3D ResNeSt-like Convolutional Neural Network for Alzheimer's Disease Classification

Lorenzo Mandelli 1747091

Alessia Carotenuto 1764400

Abstract

Diagnosing Alzheimer's disease is still a very difficult task, especially in the early stages, as there are no specific tests and therefore no certainty. The purpose of this work is to devise an early diagnosis tool, through magnetic resonance imaging data, with the aim of eventually being able to slow down and/or stopping the evolution of the disease. Moreover, we were also able to diagnose mild cognitive impairment, i.e. by extending the problem from a binary classification one to a ternary classification. Specifically, we make use of artificial intelligence and machine learning techniques, i.e. using residual 3D convolution neural networks.

1 Introduction

Alzheimer's disease (AD) is a progressive dementia where *dementia* is a general term that describes a set of symptoms associated with a decline in memory or other thinking skills severe enough to reduce a person's ability to perform ordinary activities [13]. 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 [12]. By the final stages of AD, a significant loss of brain volume occurs, better defined as widespread "brain atrophy". The observable signs include the shrinkage of cerebral and hippocampal cortex and the enlargement of brain ventricles. In Figure 1 there is a comparison between a healthy brain and a brain of a person with AD.

Symptoms of AD typically begin with mild cognitive impairment (MCI). MCI refers to relatively minor impairments in thought processes and memory, whereas AD is a specific disease in which memory and functioning continue to decline significantly over time [9]. Some researchers would rather identify MCI as an initial stage of AD, especially because studies have shown changes in the brains of people with MCI that are similar to those that are present in AD. However, a significant proportion of people who are diagnosed with MCI never progress to the advanced stages of AD or show other symptoms of dementia, hence MCI is the main differential diagnosis of AD. Nowadays, these pathologies are diagnosed with several difficulties through clinical and instrumental examinations such as magnetic resonance imaging - MRI, positron emission tomography - PET or diffusion tensor imaging - DTI. While MRI and DTI works about nuclear magnetic resonance phenomenon (differently from MRI, DTI enables the measurement of the restricted diffusion of water in tissues) the well known PET makes use of radioactive substances known as radiotracers to visualize and measure changes in metabolic

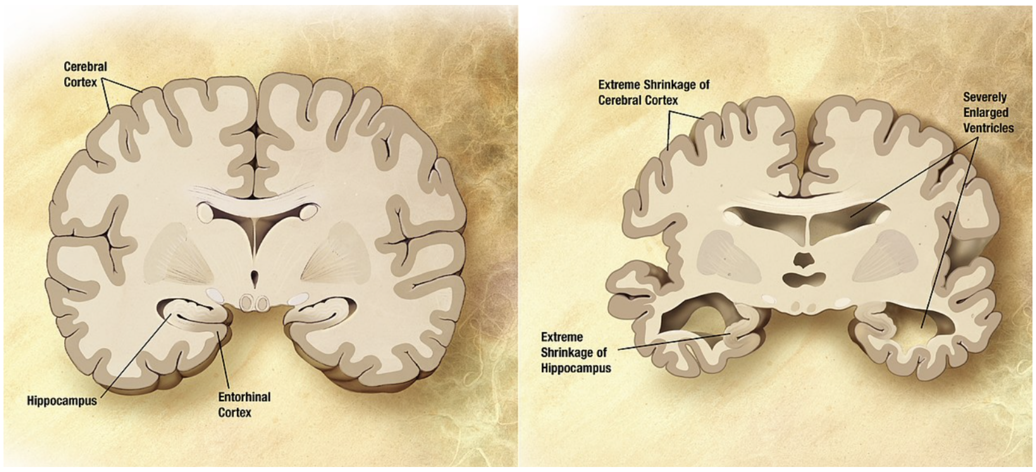

Cerebral Cortex
Hippocampus
Entorhinal Cortex
Extreme Shrinkage of Cerebral Cortex
Severely Enlarged Ventricles
Extreme Shrinkage of Hippocampus

Figure 1: Comparison between a healthy brain (left) and a brain of a person with AD (right)

processes. MRI is often preferred to other previously cited techniques because it is safer considering it does not include any ionizing radiation, but at the same time it is also more expensive. The scanning times are much longer than other radiological techniques, and the temporal resolution is generally quite low (a few images per second for lower spatial resolutions). It is also true that often the diagnosis is made at an advanced stage of the disease so as to make therapeutic treatment difficult or even impossible. Considering that early diagnosis of AD could slow down or even help stopping the disease, with the presented study we have elaborated a diagnostic tool for classifying AD, MCI and cognitive/control normal (CN) patients. The above-mentioned tool will work with the raw MRI-scans of the patient. It is, in fact, part of our work, the integration of automatic minimal pre-processing step in the proposed tool. In the following sections, we introduce some solutions that have been used during years passed reporting the average performances where possible. Next, we illustrate how we built the datasets we used for our experiments together with the data processing techniques we have chosen, focusing on why and how they have been employed. Finally, we show the models we ended up with, discussing pros and cons of the obtained results, concluding doing some considerations about what we think could be the further step for this work.

2 Related Works

The majority of earlier works were focused on the volumetric approaches that perform comparison of anatomical brain structures assuming one-to-one correspondence between subjects, like the widespread voxel-based morphometry, an automatic volumetric method for studying the differences in local concentrations of white and gray matter, the tensor-based morphometry, proposed to identify local structural changes from the gradients of deformation fields of the brain or the object-based morphometry, introduced for shape analysis of anatomical structures [5].

Some recent studies, based on computer-aided early diagnostics of AD and MCI, have shown promising results in the disease determination using structural and functional MRI as well as PET or DTI [5]. For instance in [10], the authors have employed supervised machine learning algorithms to predict AD using psychological parameters such as age, number of visits, education and the mini mental state examination (MMSE). In this work the differential diagnosis is done using MRI scans. Specifically, the algorithms used are the support vector machine (SVM) and the decision tree (DT). The first classifies by separating objects using a hyperplane where the hyperplane is drawn with the help of the margins, which are drawn with the help of the support vectors that belong to the objects; the main purpose is to maximise the distance between the hyperplane and the margin. The second, on the other hand, uses a set of rules to find a solution. The main advantages are as follows: it can solve all kinds and varieties of problems and a small change in the data can have a big impact on the result. The algorithms used give good accuracy, i.e. 85% and 83% for the SVM and DT algorithms respectively. In recent years, deep learning and image classification, has become crucial in medical imaging. As read in [8], "In the medical field, it is necessary to have a safe and fast diagnosis in order to improve treatment outcomes." The most common AI application in medical imaging are:

- classification or examinations of images;
- classification of an object or lesion;
- detection of organs and regions;
- lesion segmentation.

In [6], a classification task of AD, MCI and CN is treated using the well-known Alzheimer Disease Neuroimaging Initiative (ADNI) dataset [1], i.e. a longitudinal multi-center study designed to develop clinical, imaging, genetic, and biochemical biomarkers for the early detection and tracking of AD. Specifically, it is a binary (one-versus-one) classification task, i.e. AD vs CN, AD vs early MCI (EMCI), AD vs late MCI (LMCI), LMCI vs CN, LMCI vs EMCI and EMCI vs CN. The methodologies covered are the use of a VGG-like network architecture [11] and the use of a residual neural network derived from VoxResNet [2]. The metrics used to evaluate the models are accuracy and ROC AUC. The results of the two networks are comparable and show very good results for AD vs CN classification, while the network struggle in distinguishing AD vs EMCI/LMCI patients. The main advantage of the methodology proposed by the authors is to avoid several processing steps for feature extraction by doing the latter automatically.

As we have seen above the problem of MRI classification has always been tackled with complex multi-step pipelines for handcrafted feature generation and feature extraction. The feature obtained from the data fed standard machine learning techniques such as SVM or logistic regression used for classification. We must say that, until few years ago, was very difficult for classical machine learning techniques to use 3D brain information directly, and therefore, 1D information has often been used. Since MRI scans are 3D images, and there are spatial connections among 2D image slices, using 3D convolutional neural networks (CNNs) is suggested in several studies. We can distinguish between a more direct method where the whole MRI scan is taken as input and another very often used method where just specific regions of interest are considered as input. In the last years direct utilization and integration of 3D information has become much easier ([6] , [5]), and different studies in medical field suggests that this new way to tackle this kind of problems could be a valid solution.

3 Data

3.1 Raw Data

We used MRI scans of AD, MCI and CN patients, without discriminating them over age, sex, or any particular marker. A fundamental feature of MRI is the possibility to vary the characteristics of the image simply, by modifying the excitation sequence that the machine performs. In our case study it is suggested to use PD/T2 FSE weighting [3]. To create a T2-weighted image, magnetization can decay before measuring the MRI signal by changing the echo time (TE). This image weighting is useful for detecting oedema and inflammation, revealing white matter lesions and evaluating the anatomy of organs such as the prostate and uterus. In short, the weighting T2 or simply T2 is a measurement of transverse relaxation using long repetition time (TR) and TE times. In addition, T2 has a high signal from water-rich tissues. Moreover, MRI is a multi-plane imaging technique, as it is possible to acquire images on axial, coronal or sagittal planes and different types of sequences in one reading, otherwise known as "multi-parametric acquisition". In our case we just selected the axial projections. Originally each single image composing the brain volume was an high resolution $256 \times 256 \times 48$ (Figure 2).

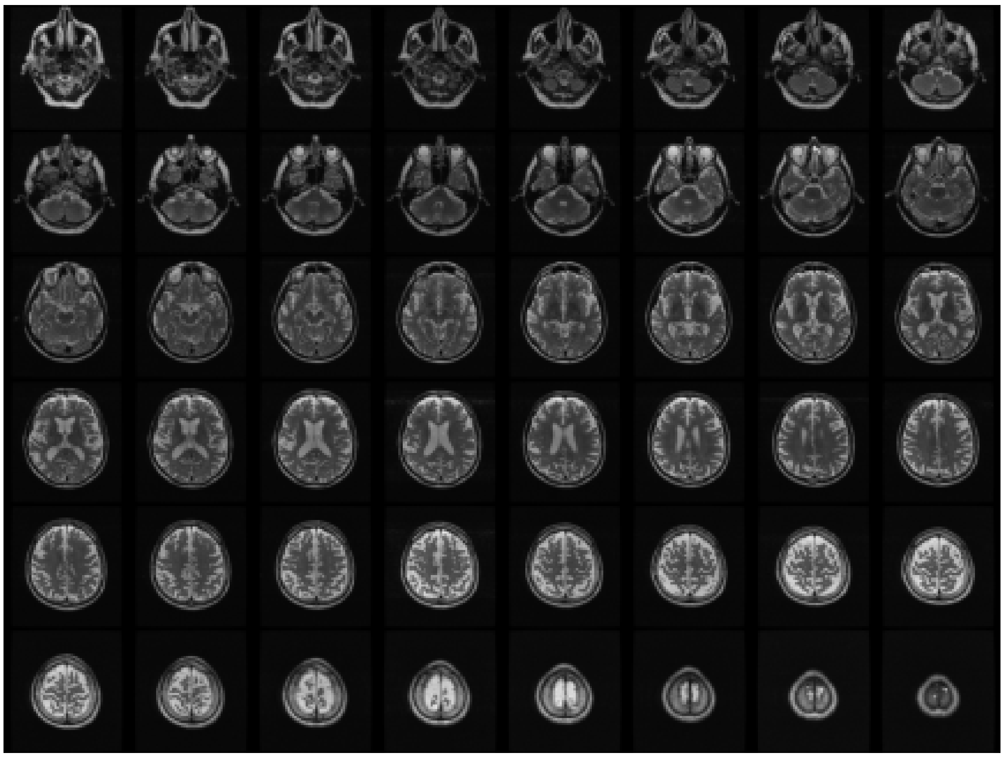


Figure 2: Original data

3.2 Preprocessed Data

The processing we have applied regards mainly the data dimensionality, pixel normalization and some filtering. We tried a large number of different combinations of these processing techniques finding out that a good choice is resizing the data as $64 \times 64 \times 32$ volumes and normalizing the pixels between $+200$ and -200 when applying one between the Sobel and Laplacian of Gaussian (LoG) operators. Since AD shows up with a consistent loss of brain volume, we aimed at having images that are as real as possible without too much preprocessing applied. Unfortunately, we noticed that using the original images did not lead to any significant results as there is too much white noise in the images, leading the network to hardly being able to generalize. As mentioned before using derivative filters, such as Sobel's filter and LoG's filter, helped in building a much more representative data. Among all the tests volumes processed with a LoG filter seems to let the network reach higher results.

3.2.1 Sobel Filter

The Sobel filter can generally be used to emphasize contours in an image. The operator uses two 3×3 kernels operator which are convolved with the original image to calculate approximations of the derivatives, one for horizontal changes and one for vertical changes. G_x and G_y are the filters, for instance we can design them as:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

If $\mathbf{A}$ is the source image by convolving $\mathbf{A}$ with G_x and G_y we obtain the images which at each point contain the horizontal and vertical derivative approximations respectively. The Figure 3 shows a visualization of the data processed with the Sobel operator.

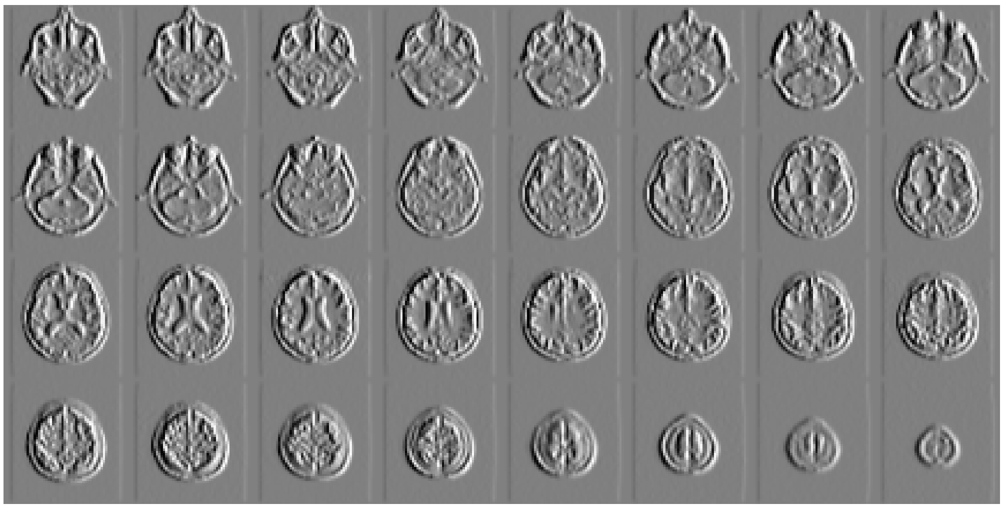


Figure 3: Images processed with a Sobel filter

3.2.2 LoG Filter

The LoG filter is used in computer vision and image processing as blob detector, i.e. its purpose is to analyse an image and detect regions that differ in properties, such as colour, brightness, etc., compared to surrounding regions. The LoG function centered in zero and with Gaussian standard deviation σ has the form:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}[1 - \frac{x^2+y^2}{2\sigma^2}]e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

In Figure 4 it is displayed a 3D visualization of the LoG function.

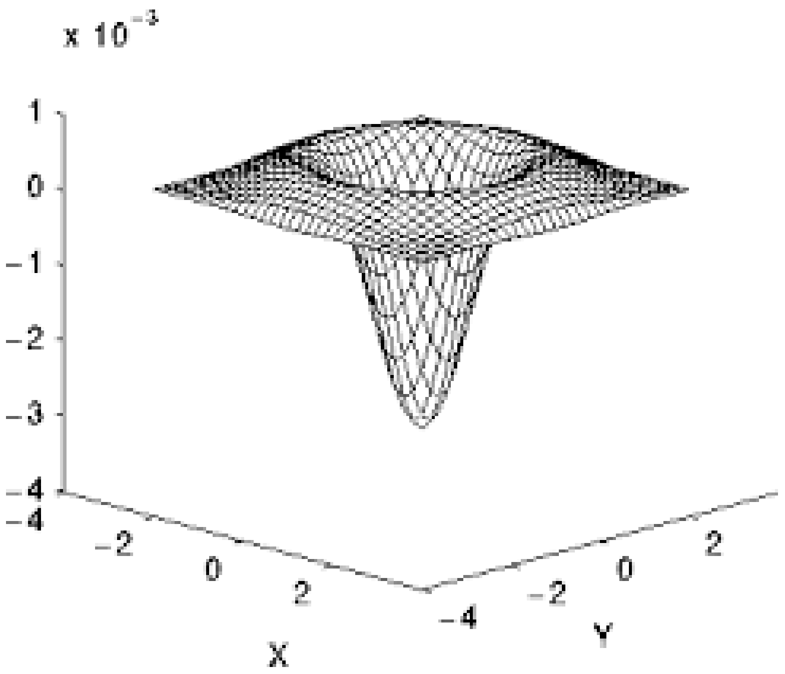


Figure 4: Laplacian of Gaussian

A discrete kernel that approximate this function for a Gaussian with $\sigma = 1.4$ is:

$$\begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -24 & -40 & -24 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \end{bmatrix}$$

The Figure 5 shows a visualization of the data processed with the LoG operator.

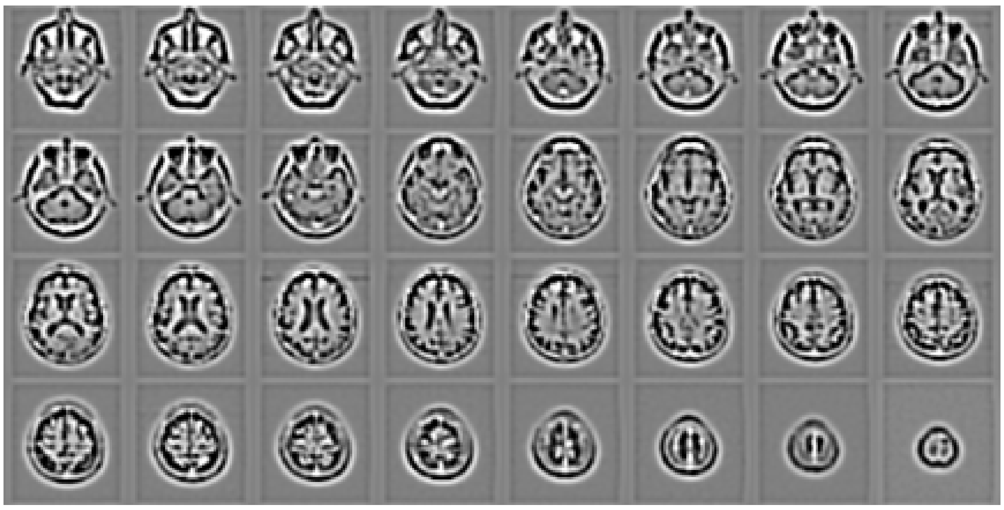


Figure 5: Images processed with a LoG filter

4 Dataset

The classification problem aforementioned could be addressed as well as a binary classification problem or a multi-class classification problem. In our study we tackled the problem as a ternary classification one, considering AD patients, CN patients and MCI patients. We used data coming from the well know ADNI [1]. We selected the maximum possible number of patients belonging to a minimal search criterion, between AD, MCI and CN patients, resulting in approximately 1000 MRI labeled NIFTI brain volumes (.nii files). We developed a simple tool able to build different kind of datasets, controlling the kind of processing applied to each volume, together with the number of samples of each class involved into the dataset. Then, we ended up having different kind of datasets, each of them containing samples already processed. This has been done mainly to speed up the training procedure which otherwise would have been way longer. We can therefore, distinguish between balanced and unbalanced datasets, and among each different kind of processing utilized, i.e ”dataset$-64 \times 64 \times 32-$Sobel$-$balanced” tells us the samples the dataset contains have been processed by resizing the original volumes to be $64 \times 64 \times 32$ and a Sobel operator has been used as a filter; moreover the balanced datasets contain an equal number of samples (268) for each class (AD, MCI and CN), while for the unbalanced ones the number of samples is not equally spread among all the classes, namely, we have 268 AD samples, 379 MCI samples and 584 CN samples. Finally the dataset which seems to be the best for this kind of learning problem, is the ”dataset$-64 \times 64 \times 32-$LoG$-$balanced”, thus containing volumes composed by 32 slices of 64×64 low resolution images filtered with a LoG operator. As a good practice the dataset is then split into training, validation and test set. We performed an ad-hoc splitting procedure, we will call it S_n, which is the result of several tests. As we can see from Table 1, the testing accuracies for models which has been trained on a dataset split in a classic way seems to do not align with the training and validation accuracies; whenever the training and validation accuracy grows the test accuracy tends to diminish, which is contradictory. On the contrary, models trained with our split S_n (Table 2), where the test and train set is built out of much more samples shows a much better

Split	Train Acc	Val Acc	Test Acc
60-20-20	0.978	0.572	0.378
70-20-10	0.981	0.569	0.310
70-15-15	0.982	0.573	0.350
80-10-10	0.990	0.600	0.310

Table 1: Results obtained with some common splitting procedure

Split	Train Acc	Val Acc	Test Acc
S_1	0.967	0.581	0.348
S_2	0.976	0.550	0.370
S_3	0.973	0.550	0.377
S_4	0.975	0.591	0.377
S_5	0.980	0.595	0.389
S_6	0.988	0.610	0.410

Table 2: Split S_1 procedure: pick the first 100 samples for AD, MCI and CN class to build the test set, split the remaining samples according to a 60-40 criterion. Split S_2 and S_3 has been obtained similarly to S_1 but splitting the remaining samples according to a 70-30 criterion and a 80-20 criterion respectively. Split S_4, S_5 and S_6 has been obtained picking the first 100 samples for AD class, 150 for MCI and 200 for CN class and splitting the remaining samples according to a 60-40, 70-30, 80-20 criterion respectively, as we did for S_1, S_2 and S_3.

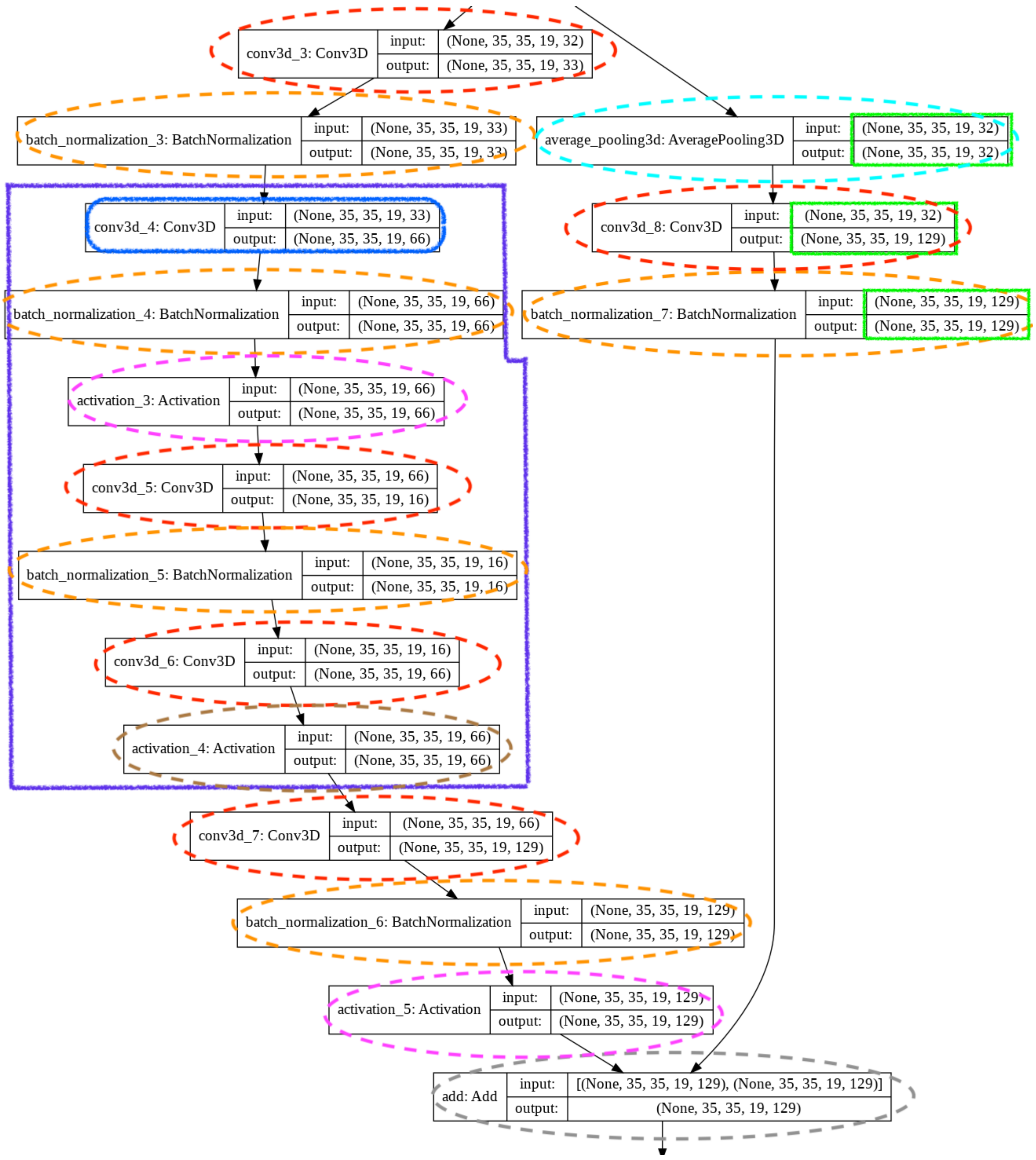
alignment between test, validation and training accuracies, hence the information we can retrieve from these metrics is certainly more coherent. It is also worth to point out, that validating models with few samples (this is true both for Table 1 and Table 2) seems letting the metrics to better describe the overall behaviour of the algorithm. Indeed, we can see how validation and test models' accuracies align much better wherever the validation set is obtained as a portion less or equal to 20% of the whole samples.

5 ResNeSt Models

Considering the complexity and heaviness of the proposed task, the most reasonable choice among any possible model is a residual one; residual models are well known to be very good candidate when tackling heavy learning problems, where an enormous amount of data are involved. In this way we can build wider and deeper architectures avoiding vanishing gradient problems and enforcing feature extraction power without using an enormous amount of learning units. Moreover the use of residual blocks and identity blocks, speed up significantly the train procedure. The most promising choice among the literature proposed residual models is a recent implementation [15] where the authors significantly modified the well know ResNeSt architecture, introducing the so called ***split-attention block***. The idea behind it is to make use of a grouped convolutional layer, exploiting the inception module principle, which divides the network into multiple branches with different convolutional kernels in order to extract features grouped by the belonging possibly different context. Instead of using literally different branches as channels, grouped convolution allows to split over different features internally, into the layer itself. Split-attention blocks enables featuremap attention across different featuremap groups. The Split-attention block is a computational unit, consisting of featuremap group and split-attention operations. This mechanism models the interdependencies of featuremap channels, which uses the global context information to selectively highlight or de-emphasize features. This attention mechanism is similar to attentional selection stage of human primary visual cortex, which finds the informative parts of objects. Human or animals perceive various visual patterns using the cortex in separate regions that respond to different and particular visual features. This strategy makes it easy to identify subtle but dominant differences of similar objects in the neural perception system. This is practically done, by using a softmax activation layer at end of each split-attention block, summing up the information obtained by each different feature group. In this case since the brain is a very sophisticated system, composed by many areas which are correlated in a very complex way, using the concept of feature separation may help significantly in simplifying the case study scenario. We started using a reduced and slightly modified version of the original ResNeSt network, then moving on a significantly modified version of it. The resulting architecture is then obtained by stacking two different convolutional blocks (conv1 and conv2) separated by a number of residual ones. The conv1 block does not implement any down sampling operation as seen in Figure 6 where the input-output data dimension remains constant (35, 35, 19), while the conv2 block implement a down sampling operation as seen in Figure 7 second block, where the input-output data dimension goes from (35, 35, 19) to (18, 18, 10) as highlighted in green. We will differentiate for further analysis between high and low dimensionality models, considering that both of them presents an initial ResNeSt-like architecture, where a $7 \times 7 \times 7$ convolution is realized by means of three $3 \times 3 \times 3$ convolutions.

```
def myResNest():
  input_im = Input(shape= (64, 64,32, 1))
  x = ZeroPadding3D(padding=(3,3, 3))(input_im)
  x = Conv3D(32, kernel_size=(3,3, 3), strides=(1,1,1),padding ='same')(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = Conv3D(64, kernel_size=(3, 3, 3), strides=(1,1, 1),padding ='same')(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = Conv3D(32, kernel_size=(3,3, 3), strides=(1,1, 1),padding ='same')(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = MaxPooling3D((3, 3,3), strides=(2,2, 2),padding='same')(x)
```

Code 1: Network tail



conv3d_3: Conv3D | input: (None, 35, 35, 19, 32) | output: (None, 35, 35, 19, 33)
batch_normalization_3: BatchNormalization | input: (None, 35, 35, 19, 33) | output: (None, 35, 35, 19, 33)
average_pooling3d: AveragePooling3D | input: (None, 35, 35, 19, 32) | output: (None, 35, 35, 19, 32)
conv3d_4: Conv3D | input: (None, 35, 35, 19, 33) | output: (None, 35, 35, 19, 66)
conv3d_8: Conv3D | input: (None, 35, 35, 19, 32) | output: (None, 35, 35, 19, 129)
batch_normalization_4: BatchNormalization | input: (None, 35, 35, 19, 66) | output: (None, 35, 35, 19, 66)
batch_normalization_7: BatchNormalization | input: (None, 35, 35, 19, 129) | output: (None, 35, 35, 19, 129)
activation_3: Activation | input: (None, 35, 35, 19, 66) | output: (None, 35, 35, 19, 66)
conv3d_5: Conv3D | input: (None, 35, 35, 19, 66) | output: (None, 35, 35, 19, 16)
batch_normalization_5: BatchNormalization | input: (None, 35, 35, 19, 16) | output: (None, 35, 35, 19, 16)
conv3d_6: Conv3D | input: (None, 35, 35, 19, 16) | output: (None, 35, 35, 19, 66)
activation_4: Activation | input: (None, 35, 35, 19, 66) | output: (None, 35, 35, 19, 66)
conv3d_7: Conv3D | input: (None, 35, 35, 19, 66) | output: (None, 35, 35, 19, 129)
batch_normalization_6: BatchNormalization | input: (None, 35, 35, 19, 129) | output: (None, 35, 35, 19, 129)
activation_5: Activation | input: (None, 35, 35, 19, 129) | output: (None, 35, 35, 19, 129)
add: Add | input: [(None, 35, 35, 19, 129), (None, 35, 35, 19, 129)] | output: (None, 35, 35, 19, 129)

Figure 6: Conv1 block: each layer has been highlighted with a different colour, namely convolutional layers in red, batch normalization layers in orange, pooling layers in light blue, grouped convolution layers in blue, ReLU activation layers in pink, softmax activation layers in brown and add layers in gray. The split-attention blocks is outlined by a purple polygon.)

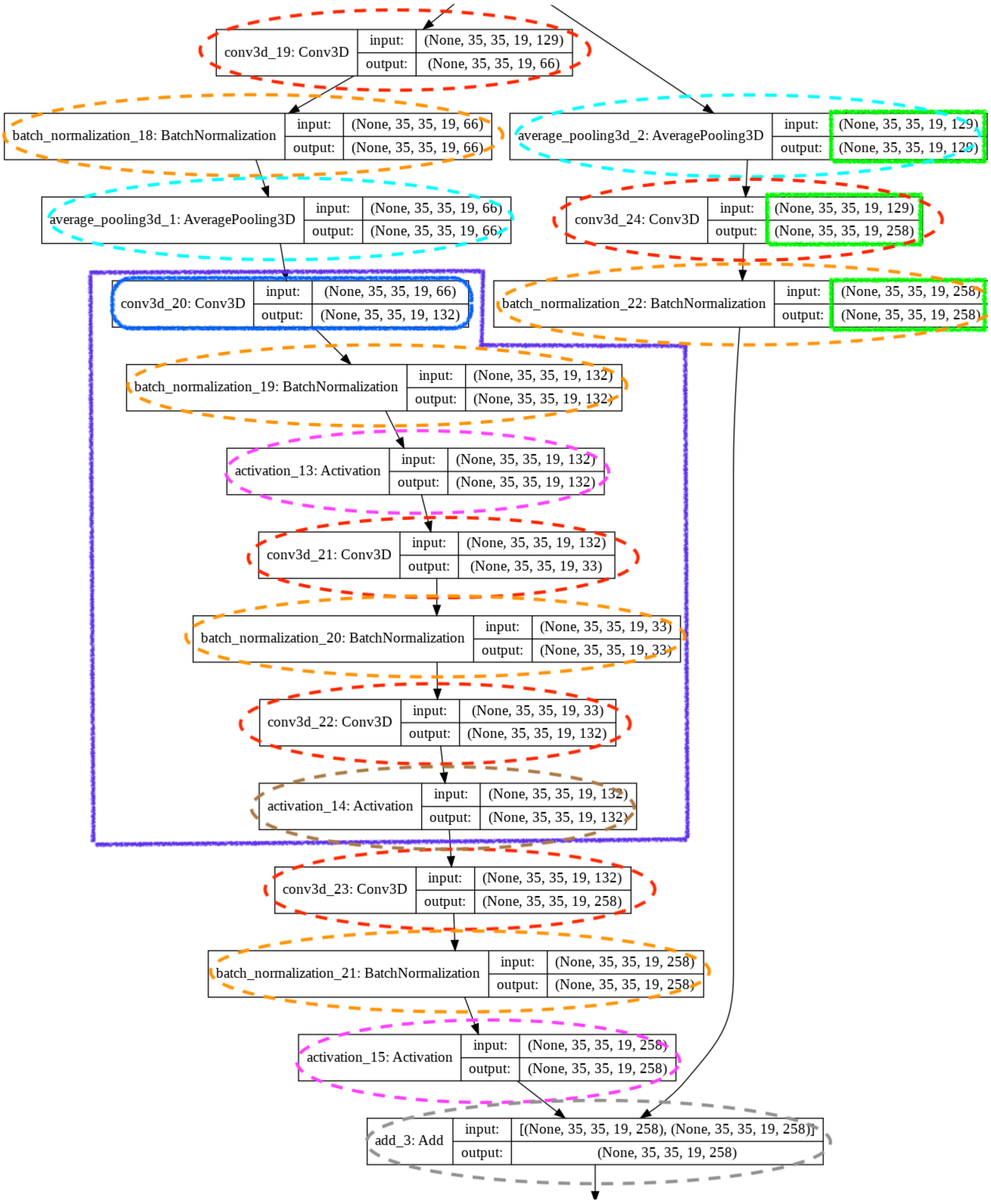

conv3d_19: Conv3D | input: (None, 35, 35, 19, 129) | output: (None, 35, 35, 19, 66)
batch_normalization_18: BatchNormalization | input: (None, 35, 35, 19, 66) | output: (None, 35, 35, 19, 66)
average_pooling3d_2: AveragePooling3D | input: (None, 35, 35, 19, 129) | output: (None, 35, 35, 19, 129)
average_pooling3d_1: AveragePooling3D | input: (None, 35, 35, 19, 66) | output: (None, 35, 35, 19, 66)
conv3d_24: Conv3D | input: (None, 35, 35, 19, 129) | output: (None, 35, 35, 19, 258)
conv3d_20: Conv3D | input: (None, 35, 35, 19, 66) | output: (None, 35, 35, 19, 132)
batch_normalization_22: BatchNormalization | input: (None, 35, 35, 19, 258) | output: (None, 35, 35, 19, 258)
batch_normalization_19: BatchNormalization | input: (None, 35, 35, 19, 132) | output: (None, 35, 35, 19, 132)
activation_13: Activation | input: (None, 35, 35, 19, 132) | output: (None, 35, 35, 19, 132)
conv3d_21: Conv3D | input: (None, 35, 35, 19, 132) | output: (None, 35, 35, 19, 33)
batch_normalization_20: BatchNormalization | input: (None, 35, 35, 19, 33) | output: (None, 35, 35, 19, 33)
conv3d_22: Conv3D | input: (None, 35, 35, 19, 33) | output: (None, 35, 35, 19, 132)
activation_14: Activation | input: (None, 35, 35, 19, 132) | output: (None, 35, 35, 19, 132)
conv3d_23: Conv3D | input: (None, 35, 35, 19, 132) | output: (None, 35, 35, 19, 258)
batch_normalization_21: BatchNormalization | input: (None, 35, 35, 19, 258) | output: (None, 35, 35, 19, 258)
activation_15: Activation | input: (None, 35, 35, 19, 258) | output: (None, 35, 35, 19, 258)
add_3: Add | input: [(None, 35, 35, 19, 258), (None, 35, 35, 19, 258)] | output: (None, 35, 35, 19, 258)

Figure 7: Conv2 block: each layer has been highlighted with a different colour with the same criterion we followed for conv1

5.0.1 High Dimensionality Models

This class of models are obtained considering the original model proposed by [15] tuned for our problem. The initial tuning procedure consists of resizing the original network, since it was used to tackle the well know ImageNet classification problem [7]. ImageNet [4] is an image database organized according to the WordNet hierarchy (currently only the nouns). Wordnet [14] is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept, in which each node of the hierarchy is depicted by hundreds and thousands of images. The resizing procedure consists in reducing the number of kernels in each layer, reducing the number of layers in each block and reducing the number of blocks used to build the final architecture. We started training this class of models with high resolution images soon understating that a resolution downgrade was needed. The model reported here has been trained with the dataset containing $64\times64\times32$ volumes already processed with the LoG filter. Initially we observed a prominent overfitting behaviour. In order to address this issue, we moved on tuning the hyper-parameters of the network, like the initial learning rate set at 0.00001, the learning policy as an exponential decay optimized adaptively (Adam optimizer), the SGD loss function, the dropout rate set as 0.5 and the number of units composing the learning head of the network aiming at balancing the model ”learning power”. We started with very poor results using networks with dozen of millions of parameters trying to solve an high-resolution 3D, very heavy learning problem, and ended up having a decent behaviour with a network having just 789,186 parameters, 781,724 trainable and 7,462 non-trainable solving a lower resolution, still heavy learning problem. Next we provide a code snippet of such a model, together with the identity block, the conv1 and conv2 block.

```
def FullModel():
  ...
  NETWORK TAIL
  ...
  x = res_conv1(x, filters=(f1, f2))
  x = res_identity(x, filters=(f1, f2))
  x = res_identity(x, filters=(f1, f2))
  x = res_conv2(x, filters=(2*f1, 2*f2))
  x = res_identity(x, filters=(2*f1, 2*f2))
  x = res_identity(x, filters=(2*f1, 2*f2))
  x = res_identity(x, filters=(2*f1, 2*f2))
  x = GlobalAveragePooling3D()(x)
  x = Dense(f1, activation='relu')(x)
  x = layers.Dropout(0.5)(x)
  x = Dense(3, activation= 'softmax')(x)
```

Code 2: Full residual model

```python
def full_res_identity(x, filters):
  x_skip = x
  f1, f2 = filters
  x = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1,1),groups=3)(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(2*f1), kernel_size=(3,3,3), strides=(1,1,1),
                                    padding='same',groups=3)(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = Conv3D(int(f1/2), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(2*f1), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.softmax)(x)
  x = Conv3D(f2, kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Add()([x, x_skip])
  x = Activation(activations.relu)(x)
```

Code 3: Full residual block

```python
def full_res_conv1(x, filters):
  x_skip = x
  f1, f2 = filters
  x = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(2*f1), kernel_size=(3,3,3), strides=(1,1,1),
                                    padding='same',groups=3)(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = Conv3D(int(f1/2), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(2*f1), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = Activation(activations.softmax)(x)
  x = Conv3D(f2, kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x_skip = AveragePooling3D(pool_size=(1,1,1), strides=(1,1,1),
                                        padding='valid')(x_skip)
  x_skip = Conv3D(f2, kernel_size=(1,1,1), strides=(1,1,1))(x_skip)
  x_skip = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x_skip)
  x = Add()([x, x_skip])
  x = Activation(activations.relu)(x)
```

Code 4: Full conv1 block

```python
def full_res_conv2(x, filters):
  x_skip = x
  f1, f2 = filters
  x = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = AveragePooling3D(pool_size=(3,3,3), strides=(1,1,1), padding='same')(x)
  x = Conv3D(int(2*f1), kernel_size=(3,3,3), strides=(1,1,1),
                                    padding='same',groups=3)(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = Conv3D(int(f1/2), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(2*f1), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = Activation(activations.softmax)(x)
  x = Conv3D(f2, kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x_skip = AveragePooling3D(pool_size=(1,1,1), strides=(1,1,1),
                                        padding='valid')(x_skip)
  x_skip = Conv3D(f2, kernel_size=(1,1,1), strides=(1,1,1))(x_skip)
  x_skip = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x_skip)
  x = Add()([x, x_skip])
  x = Activation(activations.relu)(x)
```

Code 5: Full conv2 block

5.0.2 Low Dimensionality Models

The final model we obtained by further simplifying the original ResNeSt architecture, contains 226,233 parameters, 223,323 trainable and 2,910 non trainable. Next we provide some code snippets of the general structure of a reduced model and all the layers we need to build it:

```python
def ReducedModel():
  ...
  NETWORK TAIL
  ...
  x = res_conv1(x, filters=f1)
  x = res_identity(x, filters=f1)
  x = res_identity(x, filters=f1)
  x = res_conv2(x, filters=2*f1)
  x = res_identity(x, filters=2*f1)
  x = res_identity(x, filters=2*f1)
  x = res_identity(x, filters=2*f1)
  x = GlobalAveragePooling3D()(x)
  x = Dense(f1, activation='relu')(x)
  x = layers.Dropout(0.5)(x)
  x = Dense(3, activation= 'softmax')(x)
```

Code 6: Reduced residual model

```
def reduced_res_identity(x, filters):
  x_skip = x
  f1 = filters
  x = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1,1),groups=3)(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(f1), kernel_size=(3,3,3), strides=(1,1,1),
                                    padding='same',groups=3)(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = Conv3D(int(f1/2), kernel_size=(1,1,1), strides=(1, 1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(f1), kernel_size=(1,1,1), strides=(1, 1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.softmax)(x)
  x = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Add()([x, x_skip])
```

Code 7: Reduced residual block

```
def reduced_res_conv1(x, filters):
  x_skip = x
  f1 = filters
  x = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1, 1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(f1), kernel_size=(3,3,3), strides=(1,1,1),
                                    padding='same',groups=3)(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = Conv3D(int(f1/2), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(f1), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = Activation(activations.softmax)(x)
  x_skip = AveragePooling3D(pool_size=(1,1,1), strides=(1,1,1),
                                            padding='same')(x_skip)
  x_skip = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1,1))(x_skip)
  x_skip = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x_skip)
  x = Add()([x, x_skip])
```

Code 8: Residual conv1 block

```
def reduced_res_conv2(x, filters):
  x_skip = x
  f1 = filters
  x = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1, 1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = AveragePooling3D(pool_size=(3,3,3), strides=(2,2,2), padding='same')(x)
  x = Conv3D(int(f1), kernel_size=(3,3,3), strides=(1,1,1),
                                    padding='same',groups=3)(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Activation(activations.relu)(x)
  x = Conv3D(int(f1/2), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x)
  x = Conv3D(int(f1), kernel_size=(1,1,1), strides=(1,1,1))(x)
  x = Activation(activations.softmax)(x)
  x_skip = AveragePooling3D(pool_size=(1,1,1), strides=(2,2,2),
                                            padding='same')(x_skip)
  x_skip = Conv3D(f1, kernel_size=(1,1,1), strides=(1,1,1))(x_skip)
  x_skip = BatchNormalization(epsilon=1e-05,momentum= 0.1)(x_skip)
  x = Add()([x, x_skip])
```

Code 9: Residual conv2 block

This model is by far the simplest one, resulting in faster training and better generalization learning power. We kept all the hyper-parameter we set for high dimensionality models, thus the initial learning rate set at 0.00001, the learning policy as an exponential decay optimized adaptively (Adam optimizer), the SGD loss function and the dropout rate set as 0.5. Nevertheless, in order to achieve significant improvements we needed to change the overall architecture, mainly removing some processing throughout the network, such as activation's layers both at the beginning and the end of each residual and convolutional block while adding one more group to the grouped convolution layer, thus realizing a wider and shorter network.

6 Results

Since we were using a data driven learning techniques, we initially thought that considering the maximum number of possible samples with the highest possible data resolution, could have resulted in better results. Therefore we started using very complex models, with dozen of layers resulting in millions of parameters, training on enormous high resolution datasets (thousands of 3D $256 \times 256 \times 48$ brain volume samples). Soon we found out that, not only this kind of approach is unsustainable in terms of resources needed (GPU memory, training time, tuning of a huge number of parameters), but also it leads to no results, given that the network tends to overfit, without any ability to generalize over such a complex scenario. The resulting immediate conclusion is that instead of over complicate the original, still complex, problem, we needed to simplify it. Then we started from resizing and processing the original high resolution data into simpler but still representative data, and proceeded reducing the complexity of the network. Unexpectedly we ended up with a very simple model, very small data, and pretty good results. It is worth to point out that the processing we applied is still very easy, it is mainly about down sampling the data resolution, applying a filter and eventually normalize. Realistically, this is somethings which could be done even by a non expert programmer, without any need of particular resources, or knowledge.

Next we display the results obtained, first with high dimensionality models and later with low-dimensionality models. The first model inspected has the structure introduced in Section 5.0.1 where $n = 32$, $m = 64$ and we have used only layer1 and layer2 with filters ($f1 = 32$, $f2 = 128$) and ($f1 = 64$, $f2 = 256$), respectively. This model reaches a global accuracy around 0.399. The overall behaviour is well depicted by the accuracy plot (Figure 8) showing how the network keep learning well until half of the training procedure, ending up in stabilizing about 0.61 validation accuracy. The confusion matrix obtained out of the test procedure (Figure 8) shows how the model better classify MCI samples while fails in classifying AD ones. The oscillatory behaviour is due to the high dropout rate (0.5) while the slow slope is due to the learning rate optimization policy. Here we used a pretty small learning rate, around 0.00001 with an exponential decay policy.

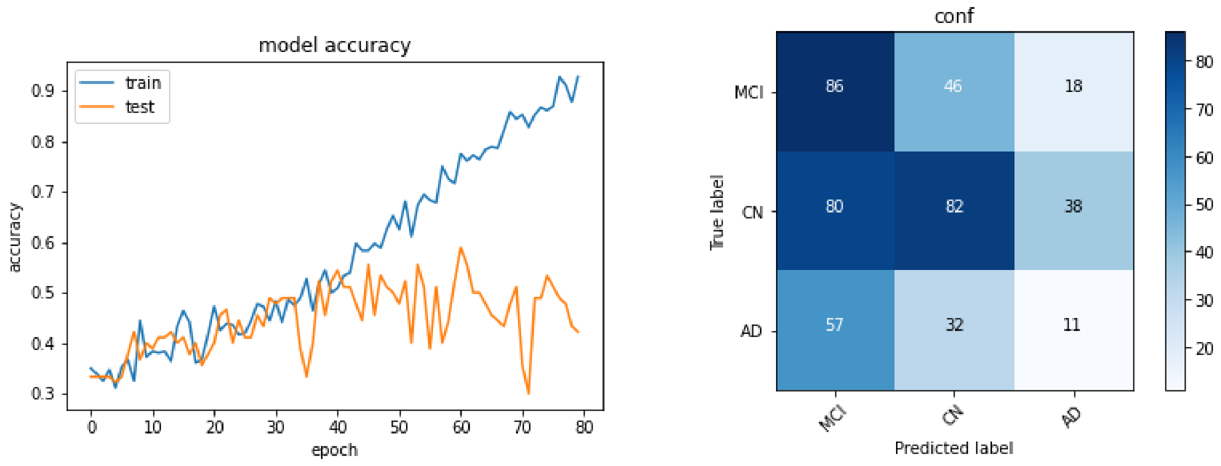


Figure 8: Accuracy plot/confusion matrix: best high resolution model

Next we have displayed the results obtained by training partially our version of a low dimensionality model (5.0.2) were we used only the first two layers with $f1 = f2 = 30$ and in this setting $n = 15$ and $m = 30$. Looking at the validation behaviour (Figure 9), which is constantly following the training one, we can see in fact how the model seems to be able to generalize pretty well. Also in this case we maintained the same high dropout rate (k = 0.5), which helps a lot in avoiding early overfit. The confusion matrix (Figure 9) shows how the network learn very well how to classify MCI samples, with a 0.64 accuracy rate, while still have difficulties classifying CN and AD samples. With an overall 0.380 accuracy, and also considering the confusion matrix information, this has represented our first good result; from now on we will display the results obtained by adjusting this same model, thus changing the initial learning rate value, dropout rates and the number of learning units (fully connected layers' units). It is worth to point out how this kind of reduced problem takes way less time and resources, to converge, maintaining decent average performances.

The next plot depict the previous model brought at convergence, thus we increased dramatically the number of epochs. Here we also changed a bit the hyper-parameters such that the model converges

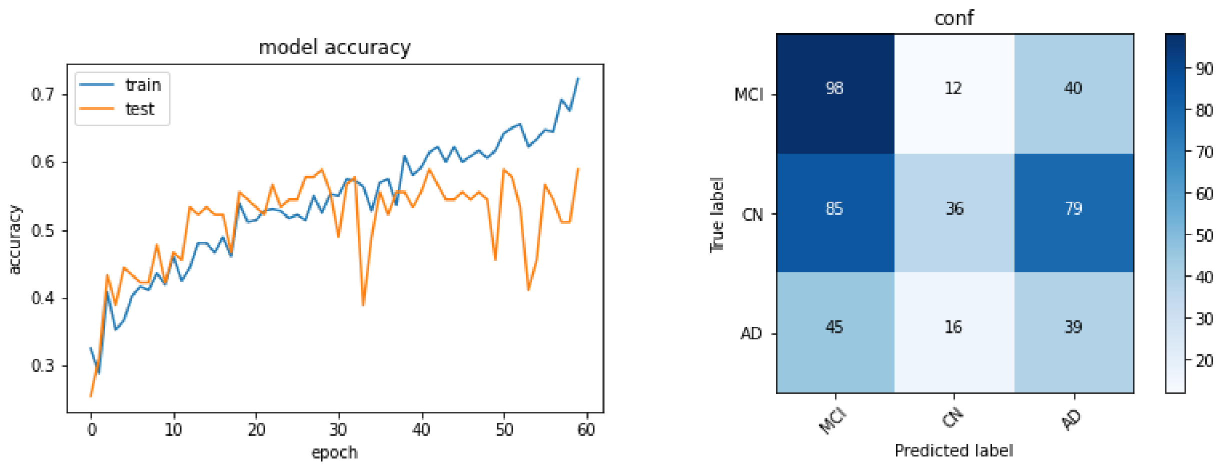


Figure 9: Accuracy plot/confusion matrix: best low resolution model with partial training

quicker, without caring too much to the resulting accuracy curve shape, trivially depicting a strong overfitting towards the end of the training (Figure 10). It is indeed true, that even with a major number of epochs, the training procedure was really fast, about 45 minutes. Nevertheless we can see how this kind of configuration seems to splendidly learn to classify MCI samples while giving up the ability of classifying AD samples. We reach a good global accuracy of 0.408 and a very good MCI classification accuracy of 0.75 (Figure 10).

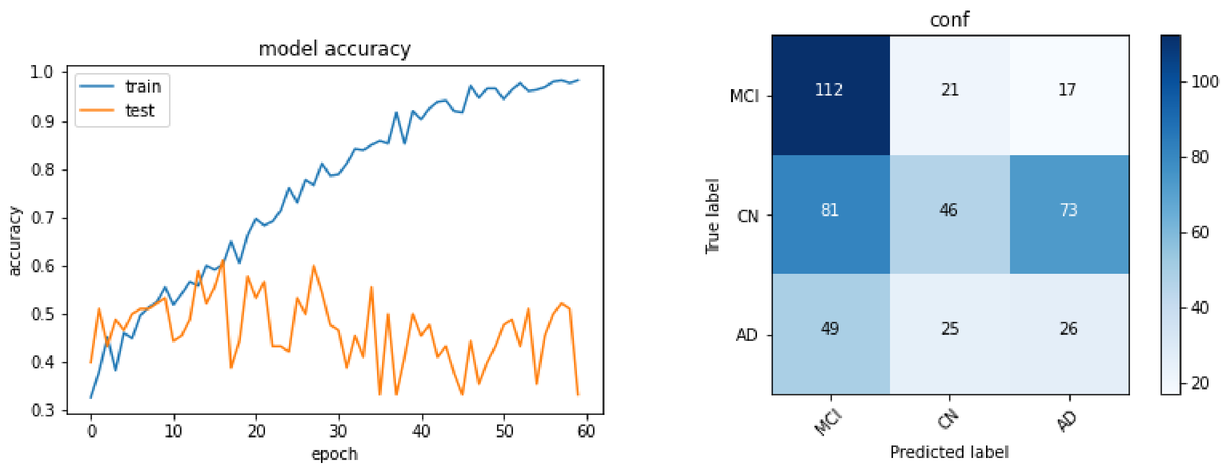


Figure 10: Accuracy plot/confusion matrix: best low resolution model with full training

After this experiment we figured out that probably a higher learning rate, leading to a fast convergence could have been a good point to try out. This could seem wrong or counter intuitive but speeding things up may result in "fortunate" configuration, letting in fact the network hit a good shape. On this track, re-training the previous model with further tuning, like the initial learning rate and trying different adaptive policies, we ended up with our best score. First of all it is clear how at a certain point the model overfits (Figure 11); this is kind of intentional, since we ended up using a pretty high learning rate, hoping for the network hitting a good spot, without caring too much about the consequent trivial overfit. On the other side we also, successfully trained a model, on a kind of complex problem, in less than a hour, obtaining an average good result. As it has been proven that residual models of this kind cut off considerably the training time, the intuition is to sacrifice the overall behaviour in order to quickly reach a good network configuration. The overall accuracy reached is 0.411 with a 0.41 accuracy rate in predicting AD samples (Figure 11). Since the main scope of the research is to learn how to classify AD over MCI and CN, we consider this a valid starting point for further analysis. It is still far from being good, but as previously mentioned, it was not meant to reach good performances yet, since we are working with very raw data, and simple models. Overall we found out a good track on which keep working.

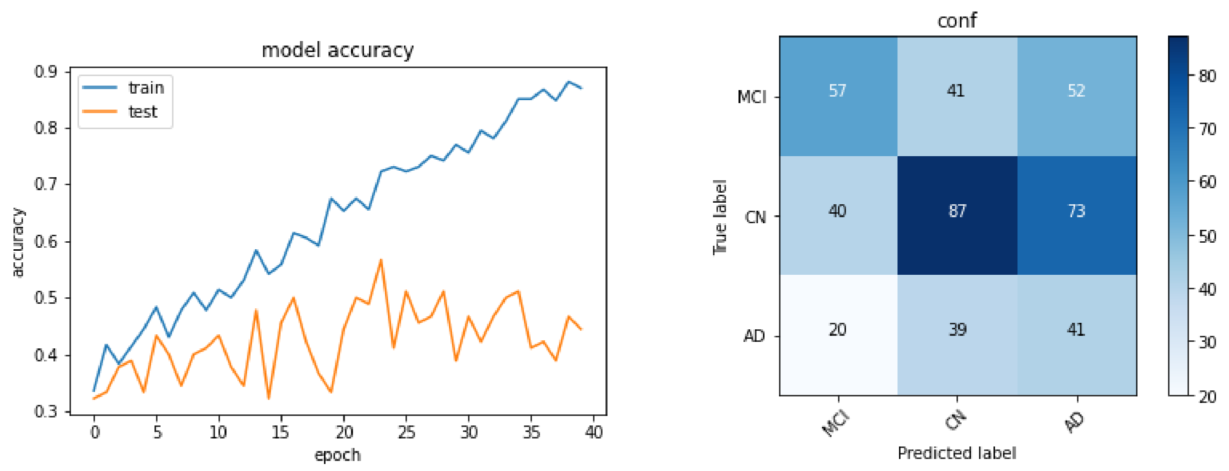


Figure 11: Accuracy plot/confusion matrix: best low resolution model, tuned

7 Conclusions and Further Developments

The proposed simplification of the well known ternary classification problem (AD, MRI and CN) for AD detection, addressed with a simplified version of the new ResNeSt-like models, seems to be a new promising way to tackle this task. We are sure that using 3D models for this kind of problem is a very good alternative to 2D CNN and 2.5D CNN classical solutions, where very complex ad-hoc preprocessing pipeline need to be applied before feeding the data to complex networks, or sometimes combining different simple network working together. The reasoning behind this consideration is how this kind of residual models beats models relying on brute force. In our case study, poorly processing the data, respectively just down sampling and filtering it with classical derivative operators, we have been able to still get good results. Moreover the introduced reduced problem impressively cut off a training procedure (in certain settings we are speaking about cutting 10 times the average training time), and still being able to get average good accuracies. Thus, it is reasonable considering the possibility of further studying which kind of processing could be applied, maybe starting from acquisition step, using a different MRI weighting, heading to selecting different planes for the analysis or using a combination of them, finally trying a different filtering procedure, building ad-hoc operators or finding a combination of the known ones. The major problems we encountered are mainly due to overfitting and poor generalization power. Then the immediate next step could be using any sort of data augmentation, like cropping or brightening. We did not insert this step since it is usually needed a proven working model before trying to over complicate the training procedure, risking otherwise to vainly, slow down things significantly. It is also true that, combining any kind of complex run-time augmentation procedure, which is supposed to slow down things, could be compensated by the velocity of this kind of model, which converges even quicker if working in the proposed reduced fashion. Recalling the aim of this work, which is to prove that we can reliably use a ResNeSt-like model for the given task, expecting to exploit its ability to considerably speed up the training, we also proved that a consistent reduction of the data resolution, respectively from $256 \times 256 \times 48$ to $64 \times 64 \times 32$, leads to use a much simpler model resulting in a much much faster training, still hitting unexpected good performances. It is worth considering the possibility of using this model in an extended diagnosis procedure. We could use the outcome of the classification task as one of the parameter for the diagnosis, or even building a much more informative data for an extended classification problem using then also information about sex, age, weight and a patient minimal assessment.

References

[1] *ADNI: Alzheimer's Disease Neuroimaging Initiative.* URL: `http://adni.loni.usc.edu/about/`.

[2] Hao Chen et al. "Voxresnet: Deep voxelwise residual networks for volumetric brain segmentation". In: *arXiv preprint arXiv:1608.05895* (2016).

[3] C. Fischbach-Boulanger et al. "T1- or T2-weighted magnetic resonance imaging: what is the best choice to evaluate atrophy of the hippocampus?" In: *European Journal of Neurology* 25.5 (2018), pp. 775–781. DOI: `https://doi.org/10.1111/ene.13601`.

[4] *Imagenet.* URL: `https://www.image-net.org/`.

[5] Alexander Khvostikov et al. "3D CNN-based classification using sMRI and MD-DTI images for Alzheimer disease studies". In: *arXiv preprint arXiv:1801.05968* (2018).

[6] Sergey Korolev et al. "Residual and plain convolutional neural networks for 3D brain MRI classification". In: *2017 IEEE 14th international symposium on biomedical imaging (ISBI 2017)*. IEEE. 2017, pp. 835–838.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[8] *L'INTELLIGENZA ARTIFICIALE NELL'IMAGING MEDICO.* URL: `https://www.pmf-research.eu/lintelligenza-artificiale-nellimaging-medico/`.

[9] *Mild Cognitive Impairment vs. Alzheimer's Disease.* URL: `https://www.verywellhealth.com/mild-cognitive-impairment-and-alzheimers-disease-98561`.

[10] J Neelaveni and MS Geetha Devasana. "Alzheimer disease prediction using machine learning algorithms". In: *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE. 2020, pp. 101–104.

[11] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[12] *What Happens to the Brain in Alzheimer’s Disease?* URL: `https://www.nia.nih.gov/health/what-happens-brain-alzheimers-disease`.

[13] *What Is Dementia?* URL: `https://www.alz.org/alzheimers-dementia/what-is-dementia`.

[14] *Wordnet.* URL: `https://wordnet.princeton.edu/`.

[15] Hang Zhang et al. “Resnest: Split-attention networks”. In: *arXiv preprint arXiv:2004.08955* (2020).