Universitat Oberta de Catalunya (UOC)

Masters in Data Science

MASTER'S THESIS

Area: Medicine Area (TFM-Med)

Deep Convolutional Autoencoder for control brain MRI

Development and Applications

Author: Adrián Arnaiz Rodríguez

Tutor: Baris Kanber

TFM Professor: Ferran Prados Carrasco

Barcelona, December 14, 2020

Créditos/Copyright

[1] https://github.com/AdrianArnaiz/Brain-MRI-Autoencoder

FICHA DEL TRABAJO FINAL

Título del trabajo:	Deep Convolutional Autoencoder for control brain MRI: Development and Applications
Nombre del autor:	Adrián Arnaiz Rodríguez
Nombre del colaborador/a docente:	Baris Kanber
Nombre del PRA:	Ferrán Prados Carrasco
Fecha de entrega (mm/aaaa):	01/2021
Titulación o programa:	Máster en Ciencia de Datos
Área del Trabajo Final:	Area Medicina (TFM-Med)
Language:	English
Keywords	Deep Learning, Brain MRI, Autoencoder

Dedication/Cite

To Be Done.

Acknowledgment

To Be Done. Si se considera oportuno, mencionar a las personas, empresas o instituciones que hayan contribuido en la realización de este proyecto.

Abstract

The analysis of brain MRI is critical for a proper diagnosis and treatment of neurological diseases. Improvements in this field lead to better health quality. Numerous branches can be still enhanced due to the nature of MRI recompilation: disease detection and segmentation, data augmentation, improvement in data collection, or image enhancement are some of them.

For several years, many approaches have been taken to address this. Machine Learning and Deep Learning emerge as very popular approaches to solve problems. Several kinds of data mining problems (supervised, unsupervised, dimension reduction, generative models, etc) and algorithms can be applied to the problem-solving of MRI. Besides, new emerging deep learning architectures for another kind of image task can be helpful. New types of convolution, autoencoders or generative adversarial networks are some of them.

Therefore, the purpose of this work is to apply one of these new techniques to T1 weighted brain MRI. We will develop a Deep Convolutional Autoencoder, which can be used to help some problems with neuroimaging. The input of the Autoencoder will be control T1WMRI and aims to return the same image, with the problematic that, inside its architecture, the image travels through a lower-dimensional space, so the reconstruction of the original image becomes more difficult. Thus, the Autoencoder represents a normative model.

This normative model will define a distribution (or normal range) for the neuroanatomical variability for the illness absence. Once trained with these control images, we will discuss the potential application of the autoencoder like noise reducer or disease detector.

Keywords: Deep Learning, Brain MRI, Deep Convolutional Autoencoder, Image denoising.

Resumen

El análisis de las resonancias magnéticas cerebrales es fundamental para un diagnóstico y tratamiento adecuados de las enfermedades neurológicas. Se pueden mejorar ámbitos del análisis debido a la naturaleza de la recopilación de resonancias: detección y segmentación de enfermedades, aumento de datos, mejora en la extracción o mejora de imágenes.

El aprendizaje automático y el aprendizaje profundo surgen como nuevas alternativas populares para resolver estos problemas. Se pueden aplicar varios enfoques de minería de datos y algoritmos para la resolución de problemas relacionados con la neuroimagen (supervisados, no supervisados, reducción de dimensionalidad, modelos generativos, etc.). Además, las nuevas arquitecturas emergentes de aprendizaje profundo, desarrolados para otro tipo de tareas de imagen, pueden ser útiles. Algunas de ellas son nuevos tipos de convolución, autoencoders o GAN.

Por lo tanto, el propósito de este trabajo es aplicar una de estas nuevas técnicas a resonancias cerebrales tipo T1. Desarrollaremos un Autoencoder convolucional profundo, que puede usarse para ayudar con algunos problemas de neuroimagen. La entrada del Autoencoder será el imágenes de control T1WMRI y tendrá como objetivo devolver la misma imagen, con la problemática de que, dentro de su arquitectura, la imagen viaja por un espacio de menor dimensión, por lo que la reconstrucción de la imagen original se vuelve más difícil. El autoencoder representa un modelo normativo.

Este modelo normativo definirá una distribución (o rango normal) para la variabilidad neuroanatómica para la ausencia de enfermedad. Una vez entrenado con imágenes de control, discutiremos la aplicación potencial del Autoencoder como reductor de ruido o detector de enfermedades.

Keywords: Aprendizaje profundo, Imágenes cerebrales de resonancias magnéticas, Autoencoder convolucional profundo, eliminación de ruido de imágenes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Not Final Note: This color ('TBD' tag in latex) are going to be used in non-final versions of the memory to highlight the non-final sentences or things to be changed in the final version.

In this chapter we will introduce the main background, and aims of the project, basing it on its non-solved tasks and relevance.

1.1 Problem overview and relevance

1.1.1 MRI general problems

Neuroimaging in medicine allows studying the morphological features of the human brain. With the objective of improving the detection systems, diagnostic and treatment, correlations between the morphological features and the neurological disorders can be addressed in order to achieve that [11]. If we improve brain magnetic resonance image analysis 1.1, we will improve the detection systems and treatments for neurological diseases, so the social relevance of the field is very important.

The relevance of the project is also shown that there are a lot of branches in which neuroimaging analysis can improve. Machine learning, and more recently Deep Learning and Computer Vision, has irrupted in this field for helping in some tasks:

- **Disease detection: segmentation and classification**. There are still several problems

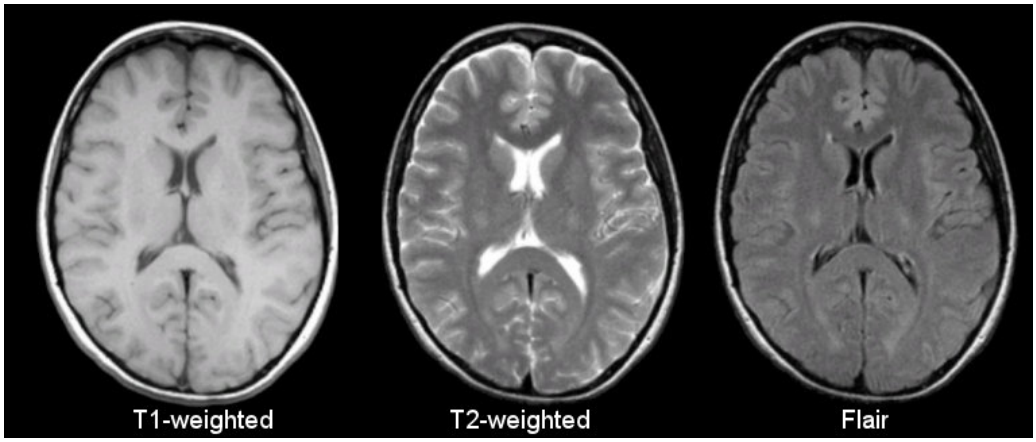

Figure 1.1: Brain MRI examples [1]

to solve in classification and segmentation problems. Brain MRIs are high dimensional, so we have to recruit big amount of images to properly develop a Machine Learning model that be able to achieve high accuracy. It is very difficult to recruit a large number of images, especially disease images. Even if it performs well, machine learning algorithms have been criticized due to the difficult of extract a clear knowledge of them (black-boxes). [3]. So a experimental approach is disease detection based on outliers from a normative model. Patients with pathologies will be outliers in the distribution build by the normative model (they will be out of normal range defined by the normative model) [12] [13]. It could be seen as an unsupervised anomaly detection technique, in which we don't need labeled data.

- **Data Augmentation**. Lack of data problem can be addressed by Data Augmentation techniques, which look for improve our Machine Learning models [14] and robustness of pipelines [4].

- **Improvement of data acquisition**. Recently high-impact FastMRI[1] release from Facebook for improving the speed in MRI scans [15].

- **Image enhancement**. Clinical evaluation is critical for good disease treatment. Experts and algorithms need good quality images to carry out their tasks. This is a problem we want to address, so we will explain it deeper in the document [10] [3].

However, some of these problems overlap. The advance in some of them leads to the advance in another. **Image reconstruction**, which is a mainly sub-problem of image enhancement, could help to achieve better results in data acquisition (i.e. reconstruct the image from less data

[1] https://fastmri.org/

collected), unsupervised anomaly detection (i.e. reconstruction of the disease image differs more than the pathology-free one), data augmentation (i.e. reconstruction from patholohy-free to abnormal and viceversa) and, obviusly, image enhancement (i.e. reconstruction of cropped parts or reconstruction without noise and artifacts). **Thus, the main purpose for this project is to apply this reconstruction techniques for noise reduction (image enhancement) and data augmentation (lesion inpainting),** being another applications discussed for future work.

1.1.2 MRI Image enhancement

We are going to focus on the problem of image enhancement, specifically the problem of image reconstruction. With this reconstruction technique, the noise and artifacts will be reduced, see 1.2, and the image quality will be improved. If we achieved good performance in this task, we would research about how to apply this solution to unsupervised disease detection or data augmentation.

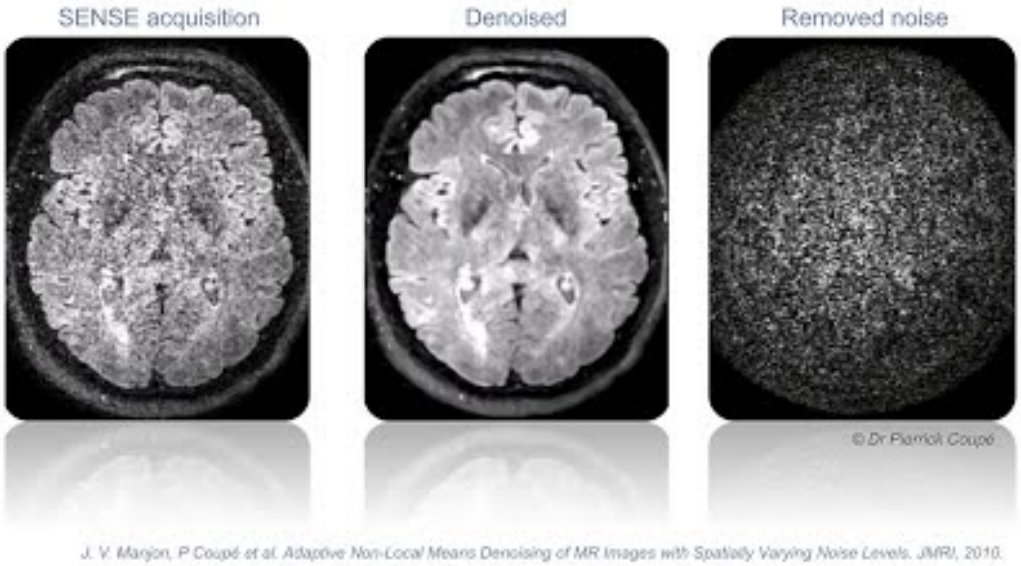

Figure 1.2: Noised and Denoised MRI [2]

Magnetic resonance images are collected with MR scans and the scan process, even though it is improved continually, adds some failures to the MR image. MR images have some random **noise** and **artifacts** due to this fact [16]. This noise and artifacts are present in the image due to different reasons: hardware-reasons (magnetic fields, etc.), body motion during scanning, thermal noise, weak signal intensity (which causes low signal-to-noise ratio), etc. The difference between noise and artifact is that noise can hide the characteristics of an image, whereas

artifacts appear to be characteristic but are not. If the 'problem' is structured, it is probably an artifact, while if it is random, it is probably noise.

1.1.3 Noise and artifact reduction with Deep Learning

Noise and artifact reduction is one of the main principal problems which are classically solved with reconstruction techniques. To address this problem many approaches have been done, all of them with some disadvantages. Advanced filtering methods [17] or retrospective correction approaches have been proposed, but, with the rise of **Deep Learning**, other methods have been proposed that take advantage of this approach. Deep Learning is very powerful in high-dimensional spaces and non-linear problems, so it can make a better job in feature extraction or information compression. Therefore, it will have good performance with images, where the **underlying structure of the images will be captured and foreign elements such as noise or artifacts can be eliminated through a noise-free reconstruction**.

There are some approaches to reduce the noise and artifacts in MRI images. An recent and outstanding review is made by D.Tamada [10]. In this paper D.Tamada summarize Deep Learning Architectures and applications to MRI. We notice the big relevance of denoiser MRI Deep Learning methods in this review. Although there are many methods, we will focus in brain MRI denoisers. A compilation of articles related with noise reduction is made in table 2.2.

There are some Deep Learning Architectures to address in the problem of denoise an brain MRI: Single-scale CNN, Denoising CNN, Autoencoders, and GAN-based architectures. **We will choose the Autoencoder Architecture for this project**.

Autoencoders [18] encode the input into a lower-dimensional space. It extracts important information from the higher-dimensional space, encodes it, and then decodes it to reconstruct the higher-dimensional spacer from the lower one. This architecture will be deeply explained in **??** in further stages of the project.

1.1.4 Our approach

In this project, we will design a **Deep Learning autoencoder for reconstructing brain magnetic resonance images reducing noise/artifacts and inpainting lessons, and study its further implications and applications like data augmentation, disease detection or data acquisition improvement. In other words, we will train a autoencoder with disease-free neuroimaging data and, with this trained autoencoder, we could define a distribution (or normal range) for the neuroanatomical variability for the illness absence with the potential purpose of removing noise or look for diseases. Once trained, the autoencoder should be able to encode a input image and reconstruct it without the noise or lesson.**

We want to highlight one main fact: the autoencoder do not remove noise or inpaint a lessons because it is trained like a denoiser or a image-filler. It do reconstruct MRI without noise or lessons because it is trained only with 'clean' and control MRI like target images, so it does not know how to reconstruct noise or lessons. Therefore, once the MRI is encoded in the latent space, the decoder will reconstruct a control clean MRI from this latent space.

If the main objective of the project is completed, we could research the application of this model in data augmentation and disease detection fields. In the case of **data augmentation**, we will then attempt to reconstruct magnetic resonance images from patients with brain pathologies, with a further view to using the autoencoder to generate 'pathology-free' versions of the said images. In the case of **disease detection**, we could take an approach like the one in [19] (creating a measure for the difference between input and output image and classify it as healthy or non-healthy based on this measure). Patients with pathologies will be outliers in the distribution build by the autoencoder (they will be out of normal range defined by the autoencoder) [12] [13]. This assumption of patients as outliers (based in [13]) is used in [19] for abnormal brain structure detection.

Of course, we will need **data**. For this project, we will use **T1-weighted MRI images of control subjects** (no disease). As the project is of fairly limited length, we won't need to detect disease as a principal objective, but only learn to reconstruct normal MRI images, so we won't need pathology images for this project. In addition, we will investigate whether our method of reconstruction can filter out noise and/or artefacts. So, in essence, we will have n 3D MRI volumes (n can be any number greater than 100) from healthy subjects, we will

preprocess it (data augmentation based on adding noise, remove parts...), and train our model to reconstruct the source MRI volumes. We will have access magnetic brain resonance images from control subjects for training the autoencoder. This data is arranged from different sources. We will use a set of T1 weighted control brain MR images for training the autoencoder. Then, we will consider to use another datasets in the experimentation stage:

- The IXI dataset[2] for training.
- Other data sources such as Open Neuro[3] for experiment with autoencoder applications (reconstruction from disease image, etc).

1.2 Personal motivation

My personal motivation to carry out the project arises from several factors. My first steps in the world of Machine Learning were in the last year of my career at the University of Burgos. I was lucky enough to collaborate last year with the ADMIRABLE[4] research department. The project that I did (in which we continue working) was on the use of biomarkers extracted from the voice for the construction of classifiers that detect Parkinson's disease[5]. The project include topics like **signal processing**, **supervised learning**, **unsupervised learning** and **transfer learning**. The project was very successful and we had a lot of impact at that time. We are currently in the process of meeting with the Burgos hospital to continue developing the model and the application (project and impact recompilation in Github[6]).

This project has fully opened me the doors the world of artificial intelligence and machine learning, which is a field of knowledge that I love. I have always liked math, problem-solving and since I started my career I love programming. Therefore, I find this field the ideal that aligns with my tastes and interests. As I said before, I have done lot of jobs with supervised learning or data analysis, but only with tabular data sets or text-datasets, so I wanted to break in the world of image processing and Deep Learning.

Then I worked half year in Ernst and Young, developing Machine Learning systems for RPA tasks (classify emails at Telefónica, Chatbot for Maxium or Fuzzy Name Matching for Xunta

[2] https://brain-development.org/ixi-dataset/
[3] https://openneuro.org
[4] http://admirable-ubu.es/
[5] https://adrianarnaiz.github.io/TFG-Neurodegenerative-Disease-Detection/
[6] https://github.com/AdrianArnaiz/TFG-Neurodegenerative-Disease-Detection

de Galicia).

In addition, I believe that the application of AI to medicine is one of the fields that may be of greater general interest to society. By advancing in the speed and quality of medical diagnoses and treatments, it will be possible to achieve health of higher quality, speed and accessibility for all. Also, computer vision and deep learning have helped to achieve big advances in this field nowadays.

Chapter 2

State of art: related works

2.1 Overview

The world relevance and impact of this problem is also shown in the related articles of this subject. The state of art of Deep Learning applied to brain MRI shows the relevance of this field. As we discussed in the introduction, in section 1, different deep learning techniques have been used to address the problems derived from brain MRI images: **classification healthy/disease, tumor segmentation, optimize data acquisition, data augmentation and image enhancement** are the principal ones.

> Nevertheless, we must highlight that all of this problems have common points of works. One of them is the purpose of our project: **learn MRI representation for reconstruction**.

The advance in some of the questions leads to the advance in another. **Image reconstruction**, which is a mainly sub-problem of image enhancement, could help to achieve better results in:

- **Data acquisition**
  - Reconstruct the image from less data collected: faster scanning process [15].
- **Disease detection and segmentation**

  - Unsupervised Anomaly Detection: detect diseases with non-labeled data: reconstruction of the disease image differs more than the pathology-free one [19].
  - Tumor segmentation (widely known as BraTS [20]): encode for extract deep image features and decoder for reconstruction of dense segmentation mask [3].

- **Data Augmentation**
  - Construction of pathology-free image from abnormal and viceversa [4] (i.e. lesion inpainting).
  - Artificial MRI Generation [14] [21][1] .

- **Image Enhancement**
  - Reconstruction of cropped parts.
  - Reconstruction without noise and artifacts [22] [5], [23], [24].
  - Definition enhancement: from low resolution to high resolution [25] [22].

We want to emphasize the actual relevance of this project. Solving MRI problems using Deep Learning isn't just about how to apply Deep Learning to another field. It is not just a Deep Learning experiment to demonstrate the power of this method. Solving problems with MRI diagnostics (classification, segmentation), MRI quality (MRI enhancement, data augmentation), or MRI acquisition are cutting edge issues in both the field of Computer Science and Healthcare (neuroimaging, neurological analysis, etc.).

2.2 Related works

We realize this in the overwhelming number of articles using different Deep Learning architectures for solving all kinds of problems with MRI. We will discuss papers addressing different problems but with one similarity: use of image reconstruction in some part of the process (preferably by using autoencoder-based solution). **However, the main purpose for this project is to apply this reconstruction techniques for noise reduction (image enhancement) and data augmentation (lesion inpainting).**

The main evidence of the big relevance and collaboration between Deep Learning and MR imaging is **FastMRI by Facebook AI**. In fact, lately, the focus is on **improving MRI**

[1] https://paperswithcode.com/paper/generation-of-3d-brain-mri-using-auto

acquisition, with techniques based on collecting fewer data and using reconstruction techniques with Deep Learning with the aim of improving image quality and acquisition speed. The high-impact in the academic field of these kind of studies is based on Facebook AI works. Facebook AI is focused on **accelerating MR imaging** with AI, and it is his main goal in healthcare nowadays. They created fastMRI[2] [15], a set of models working with some benchmark datasets in order to accelerate the MR imaging acquisition. It is open source, and you can participate in the challenge[3] with data from New York University. Recently, Facebook and NeurIPS[4] announced that the best models and projects presented for this purpose, even from groups outside Facebook, will be invited to NeurIPS, one of the most important conferences on Neural Information Processing Systems.

To continue with the different studies using reconstruction methods for distinct purposes, we describe the use of reconstruction for helping **Tumor Segmentation**. This is another main problem in the state of the art. There is a global academic challenge using labeled brain tumor MRI for BRAin Tumor Segmentation called **BRATS** [20]. This competition is compound by a MRI dataset from T1, T1c, T2 and FLAIR MRI and the goal is make the segmentation of the distinct parts of the tumor. Using this data as a benchmark[5], lots of different groups are making experiments each year to improve the results. One of these studies using reconstruction techniques is the current best outcome for BRAST 2018: **A. Myronenko** [3][6]. Although their objective is the 3D segmentation of tumors, they use a curious architecture, shown in figure 2.1, that incorporates an encoder and two decoding branches: one for the creation of tumor segmentation masks and the other for the reconstruction of images. This branch of image reconstruction is only used during training as an additional guide to regularize the encoder part. The encoder is made by **ResNet** blocks (Group Norm+ReLu+Conv). The decoder is a **variational autoencoder** (VAE) made of the distribution layer and deconvolutional upsampling layers with Group Normalization and ReLu. 2 more parts are incorporated in their main loss function for tumor segmentation: Mean square error and Kullback–Leibler divergence of the reconstruction branch.

Another problem is to **classify whether an image belongs to a control or a patient**. A recent approach is based on the construction of normative models [12], and , therefore, the image reconstruction based in this normative model. **Pinaya et al**. [19] use this technique to identify abnormal patterns in neuropshychiatric disorders towards achieving **unsupervised**

[2] https://fastmri.org/
[3] https://fastmri.org/submission_guidelines/
[4] https://sites.google.com/view/med-neurips-2020
[5] https://paperswithcode.com/task/brain-tumor-segmentation
[6] https://paperswithcode.com/paper/3d-mri-brain-tumor-segmentation-using

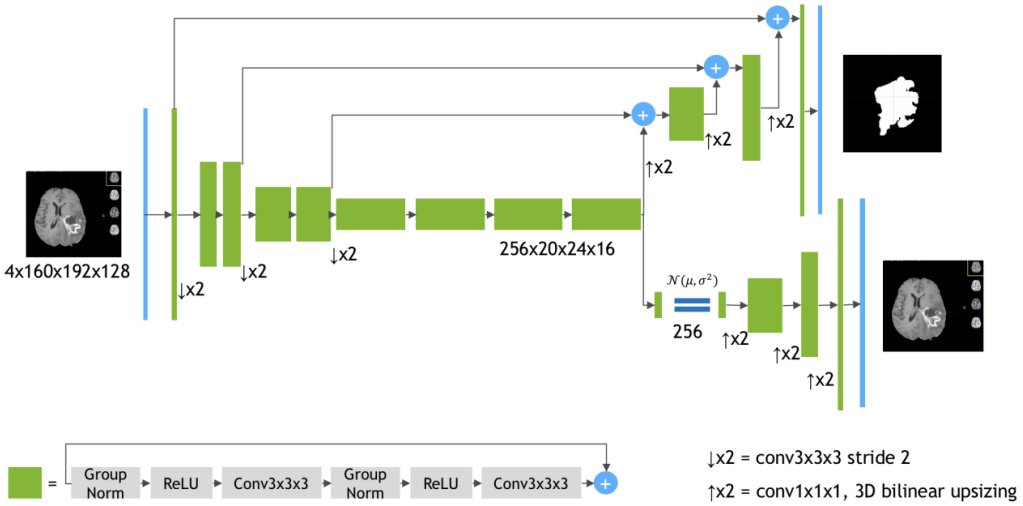

Figure 2.1: Architecture of ResNet-VAE-based network of A. Myronenko. [3]

anomaly detection, so we don't need labeled images from disease data. Classic methods and approaches based on **sMRI** (structural magnetic resonance imaging) can't get a good performance in abnormal brain structural detection because neuroanatomical alterations in neurological disorders can be subtle and spatially distributed. Another approach based on Machine Learning methods could improve performance because algorithms are sensitive to these subtle characteristics. The downside of this road is the need for a large amount of image data (control and disease) and that the models are black-boxes with no information on the critical characteristics used for the decision. They developed a **Deep semi-supervised Autoencoder**, which put unsupervised anomaly detection up for discussion. The goal of that study is to build an autoencoder which encode the structure of control brains. It means the autoencoder learn the normal distribution for healthy brains and the abnormal MR images would be outliers in that distribution. With this autoencoder defining a distribution for control patients, they define a **deviation metric** to measure the neuroanatomical deviation in patients. Patients with some disorder should be outliers in this distribution. The architecture and technique used in the experiment is the following:

- Architecture
  - Semi-supervised autoencoder: reconstruction of the image and prediction of age and sex.
  - 3 hidden layers with SELUs activation function.
  - Output layer with Linear activation function.

  - Loss function: MSE from reconstructed and original image + cross-entropy for age + cross-entropy for years + Unsupervised cross-covariance.
  - 2000 epochs.
  - ADAM optimizer (adaptive moment estimation) with adaptative learning rate.
  - 64 samples mini-batches.

- Transformation of input data:
  - Add Gaussian noise to image (0, 0.1).
  - Feature scaling (normalization).
  - One-hot encoding for `sex` and `age` labels.

We continue with a special case: **lesion inpainting** [4]. It can be seen as a **data augmentation** task (reconstructing 'pathology-free' versions from patients with any brain disease). In the work of **José V. Majón et. al.** [4] the medial purpose is **the improvement of the behavior of current brain image analysis pipelines**. These pipelines are not robust to brain MR images with lesions. For example, a task such as brain part segmentation decreases its accuracy when dealing with lesions. They proposed a **3D UNet** like network to map the image with lesion to the inpainted image (target). The encoder is made by 3 blocks of a 3D Convolutional Layer with ReLU activation, Batch Normalization and max-pooling. For the decoder they used same architecture but upsampling instead of max-pooling and, in the last step, a tri-linear interpolation layer followed by a 3D convolution layer (with 8 filters) plus a ReLU and Batch normalization layers for upsampling the image. We can see the diagram pf the architecture in figure 2.2. Everything was trained with MSE loss function. They use lesion masks to generate artificial training data. The use control cases masked out with lesion masks using the software lesionBrain [26].

Besides of all of these principal studies and objectives, there are so many more. Théo Estienne et. al.[7] [27] in 2020 realize a project based in the study from A. Myronenko [3] which we explained before. They also research Deep Learning architectures for tumor segmentation (BraTS 2018) and use a autoencoder-based network with 2 decoder branches: one for tumor segmentation an another for reconstruction. This last branch is only used for encoder-regularization. They use a fully convolutional VNet architecture, with convolutional layers, ReLU activation function and a intra-block residual connection with the output of the first activated convolution of the corresponding block. They use also direct connections from encoder to decoder part.

[7] https://github.com/TheoEst/joint_registration_tumor_segmentation

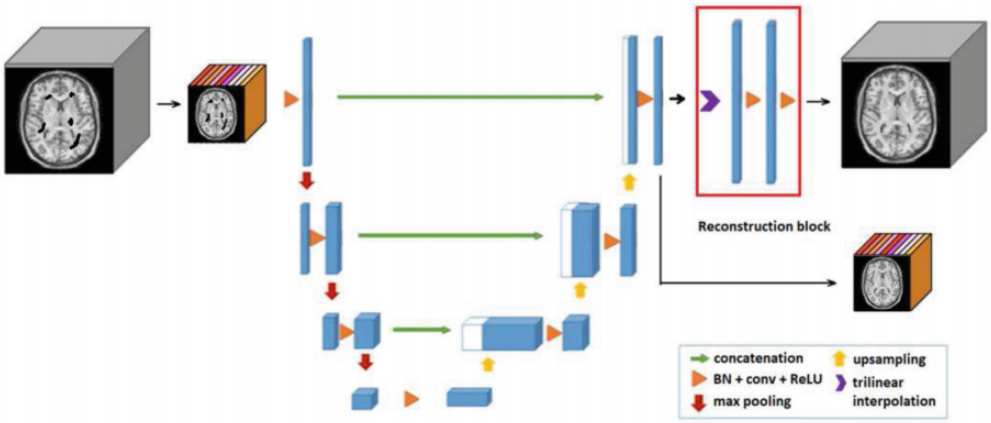

Figure 2.2: U-Net based architecture of network of J. V. Manjon et. al. [4]

We discover another study from Evan M. Yu et. al. [28][8] in which they try to learn volumetric representations from different parts of brain structure. They also use an autoencoder framework.They architecture is composed by 2 components: a spatial transformer network (STN) and a convolutional autoencoder (CAE). The autoencoder is a kind of ResNet-based one, because it uses residual blocks with skip connections, instance normalization and Leaky ReLU activation function.

In order to finalize with the review of the studies of reconstruction applications in MRI, we want to explain one last paper. This work is not focused in MRI, but it achieves very good performance in many tasks like restoration, denoising, super-resolution or image inpainting. XJ. Mao et all [22][9] published in 2016 an study about CAE with symetric skip-connections to address those objectives. They also use a residual based network which they call RED-Net. The main characteristics of this network are the skip connections (in which one layer from the encoder are added up to its symmetric layer in the decoder) and the lack of pooling layers. They don't use pooling layers due to pooling discards useful image details that are essential for these tasks. They use MSE loss function. Peak Signal-to-Noise Ratio (PSNR) and Structural SIMilarity (SSIM) index are calculated for evaluation.

The studies and tasks just explained are the prominent and recent ones but there are other areas and analysis in which MRI restoration could help. Some of them are survival prediction [29], disease progression [30] or brain connectivity analysis [31].

[8] https://paperswithcode.com/paper/a-convolutional-autoencoder-approach-to-learn
[9] https://paperswithcode.com/paper/image-restoration-using-convolutional-auto

2.3 Volumes or slices?

Brain MR images are stored in volumes. It means it are stored as 3D volumes representing somebody's head. Some projects directly use 3D volumes to achieve the purpose (i.e. 3D tumor segmentation or 3D reconstruction). It is more complicated get good results in 3D than in 2D, because results are more relevant in the field.

When working in 2D we have to consider another decisions. The first one is **what profile of the volume should we use**. Brain MRI Volume has 3 different views: **axial** (from above the head), **sagittal** (from the side of the face, profile) and **coronal** (from behind the head). We can see the different views in figure 2.3

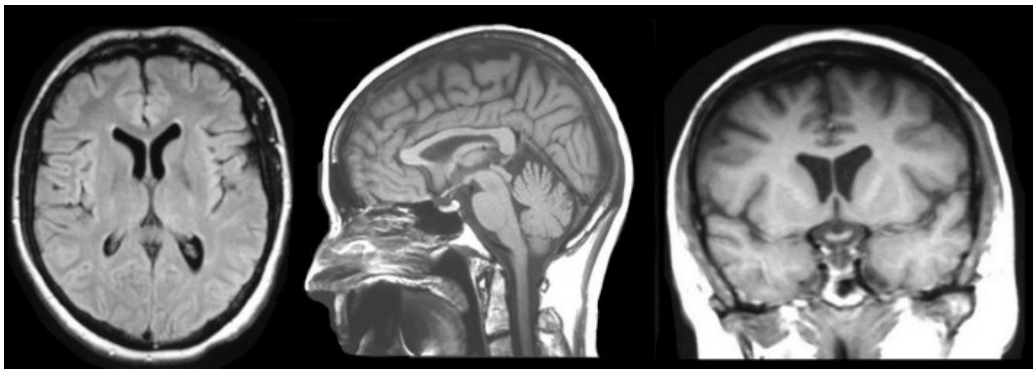

Figure 2.3: Left: axial. Middle: sagittal. Right: coronal

For non-isotropic acquisitions, we should ideally slice them so that the slices are high resolution. For example, if the **voxel** resolution is 1x1x5 mm^3, we should slice the volume so that the slices are 1x1mm^2rather than 1x5mm^2 (or 5x1mm^2). The other issue to address is **what 2D slices from the volume has relevant information**. Some of the slices are slices of the extremes of the volume, and it didn't represent relevant information about the brain structure-

In this project, we are going to work with 2D slices due to time constraints. Therefore, in this section, we will discuss whether the projects mentioned above have used volumes or images, from which volume profile they have taken the images and how the ones containing important information have been chosen. Our approach will be explained in SECTION **??** in next stage (development).

A. Myronenko [3] uses 3D volumes for the brain tumor segmentation task (BraTS 2018) with 1x1x1 mm isotropic resolution and size 240x240x155. Consequently, he does not have to get any profile or select slices. Pinaya et. al [19] use T1 weighted images, thus 2D slices. They don't say neither the profile used in the images, nor the method to select 2D slices with relevant

information. To continue, Manjón et. al [4] propose "the first 3D blind inpainting method in medical imaging" as they say. They use the same dataset that us (IXI), they work directly with 3D volumes, so neither profile nor slice election is done. They only preprocessed the volumes in order to normalize the voxels in to 1 mm3 voxel resolution. More recently, Théo Estienne et.al [27] also address the problem of 3D tumor segmentation of BraTS 2018. Evan M. Yu et. al. [28] uses 3D volumes of OASIS dataset to learn volumetric shape representations for brains structures.

As we can conclude, most projects address 3D volumes because his relevant implications, both in medical field and in deep learning field. However, we have found another articles in which they work with 2D slices. **C. Bermudez, et al** [5] uses 2D **axial** slices from BLSA dataset. All subjects were affine-registered to MNIs-space and intensity-normalized before 2D slices are selected. In addition, the only select a **single midline axial slice** from each volume. Finally they had 528 images with size 220 x 170 voxels. They use this 2D images to build 3 denoiser autoencoders with skip connections (one autoencoder for each level of noise added to train data). The architecture of this 2D denoiser brain mri autoencoder is shown in figure 2.4.

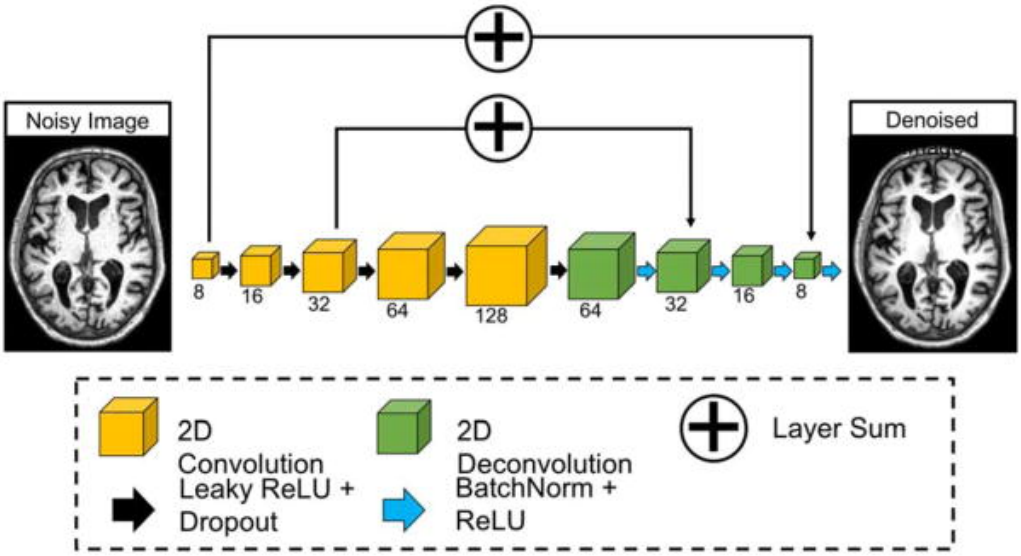

Figure 2.4: C. Bermudez et. al. Denoising CAE+skip-connections architecture [5]

2.4 Network architectures for images

The encoder part of the autoencoder should work as a feature extractor, so we can research the most known Deep Learning Architectures for images in order to use the same architecture or realize transfer learning (with frozen weights or not). We will start explaining the first approach to image processing with Deep Learning, then we will continue with residual network architectures (commonly used in related works) and we will conclude by explaining U-Net and V-Net, the other 2 commonly used networks for medical image segmentation.

2.4.1 Alexnet

First of all, we are going to introduce **Alexnet** [9], a deep convolutional neural network for image classification created by Alex Krizhevsky et. al in 2012 which, in that moment, was the best approach for ImageNet classification, improving by 10% of difference with the second best. It is very important because they apply Deep Learning and convolutional networks to the classification of images and they establish a standard from which new networks are created by adding improvements. The architecture, shown in table 2.1, is made by 5 convolutional layers and max pooling layer after convolutional 1, 2 and 5 layers, and 3 dense (fully connected) layers as classifier part. Rectified linear unit is used a a activation function (ReLU). Softmax function is used in the last layer in order to represent a probability distribution over the image classes. So AlexNet is characterized by **convolutional layers, max pooling layers, dense layers as classifier and ReLU activation function**.

Since the moment Alexnet appeared, the improvements made were almost everyone about go deeper in the layers and architecture. However, going deeper didn't solve another problems like vanishing gradient.

2.4.2 Residual networks

We realize that most of the works of the state of the art uses residual networks [3] [28] or networks with skip-connections [4] [22] [5], so we will introduce deeper than other architectures.

The emergence of this network rises from deep plain neural networks (as Alexnet) problems on training. Despite the general belief that the more layers these networks have the more it learns, an experiment made by He et. al. [6] shows that, in some cases, more layers are related

Layer	Map	Tensor size	Kernel size	Stride	Activation	Params
Input Image	1	227x227x3	-	-	-	-
Convolution 1	96	55x55x96	11x11	4	relu	34,944
Max pooling	96	27x27x96	3x3	2	relu	0
Convolution 2	256	27x27x256	5x5	1	relu	614,656
Max pooling	256	13x13x256	3x3	2	relu	0
Convolution 3	384	13x13x384	3x3	1	relu	885,120
Convolution 4	384	13x13x384	3x3	1	relu	1,327,488
Convolution 5	256	13x13x256	3x3	1	relu	884,992
Max pooling	256	6x6x256	3x3	2	relu	0
Dense 1	-	4096x1	-	-	relu	37,752,832
Dense 2	-	4096x1	-	-	relu	16,781,312
Dense 3	-	4096x1	-	-	relu	4,097,000
Output Dense	-	1000	-	-	Softmax	-

Table 2.1: AlexNet Architecture [9]

to less accuracy even using regularization techniques like L2 or Dropout. Deep plain networks suffer from performance degradation due to the loss of detail in deeper layers and vanishing gradient problem

This residual learning approach is introduced by Kaiming He et. al. in 2016 [6] with the goal of reducing the complexity of deep neural networks. The main innovation of residual networks is that the layers "learn residual functions with reference to the layer inputs, instead of learning unreferenced functions" [6] as they said. This network obtains better performance than before with less complexity in training.

In plain networks, the layers are built to approximate a mapping function from the image to a target: $H(x)$. This is equivalent to approximate the residual of this function: $F(x) = H(x) - x$ and then get the original function as $F(x) + x$. This kind of layer is called **Residual Block** in which we estimate the residuals and then we add the original input to get our original target function. We can see the architecture of this block in figure 2.5 , which is the picture of the original work. Thus, a **skip-connection** is include in the network. **This only consist in adding the input of a stack of layers to the output of this stack of layers**. We have to notice $F(x)$ and x could not have same dimension, so we have to multiply by a linear projection the original input x for matching dimensions.

This paradigm of residual learning gives rise to other ResNet-based architectures, which

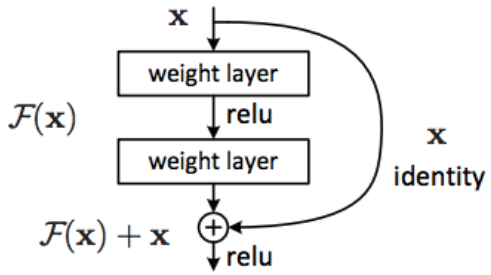

Figure 2. Residual learning: a building block.

Figure 2.5: Residual Block Architecture [6]

differs in number of layers and building block compositions. This kind of networks uses residual blocks (there are also several residual block architecture) with skip-connections embedded in some architecture. Two main examples are the ResNet34 and Resnet50 networks, which use different kinds of residual blocks. In addition, He et. al. presents some differentes ResNet architectures which we show in figure 2.6.

In addition, an Autoencoder made with residual blocks made by A. Myronenko [3] is shown in figure 2.1, in which green blocks represent residual blocks with group normalization. Residual blocks are also used in Yu et. al. [28] work.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.6: Resnet architectures presented in [6]

2.4.3 Other skip-connection-based architectures

As we explained before, deep plain networks suffer from performance degradation due to the loss of detail in deeper layers and vanishing gradient problem. Residual networks address this problems through residual blocks. But there is a inner idea in ResNets which other architectures also use: **skip-connections**. In ResNet skip-connections are added in order to estimate the residual function, but these kind of skip-connections can be added to networks with another purpose. Other networks like some Fully Convolutional Networks also use this skip-connections. Other networks like some Fully Convolutional Networks also use this skip-connections. For example, FCN-8 architecture has skip connections, in which some feature maps of the earlier layers are added to later layers.

Skip connections from one layer of the encoder to it symmetric layer of the autoencoder are added to the architecture with 2 purposes. First, to allow the signal to backpropagate straight to the lower layers and thus address the problem of gradient disappearance, facilitating deep network training and, thus, achieving improvements in restoration performance. Second, when we build a deeper network, low-detail of the image could be lost, making deconvolution difficult in recovering task. However, the skip connections pass through the feature maps which carry much image detail and helps deconvolution to recover the original image.

Some related works use this kind of architecture. C. Bermudez et. al [5] uses skip symmetric connections as can be shown in figure 2.4. This is a convolutional autoencoder with Leaky ReLU, in which the skip-connections add a layer in the encoder to their symmetric layer in the autoencoder (Like FCN but being symmetric).

However, some concrete architectures of CAE + skip-connections have been established due to their good results in some tasks. U-Net [7] is one of them, and a U-Net-Autoencoder-based architecture is used by Manjon et. al. [4] which we can see in figure 2.2.

2.4.3.1 U-Net and V-Net

U-Net [7] is one of the networks that has skip-connections between symmetric layers. U-Net is a Fully-Convolutional-based Network (FCN) that is mainly used for image segmentation, that's why Manjón [4] use a V-Net based architecture for inpainting. The principal 2 differences between a FCN and U-net are the **symmetry** of U-NET and the skip connections between the downsampling (encoder) path and the upsampling (decoder) path which use **concatenation**

operator instead of sum (sum operator is used in skip-connections in Fully Convolutional Networks).

Each of the 4 blocks used in the downsampling path is made up of 2 convolutional layers (3x3) with batch normalization and ReLU and another 2x2 MaxPooling layer. This block extract advanced features while reducing feature maps sizes. In the upsampling route, the 4 blocks used are made by a 2x2 upconvolutional layer and 2 other convolutional layers like those of the downsampling route to recover size of segmentation maps, the concatenation of the feature map of the symmetric layer of the encoder to give the location information from the encoding path to decoding path and a final 1x1 convolution. The original architecture of U-NET paper is illustrated in figure 2.7. As we can see, this original U-Net architecture is very similar to Manjon et. al. U-Net based autoencoder; see figure 2.2.

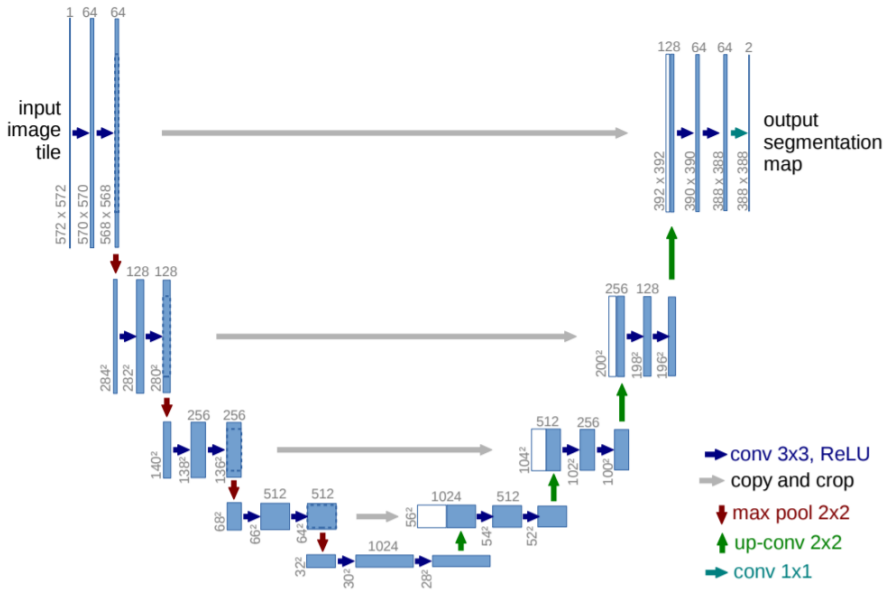

Figure 2.7: U-Net architecture of original paper: O Ronneberger et. al. [7]

U-Net has suffered some modifications, one of the most important is **V-Net**. This network was introduces by F. Miilletari et. al [32]. V-Net is used for volumetric biomedical image segmentation and it gets a very good performance in this task. This network is used in the work or T. Estienne [27] to create a V-Net-based autoencoder to solve Brats 2018 challenge. V-Net combines the skip-symmetric-connections with residual blocks. It means that the encoder and decoder parts are built with residual blocks instead of convolutional blocks as U-Net. V-Net also changes the size of kernels and convolutions in respect to original U-Net, but the main change are the residual blocks. So, V-Net combine the 2 types of skip-connections we have spoken in this research of state of art: residual blocks and larger skip-connections between

symetrich layers.

2.5 Summary of related works

We have compiled some recent and prominent works in which they use reconstruction methods for different purposes. We have generally explained their architectures and approaches.

Although all the collected architectures are based on autoencoders, it has its differences in how the autoencoder is built. First of all, it differs in the main architecture of the blocks of the autoencoder. Different kind of networks like **ResNet, UNet, VNet, Simple CAE** or **AlexNet**. Also, in some of them use the Variational Autoencoder approach, even in [3] combine ResNet and VAE. Furthermore, there are additional architecture characteristic in which studies differ.

Regardless of the main architecture, one feature is shared by all related works (works that use Deep Learning for MRI reconstruction): **skip-connections**. In the works that use ResNet-based [3] [28] architectures, skip-connections are implicit in residual blocks. In this architecture the skip-connection is used for represents the residual estimation function. In U-Net [4] and CAE with skip-connections architectures [5] [22], the skip-connection is used with 2 purposes: allow the signal to backpropagate straight to the lower layers and pass the image details from the convolutional layers to the deconvolutional layers, fighting the low-detail loss [22]. In conclusion with the architecture research, **the related work use skip connections architectures: ResNet-base and CAE+Skip-Connection-based (U-Net-based and more)**.

The **loss function** used is also a critical issue. Most of the studies researched use pixel-wise MSE, which implicit improves the evaluation metrics $PSNR$ and $SSIM$. In addition, other metrics are also used. $KLdivergence$ is added to loss function when VAE is used or $cross-entropy$ and $cross-covariance$ are added when semi-supervised autoencoder is used [19]. However, we will discuss in this project the benefits of using $PSNR$ or $SSIM$ directly in loss function.

We are going to work with **2D slices of 3D brain MRI volumes**. It means that in our project we are going to reconstruct 2D brain MR images. In order to get 2D images from volumes, we have to get slices, as we can see in image 2.8. We can get many 2D images from one brain volume. So, in the preprocessing step, we firstly must choose **what brain MRI view we are going to work with**. For non-isotropic acquisitions, we should ideally slice them so

that the slices are high resolution: select slices where voxel dimension remains equal for the 2 slice dimensions (i.e. 1x1x5 mm3 should transform into slices of 1x1 mm2 and not 1x5 mm2).

But main step in choosing 2D slices is not that, main step is choose slices which retrieves relevant information. A volume can be seen as a 3D head and some slices (i.e. from the sides) can not retrieve relevant information, because it will retrieve noise or bone parts, but it don't show information about brain structure. We can see this fact in the image 2.9, in which is shown the same volume of the image 2.8 but different slice. In order to get images with relevant information, we have recompiled some main methods in the state of art: **get fixed number of slices from all volumes, get the middle slice from the volume or develop ourselves a computer vision tool to evaluate the thickness (with opencv)** . Although the first approach seems the simplest, it is the most used for its good results.

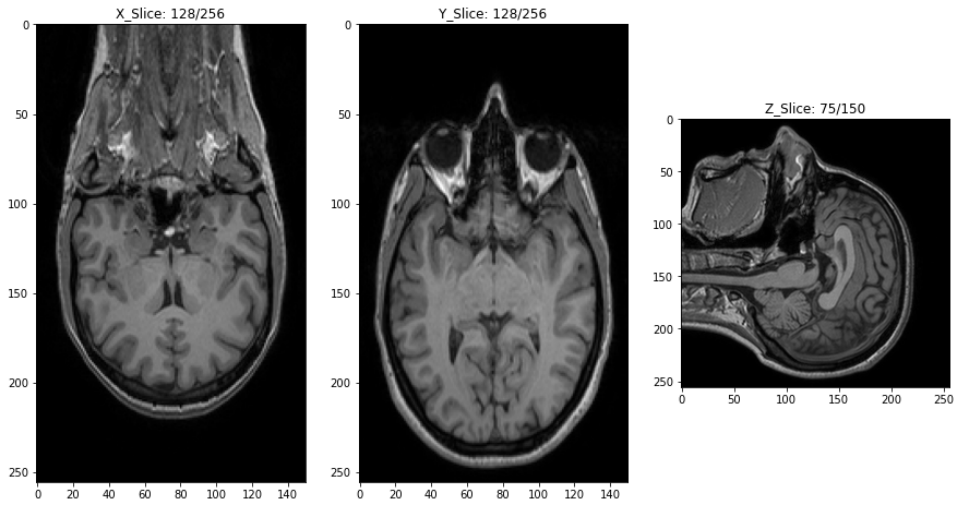

Figure 2.8: 2D slices from brain volume IXI ID 002. Different profiles can be seen. Source: myself

Brain MRI preprocessing differs depending on the objective. First of all, some of the studies use the images of the dataset directly as the **target output** of the network. Other works **enhance the image quality and contrast** before sending it like target output. To continue, other preprocessing can be made when sending brain MRI to the input layers. **Downsampling** the input images could be also useful to reduce the number of parameters of the network. So the resolution of the input image should be balanced between usability (if it is very small is not useful at all) and trainability. We can **normalize the intensity** of pixels (values from 0 to 1 or mean 0 and standard deviation 1) is very important for a neural network. Normalization of the inputs helps the training of the network due to several reasons. One of the most important is that a target variable with a large spread of values may result in large

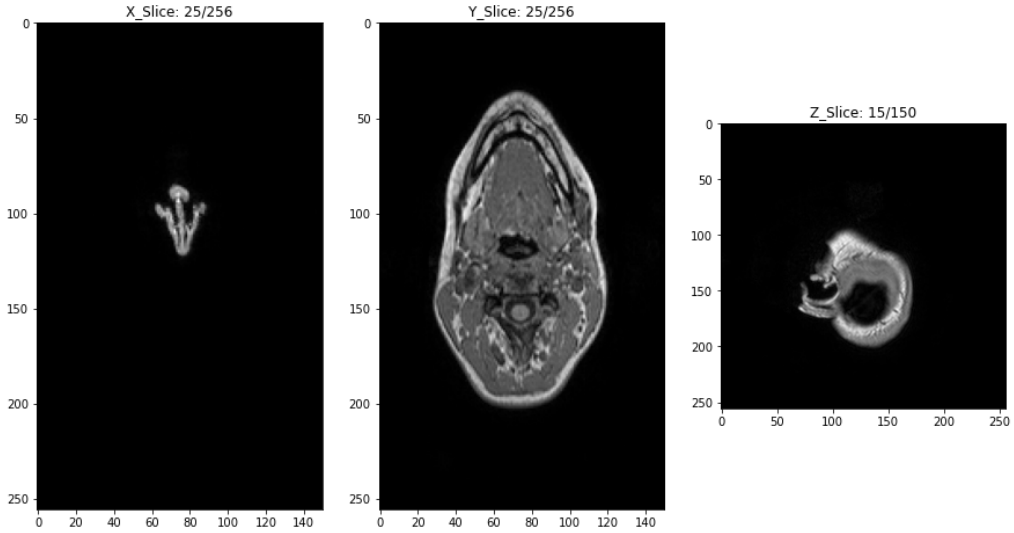

Figure 2.9: 2D slices from IXI ID 002. Slices from volume sides with no relevant information. Source: myself

error gradient values causing weight values to change dramatically, making the learning process unstable [33]. Besides, **affine transformations** like translation or rotation could be applied. In addition, we can realize **data augmentation** in real-time by adding Gaussian noise, other noise or cropping some parts (fixed rectangles or with lesion masks like lessionBrain [26]).

With this research of the art, we are ready to develop the next stages our approach to brain MRI reconstruction. We emphasize that the paper-discovery techniques used in this research are improved by new frameworks described in section 2.6.

Finally, we made a recompilation of papers reviewed in Table 1 of D. Tamada, 2020 [10] and by our own, with some removals and additions based in our goal of the project (see 2.2). We have just deeply explained some of them. From the table of D. Tamada we only obtain 1 autoencoder study [5], 3 sCNN and DnCNN approaches [34] [25] [35] and 1 GAN study. The other papers have been compiled by our own (Autoencoder based: [19] [3] [23] [22] [36] [28] [27], GAN-Autoencoder-based: [24]).

Purpose	Year, Authors	Network
Autoencoders		
Identify brain abnormal structural patterns	2018, W. Pinaya, et al [19]	**Semi-supervised autoencoder with SeLU and loss MSE+cross-variance**
3D Tumor segmentation	2018, A. Myronenko [3] [Code available]	**VAE for regularization to encoder (ResNet like)**
Lesion inpainting	2020, J. V. Manjón et al [4]	**3D UNet autoencoder with skip-connections and upsampling at end**
General image denoising and super resolution	2016, XJ. Mao et al [22] [Code available]	**Residual CAE with symmetric skip connections**
Learn Brain volumetric representation	2018, Evan M. Yu et. al. [28] [Code available]	**STN+Residual CAE with skip connections, IN and LReLU**
3D Tumor segmentation	2020, T. Estienne [27] [Code available]	**VNET Autoencoder for regularization to encoder.**
Denoising for T1 weighted brain MRI	2018, C. Bermudez, et al [5]	**Autoencoder with skip connections**
Medical image denoise	2016, L. Gondara, et al [23]	**Convolutional denoising autoencoder**
Brain MRI denoise	2019, N. Chauhan et al [36]	**Convolutional denoising autoencoder with Fuzzy Logic filters**
sCNN and DnCNN		
Denoising for T1, T2 and FLAIR brain images	2018, M. Kidoh, et al [34]	Single-scale CNN with DCT
Motion artifact reduction for brain MRI	2018, P. Johnson, et al [37]	Single-scale CNN
Denoising for multishot DWI	2020, M Kawamura et al [35]	DnCNN with Noise2Noise
GAN		
Motion artifact reduction for brain MRI	2018 BA. Duffy, et al [25]	GAN with HighRes3dnet as generator
Denoise 3D MRI	2019, M. Ran et al [24]	**Wasserstein GAN with Convolutional Autoencoder generator**

Table 2.2: Overview of studies for reconstruction based in Table 1 from D. Tamada [10] (In bold the autoencoder related architecture)

2.6 New paper-discovery frameworks

In this state of art research, we have use new techniques and frameworks among the classical ones. Connected Papers[10] is a network science framework to improve the search of papers. There are also other platforms like Papers With Code[11] and Distill[12] that improve the experience of article discovering and article visualization-interaction.

I will explain some little examples of paper discovery using this tools. *Connected Papers* retrieve us a graph about the relationship of a given paper. A paper is related with another if one is cited by the other. In addition, the graph contains papers citated by the succesors of the main one. So, the graph contains the most important prior works and Derivate works of our main paper, giving us a perfect tool for paper discovery. One example[13] made for A. Myronenko work [3] is shown in figure 2.10

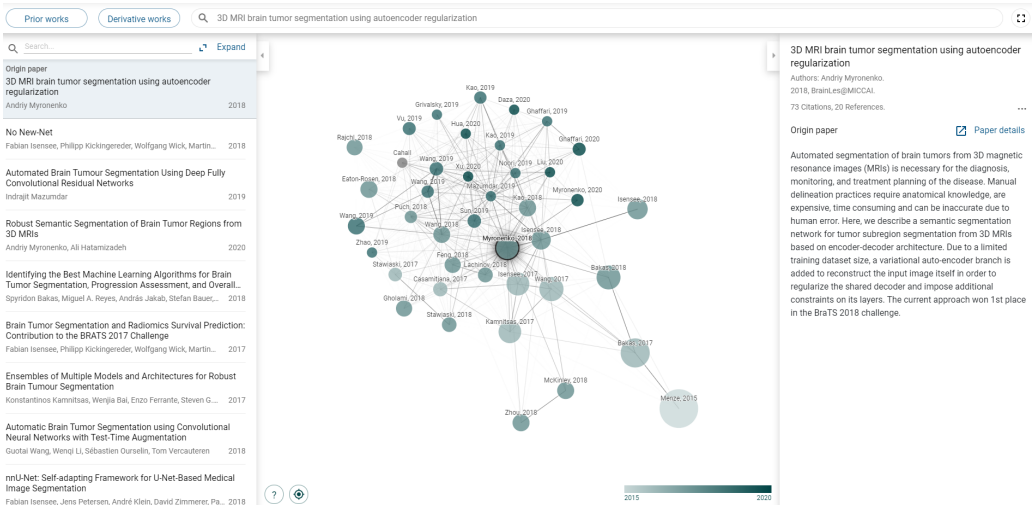

Figure 2.10: Graph of connected papers for A. Myronenko 2018 [3]

Papers with code is a web page, in which are stored papers with it official code implementations. It also group works in different subjects of study, like Medical issues, image segmentation, etc. In addition, it recompile the benchmarks for different machine learning tasks, and it make a ranking of the papers (with the code an pdf linked from the page). We show in figure 2.11 an example of the Myronenko work. We can see the abstract, the link of the paper and multiple links for implementations. Also, we can see the tasks and the benchmark results. There are

[10] https://www.connectedpapers.com/
[11] https://paperswithcode.com/
[12] https://distill.pub/
[13] https://www.connectedpapers.com/main/37a18be8c599b781cc28b6a62d8f11e8a6a75169/3D-MRI-brain-tumor-segmentation-using-autoencoder-regularization/graph

more thing that we can do and research in this framework.

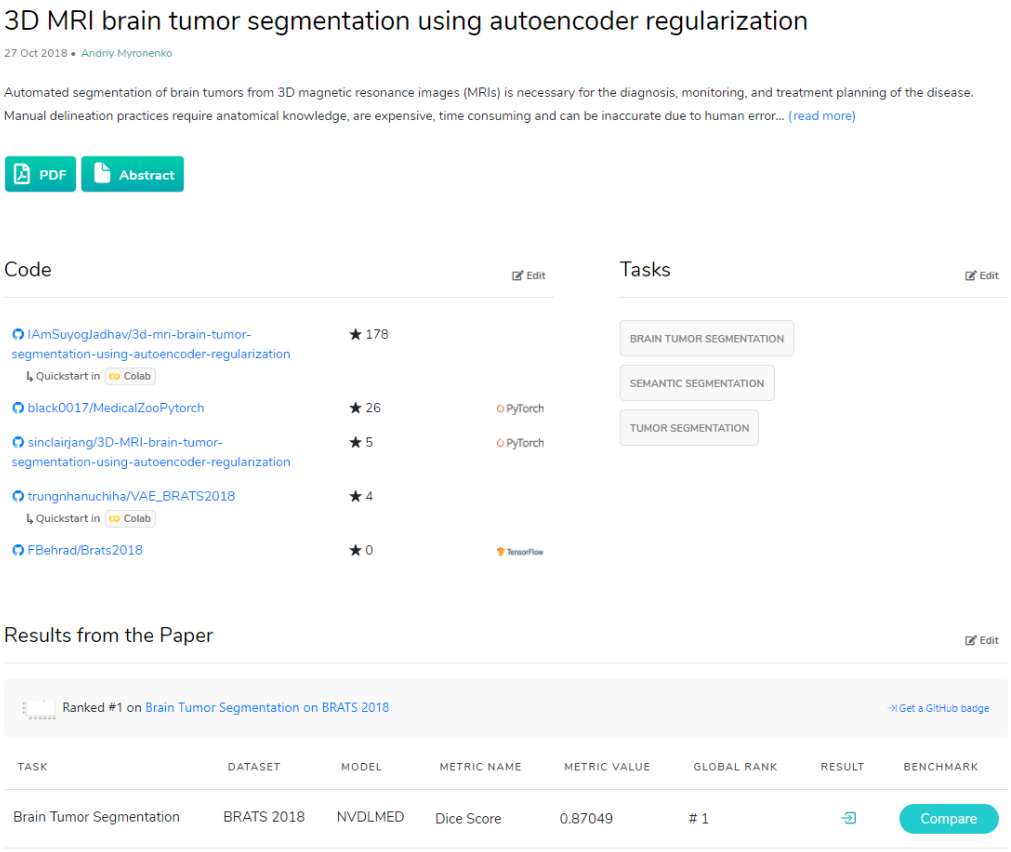

Figure 2.11: A. Myronenko work [3] in Papers with code

Chapter 3

Scope

In this chapter, we will establish the aims of the project. We have just spoken about the problem to be solved. So now we have to enumerate the concrete objectives of the project.

3.1 Hypothesis

We will build an autoencoder for reconstructing T1-weighted brain MRI. It will learn how to encode the underlying structure healthy brains in a lower-dimension space, and reconstruct the MRI from this space. Image quality will be improved reducing noise and artifacts and also could be used for lesion inpainting.

3.2 Primary aims

- To build a **autoencoder** that gets good results with control **T1-weighted brain MRI**: given a "clean" T1-WMRI, the autoencoder will return the same image as equal as we can to the original.
- Due to the nature of the autoencoder train, it will be able to remove noise or reconstruct hidden parts of the input MRI (making data-augmentation in real time).
- Research a good autoencoder architecture and parameters (loss function, batch-norm or not batch-norm, regularization, etc).

- Establish a good brain MRI pre-processing.

3.3 Secondary aims

- Develop the Deep Learning code using one of the most relevant framework, Python, and one of the best-known libraries: Tensorflow and Keras. Also create a very optimized pipeline which could reduce the time of training.
- Use an agile methodology: SCRUM. This methodology should be used in the project. We will use the Zenhub tool of Github as a helper in the project management.
- Research the reconstruct applications for image enhancement, disease detection or data augmentation.

These next objectives will be addressed if the primary ones are reached. We could see these aims like a extra for the project. If we achieved good performance in this task, we would research about how to apply this solution to disease detection or data augmentation.

- Build a semi-supervised autoencoder.
- Build a tumor detection system (based on supervised learning or based in the output of the autoencoder [19]).
- Research the activation filters of our network, in order to dive in the qualitative results.

Chapter 4

Planning and Methodology

In this chapter, we are going to discuss the scheduling for the project and the methodology used in this one.

4.1 Research plan

In this section, we are making a time planning for our project. Planning a project is a very important feature, because we can manage the time properly and we can keep a realistic task-calendar. For this purpose, we are going to elaborate a **Gantt Diagram**. This diagram is a very common resource used in project management [38].

Our diagram is a weekly Gannt Diagram. It has 17 weeks ([mm/dd/yyyy]):

- Week 1: from 09/14/2020 to 09/20/2020
- Week 17: from 01/01/2020 to 01/10/2021

It is built by all the main tasks that a master's degree final project must have and some personalized ones for this project. So we will have 6 big phases derived from project submits.


WEEKS: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Project Selection
100% complete
Scope and planning
99% complete
Title, Keywords and Abstract
95% complete
Overview, relevance and aims
99% complete
Methodology and planning
100% complete
State of art research
100% complete
Search bibliography
100% complete
Search similar code projects
100% complete
Resume state of art, redefine aims
100% complete
Project development
100% complete
Data preparation
100% complete
Deep Learning Design
100% complete
Deep Learning codification
100% complete
FINISH-TO-START
Experiment execution
100% complete
Discuss results, improvements
100% complete
Update documentation
100% complete
Documentation
100% complete
Presentation and defence
20% complete
TODAY

4.2 Methodology

In this section we must choose a common academic Data Mining development methodology, in which there are described the phases, tasks and its relationships.

The description and nature of the project are very helpful at this point because the methodology used in the project will depend on the nature of it. The main characteristic of this project is its research-oriented purpose, so we can label the project as an **academic research project**. Nevertheless, the main objective of this research is to develop a software component (a Deep Convolutional Autoencoder). We can also describe the project as a **software project**. In addition, the project is located in the Machine Learning and Deep Learning areas. These areas are very related to Maths, Statistics, and Computer Science. In all of these fields, the aim is to analyze data in a quantitative way. We analyze how the variables are related, how the autoencoder performance with a concrete measure, how it trains getting concrete metrics (how it learns, time, overfitting...), etc. So our methodology should be **quantitative**. We will take a representative sample of brain MRI, we will train the autoencoder and inference the results to all the population. All this sample and inference techniques are addressed by the validation methods of Machine Learning (Train/test, Cross-validation to reduce bias, etc).

So, due to the nature of the project, we have to apply a methodology for an **quantitative academic research project for data mining software development**.

> In a very summarized way, we will start researching the state of art, defining the problem, and proposing a model to solve the target problem. We will choose and prepare our data. Then we will develop the data mining software solution for this problem, evaluating each step. Finally, we will evaluate our model an get a conclusion for our hypothesis. Thus, **CRISP-DM methodology** embed all of these steps and it will be chosen as the project methodology.

The methodology that best suits our project is **CRISP-DM** [39]. The ***CR**oss-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining* is a framework used for creating and deploying machine learning solutions. Moreover, research and quantitative tasks can be embedded in the CRISP-DM phases (i.e. state of art research phase can fit into business understanding CRISP-DM phase and quantitative evaluation can fit into model evaluation).

As we know, agile methodologies are often used in software development. CRIPS-DM is neither an agile methodology nor a waterfall one. This methodology has clear stages, but the

order of them is not strict and we could move forward and back whenever we need, in order to improve our data mining final model. In fact, this movement between phases is widely used. Also it has a iterative cycle, in which data, data preparation, modelling and evaluation are improved wit the previous iteration feedback.

Figure 4.1 shows the phase dependencies and order. As we can see, the straight lines define the dependencies between phases as in a classical methodology. Nevertheless, We can see the circle and the two-arrowed straight lines that show the flexibility and the agile similarity of CRISP-DM.

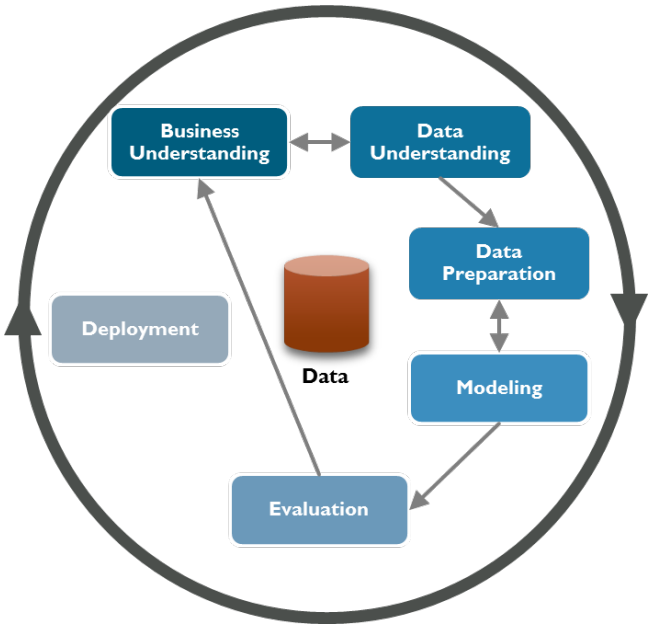

Figure 4.1: CRISP-DM Cycle

The phases of CRISP-DM [39] are the following (In later stages, references will be made to the specific sections of the document where the phases are carried out):

Business Understanding : deep analysis of the business needs. In this phase we can establish an objective. In our case, we can research the state of art for Deep Convolutional Autoencoder for brain MRI and propose a model based on this research.

Data Understanding : we should research the data sources as IXI[1], data quality and we should explore the data and its characteristics.

Data Preparation : Data should be cleaned, filtered, selected and integrated if necessary.

[1] https://brain-development.org/ixi-dataset/

We could carry out tasks like preprocessing T1 weighted brain MRI or realize data augmentation. I will be explained in-depth in section **??**.

Modeling : Specify the model to use and the architecture, parameters, etc. Maybe running several model architecture and hyper-parameter optimization to reach the most powerful model. So in can be an iterative process.

Evaluation : We must evaluate models properly to get meaningful conclusions. There are many techniques of model evaluation and it should be made carefully.

Deployment - Publication : As our main goal is academic research, this phase would be *Publication*. The tasks are: review the project and generate the final report.

Chapter 5

Project development

5.1 Pipeline and overview

We are going to explain a general pipeline of the experiment to set a clear steps to reach our goal. These steps will be deeper explained in the following sections. First, we explore all the original NIFTI volumes. To continue we select the best profile of the volumes to get the 2D images, we check the orientation of the slices and we extract the 2D images with relevant information. Then, we split the MRI volumes into 2 separated sets: train/validation and test. It is going to be a stratified splitting by age, sex and ethnicity. We also check for duplicates and more information about the volumes such as relevant slices of each volume. At this point, we can start training our models. We define different architectures and a custom data loader in order to carry out these experiments with data augmentation in an optimized way. We run experiments with different architectures, with and without data augmentation, with and without L2 regularization and also using MSE and DSSIM Loss functions. Finally, we get the test metrics, we compare them all, and we also compare the models in a qualitative way. This last evaluation shows an intuition on how the models reconstruct corrupted brain MR images.

The development, control version, and planning through issues and Sprints is made with Github and Zotero in the official repository of this Master's Thesis[1]. We will reference the appropriate documents of the repository as we explain the steps.

The diagram of this experiment pipeline is shown in figure 5.1.

[1] https://github.com/AdrianArnaiz/Brain-MRI-Autoencoder

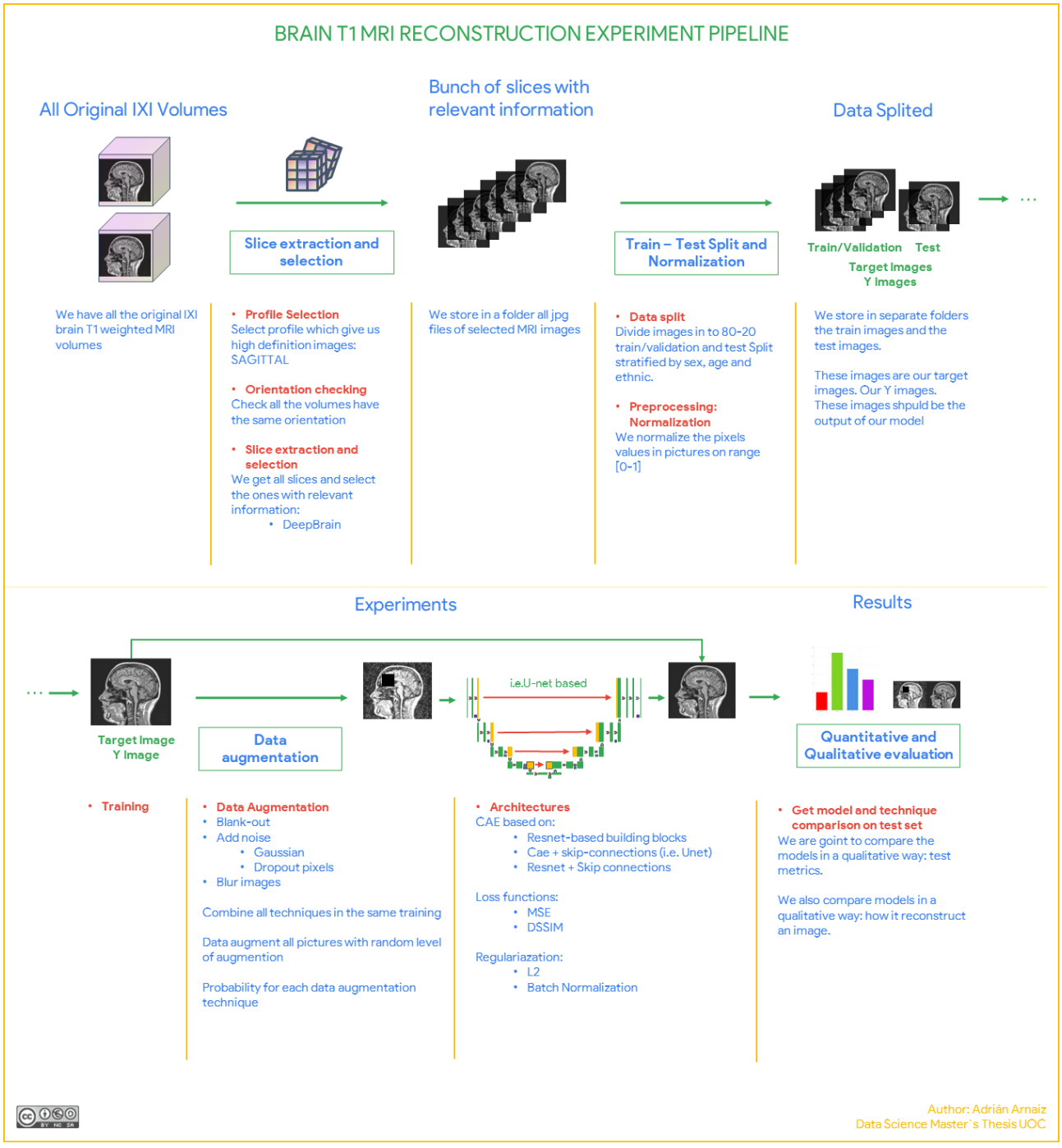

Figure 5.1: Pipeline of the project.

5.2 Dataset

5.2.1 Exploration and preprocessing

We are going to work with IXI brain T1-weighted MRI dataset[2]. This public dataset belongs to Imperial College London and it is composed by MRI images in NIFTI format. We made an initial exploration of the characteristics (could be seen in `/src/0.Set_up/MRI treatment - nibabel.ipynb`[3]) of this dataset and also some deeper ones are discovered while profile selection, orientation checking, selection of relevant slices and data splitting.

IXI dataset has a total of **584 volumes**. The dimension of voxels are $0.9375 \times 0.9375 \times 1.2mm^3$ for 576 volumes and $0.9766 \times 0.9766 \times 1.2mm^3$ for 5 volumes. Although some volumes have different voxels dimensions, every volume is isotropic for 1st and 2nd dimensions. The dimension in voxels of the images are the following: the first 2 dimensions are 256 for every volume, but there are differences in the third dimension between different MRIs. Therefore, there are 503 volumes with $256 \times 256 \times 150$, 74 volumes with $256 \times 256 \times 146$, 2 volumes with $256 \times 256 \times 130$ and last 2 volumes with $256 \times 256 \times 130$ voxels, see figure 5.2. We also realize that if we freeze the first dimension we get the coronal view, if we freeze the second dimension we get the axial view and if we freeze the third dimension we get the sagittal view. We don't care so much about the volume dimension in voxels, but voxel dimension. We don't care so much about the volume dimension in voxels, but voxel dimension. Volume dimension is significant in relevant slice selection, but we are going to use a dynamic and more complex method to improve this step. However, voxel dimension in mm^3 is relevant for profile selection. We discuss this in following sections, providing more information of the data exploration and the application of this exploration to our tasks such as profile selection.

5.2.1.1 Profile selection and orientation checking

As aforementioned, we have to select a profile to slice the 3D volume in 2D images. For non-isotropic acquisitions, we should ideally slice them so that the slices are high resolution. For example, if the voxel resolution is 1x1x5 mm3, we should slice the volume so that the slices are 1x1mm2 rather than 1x5mm2 (or 5x1mm2). Overriding this, we need to be consistent in which orientation we are slicing. In other words, if we are getting axial slices from one volume,

[2] https://brain-development.org/ixi-dataset/

[3] https://github.com/AdrianArnaiz/Brain-MRI-Autoencoder/blob/master/src/0.Set_up/MRI%20treatment%20-%20nibabel.ipynb

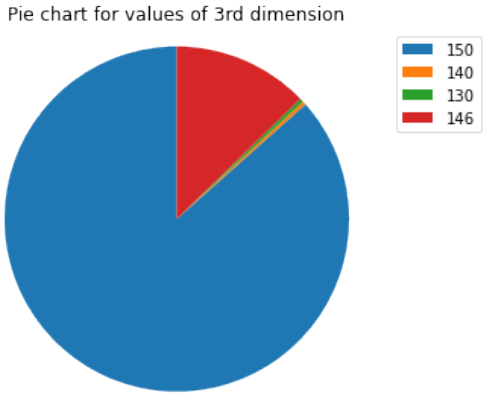

Figure 5.2: Number of volumes for each different value of 3rd volume dimension (in voxels).

we should make sure we get axial slices for all patients. Otherwise, the network will likely not train well. This process is made in `/src/1.DataPreprocessing/0.MRI Profile Selection - voxel and size inspection.ipynb`[4]. On one hand, **we realize that the first and second dimension are isotropic for all volumes, so we will freeze the 3 one to get the slices with higher resolution**. Dimensions of voxels are specified in above section. Therefore, we are using **the sagittal view of the brain** for this project. On the other hand, we check every volume orientation in their headers, **and every volume is in the same orientation**: **P, S, R**. This means that 1st voxel axis goes from anterior to **P**osterior, second voxel axis goes from inferior to **S**uperior and third voxel axis goes from left to **R**ight. Getting images with sagittal orientation, **we have a total of 86794 2D images** from the 584 volumes.

5.2.1.2 Relevant slice selection

As we discuss in section 2.3, we are going to use 2D images extracted from 3D volumes. The first step is just explained: profile selection. But we have to make a more important decision: **what 2D slices from the volume are we going to select**. In section 2.5 we show that some parts of the volume do not have any brain portion, and from those parts no relevant information can be retrieved, so we have to select images with relevant information. This is a very important step. This step has a huge impact in many aspects. First of all, the autoencoder will learn the distribution of data that we give it. If we use no relevant images in training, the autoencoder will learn strange distributions. One solution is to be very restrictive in the selection of slices. Some projects **use the single midline slice** from the volume [5], but,

[4] https://github.com/AdrianArnaiz/Brain-MRI-Autoencoder/blob/master/src/1.DataPreprocessing/0.%20MRI%20Profile%20Selection%20-%20voxel%20and%20size%20inspection.ipynb

and this is another aspect in where this step has an impact, the amount of data for training decreases abruptly. Another common approach is to **select a fixed middle range** of slices but we have the same problems. If we select a narrow range, lot of relevant slices are discarded. If we use a wide range, we take a risk of select no relevant slices. This happens because the difference between volumes: maybe in one volume the relevant slices are 20-100 and in another are 50-110.

In this project we will discuss 3 methods of relevant slice selection, 2 of them based on the distribution analysis of intensity pixel values and another one based on the use of a pre-trained neural network used for brain segmentation. All these methods are developed in the notebook `/src /1.DataPreprocessing /1. Preprocessing Intensity Inspection Select Relevant Slicing.ipynb`[5].

Non-zero intensity pixel count method

Our first candidate approach is based on the intensity of the pixels of each image, specifically the count of non-zero values of an image. With this method we suppose that pixels with intensity different from 0 belong to the body, and if we set a proper threshold on that distribution we could filter the images with some amount of brain quantity. We made an histogram of the distribution, in we have measure the non-pixel values of every slice of every volume. The mean of this distribution is 44386.72 with a standard deviation of 8771.458 and a range of [28-64709]. But the significant fact of this distribution is that it follows a similar Normal distribution: **Negative left skewed distribution** (Negative skewness or Right modal). The distribution is shown in figure 5.3.

This distribution has a clear definition and it is really nice to see it. We had big hope in defining the outliers of the distribution and solve the relevant slice selection with this approach. But we discovered that discarding only the -3STD outliers or the Q1-1.5*IQR outliers were bad approach because lot of irrelevant data pass through the filter. Because of this we were more restrictive: we discard the 25% images with less nonzero pixels (Q1, light blue vertical line in figure 5.3). We improved the performance, but the filter discarded few images in some volumes and too much images in other ones. This method is very inconsistent: the brain can not be identify by non-zero pixel values. This happens because the apparition of noise and strange bone structures at the sides of the volume (sides of the head of a patiente). This noise or bone structure is counted as non-zero value, so this images pass through the filter. The volumes where images are clean of noise could suffer too much discards. Examples of proper and bad

[5] https://github.com/AdrianArnaiz/Brain-MRI-Autoencoder/blob/master/src/1.DataPreprocessing/1.Preprocessing-IntensityInspection-SelectRelecant-Slicing.ipynb

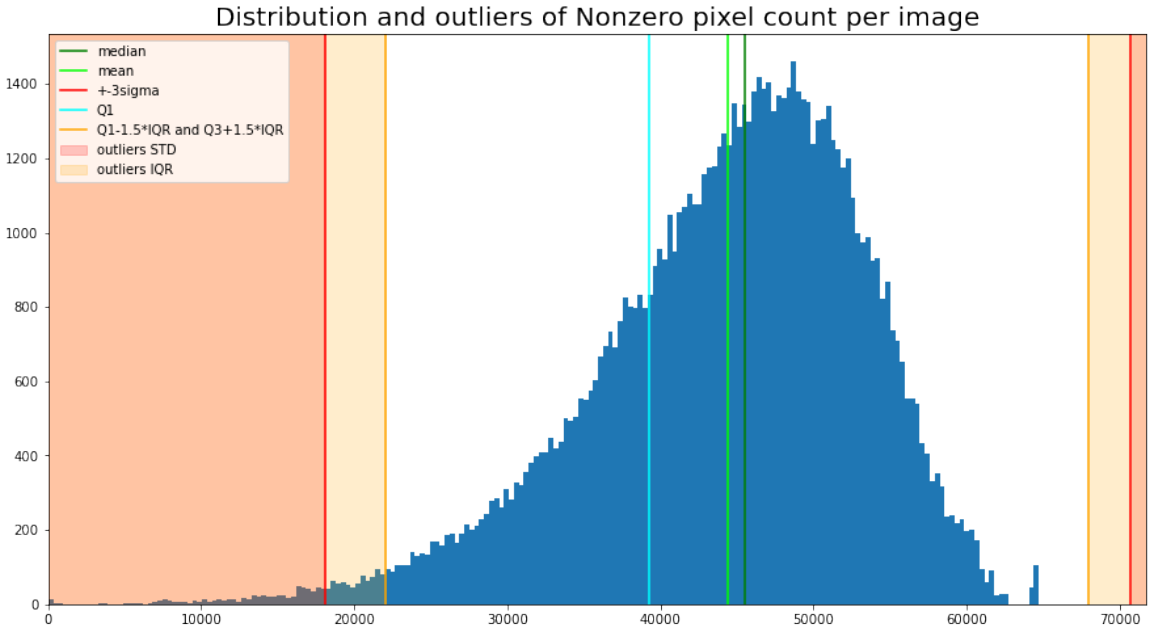

Figure 5.3: Distribution of Non-Zero pixel count per image.

filtering could be shown in see figure 5.5.

Non-Zero pixel mean intensity method

We define another approach to improve the last. We suppose that the noise has a low intensity value, thus the mean of intensity of Non-Zero values could be used as a better proxy of how much relevant information an image has. With this approach we define the distribution. The distribution of the mean intensity for Non-Zero pixels has a really strange shape. It could be similar to a power law, a very long right tail distribution. But no clear low-outliers could be defined to discard the irrelevant images. Distribution is shown in figure 5.4. Then, we define an low arbitrary threshold for the mean intensity value of Non-Zero pixels to determine whether an image is relevant or not. We set this threshold to Q1, as same as the above method. This method is more accurate than the Non-Zero count method as can be seen in . However, with this non-zero pixel mean intensity method, in some volumes too few images are discarded, adding irrelevant information to train dataset, such as volume IXI337 that can be seen in the figure 5.5.

Finally we have saved a DataFrame in pickle format (`/src/1.DataPreprocessing/nonzero_image_data.pickle`) in which we have all the relevant metadata of this two experiments. It has 3 columns: image ID (with format volumeID_sliceIDx), number of of Non-Zero pixels, and mean of intensity values of Non-Zero pixels.

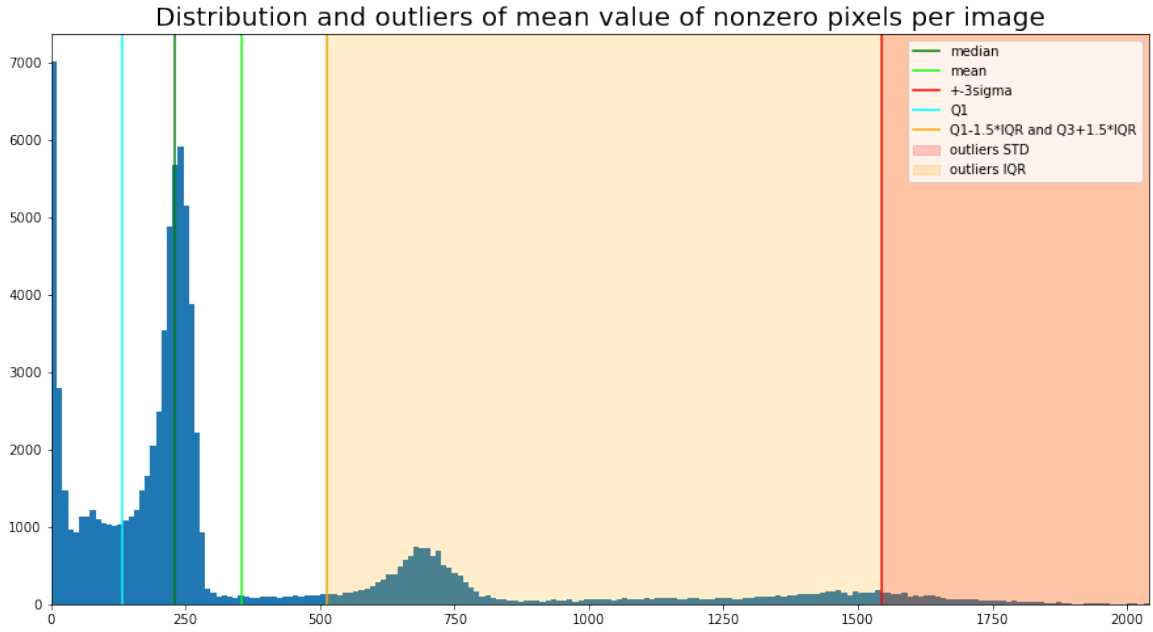

Figure 5.4: Distribution of mean of intensity of Non-Zero pixels per image.

DeepBrain

Summarizing intensity-based methods

- Nonzero pixel count is weak against noisy points and non-brain structures: it fails when there are lots of noisy points like a cloud. It also fails when there are some structures which are not a brain because the nature of the model is only count the number of nonzero pixels.
- Non-Zero pixel mean intensity method is weak against the strange structures: it fails when there are very few points but very shiny. The nature of the model is compute the mean of nonzero pixels, so, method could break when some strange structures or shiny noisy points appears in the images.

The last, better and definitive approach for slice selection is based on **brain segmentation**. We know that the best proxy of how relevant a image is, is the amount of brain a image has because the relevant information is the information about the brain structure. So, Why not estimate the amount of brain straightforward instead of approximate it through intensity-based methods? Brain segmentation is very commonly used in neuroimaging so it would be nice to use an approach like this.

Our method is to extract the amount of brain for each volume and select the slices with

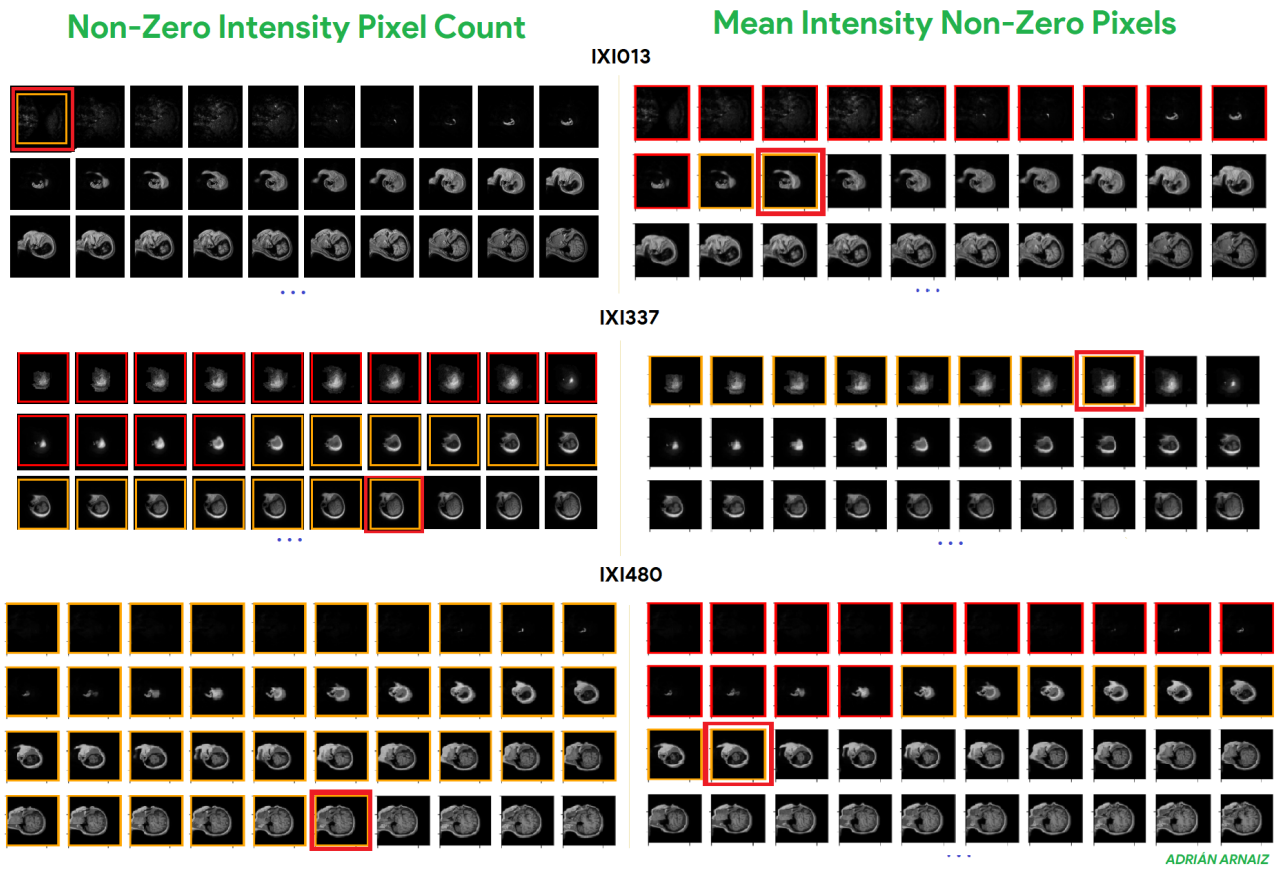

Figure 5.5: Example of discarded images with intensity-based methods in some volumes: both red and yellow framed images are discarded. Double framed (yellow and red) is the limit image discarded. Left column is the Non-Zero intensity count method and right column is the Mean intensity Non-Zero values method.

some brain (or a threshold of brain). The general idea is that there is no better proxy of how much information an image has than the amount of brain that this image has. However, build an accurate neural network for brain segmentation could be very difficult. Therefore, we will use a python library named **DeepBrain**, which uses a pre.trained neural network to perform brain segmentation.

What is the golden tool? DeepBrain[6]. This library returns a matrix with the probability of each pixel belongs to the brain. Then, we could set a mask and, finally (if necessary) segment our original image or, in our case, measure the amount of brain. This tool identify the brain using the whole volume, thus, is more accurate than treating a single slice. Another advantage is that only takes 3 seconds in getting the mask from a whole volume. As DeepBrain return

[6] https://github.com/iitzco/deepbrain

a probability for each volume, we will count the True ($p > 0.5$) values in the mask (number of pixels belonging to the brain) as a proxy of how much brain is there in the image. Some examples of DeepBrain brain segmentation are shown in figure 5.6

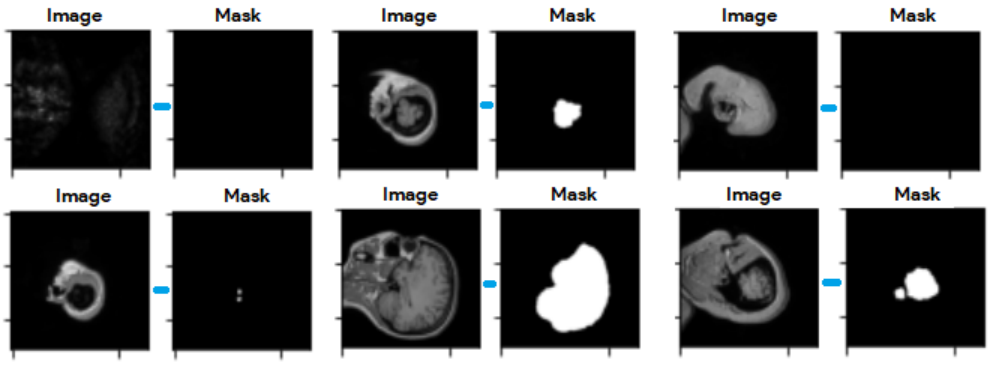

Figure 5.6: Example of DeepBrain segmentation.

We are going to set a lower threshold of brain quantity to select the images: 4.5% of brain quantity which is approximately 3000 pixels. With this threshold, we ensure that at least 4.5% of the image pixels belong to the brain and we could retrieve relevant information from the slice. With this filtering, **we reduce the 2D image dataset from the original 86794 to 59278 relevant images**. This method is very accurate and outperforms the intensity-based ones. We show the image selection with DeepBrain method in figure 5.7. We realize this method filters just the relevant ones in contrast to intensity-based methods that filter less or more than necessary.

We have coded some notebooks and scripts to develop this. We have extracted a pandas DataFrame in pickle format, with the brain quantity for each slice with format [volID_idSlice - Brainquantity (i.e IXI562-Guys-1131-T1_72, 16485)] named `deepbrain_image_data.pickle` created in `src/1.DataPreprocessing/1.Preprocessing-IntensityInspection-SelectRelevant-Slicing.ipynb`. It is also store in csv format in `src/IXI-T1/slice_brain_quantity.csv`. The final slice selection is done in a script called `deep_brain_slice_selection.py` which also use a developed Class named DeepBrainSliceExtractor from module `deep_brain_slice_extractor.py`. DeepBrainSliceExtractor Class params are: path where .nib volumes are stored and output path to save the PNG images, 2 DataFrame with test and train volume IDs and other DataFrame with the data of the brain quantity of each slice, in this case, the `deepbrain_image_data.pickle` explained above. The result relevant images in PNG are stored in `src/IXI-T1/PNG/test_folder/test` and `src/IXI-T1/PNG/train_val_folder/train_and_val`. Data splitting is explained in section 5.3.

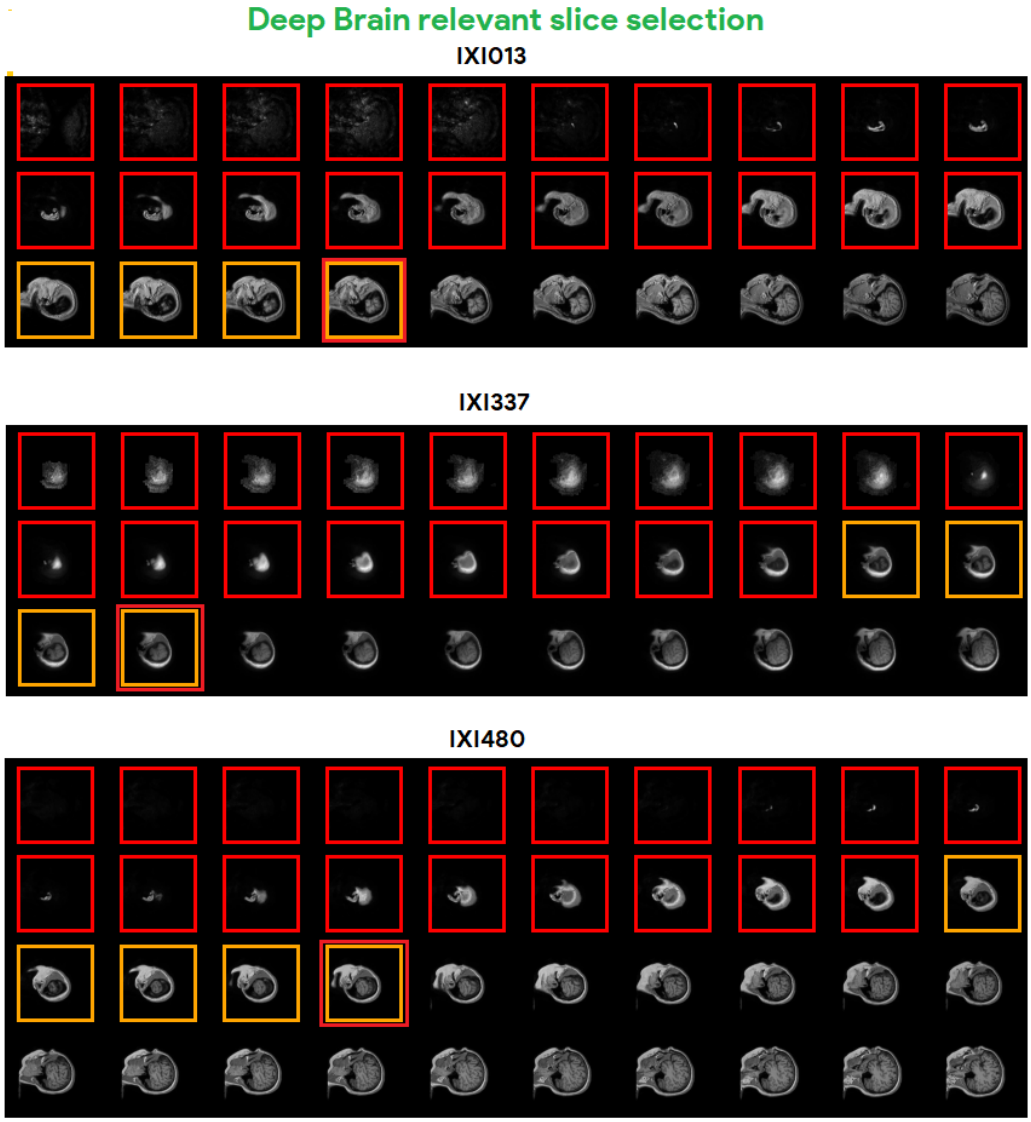

Figure 5.7: Example of discarded images with DeepBrain method in same volumes as intensity based methods: both red and yellow framed images are discarded. Double framed (yellow and red) is the limit image discarded. Red framed images has o brain pixels, and yellow framed has 0-3000 brain pixels.

5.2.2 MRI Preprocessing

As aforementioned, relevant images extracted are stored in PNG format with size $256x256$. Images could be preprocessed in may ways to potentially improve their quality. We made a small tutorial about some example ways of MR image potential enhancement using computer vision techniques (histogram equalization, center of mass centering, etc). This tutorial is made in the notebook `src/1.DataPreprocessing/0.ExamplesofMRIpreprocessing.ipynb`. A common prepossessing technique used in images is contrast enhancement made with histogram

equalization methods (HE). Histogram equalization sharpens or enhances image features, such as boundaries or contrast, for better graphical display and better analysis. However, although we see better the image, many low details could be modified and these low-details are the critical ones for neuroimaging analysis. It may increase the contrast of background noise, while decreasing the usable signal. Also histogram equalization can produce undesirable effects (like visible image gradient) when applied to images with low color depth [40] [8]. We can see the effects of different HE methods in figure 5.8.

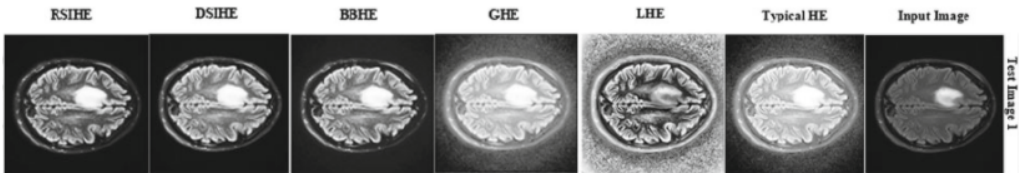

Figure 5.8: Different methods of histogram equalization [8].

Therefore, we conclude that, **even though image enhancement is a way to improve the appearance of image to human viewers, it could be harmful to critical details of brain MR images** and we are not going to apply it to our brain magnetic resonance images.

We only apply 2 preprocessing techniques to our brain slices: **downsampling and feature normalization**.

- **Downsampling**: We downsample the images from $256x256$ to $128x128$ to speed-up the training with bilinear interpolation resizing method . With this size we still identify relevant brain structures.

- **Normalization**: We normalize the intensity of pixels of the image in the range [0-1]. It is very important the compatibility of normalization and activation function of the last layer of the neural network. If the possible range of the output and the range of the input are different, errors will be bigger and reconstruction may fail. Consequently we will use sigmoid function in last layer because it output range is also [0-1].

Both of these preprocessing techniques are made **on-the-fly** with a **customized Tensorflow Data Loader** and with Tensorflow functions (`tf.image` and `tf.math` modules) to optimize the training speed. This Data Loader will be explained in section 5.4.2.

5.3 Data Split

We have to split the 59278 2D images in 3 sets: train, validation and split. We use 3 sets because we are going to several architectures, 2 loss functions and other comparable configuration. A validation dataset is a partition used to tune the hyperparameters (i.e. the architecture, regularization) of a classifier. Another critical aspect of data splitting is the independence of the data between partitions and also the similarity on the distribution of each partition. It comes from the **i.i.d** statement (independent and identically distributed).

We address the **data independence problem**, because exists a huge correlation between the near images of the same volume. For example, slice 45 and slice 46 of the same volume are going to be almost the same. These images, which are almost identical because they are near in the same volume, belong to different sets. This could lead to a big overfitting problem if both images are in different data partitions. If this happens with more similar images, the problem is obviously bigger. Therefore, we do not split the dataset straight from images. It is better to split it from volumes to avoid this big correlation.

We also address the **identically distributed problem**. In this project, we obviously have no target label. However, we can stratify the example images through their metadata. We suppose that physical features have a potential impact in the brain structure, so the stratification by these features should be positive for the training, thus, we would like 3 data partitions retain the same distribution regarding *sex, ethnic* and *age*.

In order to do this stratification, we follow these steps (made in notebook: `src/2.Experiments/0.TrainTestSplit.ipynb`[7]): We map our images and *Nifti* volumes to their metadata. We realize that were duplicated volumes, so the final number of volumes were **581**. We also discover that 18 out of 581 different volumes do not have any metadata, so the images of this 18 volumes will be placed in training set automatically. The remaining 563 volumes. The age distribution on the total population is 73.4% of adults [25, 65), 19.2% of elderly people [65, inf) and 7,4% of young people [0, 25). Sex are distributed like 55.6% male and 45.4% female. Finally, we have 7 values for ethnic, very skewed for white people, with a 80.11% of the volumes, also there are ethnic values with only 1 or 2 volumes, so we only divide between white and others. Last has the remaining 19.89%. We developed a module with 3 functions to help in the composed stratification task. It has been coded in the file `src/2.Experiments-/stratifier_complex.py`, and we have 2 functions: the first to get the report (quantitative

[7] https://github.com/AdrianArnaiz/Brain-MRI-Autoencoder/blob/master/src/2.Experiments/0.TrainTestSplit.ipynb

table) with the number of samples of the stratification, and another function to get the stratified sample. This last function is the used for getting the samples. We divide the total dataset in 2 sets: one for train and validation (which will be splitted in train and validation set in the training script) and another for test. The IDs of each partition are stored in separated pandas DataFrame (as pickle). The columns of this sets are *Volume_IXI_ID, Sex, Ethnic, Age_Group, N_Relevant_Slices*. The files which have the list of IDs for each group are `data_train_val_volumes_df.pkl` and `data_test_volumes_df.pkl`. As we explained before in section 5.2.1.2, these 2 files are needed in the *DeepBrainSliceExtractor* Class in order to make the selection of relevant slices and the stratified partition in the same script.

Once data is extracted and splitted, **the final sizes of each set are showed in table 5.1**. Besides, we can see the correct stratification between partitions. The train&validation/test distributions for each attribute are seen in figure 5.9.

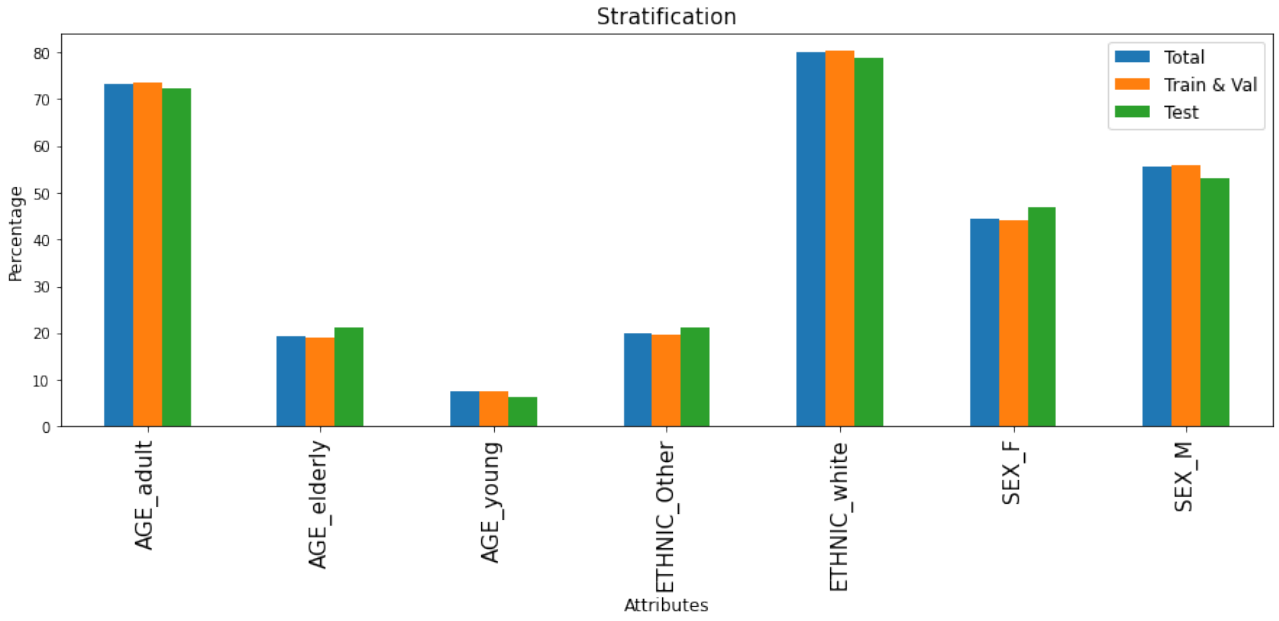

Figure 5.9: Train&validation/test distributions for each attribute to show the correct stratification

Partition	Volumes	Images	Volume%	Image%
Train	454	**46285**	78.2%	**78.1%**
Validation	80	**8169**	13.8%	**13.8%**
Test	47	**4824**	8%	**8.1%**
Total	581	59278	100%	100%

Table 5.1: Data split

5.4 Experiment

In the stage we are going to train several models with different characteristics to obtain an empirical comparison of the different strategies researched in the state of the art: different architectures, different loss functions, different residual building blocks, regularization and no regularization, and finally, the experiment with a new proposed architecture. Before starting the architecture and model explanations, we are going to offer an explanation of the development environment, the customized Tensorflow data loader designed to optimize the speed and ease of the training and the augmentation.

5.4.1 Environment

In the first place, I want to highlight we wanted to run the experiment locally. The main reason is the opportunity to develop a very optimized environment in terms of speed of data-loading and parallelism between training and data loading. We have the opportunity with this Master's thesis because later, when we develop real applications, cloud training is going to be more common, or, in the case of on-premise training, the situation would be more critical. Therefore, **we are going to face the speed optimization problem executing these experiments on local device**, pushing the boundaries of the hardware and giving more importance to the quality of the software developed. This problem could be scaled in future situations, where we will have to train bigger models with tons of data in the cloud.

We are using a HP OMEN Laptop with 16 GB of RAM, a Intel i7-9750H 2.6GHz processor and a NVIDIA GeForce GTX 1660 Ti with 6GB of GDDR6 RVAM. The operating system is Windows 10. With 6GB of VRAM is obvious that all images and weights of the model does not fit on it, so we will develop a custom data loader to read, preprocess and augment the images on-the-fly and in the fastest way. This data loader will be explained in section 5.4.2.

The code was developed in Python language. We have used several libraries, but most important are the ones related to neural networks. The Python version is 3.7.9. We use Tensorflow-gpu 2.3.1 which has the Keras 2.4.0 library built-in the module `tf.keras`. We configured CUDA 10.1 and cuDNN 7.6.5. We have to highlight that every step of the training, included the data loader, data augmentation and customized loss functions are coded with Tensorflow. This aspect make the training faster using only Tensorflow (and Keras) modules and functions due to the fact that Tensorflow builds a graph to optimize the computation and, if external libraries are used, the speed and high-performance decreases. This set-up is also explained in `src/0.Set_up/GPU_TF_enviroment.ipynb`.

5.4.2 Data Generator with augmentation

We faced the challenge of create a optimized data loader, which has to **load batches of images on-the-fly and perform image preprocessing and augmentation in a optimized and parameterizable way**. Fist of all, we created a data loader that loads *numpy* files instead of images, but we discard this approach because reading *numpy* files is slower than reading images.

We compare 3 approaches *Keras ImageDataGenerator.flow_from_directory*, a customized *Keras Data loader* implementing *Sequence* interface and, finally, a customized *Tensorflow Data Loader*, using `tf.data` module. We made a simple experiment (same architecture, parameters and data) to compare all of them. The original *Keras ImageDataGenerator* takes a mean of 5:12 minutes to run an epoch, the customized *Keras Data loader* takes a mean of 2:35 minutes to run an epoch and, finally, our customized *Tensorflow Data Loader* takes a mean of 1:12 minutes to run an epoch. We will explain what we do in next paragraph, but the summary of the reason of the speed of the last data loader is that is coded whole in Tensorflow. First, Tensorflow builds a computational graph, and this is more efficient if all of the steps are made with native Tensorflow functions. We have coded the disk load, the normalization, resizing and image augmentation with native TF functions. Second, the `tf.data` module provides functions to perform parallelism, cache data and other functionalities to load while training and make the train even faster.

So, out data loader is coded in a class named *tf_data_png_loader* in the module `src-/2.Experiments/my_tf_data_loader_optimized.py`. We coded this class to ease the creation of data loaders. With this we only have to create the class and pass the parameters: list of files path, batch size, desired size of the images, if the data is for training and if we want augmen-

tation while data is loading. With this we obtain a object *tf.data.Dataset* which dynamically will load the data in the Keras model. The steps of this DataLoader are the following:

1. We read all the files paths as *tf.Tensors*.
2. Read PNG, resize and normalize image in a parallel way using native TF functions to realize all these steps and calling with a map call to the dataset and the parameter number of parallel call set to AUTOTUNE.
3. We use *shuffle* function to randomly shuffles the elements of this dataset. We also use *repeat* function which is needed to repeat the dataset in training time.
4. We augment the images (if required) on-the-fly and in a parallel way (mapping a *tf.Dataset*). Every augmentation and creation of randomness is made with Tensorflow functions. Data augmentation is explained in section 5.4.3.
5. But the main advantage is made by *batch* and **prefetch** functions. The former allow us to read, resize and augment images in batches. For example, if we had a batch of 16 images, a multiplication is only needed instead of 16. We already know how batches works in the model, but with this function, this improvement is also made in the load and the augmentation. The latter is even more important to speed up the training. *Prefetch* function allows later elements to be prepared while the current element is being processed. This often improves latency and throughput.
6. Friendly remember: **every step is made on-the-fly while model is training or evaluating**.

5.4.3 Data Agumentation

Real time image augmentation is widely used in computer vision algorithms. The data augmentation allow to increase the size of training data and without using any disk space as it is made on-the-fly. This way, every image showed to our algorithm is going to be slightly different each epoch avoiding overfitting. With the proper data augmentation we force to the algorithm to learn how to encode the structure of a healthy brain instead of only copy the input. This leads to better learning of the representation of the healthy brain in latent space. Therefore, avoiding the c̈opyöverfitting, the reconstruction of brain MRI would be better, and we would be able to remove noise or fill empty parts not because the autoencoder remove noise, but because our autoencoder only knows hot to encode and decode the structure of the healthy brain.

Consequently, data augmentation techniques must be aligned with our goal. Adding more and more augmentation techniques is not always helpful. A clear example is in MNIST dataset, where flipping the 9 give us the number 6. For example flipping or rotation is not useful in this project. We only want to know how a brain looks like, how to represent a healthy brain in a latent space of lower dimension, thus, we do not really want our autoencoder to rotate images. In addition, when using an autoencoder like this in a production environment, the one who uses the trained model will to give the model an input image in the correct orientation.

We are going to add randomly 4 augmentations to the image. Each augmentation technique will be added with a random level and intensity. So while training, the model could get from an original input image to a fully augmented one (every augmentation is added at max level). The augmentations are added with functions from **Tensorflow and Tensorflow-addons** libraries. We use pixels **Dropout** with a random value between 0 and 5%. The **Gaussian noise** is applied to the image with a random standard deviation between 0 and 0.04. We **Blank Out** a region with a probability of 20%. The position and size of the black square is also random. It could be in any random position and the size of the side of an square goes from 10 to 40 pixels. Finally, **Blur** is made with a Gaussian 2D filter with sigma 0.6 with a probability of 10%. This augmentation process is made on-the-fly and in a parallel way by customized Tensorflow data loader explained in the section 5.4.2. The examples of augmentations are shown in figure 5.10.

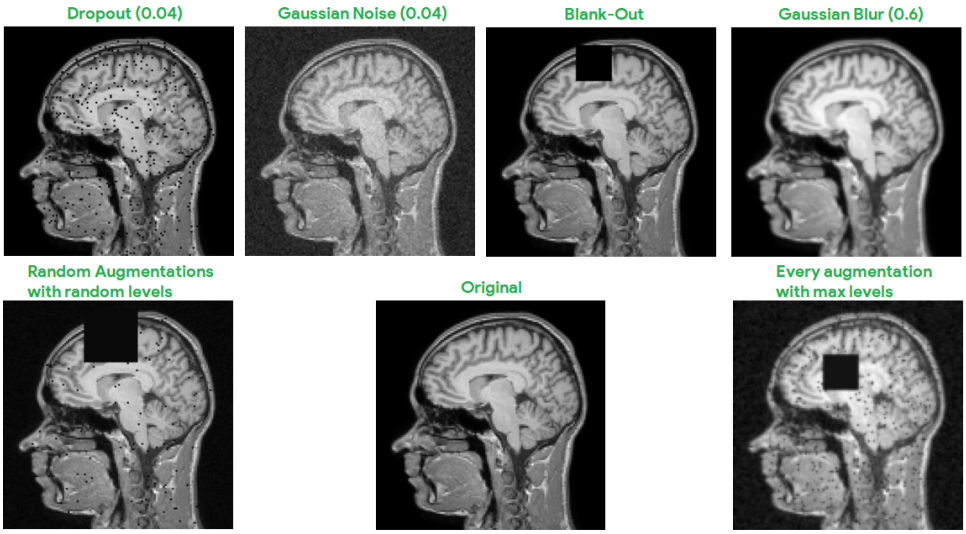

Figure 5.10: Examples of augmented images.

The **value of the current augmentations** added are the following: they cover, corrupt or modify the real brain structures (through dropout, noise, blurring or cuts-out) so the model has to learn to recover this covered or corrupted information. Dropout could represent lacks of

intensity measures for a voxel. The Gaussian noise could represent the deviation in this same measures. Region blank-out could represent corrupted images due to bigger problems or even a small lesson which would be in-painted by the autoencoder. The Gaussian blurred could represent artifacts or lack of definition due to some reasons (like up-sizing),

5.4.4 Architectures definition

This section will explain all the architectures we have trained in our experiments. We have made experiments with several architectures, different losses functions, with and without regularization and with and without data augmentation. Obviously we already know the result of the last comparison (data augmentation), but we want to test it quantitatively and empirically. In this section, we only regard the explanation of architectures, their building blocks, and their foundations. The specific experiments (regularization, losses, etc) will be explained deeper in section 5.4.5.

We wanted to pursue 2 purposes with the experiment. First, and the main one, the **comparison of residual architectures and convolutional architectures with skip-connections besides of the novel benefits of the combination of both** as we explained in the state of art 2. Second, create very shallow optimized architectures. The creation of architectures with fewer and fewer weights through the optimization of the connections and is one of the guidelines of Deep Learning. For example, residual networks come from this topic. We already know that a very deep residual network could lead to a optimal performance, but there are some downsides. First, big networks take too much train to train (even residual, although less than common convolutional). Second, the Internet Of Things breakthrough lead us to search for very light models that could work in these low-memory devices.

Therefore, considering these two guidelines we have defined 5 different [8]architectures with similar decoder architecture.

- Shallow residual autoencoder:
  - Original building block.
  - Full-pre-activation building block
- Skip connection convolutional autoencoder.

[8]Cheat Sheet of all architectures is in https://github.com/AdrianArnaiz/Brain-MRI-Autoencoder/blob/master/ArchitecturesDiagram.svg

- Myronenko Autoencoder: based on the encoder of Myronenko research [3].
- Residual U-NET: architecture proposed in this project and arise from the combination of U-NET and residual building blocks.

We would like to emphasize that neither the architectures use *Max Pooling* functions to reduce feature maps, nor does it use *Upsampling* to increase their size. The former idea comes from the fact that pooling discards useful image details that are essential for these tasks [22]. The latter comes from the intuition that increasing the size of feature maps with a more complex function such as *Conv2DTranspose* could be more helpful to the reconstruction than the *Upsampling* method. So all of them are going to use **strides with value 2** to reduce or increase the size.

5.4.4.1 Shallow Residual Autoencoders

We have discussed the benefits of Residual blocks in the training and optimization of Neural Networks in the state of the art 2. As we explained, benefits comes from the building block, which have skip-connections to force the neural network to learn the residuals. However, there are several kinds of **building blocks** regarding the order of activation function, batch normalization, weights and addition. The **original** one is shown in figure 2.5. But another identity blocks has been researched. A very good research with excellent results was made by Kaiming He et. al. in 2016, concluding that **full pre-activation block** (where BN and ReLU are both adopted before weight layers) throws promising results [41]. Therefore, we are going to 2 architectures with the only difference of the type of building blocks. Both building blocks are defined in figure 5.11. We implemented the building blocks with convolutions of kernel size 3, stride 2 if downsampling is required, and a convolution in the skip connection if we need to match the number of filters. They are also built with *BatchNormalization* Layers and Rectified Linear Unit activation function (***ReLu***) The order of the layers are defined by the type of block.

We built 2 architectures with the only difference of the type of the building block. Obviously, our autoencoders starts with the **encoder** part. This architectures have an initial *Convolutional* Layer with 32 filters with a kernel of dimension $3x3$, stride 2 and padding *same* to downsample the image to half size (64) and increase the number of filters to 32. Then we add a residual building block with downsampling which increases the number of filters to 64 and reduces the dimension of features map to half (32). To continue we add another residual building block

with neither downsampling nor feature map addition, so we keep 64 filters of $32x32$. Finally, we add another residual block with downsampling to reduce the size of the feature maps to 16 and increase the number of filters to 128, thus, this latent space has $16x16x128$ dimension. In this point we start our decoding task with the **decoder**. The building blocks of the decoder are made of a *Con2dTranspose* Layer with stride 2 to upsample the image to double size, followed by a *BatchNormalization* Layer and a *ReLu* as is seen in figure 5.12. Each of 3 decoder building blocks upsample the image to double size and reduces the number of filters in half. Finally, we have a *Convolutional* Layer to reduce the number of filters from 16 to the 1 desired for the output. This last layer has a **sigmoid** activation function to get pixel intensities from 0 to 1, like the normalized inputs.

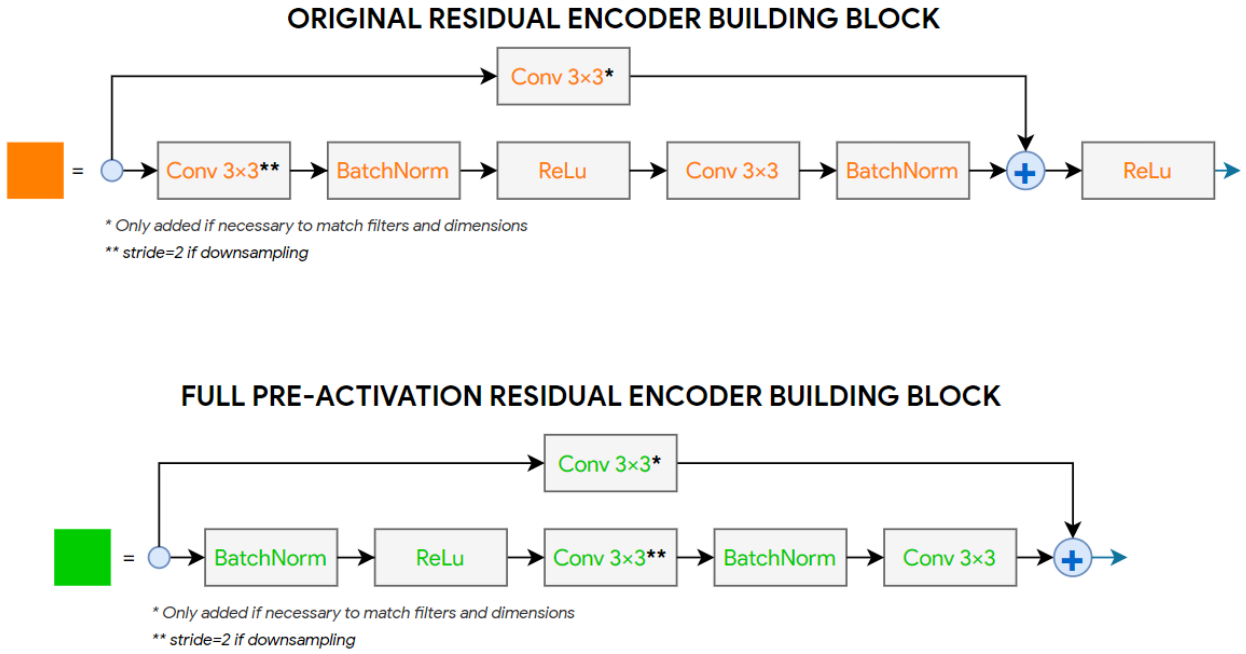

Figure 5.11: Residual building blocks used for residual autoencoders.

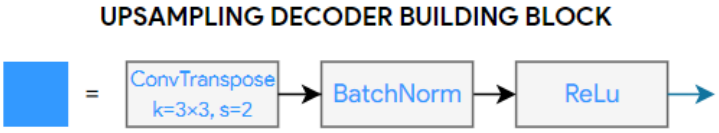

Figure 5.12: Upsampling block used in the decoder.

Both architectures are coded in the file `src/2.Experiments/residual_cae.py`. This script uses flag parameters for allowing simple configuration of what building block we want to use and if regularization is used. The final diagram of both architectures is shown in 5.13.

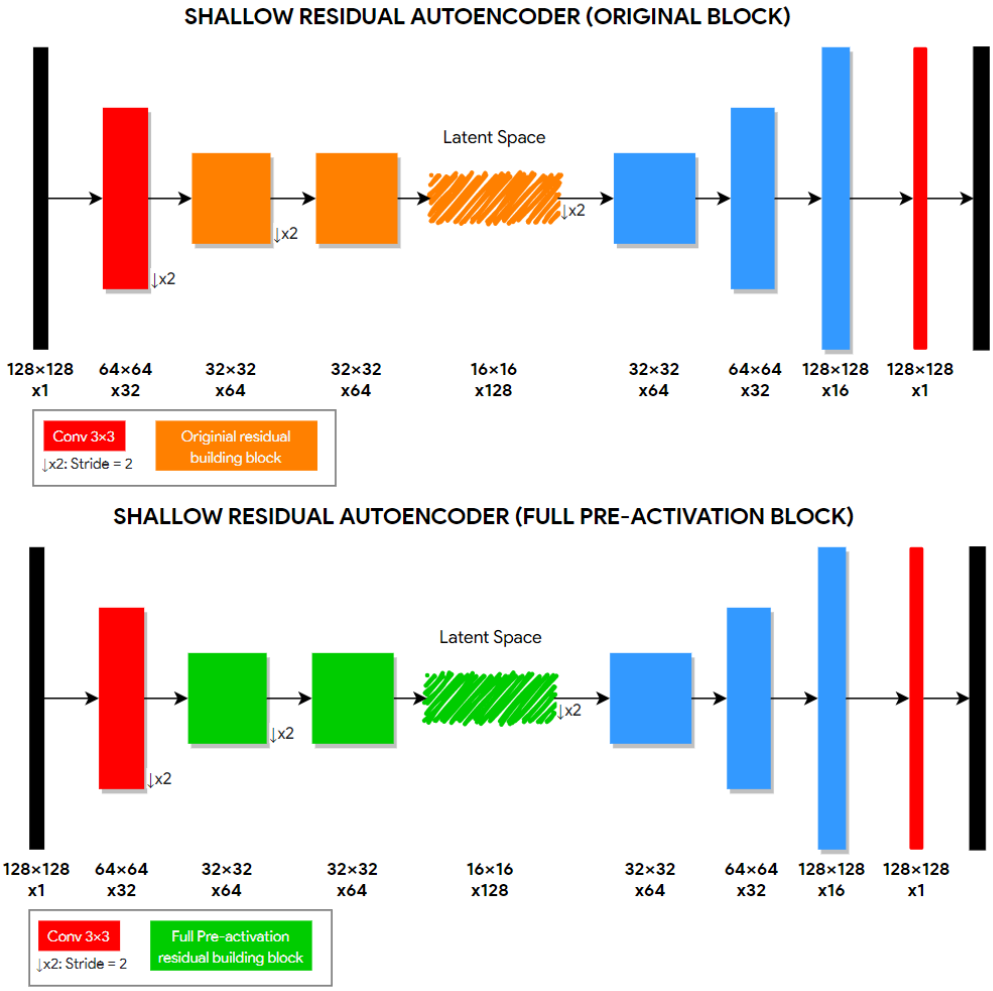

Figure 5.13: Shallow Residual Autoencoders.

5.4.4.2 Myronenko Autoencoder

We also wanted to develop a more complex, but still shallow, residual architecture. With this goal we implemented an Autoencoder based on the encoder branch which Myronenko used in his project for BRATS 2018 [3] to regularization. Although we use our decoder, the encoder part is the same that Myronenko used, but going one less level in the size downsampling and number of filters. Every building block used in this architecture has already been explained in last section 5.4.4.1. This architecture is built with *Convolutional* Layers, full-pre-activation residual building blocks and the already explained upsampling blocks. The main difference is the number of layers and the downsampling order. In this case, the image downsampling is made by a *Convolutional* Layer with stride 2, but outside the residual block instead of inside it like before. This architecture includes also a *SpatialDropout* Layer. Layer configuration (number of

filters, downsampling, number and order of layers, etc) is shown in the Myronenko architecture diagram in figure 5.14 and coded in the file `src/2.Experiments/residual_cae_myronenko.py`. Latent space is $16x16x128$ again.

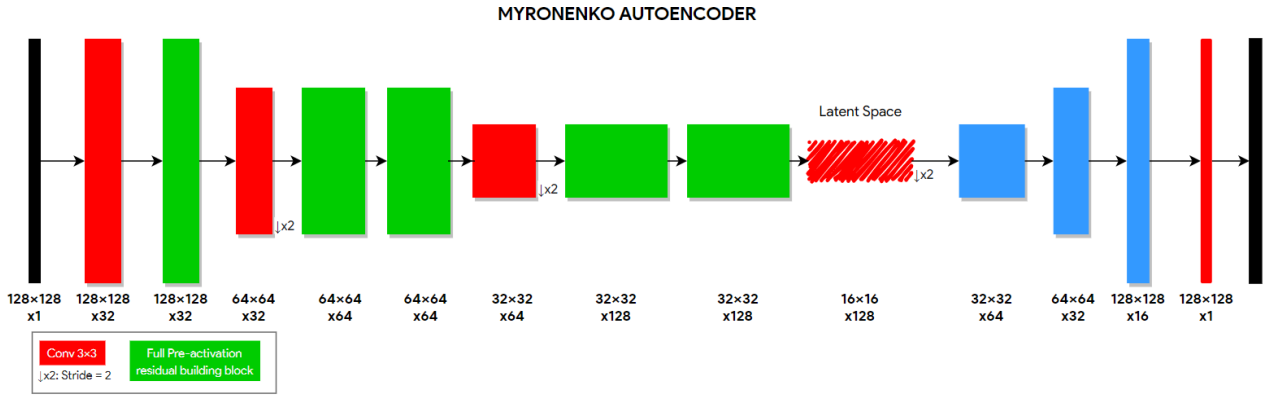

Figure 5.14: Myronenko based autoencoder.

5.4.4.3 Skip Connection Convolutional Autoencoder

The other big advance in segmentation and reconstruction architectures comes from the skip connections. As aforementioned in the state of the art 2, this skip connections are not the ones of residual building blocks. This skip connections are wider and connects encoder layers to decoder layers. Although the interpretation of latent space becomes more abstract if this technique is used, architectures with skip connections have shown a better performance due to the l̈eaköf information to the decoder part, where the decoder will learn how to combine the details of the latent space and the details of the feature maps from the encoder part [22] [4] [19]. This connections helps in the backpropagation to earlier layers and also in the detail reconstruction in the decoder. Some outstanding architectures such as FCN8 or U-Net uses this concept.

Therefore, we built an architecture with *Convolutional* Layers and skip connections, kind of Fully Convolutional Network with skip connections. This architecture do not use residual blocks. The skip connections added in this architecture are going to **add** the encoder layer to the correspondent decoder layer, so the decoder dimension would be the same. This autoencoder is built with 3 *Convolutional* Layers, each one is followed by a *BatchNormalization* and a *ReLu* function. Each one of this *Convolutional blocks* (*Conv-BN-ReLu*) downsize the image in half and double the number of filters, so finally our latent space is $16x16x128$ again. The

skip connections goes from the output of the *BatchNormalization* of the *Convolutional block* of 1st and 2nd encoder layers, to the output of the *BatchNormalization* of the 1st and 2nd decoder layers, thus, decoder block is also a little bit different because it incorporate an addition operation in the middle. This decoder block is seen in figure 5.15. The whole architecture is shown in figure 5.16 and it is coded in file `src/2.Experiments/skip_connection_cae.py`

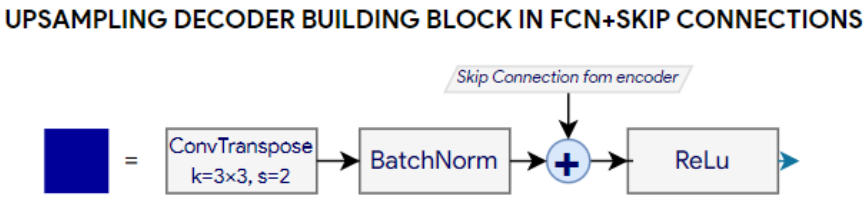

Figure 5.15: Decoder building block for Skip connection CAE.

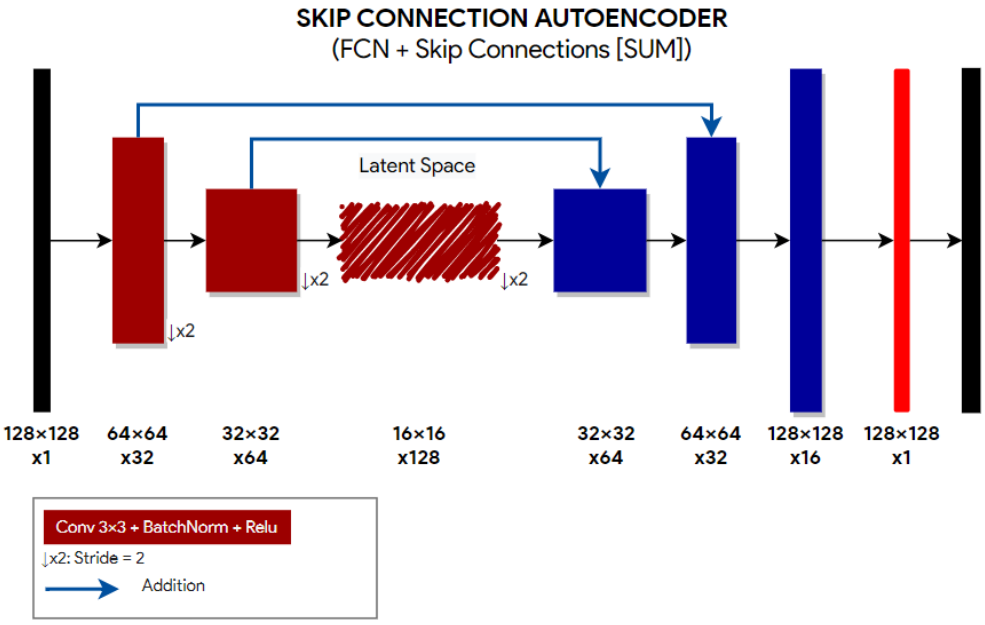

Figure 5.16: Skip connection CAE Architecture.

5.4.4.4 Residual U-NET Autoencoder

Finally, we present our **proposed method** to research the benefits of residual blocks and skip-connection combinations. We have built a U-Net autoencoder with full-pre-activation residual building blocks. U-Net is also explained in state of the art and refers to a Fully Convolutional Network with **skip connections with concatenation** instead of addition. This way, it double the number of feature maps in the decoder part, and an improvement is

shown in medicine segmentation tasks. Using concatenation instead of addition, the decoder network has the burden of make a complex operation instead of simply add it element-wise. With this architecture, **we combine these U-Net advantages with the advantages of skip connections (also inherit of U-Net) and with the aforementioned advantages of the residual building blocks**.

Regarding this, we add skip connections with concatenation to the shallow residual network with full-pre-activation building blocks already explained in section 5.4.4.1. Therefore, the diagram of the network, with the details of the connections, downsampling and feature maps is shown in figure 5.17 and coded in file `src/2.Experiments/res_skip_cae.py`.

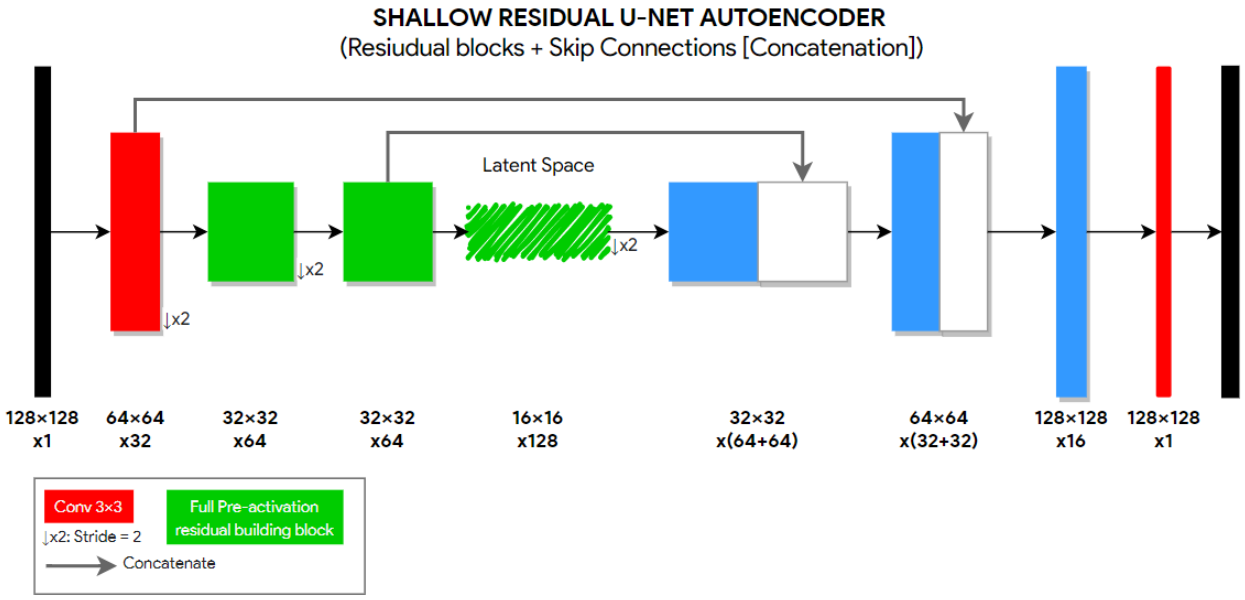

Figure 5.17: Residuel U-Net Architecture.

5.4.5 Experiments

As we explained before, we want to compare these shallow architectures to obtain the best one. Also, we want to compare another features: the consequences of the use of data augmentation, the effects of L2 regularization, and the differences in learning between **Mean Square Error** (MSE) and **structural DiSSIMilarity** (DSSIM) losses. Consequently we started with a sample experiment training some models without data augmentation and MSE loss. As we know that data augmentation is needed, we did only a few experiments without it, placing on all the importance of the project in the experiments with data augmentation. With data

augmentation we trained the architectures both with MSE and DSSIM losses. In addition, L2 regularization was added to some of them to discover the effect. However, every architecture was not trained with and without L2 because the number of experiments wuold have been huge, so we consider that a consistent experiment regarding regularization is beyond the scope of this project. Every model trained was tested quantitatively with MSE, DSSIM and **Peak Signal to Noise Ratio** (PSNR) as well as qualitatively.

Our models use **100 epochs** for training with a **batch size of 32**. We use **RMSProp optimizer** which divides the gradient by a running average of its recent magnitude. We configure the *Early Stopping* callback with a patience of 20 and a min delta of $2e-7$ if MSE loss is used or $5e-5$ if DSSIM loss instead. To optimize the training and lead it to a better convergence we use **ReduceLROnPlateau** which reduces the learning rate by a factor of 0.2 if the validation loss does not improve in 4 epochs. The min improvement also depends on the loss used.

We have coded a parametrizable python script to ease the training of the models. We only have to write the name of the architecture, a *boolean* to set the augmentation, the name of the loss metric, a *boolean* to set regularization, and the name of the residual building block used (only relevant if architecture allows residual blocks). With only this information our script configures all the experiment: creation of folder to save the model checkpoint, the Keras diagram and the *csv* of training metrics; creation of customized data loader; configuration of callbacks and its parameters: *CSVLogger*, *ModelCheckpoint* and *Earlystopping*; configuration of learning rate reducer and running of the experiment. When train has finished, we have a folder with the aforementioned documents, each one with self-explanatory name. This is coded in `src/2.Experiments/residual_cae_experiment.py`. The L2 kernel regularizer is used with a value of $1e-5$ if regularization is established.

TestMetricWrapper Python class has been coded with the goal of ease **quantitative and qualitative evaluation**. Class inputs are the paths of test images and the path of the model folders. With this information, this class give us some charts about the training steps, quantitative test metrics, and some qualitative examples. We also have a function on this class which allow us to perform a customized augmentation on a desired image and to plot the reconstruction of every model. The module is `src/2.Experiments/create_test_report.py` and the class name is *TestMetricWrapper*. It is used in the **evaluation notebooks**: `src/2.Experiments-/2.Exp_NoDAug_MSE.ipynb`, `src/2.Experiments/3.1.Exp_DAug_MSE.ipynb`, `src/2.Experiments/3.2.Exp_DAug_DSSIM.ipynb` and `src/2.Experiments/4.4.Evaluation_custom_corruptions.ipynb`

5.4.5.1 Metrics

MSE error is measured as the mean intensity pixel-wise squared error between 2 images. Besides, as SSIM is used to measuring the similarity between two images, we used DDSIM to compute the dissimilarity and being able to minimize it when used as loss function. Finally, PSNR gives the peak error in the output image. The value of this parameter should be large, as it represents the ratio of signal power-to-noise power, noise power should be minimum. PSNR is not used as loss, only as test metric.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2; PSNR = 10 \times \log_{10}\frac{peakval^2}{MSE(x,y)}; DSSIM = \frac{1 - SSIM(x,y)}{2}$$

5.4.5.2 Without data augmentation

To start with simple experiments, we compared the following architectures without data augmentation: Shallow residual autoencoder with original block, Shallow residual autoencoder with full-pre-activation block, Shallow residual autoencoder with full-pre-activation block and L2 regularization, Myronenko autoencoder, Myronenko autoencoder with regularization and Skip Connection CAE.

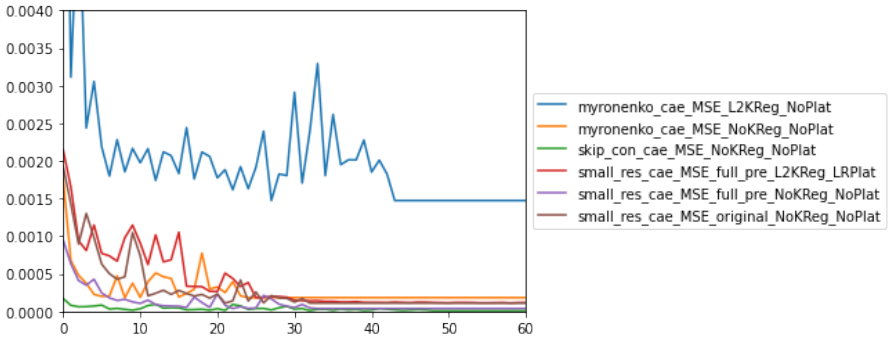

Figure 5.18: MSE evolution in the training of models without data augmentation.

The evolution of training loss (MSE) is shown in 5.18. We could see the quickly convergence of the methods, with convergence made mostly in less than 40 epochs. The table showcases the best model is **Skip Connection CAE** both in validation MSE and test MSE, DSSIM and PSNR. This big difference with the other methods could also be seen in 5.19. Most of test metrics are outstanding, showing a good reconstruction of the test set. We also realize the big similarity between validation and test loss (MSE). This similarity is a proxy of the

good stratification of our dataset and that we do not notice overfitting. The **shallow residual CAE with full-pre-activation blocks** shows also a very good performance, being better than the original block. In addition, the first intuition about L2 regularization is the fact that the same model but with regularization has lower test metrics. We are not going beyond in the explanation in this moment, we will analyze it better with the augmentation results.

Model	loss	L2	Val loss	MSE	DSSIM	PSNR
Skip connection CAE	MSE	No	**1.10e-5**	**1.07e-05**	**1.04e-03**	**49.8**
Shallow RES full-pre	MSE	No	3.92e-5	**3.82e-05**	2.64e-03	44.4
Shallow RES full-pre	MSE	Yes	1.10e-4	**8.56e-05**	4.90e-03	41.0
Shallow RES orignial	MSE	No	1.11e-4	**1.09e-04**	1.40e-02	39.7
Myronenko CAE	MSE	No	1.83e-4	**1.81e-04**	5.59e-03	37.7
Myronenko CAE	MSE	No	1.47e-3	**1.27e-03**	4.40e-02	29.3

Table 5.2: Validation and Test metrics for experiments without data augmentation

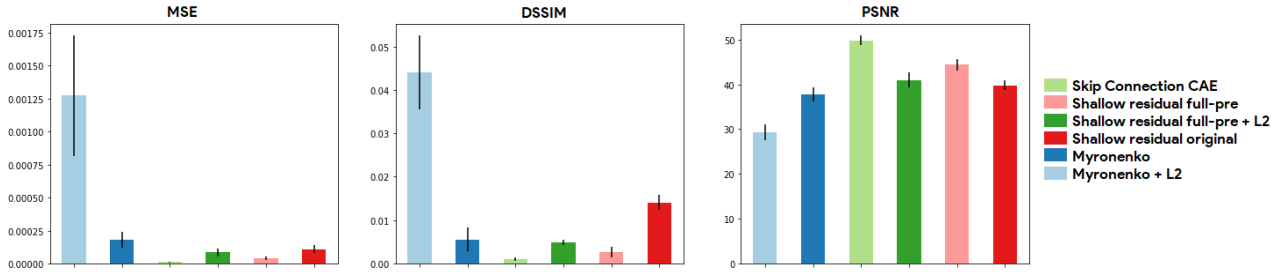

Figure 5.19: Bar charts of test metrics for experiments without data augmentation.

We have shown a very good test metrics. But qualitative reconstruction is seen in figure 5.20. We realize that all models have a very good reconstruction performance when the data is clean. However, Myronenko+L2 gives slightly blurry output images, this is the reason because its test metrics are higher. When input data is corrupted, all methods are limited to copying the input, but Myronenko+L2 removes the dropout pixels with the downside of blurring. This is a very interesting fact: although we have trained the method without data augmentation, it is able to reconstruct some corruptions.

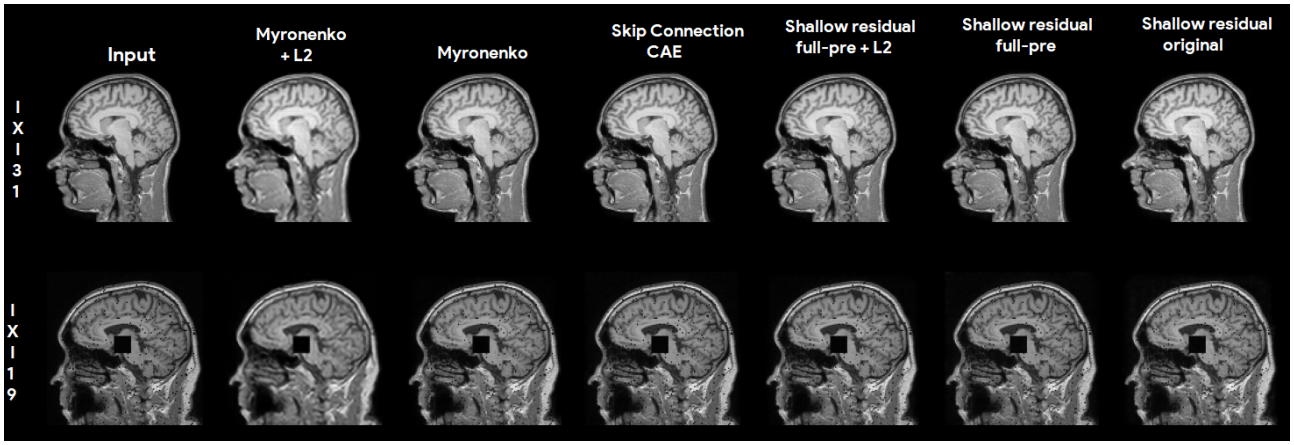

Figure 5.20: MRI reconstruction of non-augmented models from an clean input and another corrupter one.

5.4.5.3 With data augmentation

We have encouraging results with the models without data augmentation: almost every architecture is able to reconstruct a brain MRI with a huge similarity even in lower-details. We also notice that it are not able to reconstruct any image using their knowledge of how a healthy brain looks like. However, we want our models to be improved by the aforementioned benefits of data augmentation. In this experiments we went beyond the residual architectures and the skip connection ones: we propose a new architecture combining residual and U-Net as explained in section 5.4.4.4. Considering the above results, the models trained in this stage was: Shallow residual full-pre, Shallow residual full-pre+L2, Skip Connection CAE, Skip Connection CAE+L2, Myronenko CAE and the proposed Residual U-NET autoencoder. We trained each model optimizing both **MSE and DSSIM losses**, thus we have 12 final models with data augmentation.

First of all, we can see that the evolution of validation loss (figure 5.21) is more stable than the evolution without data augmentation, but, even so, the convergence is still very fast.

Table 5.3 shows the results of every model. It shows the model architecture, loss and configuration in the first 3 columns; best validation loss in the 4th column measured in the loss metric; then, it shows the MSE, DSSIM and PSNR metric on test set. In each column, result is shown in bold-style. However, in one column the best value is formatted in bold-blue and the others in bold. This column is the test metric of the trained loss. We realize **our proposed method (RES-UNET) outperforms quantitatively every model else regarding every**

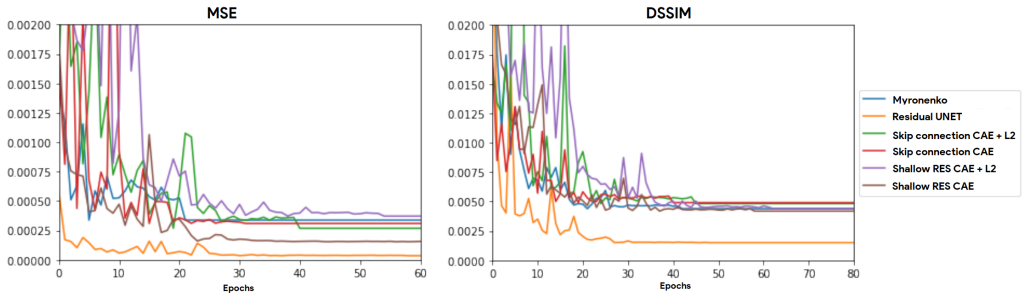

Figure 5.21: Evolution of validation loss on augmented models.

test metric. This method is the best trained with both loss functions. To continue, if we focus in comparison of regularization we can observe one clear fact: on one hand **L2 regularization is helpful when the model optimizes DSSIM loss and on the other handn this L2 regularization decreases the performance for methods that optimizes MSE loss**. Deeper conclusion and comparisons will be made in section 5.5.

Model	loss	L2	Val loss	MSE	DSSIM	PSNR
Residual U-NET	MSE	No	**3.58e-05**	**3.44e-05**	**2.95e-03**	**44.9**
Shallow RES full-pre	MSE	No	1.55e-04	**1.51e-04**	6.75e-03	38.6
Skip connection CAE	MSE	Yes	2.69e-04	**2.25e-04**	1.65e-02	36.8
Skip connection CAE	MSE	No	3.10e-04	**2.99e-04**	9.36e-03	35.7
Myronenko CAE	MSE	No	3.38e-04	**3.27e-04**	1.57e-02	35.1
Shallow RES full-pre	MSE	Yes	3.72e-04	**3.24e-04**	1.14e-02	35.2
Residual U-NET	DSSIM	No	**1.50e-03**	**7.49e-05**	**1.44e-03**	**41.8**
Shallow RES full-pre	DSSIM	Yes	4.42e-03	2.34e-04	**3.70e-03**	36.7
Shallow RES full-pre	DSSIM	No	4.19e-03	2.88e-04	**4.14e-03**	35.9
Myronenko CAE	DSSIM	No	4.39e-03	6.69e-04	**4.31e-03**	32.1
Skip connection CAE	DSSIM	Yes	4.82e-03	4.08e-04	**4.38e-03**	34.2
Skip connection CAE	DSSIM	No	4.90e-03	4.57e-04	**4.71e-03**	33.7

Table 5.3: Validation and Test metrics for experiments with data augmentation.

Qualitative brain MRI reconstructions made by models trained with data augmentation are shown in figure 5.22 for methods which optimize MSE and in figure 5.23 for methods which optimizes DSSIM. We choose a fully-augmented image to test the Reconstruction capability of the methods. **Residual U-Net**, the proposed method, provides the best reconstruction both for MSE and DSSIM loss models, showing that it is able to remove noise, dropout and blurring and being able to fill the blanked out region. We can see how this model is the more accurate filling the blanked-out region, being the unique which gives a good reconstruction of the limit

region between the skull and the brain. Most of the remaining models are also prominent in all reconstruction tasks except filling blanked-out regions.

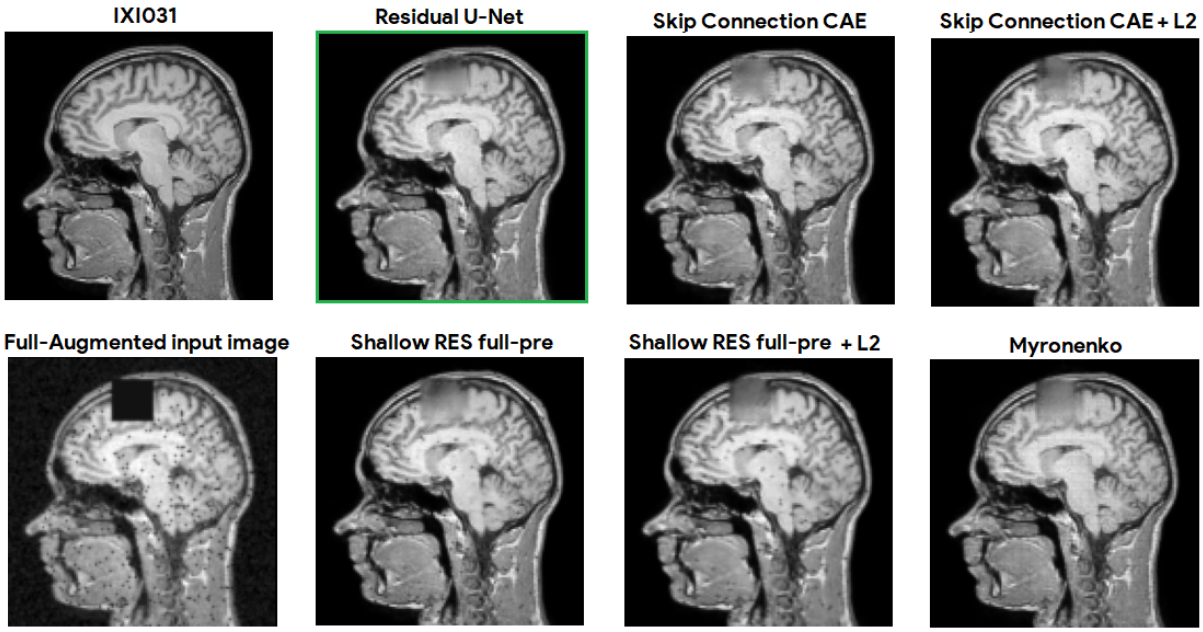

Figure 5.22: Reconstruction of corrupted input made by MSE-Augmented methods. Green-framed-image is the one chosen as the best reconstruction.

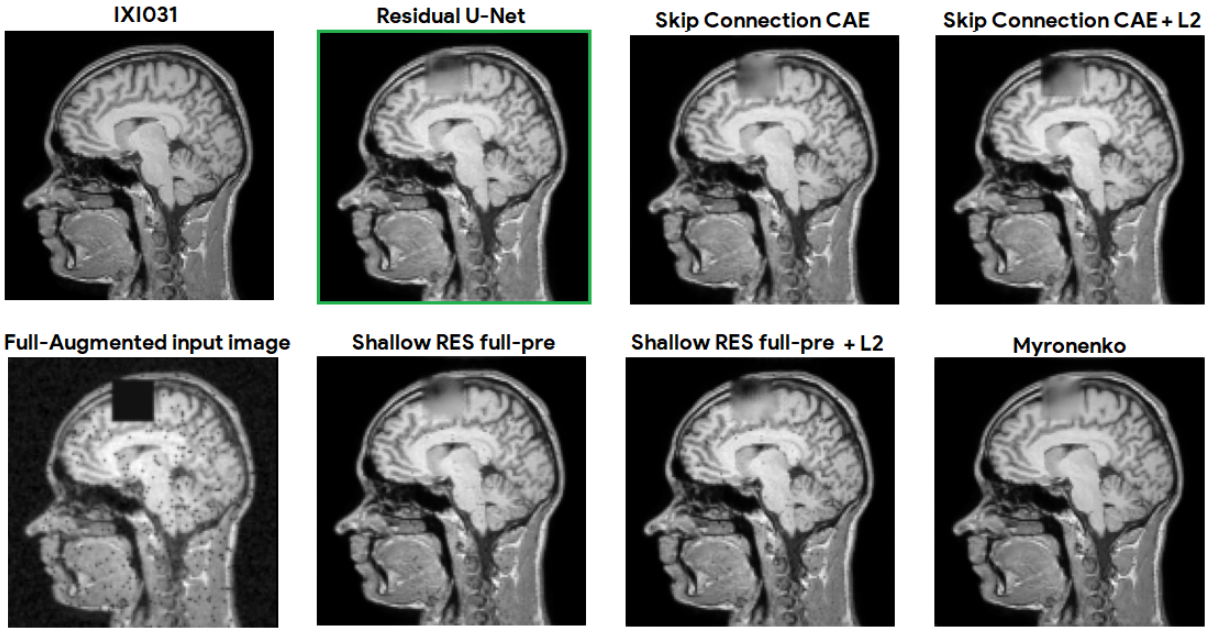

Figure 5.23: Reconstruction of corrupted input made by DSSIM-Augmented methods. Green-framed-image is the one chosen as the best reconstruction.

Deeper conclusions are going to be discussed in the nest Results Section 5.5.

5.5 Results

To obtain statistical significance for the metric difference to support our conclusions, we computed the t-test for each pair of methods. We are comparing the different models with augmentation, through it test measures, so the t-test should be the dependent t-test due to the test samples are the same for all models. We applied a pairwise-models dependent sample t-test (figure 5.1) with 4823 degrees of freedom, and the result is shown in figure 5.24 where pink, purple and black boxes represent significant difference between the methods. Due to big the biog size of the test set, most of the differences are significant even though they are small. Due to this significance, our results and conclusions are supported by this statistical test.

$$t = \frac{MSE_1 - MSE_2}{\sqrt{\frac{S_D}{\sqrt{N}}}} \qquad S_D = \sqrt{\frac{\sum_i^{N_{test}}(MSE_{1_i} - MSE_{2_i}) - \frac{\sum_i^{N_{test}}(MSE_{1_i} - MSE_{2_i})^2}{N_{test}}}{N_{test} - 1}} \tag{5.1}$$

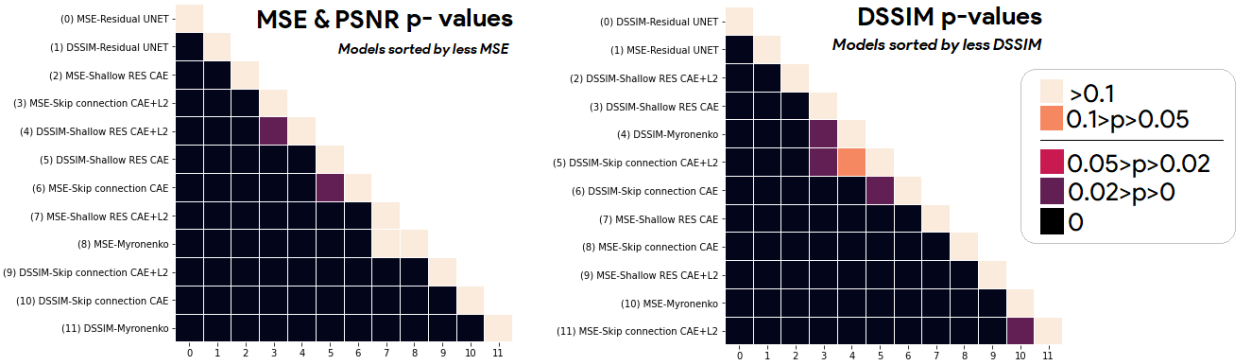

Figure 5.24: P-values for t-test pairwise comparison.

First of all, we would like to mention the obvious advantages of data augmentation. Experiments without data augmentation show us that the augmentation step could be critical in deep learning image projects. Besides, this stage provided us a first feedback about how c̈lassicalärchitectures worked. However, the big burden of the project relies on the experiments with data augmentation.

The first big conclusion of the project is that Residual U-net, our proposed architecture, outperforms the only residual or only skip-connection based ones. Combining the aforementioned benefits of residual blocks and skip connections, we develop a model which is able to get information about the structure of the brain. We measured this in a quantitative way, the table 5.3 show all the test metrics (which are also shown in a summarized and more visual way

in figure 5.25), and the t-test post-hoc method (fig. 5.24) provide us statistical significance for this differences on the metrics. We have also checked it in a quantitative way in figures 5.22 and 5.23, where we can see that structural information is used when filling the black region.

We also notice that **all methods are outstanding (both MSE and DSSIM) for removing noise and fixing image blur**, all of them being able to perform a very good reconstruction in which neither blur nor noise is appreciated. However, there are difference between architectures reconstructing the **dropout**. Shallow Residual architecture (both for DSSIM and MSE loss and also both L2 and without L2), provide reconstruction with some little dark dots in them as we can see in both figures for Shallow RES models. On the contrary, Residual U-Net, Myronenko and Skip Connection CAE are able to totally reconstruct the dropped out pixels.

Finally, a critical process in reconstruction is the filled of **blanked-out regions**. Focusing on the **difference between architectures**, we observe that Residual U-Net is the more accurate one if we observe the models trained with MSE in figure 5.22. It is the only model who is able to estimate a good shape for the region between the skull and the brain and simulate a kind of brain limit. This model is followed in accuracy by the Shallow Residual models (Both for L2 and no-L2 regularization). This 2 methods, although with lower qualitative accuracy, try to reconstruct a darker part belonging to the limit brain-skull and a lower brighter part belonging to the brain. The remaining MSE-loss-methods make a blurred reconstruction of the blanked-out region for this example.

However, **the reconstruction of blanked-out regions is the main difference between models trained with MSE and with DSSIM loss**. DSSIM models show as good reconstruction as MSE models with blurring, noise and dropout, thus, main difference is observed in the reconstruction of blancked-out parts. If we compare the same model in figures 5.22 and 5.23, we notice the best brain structure reconstruction made in DSSIM models. Residual U-Net is still the best and the structure reconstructed is better for the model trained with DSSIM than trained with MSE loss. But reconstruction improvement is more visible in the other models. The same models but trained with DSSIM loss provides better reconstruction based on the structural information of the brain. The reconstruction made by DSSIM-loss models show a big effort to simulate the skull-brain limit, painting kind of a dark line, instead of the same MSE-loss models which paint a big blurred square. This improvement is specific showed in the figure 5.23, where Shallow Residual models shows the better skull-brain limit reconstruction, but also it is seen in Myronenko and Skip Connection+L2 CAE.

As a final thought, **we notice qualitatively that L2 regularization has a better effect**

when DSSIM loss is used. We can see in figure 5.23 that Shallow residual+L2 works better than Shallow residual and also Skip Connection CAE+L2 provides better reconstruction than Skip Connection CAE without regularization. In contrast, no difference in shown between these pairs when MSE is optimized (5.22).

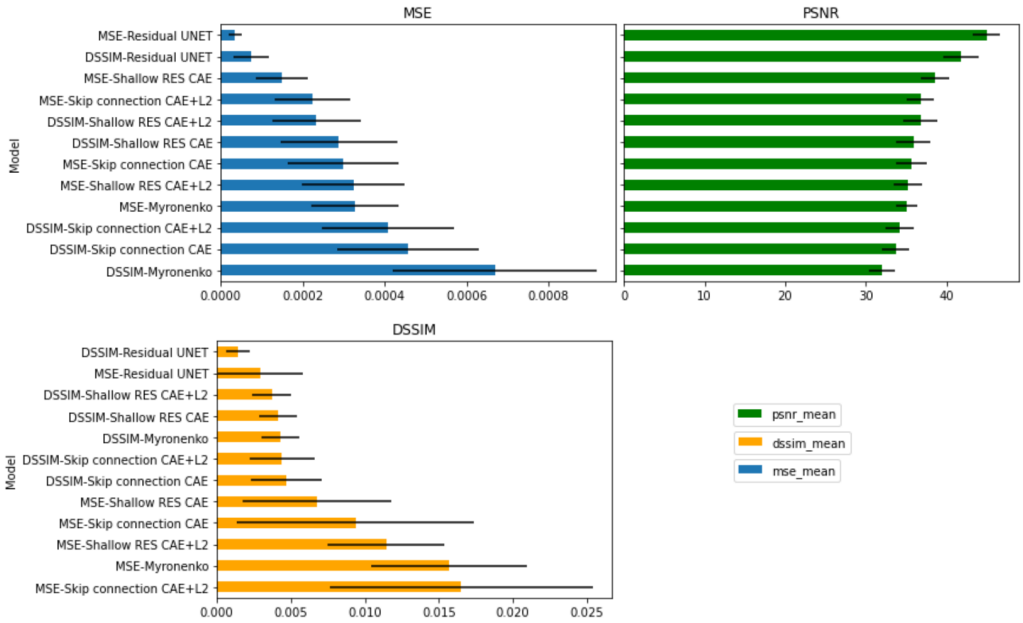

Figure 5.25: Comparison of all augmented-model test metrics.

Chapter 6

Conclusion and Outlook

This section will be made in the last PEC: REPORT

As a Masther Thesis on Data Science, we also have focus our project in keep the good habits of Data Science projects. We have follow the Crisp-dm stages and the classical data lifecycle.

6.1 Future work

Show how the MRI reconstruction is donde in the brain-sides to show the benefits of the DEPPBRAIN filtering

Make some filter exploration to dive deep into the differences in reconstruction between residual and skip connections architectures. http://deeplearning.net/tutorial/dA.html#running-the-code[1]

Make a simple experiment of potential use of this autoencoder: show how brain segmentation improves with reconstructed images instead of corrupted ones. For this task we can follow this steps: compare the acurracy of segmentation masks made directly from corrupted inputs to the accuracy of segmentation masks made from the Autoencoder reconstructions. (1) Get mask from original brain (2) get mask from corrupted mri (3) get mask from reconstructed input (4) get metrics ACU(mask, corr-mask) and ACU(mask, reconts-mask)

[1] http://deeplearning.net/tutorial/dA.html#running-the-code

Bibliography

[1] David C Preston. *Magnetic Resonance Imaging (MRI) of the Brain and Spine: Basics*. Case Western Reserve University, 2006.

[2] José V Manjón, Pierrick Coupé, Luis Martí-Bonmatí, D Louis Collins, and Montserrat Robles. Adaptive non-local means denoising of mr images with spatially varying noise levels. *Journal of Magnetic Resonance Imaging*, 31(1):192–203, 2010.

[3] Andriy Myronenko. 3d mri brain tumor segmentation using autoencoder regularization. In *International MICCAI Brainlesion Workshop*, pages 311–320. Springer LNCS, 2018.

[4] José V. Manjón, José E. Romero, Roberto Vivo-Hernando, Gregorio Rubio, Fernando Aparici, Maria de la Iglesia-Vaya, Thomas Tourdias, and Pierrick Coupé. Blind mri brain lesion inpainting using deep learning. In Ninon Burgos, David Svoboda, Jelmer M. Wolterink, and Can Zhao, editors, *Simulation and Synthesis in Medical Imaging*, pages 41–49, Cham, 2020. Springer International Publishing.

[5] Camilo Bermudez, Andrew J Plassard, Larry T Davis, Allen T Newton, Susan M Resnick, and Bennett A Landman. Learning implicit brain mri manifolds with deep learning. In *Medical Imaging 2018: Image Processing*, volume 10574, page 105741L. International Society for Optics and Photonics, 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[8] Sakshi Patel, Bharath K P, Balaji Subramanian, and Rajesh Muthu. Comparative study on histogram equalization techniques for medical image enhancement. pages 657–669, 01 2020.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] Daiki Tamada. Noise and artifact reduction for mri using deep learning. *arXiv*, abs/2002.12889, 2020.

[11] Mohammed T Abou-Saleh. Neuroimaging in psychiatry: an update. *Journal of Psychosomatic Research*, 61(3):289–293, 2006.

[12] Andre F Marquand, Iead Rezek, Jan Buitelaar, and Christian F Beckmann. Understanding heterogeneity in clinical cohorts using normative models: beyond case-control studies. *Biological psychiatry*, 80(7):552–561, 2016.

[13] Janaina Mourão-Miranda, David R Hardoon, Tim Hahn, Andre F Marquand, Steve CR Williams, John Shawe-Taylor, and Michael Brammer. Patient classification as an outlier detection problem: an application of the one-class support vector machine. *Neuroimage*, 58(3):793–804, 2011.

[14] Changhee Han, Leonardo Rundo, Ryosuke Araki, Yujiro Furukawa, Giancarlo Mauri, Hideki Nakayama, and Hideaki Hayashi. Infinite brain tumor images: Can gan-based data augmentation improve tumor detection on mr images? In *Proc. Meeting on Image Recognition and Understanding (MIRU 2018), Sapporo, Japan*, 2018.

[15] Florian Knoll, Jure Zbontar, Anuroop Sriram, Matthew J Muckley, Mary Bruno, Aaron Defazio, Marc Parente, Krzysztof J Geras, Joe Katsnelson, Hersh Chandarana, et al. fastmri: A publicly available raw k-space and dicom dataset of knee images for accelerated mr image reconstruction using machine learning. *Radiology: Artificial Intelligence*, 2(1):e190007, 2020.

[16] Errol M Bellon, E Mark Haacke, Paul E Coleman, Damon C Sacco, David A Steiger, and Raymond E Gangarosa. Mr artifacts: a review. *American Journal of Roentgenology*, 147(6):1271–1281, 1986.

[17] MA Balafar. Review of noise reducing algorithms for brain mri images. *methods*, 10:11, 2012.

[18] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[19] Walter HL Pinaya, Andrea Mechelli, and João R Sato. Using deep autoencoders to identify abnormal brain structural patterns in neuropsychiatric disorders: A large-scale multi-sample study. *Human brain mapping*, 40(3):944–954, 2019.

[20] Bjoern H Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, et al. The multimodal brain tumor image segmentation benchmark (brats). *IEEE transactions on medical imaging*, 34(10):1993–2024, 2014.

[21] Gihyun Kwon, Chihye Han, and Dae-shik Kim. Generation of 3d brain mri using auto-encoding generative adversarial networks. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 118–126. Springer, 2019.

[22] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using convolutional auto-encoders with symmetric skip connections. *arXiv preprint arXiv:1606.08921*, 2016.

[23] Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246. IEEE, 2016.

[24] Maosong Ran, Jinrong Hu, Yang Chen, Hu Chen, Huaiqiang Sun, Jiliu Zhou, and Yi Zhang. Denoising of 3d magnetic resonance images using a residual encoder–decoder wasserstein generative adversarial network. *Medical image analysis*, 55:165–180, 2019.

[25] Ben A Duffy, Wenlu Zhang, Haoteng Tang, Lu Zhao, Meng Law, Arthur W Toga, and Hosung Kim. Retrospective correction of motion artifact affected structural mri images using deep learning of simulated motion. *MIDL 2018 Conference*, 2018.

[26] Pierrick Coupé, Thomas Tourdias, Pierre Linck, José E. Romero, and José V. Manjón. Lesionbrain: An online tool for white matter lesion segmentation. In Wenjia Bai, Gerard Sanroma, Guorong Wu, Brent C. Munsell, Yiqiang Zhan, and Pierrick Coupé, editors, *Patch-Based Techniques in Medical Imaging*, pages 95–103, Cham, 2018. Springer International Publishing.

[27] Théo Estienne, M. Lerousseau, M. Vakalopoulou, Emilie Alvarez Andres, E. Battistella, Alexandre Carré, S. Chandra, S. Christodoulidis, M. Sahasrabudhe, Roger Sun, C. Robert, Hugues Talbot, N. Paragios, and E. Deutsch. Deep learning-based concurrent brain registration and tumor segmentation. *Frontiers in Computational Neuroscience*, 14, 2020.

[28] E. M. Yu and M. R. Sabuncu. A convolutional autoencoder approach to learn volumetric shape representations for brain structures. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 1559–1562, 2019.

[29] Spyridon Bakas, Mauricio Reyes, András Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Takeshi Shinohara, Christoph Berger, Sung Min Ha, Martin Rozycki, Marcel Prastawa, Esther Alberts, Jana Lipková, John B. Freymann, Justin S. Kirby, Michel Bilello, Hassan M. Fathallah-Shaykh, Roland Wiest, Jan Kirschke, Benedikt Wiestler, Rivka R. Colen, Aikaterini Kotrotsou, Pamela LaMontagne, Daniel S. Marcus, Mikhail Milchenko, Arash Nazeri, Marc-André Weber, Abhishek Mahajan, Ujjwal Baid, Dongjin Kwon, Manu Agarwal, Mahbubul Alam, Alberto Albiol, Antonio Albiol, Alex Varghese, Tran Anh Tuan, Tal Arbel, Aaron Avery, Pranjal B., Subhashis Banerjee, Thomas Batchelder, Kayhan N. Batmanghelich, Enzo Battistella, Martin Bendszus, Eze Benson, José Bernal, George Biros, Mariano Cabezas, Siddhartha Chandra, Yi-Ju Chang, and et al. Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge. *CoRR*, abs/1811.02629, 2018.

[30] Adrian Tousignant, Paul Lemaître, Doina Precup, Douglas L. Arnold, and Tal Arbel. Prediction of disease progression in multiple sclerosis patients using deep learning analysis of mri data. volume 102 of *Proceedings of Machine Learning Research*, pages 483–492, London, United Kingdom, 08–10 Jul 2019. PMLR.

[31] Mohammadi Fatemeh. *Convolutional Autoencoder for Studying Dynamic Functional Brain Connectivity in Resting-State Functional MRI.* PhD thesis, Concordia University, April 2019.

[32] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. IEEE, 2016.

[33] Christopher M Bishop et al. *Neural networks for pattern recognition.* Oxford university press, 1995.

[34] Masafumi Kidoh, Kensuke Shinoda, Mika Kitajima, Kenzo Isogawa, Masahito Nambu, Hiroyuki Uetani, Kosuke Morita, Takeshi Nakaura, Machiko Tateishi, Yuichi Yamashita, et al. Deep learning based noise reduction for brain mr imaging: tests on phantoms and healthy volunteers. *Magnetic Resonance in Medical Sciences*, pages mp–2019, 2019.

[35] Motohide Kawamura, Daiki Tamada, Satoshi Funayama, Marie-Luise Kromrey, Shintaro Ichikawa, Hiroshi Onishi, and Utaroh Motosugi. Accelerated acquisition of high-resolution diffusion-weighted imaging of the brain with a multi-shot echo-planar sequence: Deep-learning-based denoising. *Magnetic Resonance in Medical Sciences*, pages tn–2019, 2020.

[36] Nishant Chauhan and Byung-Jae Choi. Denoising approaches using fuzzy logic and convolutional autoencoders for human brain mri image. *International Journal of Fuzzy Logic and Intelligent Systems*, 19(3):135–139, 2019.

[37] PM Johnson and M Drangova. Motion correction in mri using deep learning. In *Proceedings of the ISMRM Scientific Meeting & Exhibition, Paris*, volume 4098, 2018.

[38] Irida da Cunha. *El trabajo de fin de grado y de máster: Redacción, defensa y publicación.* Editorial UOC, 2016.

[39] Rüdiger Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Springer-Verlag London, UK, 2000.

[40] Priyanka Garg and Trisha Jain. A comparative study on histogram equalization and cumulative histogram equalization. *International Journal of New Technology and Research*, 3(9), 2017.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.