# Transfer learning for Alzheimer's disease diagnosis

Michela Proietti 1739846

February 2021

**Abstract**

In this project, we have used transfer learning in order to diagnose Alzheimer's disease performing binary classification on MR images. In particular, we have done several experiments in which different pretrained networks have been used, namely VGG16 and Inception V4. In this report, we show the results obtained and we make comparisons between the different algorithms that have been used. Moreover, we show the change in performance due to the use of entropy in order to select the most informative images for training.

## 1 Introduction

Alzheimer's Disease (AD) is a neurodegenerative disease causing dementia in elderly population that is expected to become more and more common in the future. For this reason, it is very important to use machine learning in order to achieve an early diagnosis of AD, and it has already been shown that in some cases machine learning algorithms can predict AD better that clinicians. However, there do not exist large datasets of medical images, therefore if we train a deep network from scratch, we will probably have to face the problem of overfitting, because our network will explain so well the training data that it won't be able to correctly classify new images. In this project, we show the potential of transfer learning, that allows us to use a CNN that has been trained on a large-scale dataset that belongs to a different field, such as ImageNet, and just fine-tune the last fully-connected layers of the network on the dataset of interest. More specifically, we have first reproduced the experiments presented in paper [1] using VGG16 and Inception V4, and then we have done further modifications in order to improve the obtained results.

## 2 Preliminaries

All the models that have been used are neural networks that are made up by convolutional layers. Convolution is widely used in image analysis, because it is an operation that allows to consider neighbourhoods, thus letting us identify properties of the image that are not intuitable through a pixel-by-pixel analysis. Convolution is usually followed by an activation function in order to introduce a non-linearity and it can be also followed by a pooling layer.

## 2.1 Activation functions

Activation functions have a major role in how the network learns, because they allow to filter the information, keeping the important parts and neglecting the noise that is present in every type of data. The activation function is therefore able to improve the performance of a learning algorithm, but it is important to choose the type of function that is more suitable to our problem. In this project, we have used two types of activation functions:

– ReLU (Rectified Linear Unit) activation function performs a partition on the x-axis, because if the output is smaller than 0, then the information is completely ignored, while if it is greater than 0, the value is returned as it is. The advantage of using this activation function is that its derivative is very simple, and therefore it allows to have very quick computations.

$$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$$

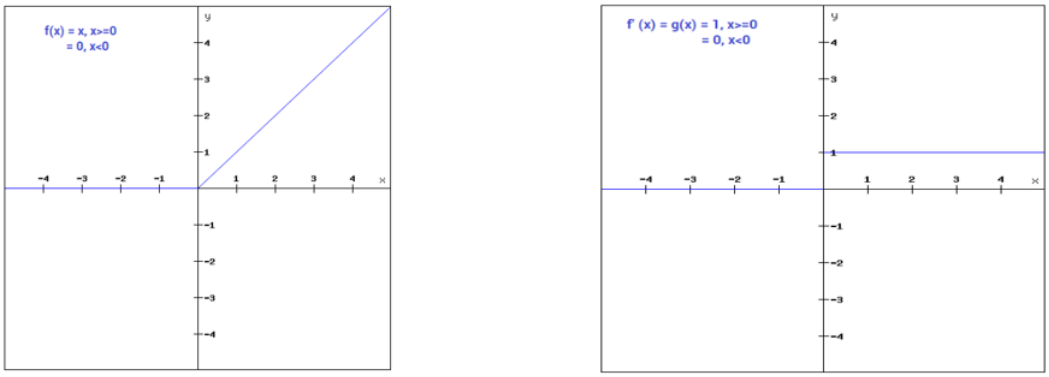
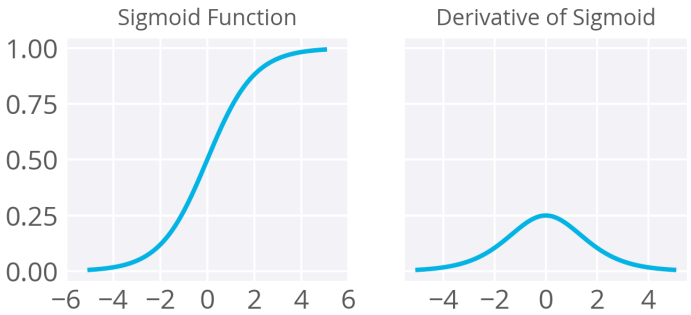



**Figure 1:** ReLU activation function

– The sigmoid activation function, instead, takes values between 0 and 1 and it is the most used in output layers in binary classification problems.




$$f(x) = \frac{1}{1 + e^{-x}}$$

**Figure 2:** Sigmoid activation function

## 2.2 Pooling

Pooling is used in order to progressively reduce the size of the feature space, to decrease the number of trainable parameters and therefore even the amount of computations that the network should perform. There are two main types of pooling:

1. Max pooling, in which to each image we apply a filter that keeps just the maximum element of each considered area.

2. Average pooling, that applies a filter that takes the average of all the elements in the considered area.

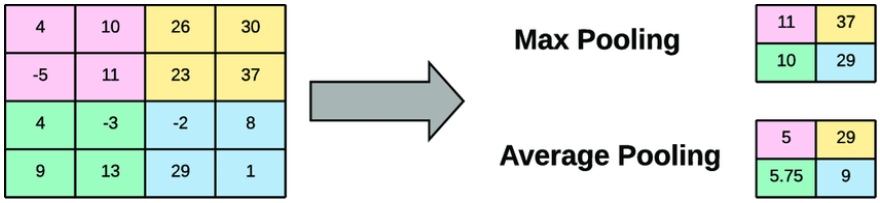



**Figure 3:** Max and average pooling examples

## 2.3 Cost function

Since we are trying to solve a binary classification problem, in all the models that have been considered, we have used binary cross-entropy loss. Cross-entropy loss is defined as follows:

$$CE = -\sum_{i=1}^{C} y_i \log \hat{y}_i$$

where $y_i$ is the groundtruth, $\hat{y}_i$ is the output of our model and C is the number of classes. In our case, $C = 2$ and therefore we can write:

$$CE = -y_1 \log \hat{y}_1 - (1 - y_1) \log (1 - \hat{y}_1)$$

## 2.4 Optimizers

In these experiments, different types of optimizers have been used, but all of them belong to the class of gradient descent methods, which means that we try to solve our problem of minimizing the loss function by moving in the direction of the negative gradient.

– SDG (Stochastic Gradient Descent) is a method in which at each iteration we select one sample in order to compute the gradient to get the next direction along which to move. For this reason, the path to reach the minimum is usually noisier, but the important thing is that we manage to get to the minimum in a possibly short training time.

$$\theta_{t+1} = \theta_t - \eta \nabla_i J(\theta)$$

– RMSProp is defined through the following equations:

$$s_t = \beta * s_{t-1} - (1 - \beta) * \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta) \oslash \sqrt{s_t + \epsilon}$$

RMSProp uses exponential decay in order to manage the size of the vector s, that is used to update the parameters of the network. RMSProp decreases the contribution of older gradients at each step, so that the magnitude of s does not become so large that it prevents learning. As a result of this exponential decay, the accumulated gradients in s are focused on recent gradients as a opposed to all previous gradients and the hyperparameter $\beta$, known as decay rate, controls how much the adaptive learning rate is focused on more recent gradients.

– Adam (adaptive momentum estimation) is an optimization method that exploits the advantages of both momentum and RMSProp, because it allows to get quickly close to the optimum while simultaneously reducing oscillations.

$$v_t = \beta_1 * v_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * {g_t}^2$$

$$\Delta\theta_t = -\eta * \frac{v_t}{\sqrt{s_t}} * g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

The first equation refers to momentum, while the second one refers to RMSProp, so by changing the value of $\beta_1$ and $\beta_2$ we can choose which one among the two we want to use more.

# 3 Methodology

As it has been said in the introduction, in this project we have used transfer learning and, in particular, we have used models that had been previously trained on Imagenet, by adding some final dense layers, in order to adapt the networks to our problem. In this section, we will show the architectures of the networks that have been used and how they have been modified.

## 3.1 VGG16

VGG16 is the first model that has been tested. It is made up by 16 trainable layers: 13 of them are convolutional layers, while the last 3 are fully connected layers. As we can see in image 4, each convolutional layer applies 3x3 kernels with stride 1 and padding same, which means that the output image has the same size of the input one, and each convolutional block is followed by a max pooling layer, that applies a 2x2 filter with stride 2. Finally, the first two dense layers are made up of 4096 nodes, while the size of the last one depends on

the application. In our experiments, we have kept the first 13 layers of the network up to the last max pooling layer, and we have added other Dense layers that have been trained on our dataset. Among these, the output layer is always made up by one single node with sigmoid activation function.

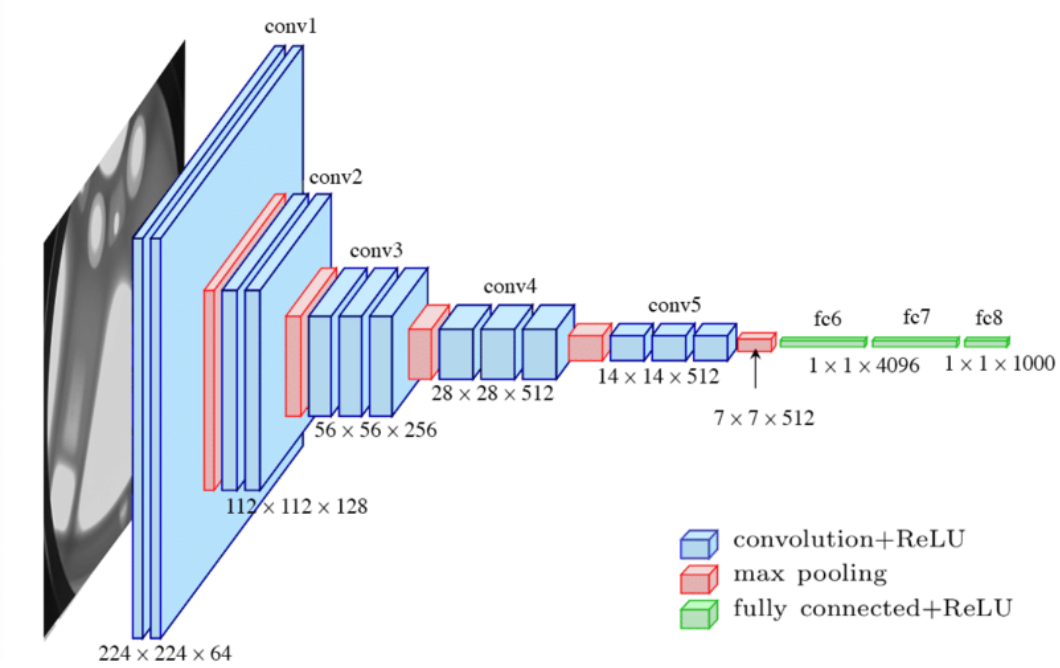



**Figure 4:** VGG16 architecture

## 3.2 Inception V4

An inception network is made up by several inception modules, that perform convolutions on an input with different sizes for the filters and max pooling is also performed. Then, the outputs are concatenated and sent to the next inception module. There exist different versions of inception networks, each of which has introduced some tricks in order to make convolution more efficient, for example factorizing convolutions with $n \times n$ filters to a combination of $1 \times n$ and $n \times 1$ convolutions. With respect to the previous versions, Inception V4 introduced specialized reduction blocks, that are used to change the width and height of the grid. As we can see in image 5, this network is much more complex than the previous one. The first part of the network, called stem, is made up by several convolutional layers with different kernels and some max pooling layers. Then, there is a alternate sequence of 3 inception blocks and 2 reduction blocks, while after the last inception block, there is an average pooling layer and dropout, that in our experiments has been set to 0.2. Finally, there is the output layer, that in our model is made up by one single node and has sigmoid activation function. The blocks represented on the left part of image 5 are the three inception blocks, while the ones represented in the upper-right part of the image are the reduction blocks that follow the first two inception modules.

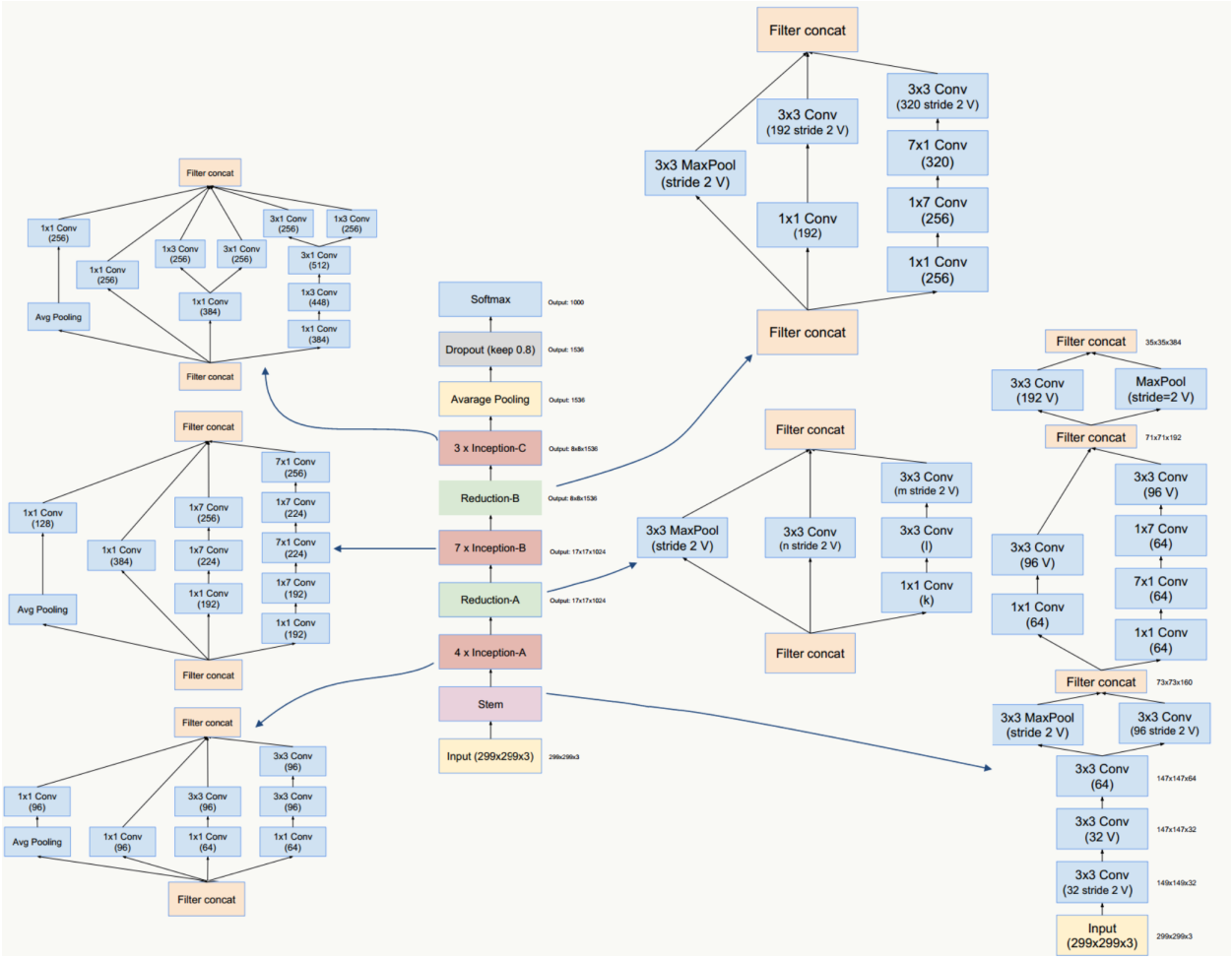



**Figure 5:** Inception V4 architecture

# 4 Dataset

In this project, we have used the same dataset that was used in the experiments presented in paper [1]. It is made up by MR brain cross-sectional images from AD and Healthy Control (HC) patients and it can be found at https://github.com/marciahon29/Ryerson_MRP/tree/master/MRI_32_Images. This dataset contains 32 images from the 3D scans of 200 subjects, 100 belonging to the AD group and the remaining to the HC group. Therefore, we have a total of 6400 images, equally distributed among the two classes, with an 80%-20% split between training and testing. Moreover, the results shown are obtained using cross-validation, in order to be sure that the reported values do not depend on a lucky distribution of the samples between train and test set. In figure 6 and figure 7, we show respectively one sample from the AD class and one sample for the HC class. The images that have been used in the paper's experiments are the most informative of each 3D scan, and they have been selected using entropy. Since just the preprocessed dataset was easily accessible to us, we first used it to reproduce the paper's results and then we have re-arranged it using entropy in order to improve the performance that we had obtained in the previous experiments.

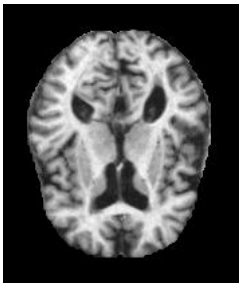


**Figure 6:** Sample from the AD class

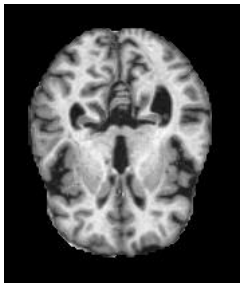


**Figure 7:** Sample from the HC class

More specifically, we have used the histogram associated to each image in the dataset to compute the associated entropy, and then we have used those images for training. In particular, we have used Shannon entropy, that measures the uncertainty of a random process and is computed as follows:

$$H = \frac{1}{\log N} \sum p_i \log p_i$$

where N is the total number of observed events and $p_i$ is the probability of event $i$.

# 5 Experiments and results

## 5.1 VGG16

First of all, we have tried to replicate the experiments done in paper [1]. Therefore, we have added to the convolutional part of the network one dense layer with 256 nodes and ReLU activation function, and the output layer with one node and sigmoid activation function. Moreover, as specified in the paper, we have set the batch size to 40 and the number of epochs to 100 and we have used a dropout of 0.5 on the first added fully-connected layer. The author of the paper also suggested using RMSProp as optimizer, since it let us have an adaptive learning rate that reduces oscillations during the training. This setup gave pretty good results, that are very similar to those showed in the paper. In this case, we have performed 5-fold cross validation and table 1 displays the values for the accuracy and the loss on the 5 obtained datasets. The mean accuracy is equal to 85.88% and it is slightly worse than the value reported in the paper.

| **Dataset** | **Test loss** | **Test accuracy** | **Single epoch's duration (s)** |
|---|---|---|---|
| 00 | 0.265 | 0.902 | 25 |
| 01 | 0.265 | 0.871 | 28 |
| 02 | 0.306 | 0.874 | 29 |
| 03 | 0.408 | 0.831 | 28 |
| 04 | 0.402 | 0.816 | 28 |

**Table 1:** Results obtained with VGG16 and paper's setup

In figure 8, we show the trend of the accuracy and loss on both the training and validation sets. As we can see, there is a very quick decrease of the loss on the training set and even the accuracy curve is quite smooth. On the contrary, for both the accuracy and loss on the validation set there are a lot of oscillations.

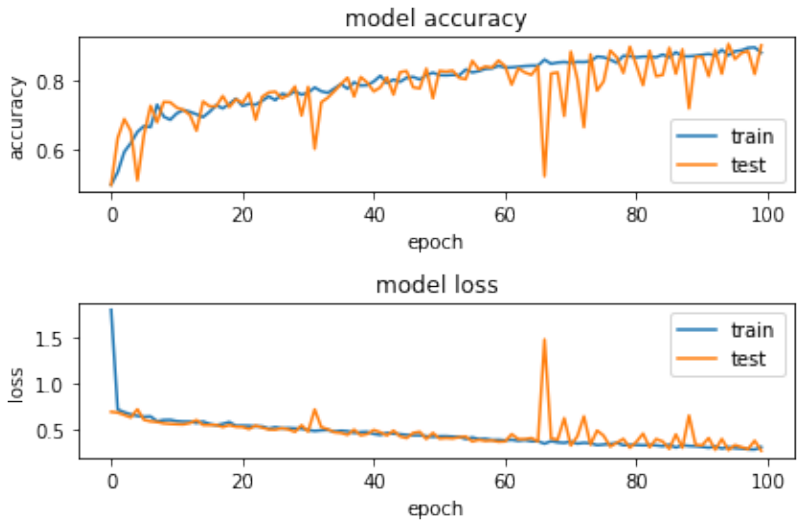



**Figure 8:** Training curves with the paper's setup

However, we have tried to modify the network by adding to the convolutional part of the original VGG16 two dense layers with 200 and 100 nodes and ReLU activation function, with dropout set to 0.4, and a final dense layer with one node and sigmoid activation function. Moreover, we have added batch normalization in order to speed up the training, in fact as we will see from the results, the duration of a single epoch is smaller than in the previous experiments, except that for the first dataset. Finally, we have used Adam optimizer in place of RMSProp, because Adam let us exploit the advantages of both momentum and RMSProp, since it allows to get quickly close to the optimum while simultaneously reducing oscillations. In this case, we have set the batch size to 64 and just 50 epochs were needed in order to obtain very impressive results, that are higher than the ones given by the paper's best model. Table 2 shows the accuracy values and the loss obtained for the 5 different datasets. With this new setup, the mean accuracy is 99.04% and it is much better than before.

| **Dataset** | **Test loss** | **Test accuracy** | **Single epoch's duration (s)** |
|---|---|---|---|
| 00 | 0.031 | 0.988 | 47 |
| 01 | 0.026 | 0.989 | 25 |
| 02 | 0.014 | 0.997 | 13 |
| 03 | 0.047 | 0.987 | 13 |
| 04 | 0.028 | 0.991 | 25 |

**Table 2:** Results obtained with VGG16 and the new setup

Image 9 shows the trends of the accuracy and the loss during training, and we can immediately notice that the loss and accuracy on the validation set have much less oscillations than with the previous setup of the network.

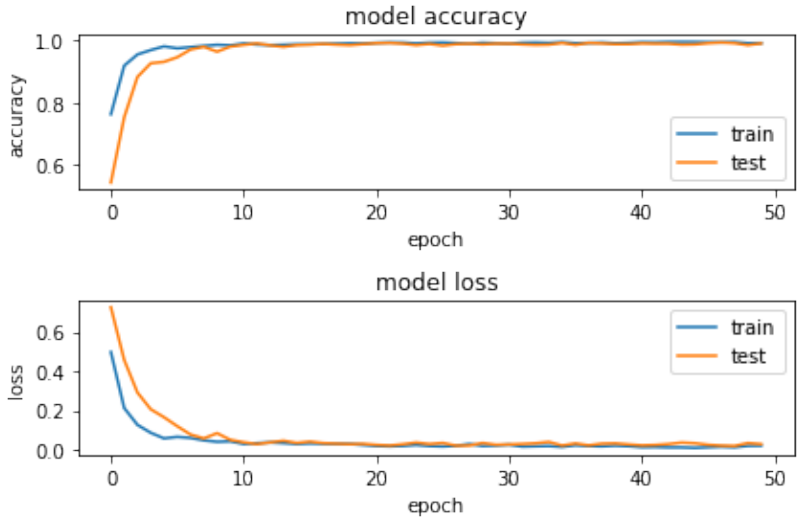



**Figure 9:** Training curves with the new setup

## 5.2 Inception V4

In our implementation of this network, we have kept exactly the same structure showed in the previous section, but we have added batch normalization after each convolutional layer and as suggested in paper [1], we have used a SDG optimizer and a learning rate of 0.0001. Going deeper into the code, we have implemented a function *conv2d_bn* which creates a convolutional layer with the number of filters, filter size, stride and padding specified as input and ReLU activation function, followed by batch normalization. Afterwards, we have created 5 functions that implement all the inception and reduction blocks and a function, called *stem_inception_reduction*, that creates the stem with all its convolutional layers, and then calls the previously mentioned functions for adding the inception and reduction modules. Finally, there is the *build_model* function, that calls *stem_inception_reduction* and then adds average pooling, dropout set to 0.2 and it flattens the output so that it can be used as input of the output layer. Then, in this function we load the weights of the pretrained Inception V4 and we compile the model, specifying SDG with momentum as optimizer and using binary cross-entropy loss.
Looking at table 3, it can be easily noticed that the results obtained with this network are far better than those obtained with the first version of VGG16 and the accuracy and loss values are slightly better also than the second version. In fact, the mean accuracy is 99.2% and even the loss on the validation set is lower than before. However, the training time is much longer, because the mean duration of one single epoch is 175 seconds, and since we train the network for 100 epochs, it means that the training lasts at least 5 hours using a GPU. As we can see from figure 10, the training curves have also many oscillations, even more than with the first version of VGG16.

| Dataset | Test loss | Test accuracy | Single epoch's duration (s) |
|---|---|---|---|
| 00 | 0.004 | 0.997 | 190 |
| 01 | 0.016 | 0.999 | 189 |
| 02 | 0.045 | 0.989 | 190 |
| 03 | 0.019 | 0.996 | 127 |
| 04 | 0.028 | 0.979 | 186 |

**Table 3:** Results obtained with InceptionV4

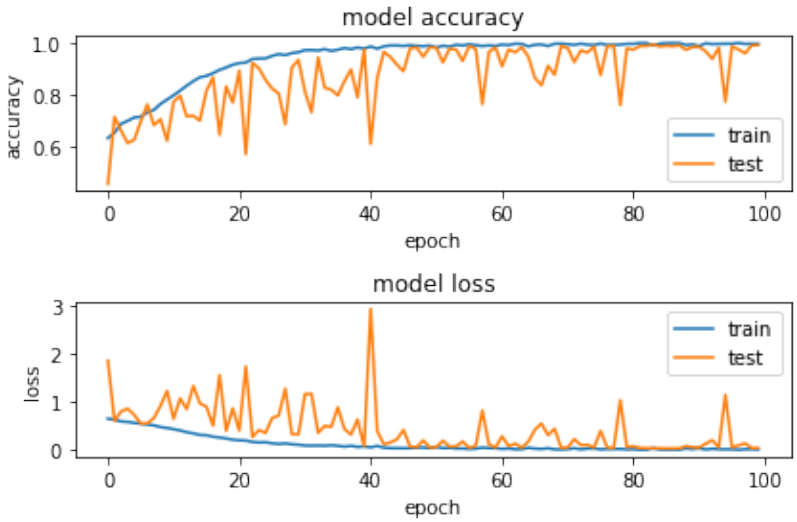



**Figure 10:** Training curves with Inception V4

## 5.3 Experiments on the entropy-based dataset

As explained in the previous section about the dataset that has been used, we have computed the entropy of histograms associated to all images in the dataset in order to get the most informative ones to be used for the training. Table 4, shows the results obtained with the 3 different networks that have been presented.

| Network | Test loss | Test accuracy | Single epoch's duration (s) |
|---|---|---|---|
| VGG16 (paper's setup) | 0.289 | 0.851 | 15 |
| VGG16 (new setup) | 0.037 | 0.990 | 26 |
| InceptionV4 | 0.034 | 0.990 | 187 |

**Table 4:** Results obtained using the entropy-based dataset

It is possible to notice that the results did not change considerably. This probably happens because the difference in entropy is not substantial, since the lowest entropy is equal to 4.69 while the highest entropy value is 6.91, because our dataset already contained the most informative images available. Moreover, for the first and the last model, there was also a slight decrease in accuracy. This is mainly caused by the fact that since the most

informative images are part of the training set, the validation set is made up by images that are more difficult to analyse, like the ones showed in figure 12, but if we had other images with higher entropy like the ones that were present in the validation sets of the previous versions of the dataset, these networks will surely be more robust and will have higher accuracy than the previous ones.

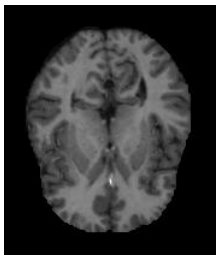


**Figure 11:** Least informative AD sample

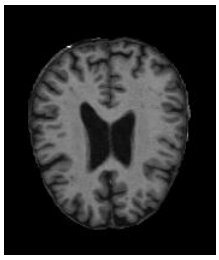


**Figure 12:** Least informative images

# 6 Conclusions

This project showed how it is possible to use transfer learning for image classification in cases in which there are few available training images, like it happens with medical images. In fact, even using networks that have been trained on completely different images, like the natural images that can be found in Imagenet, it is possible to obtain very high accuracy values on a different problem by just training few dense layers to adapt the network to the new dataset. All this is of fundamental importance when applied to the medical field, because it could make it possible to have very quick and accurate diagnosis of serious diseases, such as Alzheimer's disease.

# References


[1] Marcia Hon, Naimul Mefraz Khan, *Towards Alzheimer's Disease Classification through Transfer Learning* , 2017, avaiable at https://arxiv.org/pdf/1711.11117v1.pdf.

[2] *VGG16 architecture*, https://neurohive.io/en/popular-networks/vgg16/

[3] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, *Inception V4*, 2016, avaiable at https://arxiv.org/pdf/1602.07261v2.pdf