POLITECNICO DI MILANO

School of Industrial and Information Engineering

Master of Science in Biomedical Engineering

3D CNN for classification of brain MRI

Supervisor: Pietro Cerveri

Thesis of:

Marco Passa

Student ID: 899075

Academic Year 2019-2020

Acknowledgements

I would like to thank Prof Cerveri, without his collaboration would not possible make this partnership.

Then I would like to thank Maria Lusia Mandelli and Yann Cobigo for the help with the project.

Finally I would to thanks my family (Carla, Sara, Miria, Luca, Marco, Sesa, Elena, Cynthia e Francesca) for the inspiration and support they have always transmitted to me.

Abstract

Currently there is not established automatic methodology for identifying patients with language deficits with an AD vs. no-AD pathology. Neurologists presently diagnose the patient population under a manual process that is time-consuming and expensive. Convolutional neural network has been widely used for the image processing in the medical field in the last years for tasks that were not possible to realize with the classical software. Research has shown that the CNN can be used in the medical field for various tasks like image denoising, segmentation and classification. In this study we build two CNN model: one for 3-D MRI brain denoising and one for 3-dimensional MRI brain classification to distinguish patients with language deficits with AD pathology versus no-AD pathology.

We want to establish if it's possible use this method also with a very small data set, made by 45 brain MRI patients for each group and 45 control subjects. In this context, a good classifier is defined as a model that can classify at least with the 70% of accuracy on data on which it is not trained on.

Based on a review of the literature, there are various architecture and methods to implement efficient CNN model also with small dataset. Regarding the architecture we used the 3-dimensional version of the convolutional layer and of the other visual processing layer that is a new

introduction in the field and show very good result on 3-dimensional images. The result with the best model created did not reach the 70% of accuracy instead the 62%. The all project has not to be consider a fail because the goal of the project was very difficult to reach for the small number of data, but still we get closer and during our experiments we understood which are the best CNN architectures for image processing of 3-dimensional images.

Contents

List of figures

1 Introduction

The majority of brain diseases diagnosis are still based on the evaluation of a doctor and still are not presents solid software that can automatize the process. We want to implement a software that is able to recognize and classify two different cohort of patient with primary progressive aphasia the non-fluent (nfvPPA) and the logopedic variant (lvPPA) from the 3-dimension brain MRI image of the subject. To do this we chose to use a machine learning technique, artificial neural networks. In the present years these methods are been used with success in a lot of areas, also in the medical field area. In specific we are going to implement a convolutional neural network, that is a specific ANN specialized for image processing, that process the data directly in 3-dimension, that result from the state of art more efficient on 3-dimension images. The most common problem of the implementation of ANN in the medical field is the missing of big dataset. To solve this problem, we are going to experiment different network and different pre-processing technique for data augmentation till reach the best model for the task.

1.1 Clinical context

Primary Progressive Aphasia (PPA) is a clinically and pathologically heterogeneous neurological condition characterized by progressive and selective language and speech impairment. International guidelines were able to establish a classification of the main variants [1]. The guidelines use

all the knowledge accumulated since the last 20 years based on language dysfunction, brain atrophy and underlying pathology. In particular, two of these variants are very difficult to distinguish: the non-fluent (nfvPPA) and the logopedic variant (lvPPA). The nfvPPA is characterized by grammatical errors in sentence production, along with motor speech-based effortful and halting speech production and phonetic-motoric errors of phoneme distortions [2], damaged in the left inferior frontal gyrus, premotor cortex, supplementary motor cortex and temporal brain regions, and it is usually caused by fronto-temporal lobar degeneration (FTLD), mainly tau [3] (including Pick disease, corticobasal degeneration, progressive supranuclear palsy). The lvPPA is characterized by short-term memory deficits for sentence repetition, phonological errors of speech sound substitution, damaged in the left parietal and temporal brain regions, and it is usually caused Alzheimer disease pathology (amyloid-Beta plaque deposition) [4] [5] [6].

Up to now, in-vivo amyloid brain imaging is available through the fluorinated amyloid positron emission tomography (PET). Instead, no in-vivo imaging techniques are available to detect tau inclusions alike for FTLD.

Only pathological reports represent the gold standard to define the underlying pathology these disorders. Moreover, PET scanning is an expensive technique and often not available in routine practice. Therefore, in the clinical routine, these patients are required to go through an extensive speech and language battery requiring an huge effort from an equip of specialized professional (neurologist, behavioral neurologist, speech

language pathologist, neuroradiologist). At the Memory and Aging Center of the University California San Francisco (UCSF), it is available the best characterized cohort of Primary Progressive Aphasia with 20 years of specific knowledge of this disease. By taking advantage of this cohort, an automated classifier that learns from this cohort and predicts the underlying pathology (AD or no AD), it would be of extremely impact in the clinical contest. The main reason why it is important to predict the underlying pathology is that therapeutic interventions target specific aggregates characteristic of the pathology underlying the disease.

1.2 Artificial Neural Network in medicine

Artificial Neural Networks (ANN) offer a powerful set of tools to analyze clinical data over a vast range of medical applications.

The most common applications are prediction, differential diagnostic and segmentation task. For example, in a classification problem, we want to predict in which class a patient would fit after the neural network has been trained on specific clinical features.

Medical classification can be problematic as it is often based on human medical judgment. The use of neural network (ANN) in the medical fields have exploded in the past few years. Like in the other fields principally because the increased number of data and the dimension of data and for the introduction of GPU that make possible analyze those big data [7].

The most common problem in the medical field though are the heterogeneity of the data and the lack of copious data in respect to the other

fields [8]. The first problem can be solved by choosing a heterogenous training set and for the second there is some data augmentation techniques like rotating the images to generate new data [9].

1.3 Goals of the thesis

The cases of this project are 2:

- *Denoising;*
- *Differential diagnostic classification.*

The first case consists in creating a machine learning model that is able to take a brain scan MRI and produce the same 3-dimensional image as output but with a reduced quantity of noise. The second case is the creation of a machine learning based model that can classify the condition of a patient in 3 class:

- *Pib positive;*
- *Pib negative;*
- *Control.*

The classification is based on the MRI brain image of the patient.

1.4 Result

The result obtained shows that with the small data set of images that was present at this point in the UCSF database is not enough to use machine learning technique. The model is able to recognize the difference between the 3 different class, but the number of data is not enough to generalize the model to the hole population and to classify with good accuracy also data non present in the training set.

For having good result in term of classification the number of data has to grow at least of a factor of 10, before this quantity of data will be accessible the machine learning technique are not efficient.

2 State of art

In this chapter I am going to introduce first how is form now diagnosed and make the classification that we want to automatize and then introduce the other project that has good result in biomedical image processing generally using artificial neural network technique.

2.1 Clinical context

From the clinical point of view, the diagnosis of Alzheimer disease and his form is diagnosed manually by the doctor. The doctor bases his diagnosis on principally the MRI images of the brain.

For this study, we identify 2 cohorts of patients that were diagnosed as nfvPPA or lvPPA by the specialized equip, had a 3D T1 structural brain image, and underwent to a PET scans or had a pathological report [10].

- *Cohort 1*: 45 PPA patients diagnosed as nfvPPA, with a structural T1, a negative amyloid PET scan or FTLD as pathology;

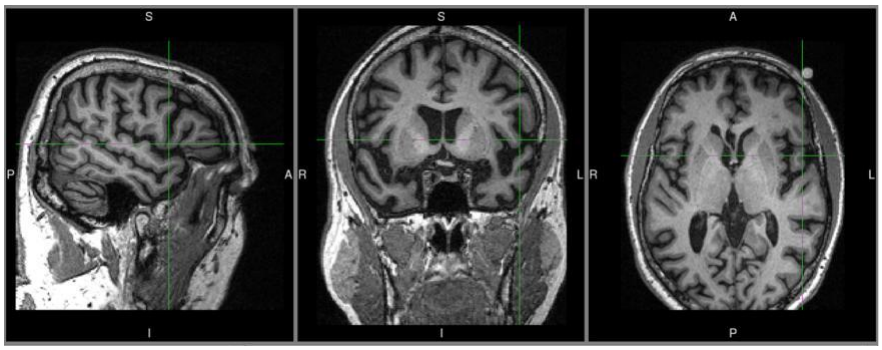


Figure 1: Example of PPA patient diagnosed as nfvPPA MRI

Cohort 2: 45 PPA patients diagnosed as lvPPA, with a structural T1 positive amyloid PET scan or AD pathology;

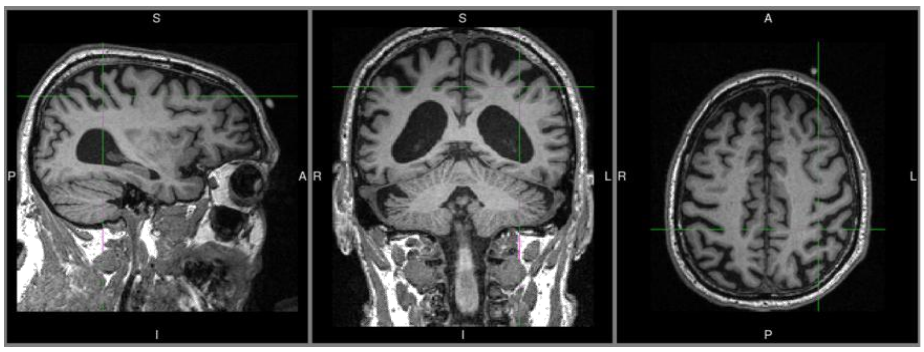


Figure 2: Example of PPA patient diagnosed as lvPPA MRI

Cohort 3: 45 Healthy subjects matched to the patients for age and gender, with a structural T1, and negative for AV-45 amyloid-PET.

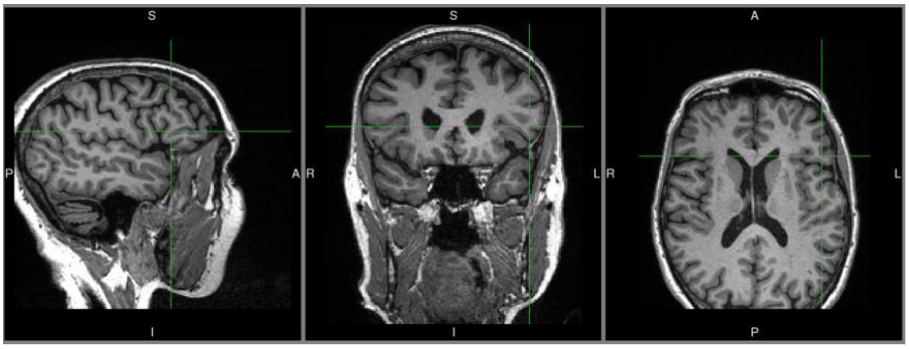


Figure 3: Example of healthy subject MRI

The cohort 1, nfvPPA patients, are identified by a degeneration of grey matter in the right temporal lobe (see fig. 1) instead the cohort 2, lvPPA patients, are identified by degeneration of gray matter in the right parietal lobe (see fig. 2). The control subjects, cohort 3, do not present gray matter degeneration in specific region of the brain (see fig. 3) [11] [12].

2.2 Methods for images processing and classification

The artificial neural networks are becoming very used in the medical image field. The main tasks for which they ANNs used are: image denoising, image segmentation, image classification.

This machine learning technique is better respect other ones like super vector machine because they do not need long preprocessing technique and has better performance on computational efficiency and accuracy of the result [13].

The bigger disadvantage of the ANNs respect the other machine learning techniques is that they need a bigger number of data to generate a model that can works correctly also with new data non present in the data set [14].

In this project we are going to use 3D visual layers instead of the classical 2D visual layers.

The better performance of the 3D visual layers respect the form in 2D is already proved on task like:

- *3D-MRI segmentation [15]*

- *3D-MRI denoising*
- *3D-MRI classification [16]*

Respect this case the difference in our project is that we have a smaller dataset that is limited to 30 images for three class for the classification case and 80 pair of MRI images and his denoised correspondence for the denoising case.

Based on the state of art, for the denoising case the number of images is sufficient to generate the model. For the classification case we did not find in the literature papers that use that number of image but only at least 10 times more.

The most used network for the denoising task is the autoencoder for the state of art, it produces good result also on 3 dimension image [17].

3 Method

In this part I explain our choose in term of software and hardware used, pre-processing and data formatting for the training, the layers used in ours model and the validation protocols.

3.1 Software and hardware

Along deep learning success in two dimensional image classification [18], and several other image processing, a lot of frameworks appeared in the machine learning landscape (TensorFlow (https://www.tensorflow.org), PyTorch (https://pytorch.org), Caffe (https://caffe.berkeleyvision.org/)).

We selected Caffe (C++) and more specifically 3D-Caffe.

The former is developed the Berkeley Artificial Intelligence Research (BAIR) Laboratory UC Berkeley, the later is an three dimensional extension of Caffe library dedicated to medical images.

The extension consists on adding a dimension the matrix on which are made the calculation from 4 to 5.

In 3D-Caffe, the blob dimensions are the space 3 dimensions representing the image, one dimension for the feature created by the convolution and one dimension for the batch-based learning strategy.

Caffe and 3D-Caffe require for input files:

- *Solver.prototxt*: the file defines the type of optimizer, all the hyper-parameters of the training, the saved parameters and the file name of the network that will be used for the training;
- *List.txt:* it contains all the path for the files used for the training;
- *Train.prototxt:* this file contains the name of the file list of the training dataset and the architecture of the training network;
- *Deploy.prototxt*: this file contains the architecture of the network used for the testing phase. It changes from the training net only for the elimination of the cost function layer.

Based on the large quantity of parameters to optimize and the size of the input, most of those libraries, including Caffe, offer graphics processing unit (GPU) processing capability support.

GPUs allowed deep learning field to grow faster by increasing the speed of training compared to CPU [19] [20].

The GPU used is a Nvidia Geforce GTX 1080 with 12 Gb of RAM (https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/).

Another feature proposed by deep learning frameworks is the Hierarchical Data Format (HDF5). HDF5 is a versatile data model that can represent complex data objects and a wide variety of metadata.

This archiving system is well suited to support multiple image format. A MATLAB API for medical images was proposed by 3D-Caffe.

A single HDF5 file contains two folder in which there are the input files and the output files saved and compressed.

In the case of the classification the output image is substituted by a number from 0 to 2 representing the class of the image (0 = Control, 1 = Pib Neg, 2 = Pib Pos).

3.2 Dataset

Participants of this study were recruited and assessed at the UCSF Memory and Aging Center (MAC).

Diagnosis for these studies was based on a multidisciplinary evaluation incorporating neurological, neuropsychological, and nursing assessment by following the international guidelines.

CNs data were obtained from a cohort of subjects recruited at the MAC via advertisements and community events. CNs underwent the same evaluation as patients and were required to have no clinically significant cognitive or behavioral complaints, performance within one standard deviation of normal on all cognitive tasks, and to have brought a knowledgeable informant to verify the absence of clinically significant cognitive or behavioral problems.

CNs were excluded if they had a history of significant mood disorders, clinically significant alcohol or drug use, significant vascular disease, visual problems that would impair test performance, other neurological conditions, and self-reported deficits in cognition.

For this study, we identify two cohorts of patients that were diagnosed as nfvPPA or lvPPA by the specialized equip, had a 3D T1 structural brain image, and underwent to a PET scans or had a pathological report.

3.2.1 Image Acquisition

All research was performed in accordance with the Code of Ethics of the World Medical Association.

All subjects provided informed consent, and the clinical and imaging protocols were approved by the UCSF Committee on Human Research.

All participants underwent whole-brain imaging on a Siemens TIM Trio 3 Tesla MRI scanner with a 12-channel head coil.

T1-weighted images were acquired with the Magnetization Prepared Rapid Gradient Echo (MP-RAGE) sequences (240 x 256 matrix; FOV = 256 mm; 160 slices; voxel size = 1.0 x 1.0 x 1.2 mm^3; TR = 2300 ms; TE = 2.98 ms; flip angle = 9°).

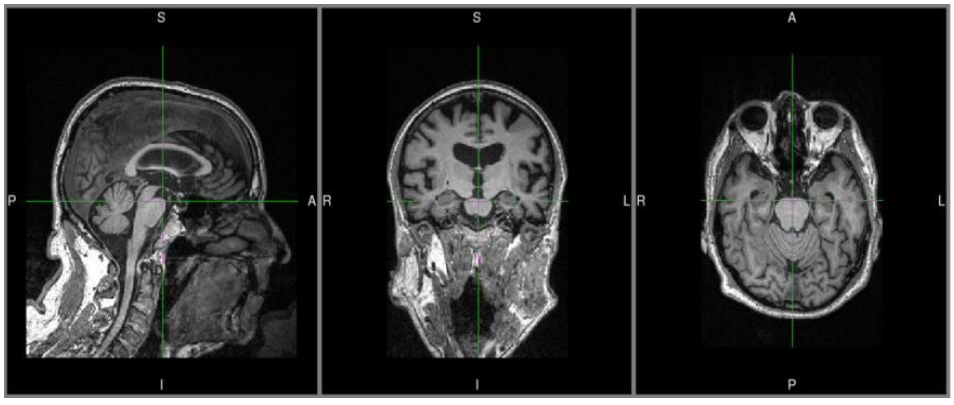


Figure 4: Example of MRI without pre-processing

3.2.2 Data augmentation and formatting

The data augmentation consists in two transformations. The first transformation was a cropping formatting of the images with the dimensions 192 x 192 x 180 voxels along the three dimensions.

This last transformation cut out the background of the images (background being everything except the brain of the subject).

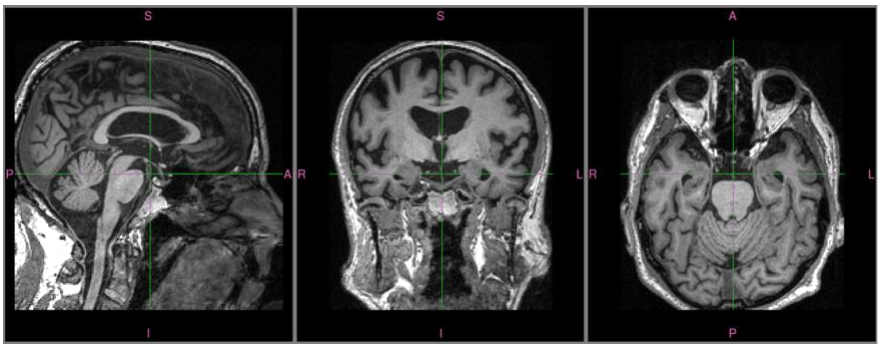


Figure 5: Example of MRI after cropping

Then new images were generated by applying a random rotation, between ∓9° along each Euclidian direction, on the original set of images. A ratio of three new images per original image was applied. Both transformations increase the dimension of the input dataset and reduce the memory foot print per image. For this task, I developed an application programming interface (API), based on Insight Toolkit (https://itk.org) libraries, rotating, cropping the 3D image, and saving the files in HDF5 format.

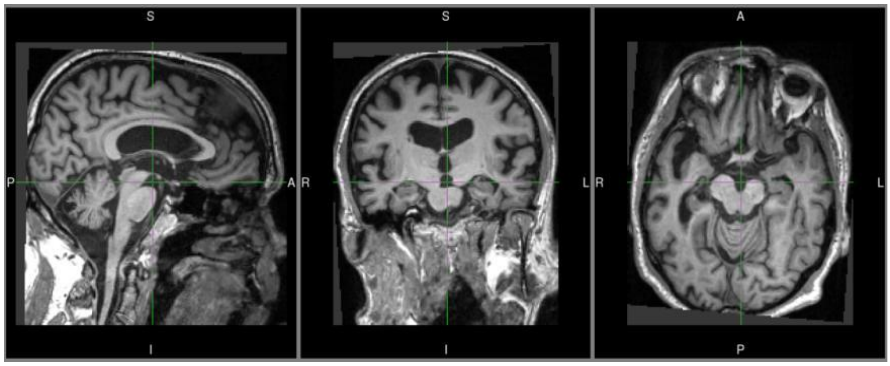


Figure 6: Example of MRI after cropping and rotation

3.3 Layers

There are different layers with different architectures based on the task that they have to perform.

The simplest layer is the fully connected layer, in which all the inputs are connected to the neurons of the layer and also all the output. The problem of this layer in image processing is that the number of parameters become too big.

To solve this problem are used more specialized layers, which architecture is already modeled for the task. This type of layers is the convolutional layer, the deconvolutional layer and the pooling layer. Usually are used in their 2 dimension version on both 2 dimension and 3 dimension image, in this project we will use their 3 dimension version. The advantage respects the 2D convolution is that the 3D convolution can process 3D data without any loss of information [21].

In the few last years it was used for segmentation task in the biomedical field [22].

One network very used for biomedical segmentation in his 2D version, but usable also in 3D version is the U-Net [23]. It consists in a convolution path to create and analyze more features, plus a deconvolution path to take back the image at the original dimension.

3.3.1 3D-Convolutional layer

Another important introduction made by LeCun was the convolutional layer that, inspired by the human neural network of vision, is the basis for image classification and processing [24].

In an image, nearby voxels can influence each other thus it is important to extract this information.

The convolutional layers allow this with the use of a filter. In this case, the filter is a kernel of a specific size (for example 3x3 or 5x5) that moves across the image. For each point on the image, a value is calculated based on the filter using a convolution operation.

The advantage of this process is that it is possible to reduce the parameters across the network whereas keeping the information of nearby voxels.

After the filters have passed over the image, a feature map is generated for each filter. These are then taken through an activation function, which decides whether a certain feature is present at a given location in the image.

The parameters that can be chosen in Caffe for the 3-dimension convolution layer are the kernel in each size, the stride, the padding, the number of outputs and the activation function of the kernel.

As there is a 3-dimension convolution layer, there are all the others visual layer in their 3-dimension version like the deconvolution and the pooling layers.

All these layers work like their 2-dimension version on the adding dimension.

The deconvolution layer is used in networks like auto encoder network and U-net. It is used to get back the image to the original dimension in tasks like denoising or segmentation, when the output have the same dimension of the input [25] [26].

It works in the opposite way of the convolution layer, generating a bigger image respect the input image of the layer. The deconvolution layer needs 4 parameters:

- *Kernel dimension*: The dimension of the kernel that will filter the image;
- *Stride:* How many pixels is moved the kernel during the filtering;
- *Pad*: Number of pixels added to the image for padding purpose;
- *Number of outputs*: Number of features generated by the deconvolution layer. (https://towardsdatascience. com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215)

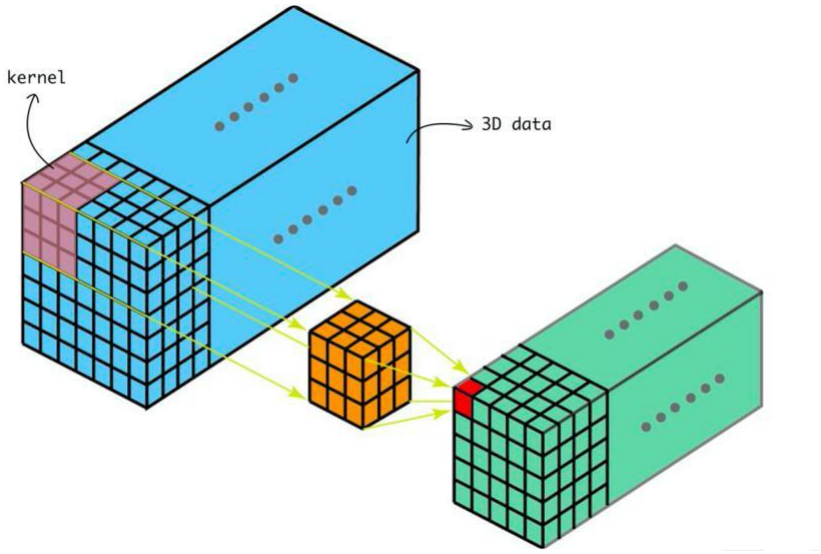


Figure 7: Graphic representation of 3D-convolution layer

3.3.2 3D-Deconvolution layer

The 3D-deconvolution layer works in the opposite way of the convolutional layer. It takes the same parameters as the convolutional layer:

- *Kernel dimension;*
- *Stride;*
- *Pad;*
- *Number of outputs.*

The difference respect to the convolutional layer is that this type of layer augments the dimension of the image. The stride number represent how much the image is going to be augmented during the processing.

The convolutional layer is used in denoising or segmentation task to get back the image to the original dimensions after reducing it in small features by the convolutional layers [27]. (https://towardsdatascience. com/review-deconvnet-unpooling-layer-semantic-segmentation-55cf8a6e380e)

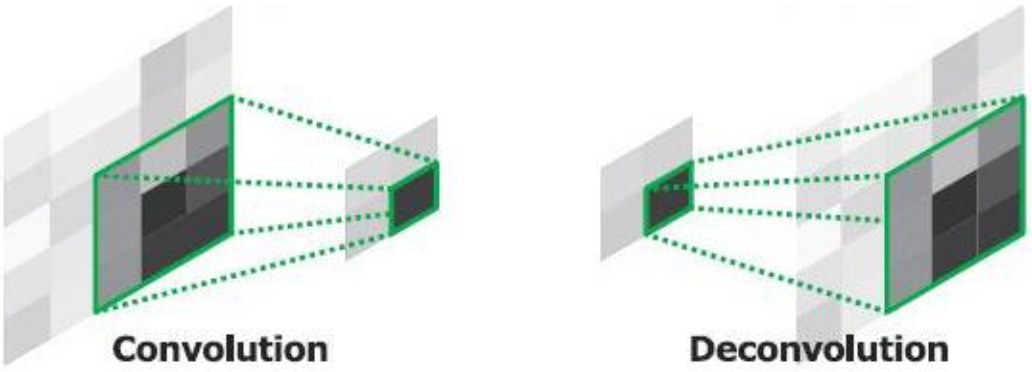


Figure 8: Graphic representation of the differences between convolution and deconvolution layers

3.3.3 3D-Pooling layer

The output features maps can be sensitive to the location of the features in the input. To overcome this problem, pooling layers have been introduced. Pooling layers select specific values on the features maps and pass through the subsequent layers.

This has the effect of making the resulting down-sampled feature maps more robust to changes in the position of the feature in the image, referred to by the technical phrase "*local translation in-variance*".

Pooling layers provide an approach to down-sampling feature maps by summarizing the presence of features in patches of the feature map. Two

common pooling methods are *average pooling* and *max pooling* that summarize the average presence of a feature and the most activated presence of a feature respectively.

The 3-D pooling layer is the 3 dimensional version of the normal pooling. The pooling is a particular version of the convolution layer where the output of the kernel is the pixel with the higher value. This layer needs 4 parameters:

- *Kernel dimension*: The dimension of the kernel that will filter the image;
- *Stride*: How many pixels is moved the kernel during the filtering;
- *Pad*: Number of pixels added to the image for padding purpose;
- *Number of outputs*: Number of features generated by the deconvolution layer. (https://principlesofdeeplearning.com/index.php/2018/08/27/is-pooling-dead-in-convolutional-networks/)

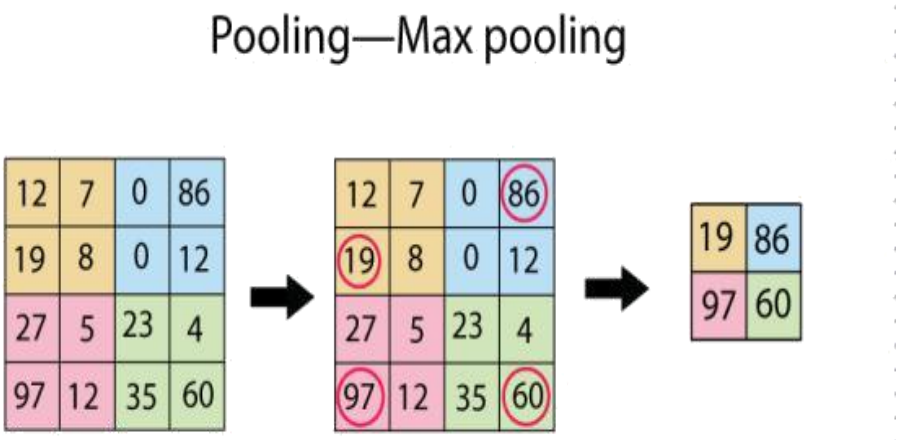


Figure 9: Graphic representation of Max pooling layer1

3.3.4 Fully connected layer

The fully connected layer connects every neuron of the previous layer with a vector of neurons long has the number of neuron of the layer before. It is use for classification task before the loss layer. (https: //www.xenonstack.com/blog/artificial-neural-network-applications/)

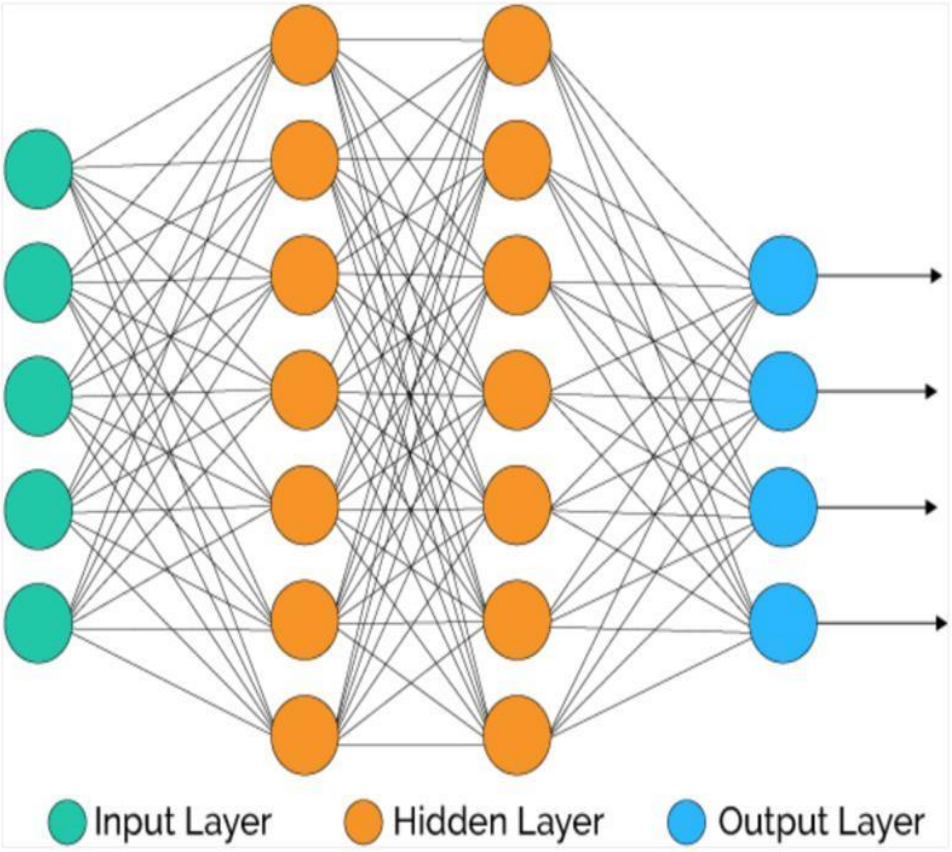


Figure 10: Graphic representation of Fully connected layer

3.3.5 Batch normalization layer

Batch normalization layer normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch

standard deviation. This strategy make the back-propagation more stable and more efficient [28].

3.3.6 ReLU layer

The ReLU layer is used in both training and test phase. It is usually used after the convolution or the deconvolution layer to process better no linear features.

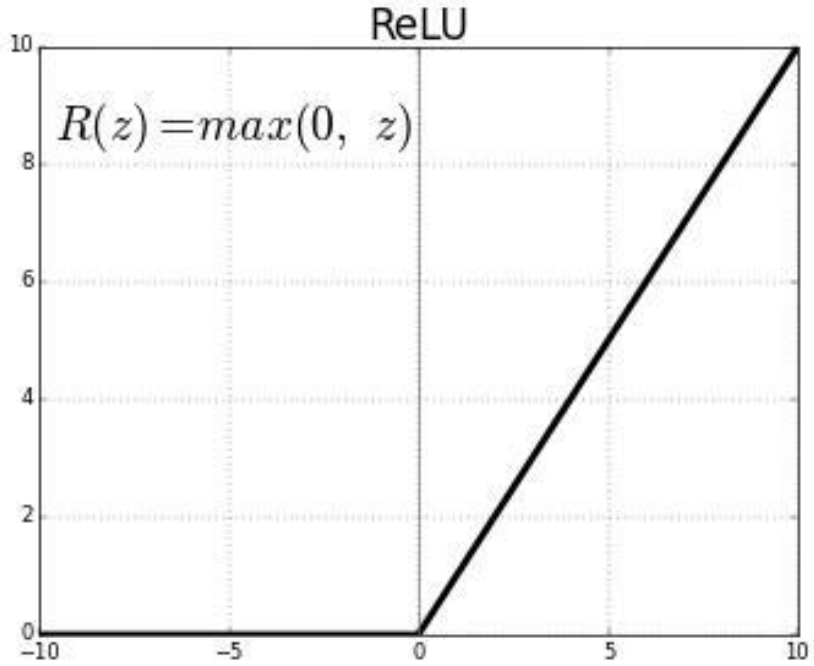


Figure 11: Graphic representation of ReLU function

3.3.7 Soft maximum layer

The soft maximum layer is used during the test phase. The soft maximum function is often used in classification task as the last layer because it generates a values of output that always are equal to 1 if are all added.

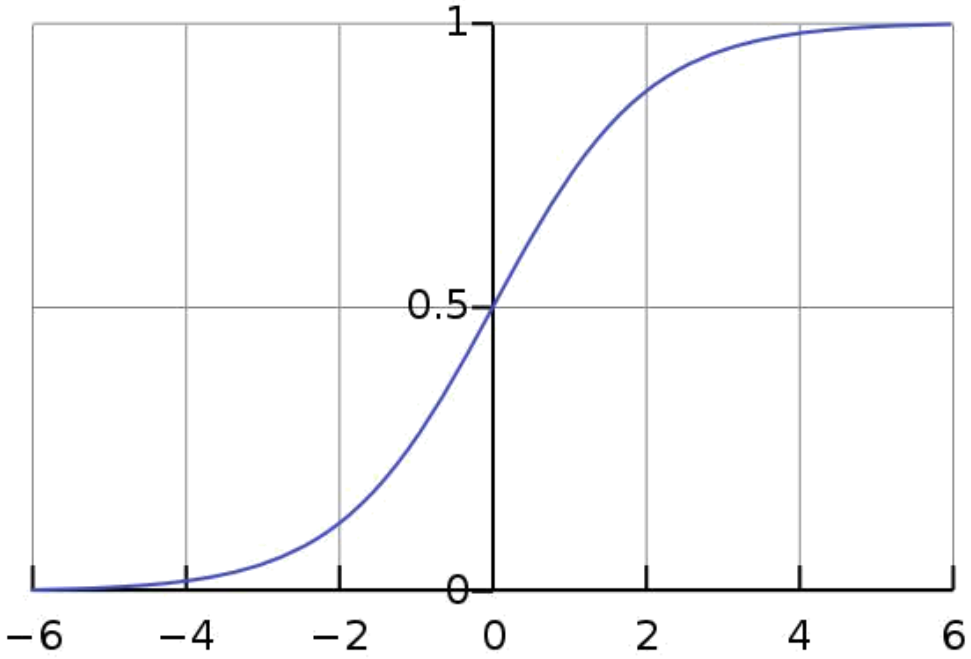


Figure 12: Graphic representation of SoftMax function

3.3.8 Soft maximum with loss layer

The soft maximum with loss layer is used during the training phase. This layer that calculates for each iteration the loss of the output of the network respect the target and back propagate it to the layer before. It also writes the value in a text file.

The soft maximum with loss layer produce output that are between 0 and 1 and also for this reason is often used when the output of the network is the probability to be of a certain class.

3.3.9 Euclidean loss layer

The Euclidean loss layer takes as input the target of the training and the output generated by the network, compute the loss with the MSE method

and back-propagate it to the previous layer. It also writes the value in a text le.

3.3.10 Accuracy layer

The accuracy layer is used during the training. It compares for every iteration the output of the network with the target, calculates the accuracy and writes it in a text file.

3.4 Test and validation protocols

To validate the result of the project we use the cross-validation test [29].

For the denoising case we used 80 images for the training set and 5 images for the test validation set.

The images were randomly chosen and we repeated the training and the test phase three times with 3 different dataset.

The results are evaluated with a histogram in which is shown the error of the output image of the network respect to the target compared to the error between the input image and the target.

For the denoising case we use a dataset of 135 MRI brain images, 45 for each class.

We perform a cross-validation test also on this dataset: we dived randomly the data set in 90 images for the training and 45 images for the testing, then we train the network and test the accuracy on the test set. We repeated the experiment three times.

The accuracy of the classification is calculated with the classical accuracy index: we take the probability output of the network for right class of the image for every image of the test set, we summed them all together and then we calculated the mean. The accuracy on the test set is calculated for every 2000 iterations and shown in a graph and it is calculated as the mean of the ratio of number of correct predictions to the total number of input samples.

4 Cases

The goal of our study is two folds:

- *Denoising;*
- *Differential diagnostic classification.*

Even though the ultimate goal would be the later, it was a necessary step to go through the former goal in purpose to really understand the impact of different strategies of layer combination to build a safe classification algorithm.

4.1 Case 1: Denoising

The implementation of the Denoising neural network follows the goal to create a well train image filtering capable to reach the results of an adaptive non-local means denoising process with spatially varying noise levels [30].

An adaptive local mean filter does not make any assumption about the noise distribution across the image, making it a perfect candidate for medical images.

The drawback of this method is the processing takes a long time and are a heavy step in image processing pipelines.

In this section we are going to show that a well designed neural network can reach the level of an adaptive non-local filter using a weak time-stamp.

4.1.1 Dataset

For the denoising case we used 3D structural FLAIR images from 80 subjects, as target the denoised images.

Data augmentation consists of rotation of 9 degrees. The total number of images used as input was 320. Then the images are cropped and saved in HDF5 format as indicate preprocessing paragraph.

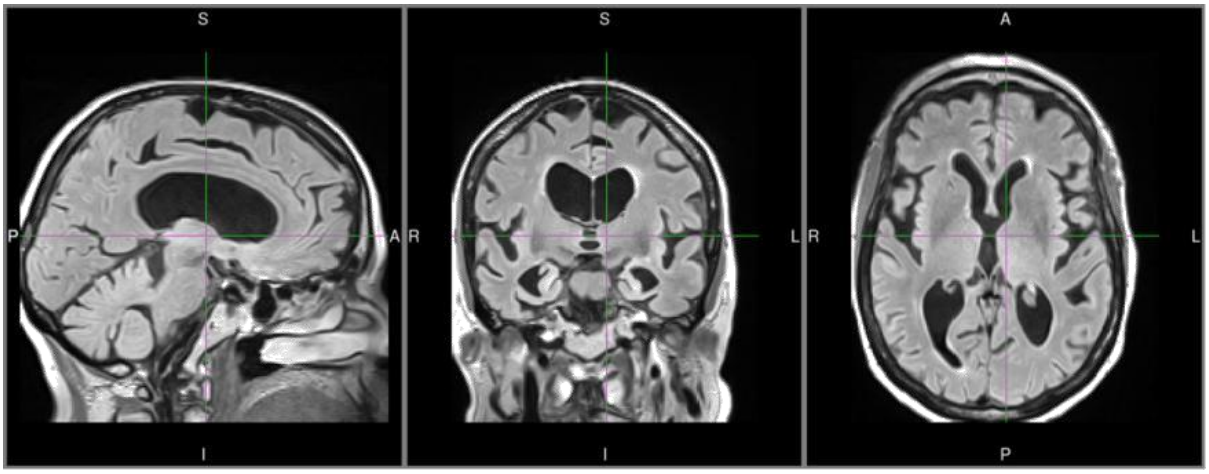


Figure 13: Example of MRI used as input

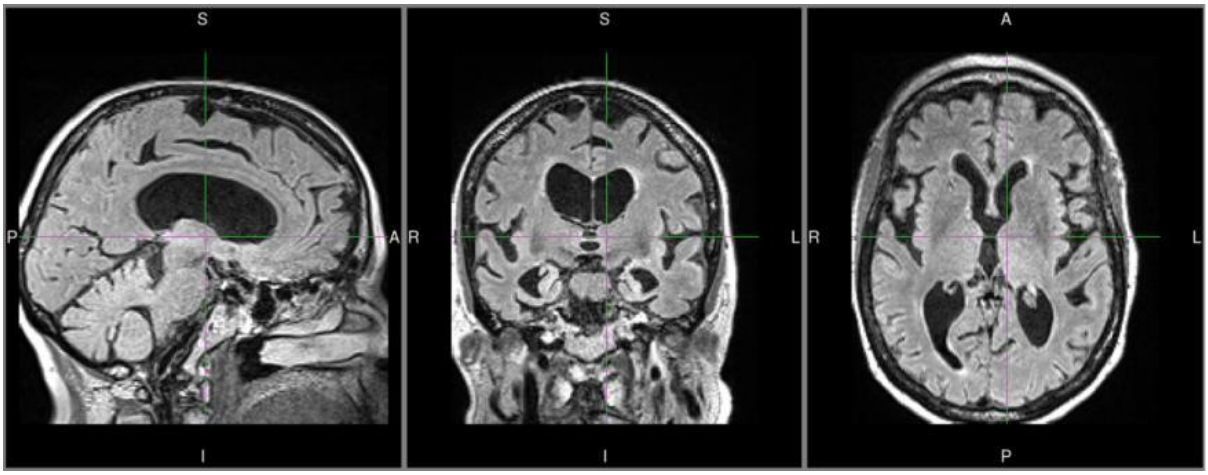


Figure 14: Example of MRI used as target

4.1.2 Training hyper-parameters

In this study we used the Adam optimizer (see Appendix for a detailed description).

Adam optimizer requires 4 hyper-parameters:

- *Learning rate*
- *Beta1*
- *Beta2*
- *X (Epsilon)*

For the denoising case we used the values of the hyper-parameters suggested by the original paper of Adam [31].

4.1.3 Neural Network architectures

For the denoising study case we start to build the simplest neural architecture (autoencoder) [32].

As loss function we choose the mean square error, also called Euclidian loss function.

The network architecture consists of: one convolution and one deconvolution with kernel of *n x n x n* voxels.

Where *n* is the number of voxels that will define the size of the kernel. For example, If *n = 1*, the output will have the same dimension as the input image.

Additional tests were performed using an additional layer ReLU.

The experiment done are the following:

- *Model 1* : 1 convolution and 1 deconvolution with kernel's size of 1, stride of 1, padding of 0 and number of features generated 32. The deconvolution layer has these parameters: kernel of 1, stride of 1, pad of 0 and number of features generated is 1, to get back the image to his original dimension. The input layer is the HDF5 Data layer that take the dataset of HDF5 file and generated input and target of the network. The loss function is the Euclidean loss (or MSE loss) that is the most used when the output are not classes.

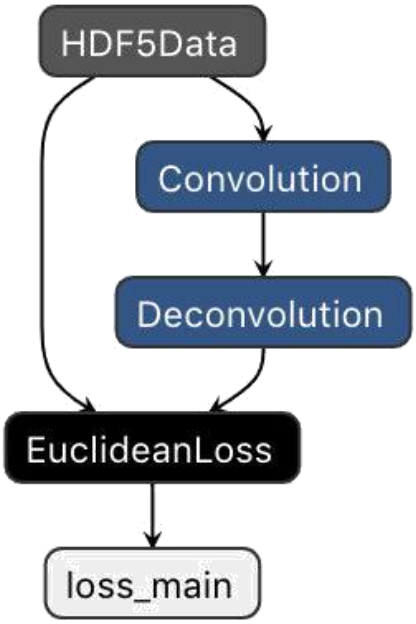


Figure 15: Scheme of the model 1 of denoising case

- *Model 2*: 1 convolution and 1 deconvolution with kernel of 3. The second model that we build is identically to the first one with the only exception that the kernel size of the convolutional and deconvolutional layer are of 3 instead of 1. The kernel of 3 is less computational efficient but analyzing the voxels at 3 x 3 x 3 groups has to recognize and reduce better the Gaussian noise.

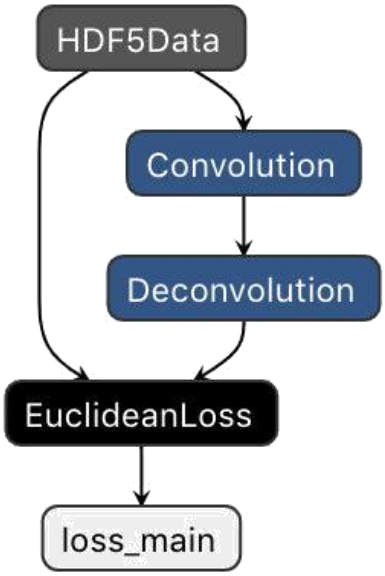


Figure 16: Scheme of the model 2 of denoising case

- *Model 3*: 1 convolution and 1 deconvolution with kernel of 5. This model is the same of the first 2, with the only difference that the kernel stride is larger: 5. This larger kernel is less efficient on the computational cost but with a larger kernel maybe is able to reduce the noise better than the model with the kernel of 3.

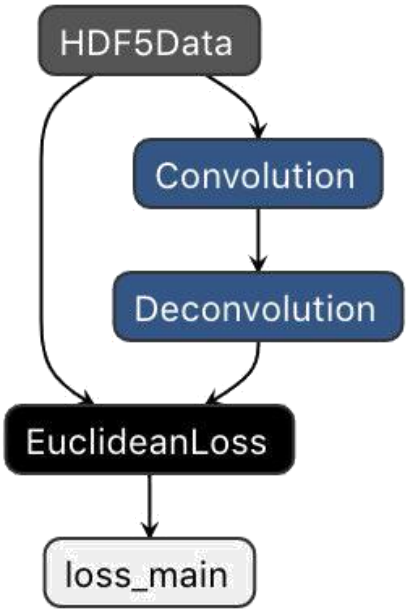


Figure 17: Scheme of the model 3 of denoising case

- *Model 4*: 1 convolution and 1 deconvolution with ReLU with kernel 3. This model is the same has the second, with the kernel size of 3, with the introduction of ReLU layers before the convolutional and deconvolutional layers. The motivation of the introduction of ReLU layers in the model is that the ReLU function is used to process better no linear features, and in the case of the structure of the brain we are dealing with a lot particular that are not linear at all.

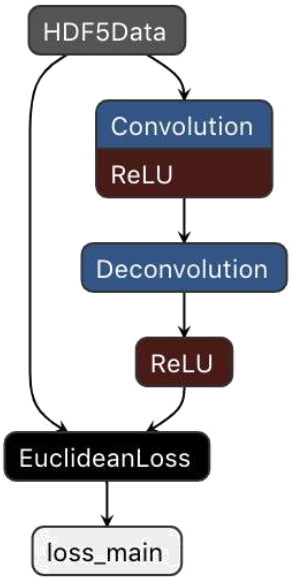


Figure 18: Scheme of the model 4 of denoising case

- *Model 5*: Three convolution and deconvolution. The last model we experiment is a more deep convolution and deconvolution autoencoder. In this new model the parameters of every convolution layers are: stride of 2, padding of 1, kernel size of 3 and the number of features created are 16 for the first, 32 for the second and 64 for the third. The deconvolutional layers have same kernel, padding and stride but the number of features generated is the opposite: 32 for the first, 16 for the second and 1 for the last. The reason why we tried this model is that a deeper convolution processes more features in their details and maybe

going deeper with the convolution and then rebuilding the same image with a deeper deconvolution layer can generate a better denoised image.

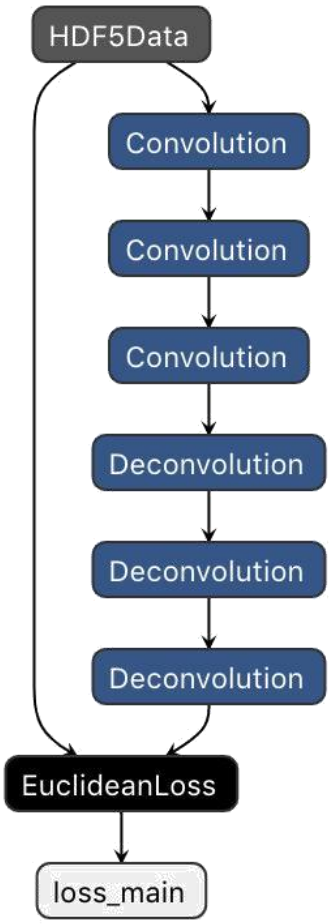


Figure 19: Scheme of the model 5 of denoising case

4.2 Case 2: Differential diagnostic classification

4.2.1 Dataset

The classification dataset consist in 120 images, 40 for each class that are control, pib negative and pib positive.

The preprocess, if the images consist only in cropping and data augmentation, creating new 3 rotated images from the original.

After data augmentation the hole dataset contains 480 MRI brain images. For the training is used 3/4 of the dataset, the other part is used for the test phase.

4.2.2 Training hyper-parameters

Also for this case we used the Adam optimizer. We start using the hyper-parameters suggested by the paper but then we reduce the learning rate to 0.000001 because 0.001 was to high learning rate for this task.

The batch size for this training is of 4 that is the biggest batch size possible with the memory of the GPU.

4.2.3 Network architecture

The fundamental element to build a image classification neural network are:

- *Convolutional layers*: they are the first layers that process the image and extract the features that classify the image

- *Fully connected layer*: after the convolutional layers, a fully connected layer is need to vectorize the image and before the softmax layer
- *Softmax layer*: it is the last layer and his output are the image's probability to be of a certain class [33].

We try different architecture to understand which is the best for this task.

From the denoising case we understood that the kernel of three is the best to catch all the features of the brain so we start experiment a convolutional neural network using that kernel size.

The architecture tested are these:

- *Model 1*: 4 convolution layers. The first model we try to use for the classification task is a 4 convolution layers with one ReLU layer, one fully connected layer and soft max with loss layer. The 4 convolutional layers has stride of 2, kernel size of 3, padding of 1 and the number of features generated is 32, 64, and 128. The ReLU layer is present to process better the non linear characteristic of the brain. The fully connected layer has to vectorize the image features generated by the convolutional layers. This layer then connect with the softmax with loss layer that is the one that calculate the prediction error and back propagate it. The accuracy layer is used during the training only

to write in a text file the values of the accuracy of the image to be classified in the right class by the network.

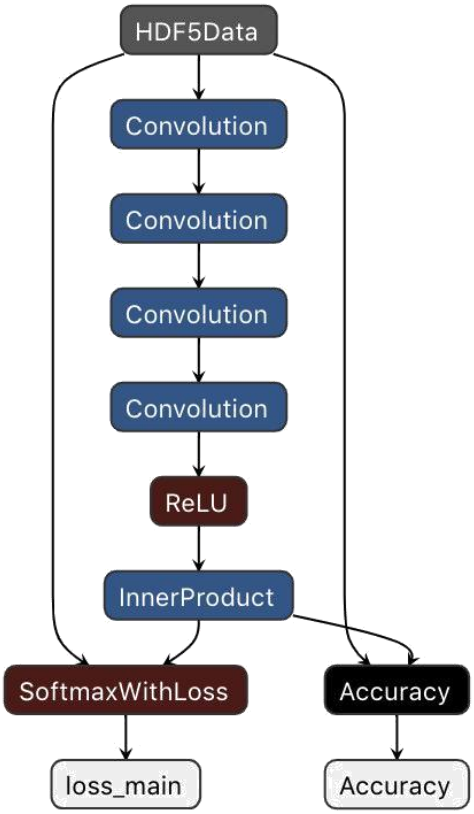


Figure 20: Scheme of the model 1 of classification case

- *Model 2*: 4 convolution layers with first layer stride of 1. The second experiment for the classification task is the same network as the first but with the first convolution layer that has a stride of 1 and padding of 0. This change makes the first layer analyze the features of the brain image in the original dimension. This is less computational efficient because increase the number of parameters to train, but analyzing the image in the original dimension it can also extract more features that characterize the brain class respect reducing the dimension of the image form the first convolution. After the 3 convolutional layers there are a ReLU, to analyze better the non linear part of the brain, the fully connected or inner product layer and the soft max with loss layer as the cost function.

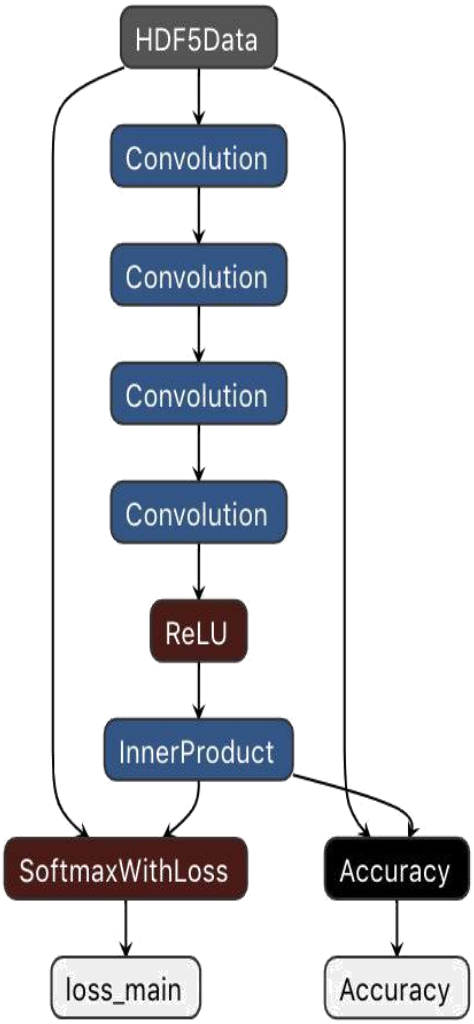


Figure 21: Scheme of the model 2 of classification case

- *Model 3*: 4 convolution layers and ReLU. The third experiment consist in a network similar to the previous one but with the introduction of ReLU layer after every convolutional layer. The ReLU layer is used to process non linear and has the brain is

made only but non linear details, the introduction of the function after every convolutional layer can improve the processing and the result of our model. The parameters of the convolutional layers are the same of the second network and the loss function used is still softmax with loss.

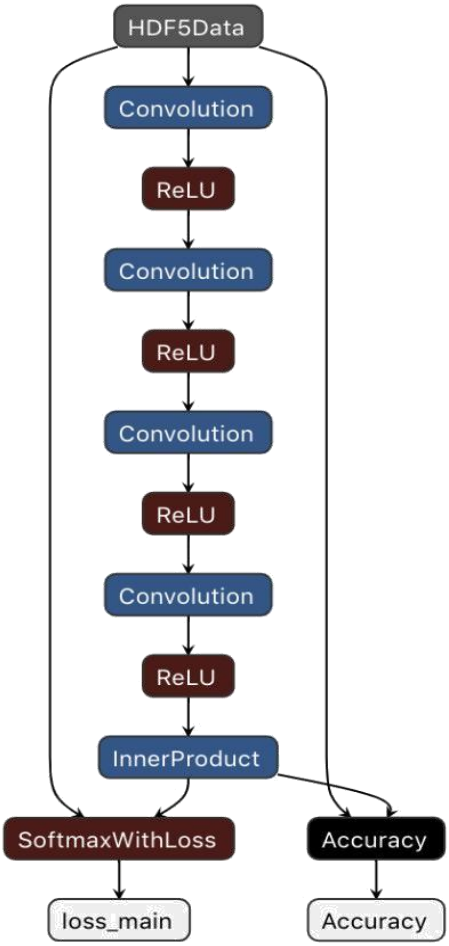


Figure 22: Scheme of the model 3 of classification case

- *Model 4*: 4 convolution layer, ReLU and pooling. The last experiment we try is the introducing of the pooling layer. In this case the stride of the convolutional layers is left at 1 and the pooling layers make the work to reduce the image of half. The pooling layer is often used in the image processing task in a block with first convolutional layer and ReLU. We reproduce this in our model putting together three blocks composed by convolutional, ReLU and pooling. After the 3 blocks there is a fully connected or inner product layer before the soft max with loss layer to calculate the loss and the accuracy layer to calculate the prediction accuracy during the training.

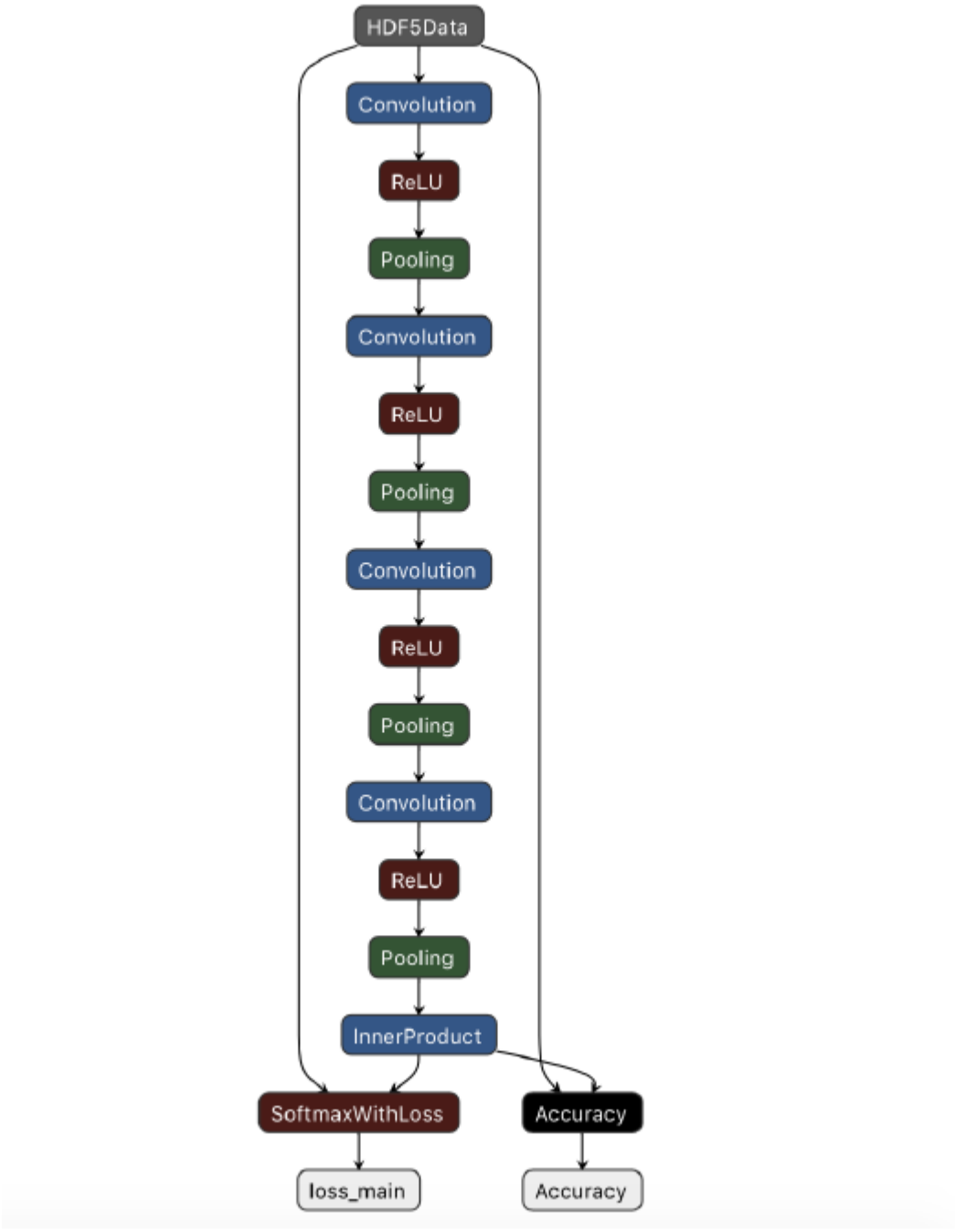


Figure 23: Scheme of the model 4 of classification case

- *Model 5*: 4 convolution layer, ReLU and batch normalization. In this experiment we introduce a batch normalization layer after every convolutional and ReLU layer to make the training more stable and effective.

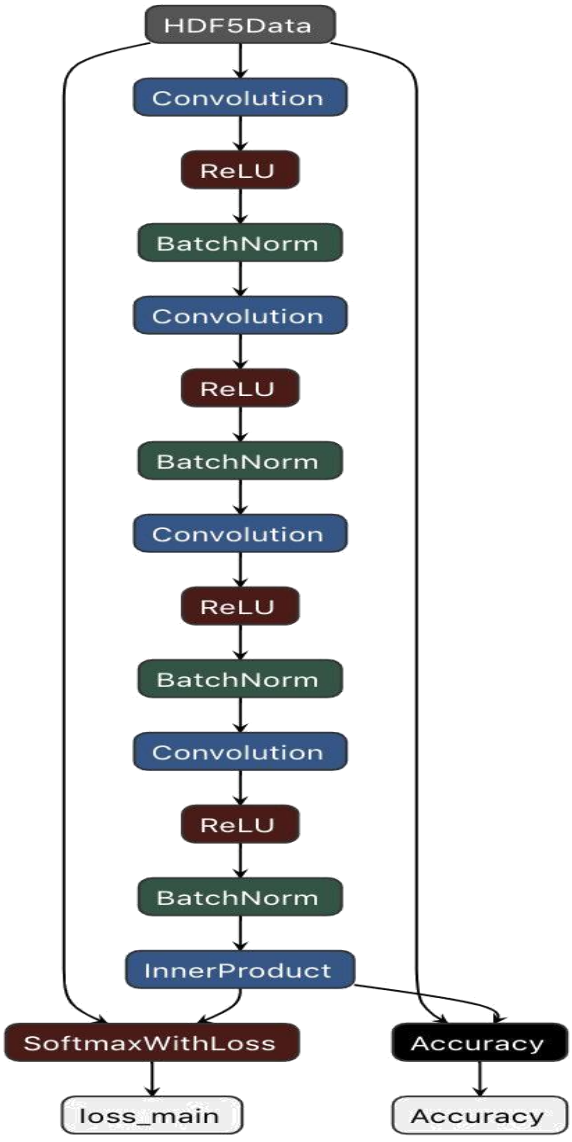


Figure 24: Scheme of the model 5 of classification case

5 Result

5.1 Case 1: Denoising

For the denoising case the image are evaluated as the difference of the output image, generated by an image non contained in the training set, respect the target image. This error is compared with the error of the input image respect the target image. To graphically show this result we build a histogram the show the two errors.

On the next result part is also showed the loss function during the training to analyze how the loss function decrease during the training.

- *Model 1*: 1 convolution and 1 deconvolution with kernel of 1. Low performance, the kernel is to small and cannot improve the gaussian noise present in the MRI images.

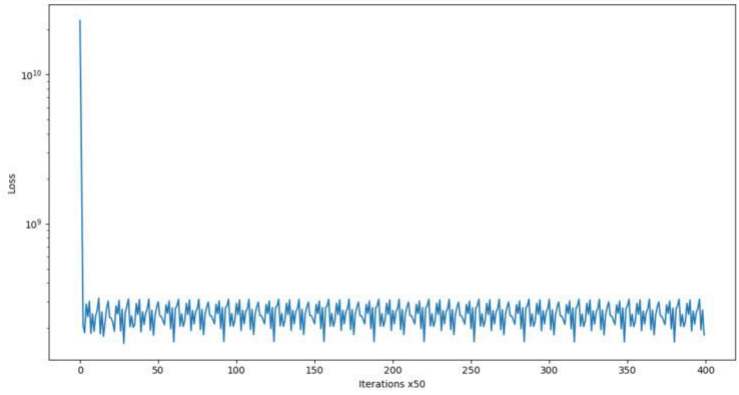


Figure 25: Loss function during the training model 1

The loss function shown in the previous images show that the number of iterations needs to reduce the loss function to his minimum are only

around 500, but the result of the network on a test image show that the network is only able to recreate the image as the input of the network.

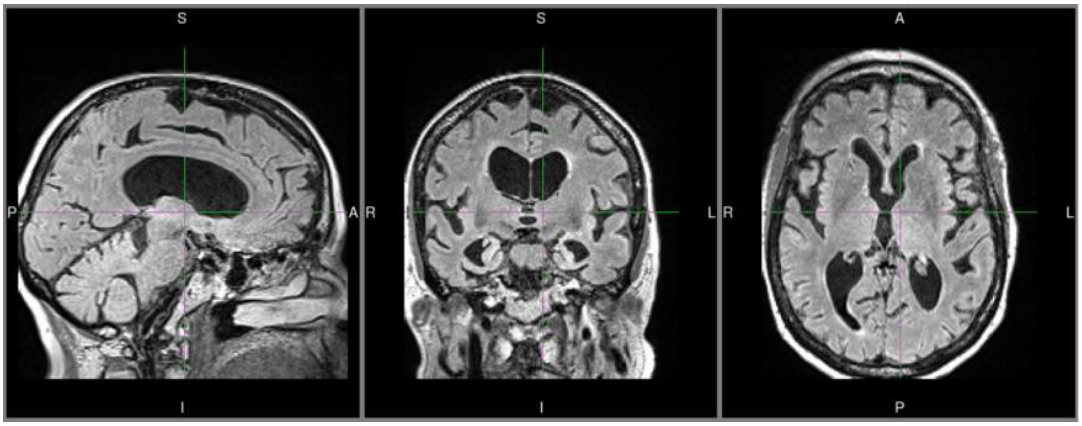


Figure 26: Image generated by the network model 1

The histogram representing the error of the output image generate by the network respect the image (green) versus the error of the input image respect the target (blue), shows that the model is not able to reduce the noise of the image and generate a better image.

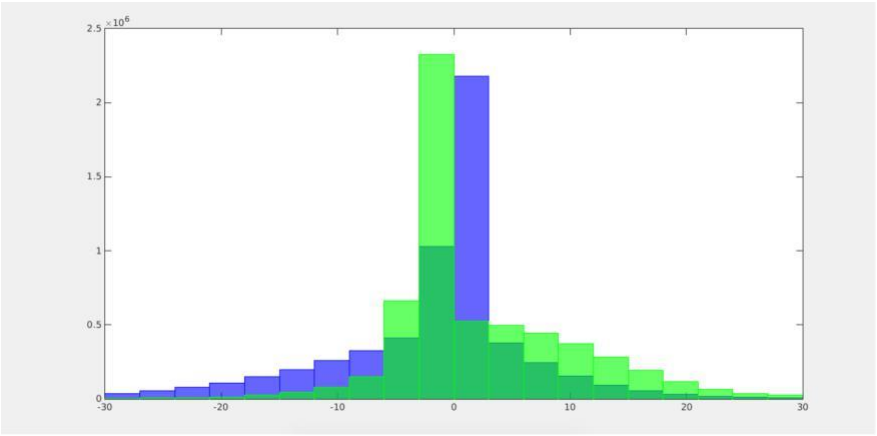


Figure 27: Histogram representing the error of the input(blue) and the output(green) respect the target model 1

- *Model 2*: 1 convolution and 1 deconvolution with kernel of 3. Good result and very efficient network. The dimension of the kernel is sufficient to reduce the white noise and the number of parameter is not increase to much respect the one with kernel 1.

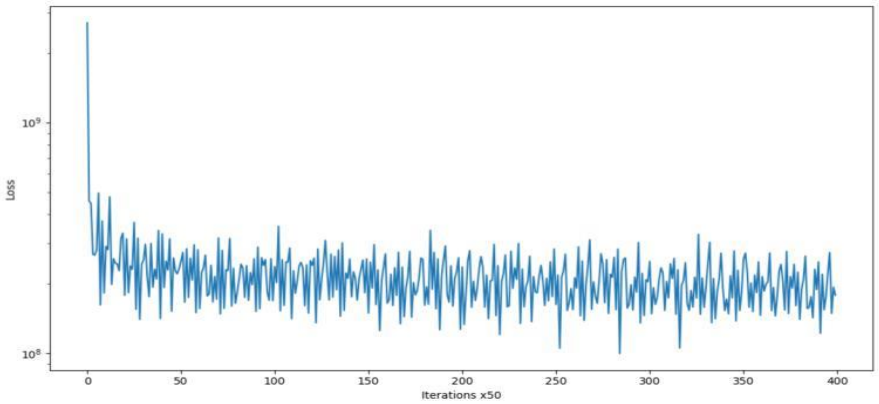


Figure 28: Loss function during the training model 2

The loss function and the output generate by this network shown better result respect the network with kernel of 1. The loss function graph shows that the function reaches a smaller value, the number of iterations to reach the minimum is close to the one of the previous experiment. The output generated by the trained network shows a better image respect the input one in which the Gaussian noise is reduced, the quality of the output image is still not good as the target image.

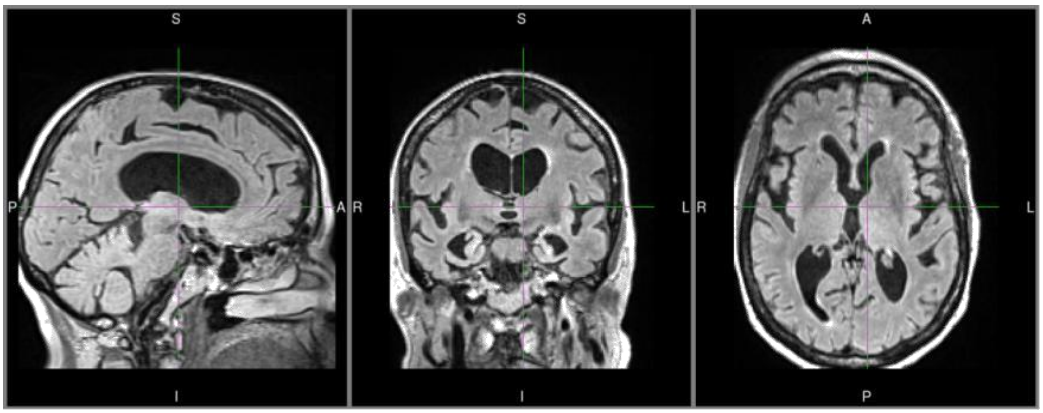


Figure 29: Image generated by the network model 2

The histogram shows that the output image has a smaller error compared with the previous experiment.

Also the big part of the error become more close to the center of the histogram that represent a small error.

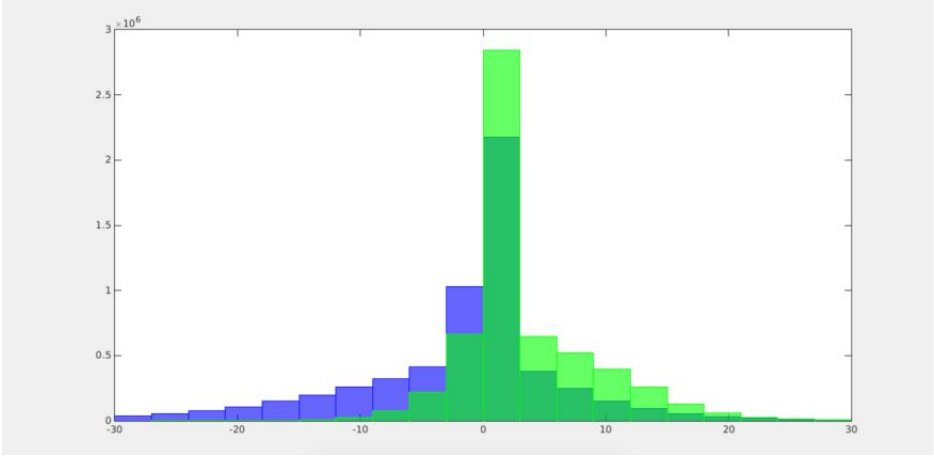


Figure 30: Histogram representing the error of the input(blue) and the output(green) respect the target model 2

- *Model 3*: 1 convolution and 1 deconvolution with kernel of 5. Same result with kernel of 3 but the network is less efficient the number of parameters to train are more.

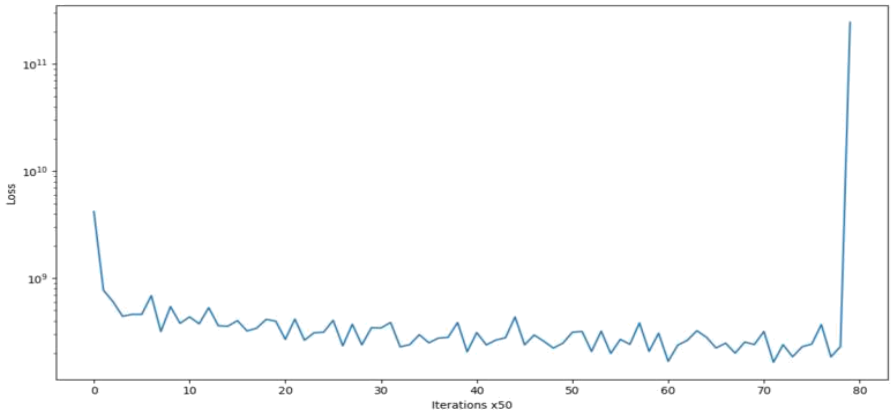


Figure 31: Loss function during the training model 3

The performance of the network with the kernel of 5 is the same of the one with kernel of 3 in term of result, instead in term of computational performance is lower. This can be seen in the loss function graph in which the number of iterations need to reach the minimum are bigger respect the previous experiments, but the value reach and the outputs of the network are the same of the experiment with the kernel of 3. The reason why this model is less efficient in term of computational performance is that a kernel of 5 has more parameters to t respect a kernel of 3 and it not shows better result to justify his use.

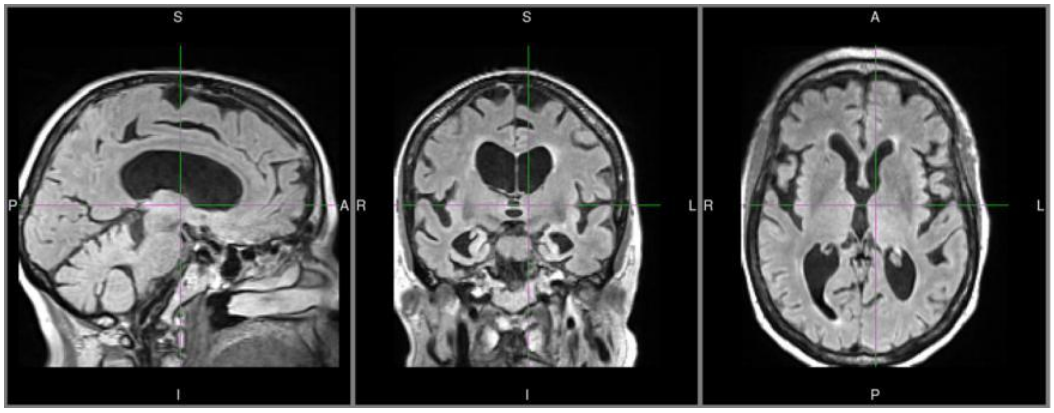


Figure 32: Image generated by the network model 3

The histogram that compares the output of the target respect to the target shows that the quality of the image generated is the same of the previous experiment. Also from the previous image, comparing it with the one of the experiment with kernel of three, we can see that the white noise is reduce and the image quality is the same as the previous experiment.

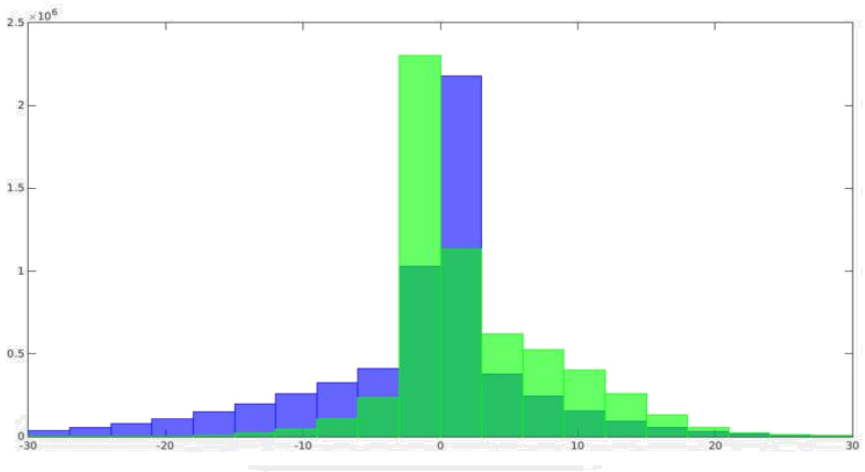


Figure 33: Histogram representing the error of the input(blue) and the output(green) respect the target model 3

- *Model 4*: 1 convolution and 1 deconvolution with ReLU with kernel 3. After choosing the best kernel size we introduce a ReLU layer after the convolutional and the deconvolution. The ReLU layer improve the result of the network.

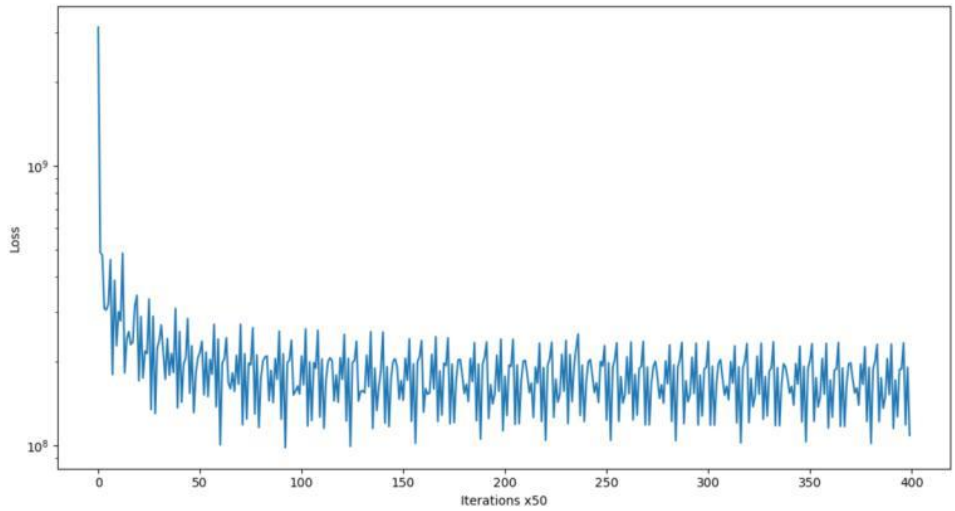


Figure 34: Loss function during the training model 4

From the next image we can see that this experiment shows the best result in term of quality of the image generated. The introduction of the ReLU layers helps the network to understand and reproduce smaller and less linear features.

The graph of the loss function during the training shows that the number of iterations too reach the minimum value is still small, around 2500 iterations. This means that our model is also computation efficient.

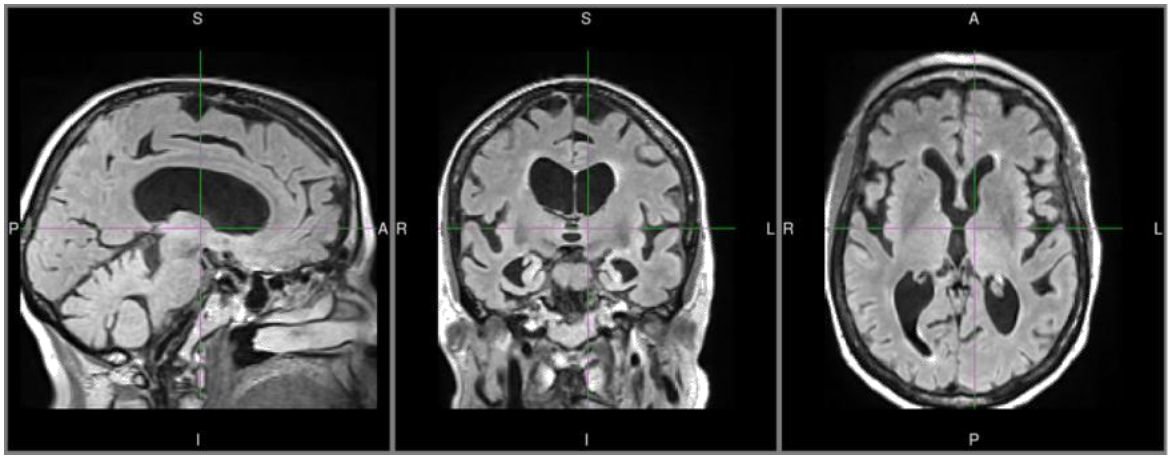


Figure 35: Image generated by the network model 4

The histogram that compares output image of the model respect to the input shows that the error is smaller respect the other case, this means that our output image is very close to the target one and that the Gaussian noise is drastically reduced.

This can be notice also comparing the visual image generated by the networks in the different experiments.

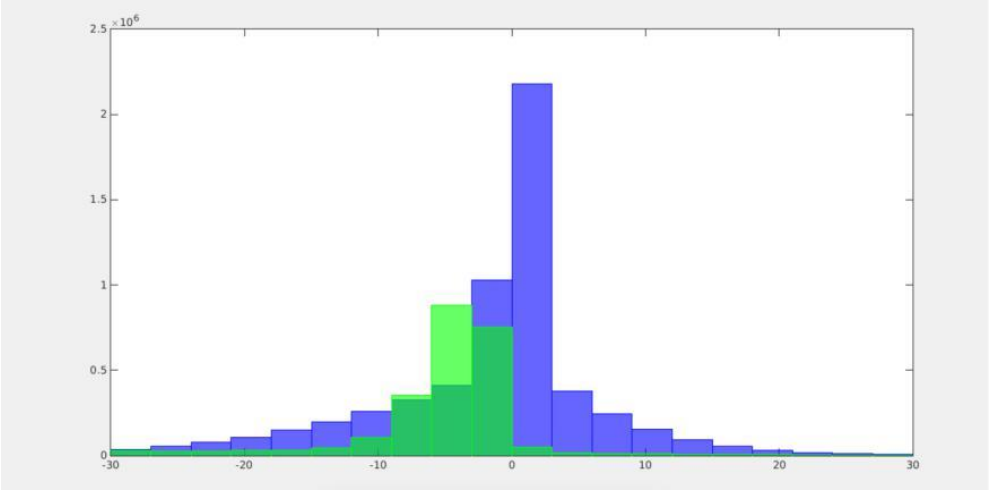


Figure 36: Histogram representing the error of the input(blue) and the output(green) respect the target model 4

- *Model 5*: 3 convolution and 3 deconvolution. The 3 level of convolution and deconvolution shows same result as the network before, but the number of iterations needed is 10 times more.

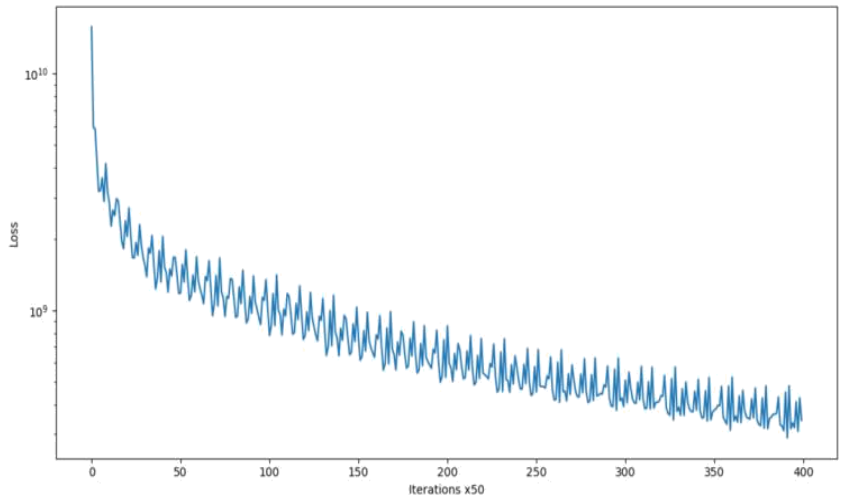


Figure 37: Loss function during the training model 5

The last model that we experiment shows that it needs a bigger number of iterations to reach his minimum.

The reason why is that this model has 3 level of convolution and each generated more features respect the previous layer, this make the model has an increased number of parameters to train respect to the other experiment and so it needs at least 100’000 iterations to reach his minimum. The result shows a image that is not good as the one of the previous experiment but still the Gaussian noise is reduced and the image is better respect the input one.

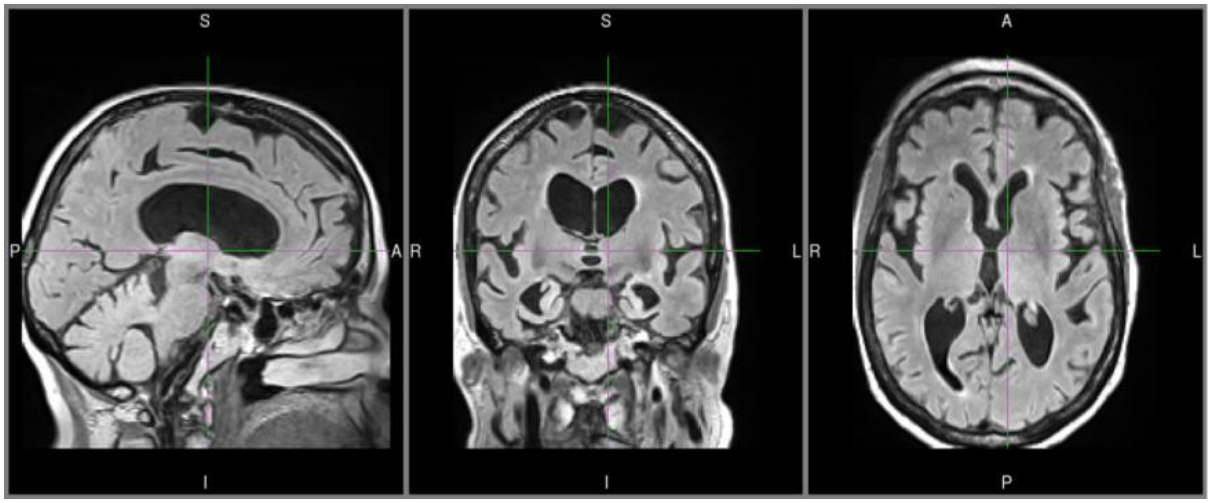


Figure 38: Image generated by the network model 5

The histogram shows that the error is bigger respect the one of the previous experiments. The reason why is that using a small dataset we cannot deploy a big network with a big number of parameters because is generated an over fitting problem caused by the too high number of parameters to fit.

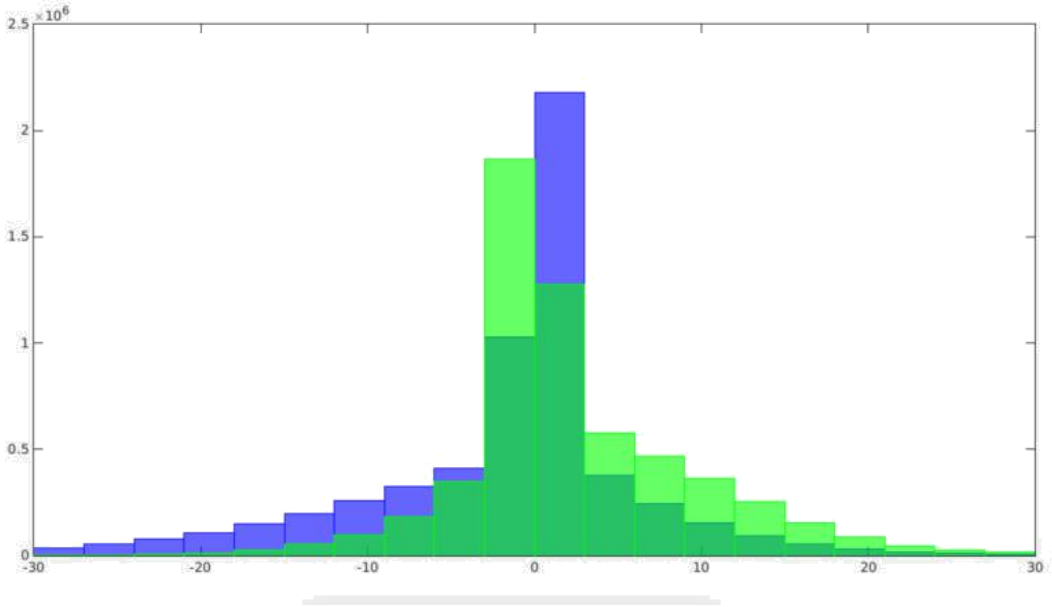


Figure 39: Histogram representing the error of the input(blue) and the output(green) respect the target model 5

5.2 Case 2: Differential diagnostic classification

- *Model 1*: 4 convolution layers. The training of the first network experiment shows that this network is not able to classify and recognize the differences between the 3 groups. This can be seen in both the loss graph, in which the loss function is not stable and decrescent, and in the accuracy one, that not reach the 70%.

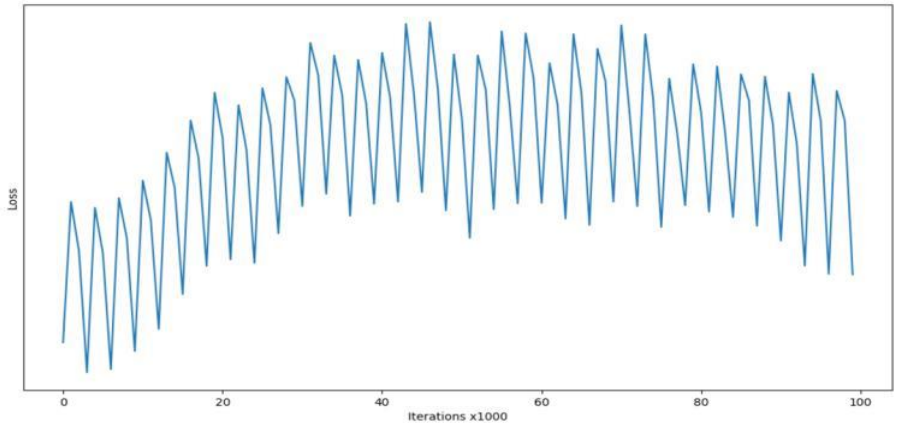


Figure 40: Loss function during the training model 1

In both the graph we can also notice that the training is not stable and continue to oscillate between a minimum and a maximum. The reason why is that the architecture of this network is not compatible with the task that we want to implement.

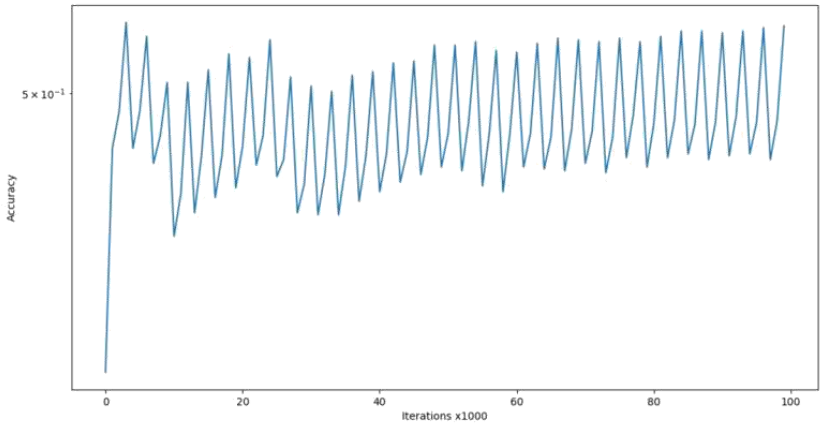


Figure 41: Accuracy during the training model 1

- *Model 2*: 4 convolution layers with the first of stride of 1. The second experiment shows better result. Processing the image with the original dimensions in the first layer the network is able to classify and recognize the difference of the training set images. The loss function is still not stable but is decrescent, the same for the accuracy that reach very good value but is not stable during the training.

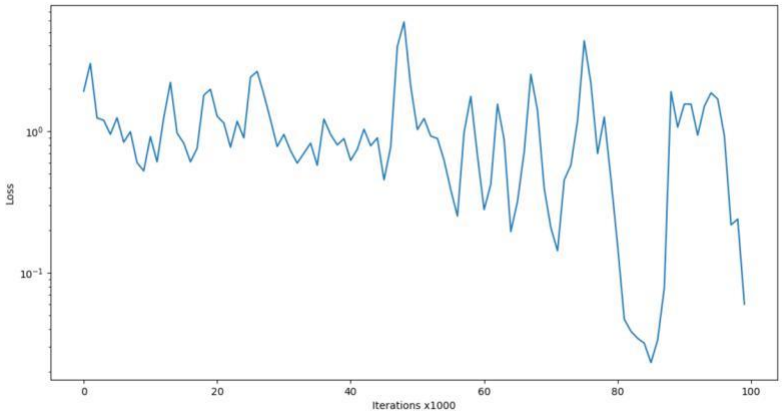


Figure 42: Loss function during the training model 2

This network shows to be able to classify the image of the training set but, using too much iterations to reach the minimum for the loss function and the maximum for the accuracy function, there will be the overfitting problem if we want to use the model with a new data not present in the training set. The reason why is that we use too many times the same image during the training to reach that number of iterations and the consequence

is that the model is too much trained on the training set images and is not able to generalize to image not present in the training set.

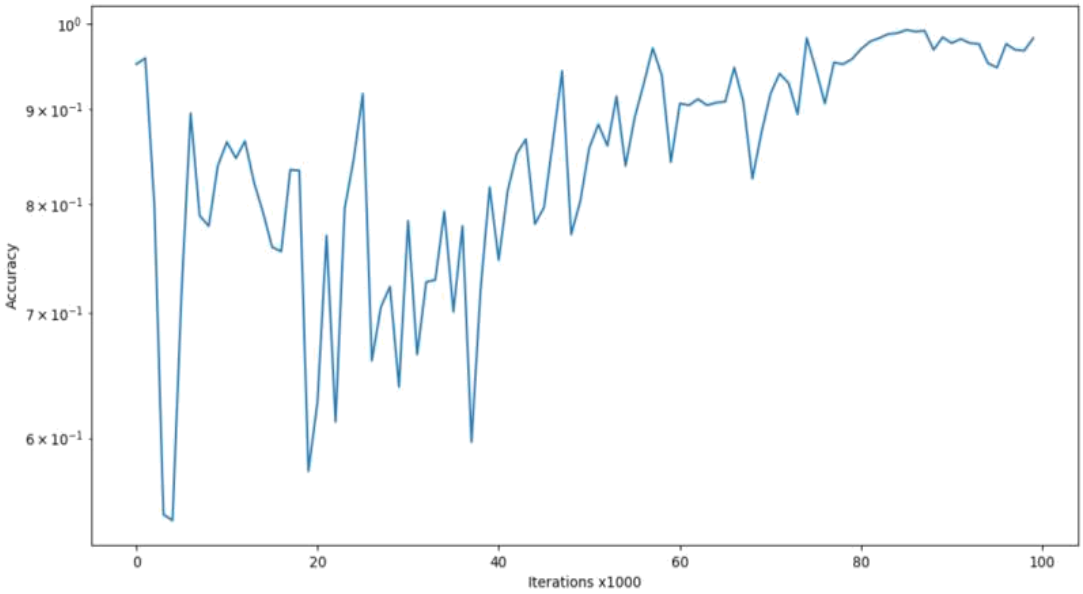


Figure 43: Accuracy during the training model 2

- *Model 3*: 4 convolution layers and ReLU. Introducing the ReLU layer after every convolution one make both the loss function and the accuracy more stable during the training.

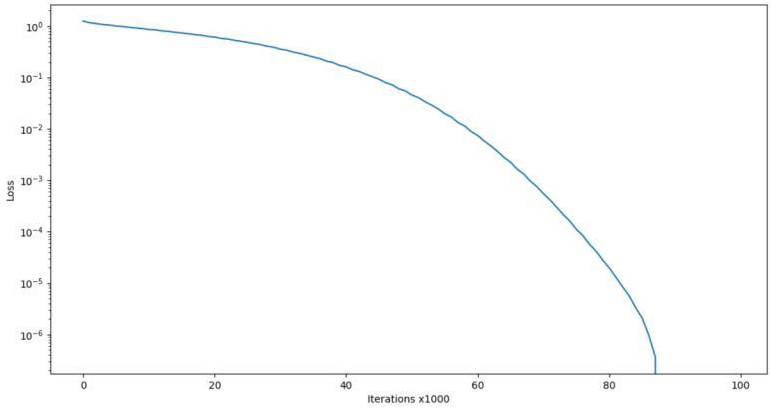


Figure 44: Loss function during the training model 3

This model shows a more stable and efficient training. The loss function decreases stable till reach his minimum. The opposite for the accuracy function that increase stable till reach his maximum of one. As the same of the denoising case the introduction of the ReLU layers after every convolutional blocks, makes the network more able to recognize the smaller and non linear details of the MRI brain scan.

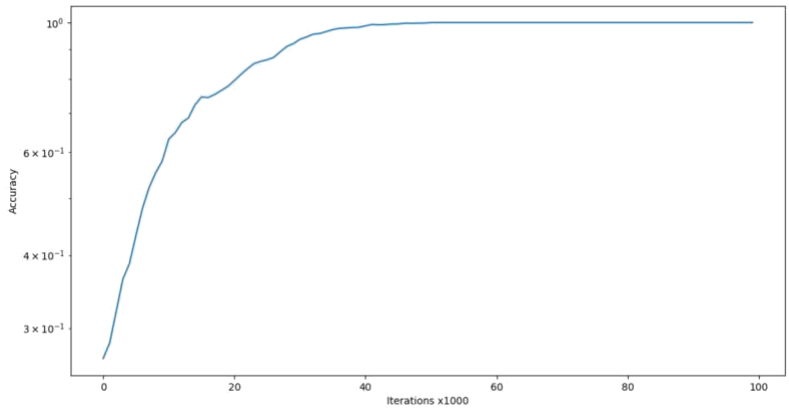


Figure 45: Accuracy during the training model 3

- *Model 4:* 4 convolution layers, ReLU and pooling. The introduction of the pooling layer not improve the performance of the network but it makes it worst.

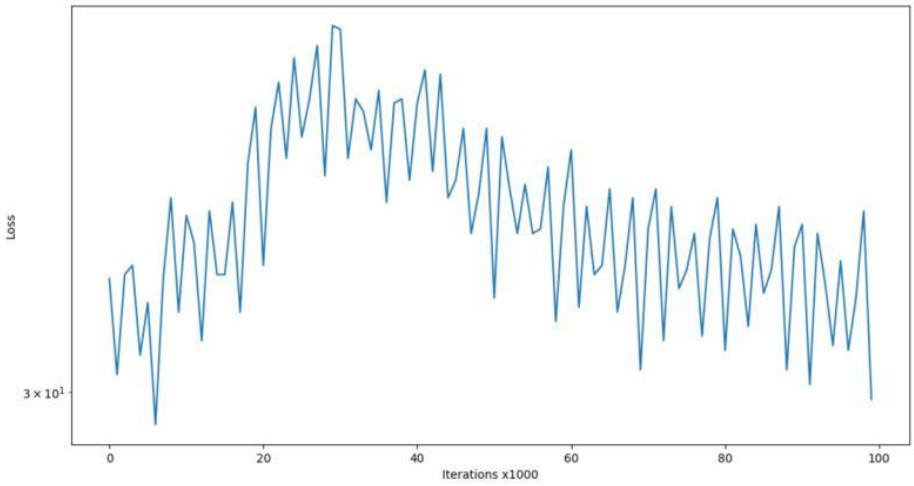


Figure 46: Loss function during the training model 4

As the same of the denoising case, the introduction of the pooling layer does not help the network to improve the result but instead makes it worst.

This can be notice by both loss function and accuracy graph, they are less stable and also the values that they reach are smaller for the accuracy and bigger for the loss function.

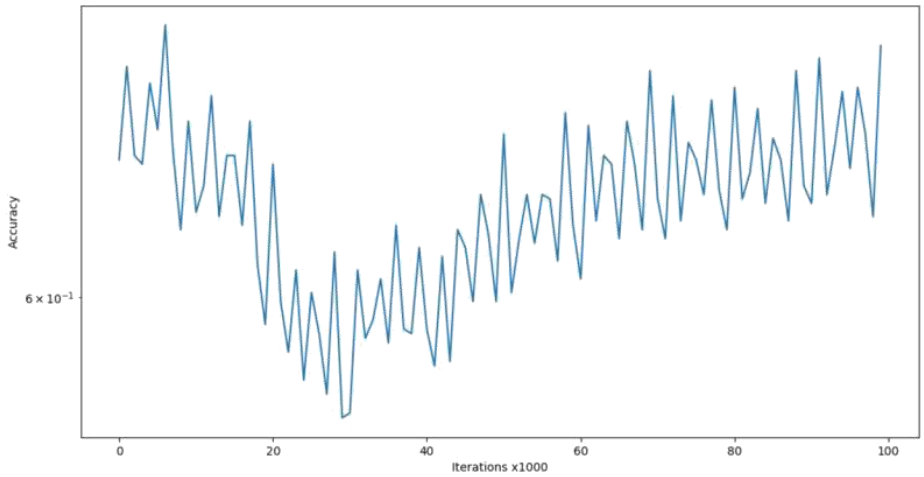


Figure 47: Accuracy during the training model 4

- *Model 5*: 4 convolution layers, ReLU and batch normalization. The introduction of the normalization layer produces the more stable and effective training.

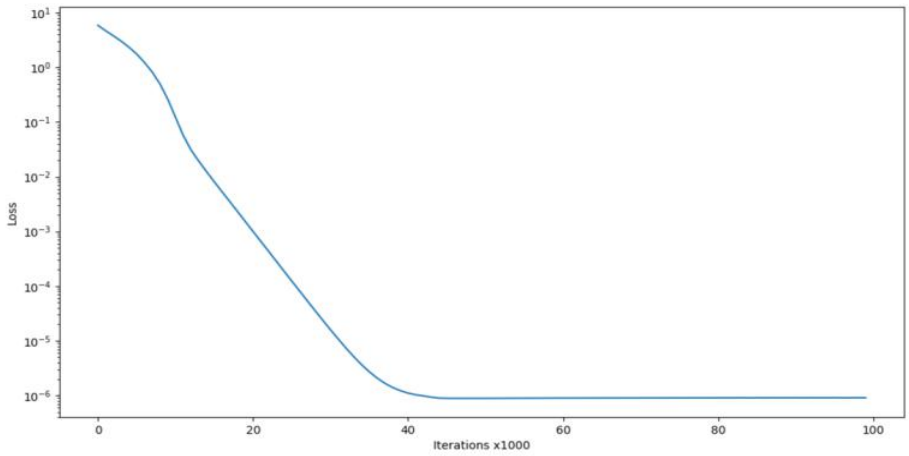


Figure 48: Loss function during the training model 5

The introduction of the batch normalization layer produces the best result of all experiments. The two graphs show that the values and the stable of the two functions are superior with this network respect the other case. The reason why is that using a patch based training, the process of normalization of the images after every convolutional block, makes the network more able to classify the images in the right group.

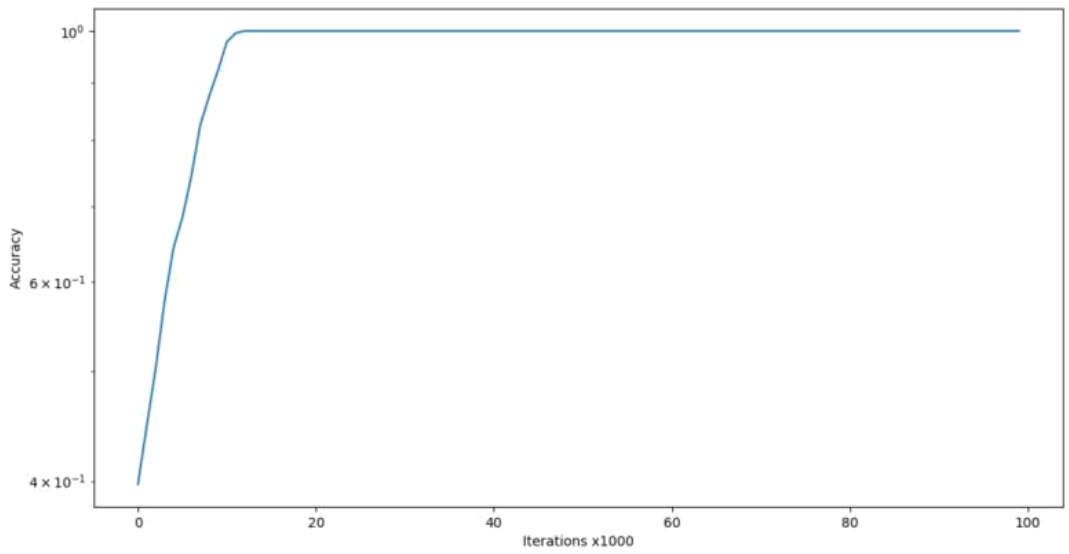


Figure 49: Accuracy during the training model 5

5.2.1 Test phase

The last model is been test with the cross-validation accuracy test described in the methods section. The reason why we test the accuracy on all the training iterations is to understand which is the best number of iterations for the model and when over fitting problem on the test data start to influence the model.

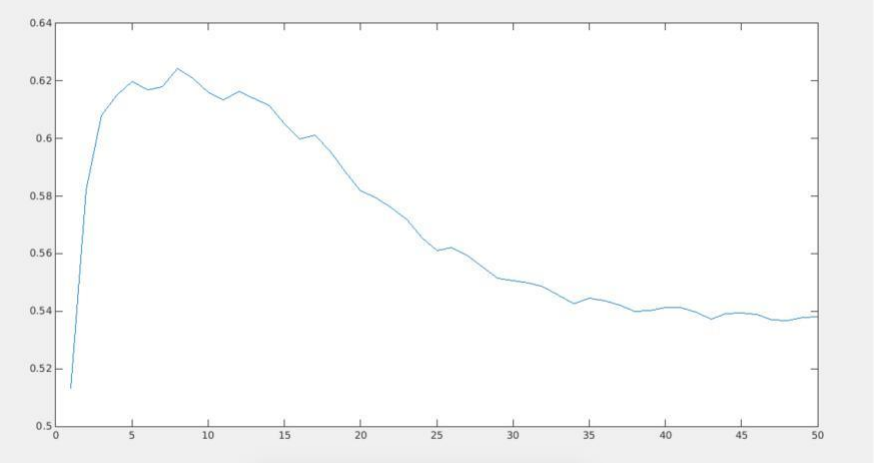


Figure 50: Accuracy on the test set every 2000 iterations model 5

The graph shows that the number of iterations to reach the maximum accuracy on the test set are 10’000, this mean that our model is computational efficient.

The value of accuracy reach is low 0.63, but also from the literature we see that the number of images that we have is too small to reach better result in term of accuracy on the test set.

After the 20’000 iterations the overfitting problem starts to affect the model and the performance bringing it back till reach the 0.54.

The reason why is that the small number of images are used too much times during the training and so the overfitting problem start very early to affect the CNN during the test phase.

6 Discussion and conclusion

In this part I will introduce the main findings of this project and explain the innovative contributions and relevant aspects of the implemented methods.

Then I will tell about the limits and possible improvements for this work and i will end the chapter with a conclusion.

6.1 Main findings

In the next picture shows the result for the classification case.

In the first column there are the model, in the second the accuracy reached during the training and in last column shows the accuracy on the test set.

For the first and third model the accuracy is not calculated because the accuracy on the training set is too low to work on the test set.

Type of network	Accuracy on the training-set	Accuracy on the test-set
Model 1	0.63	
Model 2	0.97	0.52
Model 3	1	0.54
Model 4	0.74	
Model 5	1	0.63

Figure 51: Table of result of classification accuracy

The best CNN model that we created is still not able to perform a good classification on our clinical case.

The reason why is that the number of data is too small to create a convolutional neural network model that is then able to generalize to data that are not present in the training set.

Also if the main goal of the thesis is not was not reached during our experiment we understood which type of network architecture was the best to create the most performant model.

For what concern the network architecture we discover that the best block of layer to perform the classification task is composed by convolutional layer, ReLU layer and batch normalization layer in series.

The architecture that performs the best result is composed by the block previous described repeated for 4 times but with a small number of features created.

The 4 blocks of layers perform a deep processing of the image that is needed to recognize all the features of the brain. The few numbers of features created is for the overfitting problem.

With a small data set the overfitting problem become more relevant. With a low the number of features created there are less parameter to fit and this balance in part the problem of have a small dataset regarding the over fitting problem.

The add of the ReLU layer shows increase the efficiency of the artificial neural network on both the denoising case and the classification case.

The reason why is that this layer produce a better processing of the non linear features.

The details of the brain are all non linear structure and so the ReLU layer improve the performance of the network.

The batch normalization layer improves also the efficiency of the model. The reason is that all the images has different intensity caused by different external factor and so a normalization of the batch make all the image with the same threshold and more detectable by the other layer.

Another discover about the network is that the pooling layer, if insert in the basilar block, do not help the model to classify better the MRI brain scan, but produce worst result both in the classification case and in the denoising case.

6.2 Innovative contributions and relevant aspects of the implemented methods

The most important contribution that we bring is the best type for the most efficiency convolutional neural network for image classification with a small dataset of brain MRI.

The network consists in 4 blocks repeated in series, the block is composed by:

- *3D-convolution layer;*
- *ReLU layer;*
- *Batch normalization layer.*

To make the network effective also with a small dataset the number of features generated by the network has to be low because in this way we overcome the overfitting problem.

The pooling layer, if added to the principal block, produces worst result so we suggest to do not use this layer in visual processing task instead use a convolutional layer with the stride of two to reduce the dimension of the image during the creation of deep features.

6.3 Limits and possible improvements

The limits of the project are that with a small dataset of 30 images for each class is very difficult crate a model that can classify the image with an high accuracy.

For now is not present a dataset that contains more images about the 3 class the we try to classify.

The possible improvements that I would like to try to implement is to download the ADNI dataset that contain a more then thousand brain MRI, but divided only in two class (Alzheimer and control), and try to see which is the minimum number of images necessary to produce a model with a accuracy of at least of 90% on the test set.

This experiment can show us which is the minimum number of images that we need to build a good classifier also with the 3 classes.

Another improvements that I would like to try is to use the transfer learning.

Use the information stored by the network in the first case training for the second case.

To do that we create a model that has 2 output, in specific we add together the most efficient model of the denoising case with the most efficient model of the classification case.

Then we train the model on the denoising training set.

Finish the first training we make the second with our reset all the weight and biases of the network. This technique may allow to overcame the over fitting problem and generate a model more efficient also with the dataset present today.

6.4 Conclusion

The model generated was not able to predict the test data with an accuracy superior of the 70%.

The reason why is that the ANN technique need a number of data that is not always present especially in the medical field.

We understood and reported some architecture details that make the network more efficient also with a small number of data.

My work at UCSF is still not finish and we will try other implementation and technique in this field of machine learning like the transfer learning to implement a better model.

7 Appendix

In this part I will introduce the artificial neural network and all his components.

The main components of neural networks are the layers, the cost function and the back-propagation algorithm.

These concepts are before introduced in more general way and then I start to enter in the details also with the mathematical formulas.

7.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are computational models that have emerged as a result of a simulation of the biological nervous system to perform a variety of tasks.

As a resemble of the biological system, ANNs acquire knowledge through a learning process.

ANNs have been around for 50 years and basically they are a type of machine learning used in supervised learning.

Machine learning used algorithms to extract information from data and represent it in some type of model.

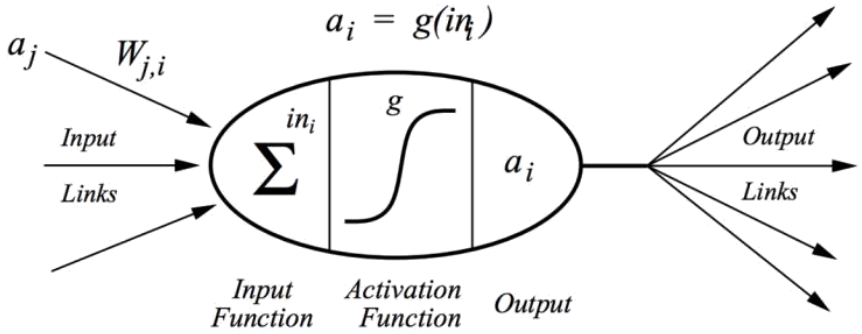
We then can use these models to infer things about other data we have not yet modeled.

Compared to the classical machine learning, ANNs attempts to create a network of “nodes” working in parallel as we assume the human brain does.

The first neural network was introduced in 1957 by Frank Rosenblatt as a simplified model of a neuron called the *perceptron* [34].

The perceptron is a linear-model binary classifier with a simple input-output relationship. The perceptron receives multiple input signals at multiplied by their weights W_i, sums them up, and feeds an activation function g(x) that defines an output *y*.

During the learning phase, the perceptron changes the weights to minimize the error until all the record inputs are correctly classified. (https://www.simplilearn.com/what-is-perceptron-tutorial)



$$a_i = g(\sum_j W_{j,i} a_j)$$

Figure 52: Main Processing Unit

The main limitation of the perceptron was its inability to solve nonlinear problems. Only years later, in 1974, the first non-linear processing capabilities of ANNs were reported and since then the interest of the

scientific community increase exponentially boosted with the increase in computational power together with the advances in computer technology [35].

ANNs are inspired by the biological neuron with three principal components:

- *Dendrites*: input connections with other neurons;
- *Cell body*: the central part of the neuron when the signal is processed;
- *Axons*: the output connections to other neurons.

ANNs are indeed represented as a set of nodes called neurons and connections between them. The connections have weights and represent their strengths.

The basic architecture of a neural network has 3 layers:

- *Input layer*: this layer is the interface with the environment;
- *Hidden layer*: this is the computational layer;
- *Output layer*: this is the layer where the output is stored. Different neural network architecture can be built using multiple layers. In particular, the use of multiple hidden layers could help in solving complex problems. This type of architecture defined the so-called multilayer Feed-Forward networks. (https://www.xenonstack.com/blog/artificial-neural-network-applications/)

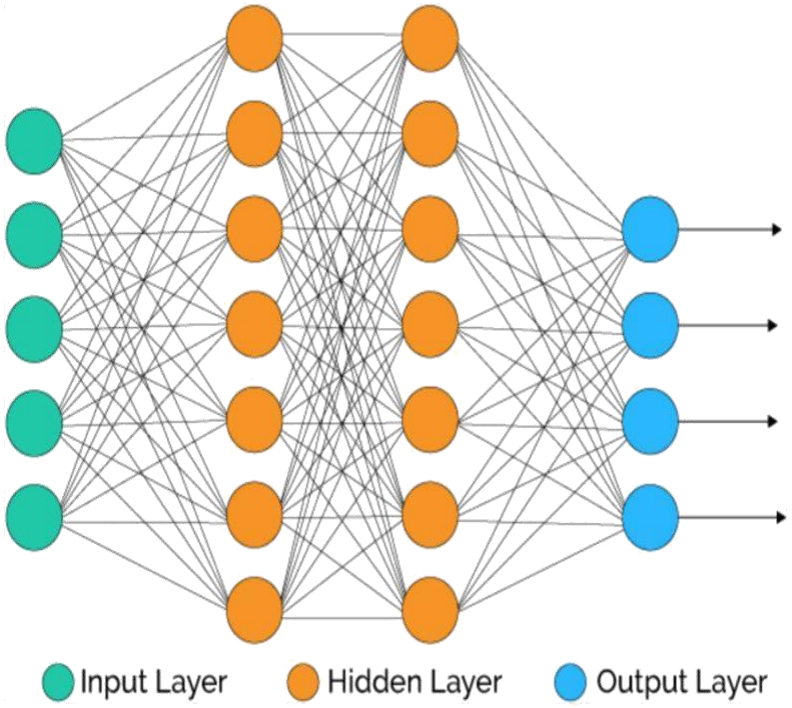


Figure 53: Example of multilayer feed-forward network implementing two hidden layers

Each layer has one or more artificial neurons which differs by each other for the activation function that serves to the specific purpose of the layer.

7.2 Activation functions and Forward Propagation

Activation functions control the behavior of the artificial network. Depending of the application, multiple activation functions are available. Most of the time the activation shape follows the sigmoid profile, simulating the ring process of neurons.

The most common functions are:

- *Linear*: function where the dependent variable has a linear relationship with the independent variable;
- *Logistic*: function that converts independent variable of near-infinite range into simple probabilities between 0 and 1. Its characteristic is then to reduce values or outliers without removing them;
- *TanH*: a hyperbolic trigonometric function, it transforms the independent variable to a range between -1 and 1. Its advantage is that can deal easily with negative number;
- *SoftMax*: generalization of logistic regression, it can be applied to continuous data and can contain multiple decision boundaries. This function is often used in a classification problem;
- *Rectified linear unit (ReLu)*: a function that activates a node only if the input is above a certain positive quantity. Above this threshold, the function has a linear relationship with the dependent variable [36].

7.3 Training Neural Networks

A characteristic of the neural networks is their iterative learning process. At each step, the weights and biases associated with the inputs are adjusted based on the correct label known in advanced (as in supervised learning).

The biases are added to the inputs to ensure that at least a few nodes per layer will be activated regardless of signal strength. Initially, random weights and biases are assigned, and the output is calculated using the activation functions in the hidden layers, the resulting output is then compared with the desired ones. Errors are then propagated back, allowing the system to adjust the weights and the biases, thereby allocating significance to certain bits of information or minimizing others.

During this phase, the algorithm adjusts the weights of the network only if the output does not match the label.

The “blame” of the error is though divided across the contributing weights. In a feed-forward multilayer network, this can be challenging because of the many weights connecting each input with the output.

The error is defined as the difference between the network output and the actual output value for the training example. The key is how we distribute the blame across the different layers of the network.

Each hidden node is responsible for a portion of error in each of the neurons to which it has a forward connection. Each portion is divided accordingly to the connection weight between the hidden and the output node.

In each layer is summed up by the total number of neurons, and progressively is updated for each layer.

7.4 Cost Functions

In the field of machine learning minimizing the cost function is the central point of the game.

The cost function represents the objective of a problem. For example, in a maximum likelihood problem, we try to estimate how much a set of data drift from a parametric function, *e.g.* a polynomial function, through a probability distribution.

The Gaussian probability distribution is used most of the time. Like every cost, the learning process is going to try to minimize it. In the example of the maximum likelihood, the optimization scheme will be to change the parameters of the parametric function to fit closer the set of points.

In other words, minimize the negative logarithm of the probability distribution in the case of a Gaussian.

In this section, I am going to present the few cost functions used throughout this study and, in a second section, different schemes to optimize it.

The last section will focus more in detail on how the back propagation is integrated in these different schemes.

7.4.1 Mean square error

As mentioned, the cost function is the key element of the optimization. It represents a metric of success of our problem. The most representative cost function, in machine learning, is the mean square error function:

$$\mathcal{L}(\omega) = \frac{1}{2n} \sum_{i=0}^{n-1} (I_i - \tilde{I}_i(\omega))^T (I_i - \tilde{I}_i(\omega)) \quad (1)$$

The mean square error measure the difference between the objective function I_i, *e.g.* an image, and the engine output $\tilde{I}_i$ for the datum *i*. The element *i* is one of *n* elements. The dot product ensures the concavity of the problem, easing the convergence.

Although very simple, the mean square error cost function is a powerful objective function and is certainly the most popular cost function across machine learning fields. However, it does not adapt well to every type of problems. In the following paragraphs, I am going to expose few cost functions useful for the type of problems encountered in this study.

7.4.2 Cross-entropy

Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. The measure is based on the average number of bits needed to identify an event drawn between estimated probability distribution and the true distribution.

The cross entropy is efficient when the number of classes are big and it is a not symmetric function.

The formula is this:

$$\mathcal{L}(\omega) = -\sum_{i=0}^{n}(I_i \log \tilde{I}_i(\omega)) \tag{2}$$

7.4.3 Soft-maximum

Soft-max function takes an N-dimensional vector of real numbers and transforms it into a vector of real number in range (0,1).

Instead of selecting one maximum value, it breaks the whole (1) with maximal element getting the largest portion of the distribution, but other smaller elements getting some of it as well. This property of soft-max function that it outputs a probability distribution makes it suitable for probabilistic interpretation in classification tasks.

Due to the desirable property of soft-max function outputting a probability distribution, we use it as the final layer in neural networks.

For this we need to calculate the derivative or gradient and pass it back to the previous layer during back-propagation.

$$\mathcal{L}(\omega) = \frac{e^{I_i(\omega)}}{\sum_{j=0}^{n} e^{(I_j)}} \tag{3}$$

7.5 Optimization

Optimizing, or minimizing, the cost function $\mathcal{L}$ is an active field of research.

One of the major algorithm for optimization is the *gradient descent* algorithm, also known as *steepest descent*. However, many other algorithms were proposed in the literature.

In this section we are going to describe the major optimization algorithm used in deep learning, starting with the Gradient descent.

7.5.1 Gradient Descent

The gradient descent [37] is a first order minimization scheme.

Let $\mathcal{L}$ be a parametric cost function depending of the set of d parameters $\omega = \{\omega_1, \omega_2, \ldots, \omega_{d-1}\}$, d being the dimensionality of the problem. The total differential of $\mathcal{L}$, can be written as:

$$\mathcal{L} = (d\omega)^T \nabla \mathcal{L} \tag{4}$$

Where the superscript T represents the transpose of the vector $d\omega$. In other words, a small variation of $\mathcal{L}$ can be thought as an infinitesimal motion in each dimension of the tangent space represented by the gradient of the cost function $\nabla\mathcal{L}$.

When the gradient is null, we consider being at one local extremum of the $\mathcal{L}$.

A second order of the cost function expansion would help the analysis of the local curvature of the function, *i.e.* analysis of the Hessien components.

However, the problem would rise up to a *O(d x d)* algorithm, quickly overwhelming the physical capabilities of now days hardware.

Instead, we can favor the variation of the weights, ω, to follow the tangent: $d\omega = -\eta\nabla\mathcal{L}$. In other words: $d\mathcal{L} = -\eta(\nabla\mathcal{L})^T\nabla\mathcal{L}$.

The dot product is a positive value, and if the *learning rate,* η , is chosen positive, then the variation of the cost function is always decreasing.

The gradient descent, as presented, is a very efficient, versatile and widely used algorithm. Between two iterations, *(i)* and *(i - 1)*, the vector of weights is updated the following way:

$$\omega^{(i)} = \omega^{(i-1)} - \eta\nabla\mathcal{L} \tag{5}$$

The choose of the learning rate value is central. If the learning rate is too small, then the convergence is slaw and the chance to converge into a local minimum is high. On the other hand, if the learning rate is too high, the chance to go over the global minimum is high and the algorithm might never converge.

However, the main flaws of the algorithm are the speed and the local extrema. Several methods were developed that overcome these difficulties. For instance, one can implement a greedy strategy to test the algorithm through a grid of learning rate values.

7.5.2 Stochastic Gradient Descend

Another variant, more adequate for neural network, is the *stochastic gradient descend* [38], also known as *on-line strategy*.

Instead of minimizing using the entire set of data, the stochastic gradient descent updates the weights at each datum, decreasing the memory burden caused by large amount and the size of data.

In deep learning, using a complex set of layers, this strategy can "bottleneck" the algorithm. A midground was found using *mini-batch* of data.

In the following paragraphs, I am going to outline the main strategies that have been developed to overcome the gradient descent algorithm limitations.

7.5.3 Momentum

The momentum method [39] was introduced in 1964 by Polyak. The algorithm was proposed to increase the speed of the gradient descent.

$$\mathbf{v}^{(i)} = \mu \mathbf{v}^{(i-1)} - \eta \nabla \mathcal{L} \tag{6}$$

$$\omega^{(i)} = \omega^{(i-1)} + \mathbf{v}^{(i)} \tag{7}$$

The additional velocity term forces the update of the weights, at the iteration (*i*), to stay close to the previous iteration (*i - 1*). The weight μ, between [0,1], regulates the importance of the previous iteration velocity, and the term $\mu v^{(u)}$ resembles to the physical momentum, explaining the name of the method.

A weight of $\mu = 0.9$ is usually used in optimization problem.

On the importance of initialization and momentum in deep learning Momentum is a strategy for make Stochastic Gradient Descend more efficient, in particular for do not be stuck in local minima.

The error on which is compute the back propagation is not calculated on only the last interaction but also on the one before multiplied for a factor beta (usually 0.9). This shrewdness creates a "momentum" that make the function more stable and do not stop in the local minima.

7.5.4 AdaGrad

Adaptive Online Gradient Descent [40] introduces a different approach to update the learning rate.

The equation (8) describes the AdaGrad algorithm.

The element $\sigma_j^{(i)}$ represents the *j-th* element of the *(i)-th* iteration of the vector σ.

The vector σ is the sum of square of all the cost function gradients until the *(i)-th* iteration. The goal of AdaGrad is to monotonically decrease the learning rate using the function σ.

$$\begin{cases} \sigma_j^{(i)} & = & \sigma_j^{(i-1)} + (\nabla_j \mathcal{L})^2 \\ \mathbf{v}^{(i)} & = & -\frac{\eta}{\sqrt{\sigma^{(i)}+\epsilon}} \odot \nabla \mathcal{L} \end{cases} \tag{8}$$

One of the drawbacks of the method comes from the important weight the function σ will apply on the learning rate after cumulating many iterations.

Causing the search of a minimum to stall. The coefficient ε is added to prevent the zero-machine division. The update of the weights is done the following way:

$$\omega^{(i)} = \omega^{(i-1)} + \mathbf{v}^{(i)} \tag{9}$$

7.5.5 AdaDelta

Adadelta [41] is also an adaptive learning rate method which improve AdaGrad algorithm equation (8).

The algorithm is presented equation (10).

The improvement is defined in two aspects and target to lower the aggressivity on the learning rate decrease.

The first part of the improvement reduced the cumulus of the squared gradient of the cost function into a window, ω, representing the ω past iterations.

In this window, the cumulus is replaced by a running average of the squared gradient of the cost function.

The second part of the improvement adds a weight, μ, on the update of the velocity, the same way it was introduced in the momentum method equation (7). As in the momentum method, the weight μ = 0.99.

$$\begin{cases} E[(\nabla\mathcal{L}^2]_w^{(i)} & = & \mu E[(\nabla\mathcal{L})^2]_w^{(i-1)} + (\mu-1)(\nabla\mathcal{L})^2 \\ \mathbf{v}^{(i)} & = & -\frac{\eta}{\sqrt{E[(\nabla\mathcal{L})^2]_w^{(i)}+\epsilon}} \odot \nabla\mathcal{L} \end{cases} \tag{10}$$

The weights are updated the same way as it was done in AdaGrad, as it was done in Adadelta is an adaptive learning rate method introduced in the 2012 and it is based on AdaGrad with a little variation: it use a different function to update the learning rate but it is still monotone decrescent and based on the past error.

Need one hyper-parameter more respect AdaGrad: β. The introduction of Beta prevents learning rate to decay to fast, that was the biggest problem of AdaGrad method.

$$\mathbf{v}^{(i)} = -\frac{\eta}{\sqrt{\sigma^{(i)} + \epsilon}} \nabla\mathcal{L} \tag{11}$$

$$\sigma^{(i)} = \beta\sigma^{(i-1)} + (1-\beta)\nabla\mathcal{L}^2 \tag{12}$$

$$\sigma^{(0)} = 0 \tag{13}$$

$$\omega^{(i)} = \omega^{(i-1)} + \mathbf{v}^{(i)} \tag{14}$$

AdaDelta works in the same way as AdaGrad with the only exception of the introduction of the term β in the function σ. The hyper-parameter β, that usually is 0.999, make the learning rate decay less fast that was the biggest problem with AdaDelta. This introduction made the new optimizer more efficient respect the older one.

7.5.6 Adam

Adam is the newer optimizer introduced in deep learning introduced in 2015.

It uses two methods already existent: AdaDelta and Momentum. The combination of these two methods shows the best result in term of velocity and stability of the convergence.

This optimizer needs three hyper-parameters plus the learning rate to run: α, β and ϵ

$$\mathbf{v}^{(i)} = -\frac{\eta}{\sqrt{\sigma^{(i)} + \epsilon}} M^{(i)} \tag{15}$$

$$M^{(i)} = \alpha M^{(i)} + (1-)\nabla\mathcal{L} \tag{16}$$

$$\sigma^{(i)} = \beta\sigma^{(i-1)} + (1-\beta)\nabla\mathcal{L}^2 \tag{17}$$

$$\sigma^{(0)} = 0 \tag{18}$$

$$\omega^{(i)} = \omega^{(i-1)} + \mathbf{v}^{(i)} \tag{19}$$

Adam use at the same time the Momentum and the AdaDelta method.

The momentum is represented by the function $\alpha M^{(i)}$ and AdaDelta by the function σ.

They work exactly in the same way as the original algorithms by they are used together to update the parameters.

The union of the two algorithms make Adam the most efficient and fast optimizer for machine learning in neural network.

7.6 Back propagation

Back propagation algorithms are a family of methods used to efficiently train artificial neural networks following a gradient descent approach that exploits the chain rule.

The main feature of back propagation is its iterative, recursive and efficient method for calculating the weights updates to improve the network until it is able to perform the task for which it is being trained.

Back propagation uses the chain rule to see how much the cost function derivatives is sensitive to every weights and biases of the network.

$$C_0 = (a^{(L)} - y)^2 \tag{20}$$

$$a^{(L)} = f(w^{(L)} a^{(L-1)} + b^{(L)}) = f(z^{(L)}) \tag{21}$$

$$\frac{dC_0}{dw^{(L)}} = \frac{dz^{(L)}}{dw^{(L)}} \frac{da^{(L)}}{dz^{(L)}} \frac{dC_0}{da^{(L)}} \tag{22}$$

$$\frac{dC_0}{da^{(L)}} = 2(a^{(L)} - y) \tag{23}$$

$$\frac{da^{(L)}}{dz^{(L)}} = f^I(z^{(L)}) \tag{24}$$

$$\frac{dz^{(L)}}{dw^{(L)}} = a^{(L-1)} \tag{25}$$

$$\frac{dC_0}{dw^{(L)}} = a^{(L-1)} f^I(z^{(L)}) 2(a^{(L)} - y) \tag{26}$$

$$\frac{dC_0}{db^{(L)}} = f^I(z^{(L)}) 2(a^{(L)} - y) \tag{27}$$

$$\frac{dC_0}{dw^{(L-1)}} = w^{(L)} f^I(z^{(L)}) 2(a^{(L)} - y) \tag{28}$$

The example is the simple type of network, 4 neurons connected in series. The goal is to decrease in most the efficient way the cost function.

The cost function is the difference between the output of the network and the target. In this case the cost function is calculate MSE method.

One time we calculate the cost function, the parameters of the network have to be optimized to make the cost function decrease.

The back propagation uses the partial derivatives of every weights and biases of the network with respect of the cost function to see how the cost function is sensitive to them all and base on this change it.

To calculate the derived of the first weight and biases the formula are shown before, to calculate the second weight and biases the back propagation uses the chain rule.

As shown in the formula before, with the chain rule is possible to use the derivative before to calculate the chaining with the derived of the weight before [42].

References

[1] Gorno-Tempini M. L., Hillis A. E. and Weintraub S. Classification of primary progressive aphasia and its variants. *Neurology*, 2011.

[2] Ogar J. M., Dronkers N. F., Brambati S. M., Miller B. L. and Gorno-Tempini M. L. Progressive nonfluent aphasia and its characteristic motor speech deficits. *Alzheimer Dis Assoc Disord,* 2007.

[3] Josephs K. A., Duffy J. R., Strand E. A., Whitwell J. L., Layton K. F., Parisi J. E. and Petersen R. C. Clinicopathological and imaging correlates of progressive aphasia and apraxia of speech. *Brain*, 2006.

[4] Mesulam M., Wicklund A., Johnson N., Rogalski E., Leger G. C., Rademaker A. and Bigio E. H. Alzheimer and frontotemporal pathology in subsets of primary progressive aphasia. *Ann Neurol*, 2008.

[5] Spinelli E. G., Mandelli M. L., Miller Z. A., Santos-Santos M. A., Wilson S. M., Agosta F. and Gorno-Tempini, M. L. Typical and atypical pathology in primary progressive aphasia variants. *Ann Neurol*, 2017.

[6] Santos-Santos M. A., Rabinovici G. D., Iaccarino L., Ayakta N., Tammewar G., Lobach I. and Gorno-Tempini, M. L. Rates of Amyloid Imaging Positivity in Patients With Primary Progressive Aphasia. *JAMA Neurol*,2018.

[7] Lahabar S. High Performance Pattern Recognition on GPU. *Academia*, 2008.

[8] R. Nayak and Ting B. K. H. Artificial Neural Networks in Biomedical Engineering: A Review. *Elsevier*, 2001.

[9] Perez L. and Wang J. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *CoRR*, 2017.

[10] Vonk J. M. J., Jonkers R., Hubbard H. I., Gorno-Tempini M. L., Brickman A. M. and Obler L. K. Semantic and lexical features of words dissimilarly affected by non-fluent, logopenic, and semantic primary progressive aphasia. *J Int Neuropsychol Soc*, 2019.

[11] Ranasinghe K. G., Kothare H., Kort N., Hinkley L. B., Beagle A. J., Mizuiri D. and Nagarajan S. S. Neural correlates of abnormal auditory feedback processing during speech production in Alzheimer's disease. *Sci Rep*, 2019.

[12] Battistella G., Henry M., Gesierich B., Wilson S. M., Borghesani V., Shwe W. and Gorno-Tempini M. L. Differential intrinsic functional connectivity changes in semantic variant primary progressive aphasia. *Neuroimage Clin*, 2019.

[13] Payan A. Predicting Alzheimer's disease: a neuroimaging study with 3D convolutional neural networks. *ArXiv*, 2015.

[14] Wen J. Convolutional Neural Networks for Classification of Alzheimer's Disease: Overview and Reproducible Evaluation. *ArXiv*, 2018.

[15] Masoumi H. Automatic liver segmentation in MRI images using an iterative watershed algorithm and artificial neural network. *ArXiv*, 2009.

[16] Ibrahim W. H. MRI brain image classification using neural networks. *ArXiv*, 2013.

[17] Chen H. Low-Dose CT with a Residual Encoder-Decoder Convolutional Neural Network (RED-CNN). *ArXiv*, 2017.

[18] Krizhevsky A. ImageNet Classification with Deep Convolutional Neural Networks. *ArXiv*, 2012.

[19] Bahrampour S. Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning. *Open-Review*, 2016.

[20] Jia Y. Caffe: Convolutional Architecture for Fast Feature Embedding. *Acm*, 2014.

[21] Yu L. Automatic 3D Cardiovascular MR Segmentation with Densely-Connected Volumetric Con-vNets. *Springer International Publishing*, 2018.

[22] Hwang H. 3D U-Net for Skull Stripping in Brain MRI. *Applied Sciences*, 2018.

[23] Abdulkadir A. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. *MICCAI*, 2016.

[24] LeCun Y. Convolutional Neural Networks Applied to House Numbers Digit Classification. *ArXiv*, 2012.

[25] Bui T. D. 3D Densely Convolutional Networks for Volumetric Segmentation. *ArXiv*, 2017.

[26] Cicek O. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. *ArXiv*, 2016.

[27] Noh H. Learning Deconvolution Network for Semantic Segmentation. ArXiv, 2015.

[28] Ioffe S. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv*, 2015.

[29] Refaeilzadeh P. Cross-Validation. *Springer*, 2008.

[30] Manjon J. V. Adaptive non-local means denoising of MR images with spatially varying noise levels. *PubMed*, 2010.

[31] Kingma D. P. Adam: A Method for Stochastic Optimization. *ArXiv*, 2014.

[32] Gondara L. Medical Image Denoising Using Convolutional Denoising Autoencoders. *ArXiv*, 2016.

[33] Giacinto G. Design of effective neural network ensembles for image classification purposes. *Elsevier*, 2001.

[34] Rosesenblatt F. The Perceptron: a probabilistic model for information storage and organization in the brain. *Cornell Aeronautical Laboratory*, 1958.

[35] Jain A. K. Artificial neural networks: a tutorial. *IEEE*, 1996.

[36] Jones S. A. Analysis of different activation function using back propagation neural network. *Journal of Theoretical and Applied Information Technology*, 2013.

[37] Hochreiter S. Learning to Learn Using Gradient Descent. *Canadian Institute for Advanced Research*, 2001.

[38] Amari, S. Backpropagation and stochastic gradient descent method. *Elsevier*, 1993.

[39] Castillo E. A Very Fast Learning Method for Neural Networks Based on Sensitivity Analysis. *Journal of Machine Learning Research*, 2006.

[40] Duchi J. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, (2011).

[41] Zeiler M. D. AdaDelta: An Adaptive Learning Rate Method. *ArXiv*, 2012.

[42] le Cun Y. A Theoretical Framework for Back-Propagation. *CMU*, 1988.