired recent works [47, 48] to formulate inpainting as a conditional image generation problem. Besides, there are widely variety of medical applications that are focused on GAN such as [49, 50]. Specially a good approach is found in [51] where a coarse-to-fine generative image inpainting framework network is introduced. In there, they proved that the contextual attention module significantly improves image inpainting results by learning feature representations for explicitly matching and attending to relevant background patches. A similar idea is found in [52]. To obtain a better loss function and more stable training using GAN, Wasserstein GAN (WGAN) was introduced by [53].

In [54], the author used GAN to detect tumours with only healthy data used in training (a small dataset of 20 patients from The Cancer Imaging Archive). Something similar is found in [55, 49], which aim is to reconstruct sequential healthy MRI slices from the previous ones.

It is important to highlight AnoGAN [56], a deep convolutional generative adversarial network to learn a manifold of normal anatomical variability, accompanying a novel anomaly scoring scheme based on the mapping from image space to a latent space. That is, a GAN is trained on healthy retinal optical coherence tomography (OCT) images, and later for a given

test image, determined the corresponding "healthy" image by performing gradient descent in the latent space. The difference between reconstructed and original image is then used to define an abnormality score for the entire image and the pixel-wise difference is used for detecting the abnormal areas in the images. In a recent follow-up work, an improved version of it (f-AnoGAN) was released [57] by replacing the costly iterative restoration method by a single forward pass through the network.

A full review of other medical applications of GANs can be found in [58].

2.3.3 Combinations

As mentioned before, a major advantage of AEs is their ability to reconstruct images with fairly high resolution thanks to a supervised training signal coming from the reconstruction objective. Unfortunately, they suffer from memorization and tend to produce blurry images. GANs have shown to produce very sharp images due to adversarial training, however the training is very unstable and the generative process is prone to collapse to a few single samples. The recent formulation of VAEs has also shown that AEs can be turned into generative models which can mimic data distributions, and both concepts have also been combined into the VAEGAN [59], yielding a framework with the best of both worlds.

Inspired by both generative models, [29] leveraged the AnoVAEGAN (see Figure 2.10), a combination of the GAN and VAE, to overcome the training instabilities of the GAN and to allow for faster feed-forward inference, which they successfully employed for anomaly segmentation in brain MRI, giving better result than either AnoGAN or VAE based applications.

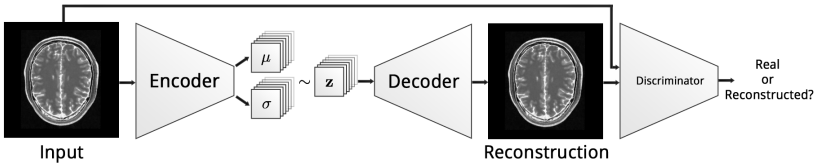


Figure 2.10: An overview of AnoVAEGAN applied to MRI. Source [29]

Chapter 3

Methodology

Since the goal of our project is to design an architecture able to obtain the healthy version of the brain MRI used as input, we have defined the following methodology for the sake of obtaining a workbench that let us design and compare the results of a bunch of models under the same constraints. Therefore, at the end of our experiments, we will be able to determine which of the architectures designed is the best.

In the following sections, the methodology used to carry out the project is explained in order to let the reader understand which data has been used, what decisions have been taken related to how the models are trained, how the models will be evaluated, and finally, all the architectures and models tested in the project.

It is important to mention that the detection of brain pathologies is out of the scope of this project. That is, our model must be able to reconstruct the healthy version of the brain used as input but it will not have any notion about if there is a pathology in it or not. Despite, this information could be obtained by subtracting the output and the input images and see which regions are different.

The first decision that was to be taken was which learning technique would be used to train our models. Two case scenarios were possible: supervised or unsupervised learning. If the former option was chosen, then a huge dataset containing the vast majority of brain anomalies with their respective healthy version of all of them would have to be used to train the models. The main drawback, apart from finding such a rich dataset, is that our model would only be able to reconstruct those anomalies in the dataset. However, if the latter option was chosen we would need to find a dataset with healthy brains and train our model to make it learn their hidden patterns: those that characterize the healthy brains. Then, because of this knowledge,

our model would be able to reconstruct any kind of anomaly. Using this technique, the model learns how to transfer healthy patterns to the output independently of the anomaly. That is because it has not been trained to know how to reconstruct each type of anomaly but to transfer healthy patterns to the output. For that reason, the first decision that has been taken (and which the following sections are directly affected by) is that unsupervised learning would be used.

3.1 MRI acquisition

As explained beforehand, a dataset with healthy brain MRIs was needed. These types of datasets usually uses NifTI format to store the volumetric brain MRI scans of the subjects.

Neuroimaging Informatics Technology Initiative[1] (NifTI) is an open file format released on 2nd September 2003 and is commonly used to store brain imaging data obtained using MRI methods. It is considered a standard to store 3D volumes of brain MRI due to it is composed of, apart from the brain slices, a header with relevant metadata such as orientation, image dimensions, data type, acquisition information, etc.

To work with NifTI format the NiBabel[2] library for Python has been used in this project.

3.1.1 IXI Dataset

The IXI Dataset[3] is a collection of nearly 600 MRI from normal, healthy subjects. For each subject, the dataset includes:

- T1, T2 and PD-weighted images.
- MRA images.
- Diffusion-weighted images (15 directions).

The images have been collected at three hospitals in London:

- Hammersmith Hospital using a Philips 3T system.
- Guy's Hospital using a Philips 1.5T system.
- Institute of Psychiatry using a GE 1.5T system.

[1] https://nifti.nimh.nih.gov
[2] https://nipy.org/nibabel/
[3] http://brain-development.org/ixi-dataset/

Moreover, the dataset contains a spreadsheet with demographic information about the subjects, but it has been not taken into account as it was not relevant for the project. Finally, the data was used due to is available under the Creative Commons CC BY-SA 3.0 license[4].

For the implementation of the project, only T1w images have been considered. So, from now on, all the information that refers to the dataset will be only related to this subset.

3.1.1.1 Analysis and exploration

The dataset is composed of 581 different healthy subjects, each stored in the beforehand mentioned NifTI format. The only relevant information for the project that can be extracted from the header is the 3D volume's spatial dimensions. Each slice of the 3D volumes has a width and height of 256px. No dimension equality is observed in regards to the number of slices (depth) each 3D is composed of. More information is given in Table 3.1

Width	Height	Depth	Number of subjects
256	256	130	2
256	256	140	2
256	256	146	74
256	256	150	503

Table 3.1: Dimension distribution of the IXI Dataset 3D volumes.

A sample of how the slices looks like can be seen in Figure 3.1. As it can be observed, the images are rotated 90° clockwise. Finally, as it has been checked, all images have been taken from a sagittal plane.

3.2 Workbench

A workbench has been designed to reach a consensus in order to determine that all our models are examined under the same conditions.

To carry out our experiment the following software stack has been used:

- Python 3.7.6
- Tensorflow[5] 2.1.0

[4] https://creativecommons.org/licenses/by-sa/3.0/legalcode
[5] https://www.tensorflow.org

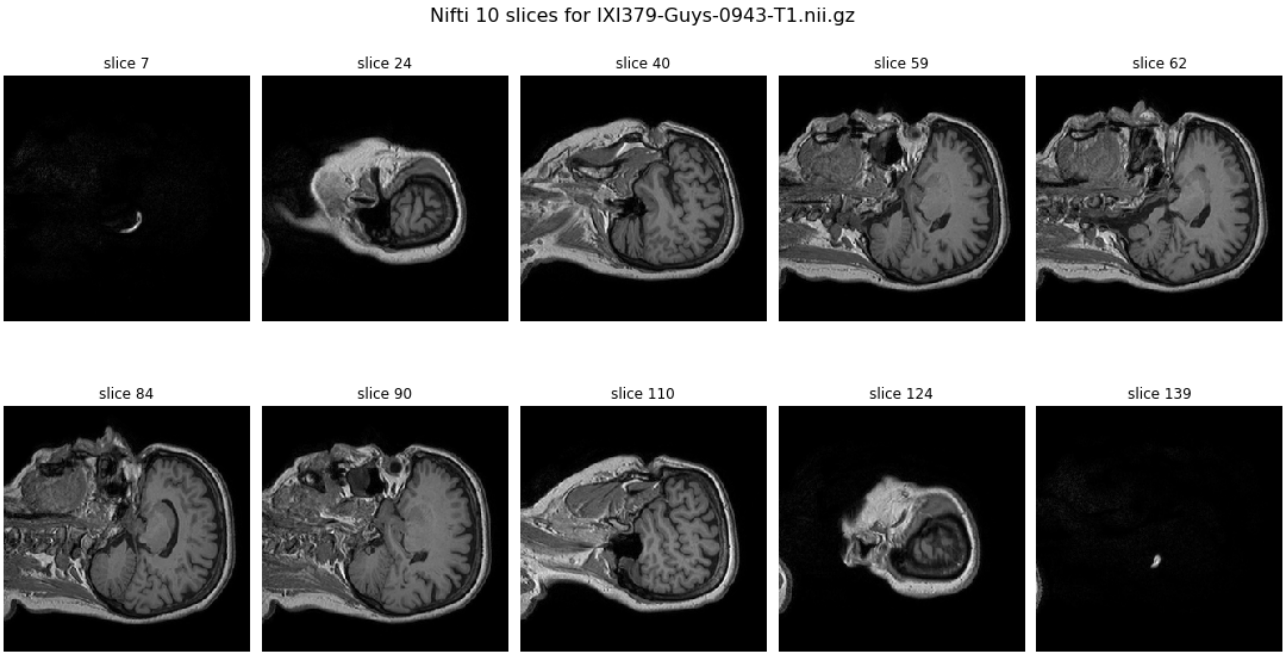


Figure 3.1: 10 samples from IXI379-Guys-0943-T1 NifTI image.

All the models have been implemented using Keras, which adds a level of abstraction on top of Tensorflow, making easier to work with neural networks by thinking only about layers in a sequential mode. That is, Keras uses Tensorflow as its backend. For this reason, in the following sections, there are mentions of Keras functions.

3.2.1 Training

As said before, the need for a workbench is crucial so that all models are affected by the same decisions made. Regarding the training phase, there are some considerations that need to be taken. It has been used some callbacks Keras offer to automatize, and to get rid of, the decision of specifying hyperparameters which, in the end, the optimum ones would be different for every model, as explained in the following paragraphs.

The first of them is the number of epochs that our models will be trained on. After doing tests with baseline models it was proven that 20 epochs (see Appendix A) were enough to reach stability in the validation loss. For that reason, and with the intention of giving some margin, all models have been trained using 30 epochs, as maximum. Due to the training process takes a long time and in order to streamline it, it has been taken into advantage of the *EarlyStopping* callback Keras provide. Thanks to it, the training process will stop once no improvement is detected in the validation loss. It has been configured with *patience* (number of epochs with

no improvement after which training will be stopped) of 3.

Another decision that has been delegated to a callback is the selection of the learning rate needed when compiling the model with an optimizer (see subsection 3.2.1.3). For the same reason explained beforehand, it has taken advantage of the *ReduceLROnPlateau* callback, which reduces the learning rate when a metric has stopped improving (validation loss in our case). It starts by using the default learning rate of the optimizer. It has been configured with *patience* (number of epochs with no improvement after which learning rate will be reduced) of 2 and a *factor* ($new\_lr = lr * factor$) of 0.2.

Finally, *CSVLogger* and *ModelCheckpoint* have been used in order to, respectively, stream epoch results to a CSV file and save the Keras model weights at some frequency. Thus, being able to save the results and weights at each epoch and use those values in the future to perform the evaluations. Moreover, all randomness' *seed* have been set in order to use the same randomness state and guarantee the reproducibility.

3.2.1.1 Data

Data processing In order to process our dataset accordingly, the considerations revealed in Section 3.1.1.1 must be taken into account.

The first thing that must be considered is the format of our dataset's images: NifTI. To keep the workflow easy, and in order to take advantage of the Keras' native classes that let the user work with images, a transformation needs to be applied. By using the med2image[6] tool, which converts medical images (such as NifTI) to a more displayable format, the images were transformed into PNG. By doing so, our dataset was now a 2D dataset in a more common format. It is important to mention that this tool adjusts the rotation automatically. So, from now on, the images are in PNG format and rotated by 90° counterclockwise in respect is observed in Figure 3.1.

Another aspect to be discussed is the number of slices that are considered. As it can be seen in Figure 3.1 not all slices contain interesting information. That is, the slices on the extremes barely show part of the subject's brain. So then, if all images were considered for training, these latter would penalize how our models are trying to figure out how to detect the hidden patterns of the healthy brains. It makes sense then to discard those images that are not interesting for our case of study. Taking into account that not all images contain the same

[6] https://github.com/FNNDSC/med2image

Number slices (depth)	Min slice	Max slice	Number of images
130	35	95	122
140	38	102	130
146	40	106	4958
150	41	109	34707

Table 3.2: Characteristics of slice selection depending on the original volumetric depth.

number of slices (as shown in Table 3.1) a fixed number cannot determined. Using a percentage is more reasonable in that case. After exploring the dataset, only the 45% of the middle slices are considered (see Table 3.2 for more information). Once applied this filtering, only 39917 images remain to be used. Finally, it is important to mention that, for the sake of speeding up training and as it has not shown a considerable reduction in terms of quality, the image size chosen to work with is 224x224px (see Appendix B).

Data augmentation pipeline Ideally, our models should preserve the healthy patterns while modifying those that are not to make them look as if they were. Data augmentation is then essential to avoid the convergence of our models towards the identity operator, that is avoid them from simply copying the input directly into the output. The intuition behind it is that occluding areas from the training images is a data augmentation procedure that helps to improve the network training [60, 61]. Apart from that, our model will be then more tolerant of unexpected changes as in real world applications.

Albeit Keras provides a class for augmenting data directly on the fly (*ImageDataGenerator*), the transformations it is able to carry out are not sufficient for what it is looked for. However, it let use a preprocessing function which is applied to each input after each image is resized and augmented using the arguments this class accepts. So then, all augmentation procedure has been delegated to a sequential pipeline that has been used as the preprocessing function, and none of the class arguments related to augmentation have been set excepting those regarding scaling. So the only function performed by this class is the downscaling from $[0, 255]$ to $[0, 1]$ due to original value would be too high for our models to process (given a typical learning rate) [62].

To perform the augmentation the *imgaug v0.4.0*[7] library has been used as it supports a wide range of augmentation techniques, allows to easily combine these and to execute them in random order (if needed). The transformations that were interesting for the project and that has been applied are:

[7] https://imgaug.readthedocs.io

- *AdditiveGaussianNoise* with a scale between $[0.04, 0.3]$.
- *SaltAndPepper* with a pixel replacement probability of 0.01.
- *CoarseDropout* with a pixel probability of being dropped from the interval $[0.02, 0.1]$.
- *GaussianBlur* with a standard deviation of the gaussian kernel from the interval $[0.1, 2.0]$.

It's important to mention that not all them are applied together. Instead the following schema is followed:

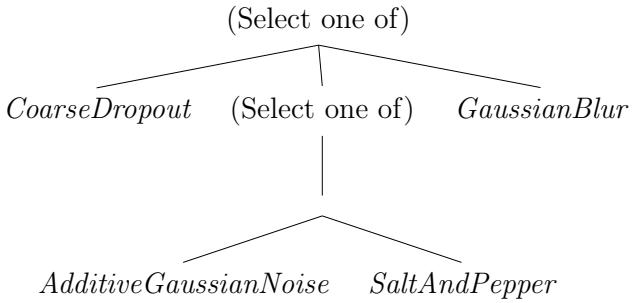


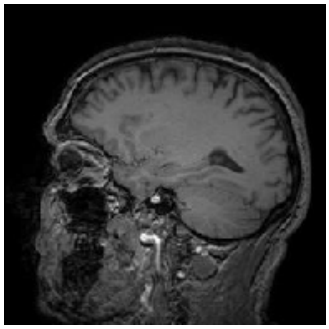
Figure 3.2: The sequential pipeline's choosing rules for applying augmentation.

Thus, it can be guaranteed that all images are applied at least one transformation either blurriness, dropout, or noise. An example of some data augmentation given an image is shown in Figure 3.3.

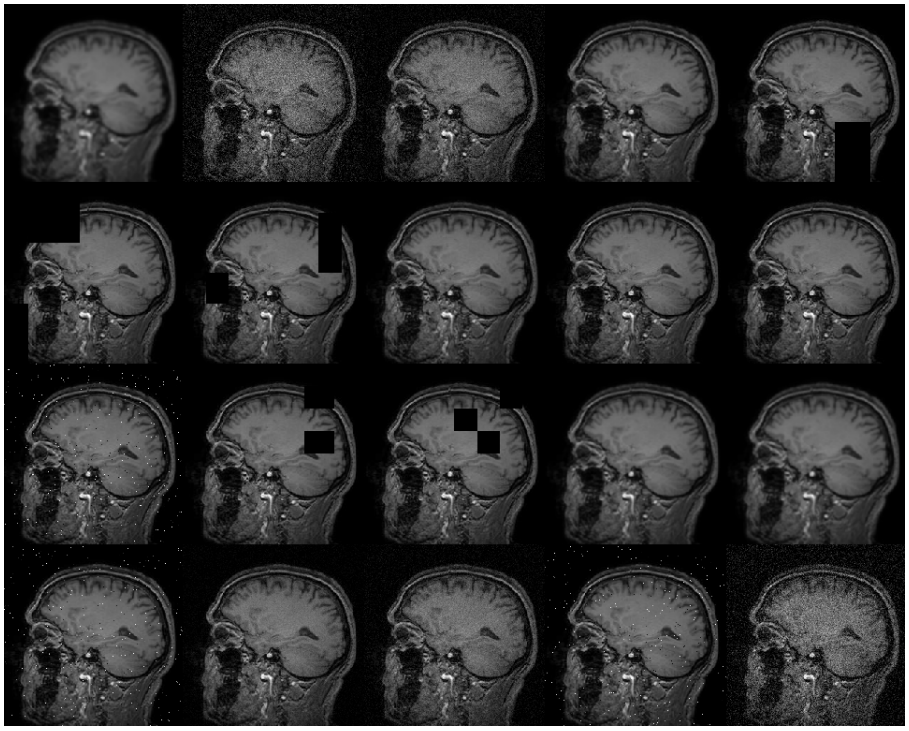
Finally, even though data augmentation is needed for the input of our models, the original ones must be used to compare them with the output. For that reason, two *ImageDataGenerator* are created: one that applies the preprocessing function mentioned while the other does not. However, both are configured to apply downscaling to images.

Data split In order to train our models properly, the data was needed to be split into train, validation, and test sets. The commonly used 80/20 ratio has been used for dividing train and test sets. Inside the train set, and another split has been done to obtain the validation set.

It has to be taken into account that will be similarities between the slice's images of the same subject. For that reason, slice's images of the same subject can not be shared among the different subsets. That is, all the slices from the same subject must be only in one of the 3 subsets and it has to be guaranteed that images in the test set have not been observed before (during the training phase) and, analogously with the validation set. In the table 3.3 one can observe the division of samples on each set.



(a) Original image



(b) Results of applying 20 times the augmentation pipeline.

Figure 3.3: Example of data augmentation.

Subset name	Split percentage	Number unique subjects	Number of images
Train	66%	383	26305
Validation	14%	81	5573
Test	20%	116	8039

Table 3.3: Subset divisions.

3.2.1.2 Loss function

Loss function is a crucial decision when talking about neural networks. It is a method of evaluating how well a given algorithm models the dataset. That is, if the prediction are totally off, the loss function will output a high number. If they are pretty good it will output a lower number. In our case, the loss function has to quantify how similar are the image given by the model (its input was an augmented image such one of Figure 3.3b) with the original one.

The most popular loss functions used in the field of image processing are: Mean Squared Error (MSE) and Structural similarity (SSIM). We choose these among the plethora of existing indexes, because they are established measures, and because they are differentiable (a requirement for the backpropagation stage).

Mean Squared Error In the literature reviewed, the use of MSE loss has been criticized for its trend to produce blurry output images [63, 64]. It is widely accepted that MSE, and consequently the Peak Singal-to-Noise Ratio (PSNR), do not correlate well with the human's perception of image quality [65]. That is, MSE simply does not capture the intricate characteristics of the human visual system.

Structural similarity The structural similarity [66] is a distance measure designed to capture perceptual similarity that is less sensitive to edge alignment and gives importance to salient differences between input and reconstruction. It has been considered as a loss function for image reconstruction, motivated by its ability to produce well-looking images from a human perceptual perspective [63, 67]. The SSIM has shown some improvement over the MSE loss for the training of an autoencoder in the context of anomaly detection. However, the SSIM loss formulation does not generalize to color images and is parametric. As a matter of fact, to use SSIM as a loss function it is needed to subtract 1 to it. That is because by definition the higher value it takes is the better are the two images compared. But, loss function has to be minimized, not maximized.

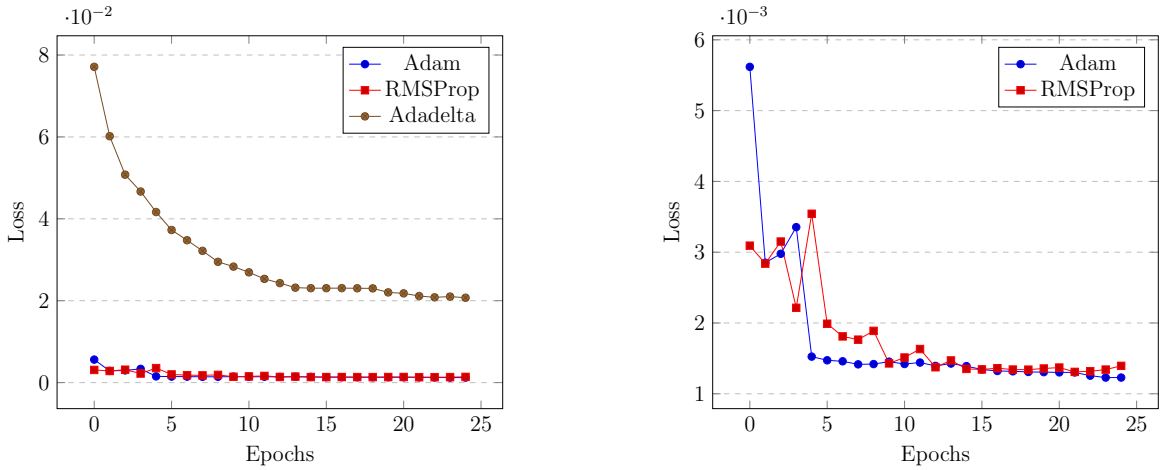
Due to there are benefit and drawbacks for both loss functions, it has been considered not to choose only one and use both of them to prove which works better for our case of study.

3.2.1.3 Optimizer

In machine learning applications, gradient based algorithms are popular for training neural networks. Recently, many improved algorithms based on Gradient Descent were presented to boost up the training process like Nesterov, Adagrad, Rmsprop, Adadelta and Adam. All of them are first-order optimization methods and they perform parameter updating during back-propagation to reduce loss of network [68]. If focusing only on those that has converged earlier in the literature we keep with: RMSProp [69], Adam [70] and Adadelta [71].

A test with a simple baseline model has been performed to see how they behave with our case of study. The encoder's model has been build using two blocks of *Conv2d* + *BatchNormalization* + *ReLU* + *MaxPooling2D*. While decoder's model has been build using two blocks of *Conv2d* + *BatchNormalization* + *ReLU* + *UpSampling2D*. The number of filters are 32 and 64 respectively for the encoder, and vice-versa for the decoder. MSE has been used as loss function.

As can be observed in the results on Figure 3.4, Adam is the optimizer that converges before. For that reason, it has been selected to train all our models.



(a) Comparison of all optimizers.

(b) Zoomed comparison.

Figure 3.4: Evolution of validation loss function using different optimizers.

3.2.2 Evaluation metrics

In order to determine how good are our models and compare them, a set of evaluation metrics need to be established.

Quality related metrics In this block, only image quality metrics are considered. The steps to evaluate the model using these metrics will be:

1. Select an augmented sample from the test set.
2. Use the previous sample as input for our model.
3. Compare the analogous original image (without the augmentation) used as the input with the given as result by the model.

PSNR The first that will be taken into account is the Peak signal-to-noise ratio (PSNR). It is quite a traditional estimator. By definition, PSNR shows a ratio between the maximum possible power of a signal and the power of corruption noise that affects the fidelity of its representation. At the end is a variation of MSE and concentrates on the pixel-by-pixel comparison. For that, it has the same drawbacks as MSE as explained in subsection 3.2.1.2. It is represented by the following formula:

$$PSNR = 20\log_{10}(MAX_I) - 10\log_{10}(MSE)$$

Here MAX_I is the maximum possible pixel value of the image. Being our image in PNG format (which uses 8 bits per sample), then this value is 255. For that reason, the maximum value

PSNR can take in our case is:

$$PSNR_{MAX} = 20\log_{10}(255) = 48.13dB$$

SSIM Other metrics that will be used is structural similarity [66] (SSIM) which is, in essence, the same as it has been explained in subsection 3.2.1.2. Basically, it is a perception-based model that considers image degradation as a perceived change in structural information, while also incorporating important perceptual phenomena, including both luminance masking and contrast masking terms. The difference with the other metrics, such as MSE or PSNR, is that these approaches estimate absolute errors. Its range is between $[0, 1]$, being 1 a perfect match between the image compared.

To make better decisions regarding behavior, the test set will be used to evaluate the model. So these metrics will be the ones that will let us understand how good the resulting images are. Therefore, descriptive statistics will be used. The measures of central tendency that will be used are the mean and the median, while measures of variability will be the standard deviation (SD) and the interquartile range (IQR); being the latter of both resilient to outliers.

Model related metrics Other metrics can be considered apart from those strictly related to the quality of the output images. Those metrics are the ones strictly related to the model itself:

- Training time
- Number of parameters

3.3 Network architectures and models

As explained at the beginning of this chapter, unsupervised learning will be used, so the model needs to be designed accordingly. Two different networks when speaking about unsupervised methods for image reconstruction (as seen in section 2.3) come to one's mind: autoencoders and Generative Adversarial Networks (GAN).

If GANs are known for their ability to produce realistic high-quality synthetic images [72], they have major drawbacks. Usually, GANs are difficult to train due to their trend to converge towards mode collapse [44]. In the context of anomaly detection (which uses image reconstruction in essence), some GAN-based solutions fail to exclude defective samples from the generative

distribution [73] and require an extra optimization step in the latent space during inference [56]. For all these reasons, it has been decided to set GAN aside and focus on autoencoders.

As explained in subsection 2.3.1, autoencoders are composed of two parts: the encoder and the decoder. The encoder is the one in charge of compressing the input image into a given latent space, while the decoder's goal is to reconstruct the image as good as possible given the information from the latent space. So then, the most arduous task falls to the encoder, which has to generate representation in the latent space good enough, but only with the information needed, to keep healthy patterns as were in the input and reconstruct those that are not. Therefore, in this chapter special focus will be on designing different architectures for the encoder part, while the decoder is the same for all of them.

All parts of the system have been designed having in mind the widely used block for convolution: *Convolution2D + Batch Normalization + ReLU*. It is important to mention that LeakyReLU [74] has been used as it has shown better results on CNN [75] and in image segmentation tasks [64]. So the **convolutional block** is defined with the following triad:

- *Convolution2D*
- *Batch Normalization*
- *LeakyReLU*

To evaluate all models under the same conditions the following consideration has been taken:

- Network depth = 4 (encoder = 3, latent space = 1)
- Initial number of convolution filters = 8. Increasing by multiplying in powers of 2. That is $[8, 16, 32, 64]$.
- Latent space dimension will be $(28, 28, 64)$.

See Table 3.4 for a visual reference on how the spatial dimension changes at every step of the architectures.

Decoder As said before, a single decoder has been designed and is the one that will be used for all models. A **decoder block** is defined as follows:

- 1 x *UpSampling2D*
- 1 x *Conv2D*

	Step	Dimension
	Input	(224, 224, 1)
Encoder	1st block	(224, 224, 8)
	2nd block	(112, 112, 16)
	3rd block	(56, 56, 32)
	Latent space	(28, 28, 64)
Decoder	1st block	(56, 56, 32)
	2nd block	(112, 112, 16)
	3rd block	(224, 224, 8)
	Output	(224, 224, 1)

Table 3.4: Spatial dimensions in every step of the architectures.

- 2 x convolutional block

The upsampling is performed using a factor of 2.

So that, the decoder has the following structure:

- 1 x Decoder block [filters=32]
- 1 x Decoder block [filters=16]
- 1 x Decoder block [filters=8]
- 1 x *Conv2D*(sigmod) [filters=1] to reconstruct the image.

3.3.1 Autoencoder (AE)

A simple autoencoder is defined to be used as baseline. A **simple encoder block** is then defined as:

- 2 x convolutional block
- 1 x *MaxPooling2D*

So then, the structue of the baseline autoencoder is:

- 1 x Simple encoder block [filters=8]
- 1 x Simple encoder block [filters=16]
- 1 x Simple encoder block [filters=32]
- 2 x convolutional block [filters=64] (latent space).

All convolution have a $3x3$ kernel, strided by a factor 2.

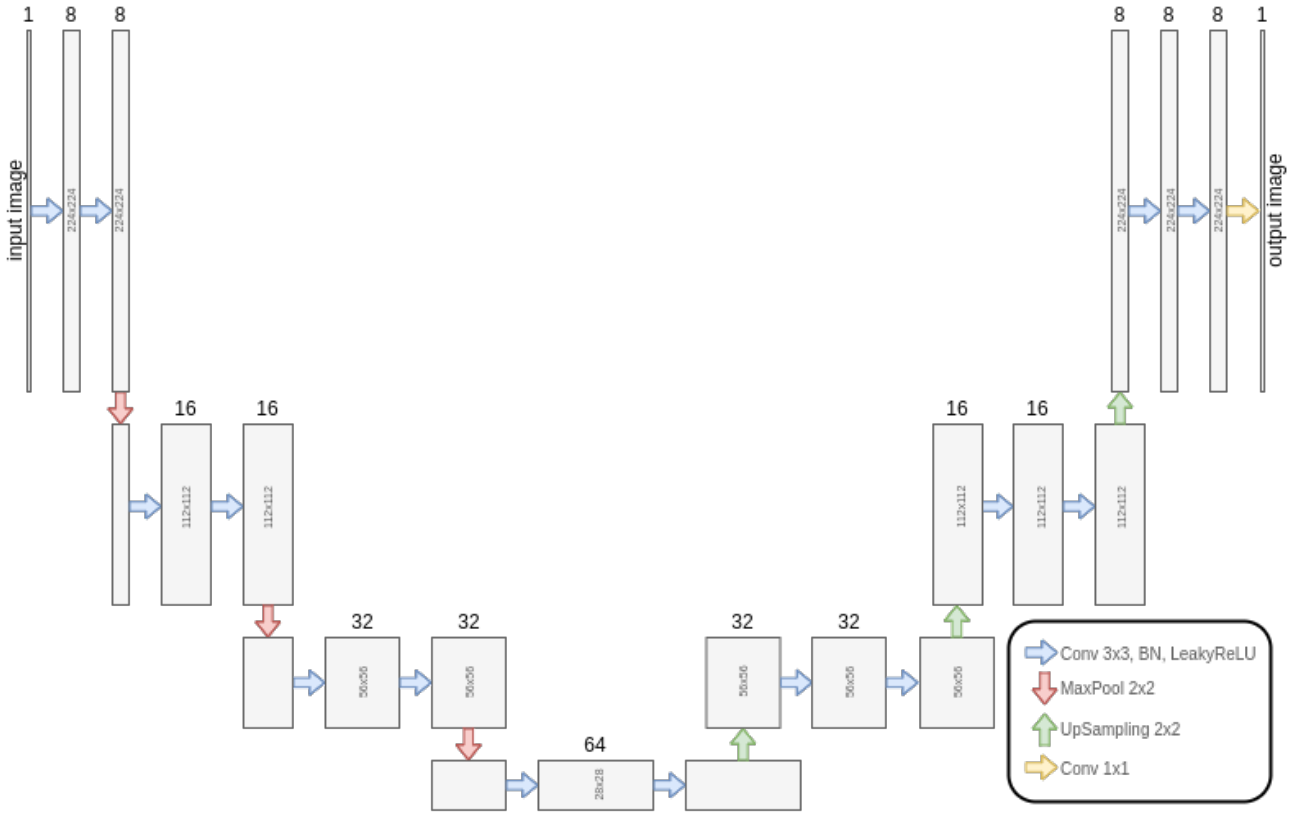


Figure 3.5: Architecture diagram of the baseline autoencoder.

3.3.2 AE + Skipping Long Connections (sLC)

To improve the supposed blurriness resulting from the baseline model and to enhance the sharpness of the reconstruction, it is considered an autoencoder equipped with skipped connections. The assumption here relies on that these connections will allow the high-frequency information to bypass the bottleneck [76].

Our design is a variant of U-Net [34] with the main difference that ours performs an addition as the merging strategy instead of concatenation (inspired by [76, 77]), which has be proved to achieve good results [78]. Besides, compared with element-wise addition, concatenation generates larger feature maps which would increase computational complexity.

Therefore, this model uses the same configuration as the baseline but adding 3 skipped long connections. That is, before every *MaxPooling2D* layer, a shortcut is added from that point into the analogous upsampling block. See the following schema to understand where the shortcuts are taken from in the encoder:

- 2 x convolutional block [filters=8] (naming output as ***shortcut_1***)

- 1 x *MaxPooling2D*
- 2 x convolutional block [filters=16] (naming output as ***shortcut_2***)
- 1 x *MaxPooling2D*
- 2 x convolutional block [filters=32] (naming output as ***shortcut_3***)
- 1 x *MaxPooling2D*
- 2 x convolutional block [filters=64]

And then the decoder is build as:

- (latent space)
- 1 x *UpSampling2D*
- 1 x *Conv2D* [filters=32]
- (Adding ***shortcut_3***)
- 2 x convolutional block [filters=32]
- 1 x *UpSampling2D*
- 1 x *Conv2D* [filters=16]
- (Adding ***shortcut_2***)
- 2 x convolutional block [filters=16]
- 1 x *UpSampling2D*
- 1 x *Conv2D* [filters=8]
- (Adding ***shortcut_1***)
- 2 x convolutional block [filters=8]
- 1 x *Conv2D*(sigmod) [filters=1] to reconstruct the image.

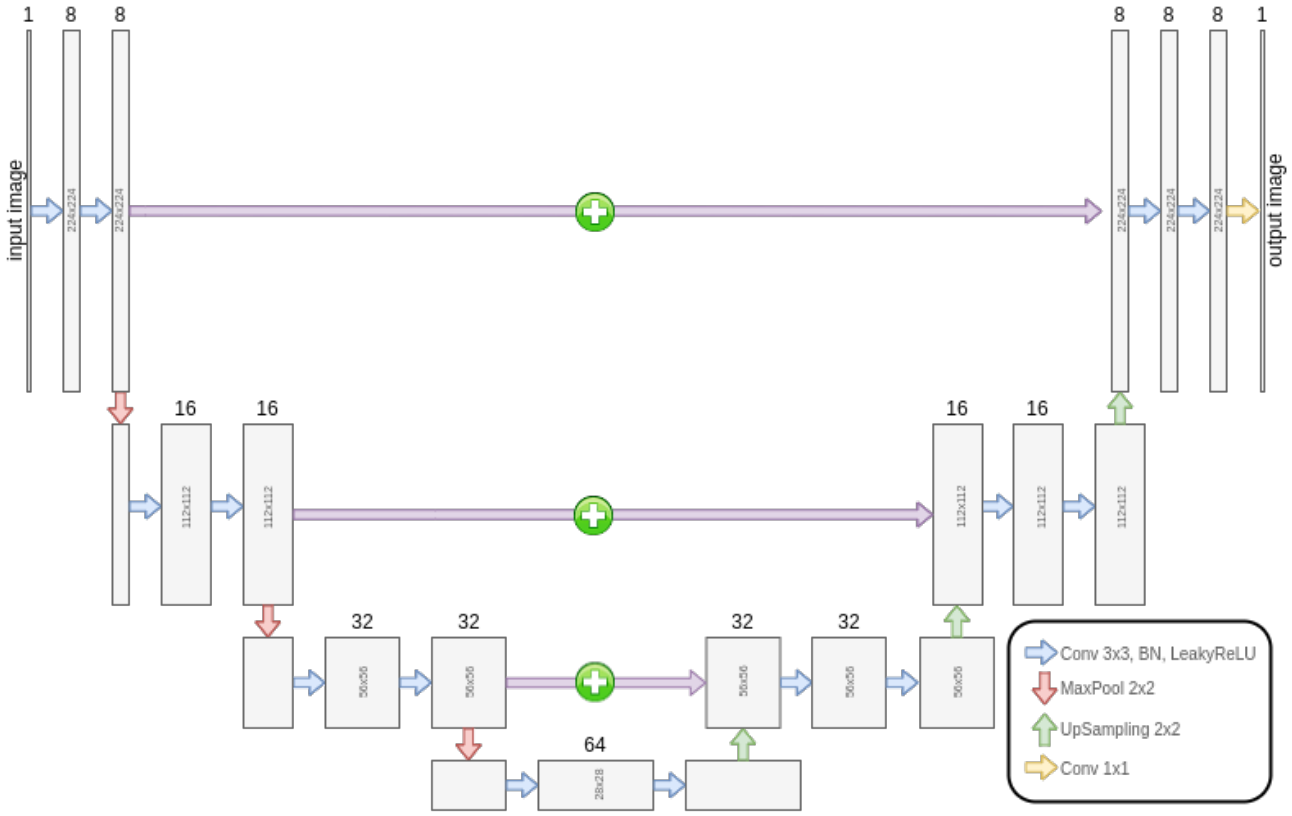


Figure 3.6: Architecture diagram of the autoencoder with skipped long connections.

3.3.3 AE + Residual block (RB)

A direct relationship that comes to mind when talking about skip connection is the Residual Block (see Figure 3.7) introduced in the Resnet architecture [22]. Residual Blocks are skip-connection blocks that learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. Formally, denoting the desired underlying mapping as $H(x)$, it is let the stacked nonlinear layers fit another mapping of $F(x) := H(x) - x$. The original mapping is recast into $F(x) + x$ [22]. The additional x acts like a residual, hence the name "residual block".

The intuition is that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers. Having skip connections allows the network to more easily learn identity-like mappings [22].

Depending on if the input of the block is downscaled or not the shortcut will be different. In case the input is not downscaled the structure is as follows (named **residual block / identity** from now on):

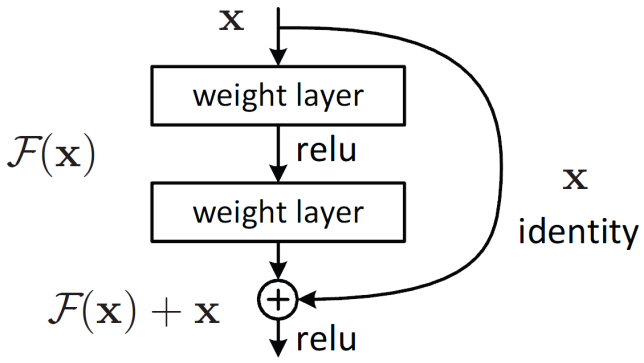


Figure 3.7: Residual block. Source [22]

- *input* (extracting ***shortcut***)
- 1 x convolutional block
- 1 x *Conv2D*
- 1 x *BatchNormalization*
- (Adding ***shortcut***)
- 1 x *LeakyReLU*

In case the input is downscaled the structure is the same as before with the difference that the shortcut is passed through a *Conv2D* + *BatchNormalization* before being added (named **residual block / conv** from now on).

So, a **residual block** of N blocks will always start with one **residual block / conv** followed by N-1 **residual block / identity**.

Motivated by [79, 80, 81] a convolutional layer is introduced at the very beginning. The encoder design is the following:

- 2 x convolutional block [filters=8]
- 1 x *MaxPooling2D*
- 1 x (*residual block / conv*) + 2 (*residual block / identity*) [filters=16]
- 1 x (*residual block / conv*) + 3 (*residual block / identity*) [filters=32]
- 1 x (*residual block / conv*) + 1 (*residual block / identity*) [filters=64]

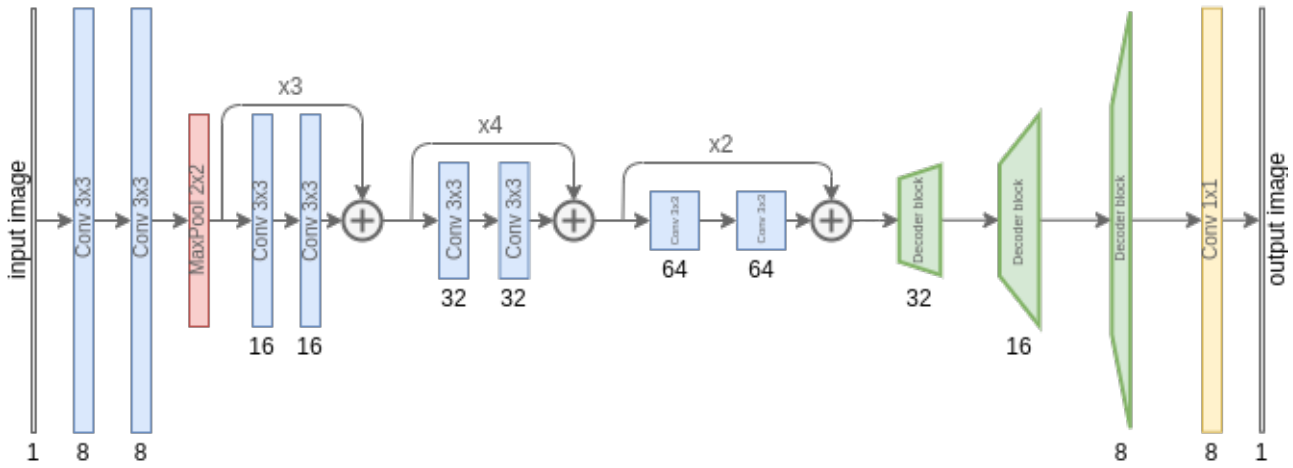


Figure 3.8: Architecture diagram of the autoencoder using residual blocks.

3.3.4 AE + RB + sLC

By mixing the advantages of short and long skipped connections the resulting model not only would have to perform better but to converge faster. This reasoning is widely supported by the image super-resolution literature [82, 83, 84, 85, 81, 76, 79], so it is expected to obtain the best results with this architecture.

3.3.4.1 First version

In this first version, inspired by [79, 86], it is supported that skipped long connection have to be placed only after the residual blocks, considering the output between the last (*residual block / identity*) and the first (*residual block / conv*) of the next block (see Figure 3.9). That is, in our case only 2 skipped connection are needed:

1. Between the first residual block ending (those with filters=16) and its analogous upsampling block.

2. Between the second residual block ending (those with filters=32) and its analogous upsampling block.

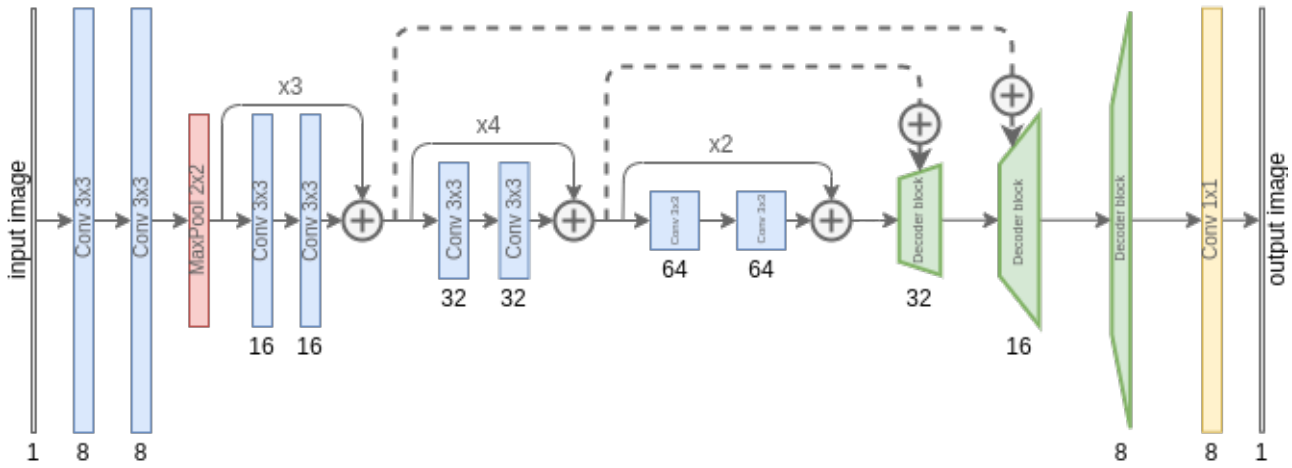


Figure 3.9: Architecture diagram of the first version of the autoencoder using residual blocks and skipped connections.

3.3.4.2 Second version

In the second version, motivated by [81, 82, 83, 84, 76], it is supported that and extra skipped long connection is needed after the very first downscaling, that is after the *MaxPooling2D* layer (see Figure 3.10). So then, the skipped connections are:

1. Between the MaxPooling2D layer and its analogous upsampling block.
2. Between the first residual block ending (those with filters=16) and its analogous upsampling block.
3. Between the second residual block ending (those with filters=32) and its analogous upsampling block.

Assuming that this extra connection will add high frequency information into the decoder it is expected to be the best of both versions.

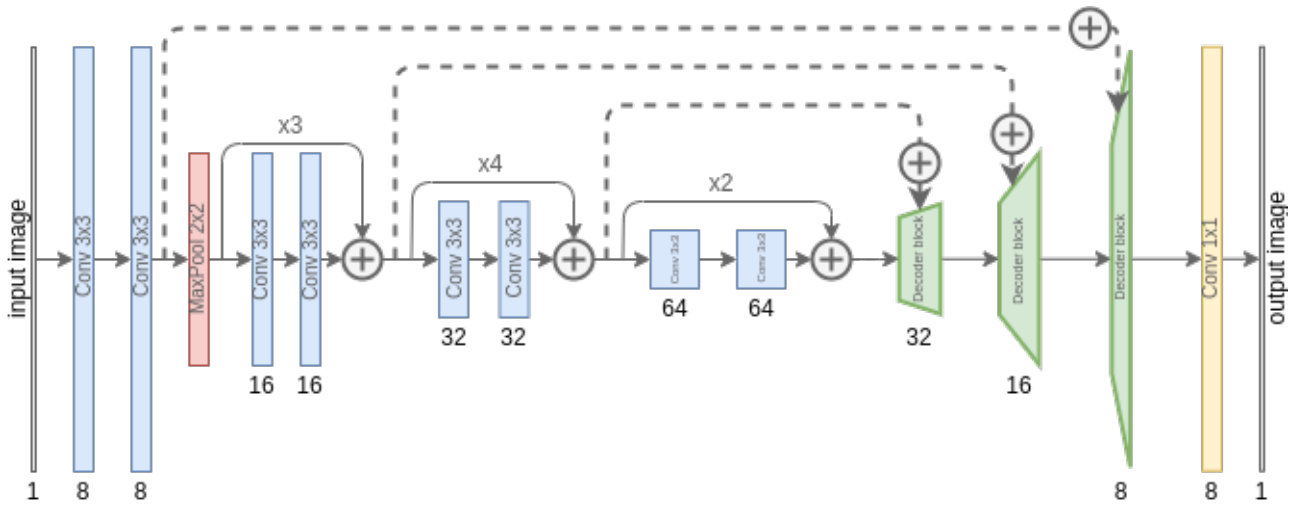


Figure 3.10: Architecture diagram of the second version of the autoencoder using residual blocks and skipped connections.

Chapter 4

Results

All the experiments carried on in this project has been developed using Tensorflow and Keras as explained in the methodology section. Since training neural network can take quite long, and taking advantages of the benefits of using GPU to speed up this process, all models have been trained on an NVIDIA GeForce GTX 1660 Ti Mobile (*CUDA v10.1* and *cuDNN v7.6*) using an Ubuntu 18.04 LTS with 16GB RAM.

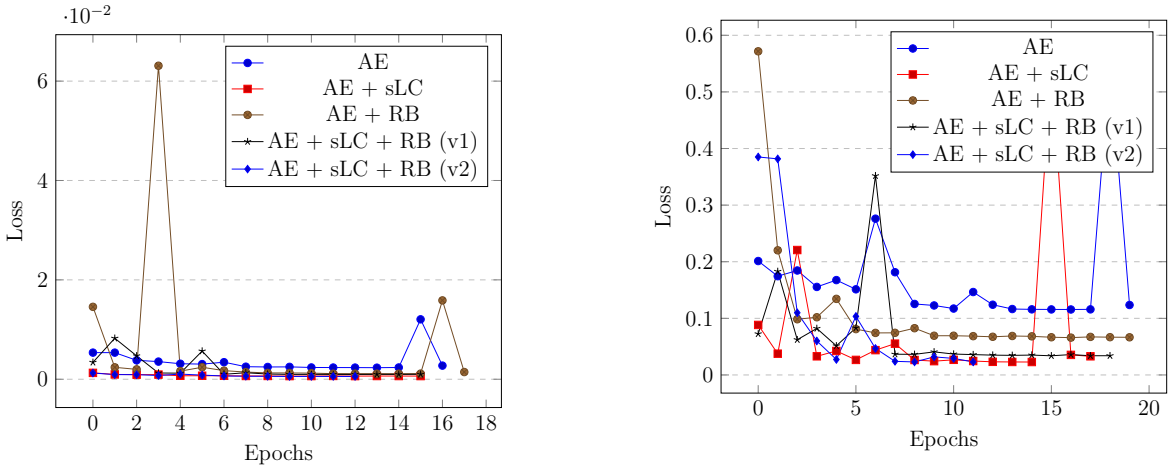
In the following sections, the reader will find the results of the architectures explained in the section 3.3 using the workbench specified in the section 3.2. In order not to overload the GPU, a *batch size* of 16 has been used to train and evaluate the models. The implementation of the code and the results are available at https://github.com/epou/msc_workbench

4.1 Training results

The first consideration of the models is to analyze their behavior during the training phase. That is, studying how the architectures are modeling the desired data in terms of the loss function.

Due to overfitting has not been detected during the process (see Appendix C), only the validation loss values have been plotted in order to make graphs less cluttered. As show in the Figure 4.1 both models *AE + sLC* and *AE + sLC + RB (v2)* are the ones which presents a faster convergence and lower values. It is important to remember that, thanks to the use of the *EarlyStopping* callback, not all models have been trained with a fixed number of epochs, but until they did not show improvement anymore. Same conclusions can be drawn by looking at Table 4.1 where the minimum validation loss among all epochs is shown.

As expected, the Autoencoder used as the baseline is the worst model. By adding the skipped long connections into the model a considerable improvement is reached. Besides, the effect of the residual blocks is translated into an improvement, but not as significant as the one introduced by the skipped long connections. When mixing skipped long and short connections, the resulting model have induced improvements with respect to the baseline model, but only the version 2 is the one that offers the best results when using MSE as the loss function. Otherwise, *AE + sLC* is the winner when using SSIM as loss function, although *AE + sLC + RB (v2)* offer very similar results.



(a) Using MSE as loss function.

(b) Using SSIM as loss function.

Figure 4.1: Evolution of validation loss using both loss functions.

Model name	**Min validation loss**	
	MSE	**SSIM**
AE	0.002324843	0.115646767
AE + sLC	0.000631142	**0.022841621**
AE + RB	0.001123002	0.065956026
AE + sLC + RB (v1)	0.000963927	0.033854349
AE + sLC + RB (v2)	**0.000559096**	0.0229807

Table 4.1: Minimum validation loss values results for both loss functions.

The elapsed time and the number of parameters for each model is an important consideration too. As it was supposed, performing an addition as the merging strategy for the skipped long connections does not alter the number of parameters needed but it does in the mean epoch training time as shown in Table 4.2. Moreover, it can be seen that using residual blocks implies an increase in the elapsed training time and an increase in the number of parameters to be trained too (needing 2.3 times more parameters when using residual blocks). Finally, using SSIM loss seems to be computationally more expensive than using MSE.

Model name	Epoch training time (sec)		Num parameters
	MSE	SSIM	
AE	160	173	109993
AE + sLC	164	177	109993
AE + RB	217	227	253401
AE + sLC + RB (v1)	218	229	253401
AE + sLC + RB (v2)	222	232	253401

Table 4.2: Comparative of the epoch training time and the number of parameters for each model.

4.2 Image results

In order to evaluate the performance of our models, the test set has been used to determine how good the output images of our models are for each loss function. That is, the results shown in the following subsections are extracted by only using the test images, the ones that our models have not seen during the training. The models used in this section has been built by restoring the weights that led to the best validation loss as seen in Table 4.1, not the ones obtained in the last epoch.

4.2.1 Quantitative

As explained in subsection 3.2.2 the two evaluation metrics used to compare how good are the output images are PSNR and SSIM. Besides, it must be taken into consideration that two loss functions have been used to train all the models: MSE and SSIM.

MSE as the loss function In this case scenario the model that has given the best results is the *AE + sLC + RB (v2)*, outperforming in both PSNR and SSIM metrics.

Focusing on the PSNR (see Table 4.3), it can be seen that the baseline model is the worst of all of them. As it can be observed, by adding skipped long connection to it, an improvement of 8.58dB is obtained. An increase is also observed when using residual blocks, but not as substantial as in the previous case, being this time an extra 3.5dB improvement. Finally, by mixing both cases an enhancement is seen too, being the *v2* the best of all, with an average PSNR of 36.11dB, that is a difference of 9.4dB in respect to the baseline. Apart from being the one with the best average result, it can be observed that both the minimum and maximum results are greater than the others, although both standard deviation and IQR is worse than the model with only skipped long connections. The same behavior is observed with the Q1, median and Q3 values.

Regarding the SSIM (see Table 4.4), a similar behavior with the PSNR table is seen. The baseline model continues being the worst model, and the introduction of skipped long connections is translated into the most relevant improvement, reaching an increase of 0.129. Using only residual blocks also implies an enhancement with respect to the baseline of 0.1. By skipping both long and short connections an improvement is observed too with the baseline. The main difference is that in this case, only the *v2* showed better results with respect to the *AE + RB*. In this case, the minimum and maximum values of this version offer the best results too. Focusing on the Q1, median, Q3 and IQR the same behavior is seen. Finally, for both metrics, the standard deviation and the IQR presents higher values for *AE + sLC + RB (v2)* in respect to *AE + sLC*.

Model name	min	max	mean	standard deviation	Q1	median	Q3	IQR
AE	16.909	33.543	26.707	1.887	25.55	26.36	27.748	2.196
AE + sLC	19.689	44.19	35.293	5.649	30.335	35.717	41.34	11.004
AE + RB	20.183	37.246	30.21	2.351	28.957	30.164	31.520	2.56
AE + sLC + RB (v1)	20.406	39.249	31.25	2.929	29.227	31.728	33.254	4.028
AE + sLC + RB (v2)	19.72	45.398	**36.11**	5.97	30.65	**36.845**	42.36	11.709

Table 4.3: PSNR values when training models using MSE as the loss function.

Model name	min	max	mean	standard deviation	Q1	median	Q3	IQR
AE	0.597	0.888	0.808	0.033	0.789	0.809	0.827	0.0376
AE + sLC	0.713	0.9847	0.9375	0.039	0.920	0.950	0.9655	0.045
AE + RB	0.708	0.9557	0.9085	0.026	0.898	0.913	0.925	0.0267
AE + sLC + RB (v1)	0.734	0.963	0.904	0.034	0.891	0.909	0.929	0.038
AE + sLC + RB (v2)	0.735	0.989	**0.944**	0.039	0.9245	**0.9595**	0.972	0.047

Table 4.4: SSIM values when training models using MSE as the loss function.

SSIM as the loss function A noticeable increase in the results of both metrics for almost all models (in exception of *AR + RB*) is perceived when using SSIM as the loss function, as can be observed in Figure 4.2 and Figure 4.3. The model that has given the best results continues being *AE + sLC + RB (v2)* with a PSNR improvement of 10dB and a SSIM increase of 0.0934 in respect to the baseline. Comparing the two best models, both the standard deviation and the IQR has been reduced from the *AE + sLC + RB (v2)* in respect to *AE + sLC*.

Focusing on the PSNR (see Table 4.5) one can observe that by introducing skipped long connections the average result is increased substantially. The same happens when using residual blocks, but the enhancement is not so substantial. An increase of 9.74dB is observed for the *AE*

+ sLC in respect to the baseline. Only an enhancement of 1.56dB occurs when using residual blocks. When skipping long connections and using residual blocks together an increase of 10dB is observed with respect to the baseline with a narrower standard deviation and IQR, becoming then the best of all of them.

As can be derived from Table 4.6, using SSIM as the loss function provides better results for the SSIM metric than using MSE function (see Figure 4.2 and Figure 4.3 too). The model ranking regarding their results is same as the other tables, being skipping long connections a more relevant addition in terms of improvement than skipping short connections only. Then *AE + sLC + RB (v2)* provides the best results, with a difference of 0.0934 in SSIM in respect to the baseline and a narrower standard deviation and IQR than the *AE + sLC* model.

Model name	min	max	mean	standard deviation	Q1	median	Q3	IQR
AE	17.723	34.454	27.144	2.153	25.87	26.758	28.416	2.544
AE + sLC	17.07	48.905	36.89	6.374	30.971	**38.88**	42.133	11.162
AE + RB	16.64	35.673	28.721	2.430	27.463	28.521	30.170	2.707
AE + sLC + RB (v1)	18.84	42.689	33.159	3.906	30.163	33.984	35.667	5.504
AE + sLC + RB (v2)	18.79	49.629	**37.132**	6.239	31.372	38.599	42.05	10.67

Table 4.5: PSNR values when training models using SSIM as the loss function.

Model name	min	max	mean	standard deviation	Q1	median	Q3	IQR
AE	0.644	0.951	0.884	0.029	0.869	0.886	0.902	0.033
AE + sLC	0.777	0.99909	0.9763	0.029	0.960	0.9928	0.9984	0.037
AE + RB	0.673	0.977	0.934	0.026	0.925	0.940	0.950	0.025
AE + sLC + RB (v1)	0.768	0.993	0.966	0.027	0.953	0.978	0.985	0.031
AE + sLC + RB (v2)	0.802	0.99916	**0.9774**	0.028	0.963	**0.9933**	0.9984	0.035

Table 4.6: SSIM values when training models using SSIM as the loss function.

4.2.2 Qualitative

As seen in the quantitative subsection, the *AE + sLC + RB (v2)* model is the one that has obtained the best results, followed by the *AE + sLC* model. To prove the quality of the results given by all models, 10 random samples from the test set has been extracted and reconstructed by all models, as it can be seen in both Figure 4.4 and Figure 4.5.

Focusing on the first experiment, 5 images have been obtained (see Figure 4.4a). The first 3 images and the latest has been augmented by dropping pixels, while in the fourth a Gaussian noise has been added to it. Regarding the results of the models trained using MSE as the loss function (see Figure 4.4b), it can be observed that the additive Gaussian noise has been

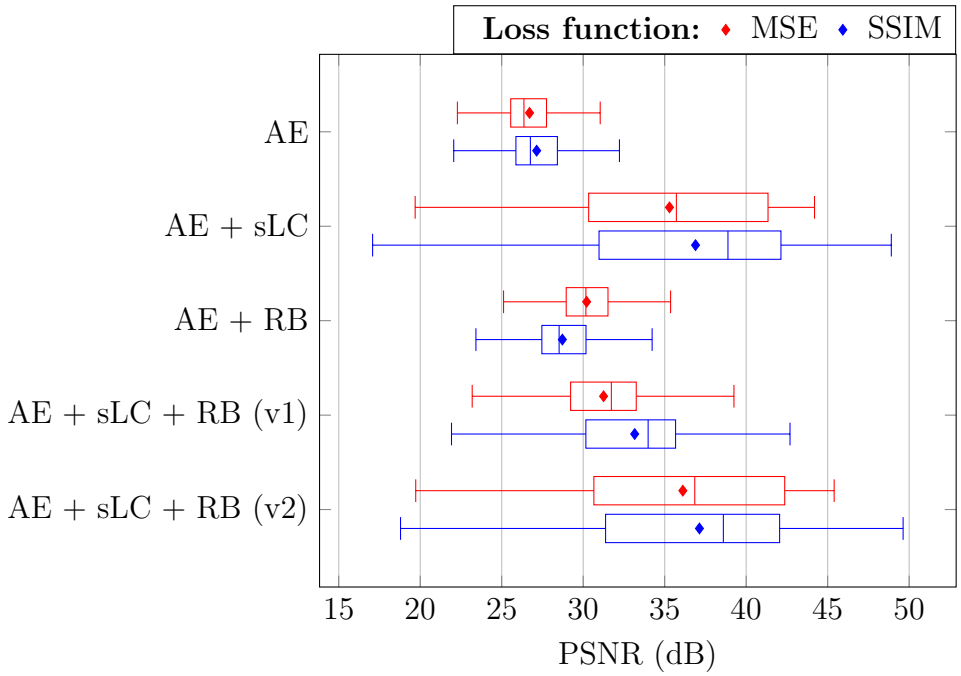


Figure 4.2: Boxplot of the PSNR results for each loss function.

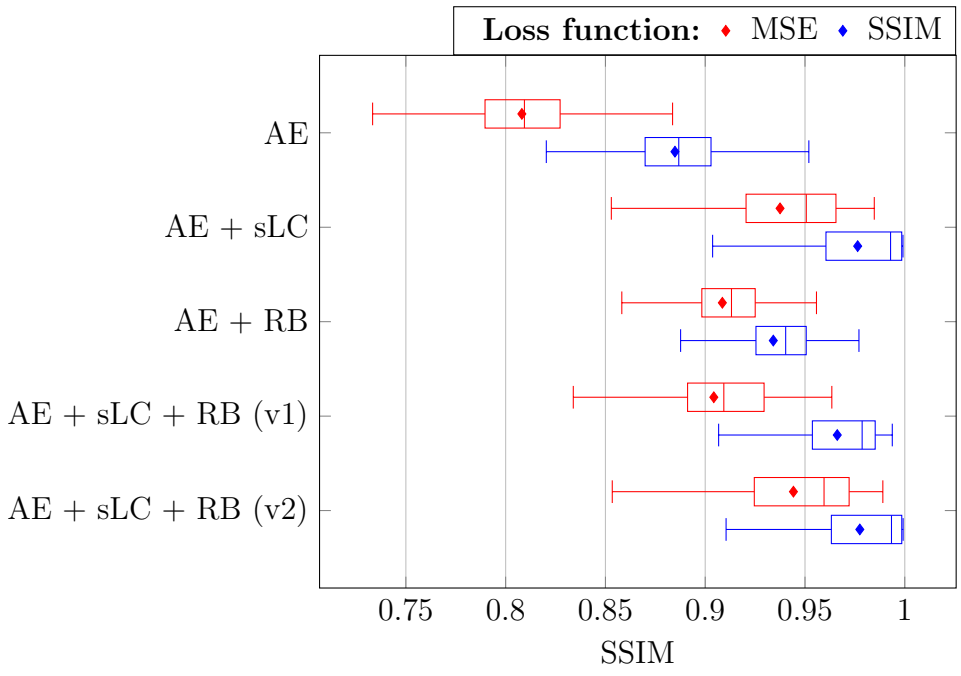
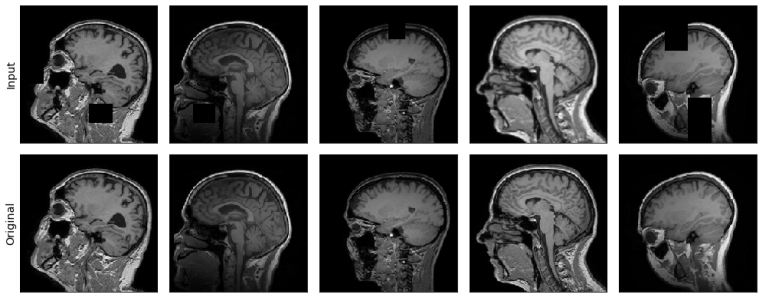


Figure 4.3: Boxplot of the SSIM results for each loss function.

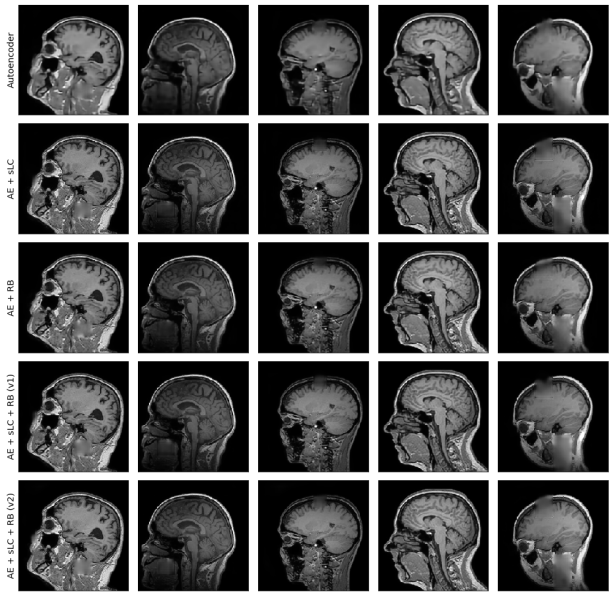
removed by all models, and (in except the baseline) the images have been reconstructed with a good level of sharpness. Besides, as it was expected, the baseline produces blurry results, even affecting those regions on the image that has not been altered by the augmentation process. The regions that have been blackout tend to generate edgy results on those models that use skipped long connections. The use of residual blocks seems to produce a better prediction of the content of the dropped pixels but, those models that use skipped long connections too are taking profit from it and produce a better quality image overall, with better sharpness and contrast.

When using SSIM as the loss function a general improvement can be noticed. Moreover, it can be observed that images are a little bit darker in general. This can be especially perceived in the fourth image. Regarding dropped regions, SSIM are making *AE + sLC + RB (v2)* to have the best results, indubitably. The blacked-out regions are predicted well and less edgy results are obtained.

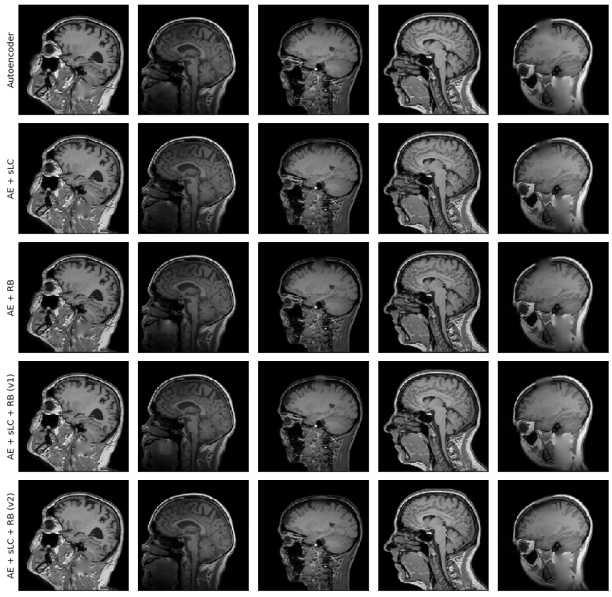
Very similar things happen in the second experiment (see Figure 4.5) regarding dropped pixels and additive Gaussian noise. The main difference in this experiment is the introduction of the salt-and-pepper noise, as seen in the last image of Figure 4.5a. As can be observed, this type of noise has been removed from the image easily by all models. The ones trained using SSIM as the loss function produce slightly better images in terms of sharpness. Regarding the blacked-out regions, similar conclusions can be derived as in the previous experiment.



(a) Experiment images to be used.

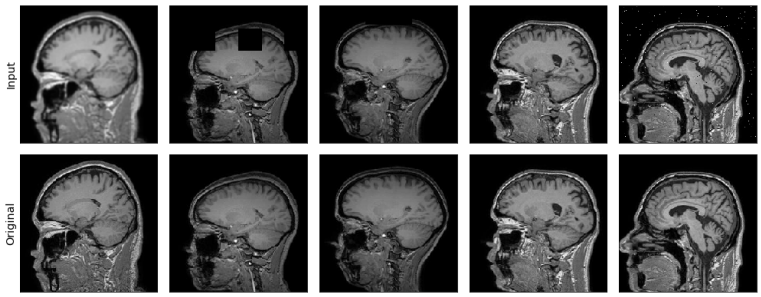


(b) Results using models trained using MSE as the loss function.

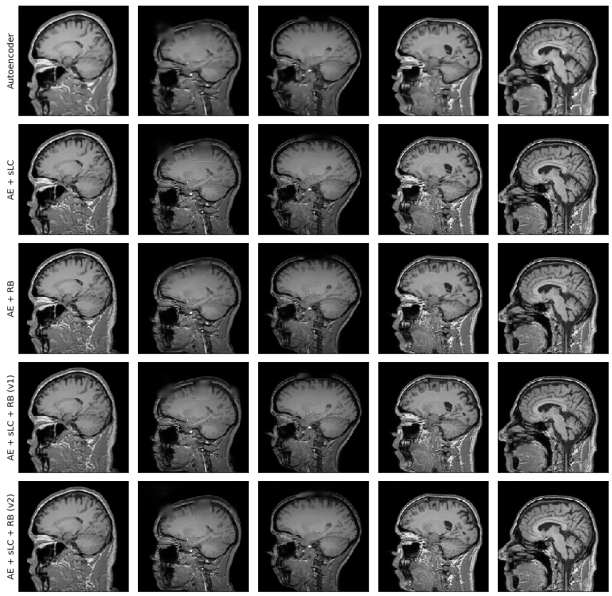


(c) Results using models trained using SSIM as the loss function.

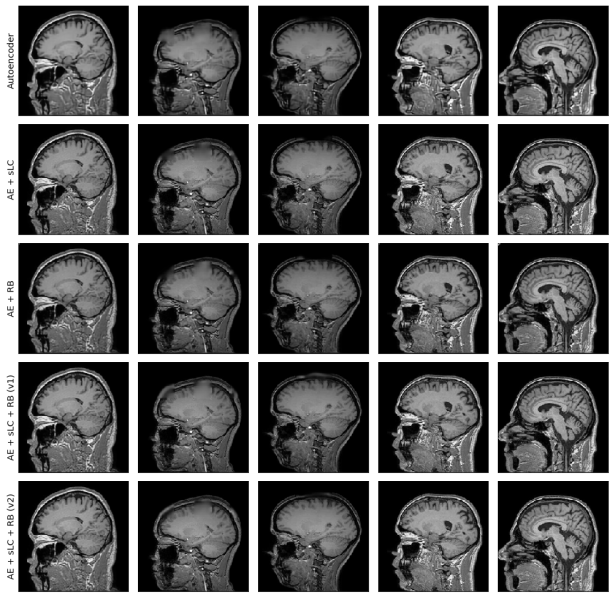
Figure 4.4: Results for the first set of image samples.



(a) Experiment images to be used.



(b) Results using models trained using MSE as the loss function.



(c) Results using models trained using SSIM as the loss function.

Figure 4.5: Results for the second set of image samples.

Chapter 5

Conclusions

During the course of the project, different ways of reconstructing brain magnetic resonance images to obtain the healthy version of the inputs, focusing on unsupervised learning methods, have been studied. That is, instead of learning how to reconstruct each type of pathology (being that the supervised alternative), a different approach has been considered. Starting from the base that finding a dataset containing such a large number of pathologies with their respective healthy versions would have been barely impossible, we have opted for a model which mission was to learn the intrinsic patterns shared between the healthy brains in order to force them into the output.

After diving into the actual literature review in order to understand how the state of the art models work, it was seen that two architectures were mainly used to carry on with this type of tasks: autoencoders and GAN. At first sight, seemed that GAN provided promising results but their main drawback is that is difficult to train due to their trend to converge towards mode collapse. Being that said, GAN was set aside and the goal was focused on improving the autoencoders' performance.

Therefore, and for the sake of comparability, a workbench was developed in order to evaluate all the designed models under the same circumstances. After some research, it was decided to train our models using both MSE and SSIM as the loss functions and using the Adam optimizer. In order to avoid our models to learn only to copy the input into the output, the brain images (from the *IXI Dataset*) were corrupted with different types of noise and masking some regions. Moreover, the data were split in such a way that none of the images of the same subject were shared among the splits.

Inspired by the introduction of the residual blocks in the Resnet architecture and the im-

provement that this entailed, it was decided to try skipping connections in order to prove if that would improve our models too. Besides, apart from that, the use of long shortcuts also introduced improvements, as found in the literature review. Therefore, a simple autoencoder was used as the baseline and different combinations of short and long skipped connections were added to see how they behave. Actually, 4 different combinations has been tried: only skipping long connections (*AE + sLC*), only using residual blocks (*AE + RB*), combining both but omitting the first skipped long connection (*AE + sLC + RB (v1)*), and fully combining both (*AE + sLC + RB (v2)*).

The model with the best results has been *AE + sLC + RB (v2)* which takes advantages of both the high frequencies using the long shortcuts and the accuracy improvement of the residual blocks. In fact, it has achieved a PSNR improvement of 9.4dB with respect to the baseline when using MSE as the loss function, and 9.74dB when using SSIM instead. In terms of the SSIM metrics, an enhancement of 0.136 has been reached when using MSE as the loss function, and 0.0934 when using SSIM loss. The use of only residual block also introduces improvements but not as significant as using skipped long connections. The assumption behind this behavior is that the long shortcuts let the models bypass the high frequencies of the images directly to the output and, despite this can produce edgy results on the blacked-out regions, the image sharpness is better overall. Besides, the use of the first skipped long connection is relevant, as has been proved, being the results of *AE + sLC + RB (v1)* worse than the ones provided by *AE + sLC + RB (v2)*.

Finally, the use of SSIM as the loss function provides the best results for both PSNR and SSIM metrics. Even PSNR, which has a direct relationship with MSE, is positively affected. This fact can be verified in both quantitative and qualitative results, producing images with better structures and contours being these reconstructed with less blurriness and artifacts.

5.1 Future developments

- Given the good results of using SSIM as the loss function, more complex SSIM-related function such as MS-SSIM [87] can be tried.
- Use dropout layers and L2 regularizer to verify if the models can generalize better.
- Using deeper models to see if the results are improved.
- Try with other initial feature maps and see the performance.

- Study the effect of increasing or reducing the size of the latent space.
- Use different optimizers.
- Use the *Conv2DTranspose* Keras layer instead of the *Upsampling2D + Conv2D* tuple for the decoder part.
- Use more healthy brain's datasets to see if the model can generalize better.
- Use different blocks and combinations, such as Inception blocks, which results looks promising [88, 77, 89, 90].
- Use variational autoencoders (VAEs) for the results by aggregating means of Monte-Carlo reconstructions [29, 38, 91].
- Using irregular shapes to mask-out region in order to avoid overfitting the additive shortcut in long connections [92].
- Use GAN-based architectures.

Bibliography

[1] D. Tamada, *Review: Noise and artifact reduction for mri using deep learning*, 2020. arXiv: 2002.12889 [eess.IV].

[2] M. A. Bruno, E. A. Walker, and H. H. Abujudeh, "Understanding and confronting our mistakes: The epidemiology of error in radiology and strategies for error reduction", *RadioGraphics*, vol. 35, no. 6, pp. 1668–1676, 2015, PMID: 26466178. DOI: 10.1148/rg.2015150023. [Online]. Available: https://doi.org/10.1148/rg.2015150023.

[3] M. T. Abou-Saleh, "Neuroimaging in psychiatry: An update", *Journal of Psychosomatic Research*, vol. 61, pp. 289–293, Sep. 2006. DOI: 10.1016/j.jpsychores.2006.06.012.

[4] S. Klöppel, A. Abdulkadir, C. R. Jack, N. Koutsouleris, J. Mourão-Miranda, and P. Vemuri, "Diagnostic neuroimaging across diseases", *NeuroImage*, vol. 61, pp. 457–463, Jun. 2012. DOI: 10.1016/j.neuroimage.2011.11.002.

[5] W. H. L. Pinaya, A. Mechelli, and J. R. Sato, "Using deep autoencoders to identify abnormal brain structural patterns in neuropsychiatric disorders: A large-scale multi-sample study", *Human Brain Mapping*, vol. 40, pp. 944–954, Oct. 2018. DOI: 10.1002/hbm.24423.

[6] S. Durston, "A review of the biological bases of adhd: What have we learned from imaging studies?", *Mental Retardation and Developmental Disabilities Research Reviews*, vol. 9, pp. 184–195, 2003. DOI: 10.1002/mrdd.10079.

[7] I. Ellison-Wright, D. C. Glahn, A. R. Laird, S. M. Thelen, and E. Bullmore, "The anatomy of first-episode and chronic schizophrenia: An anatomical likelihood estimation meta-analysis", *American Journal of Psychiatry*, vol. 165, pp. 1015–1023, Aug. 2008. DOI: 10.1176/appi.ajp.2008.07101562. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2873788/.

[8] A. Taboada-Crispi, H. Sahli, M. Orozco Monteagudo, D. Hernandez Pacheco, and A. Falcon, "Anomaly detection in medical image analysis", in. Jan. 2009, pp. 426–446. DOI: 10.4018/978-1-60566-314-2.ch027.

[9] V. Hodge and J. Austin, "A survey of outlier detection methodologies", *Artificial Intelligence Review*, vol. 22, pp. 85–126, Oct. 2004. DOI: 10.1023/b:aire.0000045502.10941.a9. [Online]. Available: https://link.springer.com/article/10.1023/B:AIRE.0000045502.10941.a9.

[10] C. Baur, S. Denner, B. Wiestler, S. Albarqouni, and N. Navab, *Autoencoders for unsupervised anomaly segmentation in brain mr images: A comparative study*, 2020. arXiv: 2004.03271 [eess.IV].

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, pp. 436–444, May 2015. DOI: 10.1038/nature14539.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[13] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.

[14] J. Tompson, A. Jain, Y. LeCun, and C. Bregler, *Joint training of a convolutional network and a graphical model for human pose estimation*, 2014. arXiv: 1406.2984 [cs.CV].

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV].

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Dec. 1998. DOI: 10.1109/5.726791.

[17] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?", in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 2146–2153.

[18] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", vol. 15, Jan. 2010.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014, ISSN: 1532-4435.

[20] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015. arXiv: 1512.00567 [cs.CV].

[22] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[23] F. Chollet, *Xception: Deep learning with depthwise separable convolutions*, 2017. arXiv: 1610.02357 [cs.CV].

[24] M. Oliveira, B. Bowen, R. McKenna, and Y.-S. Chang, "Fast digital image inpainting.", Jan. 2001, pp. 261–266.

[25] Q. Li, "Image inpainting based on sparse representation with histogram dictionary", *Journal of Computers*, pp. 1145–1155, Jan. 2018. DOI: 10.17706/jcp.13.10.1145-1155.

[26] S. Munduru and M. Rao Kothari, "Image filling by using texture synthesis", *International Journal of Scientific Engineering and Technology Research*, vol. 03, pp. 2355–2363, 2014. [Online]. Available: http://ijsetr.com/uploads/534162IJSETR1377-399.pdf.

[27] M. Elad, J.-L. Starck, P. Querre, and D. Donoho, "Simultaneous cartoon and texture image inpainting using morphological component analysis (mca)", *Applied and Computational Harmonic Analysis*, vol. 19, pp. 340–358, Nov. 2005. DOI: 10.1016/j.acha.2005.03.005.

[28] C. Guillemot and O. Le Meur, "Image inpainting : Overview and recent advances", *IEEE Signal Processing Magazine*, vol. 31, pp. 127–144, Jan. 2014. DOI: 10.1109/msp.2013.2273004.

[29] C. Baur, B. Wiestler, S. Albarqouni, and N. Navab, "Deep autoencoding models for unsupervised anomaly segmentation in brain mr images", *Lecture Notes in Computer Science*, pp. 161–169, 2019, ISSN: 1611-3349. DOI: 10.1007/978-3-030-11723-8_16. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-11723-8_16.

[30] H. E. Atlason, A. Love, S. Sigurdsson, V. Gudnason, and L. M. Ellingsen, *Unsupervised brain lesion segmentation from mri using a convolutional autoencoder*, 2018. arXiv: 1811.09655 [eess.IV].

[31] J. Kim, J. Lee, and K. Lee, "Accurate image super-resolution using very deep convolutional networks", Nov. 2015.

[32] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising", *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[33] D. Lee, J. Yoo, and J. C. Ye, "Deep residual learning for compressed sensing mri", in *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, 2017, pp. 15–18.

[34] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV].

[35] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, *Noise2noise: Learning image restoration without clean data*, 2018. arXiv: 1803.04189 [cs.CV].

[36] A. J. Plassard, L. T. Davis, A. T. Newton, S. M. Resnick, B. A. Landman, and C. Bermudez, "Learning implicit brain mri manifolds with deep learning", *Medical Imaging 2018: Image Processing*, E. D. Angelini and B. A. Landman, Eds., May 2018. DOI: 10.1117/12.2293515. [Online]. Available: http://dx.doi.org/10.1117/12.2293515.

[37] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, *Self-normalizing neural networks*, 2017. arXiv: 1706.02515 [cs.LG].

[38] D. Zimmerer, S. A. A. Kohl, J. Petersen, F. Isensee, and K. H. Maier-Hein, *Context-encoding variational autoencoder for unsupervised anomaly detection*, 2018. arXiv: 1812.05941 [cs.LG].

[39] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury, and L. S. Davis, *Learning temporal regularity in video sequences*, 2016. arXiv: 1604.04574 [cs.CV].

[40] Y. S. Chong and Y. H. Tay, *Abnormal event detection in videos using spatiotemporal autoencoder*, 2017. arXiv: 1701.01546 [cs.CV].

[41] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2014. arXiv: 1312.6114 [stat.ML].

[42] N. Pawlowski, M. Lee, M. Rajchl, S. Mcdonagh, E. Ferrante, K. Kamnitsas, S. Cooke, S. Stevenson, A. Khetani, T. Newman, F. Zeiler, R. Digby, J. Coles, D. Rueckert, D. Menon, V. Newcombe, and B. Glocker, *Unsupervised lesion detection in brain ct using bayesian convolutional autoencoders*, 2018. [Online]. Available: https://openreview.net/pdf?id=S1hpzoisz.

[43] H. Uzunova, S. Schultz, H. Handels, and J. Ehrhardt, "Unsupervised pathology detection in medical images using conditional variational autoencoders", *International Journal of Computer Assisted Radiology and Surgery*, vol. 14, pp. 451–461, Dec. 2018. DOI: 10.1007/s11548-018-1898-0.

[44] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, 2014. arXiv: 1406.2661 [stat.ML].

[45] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, "Generative visual manipulation on the natural image manifold", in *Proceedings of European Conference on Computer Vision (ECCV)*, 2016.

[46] R. A. Yeh, C. Chen, T. Y. Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, *Semantic image inpainting with deep generative models*, 2017. arXiv: 1607.07539 [cs.CV].

[47] Y. Li, S. Liu, J. Yang, and M.-H. Yang, *Generative face completion*, 2017. arXiv: 1704.05838 [cs.CV].

[48] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion", *ACM Transactions on Graphics*, vol. 36, pp. 1–14, Jul. 2017. DOI: 10.1145/3072959.3073659.

[49] C. Han, L. Rundo, K. Murao, T. Noguchi, Y. Shimahara, Z. Milacski, S. Koshino, E. Sala, H. Nakayama, and S. Ichi Satoh, *Madgan: Unsupervised medical anomaly detection gan using multiple adjacent brain mri slice reconstruction*, Jul. 2020. [Online]. Available: https://arxiv.org/pdf/2007.13559.pdf.

[50] K. Armanious, C. Jiang, M. Fischer, T. Küstner, T. Hepp, K. Nikolaou, S. Gatidis, and B. Yang, "Medgan: Medical image translation using gans", *Computerized Medical Imaging and Graphics*, vol. 79, p. 101 684, Jan. 2020. DOI: 10.1016/j.compmedimag.2019.101684.

[51] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, *Generative image inpainting with contextual attention*, 2018. arXiv: 1801.07892 [cs.CV].

[52] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. Huang, *Free-form image inpainting with gated convolution*, 2019. arXiv: 1806.03589 [cs.CV].

[53] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein gan*, 2017. arXiv: 1701.07875 [stat.ML].

[54] S. Benson and R. Beets-Tan, "Gan-based anomaly detection in multi-modal mri images", Jul. 2020. DOI: 10.1101/2020.07.10.197087.

[55] C. Han, L. Rundo, K. Murao, Z. Milacski, K. Umemoto, E. Sala, H. Nakayama, and S. Ichi Satoh, *Gan-based multiple adjacent brain mri slice reconstruction for unsupervised alzheimer's disease diagnosis*, 2020. [Online]. Available: https://arxiv.org/pdf/1906.06114.pdf.

[56] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, *Unsupervised anomaly detection with generative adversarial networks to guide marker discovery*, 2017. arXiv: 1703.05921 [cs.CV].

[57] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "F-anogan: Fast unsupervised anomaly detection with generative adversarial networks", *Medical Image Analysis*, vol. 54, pp. 30–44, May 2019. DOI: 10.1016/j.media.2019.01.010. [Online]. Available: https://arxiv.org/pdf/1703.05921.pdf.

[58] X. Yi, E. Walia, and P. Babyn, "Generative adversarial network in medical imaging: A review", *Medical Image Analysis*, vol. 58, p. 101 552, Dec. 2019. DOI: 10.1016/j.media.2019.101552.

[59] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, *Autoencoding beyond pixels using a learned similarity metric*, 2016. arXiv: 1512.09300 [cs.LG].

[60] R. Fong and A. Vedaldi, *Occlusions for effective data augmentation in image classification*, 2019. arXiv: 1910.10651 [cs.CV].

[61] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation", in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[62] F. Chollet, *Building powerful image classification models using very little data*, Keras.io, Jun. 2016. [Online]. Available: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

[63] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, *Loss functions for neural networks for image processing*, 2018. arXiv: 1511.08861 [cs.CV].

[64] P. Bergmann, S. Löwe, M. Fauser, D. Sattlegger, and C. Steger, "Improving unsupervised defect segmentation by applying structural similarity to autoencoders", *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2019. DOI: 10.5220/0007364503720380. [Online]. Available: http://dx.doi.org/10.5220/0007364503720380.

[65] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "A comprehensive evaluation of full reference image quality assessment algorithms", in *2012 19th IEEE International Conference on Image Processing*, 2012, pp. 1477–1480. DOI: 10.1109/ICIP.2012.6467150.

[66] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity", *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[67] J. Snell, K. Ridgeway, R. Liao, B. D. Roads, M. C. Mozer, and R. S. Zemel, *Learning to generate images with perceptual similarity metrics*, 2017. arXiv: 1511.06409 [cs.LG].

[68] R. Y. Yu, *Comparison of optimizers in neural networks - fishpond*, tiddler.github.io, Dec. 2016. [Online]. Available: https://tiddler.github.io/optimizers/.

[69] G. Hinton, N. Srivastava, and K. Swersky, *Neural networks for machine learning lecture 6a overview of mini–batch gradient descent*. [Online]. Available: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[70] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[71] M. D. Zeiler, *Adadelta: An adaptive learning rate method*, 2012. arXiv: 1212.5701 [cs.LG].

[72] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2016. arXiv: 1511.06434 [cs.LG].

[73] S. Akçay, A. Atapour-Abarghouei, and T. P. Breckon, *Skip-ganomaly: Skip connected and adversarially trained encoder-decoder anomaly detection*, 2019. arXiv: 1901.08954 [cs.CV].

[74] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models", 2013.

[75] B. Xu, N. Wang, T. Chen, and M. Li, *Empirical evaluation of rectified activations in convolutional network*, 2015. arXiv: 1505.00853 [cs.LG].

[76] A.-S. Collin and C. D. Vleeschouwer, *Improved anomaly detection by training an autoencoder with skip connections on images corrupted with stain-shaped noise*, 2020. arXiv: 2008.12977 [eess.IV].

[77] J. Wei and Z. Fan, *Genetic u-net: Automatically designing lightweight u-shaped cnn architectures using the genetic algorithm for retinal vessel segmentation*, 2020. arXiv: 2010.15560 [eess.IV].

[78] Z. Wang, N. Zou, D. Shen, and S. Ji, *Non-local u-net for biomedical image segmentation*, 2020. arXiv: 1812.04103 [cs.CV].

[79] A. Shvets, V. Iglovikov, A. Rakhlin, and A. Kalinin, "Angiodysplasia detection and localization using deep convolutional neural networks", Apr. 2018.

[80] Ming Liang and Xiaolin Hu, "Recurrent convolutional neural network for object recognition", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3367–3375. DOI: 10.1109/CVPR.2015.7298958.

[81] K. Bajaj, D. K. Singh, and M. A. Ansari, "Autoencoders based deep learner for image denoising", *Procedia Computer Science*, vol. 171, pp. 1535–1541, 2020, Third International Conference on Computing and Network Communications (CoCoNet'19), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.04.164. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050920311431.

[82] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, *The importance of skip connections in biomedical image segmentation*, 2016. arXiv: 1608.04117 [cs.CV].

[83] L. Bao, F. Ye, C. Cai, J. Wu, K. Zeng, P. C. van Zijl, and Z. Chen, "Undersampled mr image reconstruction using an enhanced recursive residual network", *Journal of Magnetic Resonance*, vol. 305, pp. 232–246, 2019, ISSN: 1090-7807. DOI: https://doi.org/10.1016/j.jmr.2019.07.020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1090780719301454.

[84] X.-J. Mao, C. Shen, and Y.-B. Yang, *Image restoration using convolutional auto-encoders with symmetric skip connections*, 2016. arXiv: 1606.08921 [cs.CV].

[85] Y. Tai, J. Yang, and X. Liu, "Image super-resolution via deep recursive residual network", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[86] J. Kim, J. K. Lee, and K. M. Lee, *Deeply-recursive convolutional network for image super-resolution*, 2016. arXiv: 1511.04491 [cs.CV].

[87] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment", in *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, vol. 2, 2003, 1398–1402 Vol.2. DOI: 10.1109/ACSSC.2003.1292216.

[88] N. Sarafijanovic-Djukic and J. Davis, "Fast distance-based anomaly detection in images using an inception-like autoencoder", *Lecture Notes in Computer Science*, pp. 493–508, 2019, ISSN: 1611-3349. DOI: 10.1007/978-3-030-33778-0_37. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-33778-0_37.

[89] Z. Zhang, C. Wu, S. Coleman, and D. Kerr, "Dense-inception u-net for medical image segmentation", *Computer Methods and Programs in Biomedicine*, vol. 192, p. 105 395, 2020, ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2020.105395. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169260719307904.

[90] I. Delibasoglu and M. Cetin, "Improved U-Nets with inception blocks for building detection", *Journal of Applied Remote Sensing*, vol. 14, no. 4, pp. 1–15, 2020. DOI: 10.1117/1.JRS.14.044512. [Online]. Available: https://doi.org/10.1117/1.JRS.14.044512.

[91] X. Chen, S. You, K. C. Tezcan, and E. Konukoglu, "Unsupervised lesion detection via image restoration with a normative prior", *Medical Image Analysis*, vol. 64, p. 101 713, 2020, ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2020.101713. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1361841520300773.

[92] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, *Image inpainting for irregular holes using partial convolutions*, 2018. arXiv: 1804.07723 [cs.CV].

Appendix A

Selecting the number of epochs

To determine a reasonable number of epochs to be used for all the models a baseline model has been trained using such a high number of epochs. In this case, 50 epochs were set and, in case the model was still training it will be increased until stability was reached. A baseline model has been compiled using MSE as the loss function and Adam as the optimizer.

As seen in Figure A.1 it can be safely said that from epoch number 20 onwards the model is not learning.

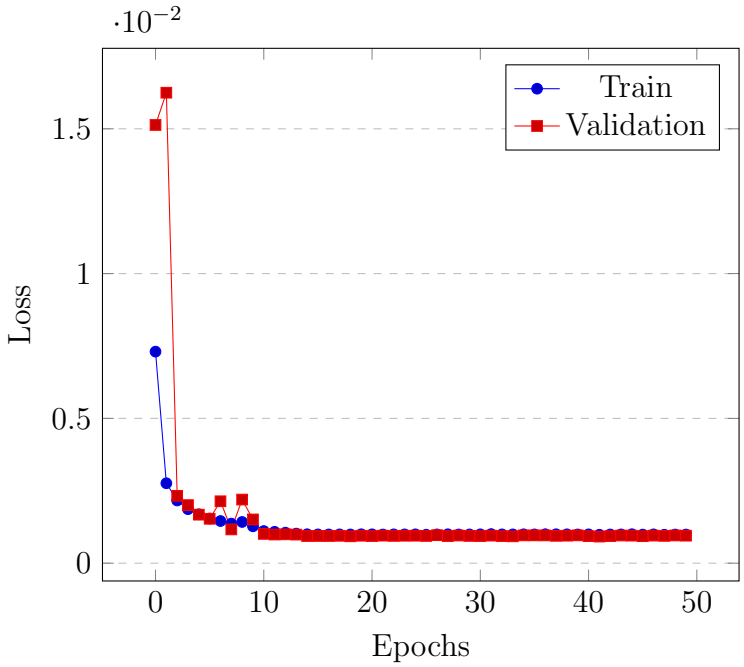


Figure A.1: Training baseline model with 50 epochs.

Appendix B

Behavior using different image sizes

A comparison between different image sizes as input has been performed. To do so, a baseline model has been used, using MSE as the loss function and Adam as the optimizer. A seen in Figure B.1, starting at 192x192px and going down it is noted that peaks appear during training. If attention is focused on Figure B.1 and Figure B.3 one can observe a trade-off between speediness and quality. The bigger the image is the higher time is needed to train the model. Besides, the first impact one gets when observing Figure B.3 is the fact that 96x96px breaks the tendency of the curves. One explanation of that can be that the size of the image is so small that the pixels have been compacted to a level that metrics start failing due to sharpness loss. So then candidates seems to be 256x256px and 224x224px. Even there is a loss in PSNR, the SSIM is almost the same in both cases. It is known that SSIM models better how the human vision system understands the images (see subsection 3.2.1.2), it seems reasonable using 224x244px. Moreover, using this size implies a considerable reduction in the training elapsed time. Finally, as seen in Figure B.1 both 256x256px and 224x224px have similar behavior but, because of the reasons in beforehand mentioned, 224x244px is considered the best.

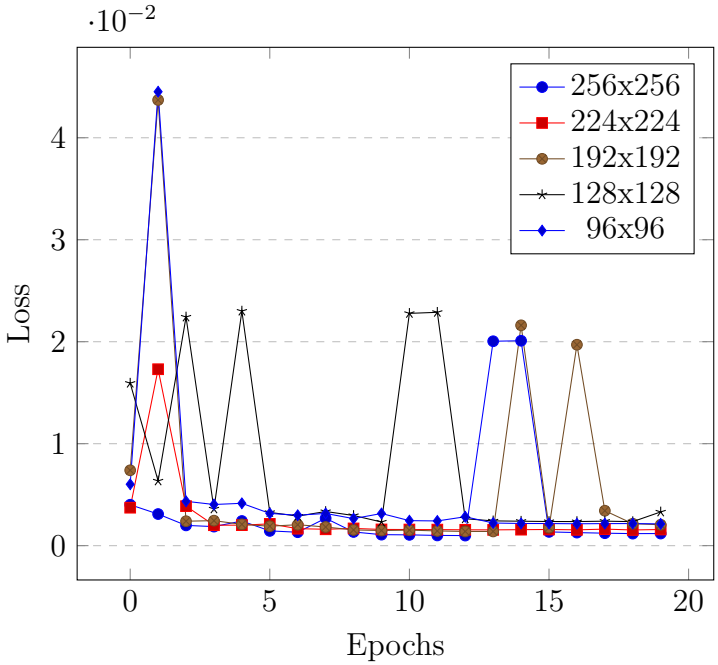


Figure B.1: Image size comparison

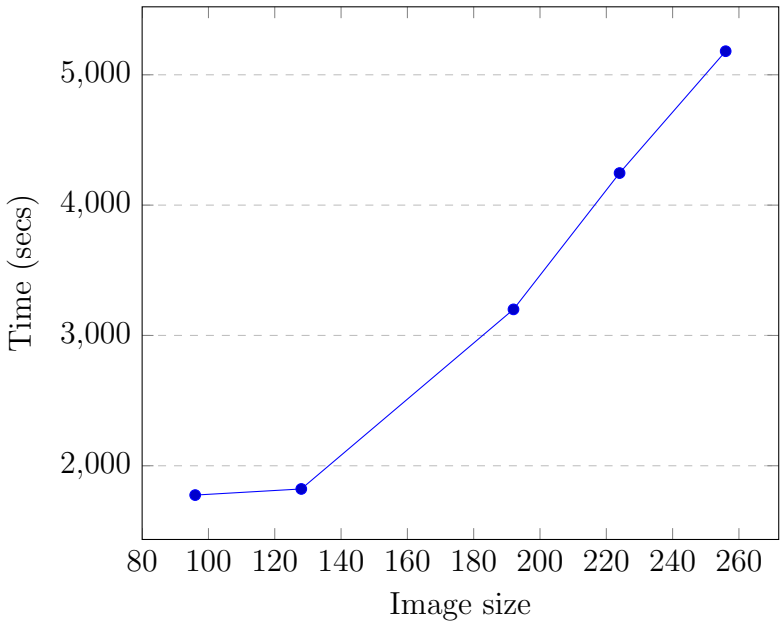


Figure B.2: Training elapsed time in respect of image size.

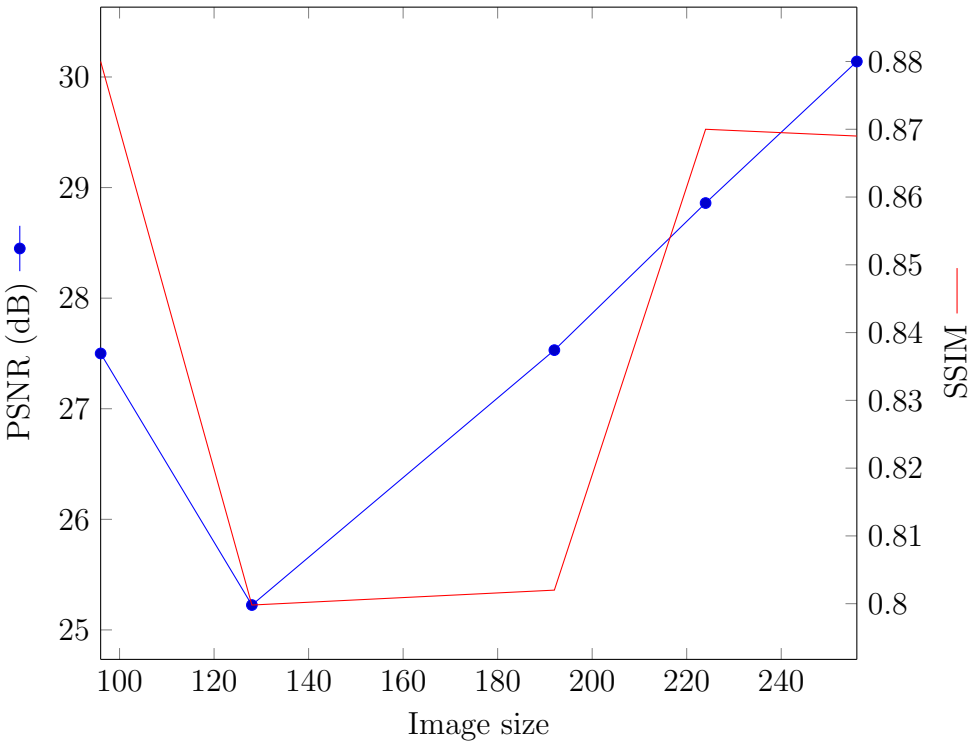


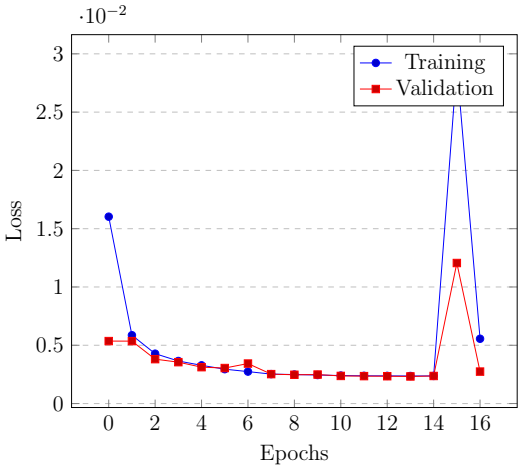
Figure B.3: PSNR and SSIM vs image size.

Appendix C

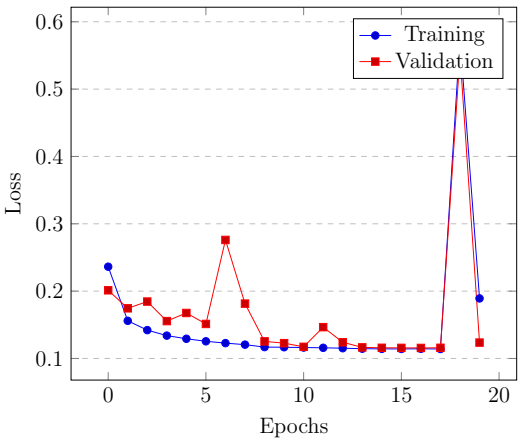
Evolution of training results

In the following sections, the reader will found all the results for each architecture model. The results are related to the evolution of the parameters (both loss function and metrics) during the training phase. It is important to mention that, due to the effect of the *EarlyStopping* callback, not all the models have been trained with the same number of epochs but until they do not present improvement anymore.

C.1 Autoencoder (AE)

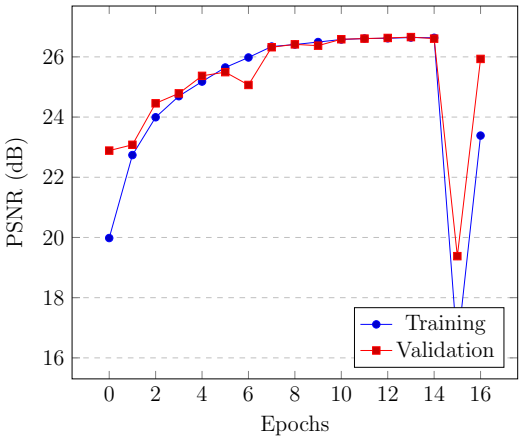


(a) Using MSE as loss function.

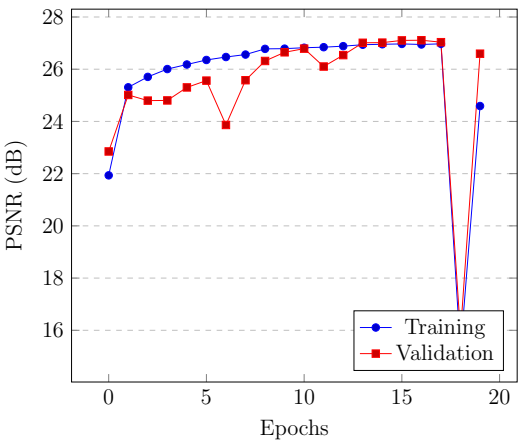


(b) Using SSIM as loss function.

Figure C.1: Evolution of validation loss using both loss functions on the autoencoder model.

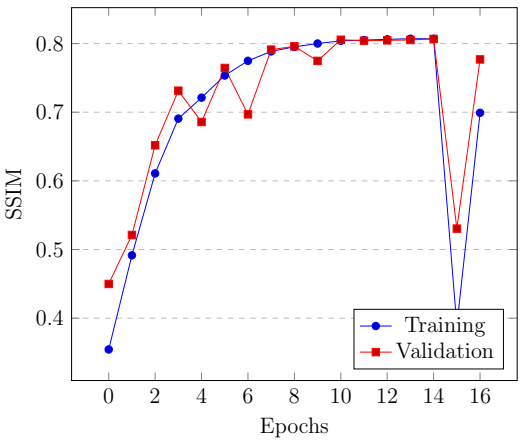


(a) Using MSE as loss function.

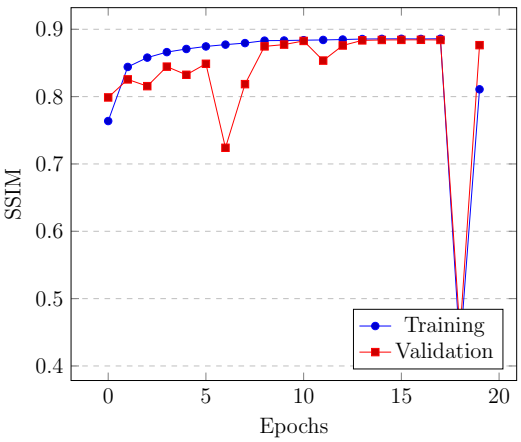


(b) Using SSIM as loss function.

Figure C.2: Evolution of PSNR using both loss functions on the autoencoder model.



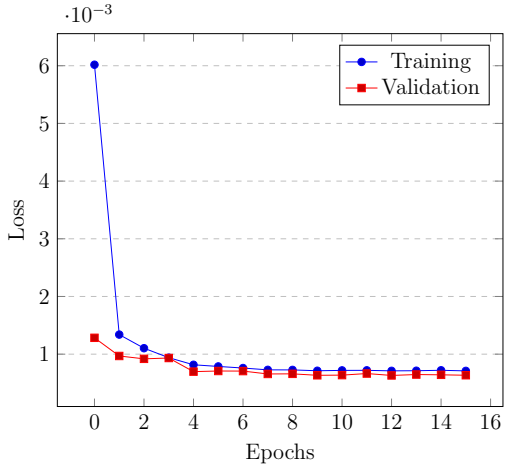
(a) Using MSE as loss function.



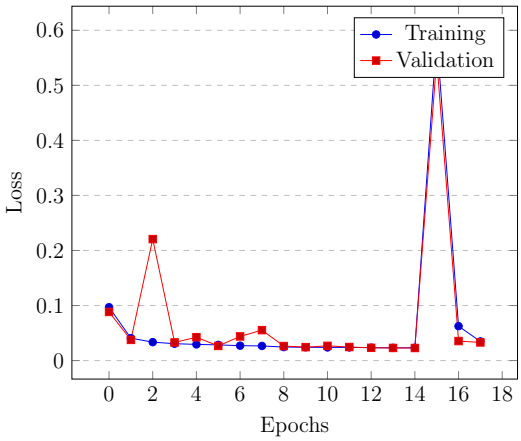
(b) Using SSIM as loss function.

Figure C.3: Evolution of SSIM using both loss functions on the autoencoder model.

C.2 AE + Skipping Long Connections (sLC)

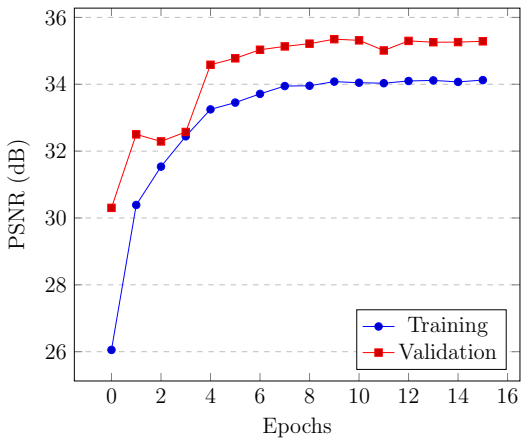


(a) Using MSE as loss function.

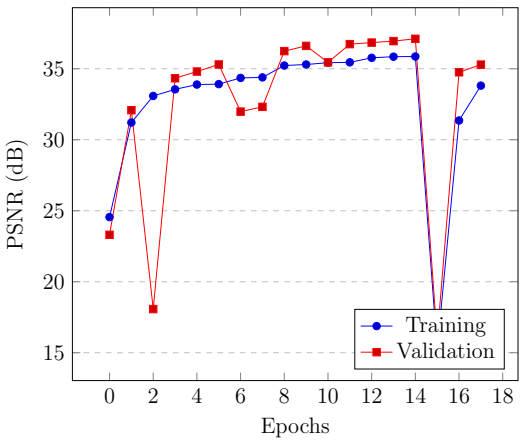


(b) Using SSIM as loss function.

Figure C.4: Evolution of validation loss using both loss functions on the autoencoder + skipped long connections model.

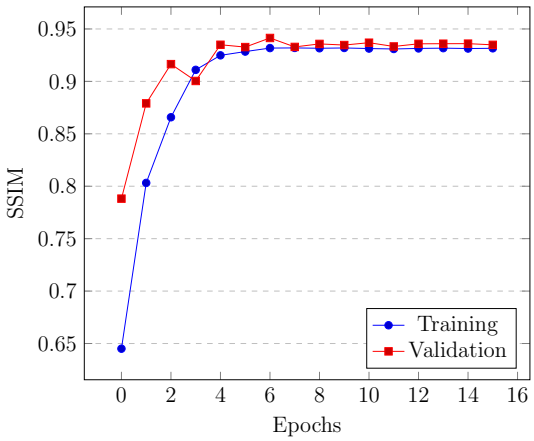


(a) Using MSE as loss function.

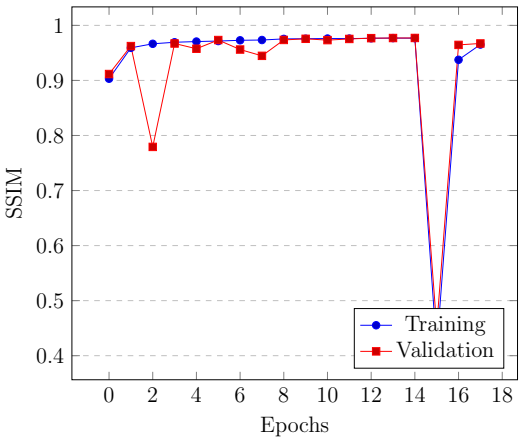


(b) Using SSIM as loss function.

Figure C.5: Evolution of PSNR using both loss functions on the autoencoder + skipped long connections model.



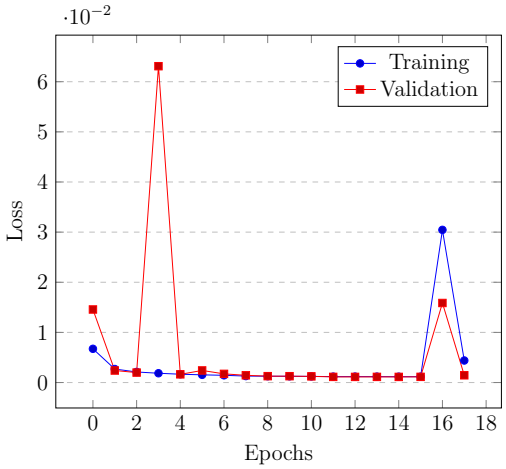
(a) Using MSE as loss function.



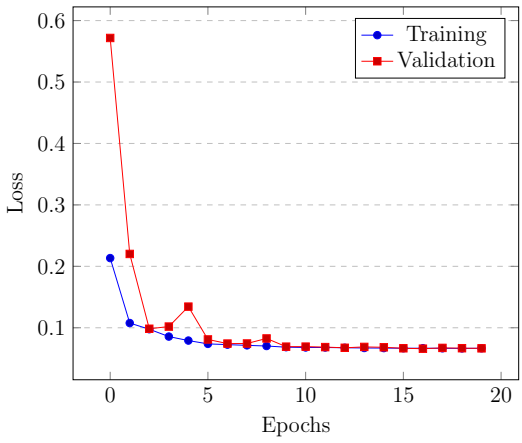
(b) Using SSIM as loss function.

Figure C.6: Evolution of SSIM using both loss functions on the autoencoder + skipped long connections model.

C.3 AE + Residual block (RB)

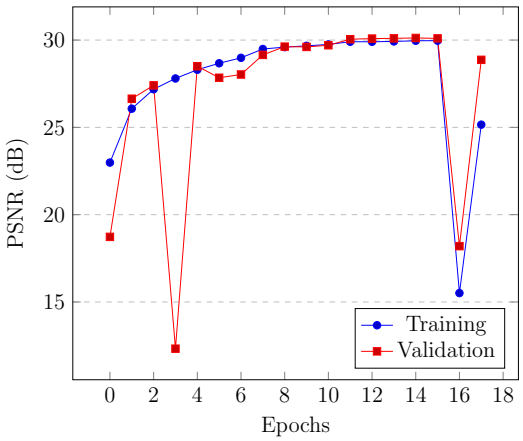


(a) Using MSE as loss function.

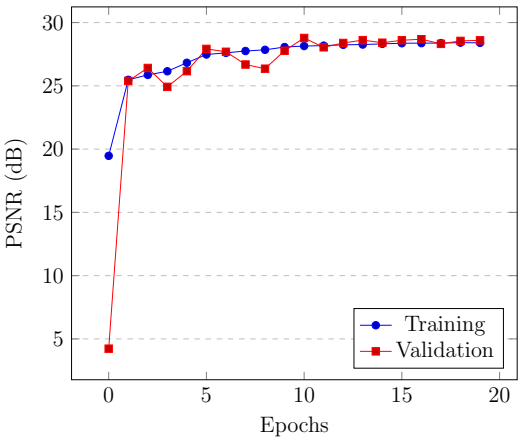


(b) Using SSIM as loss function.

Figure C.7: Evolution of validation loss using both loss functions on the autoencoder + residual blocks model.

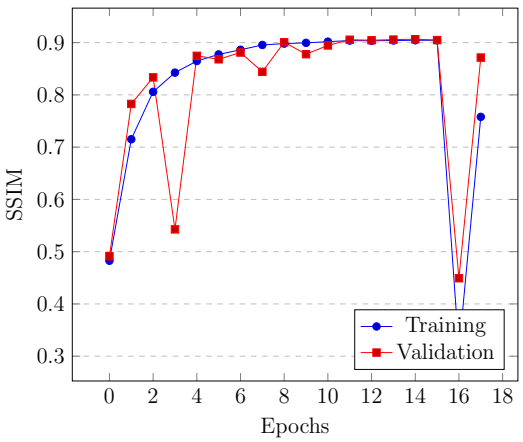


(a) Using MSE as loss function.

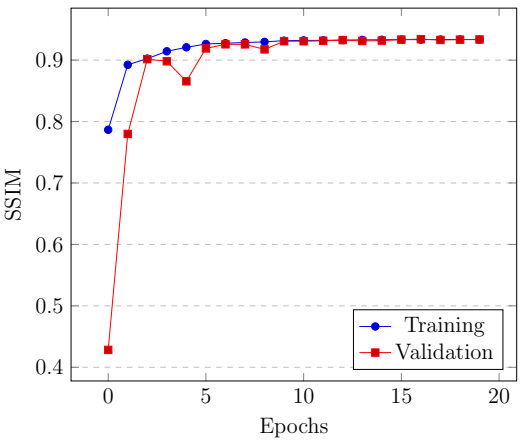


(b) Using SSIM as loss function.

Figure C.8: Evolution of PSNR using both loss functions on the autoencoder + residual blocks model.



(a) Using MSE as loss function.

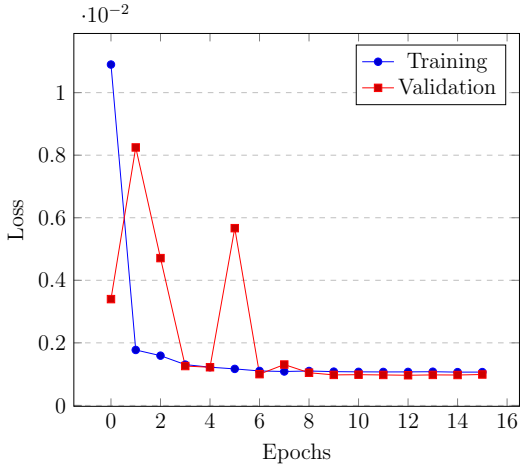


(b) Using SSIM as loss function.

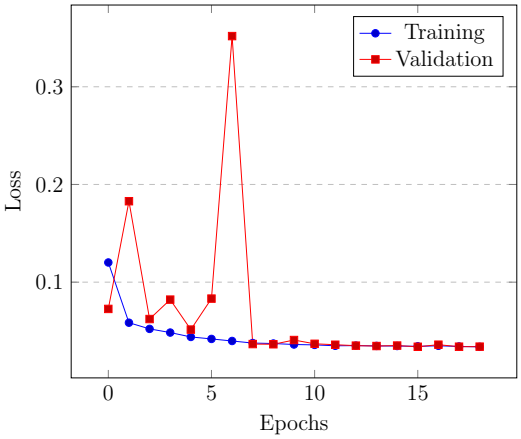
Figure C.9: Evolution of SSIM using both loss functions on the autoencoder + residual blocks model.

C.4 AE + RB + sLC

C.4.1 First version

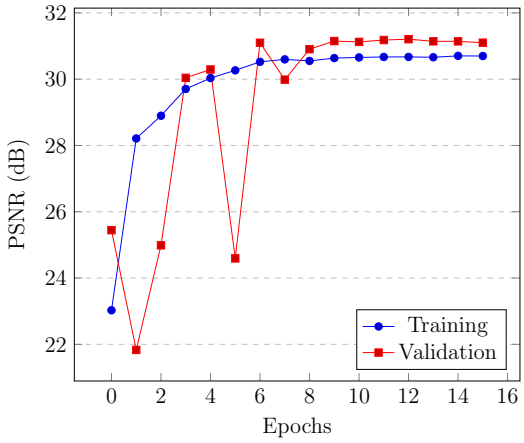


(a) Using MSE as loss function.

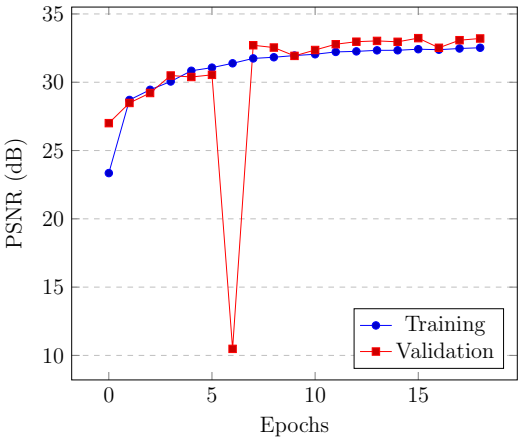


(b) Using SSIM as loss function.

Figure C.10: Evolution of validation loss using both loss functions on the autoencoder + sLC + RB (v1) model.

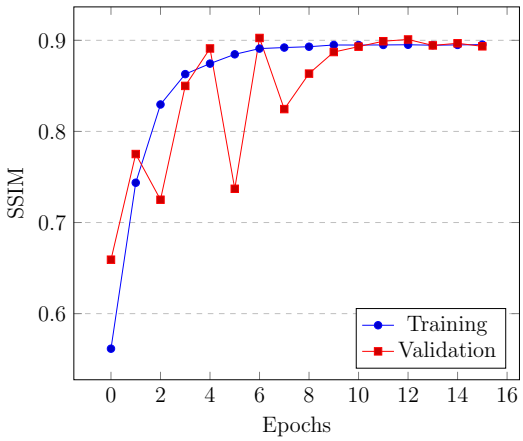


(a) Using MSE as loss function.

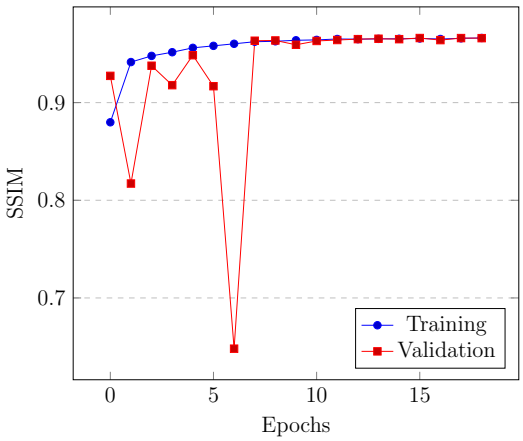


(b) Using SSIM as loss function.

Figure C.11: Evolution of PSNR using both loss functions on the autoencoder + sLC + RB (v1) model.



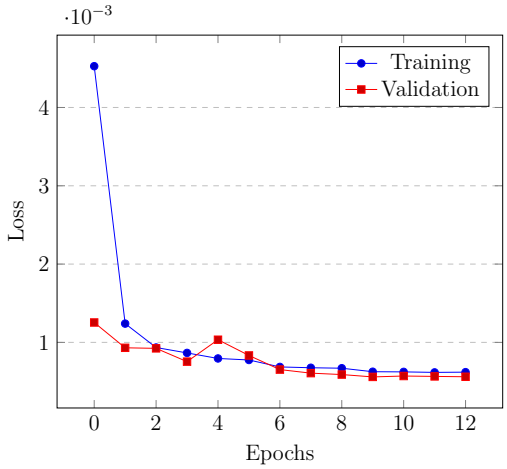
(a) Using MSE as loss function.



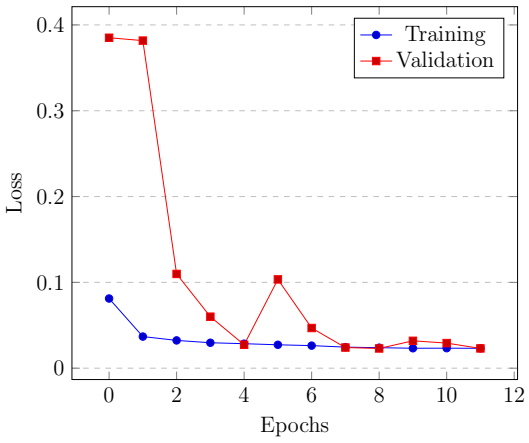
(b) Using SSIM as loss function.

Figure C.12: Evolution of SSIM using both loss functions on the autoencoder + sLC + RB (v1) model.

C.4.2 Second version

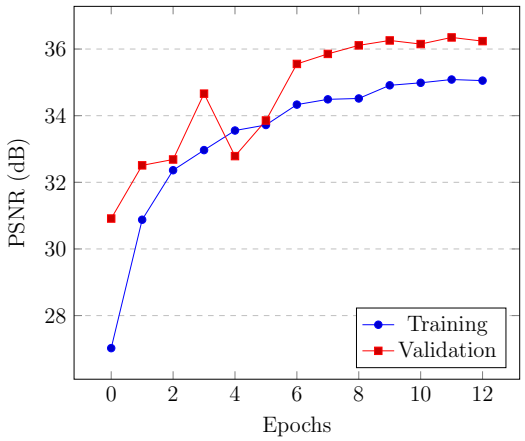


(a) Using MSE as loss function.

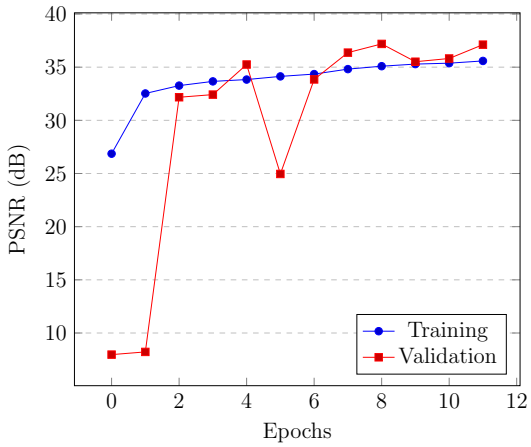


(b) Using SSIM as loss function.

Figure C.13: Evolution of validation loss using both loss functions on the autoencoder + sLC + RB (v2) model.

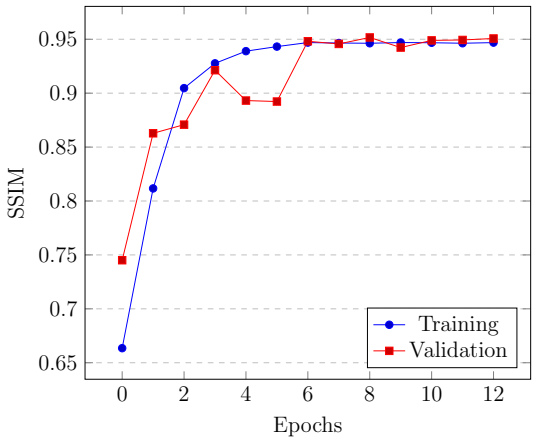


(a) Using MSE as loss function.

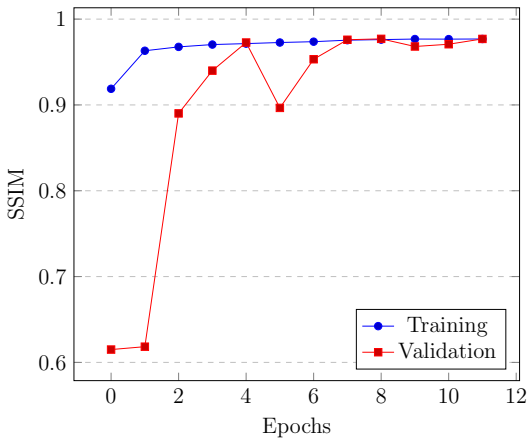


(b) Using SSIM as loss function.

Figure C.14: Evolution of PSNR using both loss functions on the autoencoder + sLC + RB (v2) model.



(a) Using MSE as loss function.



(b) Using SSIM as loss function.

Figure C.15: Evolution of SSIM using both loss functions on the autoencoder + sLC + RB (v2) model.