UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in Biomedical Engineering

EARLY DIAGNOSIS OF ALZHEIMER'S DISEASE: EFFECTS OF REGISTRATION, BRAIN EXTRACTION AND AUGMENTATION OF 3D MRI SCANS COMBINED WITH NEURAL NETWORK PARAMETER TUNING

Relatore:

Prof. Laura Burattini

Tesi di Laurea di:

Giacomo Covella

Correlatore:

Prof. Aldo Franco Dragoni

Dott. Agnese Sbrollini

Anno Accademico 2020/2021

Acknowledgments

First, I am extremely grateful to my supervisor, Selene Tomassini for her invaluable advice, continuous support, and professionalism during my thesis study.

I would also like to express my gratitude to my friends and fellow colleagues, Giacomo, Federico and Alessio for sharing this long journey with me.

Thanks to my life-long friends, Vittorio, Matteo and Niccolò for being the brothers I have never had.

Lastly, my family deserves endless gratitude: my father for always believing in everything I do, my mother for teaching me to be a strong person, and my grandparents for always being proud of my choices. To my family, I give everything, including this.

Abstract

Alzheimer’s Disease (AD), considered as the most widespread form of dementia, is a neurological condition that results in a progressive brain atrophy and mental deterioration. Even though an effective treatment for AD still not exists, an early detection of this pathology may help in slowing down the disease progression. Among all available neuropsychological data, T1-weighted structural magnetic resonance images, which stress the difference between white and grey matter, appear to be the best choices to recognize the disease. In recent years, an increased interest in AD detection from magnetic resonance images using artificial intelligence algorithms have emerged. Specifically, many deep learning approaches, which integrates the feature step in the learning one, have been proposed, most of them involving 3D convolutional neural networks. This work is intended to investigate the effects of registration, brain extraction and data augmentation of magnetic resonance scans, together with the parameter tuning of a Convolutional Long-Short Term Memory (ConvLSTM)-based neural architecture, on the model performance for AD classification. To this aim, 275 scans were selected from the OASIS-3 dataset (145 AD and 130 cognitively-healthy patients). Pre-processing consisted in linear affine registration with three different templates, brain extraction and data augmentation (horizontal flipping and rotation). Nine experiments were conducted: each experiment, expect for the first, started from the previous experiment and added something new. First, the performance of the proposed framework was quantified on the raw data. Then, the impact of the registration step using three different templates was evaluated, leading to the identification of the best one. Brain extraction, data augmentation and, finally, parameter tuning were gradually added till the last experiment. Results show how the model performance tend to increase step by step until the parameter tuning, which lead to an accuracy of 85%, a sensitivity of 88% and an area under the curve of 92%. In conclusion, the work hereby presented demonstrates the crucial importance of pre-processing steps such as registration and brain extraction, data augmentation and parameter tuning in enhancing the AD classification performance of the ConvLSTM-based framework.

Summary

Introduction

Alzheimer’s disease, the most common form of dementia, is an incurable neurological condition that results in a progressive brain atrophy and mental deterioration. The diagnosis of Alzheimer’s disease, which is still very difficult in clinical practice, is largely based on clinical history and neuropsychological data including magnetic resonance images. In recent year, an increased interest in deep learning-based approaches for Alzheimer’s disease detection have emerged.

This work is intended to investigate the effects of registration, brain extraction and data augmentation of magnetic resonance scans, together with the parameter tuning of a convolutional long-short term memory-based neural architecture, on the model performance for Alzheimer’s disease classification.

In the first chapter, a general overview of the anatomy and physiology of the human brain is provided.

In the second chapter, an accurate description of the Alzheimer’s disease and the possible causes that can lead to the disease are reported.

In the third chapter, magnetic resonance imaging is introduced with a deeper focus on structural T1-weighted magnetic resonance images.

In the fourth chapter, after a brief comparison between machine learning and deep learning in Alzheimer’s disease detection, an overview about typical deep learning architecture for Alzheimer’s disease classification is reported. Follows a state of art review of 3D convolutional neural network for Alzheimer’s disease classification.

The fifth chapter constitutes the methodological part of this work. The description of the main pre-processing tools adopted, the data augmentation techniques involved, and the structure of the proposed neural network are provided along with the list of experiments conducted.

In the sixth chapter, after a brief explanation of the metrics involved in the quantification of the neural network performance, the results of each experiment are reported.

Finally, in the seventh chapter, the results are discussed, and conclusions are given.

1. Human brain anatomy and physiology

The nervous system is commonly divided into the Central Nervous System (CNS) and the Peripheral Nervous System (PNS) [1]. CNS is composed by the brain and the spinal cord, while the PNS is made up of nerves and ganglia that connect the spinal cord with the rest of the body. The PNS can be further divided in somatic nervous system, associated with the voluntary control of body movements via skeletal muscles, and autonomic nervous system, which exert involuntary control over smooth muscle and gland.

For what concern this work, we focus specifically on the brain, providing a broad overview of it from the histological and structural point of view [1-2].

The brain is involved in the control of body movements, thoughts, memory, and speech and in the regulation of many organs within the body. Moreover, through the five senses of sight, smell, hearing, touch, and taste, it receives information about the world surrounding us.

Two types of cells constitute the brain: neurons and glia cells [2]. Neurons are electrically excitable cells, which can communicate between them thanks to specific structures called synapses. A typical neuron is composed by three main parts: the cell body, also called soma, the axon, a slender projection of the neuron responsible of delivering action potentials away from the cell nerve body, and finally dendrites, branched protoplasmic extensions of the neurons which gather synaptic inputs coming from different nerve cells. On the other hand, glia cells are non-neuronal cells which maintain homeostasis in the nervous system, also providing protection and support to the neurons.

In the brain we can distinguish between grey and white matter (Fig. 1) [2]. The white matter, which forms the bulk of the deep parts of the brain, is composed by neuronal axons and oligodendrocytes; the latter are a type of glia cell that speed up the axonal conduction wrapping the axons with myelin sheets. Contrarily, the grey matter, distributed instead on the surface of the brain, consists of neuronal cell bodies, synapses, glia cells and capillaries.

Protection for the human brain comes from the skull, meninges, and cerebrospinal fluids [2]. The latter is a clear, watery substance that helps to cushion the brain and spinal cord from injury. Indeed, the nervous tissue is extremely delicate and can suffer damage by the smallest amount of force.

Finally, the so-called blood-brain barrier prevents the interaction between the brain and any potentially harmful substance floating in the blood.

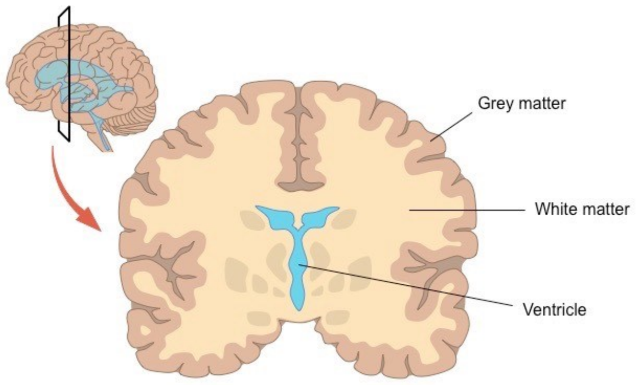


Figure 1 - White and grey matter [3].

From a structural point of view (Fig. 2), the human brain can be differentiated in five main parts: the brainstem, the diencephalon, the cerebellum, the cerebrum and finally the ventricular system [2].

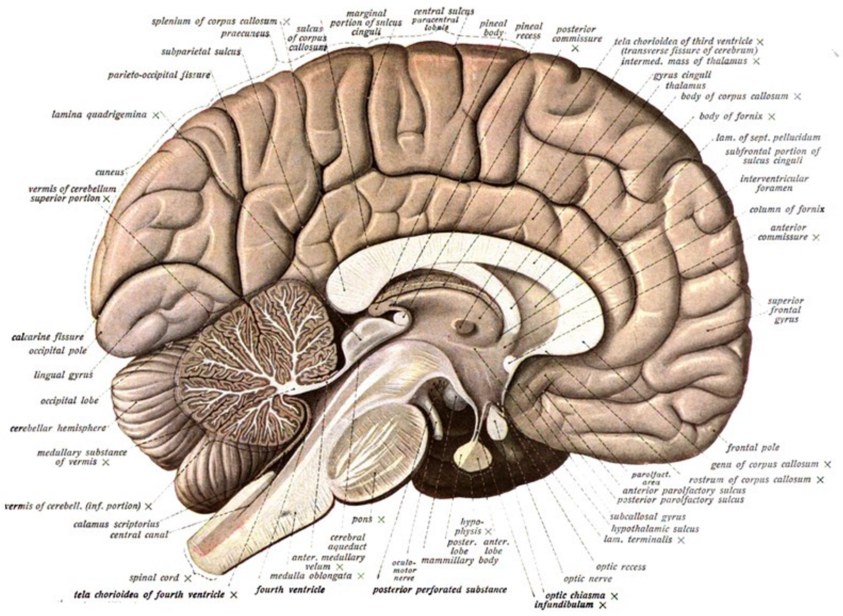


Figure 2 - Brain structure [4].

The brainstem is the lower extension of the brain, located in front of the cerebellum and connected to the spinal cord. It consists of three structures: the midbrain, pons, and medulla oblongata. It serves as a relay station, passing messages back and forth between various parts of the body and the cerebral cortex. Many simple or primitive functions that are essential for survival are located here.

The diencephalon includes two structures, namely the thalamus and the hypothalamus. The thalamus can be considered as a relay center of the brain. It receives afferent impulses from sensory receptors and distributes the processed information to the appropriate cortical area. It is also responsible for regulating consciousness and sleep. The hypothalamus is a small structure that handles information coming from the autonomic nervous system. It plays a role in controlling functions such as eating, sexual behavior and sleeping and it regulates body temperature, emotions, secretion of hormones and movement.

The cerebellum, composed by two specular cerebellar hemispheres, lies behind the pons. It has a major role in tuning motor activity and movements, helping to maintain posture, sense of balance or equilibrium. The cerebellum may also be involved in the regulation of some cognitive function, from language to emotional response, but its movement-related functions are the most solidly established.

The cerebrum, the uppermost part of the CNS, is the largest part of the brain: it is composed of two structurally identical cerebral hemispheres, each of them containing the cerebral cortex on the surface, as well as many subcortical structures like the hippocampus, basal ganglia and olfactory bulbs. The cerebral cortex, a folded structure characterized by the presence of ridges (gyri) surrounded by one or more furrows (sulci), is generally classified into five lobes. The frontal lobe, placed at the front of each hemisphere, is important for cognitive function and has a role in the control of voluntary movements and activity; the parietal lobe, immediately behind the frontal one, processes information about temperature, taste, touch and movements; the occipital lobe is the visual processing center and it's placed in the back part of the brain; the temporal lobe, placed laterally, contains regions dedicated to processing sensory information, particularly important for hearing, recognizing language, and forming memories; finally, the limbic lobe is an arc-shaped region of cortex on the medial surface of each cerebral hemisphere considered to be the epicenter of emotional and behavioral expression (the hippocampus, a region of particular interest for this work, it's included by many authors in the limbic lobe).

The ventricular system is a set of communicating cavities within the brain. These structures are responsible for the production, transport, and removal of cerebrospinal fluid, which bathes the central nervous system. The left and right lateral ventricles (the first and the second) are located within their respective hemispheres of the cerebrum. They both communicate, thanks to a small opening, with the third ventricle through the so-called Foramen of Munro. The third ventricle, located in the center of the brain close to the thalamus and hypothalamus, connects with the fourth ventricle through a long tube called the Aqueduct of Sylvius.

The CNS may be affected by many different neurodegenerative disorders, in which a progressive loss of structure or function of neurons hampers the normal functioning of the brain. This work is focused on a particular form of dementia that goes under the name of Alzheimer’s disease.

2. Alzheimer’s disease

Around the world, around fifty million people suffer from dementia [5]. Dementia is a chronic or persistent disorder of the mental processes that may be caused by brain disease or injury. People with this syndrome show a deterioration in memory, thinking, behavior and in the ability to perform everyday activities.

Alzheimer’s Disease (AD) is deemed as the most common form of dementia, contributing to 60-70% of the cases as reported by the World Health Organization [5]. This pathology has a non-negligible psychological, physical, social, and economic impact, not only on patients, but also on the so-called caregiver and on the society at large. From a clinical perspective, AD manifests in most cases as an insidiously progressive decline in functional and cognitive status, with major influences on memory and intellectual functions [6]. Pathologically, AD is mostly characterized by an excessive density of amyloid β-protein plaques and neurofibrillary tangles, as demonstrate by postmortem brain microscopy [6].

Even though the real causes leading to AD development are not still clear, many hypotheses have been formulated in past years. According to the cholinergic hypothesis [7], AD may be caused by a reduction in the acetylcholine production, but this idea has lost widespread support mainly due to the ineffectiveness of drugs intended to treat acetylcholine deficiency. On the other side, many studies give credit to amyloid beta (Aβ) deposits [8-9] and tau protein [10] abnormalities as the fundamental causes of the disease. Finally, a small number of patients (probably fewer than 1%) have early-onset AD because they have inherited autosomal dominant genetic mutations [11].

In AD, as neurons are injured and die throughout the brain, connections between networks of neurons may break down, and many brain regions begin to shrink (Fig 3). By the final stages of AD, this process, called global brain atrophy, is widespread, causing significant loss of brain volume. Some studies asserted that AD atrophy targets the anterior hippocampal regions and the temporo-parietal cortical areas, even relatively early in the clinical expression of the dementia syndrome [12].

Ageing is considered as the most important risk factor for AD [12]. Young people show better energy metabolism, higher levels of growth factors and more efficient mechanisms for clearing misfolded proteins and repairing cells. Lack of these protective mechanisms may contribute to the development of AD [13]. Moreover, studies have tried to find the possible link between age and AD demonstrating that, even though they exert independent gray matter atrophy patterns, there is a substantial overlapped in the hippocampus and entorhinal cortex [14].

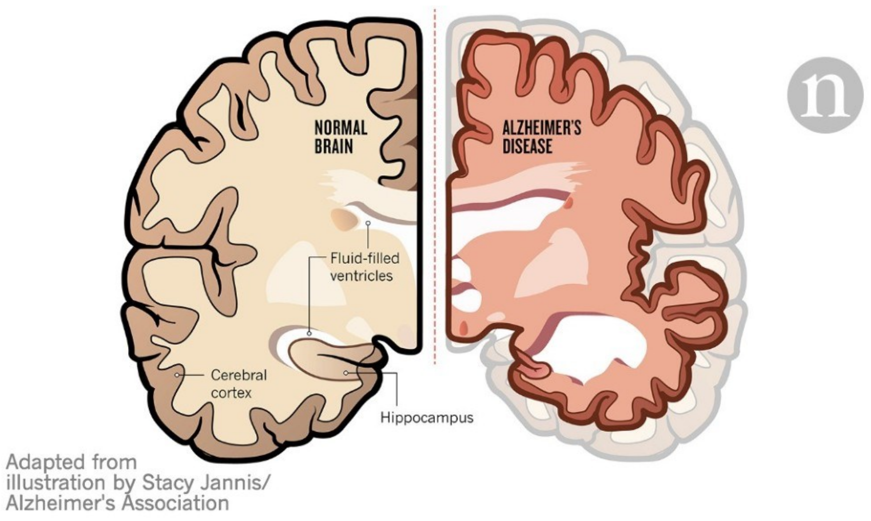


Figure 3 - Standard brain (left) vs. brain with Alzheimer disease (right) [15].

Effective treatments to reverse, arrest, or prevent AD have not been identified yet. Nevertheless, given the disproportionate aging of the population, the impact of AD on society and economy will continue to rise [16]. For this reason, the early detection of AD development is still a hot topic in the scientific community. Indeed, an immediate therapeutic intervention during the initial stages of the syndrome appears to have a positive impact on the disease progression [17].

During the last decades, interest in neuroimaging for the study and diagnostic support of neurodegenerative diseases has been rapidly rising. Magnetic resonance imaging (MRI) is widely utilized in hospitals for AD identification because of its extraordinary resolution, good contrast, and high availability [18].

3. Magnetic resonance imaging

MRI is a medical imaging process, based on the phenomenon of Nuclear Magnetic Resonance (NMR), which is nowadays widespread accepted for many medical explorations. It uses magnetic fields and Radio Frequency (RF) signals to obtain anatomical information about the human body as cross-sectional images in any desired direction [19]. In radiology, MRI is commonly used to visualize body soft tissue with great contrast, including of course the brain, and this permit to easily discriminate between healthy and diseased tissue [19-20].

MRI counts many positive aspects:

- It is a noninvasive imaging technique;
- It is harmless thanks to the use of RF wave, which are non-ionizing radiation;
- It has an optimal spatial resolution, and it provides anatomical, functional, and physiological data about tissues;
- It is a very flexible imaging tool thanks to a variety of pulse sequences that can easily be tuned to offer specific visualizations.

On the other side:

- MRI machines are quite expensive;
- It usually requires long scan times;
- It is not comfortable for patient with claustrophobia;
- Patients with non-MRI-compatible implants cannot receive a scan.

Let spend few words on the working principle behind the MRI technology.

The patient is inserted inside the MRI machine, in which a strong and homogenous magnetic field is generated. An equilibrium state is immediately reached: indeed, the magnetic moments of patient's body atomic nuclei become aligned with the external magnetic field. What follows take the name of excitation: the RF coils, electromagnetic coils able to generate and receive electromagnetic waves, perturb the nuclei equilibrium state, by sending electromagnetic waves that resonate at a particular frequency know as Larmor frequency. At this point, patient's nuclei absorb the RF waves and, when the pulse ends, they return to the equilibrium condition, releasing the previously stored energy. This energy, which represent the raw MRI signal, can be detected with the same RF coil in a process called reception. Finally, complex algorithms permit to compose an MRI image from the recorded signals.

There are several different types of MRI procedures, but for what concern this study, we are going to describe structural MRI (sMRI) only.

The sMRI has a high spatial resolution and provides mostly information on the anatomical structure [20]. Indeed, it makes use of the small differences between voxels (the unitary element of the recorded 3D volume) to differentiate neighboring types of tissue.

The type, number, relaxation, and resonance properties of atomic nuclei within a voxel deeply influence the MR signal: we define as contrast, the signal difference between any two types of tissue. To emphasize one specific property of the tissues of interest, we "weight" the image, choosing between a vast repertory of pulse sequences i.e., programs that tells the scanner what to turn on and turn off and when. So, tuning the relevant time parameters of the sequence, we may obtain Proton Density (PD), T1 or T2 images (Fig. 4) [20].

Proton-density images, as the name implies, provide contrast based on the sheer number of protons in a voxel, which of course differs in different tissue types.

T1-contrast images are useful to analyze the structure of the brain from the anatomical point of view, reliably differentiating between grey matter and white matter. Moreover, strong T1 contrast is present between fluid and more solid anatomical structures, making T1 images suitable for morphological assessment of the normal or pathological anatomy e.g., AD.

T2-contrast images have maximal signal in fluid-filled regions, which is important for many clinical applications. Many tumors, arteriovenous malformations, and other pathological conditions show up most readily under T2 contrast.

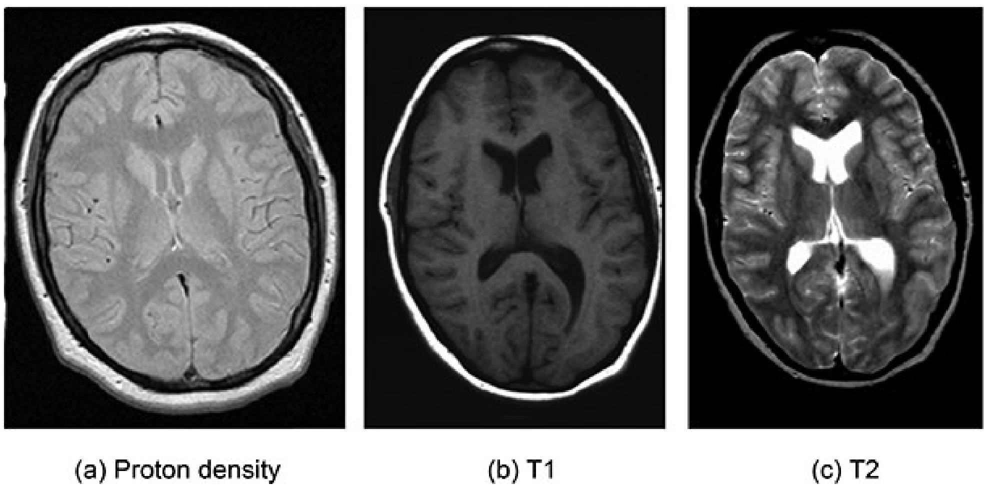


Figure 4 - Proton density (a), T1-weighted (b) and T2-weighted (c) axial scans [21].

The current work is focused on T1-weighted brain volumes. In fact, structural imaging based on magnetic resonance is an integral part of the clinical assessment of patients with suspected AD [22]. The presence of brain and hippocampal atrophy is a sensitive index of neurodegeneration, and it can be easily appreciated from a T1-contrast scan.

4. Alzheimer’s disease classification with deep learning

4.1 Machine learning vs. deep learning

In past years, medical imaging has been shown to be of great importance for the early detection, diagnosis, and treatment of diseases [23]. In clinical practice, the interpretation of medical images has been always performed by human experts such as radiologists and physicians. Nevertheless, nowadays, researchers and doctors have started to benefit from computer-assisted interventions: large variations in pathology and potential fatigue of human experts underline the importance of novel computer-based technologies to support diagnosis [23].

For what concern AD, a definitive diagnosis of dementia is still something very difficult to achieve [24]. Machine Learning (ML) techniques, mainly Support Vector Machine (SVM) classifiers and Artificial Neural Network (ANN), and Deep Learning (DL) techniques have been found to be very useful for the diagnosis of AD from MRI scans [25].

Classification studies using ML-based approaches usually require four fundamental steps: feature extraction, feature selection, dimensionality reduction, and feature-based classification algorithm selection [26]. These procedures normally require very precise knowledges and several stages of optimization, which is expensive from temporal point of view [26]. Moreover, meaningful, or task-related features were mostly designed by human experts based on their knowledge, which in turn make the use of ML techniques challenging for nonexperts [23]. Many studies and competitions have also demonstrated that ML approaches are not suitable for investigating such complex pathologies as AD: successful classification require a strong ability to discriminate specific feature among similar brain image patterns [27].

To overcome these difficulties, DL, which uses raw neuroimaging data to generate features through “on-the-fly” learning, is attracting considerable attention [26]. DL integrates the feature step in the learning one, requires minor pre-processing and discover informative feature in a self-taught manner [25]. The recent success of DL is mostly due to three reasons: advancements of high-tech Central Processing Units (CPUs) and Graphics Processing Units (GPUs), the availability of a huge amount of data and finally the continuous developments of learning algorithms [23]. For these reasons, many researchers are currently focusing on novel DL algorithms to improve the classification accuracy for AD [25]. Following this tendency, we also decided to focus this work on a DL-based approach.

An overview of the state of art of AD classification with DL is now provided.

4.2 Typical deep learning architecture for Alzheimer's disease classification

Among all the possible existing DL architectures, Convolutional Neural Networks (CNNs) have been proposed to assist diagnosis of AD in many recent studies [28]. In CNN, an image is usually taken as input and learnable weights with biases are assigned to different aspects in the image subsequently differentiating one picture from the other [29].

Based on the type and dimensionality of input of the network, we can distinguish between [24]:

- 2D slice-level CNNs;
- 3D patch-level CNNs;
- Region Of Interest (ROI)-based CNNs;
- 3D subject-level CNNs.

In 2D CNN, 2D slices extracted from the 3D MRI volumes are given as input to the network. Well-established 2D CNNs, optimal for natural images classification, already exist and many AD classification studies borrowed them, adapting their structure [28]. Even tough, working with a 3D volume provides huge 2D slices training dataset, the main limitation of 2D CNN is that convolutional filters analyze each slice separately, loosing completely the spatial information of the 3D volume [29].

To compensate the lack of 3D information, 3D patch-level CNNs provide, as input to the network, a series of 3D patches directly extracted from the 3D brain volumes [28]. The main drawbacks of this approach are the overall complexity of the architecture and the variable nature of the patch definition step [28].

The patch definition problem is partially overcome in ROI-based CNN studies, in which patches were extracted from portion of the brain which are known to be informative (especially the hippocampus) [28].

Recently, also thanks to the boost in computer performance, many researchers started to follow a 3D subject level-oriented approach [28]. In these 3D CNNs, the whole 3D brain is taken as input to the network, fully integrating the spatial content of the volume. Despite the reduction of the size of the datasets (each brain constitutes just one dataset sample) leading to a higher risk of overfitting, 3D CNNs and its derivatives are currently highly investigated architecture for the AD diagnosis [28].

Focusing on the latter category, I now report some recent studies in chronological order.

4.3 3D convolutional neural networks for Alzheimer's disease classification – State of art

In all the studies I am going to report, at least one patient-level classification has been conducted: the whole MRI brains, not a segmented portion, are given as inputs to the 3D CNN.

In 2017, Lou et al. [24] proposed a 3D CNN consisting of 3 convolutional layers, each followed by normalization and spatial max-pooling, a fully connected layer, and a classification layer. Alzheimer's Disease Neuroimaging Initiative (ADNI) database was used for the training, validation and test phases, leading to high AD recognition accuracy with a sensitivity of 100 % and a specificity of 93%. Brain registration was accomplished through a series of pre-processing steps, including Gradwarp, B1 non-uniformity, N3, and scale. Moreover, random zooming and in and out cropping were used to augment the dataset.

In the same year, Korolev et al. [30] proposed another 3D-CNN architecture, emphasizing to achieve better performance without incorporating feature extraction steps. In the study, a common feedforward network with convolutional and pooling layers named VoxCNN and a residual NN called ResNet, were compared. The first one, trained, validated, and tested on a subset of ADNI structural MRI data, that has been pre-processed with alignment and skull-stripping marked as "Spatially Normalized, Masked and N3 corrected T1 images", led to an accuracy of 79% and an Area Under the receiver operating characteristic Curve (AUC) of 88% in discriminating control patients from AD patients.

Three other studies have been conducted in 2018.

Ullah et al. [31] came up with a 3D CNN model to detect AD and other forms of dementia from 3D magnetic resonance image. The neural network structure consists of total 6 layers: two couple of convolutional and pooling layers, followed by a fully connected layer and a classification layer. The experiments were conducted on Open Access Series of Imaging Studies (OASIS) database, leading to an accuracy of 80.25%.

Bäckström et al. [32] proposed a 3D CNN for automatic learning of features and detecting AD on a pre-processed large size MRI dataset. The 3D CNN, consisting of five convolutional layers, reached an accuracy of 98.73% in AD classification. The experiments were conducted on ADNI database: "cortical reconstruction", including motion correction and conform, non-uniform intensity normalization, Talairach transform computation, intensity normalization and skull and neck removal, were performed by the dataset provider. This study has contributed to find the impact of hyper-parameter assortment on the performance of the proposed AD classifier.

Basaia et al. [33] presented an "all convolutional" 3D CNN-based model constituted by 12 repeated blocks of convolutional layers, an activation layer, a fully connected layer, and classification layer. High levels of accuracy were achieved in classification tests using both ADNI database only (99%) and combined ADNI + private dataset (98%). Images from both datasets were normalized to the MNI space using Statistical Parametric Mapping (SPM) and the Diffeomorphic Anatomical Registration Exponentiated Lie Algebra (DARTEL) registration method. Data augmentation strategy consisted of deformation, flipping, scaling, cropping and rotation of images and it was applied to both training and validation set.

More recently, in 2020, Xia et al. [34] proposed a novel unified CNN framework for AD classification, where both 3D CNN and 3D Convolutional Long Short-Term Memory (3D CLSTM) are employed. The best model, trained and tested on ADNI database, achieved an accuracy of 94.19%, sensitivity of 93.75%, specificity of 94.57% and AUC of 96%. All MRI volume were resized, and horizontal flipping was used as unique data augmentation strategy on training set.

5. Methodology

5.1 Pre-processing tools

Nowadays, the amount of heterogeneous biomedical data is increasing more and more thanks to novel technologies and sensing techniques [35]. For what concern biomedical images analysis, this heterogeneity creates a new challenge. Image pre-processing techniques are mandatory for any image-based applications [35]. Although, most of the time the significance of these techniques is underestimated, pre-processing procedures are essential to ensure the success of the subsequent steps [35].

In the context of MRI classification, a suitable pre-processing pipeline should be followed to increase the accuracy of the NNs [35]. Among the several pre-processing techniques, in this work we followed three independent pre-processing steps: brain registration, brain extraction and data normalization.

Ideally, when brains are compared across individuals, positions, and sizes in one brain must correspond to positions and sizes in another brain. In practice, brains are so variable in shape that there simply may not exist a point-to-point correspondence across any two brains, or even in the same brain over time [36]. Brain registration consists of adapting a brain to another reference image, which is usually called atlas or template, seeking that the same regions of both represent the same anatomical structures [36]. Thanks to brain registration it is simpler for a NN to identify a certain region of the images as relevant since the same information would be represented in all of them. Nowadays, there is plenty of image registration algorithms, both for linear and non-linear registration [36]. Among the several registration algorithms, FMRIB's Linear Image Registration Tool (FLIRT), which performs linear registration, is the one chosen for this work.

Brain extraction is considered as one of the critical pre-processing steps that helps in precise diagnosis of brain diseases [37]. The idea behind brain extraction is that it removes all non-brain tissue-like dura matter, eyes, fats, muscles, exterior blood vessels, and skull and it leaves only the brain region [37]. The removal of these non-brain tissues reduces the computational burden and increases the accuracy of various neuroimaging classification algorithms [37]. Among the several brain extraction procedures and algorithms, Brain Extraction Tool (BET) of the FMRIB Software Library is the one chosen for this work.

Finally, data normalization can be subdivided in voxel intensity normalization and spatial normalization. The former consists in adapting the pixel-values range according to a certain criterion: in this work, whitening, which consist in subtracting the mean from each voxel intensity and then dividing by the standard deviation, is performed [38]. The latter consists of adapting the voxel dimension to the desired one [38]. In this work, there was no need of spatial normalization because the employed database consisted of structural MRI volumes with voxel of dimension 1mm^3 (i.e., a suitable spatial resolution for AD classification).

5.1.1 The FMRIB Software Library

The FMRIB Software Library, abbreviated FSL, is a software library containing image analysis and statistical tools for functional, structural and diffusion MRI brain imaging data [39]. New versions of FSL are yearly released by the University of Oxford, granting the presence of state of art tools.

Figure 5 - FMRIB Software Library (FSL) logo [39].

For what concern the pre-processing pipeline of this work, FSL FLIRT tool was exploited for linear brain registration, while FSL BET tool was exploited for brain extraction.

5.1.1.1 Algorithm for linear affine registration

FLIRT is a fully automated robust and accurate tool for linear (affine) intra- and inter-modal brain image registration [40]. The FLIRT algorithm translates, rotates, zooms and shears one image to match it with another, exploiting a global optimization method that is specifically tailored to brain registration [41].

It should be mentioned that FSL also provides an interesting tool for nonlinear brain registration called FMRIB's Nonlinear Image Registration Tool (FNIRT). This tool, exploiting local deformation, is particularly suitable to account for local changes due to brain atrophy and, for this reason, it is recommended for the registration of disease brain [42], like in AD case. Nevertheless, FNIRT requires too much computational burden, and this explain the usage of FLIRT in this work.

FSL FLIRT tool allows to register brains both to a default template and a generated template. The latter may be obtained as an average of a portion or the totality of the dataset of interest.

What follows is the Python code involved in linear brain registration.

```
fslpython
import os
from fsl.wrappers import flirt
```

Once FSL is opened, os library and FLIRT are imported.

```
for input in os.listdir(path):
    output = os.path.join(path,input.replace('.nii.gz',''))+'_LREG.nii.gz'
    flirt(ref=ref, src=os.path.join(path,input), out=output)
```

In this portion of code, the registered brains are generated. The variable '*path*' contains the path of the directory where all the sMRI brains are stored in '.nii.gz' format. The variable *'ref'* contains the path of the template that is going to be used for the registration. In the for loop, thanks to the command *'os.listidir(path)'*, the path of each '.nii.gz' file is extracted and used to compose the output file path *('output')*. The *'flirt'* command takes the template path (*'ref'*) and the single MRI file path (*'src'*) as inputs and saves the registered MRI file in the provided output path (*'out'*).

5.1.1.2 Brain extraction tool algorithm

BET (Brain Extraction Tool) deletes non-brain tissue from an MRI image of the whole head [43]. If good quality T1 input images are provided, it can also estimate the inner and outer skull surfaces, and outer scalp surface [43]. BET uses a deformable model that evolves to fit the brain's surface by the application of a set of locally adaptive model forces [44].

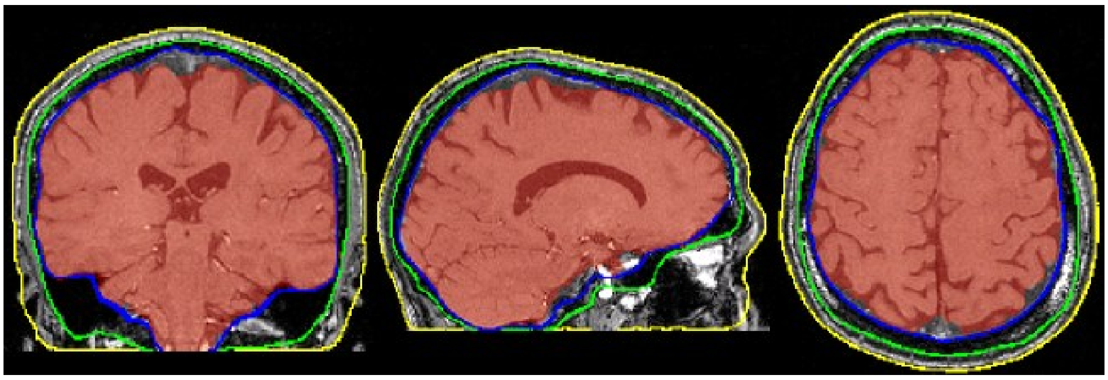


Figure 6 – Brain selection exerted by Brain Extraction Tool (BET) in coronal, sagittal and axial sMRI scans (from the left to right). The portion highlighted in red and enclosed in the blue line is the brain, the green and yellow lines highlight the inner and outer skull surface respectively [43].

BET allows to tune the so-called fractional intensity threshold: smaller value gives larger brain outline estimated, while larger value perform a stricter brain selection. In this work a threshold of 0.3 has been adopted. Moreover, the 'robust' option was activated: a more "robust" brain centre estimation is involved, resulting in a better final estimate especially when the input data contains a lot of non-brain matter (like for example neck portions).

What follows is the Python code involved in brain extraction.

```
fslpython
import os
from fsl.wrappers import bet
```

Once FSL is opened, os library and BET are imported.

```
for input in os.listdir(path):
    output = os.path.join(path,input.replace('.nii.gz',''))+'_brain.nii.gz'
    bet(os.path.join(path,input),output, robust=True,fracintensity=0.3)
```

In this portion of code, the brains are extracted. The variable '*path*' contains the path of the directory where all the sMRI volumes, previously registered, are stored in '.nii.gz' format. In the for loop, thanks to the command *'os.listidir(path)'*, the path of each '.nii.gz' file is extracted and used to compose the output file path *('output')*. The *'bet'* command takes the single MRI file path (*'os.path.join(path,input)'*) as input and saves the extracted MRI brain file in the provided output path (*'output'*). Inside the *'bet'* command, the 'robust' option is activated through *'robust=True'* and the fractional intensity threshold is set to 0.3 through '*fracintensity=0.3'*.

5.2 Data augmentation

Deep CNN are heavily reliant on big data to avoid overfitting [45]. Overfitting refers to the situation when a NN perfectly models the training data, partially losing its generalization capabilities [45]. Unfortunately, many application domains do not have access to big data, such as medical image analysis [45].

In the specific case of this study, overfitting problem is even more prominent, given the choice of patient level-oriented classification. Each MRI volume accounts for a single sample, thus leading to limited size training set. Data augmentation, a data-space solution to the problem of limited data, constitute one of the most powerful approaches to prevent overfitting [45].

Data augmentation in DL is a technique used to increase the amount of available data by adding slightly modified copies of already existing data or newly created synthetic data from existing data [45]. Other than decreasing overfitting, data augmentation also increases the performance of the model [46].

In the current work, affine volumes transformation has been used to increase the dimensionality of training set. With the affine approach, the 3D MRI volumes undergo different operations like rotation, zooming, cropping, flipping, or translations [47]. Horizontal flip and rotation around the brain dorsoventral axis are the two affine transformations specifically chosen for the augmentation of the training set of this study.

It is worth mentioning that many AD classification studies suffer from data leakage because of wrong or late dataset split [28]. Procedures such as data augmentation must never use the test set and thus be performed after the training/validation/test split to avoid biasing the results [28]. What may happen is that, if data augmentation is performed before the training/validation split for example, then images generated from the same original image may be found in both sets, leading to inaccurate and biased performances.

An overview of the two affine transformations exploited in this study is now provided.

5.2.1 Horizontal flipping

Flipped volumes consisted of a mirror reflection of the original volume along one selected axis [47]. Given the morphology of the brain, which is constituted by two anatomically symmetrical hemispheres, horizontal flip appears to be the smartest choice. Indeed, in the case of vertical flip, up and down portions of the brain are not 'interchangeable', while, flipping along the horizontal axis swaps the left hemisphere with the right one, and vice versa. Among the possible augmentation techniques for MRI brain volumes, horizontal flip appears to be perfect for AD classification, given that AD atrophy affects each hemisphere equally [14].

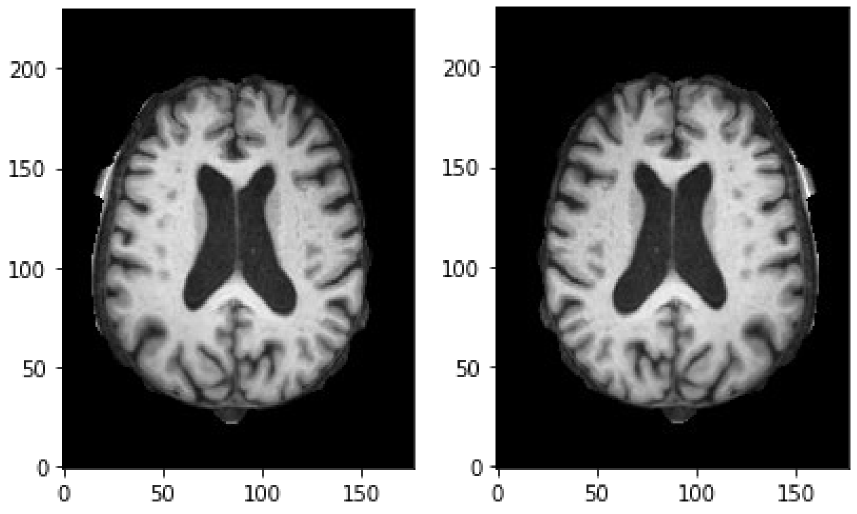


Figure 7 - One axial slice of an MRI volume after FLIRT and BET (on the left) and the same slice after the 3D horizontal flipping of the MRI volume (on the right).

5.2.2 Rotation around brain dorsoventral axis

Rotating a volume by an angle α around the brain dorsoventral axis can be exploited to further augment the training set dimensionality. This operation is followed by appropriate interpolation to fit the original volume size. In this work, MRI volumes were rotated of 30° and black voxels were used for interpolation.

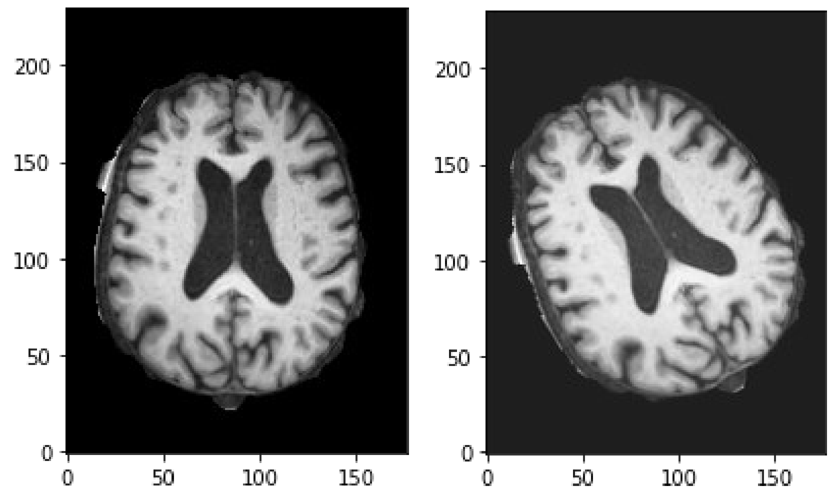


Figure 8 - One axial slice of an MRI volume after FLIRT and BET (on the left) and the same slice after the 30° rotation of the MRI volume (on the right).

5.3 Classification

For what concern classification, a bidirectional convolutional long short-term memory (ConvLSTM) neural network was used in this work. For a better understanding of the potential of this type of network, a brief overview of standard deep neural networks, convolutional neural networks and long short-term memory networks are provided. Then, the proposed architecture is presented.

5.3.1 Deep neural networks

An Artificial Neural Network (ANN) can be seen as a series of algorithms able to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates [45]. An ANN is composed by interconnected 'nodes' organized in a series of layer. A dense net is an ANN in which each layer is densely (i.e., fully) connected to the adjacent layers.

A classic example of dense net is the Multilayer Perceptron (MLP), which consist of an input layer, a hidden layer and an output layer. Except for the input nodes, each node can be considered as a sort of neuron, which receives a series of inputs and delivers an appropriate output to all the nodes of the subsequent layer [45].

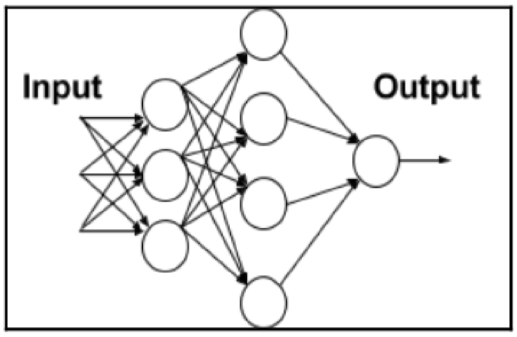


Figure 9 – Multilayer perceptron (MLP) architecture [45].

At the level of each node (except the input nodes), a weighted sum of all the inputs is computed and a constant value usually equal to 1, defined as bias, is added to the result. The obtained value is then given as input to a proper activation function, for example a sigmoid or a rectified linear unit, which produces the final output that will be passed to the next layer.

At the beginning of the training session, all the weights have some random assignment. A generic MPL learns from training data through a process called back propagation [45]. In few words, thanks to the feed forward propagation of the input values through the architecture, output values are generated by the NN. Then the mismatch between the prediction made by the MPL and the actual output (i.e., the error) is propagated backward from the output to the input layer, while an appropriate optimizer algorithm, such as a gradient descent, adjusts the neural network weights trying to reduce the error as much as possible. An ANN is defined as a deep neural network when it is characterized by a significant number of hidden layers of neurons [45].

5.3.2 Convolutional neural networks

CNN is a type of deep learning model inspired by the organization of animal visual cortex, mostly applied to analyze visual imagery [46]. The strength of CNN resides in the fact that they are designed to preserve the spatial locality in images and to learn via progressive levels of abstraction [46].

The structure of a generic CNN usually includes three types of layers: convolution, pooling, and fully connected layers [46]. The first two perform feature extraction, whereas the third maps the extracted features into final output, such as classification.

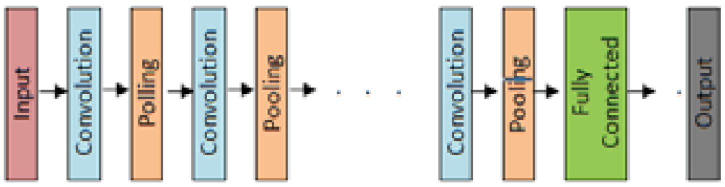


Figure 10 – Convolutional Neural Network (CNN) architecture [46].

The convolutional layer plays a key role in a CNN [47]. In a convolution, a specialized type of linear operation suitable for feature extraction, a small array of numbers, defined as kernel, is applied across the input image. At each location of the input image, an element-wise product between each element of the input image patch and the kernel is computed and the sum of these products is stored in the corresponding position of the output image (i.e., the feature map). To extract many characteristics from the original image, an arbitrary number of kernels are applied to the input image and each generated feature map is assigned to a neuron of the subsequent layer. The kernels, which can be considered as different feature extractors, are later configured based on the back propagation technique.

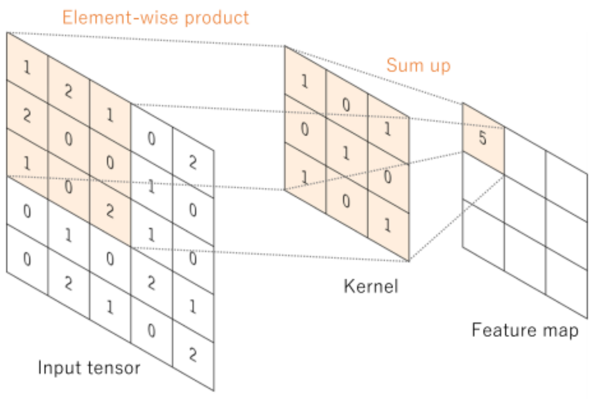


Figure 11 - Example of how the convolution acts [47].

The convolution operation does not allow the center of each kernel to overlap the outermost element of the input image, leading to a reduced image dimension [47]. Padding, especially zero padding, may be used to have feature maps of the same dimension of the original input image [47].

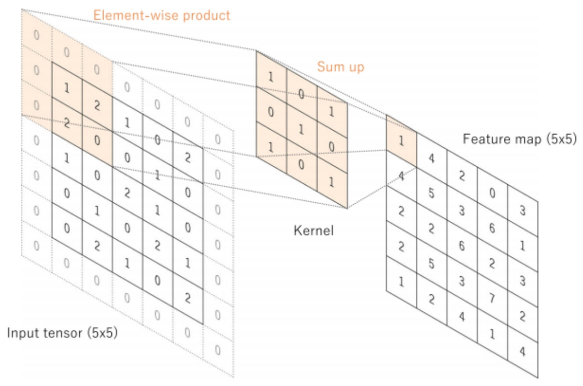


Figure 12 - Example of how the zero padding acts [47].

Pooling layers execute, instead, a down sampling operation which reduces the dimension of the feature maps in order to introduce a translation invariance to small shifts and distortions and decrease the number of subsequent learnable parameters [47]. Max pooling is the most common form of pooling operation. It selects patches of the feature map and provides as outputs the maximum value in each patch. In most of the cases, a 2 × 2 filter with a stride of 2 is commonly employed, reducing the dimension of feature maps by a factor of 2 [47].

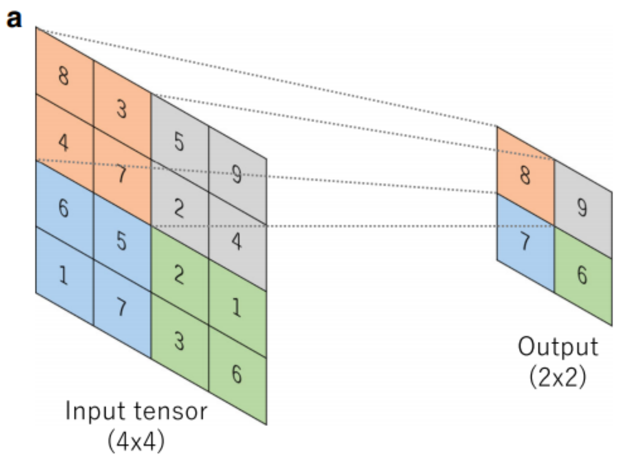


Figure 13 - Example of max pooling operation [47].

Finally, in the last part of a CNN, there is one (or more) fully connected layer(s). This layer flattens the feature maps of the previous layer, turning them in one-dimensional arrays of numbers. Many dense layers may follow until the output layer, usually performing classification, is reached [47].

Thanks to this peculiar structure, CNN automatically and adaptively learn spatial hierarchies of features through back propagation, outperforming classical dense net in image classification [47].

3D CNNs, which work on volumes instead of images, share the same structure of classical CNN but the involve 3D convolution using 3D kernel and 3D max pooling operations [48].

5.3.3 Recurrent neural network: long short-term memory networks

Recurrent Neural Networks (RNN) are a class of neural networks that exploit the sequential nature of their input [45]. They are mostly suitable for time series, where the occurrence of an element in the sequence is dependent on the elements that appeared before it [45].

MLP neurons tend to consider each input as independent from the other, whereas the RNN neuron try to incorporate the interdependency between subsequent inputs by having a hidden state, or memory, that holds the essence of what has been seen so far [45].

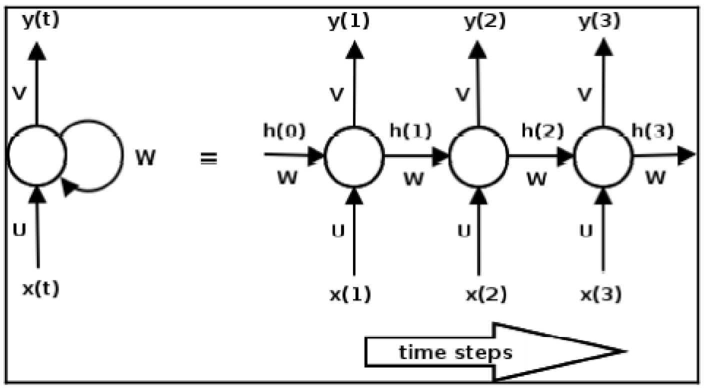


Figure 14 – Recurrent Neural Network (RNN) node (on the left) and its unrolled version (on the right) [45].

The hidden state value at each time instant is obtained as a function of the current input and the value of the hidden state at the previous time step, whereas the output of the neuron is a function of the computed current hidden state [45].

1) $h_t = \tanh\,(Wh_{t-1} + Ux_t)$

2) $y_t = sofmax(Vh_t)$

The weight matrices U, V, and W are updated during learning thanks to a process called back propagation through time. Indeed, in this specific case, the gradient at each output depends not only on the current time step, but also on the previous ones [45]: in each time step we must sum up all the previous contributions until the current one.

Unfortunately, RNNs may suffer from the vanishing or exploding gradients problems, which hampers learning of long data sequences [45]. In the vanishing gradient problem for example, the back propagation trough time may cause the gradient to become smaller and smaller, causing the parameter updates to become insignificant i.e., no real learning is done.

Long short-term memory (LSTM) NN, a particular case of RNN, have been designed to deal with the vanishing gradient problem and learn long term dependencies more effectively [45]. The following diagram shows how recurrence is implemented in a LSTM.

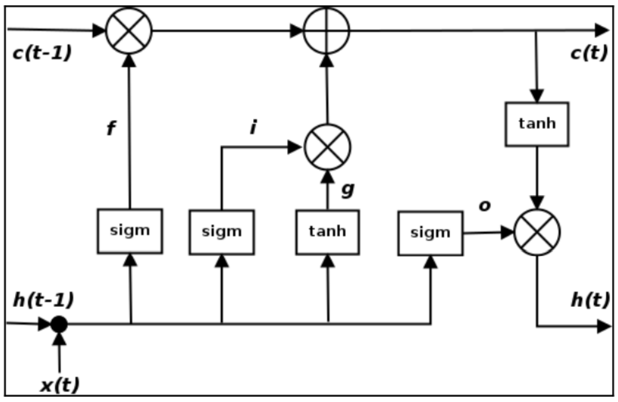


Figure 15 – Long Short-Term memory (LSTM) cell diagram [45].

In LSTM, the current input and the previous instant state are combined in four different ways to obtain i, o, f and g. The first three parameters, computed using the same equations but with different parameter matrices, are respectively the input, output and forget gates. The last parameters, g, is identical to the state variable in the standard RNN cell. All these parameters are involved in the computation of the current hidden state and the current cell state, the latter a new state variable introduced in LSTM. The equations of interest are now reported.

$$3)\ \ i = \sigma(W_i h_{t-1} + U_i x_t)$$
$$4)\ \ f = \sigma(W_f h_{t-1} + U_f x_t)$$
$$5)\ \ o = \sigma(W_o h_{t-1} + U_o x_t)$$
$$6)\ \ g = \tanh(W_g h_{t-1} + U_g x_t)$$
$$7)\ \ c_t = (c_{t-1} \otimes f) \oplus (g \otimes i)$$
$$8)\ \ h_t = tanh(c_t) \otimes o$$

We can notice that, thanks to this way of combining the previous memory and the new input, the LSTM can freely choose to ignore the old memory (setting forget gate to 0) or to ignore the newly computed state (setting the input gate to 0), making the overall architecture resistant against problems like the vanishing gradient one [45].

5.3.4 3D convolutional long short-term memory networks

Even tough LSTM shows great results in modeling long-term dependencies, it fails to model the spatial relationship among pixels in images when employed in image classification tasks [34]. Convolutional Long Short-Term Memory (ConvLSTM) NN can easily solve this problem, exploring the spatial information while keeping the long-term interactions [34]. ConvLSTM cell structure is identical to classic LSTM cell structure, but internal matrix multiplications are exchanged with convolution operations [49]. As a result, the data that flows through the ConvLSTM cells keeps the input dimension instead of being just a 1D vector with features [49].

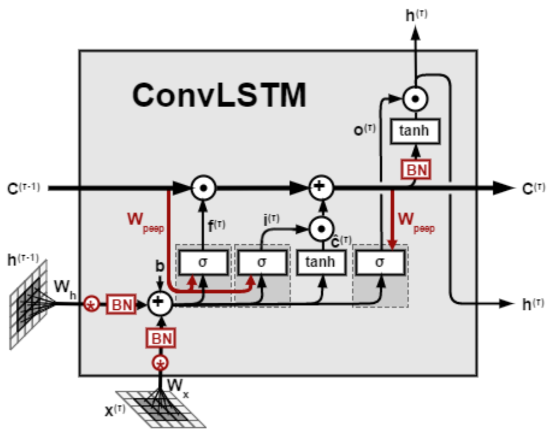


Figure 16 – The Convolutional Long Short-Term Memory (ConvLSTM) cell structure [49].

In the specific case of this work, bidirectional ConvLSTM is an interesting choice to analyze the MRI brain volumes. Indeed, while capturing the 2D spatial information thanks to the convolution with several multiple kernels, the NN treats the slice sequence in its entirety, applying the information of the already processed slice in the analysis of the subsequent ones.

5.3.5 Proposed neural network architecture

The structure of the proposed ConvLSTM NN [50] is graphically represented in the following image.

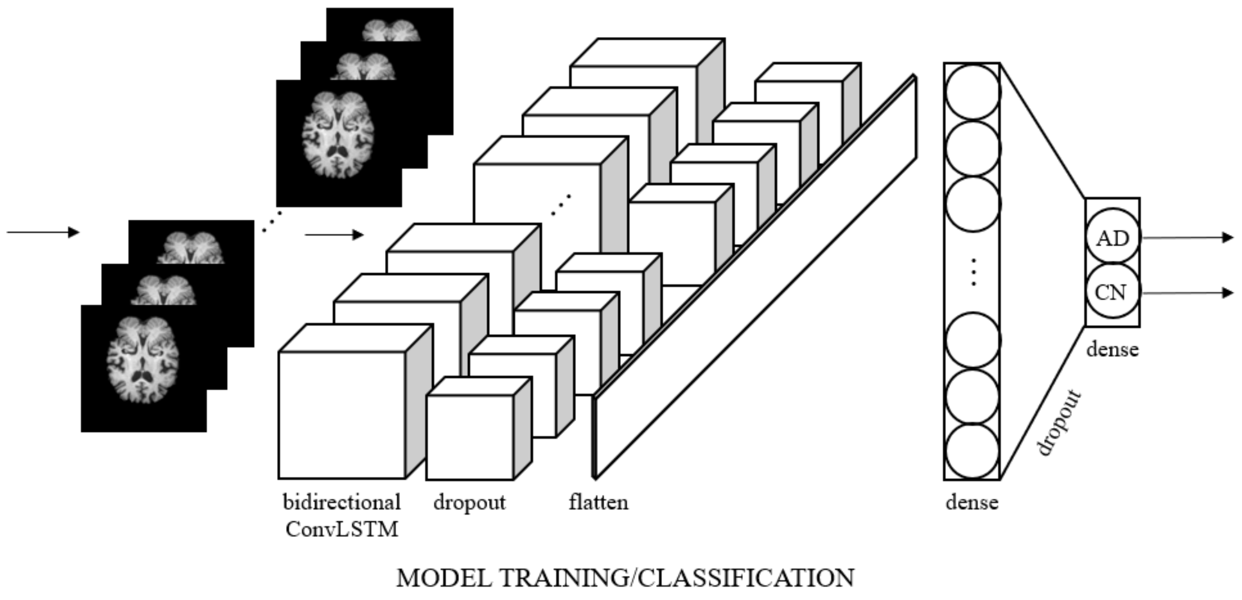


Figure 17 - Proposed architecture, where AD and CN stand for Alzheimer Disease and Cognitively Normal patients, respectively [50].

The proposed NN is composed by 6 subsequent layers.

The first layer is a ConvLSTM, which takes as input the whole MRI brain as a sequence of slices. Thanks to an improved convolution mechanism, this layer can capture and extract discriminative features.

A dropout layer comes next. Dropout is a well-established regularization technique to prevent overfitting [51]. Basically, the dropout layer decides to randomly ignore, according to a certain dropout probability (defined by the designer), some neurons during the training phase [51]. Ignoring a neuron means temporarily removing it from the network, along with all its incoming and outgoing connections [51]. This makes the training process noisy, forcing neurons within a layer to probabilistically take on more or less responsibility for some inputs (increasing the corresponding weight or not) [51].

Then follows a flatten layer that convert the extracted feature maps into a 1D array.

The next layer is a standard dense layer with a Rectified Linear Unit (ReLU) function as activation function. A ReLU is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. This type of function should be used exclusively in the hidden layers and not in the output layer. This because, if a neuron turns to be 0, it is unlikely for it to recover.

Then, another dropout layer is involved.

The last layer is another standard dense layer. This time the Softmax activation function is used so to output a real number between 0 and 1. Indeed, a Softmax activation function squashes the output to real values between 0 and 1, thus it must be used in the output layer. The final classification is based on the Softmax output value.

5.4 Data

In this work a dataset belonging to the OASIS database [52]. Thanks to OASIS project, which aim to facilitate future discoveries in basic and clinical neuroscience, neuroimaging data sets of the brain freely accessible by the scientific community [52].

OASIS
OPEN ACCESS SERIES
OF IMAGING STUDIES

Figure 18 - OASIS logo [51].

From the dataset OASIS-3, one raw T1-weighted sMRI scan was selected from each patient and, in case of multiple scans per patient, the one acquired first was chosen so to avoid intra-patient bias. OASIS-3 provides scans coming from anonymized between with an age between 42 and 97. The resulting dataset was finally composed by 275 scans, 130 coming from controlled patients and 145 coming from patient affected by AD. Each scan, stored as NIFTI files, has a dimension of 176x256x256, with voxels of size 1mm^3.

5.5 Experiments

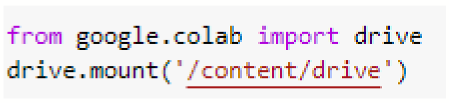
The present work consists of 9 experiments aimed at the quantification of the effect of registration, brain extraction, data augmentation and parameter tuning on the AD classification performances of the proposed NN architecture. Each experiment, expect for the first, starts from the previous experiment and adds something new.

The pro version of Google Colaboratory (also known as Colab) [53], a free Jupyter notebook environment that runs in the cloud and stores its notebooks on Google Drive, was used to run the Python codes involved in this work. GPU hardware acceleration and high RAM were needed to work with such complex NN architecture as ConvLSTM.

5.5.1 First experiment – raw structural magnetic resonance scans

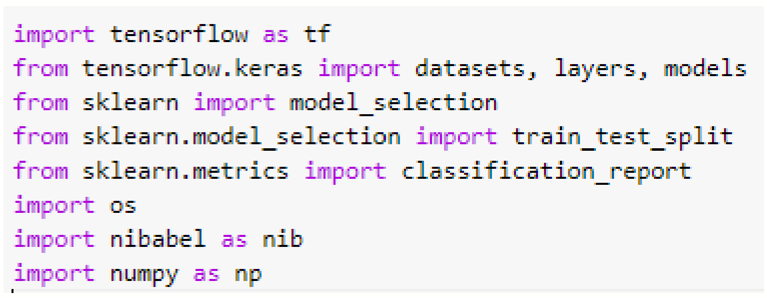
In the first experiment, the NN was trained, validated, and tested on the raw dataset. The preprocessing consisted just in data normalization.

What follows is the Python code involved in this experiment.



```
from google.colab import drive
drive.mount('/content/drive')
```

The function *'drive'* is imported from *'google.colab'* library to mount the drive directory in the current session.



```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import os
import nibabel as nib
import numpy as np
```

In this patch of code, the main involved libraries (i.e., tensorflow, sklearn, os, nibabel and numpy) and function are imported.

```
img_path = "/content/drive/MyDrive/Tirocinio & tesi G. Covella/Dataset/OASIS-3"
img_height, img_width = 230, 176
seq_len = 50 # number of slices
```

In *‘img_path’*, a string containing the path of the dataset directory is stored. The desired image height and image width are then store in *‘img_height’* and *‘img_width’* respectively. The reason why image height was set to 230 instead of 256 will be explained later. Finally, in *‘seq_len’* the number of slices selected from each volume is stored. Just 50 slices were used of the original 256 in order to avoid the Out Of Memory (OOM) error, given the high computational power required by such a task.

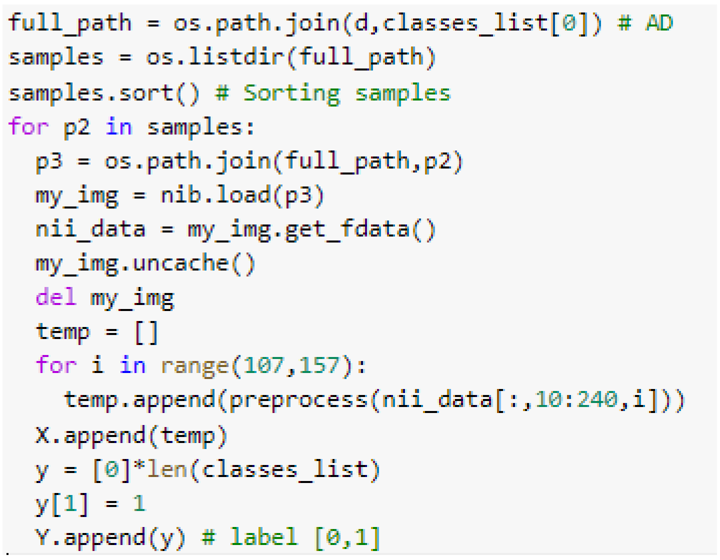
```
def preprocess(img):
  # Input normalization
    mean = img.mean()
    std = img.std()
    return (img - mean) / std
```

Here, the *‘preprocess’* function is defined. It takes an MRI slice as input, and it executes the whitening of the image. Provided that ‘img’ is a Nibabel image, the mean of the voxels intensity can be computed with the method *‘.mean()’*, while the standard deviation can be extracted with the method *‘.std()’*. The function returns the whitened version of the original image: the mean (*‘mean’*) is subtracted from each voxel intensity and then everything is divided by the standard deviation (*‘std’*).

```
X = []
Y = []

d = img_path
classes_list = os.listdir(d)
classes_list.sort()
```

‘X’ and *‘Y’* are initialized as empty lists. The content of the string *‘img_path’* is stored in a new string *‘d’*, this to simplify its subsequent usage. Thanks to the function *‘os.listdir’*, a list containing the names of the element inside the directory identified by *‘d’* is generated. Inside the OASIS-3 directory,

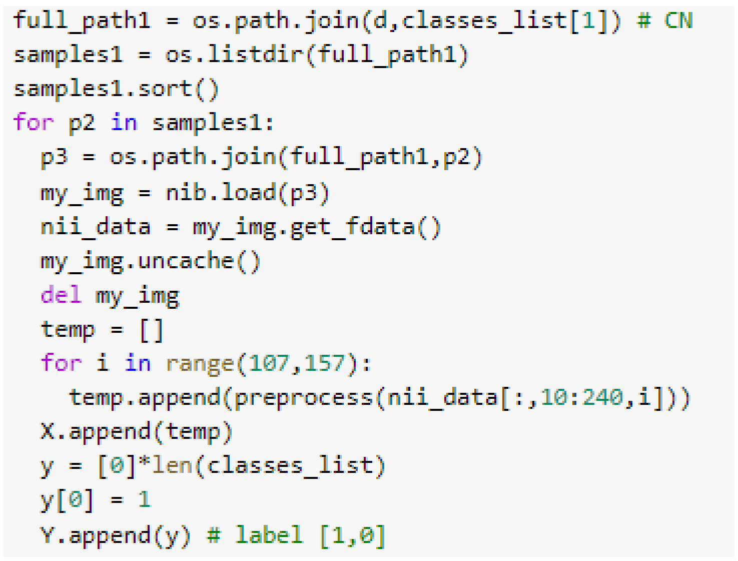
two directories, namely ‘AD’ and ‘CN’, are located. *‘classes_list’* is then sorted so to have ‘AD’ as first element and ‘CN’ as second one.



```
full_path = os.path.join(d,classes_list[0]) # AD
samples = os.listdir(full_path)
samples.sort() # Sorting samples
for p2 in samples:
  p3 = os.path.join(full_path,p2)
  my_img = nib.load(p3)
  nii_data = my_img.get_fdata()
  my_img.uncache()
  del my_img
  temp = []
  for i in range(107,157):
    temp.append(preprocess(nii_data[:,10:240,i]))
  X.append(temp)
  y = [0]*len(classes_list)
  y[1] = 1
  Y.append(y) # label [0,1]
```

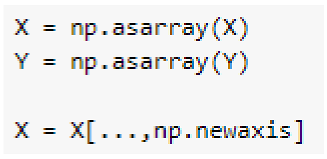
The path of the ‘AD’ directory is obtained joining together *‘d’* and the first element of *‘classes_list’* in a single string *‘full_path’* thanks to the function *‘os.path.join’*. Then, again with *‘os.listdir’*, a list containing the names of all the element of the directory ‘AD’ is generated and named *‘samples’*. The sample names are then sorted alphabetically thanks to the list method *‘.sort’*. Inside a for cycle, which accomplish a number of iterations equal to the length of the list *‘samples’*, the *‘X’* and *‘Y’* start to be populated.

For each MRI file, the following operations are executed. In *‘p3’* the file path is stored joining together *‘full_path’* and the current sample name *‘p2’*. The function *‘nib.load’* loads in *‘my_img’* the nifti file related to the path *‘p3’*. The method *‘.get_fdata’* turns the loaded image in a numpy array, which is stored in *‘nii_data’*. After that, my_img is removed from the cache and deleted. A new empty list *‘temp’* is generated. In a for cycle, 50 slices of the original MRI scan (from the 107 to the 156, both included) are given as input to the function *‘preprocess’* and then appended to *‘temp’* thank to the method *‘.append’*. Of each slice, all the columns are maintained, while only 230 rows of 256 are selected (from the 10th to the 239th rows, both included). Indeed, each MRI volume presents an empty portion in the frontal part and in the posterior part, so, with this precise selection, these non-significant portions are partially deleted. Once all the slices have been appended, ‘temp’ is appended to the *‘X’* list. Finally, an 1x2 array *‘y’* filled with zeros is generated, a ‘1’ is assigned in the second column ([0,1] is the AD label) and the *‘y’* array is appended to *‘Y’*.



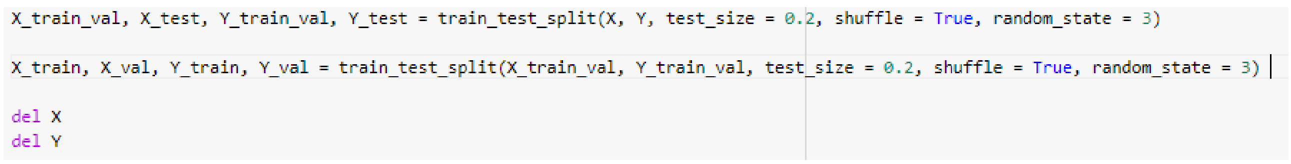
```
full_path1 = os.path.join(d,classes_list[1]) # CN
samples1 = os.listdir(full_path1)
samples1.sort()
for p2 in samples1:
  p3 = os.path.join(full_path1,p2)
  my_img = nib.load(p3)
  nii_data = my_img.get_fdata()
  my_img.uncache()
  del my_img
  temp = []
  for i in range(107,157):
    temp.append(preprocess(nii_data[:,10:240,i]))
  X.append(temp)
  y = [0]*len(classes_list)
  y[0] = 1
  Y.append(y) # label [1,0]
```

The same procedure is followed for the ‘CN’ samples. This time the assigned label in the *‘Y’* list is [1,0].



```
X = np.asarray(X)
Y = np.asarray(Y)

X = X[...,np.newaxis]
```

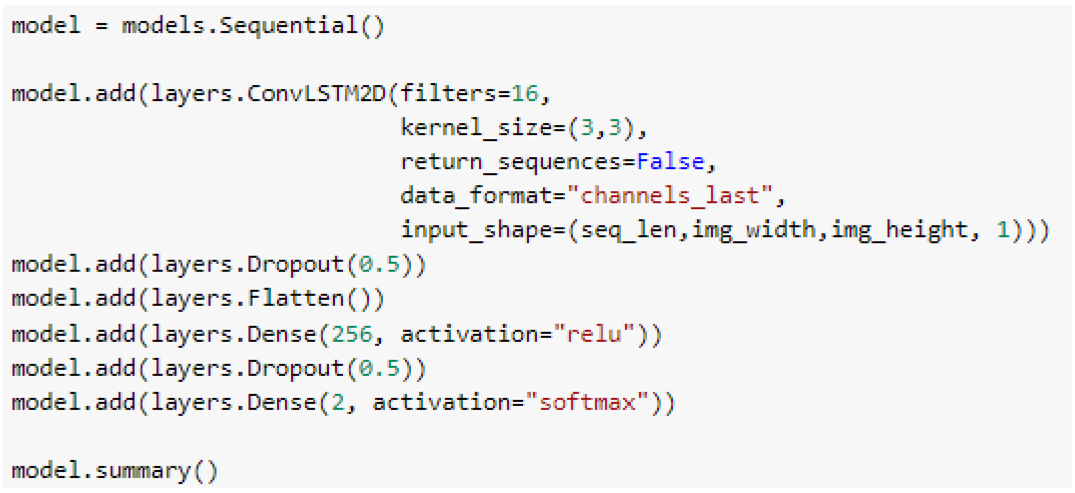
Thanks to the function *‘np.asarray’*, *‘X’* and *‘Y’* are turned into NumPy arrays. A new axis is then added to *‘X’* thanks to the function *‘np.newaxis’*. This is necessary to match the dimension requested by the training function that will be used later.



```
X_train_val, X_test, Y_train_val, Y_test = train_test_split(X, Y, test_size = 0.2, shuffle = True, random_state = 3)

X_train, X_val, Y_train, Y_val = train_test_split(X_train_val, Y_train_val, test_size = 0.2, shuffle = True, random_state = 3)

del X
del Y
```

Here, *‘X’* and *‘Y’* are split in training, validation, and test sets. In the first line, the function *‘train_test_split’* is used to divide the test set from the rest of the dataset. Along with *‘X’* and *‘Y’* as inputs, in the function *'test_size'*, *'shuffle'* and *'random state'* are specified. *'test_size'* defines how to perform the split: in this case, assigning '0.2', the test set will be equal to the 20% of the initial dataset. *'shuffle'*, set to true, shuffles the data before splitting. *'random_state'* controls the shuffling applied to the data before applying the split: each assigned integer (3 in this case) is related to a specific shuffle.

The remaining 80% of the original dataset is again split in training set (80%, so 64% of the original dataset) and validation set (20%, so 16% of the original dataset). 'X' and 'Y' are then deleted to free memory.



```
model = models.Sequential()

model.add(layers.ConvLSTM2D(filters=16,
                            kernel_size=(3,3),
                            return_sequences=False,
                            data_format="channels_last",
                            input_shape=(seq_len,img_width,img_height, 1)))
model.add(layers.Dropout(0.5))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation="relu"))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation="softmax"))

model.summary()
```

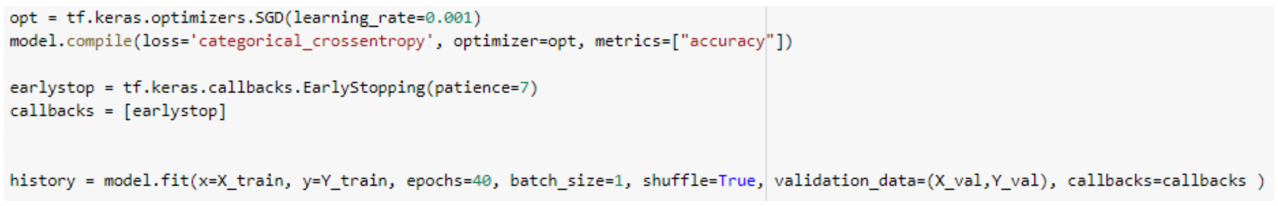
The model is finally defined. *'model'* is defined as sequential model with the function *'models.Sequential'*. The stack of six layers will be attached to the sequential model through the method *'.add'*.

The first layer is assigned thanks to the function *'layers.ConvLSTM2D'*. *'filters'* allows to select the number of kernels that will be convoluted to the inputs, *'kernel size'* specifies the desired kernel size, *'return sequence'* set to false impose to return the whole sequence of outputs and not just the last one, *'data_format'* set to *'channels_last'* specifies that the channel is in the final part of the input and *'input_shape'* specifies the input shape. For this experiment, 16 kernels of dimension 3x3 were used.

The second and the fifth layers are assigned thanks to the function *'layers.droput'*, specifying in this case a dropout rate of 0.5.

The third layer is assigned thanks to the function *'layers.Flatten'*.

The fourth and sixth layers are assigned thanks to the function *'layers.Dense'*, specifying the number of nodes and the activation function with the attribute *'activation'*. In the fourth layer, 256 nodes are assigned with a *'relu'* activation function, while, in the sixth layer, 2 nodes are assigned with a *'softmax'* activation function.



```
opt = tf.keras.optimizers.SGD(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=["accuracy"])

earlystop = tf.keras.callbacks.EarlyStopping(patience=7)
callbacks = [earlystop]


history = model.fit(x=X_train, y=Y_train, epochs=40, batch_size=1, shuffle=True, validation_data=(X_val,Y_val), callbacks=callbacks )
```

In *'opt'* a stochastic gradient descendent optimizer is assigned with the function *'tf.keras.optimizers.SGD'*, specifying with *'learning_rate'* a learning rate of 0.001.

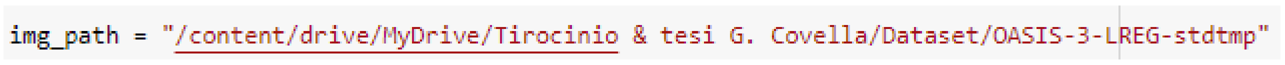
The method *'.compile'* is used to compile *'model'*, specifying *'loss'*, *'optimizer'* and *'metrics'*. *'loss'*, set in this case to *'categorical_crossentropy'*, allows to select the desired objective function to optimize. *'optimizer'*, here equal to *'opt',* allows the selection of the optimizer. *'metrics'*, set in this case to *'accuracy',* allows the selection of the metrics to be evaluated by the model during training and testing.

In *'earlystop'*, with the function *'tf.keras.callbacks.EarlyStopping'*, a training stop condition is set. This function stops training when a monitored metric has stopped improving. In this case imposing *'patience'* equal to 7, training will be stopped after 7 epochs without improvements. *'earlystop'* is then assigned to *'callbacks'*.

Finally, in the training process is executed thanks to the function *'model.fit'*. In *'x'* the training set is assigned, in *'y'* the labels of the training set are assigned, in *'epochs'* the maximal number of epochs is assigned (40 in this case is assigned), in *'batch_size'* the number of sample (MRI volume in this case) per gradient upgrade is specified (in this case, no more than one volume can be processed before each gradient upgrade due to limited memory), in *'validation_data'* the validation set and its labels are assigned, in *'callbacks'* the list of callback are specified. *'shuffle'*, if set to true, will shuffle training data before each epoch.

5.5.2 Second experiment – registration with a standard template

In the second experiment, the raw sMRI scans were registered with respect to a standard template, directly provided by FSL (MNI152_T1_1mm.nii.gz). The registered dataset was then used to train, validate, and test the NN. The code of this experiment is identical to the one of the first, except for the line of code that stores in *'img_path'* the path of the directory containing the registered dataset.



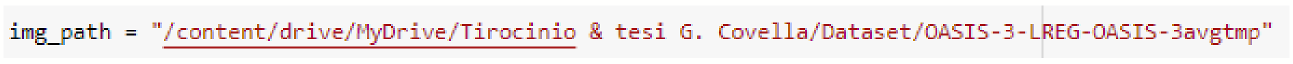
```
img_path = "/content/drive/MyDrive/Tirocinio & tesi G. Covella/Dataset/OASIS-3-LREG-stdtmp"
```

5.5.3 Third experiment – registration with OASIS-3 average template

In the third experiment, the raw sMRI scans were registered with respect to a template generated averaging the raw scans.

The procedure to generate a template with FSL is now explained. From the command prompt, thanks to the utility *'fslmerge'*, the scans to average are concatenated into a single output. Then, with the utility *'fslmath'* (a program to allow mathematical manipulation of images), the average of all the scans is computed so to generate a new template based on the whole dataset.

The registered dataset was then used to train, validate, and test the NN. The code of this experiment is identical to the one of the first, except for the line of code that stores in *'img_path'* the path of the directory containing the registered dataset.

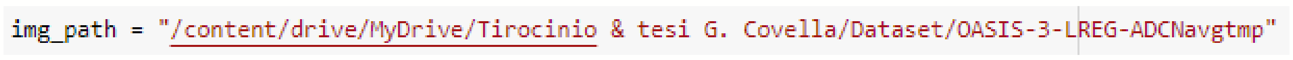


```
img_path = "/content/drive/MyDrive/Tirocinio & tesi G. Covella/Dataset/OASIS-3-LREG-OASIS-3avgtmp"
```

5.5.4 Fourth experiment – registration with AD / CN average templates

In the fourth experiment, the raw sMRI scans were registered with respect to two separate templates. One template, generated averaging the AD scans, was used to register the AD scans, the other, generated averaging the CN scans, was used to register the CN scans.

The registered dataset was then used to train, validate, and test the NN. The code of this experiment is identical to the one of the first, except for the line of code that stores in *'img_path'* the path of the directory containing the registered dataset.

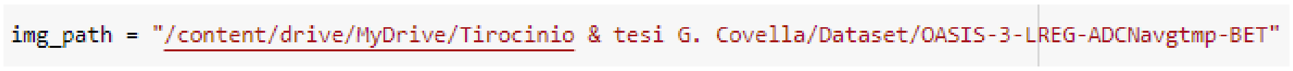


```
img_path = "/content/drive/MyDrive/Tirocinio & tesi G. Covella/Dataset/OASIS-3-LREG-ADCNavgtmp"
```

5.5.5 Fifth experiment – brain extraction

In the fifth experiment, the raw sMRI scans were registered with respect to the two separate templates of the fourth experiment and brain extraction was then executed on the whole dataset.

The preprocessed dataset was then used to train, validate, and test the NN. The code of this experiment is identical to the one of the first, except for the line of code that stores in *'img_path'* the path of the directory containing the registered dataset.

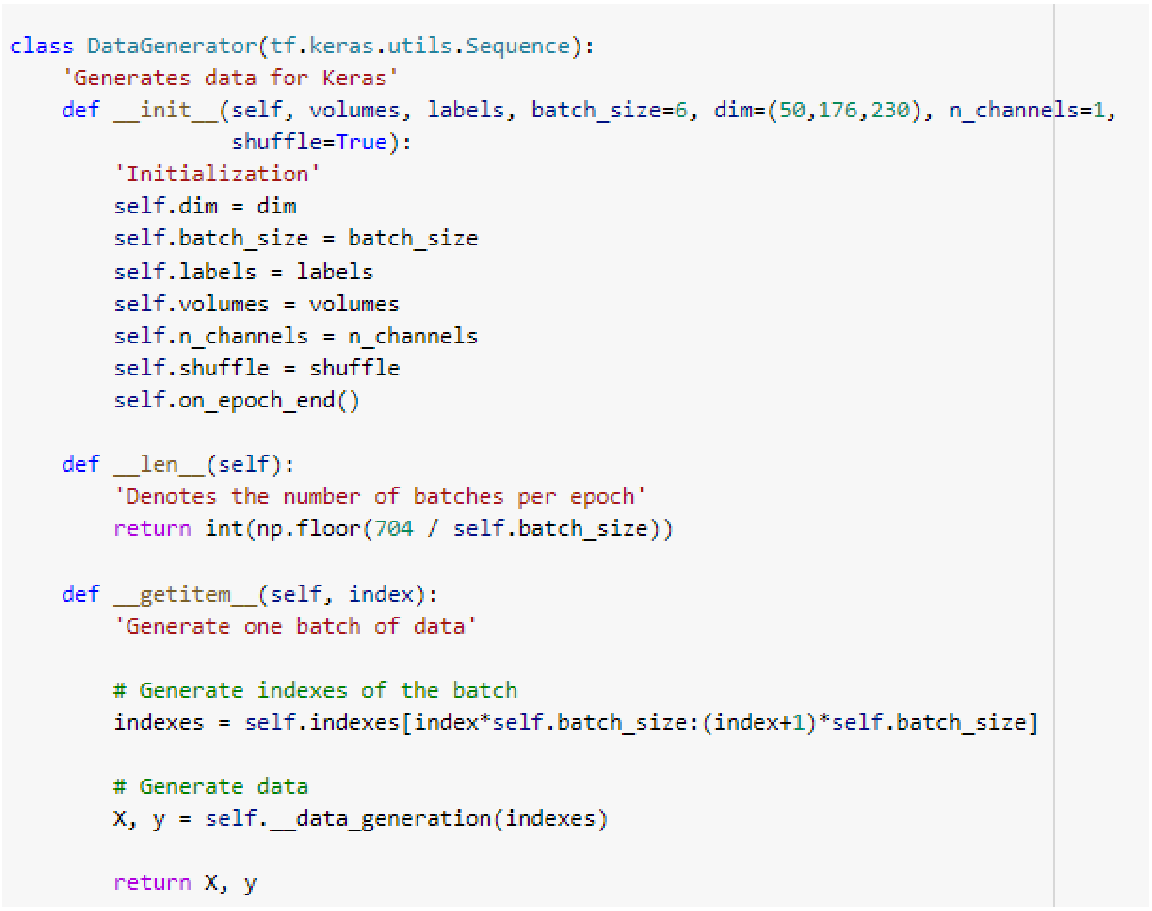


```
img_path = "/content/drive/MyDrive/Tirocinio & tesi G. Covella/Dataset/OASIS-3-LREG-ADCNavgtmp-BET"
```

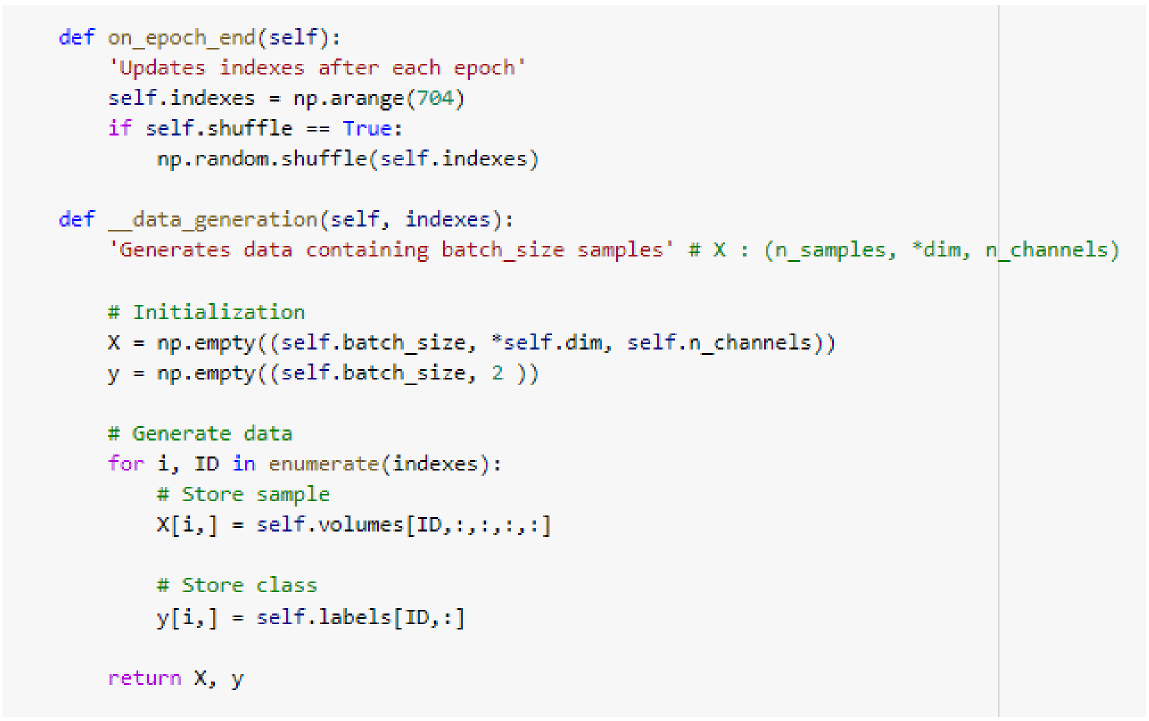
5.5.6 Sixth experiment – generator method

In the sixth experiment, the preprocessing and the data augmentation were executed as in the seventh experiment but this time batches of training data were provided in a different way to the training algorithm.

A data generator has been implemented, allowing to increase the size of the training batch from 1 (*'batch_size'* was set to 1 in all previous experiments to avoid OOM error) to 6 volumes. A larger batch size should help preventing overfitting. The code involved in the design of the data generator is now reported.



```python
class DataGenerator(tf.keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, volumes, labels, batch_size=6, dim=(50,176,230), n_channels=1,
                 shuffle=True):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.labels = labels
        self.volumes = volumes
        self.n_channels = n_channels
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(704 / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'

        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # Generate data
        X, y = self.__data_generation(indexes)

        return X, y
```



```python
    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(704)
        if self.shuffle == True:
            np.random.shuffle(self.indexes)

    def __data_generation(self, indexes):
        'Generates data containing batch_size samples' # X : (n_samples, *dim, n_channels)

        # Initialization
        X = np.empty((self.batch_size, *self.dim, self.n_channels))
        y = np.empty((self.batch_size, 2 ))

        # Generate data
        for i, ID in enumerate(indexes):
            # Store sample
            X[i,] = self.volumes[ID,:,:,:,:]

            # Store class
            y[i,] = self.labels[ID,:]

        return X, y
```

First, the class ‘*DataGenerator*’, which will be used for real-time data feeding to the model, is defined as a *‘tf.keras.utils.Sequence’*, a base object designed to fit a sequence of data like, such as a dataset.

The *'__init__'* function, used to assign values to object (*'self'*) properties, takes as inputs the training volumes as *'volumes'*, the training labels as 'labels', the desired batch size as *'batch_size'*, the size of the single volume as *'dim'*, the number of channels as *'n_channels'* and a Boolean value for *'shuffle'*. In this case, the latter four inputs are fixes according to the dataset characteristics (*'shuffle'* is set to true so to shuffle the training volumes order at each epoch). The *'__init__'* assigns the different inputs to the 'self' properties and calls the *'one_epoch_end'* function.

The *'__len__'* function simply counts the number of batches for each epoch. It returns an integer which is obtained as the floor of the division between the number of training scans and the size of the batch. The floor (the floor of the scalar x is the largest integer i, such that i <= x) is computed with the *'np.floor'* function.

The *'__one_epoch_end__'* function is needed to shuffle the order of the training scans at each epoch. In *'self.indexes'* is first stored a vector containing all the indexes in ascending order thanks to the function *'np.arange'*. Then, if *'self.shuffle'* is set to true (as in this experiment), thanks to the function *'np.shuffle.random'* the *'self.indexes'* vector of indexes is randomly shuffled.

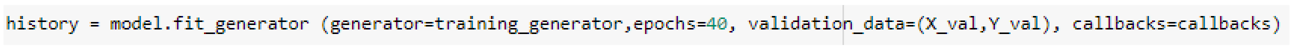
The *'__get_item__'* function is the one that returns each batch, returning the batch of data *'X'* and its labels *'y'*. In *'indexes'* a number of indexes equal to the batch size is extracted. The extraction starts from the beginning of *'self.indexes'* vector, then, at each iteration of the data generator, the variable *'index'* is incremented of 1 and the subsequent batch of indexes is extracted until the end of the *'self.indexes'* vector. Then, the *'__get_item__'* function calls the *'__data_genereation__'* function, providing '*indexes'* as input.

The *'__data_genereation__'* function creates two empty NumPy vectors of the dimension of a batch, one for the scans *'X'* and one for the labels *'y'*. Then the two vectors are populated with a for loop according to the indexes contained in *'indexes'*. Finally, *'X'* and *'y'* are returned.

```
training_generator = DataGenerator(X_train_final, Y_train_final)
```

Once the *'DataGenerator'* class is defined, *'training_generator'* is initialized as a *'DataGenerator'* object, providing as inputs *'X_train'* and *'Y_train'*.

Finally, due to the use of a data generator, also the line used to fit the model is different.



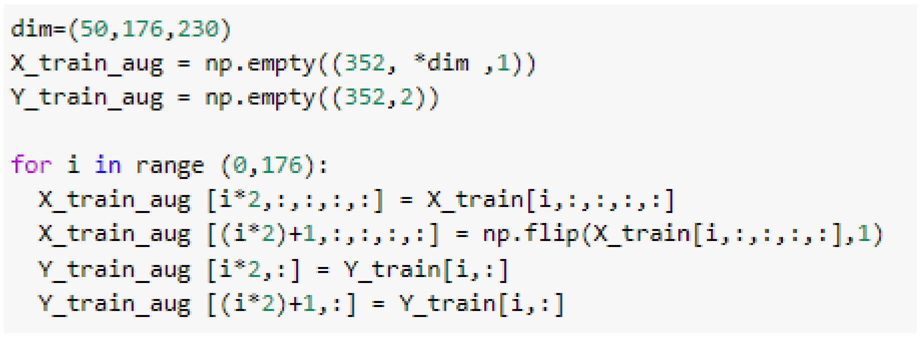
```
history = model.fit_generator (generator=training_generator,epochs=40, validation_data=(X_val,Y_val), callbacks=callbacks)
```

This time *'model.fit_generator'* method is used. It takes the same inputs of the standard *'model.fit'* method but there is no need of specifying the batch size and *'generator'* replaces *'x'* and *'y'* inputs. Of course, *'training_generator'* is assigned to *'generator'*.

5.5.7 Seventh experiment – data augmentation with horizontal flipping

In the seventh experiment, the raw sMRI scans were registered with respect to the two separate templates of the fourth experiment and brain extraction was then executed to the whole dataset. Data augmentation with horizontal flipping was adopted to increase the dimensionality of the training set.

The portion of code involved in data augmentation with only horizontal flipping is the only difference with respect to the code of the fifth experiment.



```
dim=(50,176,230)
X_train_aug = np.empty((352, *dim ,1))
Y_train_aug = np.empty((352,2))

for i in range (0,176):
  X_train_aug [i*2,:,:,:,:] = X_train[i,:,:,:,:]
  X_train_aug [(i*2)+1,:,:,:,:] = np.flip(X_train[i,:,:,:,:],1)
  Y_train_aug [i*2,:] = Y_train[i,:]
  Y_train_aug [(i*2)+1,:] = Y_train[i,:]
```

As already known, *'X_train'* and *'Y_train'*, defined outside this patch of code, contain the sequence of the pre-processed MRI brains and the sequence of the class labels, respectively. In *'dim'* the dimension of a single preprocessed MRI brain is defined. Thanks to the *'np.empty'* function, two suitable empty NumPy vectors are defined to host the augmented sequence of MRI brains and labels, respectively *'X_train_aug'* and *'Y_train_aug'* (there are 176 scans in the training set so these new vectors must be able to contain '2 times 176' elements). Inside the for loop the two empty vectors are populated.

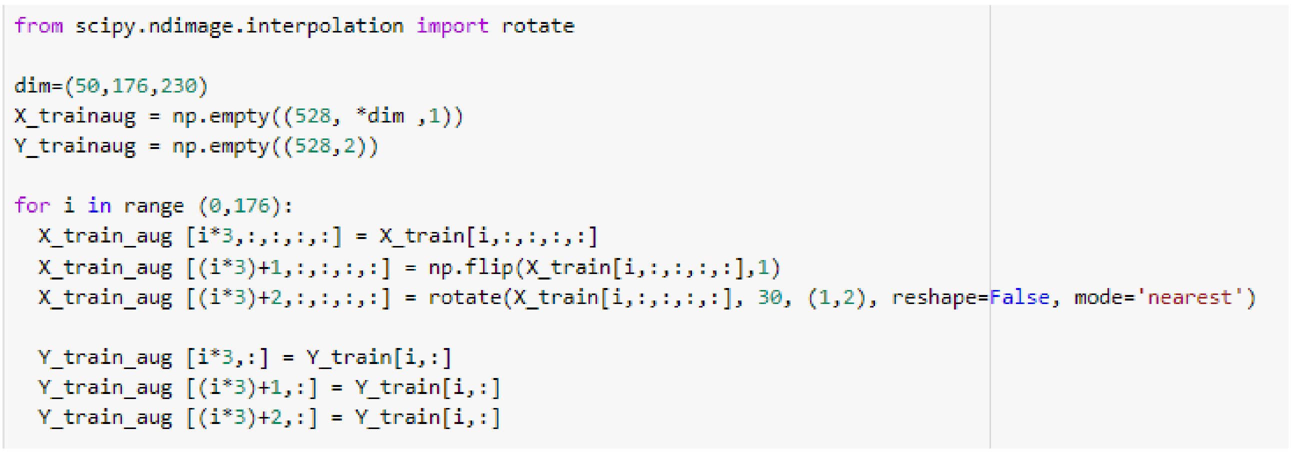
The horizontal flip is executed by the 'np.flip' function take takes as input the volume to flip and a number that identify the flip axis (in this case with '1', the horizontal axis is selected).

The generator method was used to train the NN.

5.5.8 Eighth experiment – data augmentation with horizontal flipping and 30-degree rotation

In the eighth experiment, the raw sMRI scans were registered with respect to the two separate templates of the fourth experiment and brain extraction was then executed to the whole dataset. Data augmentation with horizontal flipping and left 30-degree rotation was adopted to increase the dimensionality of the training set.

The portion of code involved in data augmentation the only difference with respect to code of the sixth experiment.



```
from scipy.ndimage.interpolation import rotate

dim=(50,176,230)
X_trainaug = np.empty((528, *dim ,1))
Y_trainaug = np.empty((528,2))

for i in range (0,176):
  X_train_aug [i*3,:,:,:,:] = X_train[i,:,:,:,:]
  X_train_aug [(i*3)+1,:,:,:,:] = np.flip(X_train[i,:,:,:,:],1)
  X_train_aug [(i*3)+2,:,:,:,:] = rotate(X_train[i,:,:,:,:], 30, (1,2), reshape=False, mode='nearest')

  Y_train_aug [i*3,:] = Y_train[i,:]
  Y_train_aug [(i*3)+1,:] = Y_train[i,:]
  Y_train_aug [(i*3)+2,:] = Y_train[i,:]
```

First, *'rotate'* from *'scipy'* library and *'numpy'* library are imported. Also in this case, *'X_train'* and *'Y_train'*, defined outside this patch of code, contain the sequence of the pre-processed MRI brains and the sequence of the class labels, respectively. In *'dim'* the dimension of a single MRI brain is defined. This time, the two empty NumPy vectors have higher dimensionality (176 times 4). Inside the for loop the two empty vectors are populated.

The horizontal flip is still executed by the 'np.flip' function.

The rotation is executed by the *'rotate'* function, which requires the volume to rotate, the rotation angle (*'30'* in this case) and a couple of integers to identify the rotation axis (in this case with '(1,2)', the dorsoventral axis is selected). Moreover, *'reshape'* is set to false so to keep the brain original size and *'mode'* is set to *'nearest'* so to interpolate the blank voxels with the same content of the image borders (black voxel in this case).

Also in this case, the generator method was used to train the NN.

5.5.9 Ninth experiment – parameter tuning

In the last experiment, the code involved is identical to the one of the eighth experiment and the focus is on parameter tuning. The tunable parameters of interest are the number of filters in the ConvLSTM layer and the dropout thresholds in the two dropout layers. Several combinations of these parameters have been tried to find the one which could maximize the NN performances.

The parameter tuning led to this NN configuration.

- **Batch size**: 10
- **Number of kernels**: 8
- **First dropout threshold**: 0.6
- **Second dropout threshold**: 0.5

6. Results

6.1 Metrics

To evaluate the performances of each experiment model, a series of metrics have been adopted [54].

- **Accuracy**: the ratio between the predictions that the model got right (True Positive (TP) + True Negative (TN)) and the total number of predictions (TP + TN + False Positive (FP) + False Negative (FN)).

$$9)\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: the ratio between TP and TP + FP. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

$$10)\, Precision = \frac{TP}{\mathrm{TP + FP}}$$

- **Recall**: the ratio between TP and TP + FN. The recall is intuitively the ability of the classifier to find all the positive samples.

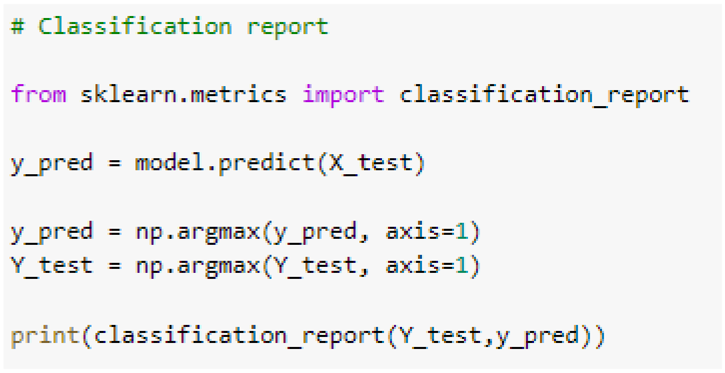
$$11)\, Recall = \frac{TP}{\mathrm{TP + FN}}$$

- **F1-score**: the ratio between 2 times the product of precision and recall and the sum of precision and recall. It can be interpreted as the weighted harmonic mean of the precision and recall, where an F1-score reaches its best value at 1 and worst score at 0.

$$12)\, F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

- **Support**: is the number of effective occurrences of each class in the test set.

These metrics were evaluated using the following Python code.



```
# Classification report

from sklearn.metrics import classification_report

y_pred = model.predict(X_test)

y_pred = np.argmax(y_pred, axis=1)
Y_test = np.argmax(Y_test, axis=1)

print(classification_report(Y_test,y_pred))
```

First *'classification_report'* is imported from *'sklearn.metrics'* library. The method *'.predict'* is used to predict the labels of the test set *'X_test'* according to the trained model *'model'*. Then, '*y_pred'* and *'Y_test'* are flattened into 'n x 1' vectors (where n is the number of test samples), using the function 'np.argamx'. Indeed, '*y_pres'* and *'Y_test'* are 'n x 2' vectors where each rows represent the label of the test sample. The function *'np.argamax'* return for each couple of value (lable) the index of the higher value (so, 0 or 1), generating a 'n x 1' vector. Finally, the results of *'classification_report'*, which takes *'Y_test'* (after *'np.argmax'* of previous code) and '*y_pred'* as inputs, are printed.

Moreover, to further evaluate the model performances, the Receiver Operating Characteristics (ROC) curve and the Area Under the ROC Curve (AUC) have been computed [55]. The ROC curve plots the true positive rate ($\frac{\mathrm{TP}}{\mathrm{TP + FN}}$) (sensitivity or recall) versus the false positive rate ($\frac{\mathrm{FP}}{\mathrm{FP + TN}}$) (1 – specificity) at different classification thresholds. AUC measures the entire two-dimensional area underneath the entire ROC curve: it provides an aggregate measure of performance across all possible classification thresholds.

ROC plot and AUC were obtained using the following Python code.

```
# ROC+AUC
import sklearn.metrics as metrics

# Calculate fpr and tpr for ALL thresholds of the classification
probs = model.predict(X_test)
preds = probs[:,1]

fpr, tpr, threshold = metrics.roc_curve(Y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# Plot
import matplotlib.pyplot as plt
plt.title('ROC curve')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'w--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True positive rate (sensitivity)')
plt.xlabel('False positive rate (1-specificity)')
plt.show()
```

First *'skleran.metrics'* is imported. The prediction of the model 'model' on the test set 'X_test' are stored in *'probs'*, while in *'preds'* the second column of *'probs'* is stored. Thanks to the function *'metrics.roc_curve'*, which takes as inputs the true test labels *'Y_test'* and the predicted one *'preds'*, the false positive rate (*'fpr'*), the true positive rate (*'tpr'*) and the thresholds at which they're evaluated (*'threshold'*), are computed. The AUC is then computed with the function *'metrcis.auc'*, which takes as inputs the *'fpr'* and *'tpr'*.

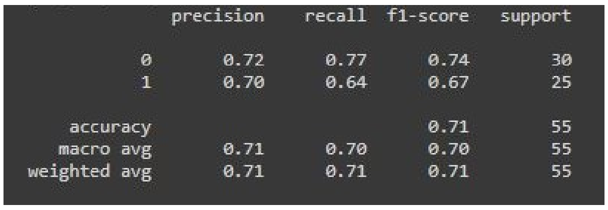
Finally. the ROC curve is plotted along with the corresponding AUC thanks to the functions of the *'matplotlib.pyplot'* library.

6.2 Performances evaluation

The results, in term of classification report and ROC/AUC, are now reported for each experiment.

6.2.1 First experiment results

Figure 19 depicts the classification report of the first experiment.



```
              precision    recall  f1-score   support

           0       0.72      0.77      0.74        30
           1       0.70      0.64      0.67        25

    accuracy                           0.71        55
   macro avg       0.71      0.70      0.70        55
weighted avg       0.71      0.71      0.71        55
```

Figure 19 – Classification report first experiment.

Figure 20 reports the ROC and the respective AUC of the first experiment.

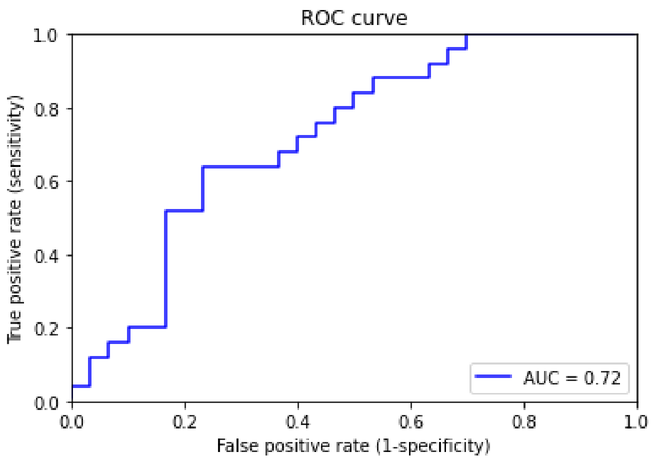
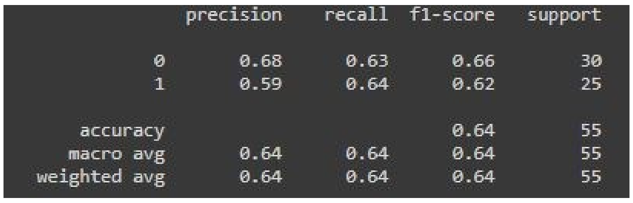


Figure 20 - Receiver Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for first experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate.

6.2.2 Second experiment results

Figure 21 the classification report of the second experiment.



```
              precision    recall  f1-score   support

           0       0.68      0.63      0.66        30
           1       0.59      0.64      0.62        25

    accuracy                           0.64        55
   macro avg       0.64      0.64      0.64        55
weighted avg       0.64      0.64      0.64        55
```

Figure 21 - Classification report second experiment.

Figure 22 reports the ROC and the respective AUC of the second experiment.

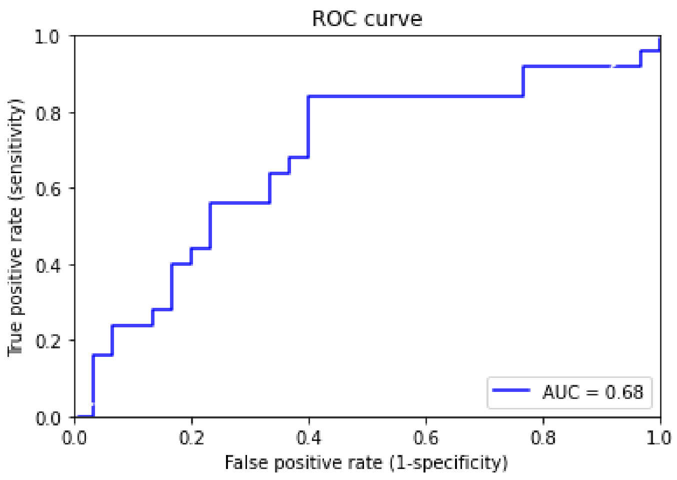
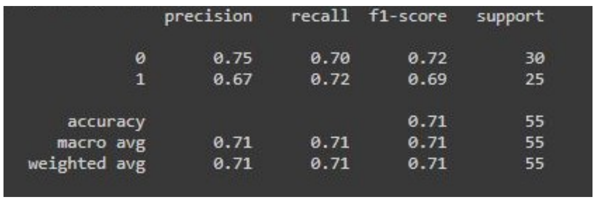


Figure 22 - Receiver Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for second experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate.

6.2.3 Third experiment results

Figure 23 the classification report of the third experiment.



```
              precision    recall  f1-score   support

           0       0.75      0.70      0.72        30
           1       0.67      0.72      0.69        25

    accuracy                           0.71        55
   macro avg       0.71      0.71      0.71        55
weighted avg       0.71      0.71      0.71        55
```

Figure 23 - Classification report third experiment.

Figure 24 reports the ROC and the respective AUC of the third experiment.

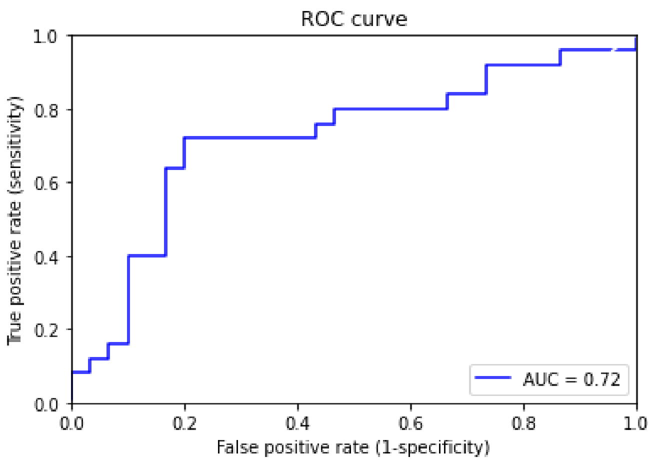
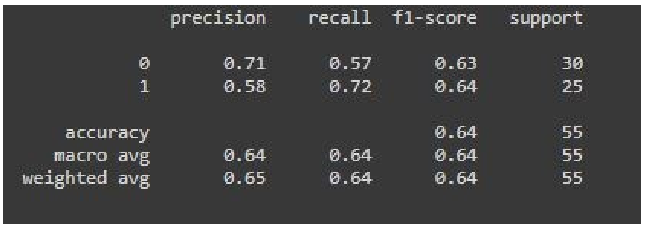


Figure 24 - Receiver Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for third experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate.

6.2.4 Fourth experiment results

Figure 25 the classification report of the fourth experiment.



```
              precision    recall  f1-score   support

           0       0.71      0.57      0.63        30
           1       0.58      0.72      0.64        25

    accuracy                           0.64        55
   macro avg       0.64      0.64      0.64        55
weighted avg       0.65      0.64      0.64        55
```

Figure 25 - Classification report fourth experiment.

Figure 26 reports the ROC and the respective AUC of the fourth experiment.

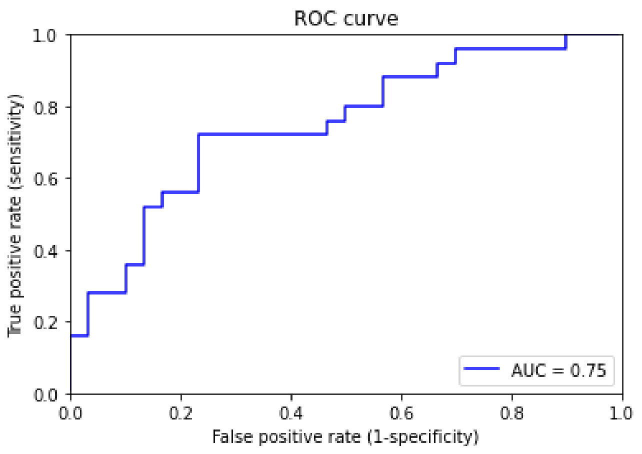
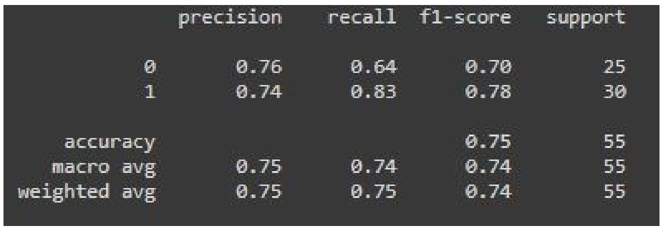


Figure 26 - Receiver Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for fourth experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate.

6.2.5 Fifth experiment results

Figure 27 the classification report of the fifth experiment.



```
              precision    recall  f1-score   support

           0       0.76      0.64      0.70        25
           1       0.74      0.83      0.78        30

    accuracy                           0.75        55
   macro avg       0.75      0.74      0.74        55
weighted avg       0.75      0.75      0.74        55
```

Figure 27 - Classification report fifth experiment.

Figure 28 reports the ROC and the respective AUC of the fifth experiment.

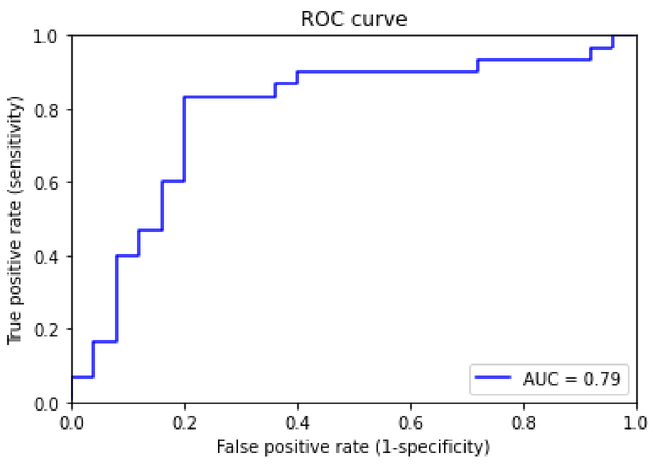
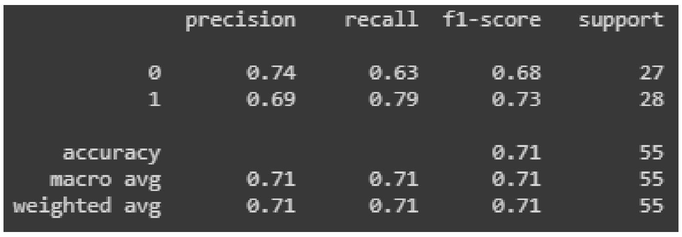


Figure 28 - Receiver Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for fifth experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate.

6.2.6 Sixth experiment results

Figure 29 the classification report of the sixth experiment.



```
              precision    recall  f1-score   support

           0       0.74      0.63      0.68        27
           1       0.69      0.79      0.73        28

    accuracy                           0.71        55
   macro avg       0.71      0.71      0.71        55
weighted avg       0.71      0.71      0.71        55
```

Figure 29 - Classification report sixth experiment.

Figure 30 reports the ROC and the respective AUC of the sixth experiment.

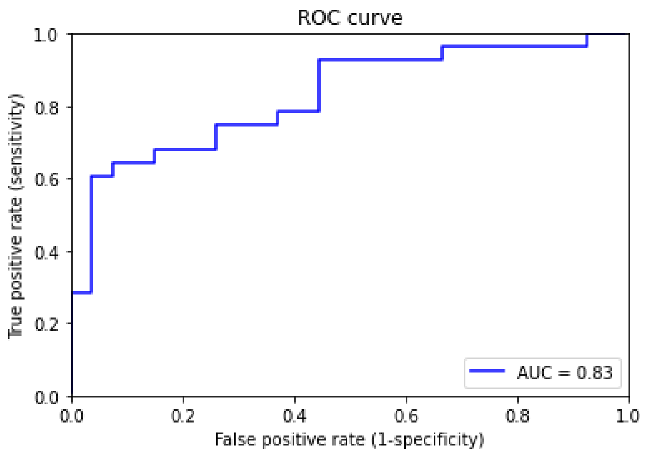
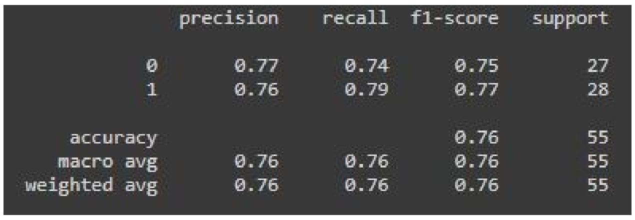


Figure 30 - Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for sixth experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate.

6.2.7 Seventh experiment results

Figure 31 the classification report of the seventh experiment.



```
              precision    recall  f1-score   support

           0       0.77      0.74      0.75        27
           1       0.76      0.79      0.77        28

    accuracy                           0.76        55
   macro avg       0.76      0.76      0.76        55
weighted avg       0.76      0.76      0.76        55
```

Figure 31 - Classification report seventh experiment.

Figure 32 reports the ROC and the respective AUC of the seventh experiment.

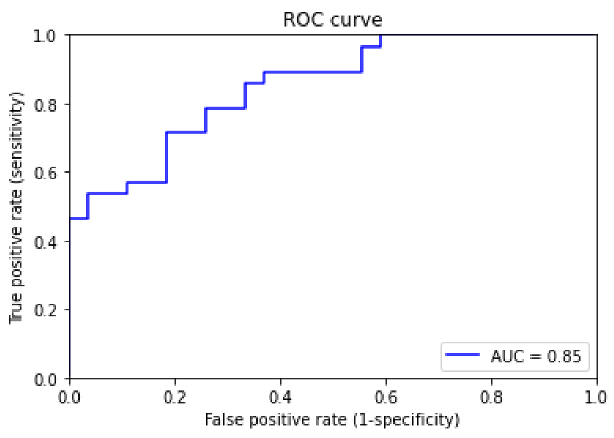
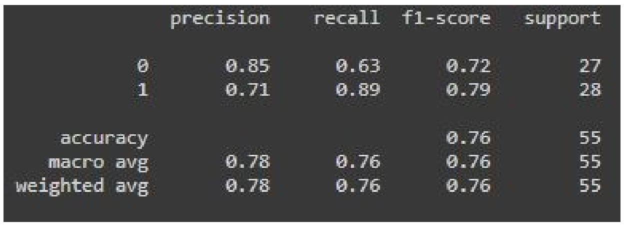


Figure 32 - Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for seventh experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate.

6.2.8 Eighth experiment results

Figure 33 the classification report of the eighth experiment.



```
              precision    recall  f1-score   support

           0       0.85      0.63      0.72        27
           1       0.71      0.89      0.79        28

    accuracy                           0.76        55
   macro avg       0.78      0.76      0.76        55
weighted avg       0.78      0.76      0.76        55
```

Figure 33 - Classification report eighth experiment.

Figure 34 reports the ROC and the respective AUC of the eighth experiment.

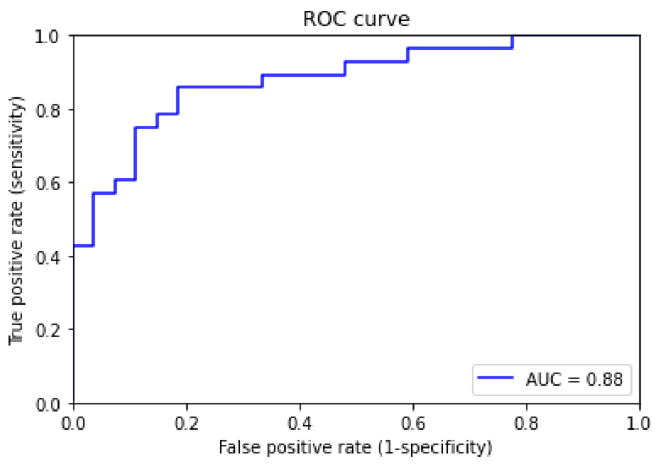
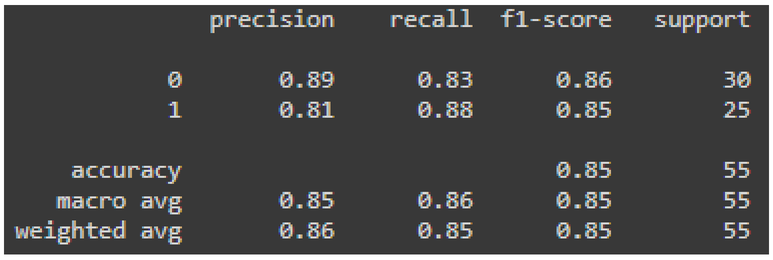


Figure 34 - Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for eighth experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate

6.2.9 Ninth experiment results

Figure 35 the classification report of the ninth experiment.



```
              precision    recall  f1-score   support

           0       0.89      0.83      0.86        30
           1       0.81      0.88      0.85        25

    accuracy                           0.85        55
   macro avg       0.85      0.86      0.85        55
weighted avg       0.86      0.85      0.85        55
```

Figure 35 - Classification report ninth experiment.

Figure 36 reports the ROC and the respective AUC of the ninth experiment.

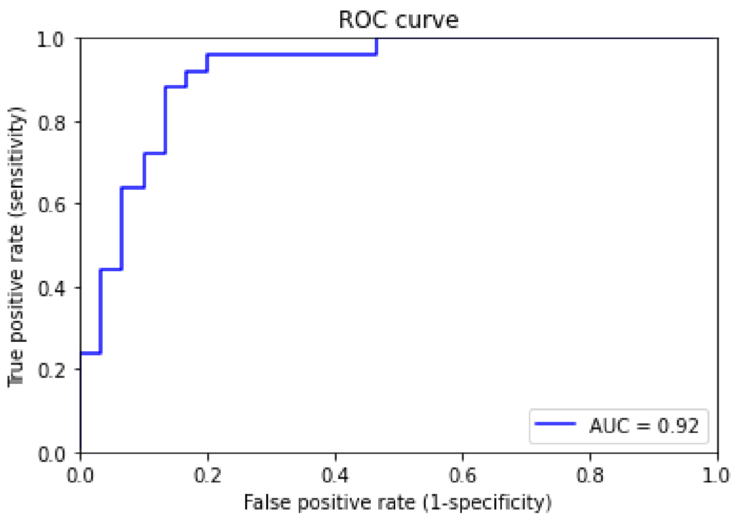


Figure 36 - Operating Characteristics (ROC) curve and Area Under the ROC Curve (AUC) for ninth experiment. On the X-axis there is the false positive rate, while on the Y-axis there is the true positive rate.

7. Discussion

In this work, nine experiments have been conducted to quantify the improvements of scans registration, brain extraction, data augmentation and parameter tuning on the AD classification performances of the proposed ConvLSTM-based framework.

In the first experiment, raw scans, on which only intensity normalization was conducted, were given as input to the neural network. Low performances were recorded, with an accuracy of 71%, a sensitivity of 64% and an AUC of 72%.

In the subsequent three experiments, different types of registration were performed on the raw scans.

Registration to a standard template, as done in the second experiment, led to results even worse than the ones of the first experiment: no improvements in the three metrics of interest are reported, with an accuracy of 64%, a sensitivity of 64% and an AUC of 68%. These unexpected results may be explained by the fact that the chosen standard template (MNI152_T1_1mm.nii.gz) may be not suitable for the registration of OASIS-3 scans, due to their spatial properties.

Experiments three and four, still concerning registration, led instead to a slight improvement of the NN performance. An accuracy of 71%, a precision of 72% and an AUC of 72% were reported using the templated generated over the whole dataset. On the other side, using two separate templates for AD and CN scans, an accuracy of 64%, a precision of 72% and an AUC of 75% were recorded. As we can appreciate from these results, in both cases quite insignificant improvements were obtained. Nevertheless, brain registration should not be seen as a pointless operation: indeed, OASIS-3 dataset includes scans that are already quite similar between them thanks to a standardized acquisition procedure. Thus, registration, even if leading to small enhancements in this work, should be consider a crucial operation in brain scans pre-processing. Registration to two separate AD/CN templates was chosen as best one, mostly due to a higher AUC, and it was involved in all subsequent experiments.

In the fifth experiment, brain extraction was introduced, and this led to a remarkable boost of the NN performances: an accuracy of 75%, a precision of 83% and an AUC of 79% were reached. The removal from the scans of all non-brain portion appears to significantly favour the NN ability to focus on the truly discriminative features which can best distinguish an AD scan from a CN one.

The introduction of the generator method, accomplished in the sixth experiment, allowed to increase the training batch size from 1 to 6: an increased batch size helped preventing overfitting, enhancing the ability of the NN to generalize. An accuracy of 71%, a precision of 79% and an AUC of 83% were reported.

Data augmentation was firstly introduced in the seventh experiment, where, thanks to horizontal flipping, the size of the training set was doubled. A larger training set partially solved the problem linked to the small size of OASIS-3 dataset, leading to an increase of the performance: an accuracy of 76%, a sensitivity of 79% and an AUC of 85%.

In the eighth experiment, the size of the training set was further expanded with scan rotation. Better performance was inevitably recorder, remarking the importance of having a proper size dataset to train the NN. Even tough rotation leads to scans that are, from the spatial point of view, very different from the original ones (this does not happen with horizontal flipping), the inclusion of the rotated scans led to an accuracy of 76%, a sensitivity of 89% and an AUC of 88%.

In the final experiment, parameter tuning allowed to maximise the performance of the NN. Among all the combinations of parameters that were tried, one of them led to very satisfactory results: an accuracy of 85%, a sensitivity of 88% and an AUC of 92%. These results underline the importance of parameter tuning as final step of a NN design process.

In conclusion, the work hereby presented demonstrates the crucial importance of pre-processing steps such as registration and brain extraction, data augmentation and parameter tuning in enhancing the AD classification performance of the ConvLSTM-based framework.

BIBLIOGRAPHY

[6] Khachaturian ZS. Diagnosis of Alzheimer's Disease. Arch Neurol. 1985; 42(11):1097–1105. doi:10.1001/archneur.1985.04060100083029

[7] Francis P.T., Palmer A.M., et al. The cholinergic hypothesis of Alzheimer's disease: a review of progress. Journal of Neurology, Neurosurgery, and Psychiatry. 1999; 66 (2): 137–47. doi:10.1136/jnnp.66.2.137

[8] Hardy J., Allsop D. Amyloid deposition as the central event in the aetiology of Alzheimer's disease. Trends in Pharmacological Sciences. 1991; 12 (10): 383–88. doi:10.1016/0165-6147(91)90609-V

[9] Selkoe D.J., Hardy J. The amyloid hypothesis of Alzheimer's disease at 25 years. EMBO Mol Med. 2016; 8(6):595-608. doi:10.15252/emmm.201606210

[10] Mudher A, Lovestone S. Alzheimer's disease-do tauists and baptists finally shake hands?. Trends in Neurosciences. 2002; 25 (1): 22–26. doi:10.1016/S0166-2236(00)02031-2

[11] Bateman R.J., Aisen P.S., et al. Autosomal-dominant Alzheimer's disease: a review and proposal for the prevention of Alzheimer's disease. Alzheimers Res Ther. 2011; 3(1):1. doi:10.1186/alzrt59

[12] Apostolova L.G., Thompson P.M. Mapping progressive brain structural changes in early Alzheimer's disease and mild cognitive impairment. Neuropsychologia. 2008; 46(6):1597-612. doi: 10.1016/j.neuropsychologia.2007.10.026

[13] Mucke, L. Alzheimer's disease. Nature. 2009. 461, 895–897. doi:10.1038/461895a

[14] Raji C.A., Lopez O.L., et al. Age, Alzheimer disease, and brain structure. Neurology. 2009; 73(22):1899-1905. doi:10.1212/WNL.0b013e3181c3f293

[15] Drew L. An age-old story of dementia. Nature. 2018; 559(7715):S2-S3. doi: 10.1038/d41586-018-05718-5

[16] Sloane P.D., Zimmerman S., et al. The public health impact of Alzheimer's disease, 2000-2050: potential implication of treatment advances. Annu Rev Public Health. 2002; 23:213-31. doi: 10.1146/annurev.publhealth.23.100901.140525

[17] Stephen T., Peter P. Alzheimer’s Disease – The Importance of Early Detection. European Neurological Review. 2008; 3(2):18-21 doi: 10.17925/ENR.2008.03.02.18

[18] Zhang Y., Wang S., et al., Detection of Alzheimer’s disease and mild cognitive impairment based on structural volumetric MR images using 3D-DWT and WTA-KSVM trained by PSOTVAC. Biomed. Signal Process. 2015; Control, vol. 21, pp. 58–73. doi: 10.1016/j.bspc.2015.05.014

[19] Dhawan A.P. Medical image analysis. IEEE Press Series in Biomedical Engineering. John Wiley & Sons, IEEE Press Series in Biomedical Engineering. 200;. pp. 99-138. doi: 10.1002/9780470918548.ch5

[20] Le Bihan D. How MRI Makes the Brain Visible.. Springer, Singapore. 2020; doi: 10.1007/978-981-13-7908-6_2

[21] Sperrin M., Winder J. Magnetic resonance imaging. Scientific Basis of the Royal College of Radiologists Fellowship (2nd Edition). 2019. doi:10.1088/978-0-7503-2148-8ch7

[22] Frisoni G.B., Fox N.C., et al. The clinical use of structural MRI in Alzheimer disease. Nat Rev Neurol. 2010; 6(2):67-77. doi: 10.1038/nrneurol.2009.215

[23] Shen D., Wu G., et al. Deep Learning in Medical Image Analysis. Annu Rev Biomed Eng. 2017; 19:221-248. doi: 10.1146/annurev-bioeng-071516-044442

[24] Luo S., Li X., et al. Automatic Alzheimer’s disease recognition from MRI data using deep learning method. Journal of Applied Mathematics and Physics. 2017; vol. 5, no. 9, pp. 1892–1898. doi: 10.4236/jamp.2017.59159

[25] Tanveer M., Richhariya M., et al. Machine Learning Techniques for the Diagnosis of Alzheimer’s Disease: A Review. ACM Trans. Multimedia Comput. 2020; doi: 10.1145/3344998

[26] Jo T., Nho K., et al. Deep Learning in Alzheimer's Disease: Diagnostic Classification and Prognostic Prediction Using Neuroimaging Data. Front Aging Neurosci. 2019; 11:220. doi:10.3389/fnagi.2019.00220

[27] Ebrahimighahnavieh M.A., Luo S., et al. Deep learning to detect Alzheimer's disease from neuroimaging: A systematic literature review. Comput Methods Programs Biomed. 2020; doi: 10.1016/j.cmpb.2019.105242

[28] Wen J., Thibeau-Sutre E., et al. Convolutional neural networks for classification of Alzheimer’s disease: Overview and reproducible evaluation. Medical Image Analysis. 2020; vol. 63, p. 101694. doi:10.1016/j.media.2020.101694

[29] Noor M. B. T., Zenia N. Z., et al. Application of deep learning in detecting neurological disorders from magnetic resonance images: A survey on the detection of Alzheimer’s disease, Parkinson’s

disease and schizophrenia. Brain Informatics. 2020; vol. 7, no. 1, pp. 1–21. doi:10.1186/s40708-020-00112-2

[30] Korolev S., Safiullin A., et al. Residual and plain convolutional neural networks for 3D brain MRI classification. IEEE 14th International Symposium on Biomedical Imaging. 2017; pp. 835–838. doi: 10.1109/ISBI.2017.7950647

[31] Ullah H. T., Onik Z. A., et al. Alzheimer's disease and dementia detection from 3D brain MRI data using deep convolutional neural networks. IEEE 3rd International Conference for Convergence in Technology. 2018; pp. 1–3. doi: 10.1109/I2CT.2018.8529808

[32] Backström K, et al. An efficient 3d deep convolutional network for alzheimer's disease diagnosis using mr images. Proc. ISBI. 2018; pp. 149–153. doi: 10.1109/ISBI.2018.8363543

[33] Basaia S., Agosta F., et al. Automated classification of Alzheimer's disease and mild cognitive impairment using a single MRI and deep neural networks. NeuroImage: Clinical. 2019; vol. 21, p. 101645. doi: 10.1016/j.nicl.2018.101645

[34] Xia Z., Yue G., et al. A novel end-to-end hybrid network for Alzheimer's disease detection using 3D CNN and 3D CLSTM. IEEE 17th International Symposium on Biomedical Imaging. 2020; pp. 1–4. doi: 10.1109/ISBI45749.2020.9098621

[35] Hemanth J., Anitha, J. Image pre-processing and feature extraction techniques for magnetic resonance brain image analysis. Computer Applications for Communication, Networking, and Digital Contents. 2012; pages 349–356. doi: 10.1007/978-3-642-35594-3_47

[36] Klein A., Andersson J., et al. Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration. *Neuroimage*. 2009; 46(3):786-802. doi:10.1016/j.neuroimage.2008.12.037

[37] Fatima A., Shahid A. et al. State-of-the-Art Traditional to the Machine- and Deep-Learning-Based Skull Stripping Techniques, Models, and Algorithms. J Digit Imaging. 2020; 33, 1443–1464 doi:10.1007/s10278-020-00367-5

[41] Jenkinson M., Smith S. A global optimisation method for robust affine registration of brain images. Med Image Anal. 2001; 5(2):143-56. doi: 10.1016/s1361-8415(01)00036-6

[44] Smith S.M. Fast robust automated brain extraction. Hum Brain Mapp. 2002; 17(3):143-155. doi:10.1002/hbm.10062

[45] Antonio G. and Sujit P. Deep Learning with Keras. Packt Publishing. 2017; doi:10.5555/3153803

[46] Sultana F., Sufian A., et al. Advancements in Image Classification using Convolutional Neural Network. Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN). 2019; doi: 10.1109/ICRCICN.2018.8718718

[47] Yamashita R., Nishio,M., et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging. 2018; 9, 611–629. doi: 10.1007/s13244-018-0639-9

[50] Tomassini S., Falcionelli N., et al. An End-to-End 3D ConvLSTM-based Framework for Early Diagnosis of Alzheimer's Disease from Full-Resolution Whole-Brain sMRI Scans. 34th International Symposium on Computer-Based Medical Systems (CBMS). 2021; doi:10.1109/CBMS52027.2021.00081

[51] Srivastava I., Hinton G., et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014.; 15(56):1929−1958. doi: 10.5555/2627435.2670313

SITOGRAPHY

[1] https://www.ncbi.nlm.nih.gov/books/NBK542179/

[2] https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Anatomy-of-the-Brain

[3] https://ib.bioninja.com.au/options/option-a-neurobiology-and/a2-the-human-brain/brain-matter.html

[4] https://en.wikipedia.org/wiki/Human_brain

[5] https://www.who.int/en/news-room/fact-sheets/detail/dementia

[38] https://medium.com/tensorflow/an-introduction-to-biomedical-image-analysis-with-tensorflow-and-dltk-2c25304e7c13

[39] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki

[40] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FLIRT

[42] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FNIRT

[43] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/BET

[48] https://towardsdatascience.com/step-by-step-implementation-3d-convolutional-neural-network-in-keras-12efbdd7b130

[49] https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

[52] OASIS Brains - Open Access Series of Imaging Studies (oasis-brains.org)

[53] https://colab.research.google.com/

[54]https://scikitlearn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

[55] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc