Facultad

De

Ciencias

APPLICATIONS OF DEEP LEARNING IN THE ANALYSIS OF MEDICAL IMAGES

(Aplicaciones de deep learning en análisis de imágenes médicas)

Trabajo de Fin de Grado

para acceder al

GRADO EN FÍSICA

Autor: Ana Ramírez Frías

Director: Ignacio Heredia Cacha

Co-Director: Lara Lloret Iglesias

Junio - 2018

ACKNOWLEDGEMENT

I would like to thank my parents for all their support throughout the years,
without them I couldn't have done this work.
Thanks to the rest of my family and my friends, who had encouraged me greatly
when I needed it the most.

Thanks to the Universidad de Cantabria for guiding me for four years,
and IFCA for its help and resources.
Thanks to Jose María Saiz Vega for his priceless help and patience.
And ultimately, thanks to Lara Lloret who gave me the opportunity to be part
of this project and encouraged me and helped me finish it.

RESUMEN

Hay muchas enfermedades que requieren el análisis de imágenes médicas para ser diagnosticadas. Históricamente este análisis ha sido llevado a cabo por expertos que en ocasiones no están de acuerdo entre ellos, y en algunos casos un diagnóstico rápido y adecuado puede llegar a salvar vidas. En los últimos años ha habido un desarrollo de técnicas de aprendizaje automático aplicadas en el campo de imágenes médicas. Automatizar el diagnóstico reduciría los tiempos de espera, mejoraría la efectividad y detectaría anomalías que habrían pasado desapercibidas en otro caso. El propósito de este TFG es estudiar la viabilidad de aplicar métodos de Deep Learning, en concreto redes neuronales convolucionales para la clasificación de imágenes, al análisis de imágenes médicas. Los datos utilizados se corresponden a un conjunto de radiografías de tórax proporcionadas por el *NIH Clinical Center* y etiquetadas de acuerdo a su diagnóstico. La meta de este proyecto es determinar si este tipo de detección asistida por ordenador ayudaría con la interpretación de hallazgos en imagen médica, reduciendo el tiempo que normalmente conlleva producir un diagnóstico y excediendo la efectividad de los radiólogos.

Después de describir el Deep Learning en profundidad, y de construir un modelo con redes neuronales convolucionales con una eficacia de aproximadamente el 60% a la hora de clasificar pulmones sanos y enfermos, se ha concluído que con los recursos apropiados este método puede ayudar a realizar el análisis médico pero se debe ser consciente de sus limitaciones.

Palabras clave: Deep Learning, Imagen Médica, Redes Neuronales Convolucionales, Ciencia Abierta

ABSTRACT

There are a lot of diseases that require the analysis of medical images to be diagnosed. Historically this analysis has been done by experts who often don't even agree with each other, and in some cases a quick and appropriate diagnosis could lead to saving a life. In recent years there has been a development of Machine Learning techniques applied in the field of medical imaging. Automatising diagnosis would reduce wait times, improve effectiveness and detect findings that may have been otherwise overlooked. The purpose of this project is to study the viability of applying Deep Learning methods, concretely Convolutional Neural Networks for image classification, for analysing medical images. The dataset used corresponds to a set of chest x-rays provided by the *NIH Clinical Center* and tagged according to their diagnosis. The goal of this project is determining if this type of computer-aided detection would help with medical imaging findings interpretation, reducing the time diagnosing usually takes and exceeding the effectiveness of radiologists.

After describing Deep Learning in depth, and building a model with Convolutional Neural Networks with an accuracy of approximately 60% in classifying healthy and unhealthy lungs, it has been concluded that with the right resources this method can perform medical analysis but there must be awareness of its limitations.

Keywords: Deep Learning, Medical Imaging, Convolutional Neural Networks, Open Science

INDEX

1.- INTRODUCTION

1.1.- OBJECTIVES

This project aims to study if it is possible to develop a program capable of diagnosing diseases with medical images of chest x-rays. The diseases we take into account, as they appear in the dataset are: atelectasis, consolidation, infiltration, pneumothorax, edema, emphysema, fibrosis, effusion, pneumonia, pleural thickening, cardiomegaly, nodule mass and hernia.

The *NIH Clinical Center* provided the scientific community with a public chest x-ray dataset, and our objective is the same as theirs [1], which reads:

"[...]the hope is that academic and research institutions across the country will be able to teach a computer to read and process extremely large amounts of scans, to confirm the results radiologists have found and potentially identify other findings that may have been overlooked.
In addition, this advanced computer technology may also be able to:

> *help identify slow changes occurring over the course of multiple chest x-rays that might otherwise be overlooked*
>
> *benefit patients in developing countries that do not have access to radiologists to read their chest x-rays, and*
>
> *create a virtual radiology resident that can later be taught to read more complex images like CT and MRI in the future. "*

Therefore, our goal is improving the methods of diagnosing by helping doctors read the x-rays so they spend less time with each patient without compromising their effectiveness (according to the newspaper *20 minutos* the average wait time in Spain to have the results of a medical test is four weeks [2]), detecting findings that may have been overlooked (humans are not 100% effective and they can err on reading the images) and even losing the need of a

doctor, which is a massive help for places that have very few of them. Nonetheless, this is a long term objective, so the focus of this document is to study what efficiency can be reached when there is little information to work with, exploring the possibility of using Artificial Intelligence (AI) technology in medical imaging.

1.2.- OPEN SCIENCE: BIG DATA AND THE WORTH OF OPEN DATA

Big Data (BD) is a set (or an ensemble of sets) of data whose complexity, size and growth rate make it difficult to process, analyze or manage it through conventional tools or technology in time for them to still be useful. There is no consensus on the size of Big Data, but it is usually considered to be a set of data bigger than 40 terabytes. The complexity of BD is due to its unstructured nature. It is generated mainly by modern technology, like online shopping, social media and online searches in electronic devices such as smartphones and computers [3].

What makes Big Data important is that it provides a huge source of information, useful for enterprises and organizations, as its analysis helps to improve efficiency and to find out and solve problems. In our case, having a lot of information about diagnosis of medical images that is easy to access and process enables us to analyze new medical images through a computer program, so the health care system gets faster and offers additional information to diagnosis based on human analysis only.

With the advantages of Big Data, the scientific community can develop many applications like the one this project takes on, but it needs the free access to that data. Therefore, open data is key for development, as *The Center For Science Data* [4] puts it "*At the Center for Open Science, we believe an open exchange of ideas accelerates scientific progress towards solving our most persistent problems. The challenges of disease, poverty, education, social justice, and the environment are too urgent to waste time on studies lacking rigor, outcomes that are never shared, and results that are not reproducible. Making your work openly visible to other researchers invites collaboration, allows others to benefit from and build on your work, and facilitates replication*"

This project goes in line with this principles, and the sources we have used are all open: images provided by the *NIH Clinical Center* which are Open Data (and Science Data too), and both a programming language (Python) and a hosting service (Github) which are Open Source.

1.3.- MACHINE LEARNING IN MEDICAL IMAGING

A tool that can help in rendering medical diagnosis is a technique known as Machine Learning (ML), whose algorithms extract the most representative features of the images under study, and exploit their correlations to perform a classification allowing to make a prediction or diagnosis. Most of the Machine Learning methods are open source, so it is easy to apply them to images, but one needs to be cautious as the methods can err and result in misleading metrics. One of the methods is Deep Learning (DL), and it identifies image features as part of its learning process, so it has the advantage of not requiring a first step of feature engineering in which the user must identify the most representative characteristic by hand. It has been used in medical imaging and it is expected to have a greater influence in the future [5], so whoever works in medical imaging must be aware of how it works.

"Computer-aided detection and diagnosis performed by using machine learning algorithms can help physicians interpret medical imaging findings and reduce interpretation times"[6]. Nevertheless, if the strengths and weaknesses of this technology are not understood, it can be misused and lead to wrong diagnosis. Therefore, the efficiency and risks of these methods need to be studied and comprehended.

Some studies predict that out of the seven major causes of death worldwide, four will be lung diseases by 2020, and some of those diseases are already at that top, like pneumonia [7]. Managing pneumonia rely on analysing chest radiographs, so it is considered to be a type of medical image with research priority for computer analysis and we will focus on it, specifically chest X-rays.

2.- DEEP LEARNING: DESCRIPTION OF THE METHOD

2.1.- BRIEF INTRODUCTION TO DEEP LEARNING

In 1959 the concept of Machine Learning [8] was first introduced, as a subfield of Artificial Intelligence, and it aimed at programming computers to simulate the behaviour of the human mind; having a machine reproduce human intuition. Our brain is able to make abstract connections between images, sounds or written words and the concepts they represent, so a machine simulating that process would be able to receive as input (for example) a picture of a table, a voice record saying "table", or the string 'table', and *know* it represents a table, an item of furniture with a flat top and one or more legs supporting it. Furthermore, it would be able to differentiate a table from a stool, although the physical description of them is very similar. A lot of technology which is able to do this process has been developed in the past few years, like the filters of Snapchat (they detect a face through the camera and the position of its features) or the voice command devices, like Siri. This is just a small part of AI, as its traditional goals are reasoning, knowledge, planning, learning, natural language processing, perception and the ability to move and manipulate objects, but this example already shows that there are processes too complex for computers to do by means of regular programming; we need to develop new methods. In conclusion: Artificial Intelligence tries to develop a machine that is able to perform tasks that are usually associated with the human intelligence by implementing new ways of programming.

Regular programming consists on a well defined method based on the introduction of equations and formulas so a program takes data and gives back certain result, while in ML the goal is to find out algorithms that allow a program to "learn" how to return a result given certain data as input without having programmed a series of equations that define the process needed: that is why we consider it to be simulating the behaviour of a brain; we don't necessarily connect two concepts innately (we don't have a well defined program) but we can learn what their connection is (we have an algorithm that allows our learning process). In our case, DL, this algorithm consists on Convolutional Neural Networks (CNNs) which we will describe later. What they do is transform data in certain way so the new representation allows the machine to solve the problem.

Deep Learning is part of Machine Learning, this field was first presented in 1986 but it wasn't really developed until the 21st century, when graphics processing units (GPUs) improved and increased the speed of the Deep Learning training process by a factor of 100. The advantage of this method was previously described; it does not require a step of feature extraction by the programmer that other ML methods do require, it only needs data (which often has to be labeled) and it extracts the features as it trains. Another relevant factor for the rise of this method was the use of internet for accessing data, as Big Data is needed.

Deep Learning is great at tasks that are part of artificial vision (which requires the processing of images) like style transfer, detection of objects and image classification to name a few. In this document we will be focusing on the last one.

Before classifying an image, it has to go through certain steps we call the image processing [9]:

- Image capture: the image is taken by a camera or, in this case, a medical scanner, and then it is digitized.

- Pre-processing or enhancement: the digitized image is slightly changed in order to emphasize important features.

- Segmentation: some features are selected; for example in an X-ray of the chest a feature to select would be the outline of the body.

- Description: as high level processing, the radiometric and photometric descriptors are extracted.

In image classification, the learning process is achieved by giving the machine a lot of labeled processed images as input, with an algorithm that consists on layers of filters, which are the mean to transform the original pictures into different representations, that analyzes the pixels of the image (which the machine sees as arrays of numbers) and finds patterns. This kind of process is called supervised learning, and if the input data wasn't labeled it would be called unsupervised learning. The amount of layers needed is what gives the model its depth,

thus its name Deep Learning. With this method, each time the program runs, as its goal is to label images, it knows where it has its errors (the right labels are already part of the input), so the program can improve with each run (in DL, everytime the program runs is called an epoch) through the study of the loss function. In this case, improving means learning what patterns are key for the classification of an image by modulating the parameters of the layers.

The programming language used is Python, released in 1991 by Guido Van Rossum as an easily readable multi-paradigm interpreted high-level programming language[10]. Its development takes place in GitHub, a web-based hosting service for version control using git, and the command shell used for interactive computing in Python is Jupyter Notebook. The main advantage of using Python is that it requires simple syntax, so programmers need fewer lines of code in comparison to other languages.

2.2.- THE STRUCTURE OF THE ALGORITHM

We mentioned CNNs as they are part of the algorithm we are going to use, its model (the prediction apparatus that takes an image as input and predicts the label as output), but there are other elements and they can be briefly described.

Dataset specification involves having the input dataset with each image as a pair of an input vector, which contains the image as an array of numbers (from 0 to 255, the value of an individual pixel, although it is usually normalized, so all values are divided by 255), and an output scalar, the label that represents the class (in this case, the disease).

As it has been mentioned before, the ability to learn comes from the study of the *loss* function. It represents the inefficiency of the algorithm with an equation that has higher values the more errors the program commits, and decreases as the model improves.

Therefore, the optimization procedure consists on varying the parameters of the model in order to decrease the loss function, studying its minimum by equating its partial derivative with respect to the parameters to zero. This particular method is called *Gradient Descent*,

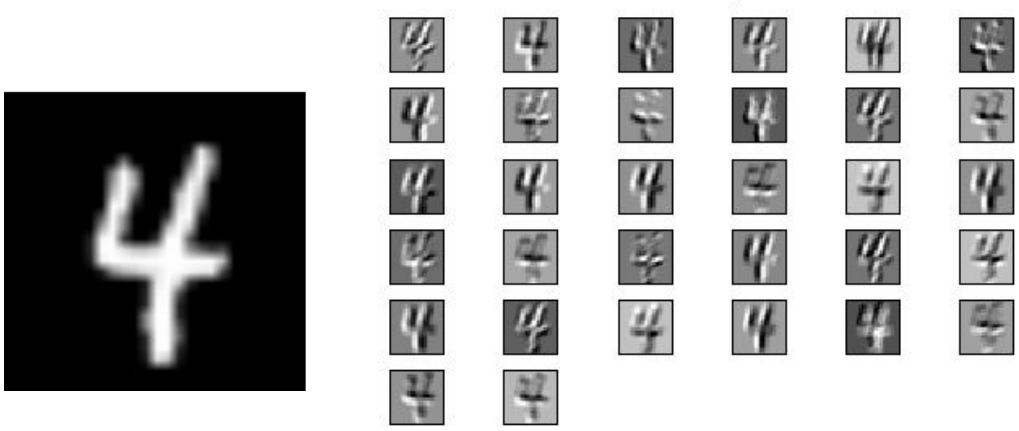
where the parameters are updated on each iteration on the basis of the gradient of the *loss* function.

2.2.1.- CONVOLUTIONAL NEURAL NETWORKS

This model consists on a series of layers that learn local patterns of characteristics of the input. Neural Networks were inspired by the human biological nervous system [11], so they first consisted of two layers, the input layer and the output layer, which were directly connected and could classify linearly separable patterns. As the patterns are usually more complex, more layers (the hidden layers) need to be added between those two. The structure ends up being a series of layers which are made of interconnected neurons, and in order for a layer to forward an output to the next layer which is different from their input, those neurons have to perform some processing on the data they have received.

The architecture of a typical CNN is a series of stages. It starts with three types of layers: the convolutional layers (to create a map of features), the activation layers (to make the weighted sums that occur in the convolutional layers non-linear) and the pooling layers (to merge similar features into one). The first type consists on filters, 2D windows that go through the image studying its pixels, in this case, their values on the correspondent array. The most common size for this filters is 3x3, and it is the one we will use: smaller filters mean higher precision but if they are too small they can't detect patterns. When applying the filters, you can add a couple of pixels outside the image (padding) with value 0 so all the pixels of the input image are equally relevant, even the corners, and you can also set the movement of the filter through the image (stride). We won't be using padding nor striding techniques (we use padding=0 and striding=1), because the characteristics that we need to highlight in the images aren't in the borders of the picture. As every image is a big set of numbers, it is easier for the machine to process the data if we apply the second type of layers we have mentioned, specifically MaxPooling, which consists on only taking the pixel with the highest value of all the non-overlapping NxN cells in the image (in our case we have chosen N=2), so the dimensionality of the map of characteristics gets reduced.

Once the filters (the nodes of the convolutional layer) analyze the input, they create a feature map [12], and its units are connected to the feature map of the previous layer.



(a) Original image (b) Feature maps

Figure 1: Image from MNIST and its feature maps activated by certain kernels.

The way they are connected is called a filter bank and it consists of a set of weights. Therefore, every time a filter is applied, the input goes through changes defined by the parameters of the filter bank, and each filter bank is applied to all units of a single feature map [13]. As the input is an array of numbers (x), a filter will change it by applying a weight (w) and a bias (b), so for the whole array the change can be defined as

$$\sum_{i}^{n} x_i w_i + b_i \tag{1}$$

It is a sum, so applying one convolutional layer after another could be reduced as only one layer, the linear combination of all of them. To avoid that, we need a nonlinear activation function to get rid of that linearity. We will be using two activation functions; ReLU (rectified linear unit) and the sigmoid function.

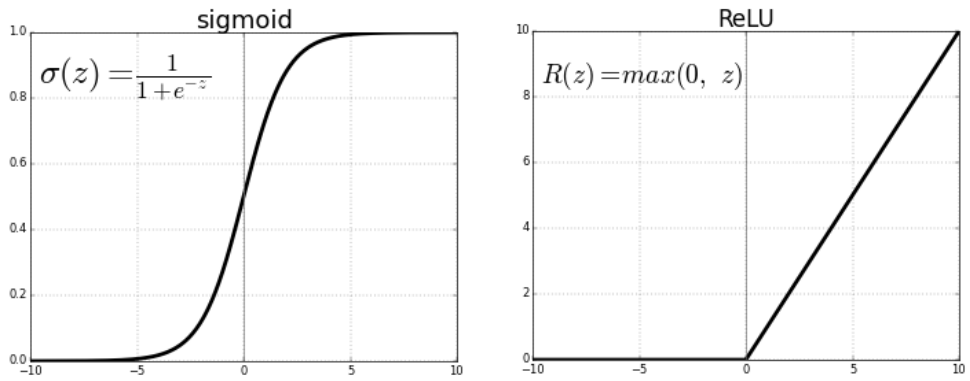


Figure 2: Graphic representation of the sigmoid and ReLU activation functions.

The ReLU's result goes from zero to infinity, so we will use it for the input layer and the hidden layers, while the sigmoid results in a value between 0 and 1, so it produces a probability for a model of two classes, therefore we use it for the output layer [14]. The node of a convolutional layer plus the activation function can be represented as [15]

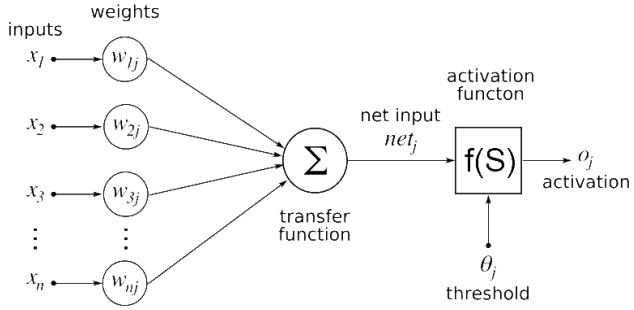


Figure 3: Sketch of the node of a convolutional layer plus the activation function.

Our model can now be described as shown in the next figure [16], with the groups of Σ being the filter banks and f' the activation function:

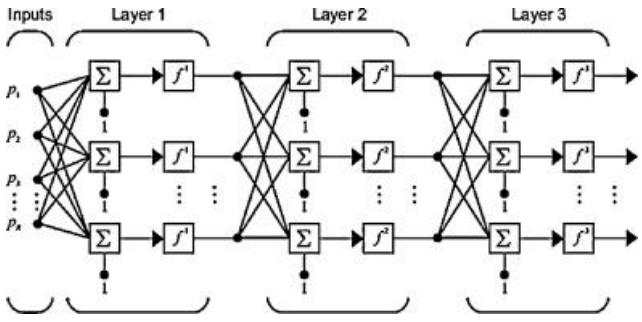
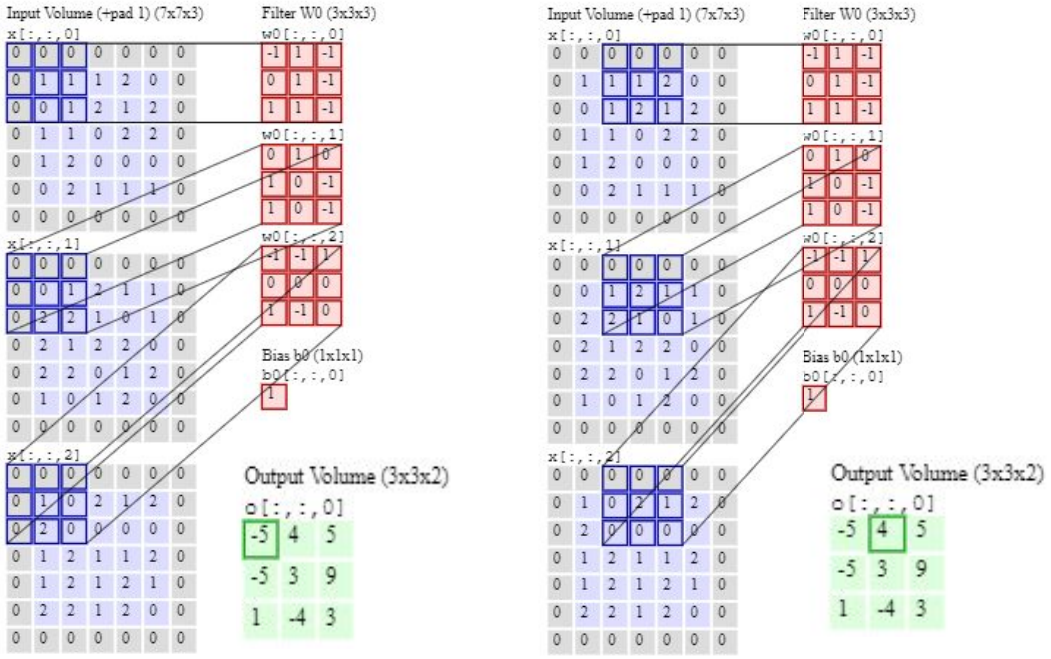


Figure 4: Sketch of three layers of a model where the bias is 1.

The size of the input is (N, H, W, C), where N is the number of images in the input set, H is their height, W is their width and C is the number of channels. When the original picture is black and white, it only needs one (C=1) array of numbers to be represented, so the number of outputs from the first layer is the number of nodes it has. It happens the same with images in colour; even though the first layer receives three (C=3) arrays (red green blue or RGB) of numbers, each one of its filters produces only one output: the sum of the three numbers that define the pixel [17].



(a) Filter applied to a local region of the input (b) Filter applied to another local region of the input

Figure 5: Example of applying one filter to a RGB image. Although one local region implies three channels as input, the output has only one channel.

Other couple of layers [18][19] which are usually alternated with the convolutional and the pooling layers in the model are:

- Dense layer or fully connected layer: performs classification on the features extracted by the convolutional layers, and every node in this layer is connected to every node in the preceding one. It is used to finish a model, because its number of nodes (n) defines its number of outputs (o). How they are related depends on the activation function, in our case because of using a sigmoid function (it goes from 0 to 1) the

relationship is $n=o-1$. Therefore, if we want the prediction of two classes, one for findings (disease) and another one for no findings (healthy patient), we use a dense layer with one node, returning a value when predicting a class of either 0 or 1.

- ➢ Flatten layer: flattens the input (a matrix transforms into a vector), so if the previous layer produces an output with shape (a,b,c), the flatten layer will produce an output with shape (a*b*c).

When we say a model is trained we mean it already has the weights and biases suitable for a specific dataset. Using a complete pretrained model or just some layers of it with another dataset is called transfer learning. Pretrained models like AlexNet can be found in OS libraries, and they are useful as training from scratch a model with a large dataset needs time and a good processor. Usually, those models are trained on ImageNet [20], a big online dataset with more than 14 million images divided in over 20 thousand categories, but there are more available datasets like MNIST, CIFAR-10 and CIFAR-100. There are also convolutional networks (ConvNets) available that aren't trained but have certain architecture that has been proven to be useful applied in different models, like densely connected convolutional networks (DenseNet), MobileNet and ResNet50.

On December 2017, a ML group in Stanford [21] published their own mode called CheXNet, which uses the ConvNet DenseNet and was applied to the whole dataset provided by *NIH Clinical Center* (we only use a sample of it). Although their approach is different than ours (their two classes are not healthy patients and patients with disease, but finding pneumonia or not), it will be of interest to discuss their conclusion and compare it with our results.

2.2.2.- CLASSIFICATION OF IMAGES

Once the model is constructed, the learning process starts. There are some concepts which need to be explained so the DL method is understood:

- ➢ Training: when running the *fit* function, the model applies its layers to a set of labeled images called the train set and calculates weights and biases, as the model knows what the right predictions are because of having the right labels as input. The weights

and the biases of the algorithm aren't calculated after going over the whole train set, but over smaller groups of images from that set, called *batches*. Then, the model evaluates (with the parameters it has calculated) another set of labeled images called the validation set, and compares its predictions to the right labels to compute the accuracy of the model. The whole process described is called an epoch, and every epoch redefines the parameters of the model. The number of epochs is defined in the *fit* function, and you can adjust it by introducing a *callback* function, which stops the training process when the accuracy (or the loss) achieved isn't increasing (or decreasing) more (or less) than a certain quantity when running another epoch. The size of the *batches* is also defined in the *fit* function.

➢ Loss function: during the training process the model changes the parameters by studying the *loss* function (also known as *cost* function) when testing the validation set, which represents how much the model is failing by comparing the predictions to the labels in the form of an equation. The loss function we will be using is *binary crossentropy*, defined as

$$Hy'(y) = -\sum_i [\, y'_i log(y_i) + (1-y'_i) log(1-y_i)] \qquad (2)$$

where y_i is the right label of the i image and y'_i is the prediction for that image, thus the *loss* function H decreases when the accuracy increases. It is introduced in the *compile* method, which configurates the learning process.

➢ Optimizer: as we want to have the minimum *loss* function, we include an *optimizer* function in the *compile* method, which reduces it by studying its derivative. We previously described *Gradient Descent*, but there is a simplified version of this method called *Stochastic Gradient Descent* (SGD) [22], which instead of studying the *loss* function that results from all the samples of the dataset it only studies the loss gradient of one randomly picked sample at each iteration. In our program, we will be using the *optimizer Adam* [23], which differs from SGD by calculating an exponential moving average of the gradient and the squared gradient, with two parameters (β_1 and β_2) controlling the decay rates of these moving averages. Its equation is defined as

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

➢ Test: the labeled images used to evaluate the model is called the test set. The program can now predict classes for data it hasn't "seen" before and check how its performance is. This is done with the *predict* method, and it doesn't change any parameter of the model. To show the results of testing we will be using a confusion matrix [24].

➢ Model complexity: one would think that the accuracy of the model increases with the number of epochs during training. Nonetheless, the model could suffer from overfitting: the model is too adjusted to the data from the train and validation sets and errs when applied to a set which contains data that isn't similar to the one used during training. On the other hand, the model could also suffer from underfitting [25] and not be able to capture complexity of the problem sufficiently. In both cases the generalization ability of the model, which is its ability to produce meaningful results from data that were not previously observed, suffers. Therefore, it is needed to take into account the complexity of the problem when designing a machine learning model.

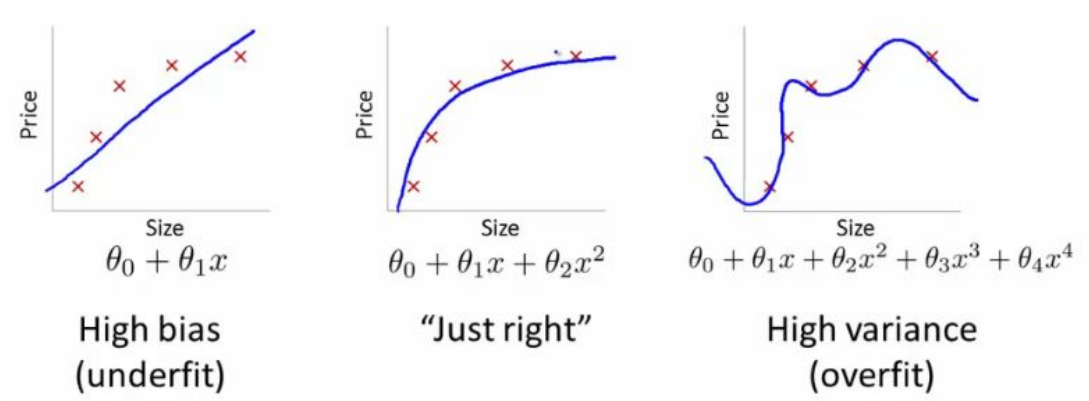


Figure 6: Graphic examples of the generalization ability of a model

2.3.- DATA MANAGEMENT

We have 112120 images given by the *NIH Clinical Center* and they are available online as Open Data [26]. Because of the restrictions given by the processor we have available, only 5606 images have been used to train, validate and test the model. We then have used another 4999 images to do further testing.

We have labeled the images that show any of the diseases mentioned in *Objectives* as 1 and the images who belong to healthy patients as 0, which means class 0 for the images that corresponded to 'No Finding' in the documentation and class 1 for the rest of images. Doing this classification instead of recognizing each disease individually is due to:

- ➢ Some diseases suffered from class imbalance, which means that there were very few medical images that showed them.

- ➢ A big part of the non healthy patients had two or more of the diseases showing in their chest x-ray.

- ➢ Some of the diseases would have needed a complementary side x-ray to differentiate from one another.

➢ Our processor was only able to handle a small sample of the whole dataset.

When the dataset is too small, or all images are too similar, data augmentation [27] can be applied. This method enlarges the dataset by adding images that are variations of the original ones. In our dataset we already have images that aren't centered, that are rotated and that vary in a range of sizes, so data augmentation to avoid similarities is not needed. Nevertheless, it could be used to enlarge the dataset.

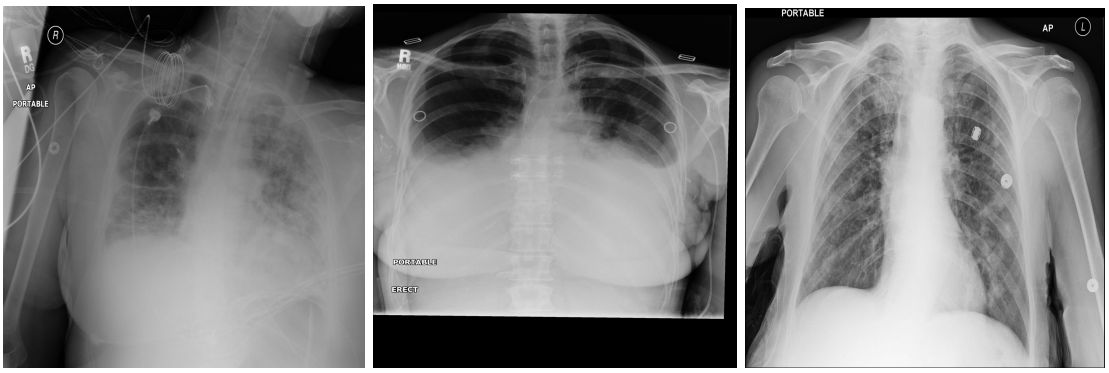


Figure 7: Examples of images from our dataset. It shows that some of the original images aren't centered and some even show some rotation.

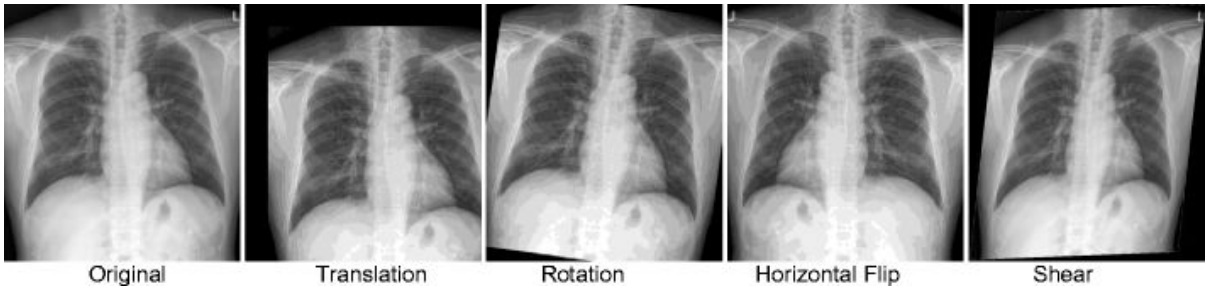


Figure 8: Example of applying augmentation to a chest x-ray.

3.- METHODOLOGY

3.1.- TOOLS

The hardware used has been a processor Intel® Xeon ® CPU E7-4870, core of 2'40GHz.

Part of the software used has previously been briefly described: the hosting service is GitHub, and the programming language is Python. The web application where we have coded is Jupyter Notebook and the algorithms were all coded using Keras, an Application Programming Interface (API) which uses as backend Google's TensorFlow [28], with Keras acting as a model definition add-on for it. All the coding was done using the built-in TensorFlow models and libraries, which we have listed in the code down below. Everything is Open Source.

```
import keras
import os, shutil
from keras import layers
from keras import models
from keras import optimizers
from keras import regularizers
from keras import initializers
from keras import callbacks
from keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
from PIL import image
from sklearn import svm, datasets
from sklearn.metrics import confusion_matrix
import itertools
```

3.2.- CODING

To create a model, we first started varying the amount and characteristics of the convolutional layers, the pooling layers and the dense layers, trying different optimizers, varying their learning rates, changing the size of the input data, changing the batch size and trying different activation functions. The model we finally decided on was the one which provided the highest average of accuracy between training and testing.

Once we had the model we compiled it three times, creating three programs A, B and C, in order to study if it made a difference varying the input (colour and normalization) and the trainset-testset ratio.

Program	Normalized Data	Colour/ Channels	Train set Size	Validation set Size	Test set Size
A	Yes	RGB / 3	3000 images	500 images	2106 images
B	No	Greyscale / 1	3000 images	500 images	2106 images
C	Yes	RGB / 3	4606 images	500 images	500 images

Table 1: Main characteristics of the programs.

3.2.1.- PREPARATION OF THE IMAGES

This is the code used to prepare the data, where:

-**labels** is the name of the cvs document that has the labels of the dataset.

-**path_images** is the folder where the images are stored.

-**colour** is 'RGB' for saving the images in colour and 'L' for black and white.

-**im_arr = im_arr/255** is the command to normalize the data,

-α is the length of the train set.

-β is the sum of the lengths of the train and validation sets.

-**arrays** is the name of the file where we save the dataset (with labels) as arrays

```
metadata = pd.read_csv('labels.csv')
metadata = metadata.as_matrix()

paths, labels = [], []
for i, row in enumerate(metadata):
    im_name = row[0]
    label = row[1]
    if label == 'No Finding':
        label = 0
    else:
        label = 1
    paths.append(im_name)
    labels.append(label)

paths, labels = np.array(paths), np.array(labels)
args = np.arange(len(paths))
np.random.seed(100)
```

```
np.random.shuffle(args)
paths = paths[args]
labels = labels[args]

images = []
for im_path in paths:
    im = Image.open('path_images/{}'.format(im_path))
    im = im.resize((150, 150))
    im = im.convert('colour')
    im_arr = np.array(im)
    im_arr = im_arr/255
    images.append(im_arr)
images = np.array(images)

X_train, y_train = images[:α], labels[:α]
X_val, y_val = images[α:β], labels[α:β]
X_test, y_test = images[β:], labels[β:]

d = {'X_train': X_train,
     'y_train': y_train,
     'X_val': X_val,
     'y_val': y_val,
     'X_test': X_test,
     'y_test': y_test}

with open('arrays.pkl', 'wb') as f:
        pickle.dump(d, f)
```

3.2.2.- MODEL

Model description, where **C** (channels) equals 1 for the images in greyscale and 3 for the images in RGB

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(150,150, C)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.Adam(lr=0.0003),
              metrics=['acc'])
```

```
history = model.fit(
    x_train, y_train,
    batch_size=10,
    epochs=100,
    verbose=1,
    callbacks=callbacks.EarlyStopping(monitor='acc', min_delta=0.0005, patience=2),
    validation_data=(x_val, y_val))
```

3.3.- RESULTS

Program	Epochs	Training Accuracy Achieved	Loss
A	57	0’9727	0’0877
B	10	0’8197	0’4103
C	26	0’9520	0’1321

Table 2: summary of the training process of the three programs (to see graphically the training accuracy and loss, check *Appendix A*)

To test the programs we used the following command:

```
predictions = model.predict_classes(x_test)
```

This step produced the next confusion matrices (code in *Appendix C*):

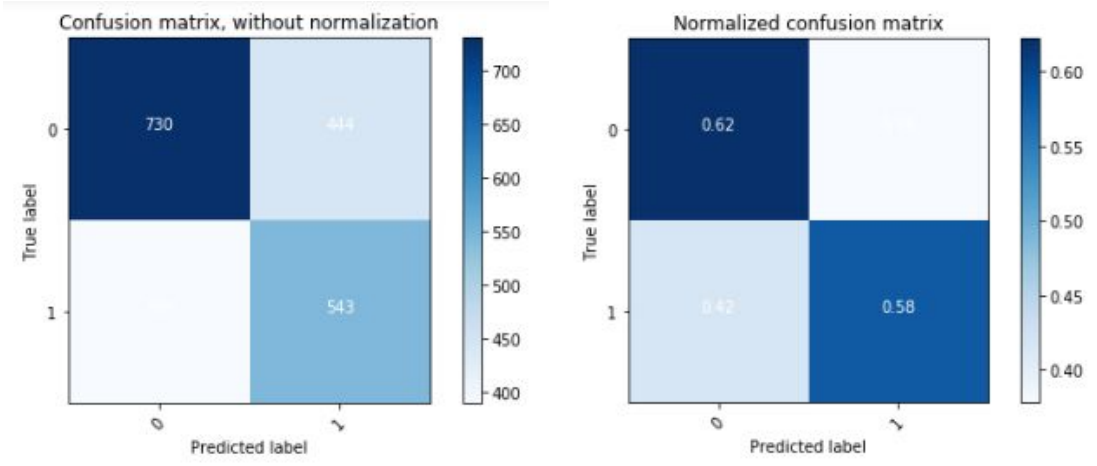


Figure 9: Confusion matrices of program A testing with its initial test set.

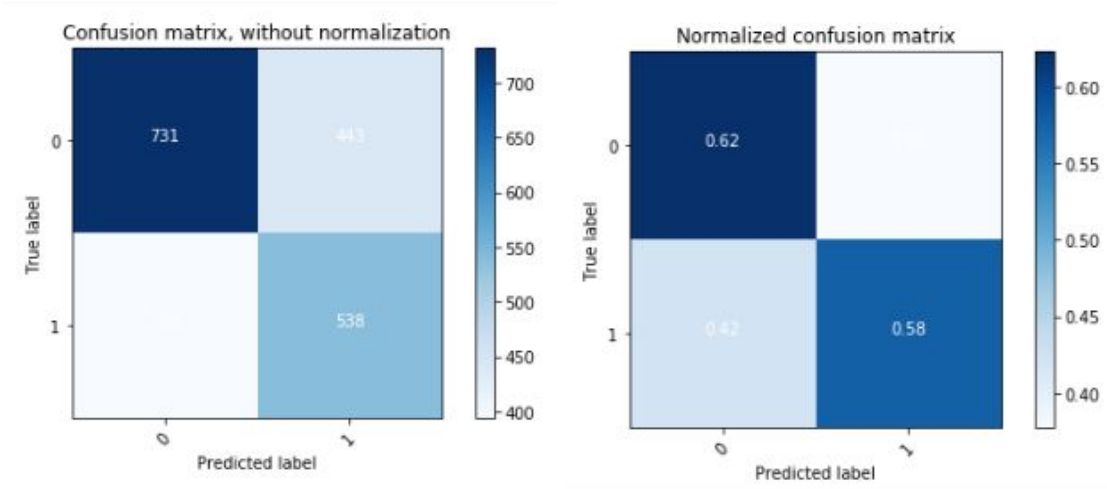


Figure 10: Confusion matrices of program B testing with its initial test set.

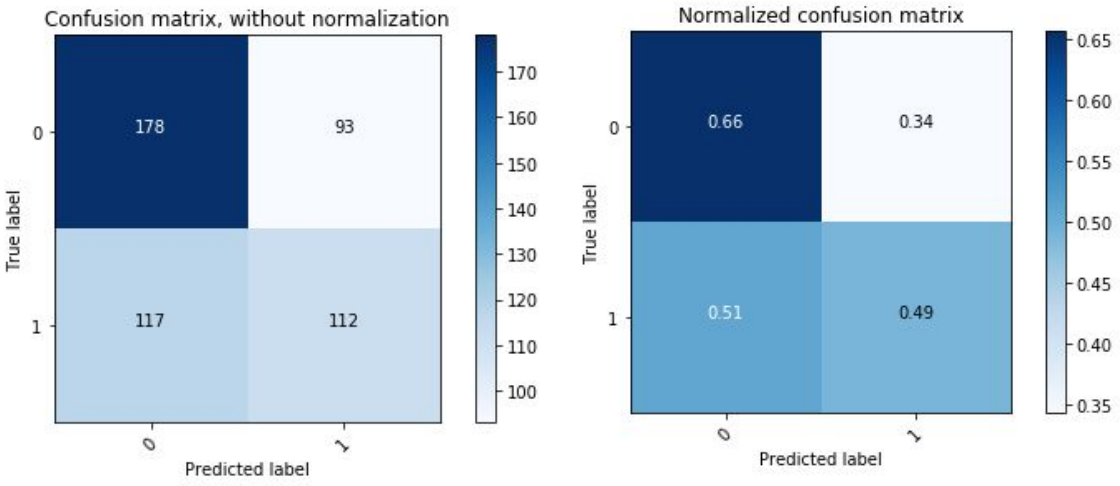


Figure 11: Confusion matrices of program C testing with its initial test set.

We also tested the three programs with another sample of the *NIH Clinical Center* dataset, this one consisting on 4999 images, resulting in the following confusion matrices:

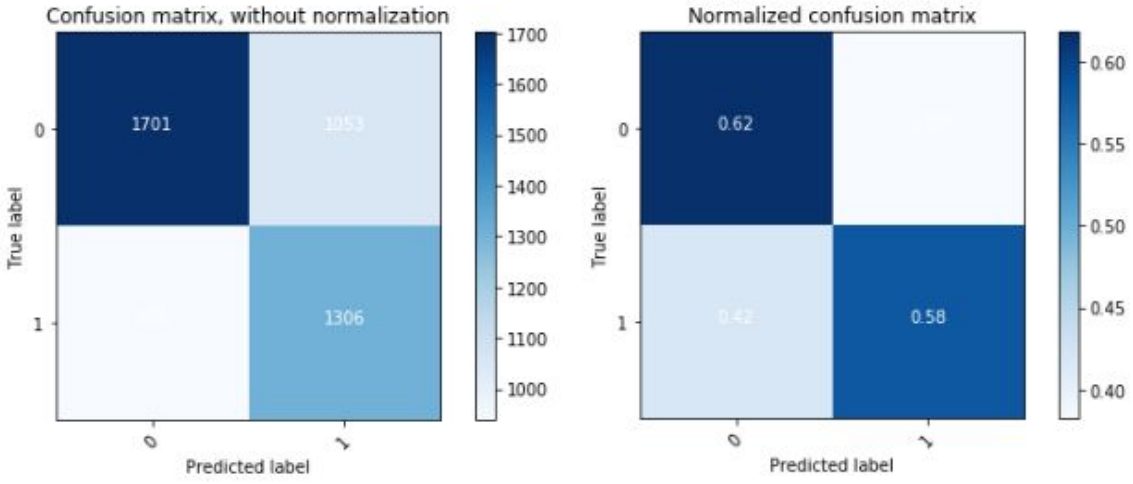


Figure 12: Confusion matrices of program A testing with the second sample.

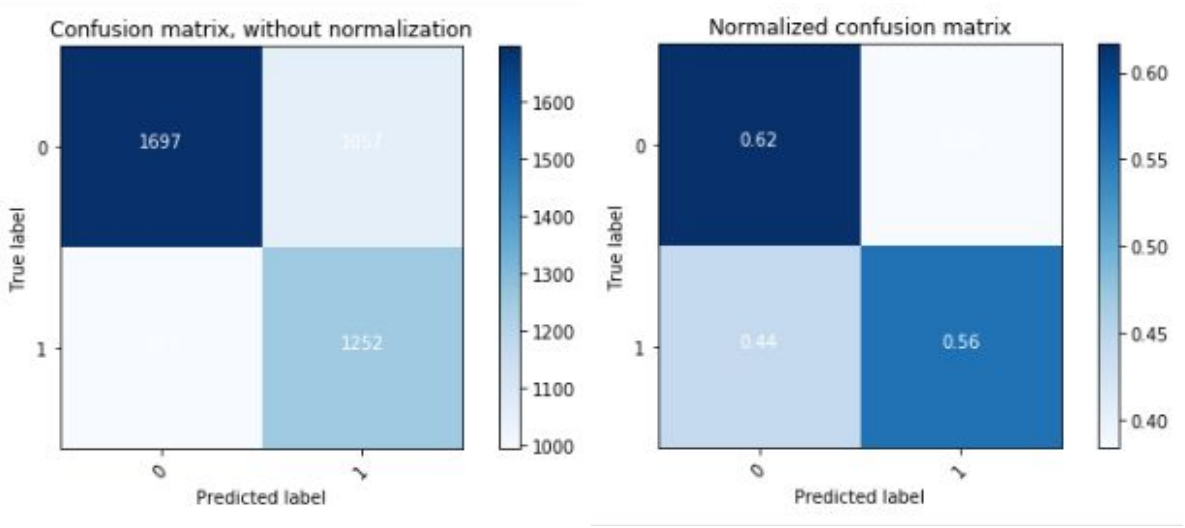


Figure 13: Confusion matrices of program B testing with the second sample.

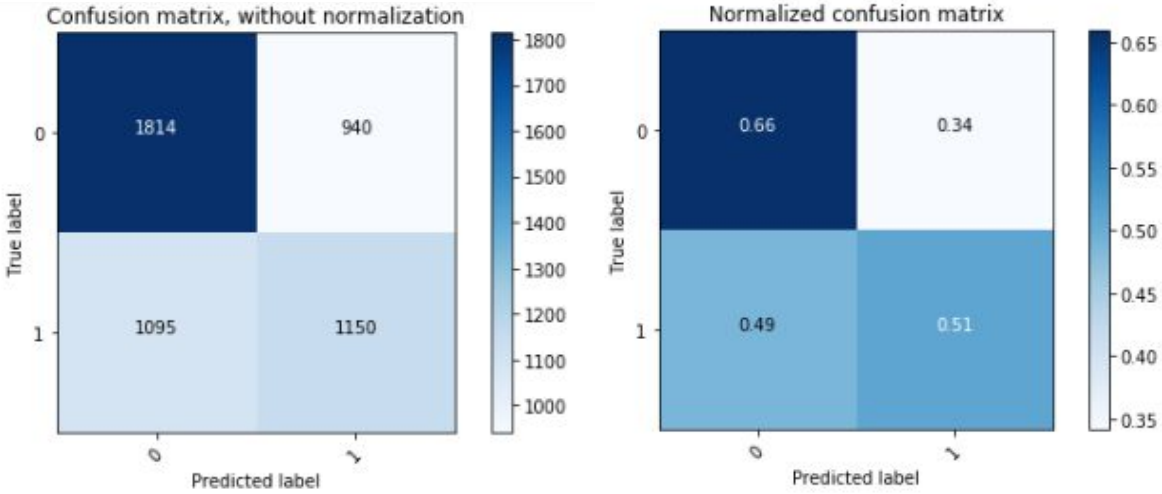


Figure 14: Confusion matrices of program C testing with the second sample.

These results show that the model is more effective at predicting correctly healthy images than for images that show diseases. All programs show similar accuracy in both tests made, with variations in accuracy of no more than 2%. Using RGB data as input doesn't make a big difference compared to using data in greyscale, so this aspect won't be discussed. Nevertheless, using more images for training increases the accuracy in predicting healthy x-rays but reduces the accuracy in predicting x-rays that show diseases.

4.- DISCUSSION AND CONCLUSIONS

It is interesting to compare the results of the programs when testing them with the same sample of 4999 images:

Program	Channels	Size Train Set	Test Acc. Class 0	Test Acc. Class 1
A	3	3000	62%	58%
B	1	3000	62%	56%
C	3	4606	66%	51%

Table 3. Summary of the results of the three programs.

The results from programs A and B barely differ, which concludes that analyzing the images in greyscale or in RGB doesn't have much effect on the accuracy of the program, but when comparing the results of program A with the ones of program C they do indicate that the size of the train set affects the accuracy of the model.

The train set used for the learning process in program C had 1606 images more than the train set in program A, specifically 903 more images of healthy lungs (class 0) and 703 more images of lungs with diseases (class 1). The results summarized in the table agree with the estimation[29] that in the dataset more than 50% of the class 0 images available are correctly labeled. Therefore, for class 0 a bigger train set implies having more reliable data to analyze and the model gets better at predicting (goes from 62% to 66% acc.). On the other hand, more than 50% of the class 1 images are estimated to be wrongly labeled, so for this class a bigger train set implies analyzing more of the wrong features, thus the model learns 'false' information and its accuracy decreases (goes from 58% to 51% acc.).

This project has successfully studied the applications of DL in the analysis of medical images but has failed to produce a useful model due to the lack of resources, like not having a processor that could handle a big dataset, not having the time to work on a much more complex model or not finding a reliable set of medical images. This last reason is relevant to the development of Deep Learning in general (it needs more open data to improve) and to

medical imaging in particular, as diagnoses by different pathologists tend to differ greatly. In medical imaging, specifically when analysing lung diseases, it is important to take into account non-imaging information about the patient to make diagnosis, like clinical history or previous results. Therefore, a program which is only studying the x-rays won't be optimal predicting as it lacks relevant information. In 'CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning' the Stanford group studied, as it was mentioned before, the whole dataset offered by the *NIH Clinical Center*. They achieved an accuracy of 85% and this result shows how promising applying Deep Learning techniques in medical imaging is, even with a dataset with wrong labels and images that aren't centered a good accuracy can be reached (the Stanford group compared the harmonic average of the precision and recall of the models with the performance of four radiologists and found CheXNet to produce higher score).

The keys to a successful development of DL are having more reliable open data available and having a standard on how to take the images, which would mean for the data to be all centered so it is easier for the algorithm to study it. Even though standardizing medical imaging is a long term goal, it has been shown in this work that having large non-standardized non-reliable datasets is already improving the ML methods in the field of medical imaging, which should be a call to the scientific community to promote sharing data and their studies in this topics: open reproducible research is key for the future of Deep Learning.

In conclusion, our research studies applying Deep Learning in chest x-rays diagnosis but, although it results in an appropriate model in context of this work, it doesn't result in coding an optimal model because of the limitations of the hardware and the lack of time to increase the complexity of the model. As proven by the Stanford group, when the work isn't limited to the format of a TFG it can reach better results.

BIBLIOGRAPHY

[1]'NIH Clinical Center provides one of the largest publicly available chest x-ray datasets to scientific community'(2017) *National Institutes of Health (NIH)*. Available at: https://www.nih.gov/news-events/news-releases/nih-clinical-center-provides-one-largest-publicly-available-chest-x-ray-datasets-scientific-community (Accessed: 27 June 2018).

[2]20Minutos (2012) *El tiempo medio de espera para una prueba es de 54 días y de 82 para tener un diagnóstico*, *20minutos.es - Últimas Noticias*. Available at: https://www.20minutos.es/noticia/1655301/0/ocu-estudio/listas-de-espera/especialistas-sanidad-publica (Accessed: 27 June 2018).

[3]'Big Data: ¿En qué consiste? Su importancia, desafíos y gobernabilidad'. *PowerData* (n.d.) Available at: https://www.powerdata.es/big-data (Accessed: 27 June 2018).

[4]*Center for Open Science* (2013). Available at: https://cos.io/ (Accessed: 27 June 2018).

[5]Erickson, B. J. et al. (2017) 'Machine Learning for Medical Imaging', *RadioGraphics*, 37(2), pp. 505–515. doi: 10.1148/rg.2017160130.

[6]Schoepf, U. J. and Costello, P. (2004) 'CT Angiography for Diagnosis of Pulmonary Embolism: State of the Art', *Radiology*, 230(2), pp. 329–337. Doi: 10.1148/radiol.2302021489.

[7]Mathers, C. D. and Loncar, D. (2006) 'Projections of global mortality and burden of disease from 2002 to 2030', *PLoS medicine*, 3(11), p. e442. doi: 10.1371/journal.pmed.0030442.

[8]Samuel, A. L. (1959) 'Some Studies in Machine Learning Using the Game of Checkers', *IBM Journal of Research and Development*, 3(3), pp. 210–229. doi: 10.1147/rd.33.0210.

[9] Zapletel, O. (2017) *Image Recognition by Convolutional Neural Networks - Basic Concepts*. Master thesis. Brno University of Technology.

[10]'General Python FAQ' — *Python 2.7.15 documentation* (Last updated on Jun 26, 2018). Available at: https://docs.python.org/2/faq/general.html (Accessed: 27 June 2018).

[11]Razzak, M. I., Naz, S. and Zaib, A. (2017) 'Deep Learning for Medical Image Processing: Overview, Challenges and Future', *arXiv:1704.06825 [cs]*. Available at: http://arxiv.org/abs/1704.06825 (Accessed: 27 June 2018).

[12]Ho, S. (2018)'Handwritten digit recognition with a CNN using Lasagne'. 2 March. Available at: https://www.simonho.ca/machine-learning/mnist-cnn-lasagne/ (Accessed: 27 June 2018).

[13]LeCun, Y., Bengio, Y. and Hinton, G. (2015) 'Deep learning', *Nature*, 521(7553), pp. 436–444. doi: 10.1038/nature14539.

[14]Sharma, S. (2017) 'Activation Functions: Neural Networks', *Towards Data Science*. Available at: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (Accessed: 27 June 2018).

[15]Sadhu, T. (2012) 'Machine Learning: Introduction to the Artificial Neural Network', *Durofy*. Available at: http://durofy.com/machine-learning-introduction-to-the-artificial-neural-network/. (Accessed: 27 June 2018).

[16]Kumar, Y. J., Salim, N. and Raza, B. (2012) 'Cross-document Structural Relationship Identification Using Supervised Machine Learning', *Appl. Soft Comput.*, 12(10), pp. 3124–3131. doi: 10.1016/j.asoc.2012.06.017.

[17] 'CS231n Convolutional Neural Networks for Visual Recognition', *Stanford University* (2017). Available at: http://cs231n.github.io/convolutional-networks (Accessed: 27 June 2018).

[18] 'A Guide to TF Layers: Building a Convolutional Neural Network'. *TensorFlow*.(Last updated May 25, 2018.) Available at: https://www.tensorflow.org/tutorials/layers (Accessed: 27 June 2018).

[19]*Keras Documentation* (n.d.). Available at: https://keras.io/ (Accessed: 27 June 2018).

[20]*ImageNet* (2016). Available at: http://image-net.org/ (Accessed: 27 June 2018).

[21]Rajpurkar, P. et al. (2017) 'CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning', *arXiv:1711.05225 [cs, stat]*. Available at: http://arxiv.org/abs/1711.05225 (Accessed: 27 June 2018).

[22]Bottou, L. (2010) 'Large-Scale Machine Learning with Stochastic Gradient Descent', *Proceedings of COMPSTAT'2010*. Physica-Verlag HD, pp. 177–186. doi: 10.1007/978-3-7908-2604-3_16.

[23] Kingma, D. P. and Ba, J. (2014) 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980 [cs]*. Available at: http://arxiv.org/abs/1412.6980 (Accessed: 27 June 2018).

[24]'Confusion matrix' — *scikit-learn 0.19.1 documentation* (n.d.). Available at: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html (Accessed: 27 June 2018).

[25] 'efficiency - When is a Model Underfitted?' (2014) *Data Science Stack Exchange*. Available at: https://datascience.stackexchange.com/questions/361/when-is-a-model-underfitted (Accessed: 27 June 2018).

[26]'NIH Chest X-rays', *Kaggle* (n.d.). Available at: https://www.kaggle.com/nih-chest-xrays/data (Accessed: 27 June 2018).

[27] Lakhani, P. et al. (2018) 'Hello World Deep Learning in Medical Imaging', *Journal of Digital Imaging*, 31(3), pp. 283–289. doi: 10.1007/s10278-018-0079-6.

[28]Abadi, M. et al..(2015) 'TensorFlow: Large-scale machine learning on heterogeneous systems'. Software available from tensorflow.org.

[29]Oakden-Rayner, L. ~ (2017) 'Exploring the ChestXray14 dataset: problems', *Luke Oakden-Rayner*, 18 December. Available at: https://lukeoakdenrayner.wordpress.com/2017/12/18/the-chestxray14-dataset-problems/ (Accessed: 27 June 2018).

ACRONYMS

AI Artificial Intelligence. 6, 8

API Application Programming Interface. 19

BD Big Data. 6, 9

CNN Convolutional Neural Network.8, 9, 11

ConvNet Convolutional Network. 15

CT Computed Tomography. 5

DenseNet Densely Connected Convolutional Network. 15

DL Deep Learning. 7-10, 14, 27, 28

GPU Graphics Processing Unit. 9

ML Machine Learning. 7-10, 15, 17, 28

MRI Magnetic Resonance Imaging. 5

ReLU Rectified Linear Unit. 11

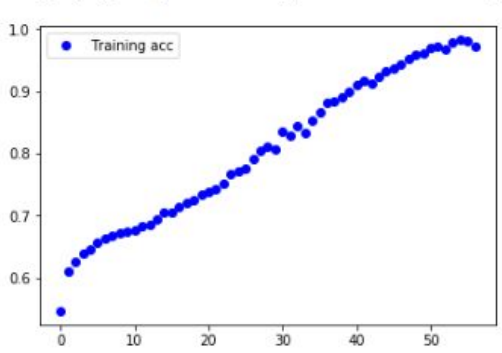
RGB Red Green Blue. 13, 14, 21, 25

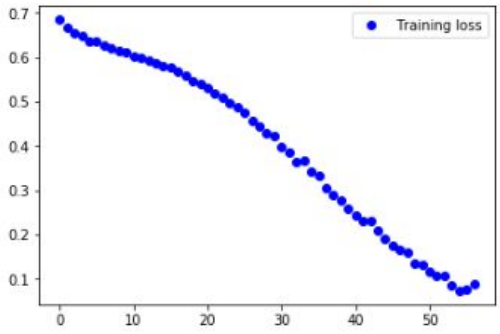
SGD Stochastic Gradient Descent. 16

APPENDIX A

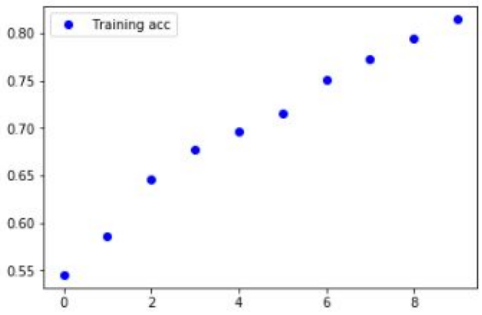
Training accuracy and training loss graphs of the three programs:

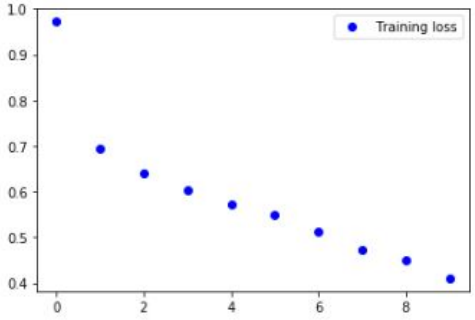
Program A



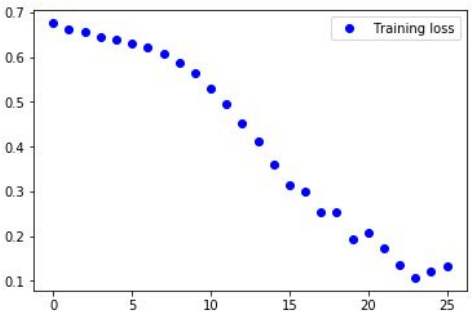


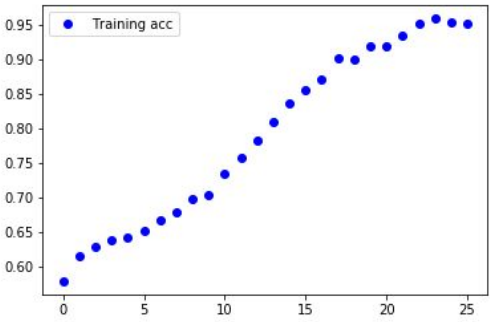
Program B





Program C





APPENDIX B

Code for the confusion matrix

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 1.3
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
cnf_matrix = confusion_matrix(y_test, predictions)
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=[0,1],
                      title='Confusion matrix, without normalization')
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=[0,1], normalize=True,
                      title='Normalized confusion matrix')
plt.show()
```

APPENDIX C

Model summary

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_4 (Conv2D)	(None, 15, 15, 256)	295168
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 256)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 512)	6423040
dense_2 (Dense)	(None, 1)	513

Total params: 6,811,393
Trainable params: 6,811,393
Non-trainable params: 0