Project sep23 cds int medical data

by:

Ricardo Teixeira

Karim Osman

# **The MRI project proposal**

This project started with a proposal to make brain MRI image classification.

We had 147 MRI images that are 3 dimensional images in grayscale.

We develop two functions to visualize images with 3 dimensions:

* the first function receives X,Y, and Z coordinates and presents 3 images with the MRI cuts in the 3 axes.
* the second function receives 1 character to the axes and the Start and Stop values and presents parallel cuts over the selected axes from the Start to Stop value over the axes.

We closed this MRI project proposal because we have very few images and we were unable to obtain the patient's agreement to use the MRI images that are mandatory by GPDR and GxP applicable to Medical Information.

# **The lung X-Ray classification**

This is also an image classification problem that is publicly available for data scientists to make image classification tests.

For the project we get:

* 3616 x-ray images from patients with Covid,
* 6012 images with Lung Opacity,
* 1345 images with Viral Pneumonia,
* 10192 images from patients with normal lung x-ray and for each image,
* the images masks to the lung area,
* an excel file with information about the images.

We implement the project as a Google colab notebook in a shared folder into Google Drive. The images and the Excel file are also loaded into the Drive shared folder.

We load the excel data into a dataframe and add information about the drive path where the images are.

## ***Image evaluation***

The first thing we did was make a function to be able to visualize the images in the notebook.

We detect that some images are grayscale other 3 Chanel and there are images with different sizes and for each image there are the correspondent Mask images for the lung area.

## ***Image preparation***

As we need to cut the images by applying the masks to let the training be focused only on the lung area and we need to normalize them, we decide pre-process the images and put them all in just one folder.

Doing the normalization as a pre-process task and save the processed images we will save time later when we will use them to train our model.

We maintain the image information about the dataframe, the classification and the path by updating the dataframe.

The pre-processing does:

* if the image is grayscale generate 3 identical images for each Chanel RGB
* normalize the pixel intensity between 0 and 255
* save the image as a JPG
* resize the image to 244\*244\*3
* apply the mask to let information only on the lung area by multiplying the image by the mask image.
* save all images in the same folder

At the end we have a dataframe that contains the image file name, the path, the classification and all pre-processed images in the same folder.

After prepare the images we obtain standardized images like this:



## ***Balanced data-set***

We need a balanced train data-set to avoid model bios.

As our images are not balanced we create a script to do it with 3 options:

* a – all images (not balanced)
* b – approximately 1400 images for each category
* c – 6000 images for each category with images repetition, that obliges to use data augmentation to avoid overfitting in the duplicated images.

after tuning the values we use a fix percentage split for the 3 data-sets

* 30% to Validation
* 10% to Test
* 60% to Train



## ***Data load to the models***

As we have the images defined in the dataframe we use iterators to load the Training, Validation, and Test data sets.

If we use the option c as we need data augmentation we define that in the generator that is used in the iterators. This is done because we are using duplicates for covid and viral pneumonia, so we randomly flip the image, zoom the image 10 %, rotate the image until 15 degrees randomly, and shift the image until 10% in any direction.

We also adjust the deep of the pixel because the model needs values between 0 and 1.

## ***Models***

As this exercice is mainly to learn we make several tests between bot of us.

| Model | Comment |
| --- | --- |
| Keras model based on lesson exercises | It has an accuracy near 68% |
| VGG 19 | Didn't converge |
| VGG 16 | Didn't converge |
| VGG 16 simplified | After training with 12 epoc and data option c it had an |
| RESTNET 50 | After training with 15,25,30,50,100 epochs we get an accuracy 69.58%,78%,73%,86%,83%, 89.69% and data option B |
| U-NET | After training with 15,12,50,100 epochs we get an accuracy 98% , 98%,88%,89.39%and data option B |

***Train Limitations***

We used the google colab as a free version, so we have time limits to use as CPU(approx 7 hour / day) and GPU (approx 2 hours / day).

do these limitations if we train using dataset Option b we can train multiple epochs one run (one continuous execution time from Google).

On the other hand, if we use the Option C, do it to its size, we can train only one or two epochs by run. This limitation obliges us to store the trained model and load it in the next day to continue the train. Like this we lose the history and we can't present the charts from the loss function and accuracy over epoch.

## Train strategies

Considering we are using Google servers' colab notebooks we have time limitation to make the training.

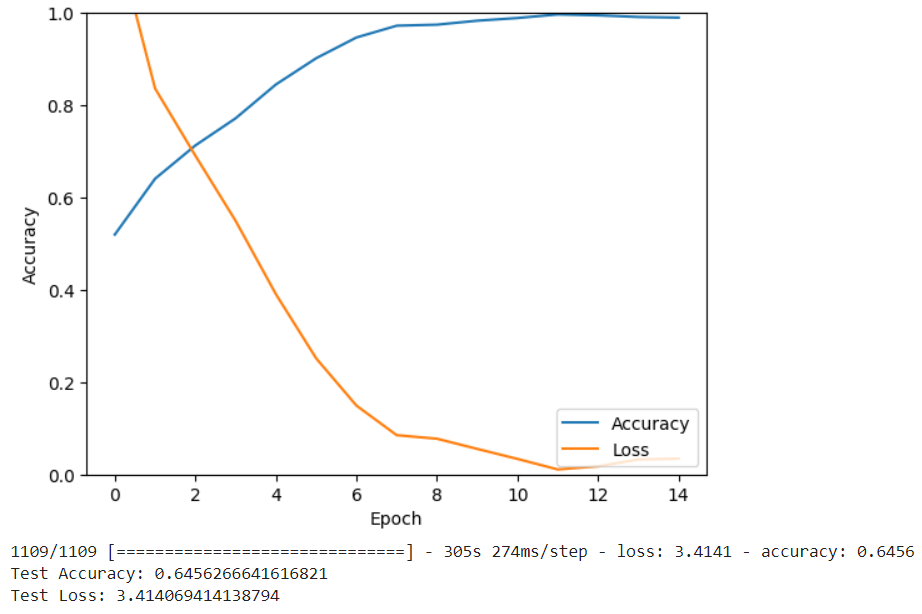
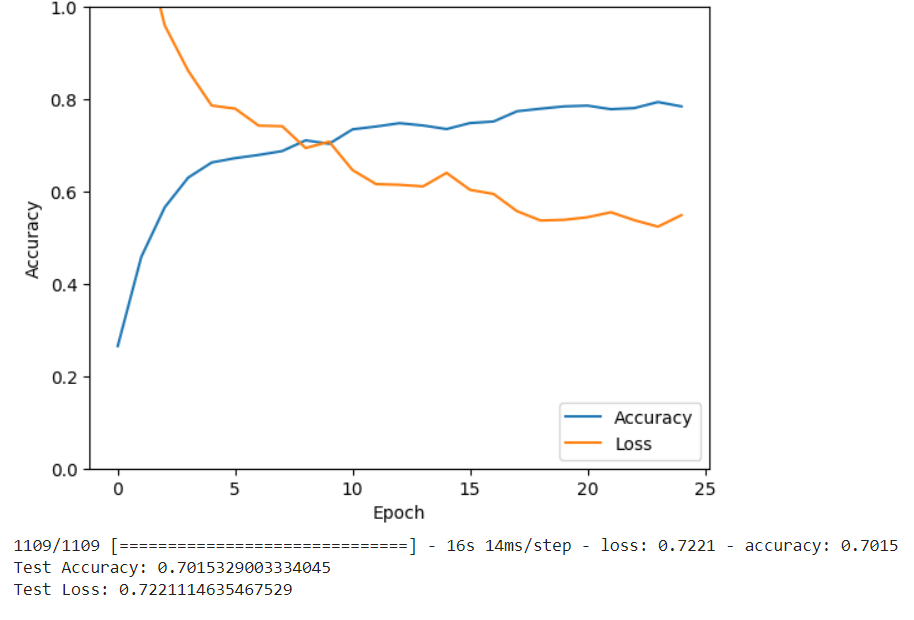
We use two different strategies:

* train with large number of epochs and use data balance selection option b (1400 images for each category)
* train with smaller number of epochs and use balance selection option c (6000 images for each category)

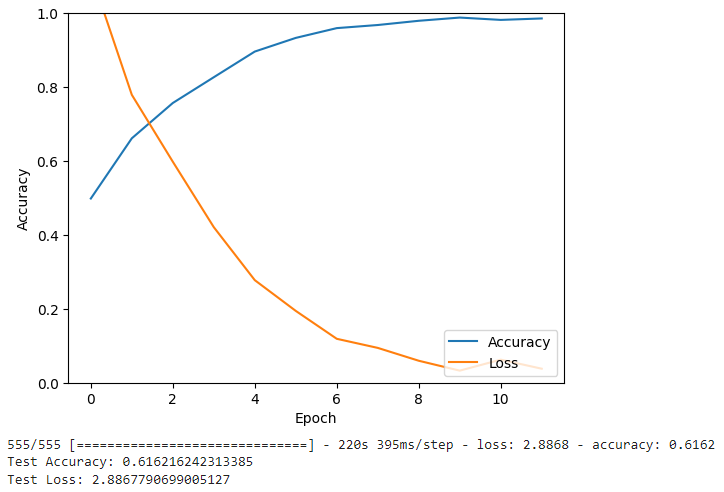
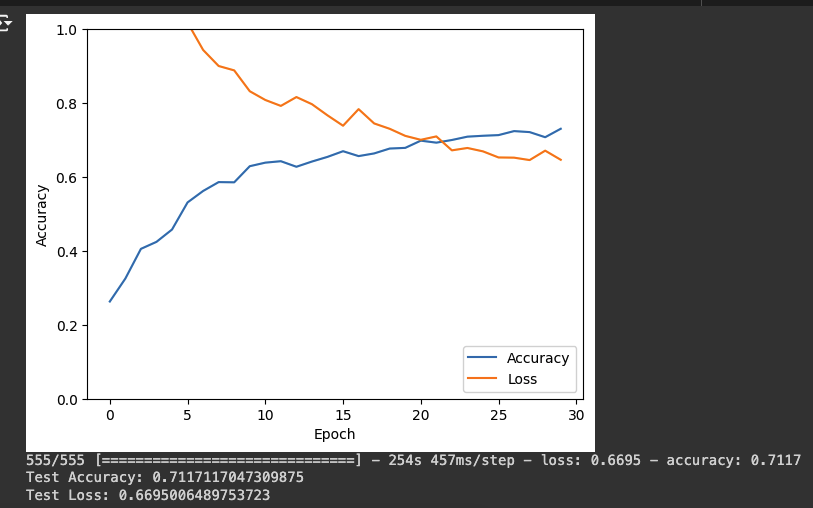
We also used callbacks to save the model and be able to reload it and retrain it.

## ***Results of Test data set with different number of epochs***

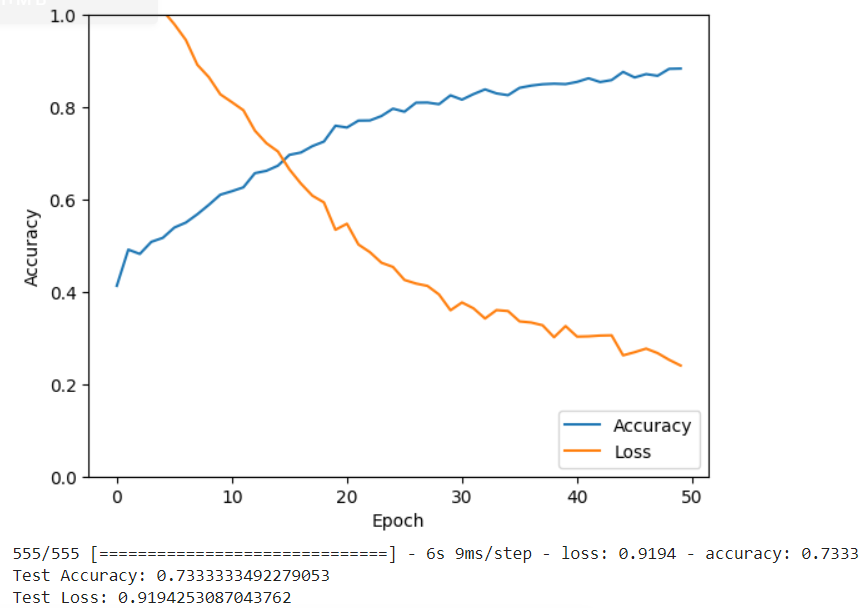
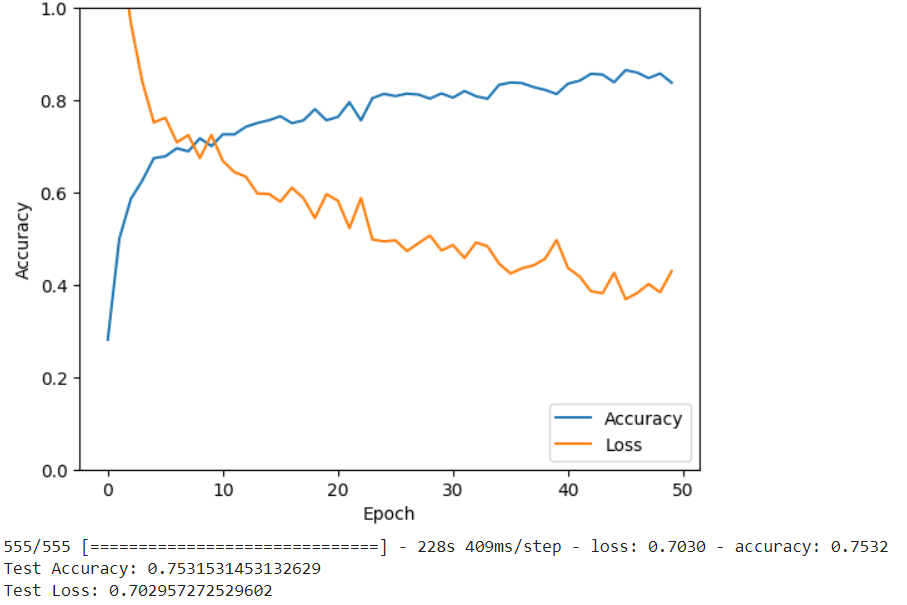
* **25 15**



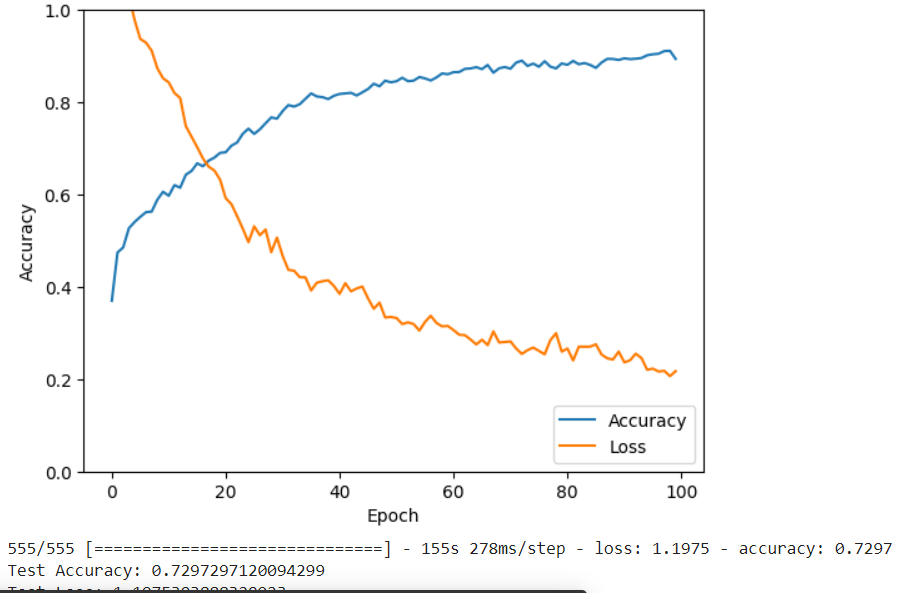
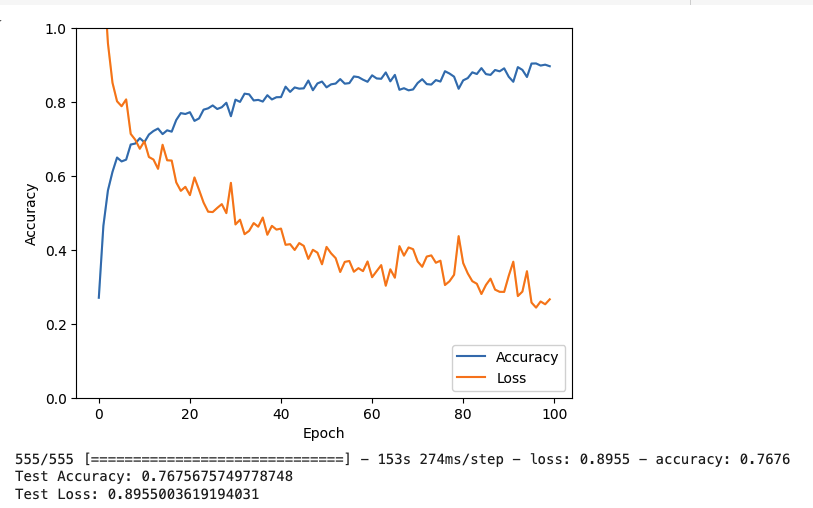
* **30 12**



* **50 50**



* **100 100**



| Model | Train conditions | Train results | Test results |
| --- | --- | --- | --- |
| Keras CNN with 1 Conv and 2 dense layers | Dataset used option b, 5 epochs |  | Test Accuracy: 0.6663661003112793  Test Loss: 1.5471841096878052 |
| Keras CNN with 1 Conv and 3 dense layers | Dataset used option b, 5 epochs |  | Test Accuracy: 0.6456266641616821  Test Loss: 1.8581470251083374 |
| Keras CNN with 2 Conv and 3 dense layers | Dataset used option b, 5 epochs |  | Test Accuracy: 0.695220947265625  Test Loss: 0.8363078832626343 |
| Keras CNN with 2 Conv and 3 dense layers | Dataset used option b, 7 epochs | loss: 0.2192 - accuracy: 0.9243 | Test Accuracy: 0.7042380571365356  Test Loss: 0.7575107216835022 |
| Keras CNN with 3 Conv and 3 dense layers | Dataset used option b, 7 epochs |  | Test Accuracy: 0.7150586247444153  Test Loss: 1.0974189043045044 |
| VGG Simplified with 7 Conv and 3 dense layers | Dataset used option c, 12 epochs | loss: 0.4189 - accuracy: 0.8331, val\_loss: 0.4776 - val\_accuracy: 0.7993 | Test Accuracy: 0.8274999856948853  Test Loss: 0.4335298538208008 |
| ResNet50 with 54 Conv and 3 dense layers | Kernel pre trained, dataset used option b, 25 epochs | loss: 0.5488 - accuracy: 0.7844 - val\_loss: 0.6447 - val\_accuracy: 0.7129 | Test Accuracy: 0.7015329003334045  Test Loss: 0.7221114635467529 |
|  | Kernel pre trained, dataset used option b, 15 epochs | loss: 0.0342 - accuracy: 0.9897 - val\_loss: 3.2312 - val\_accuracy: 0.6614 | Test Accuracy: 0.6456266641616821  Test Loss: 3.414069414138794 |
|  | Kernel pre trained, dataset used option b, 15 epochs | loss: 0.6841 - accuracy: 0.6958 - val\_loss: 0.7105 - val\_accuracy: 0.6867 | Test Accuracy: 0.6900901198387146  Test Loss: 0.7035412788391113 |
|  | Kernel pre trained, dataset used option b, 15 epochs | loss: 0.6465 - accuracy: 0.7304 - val\_loss: 0.7023 - val\_accuracy: 0.7084 | Test Accuracy: 0.7117117047309875  Test Loss: 0.6695006489753723 |
|  | Kernel pre trained, dataset used option b, 11 epochs | loss: 0.0379 - accuracy: 0.9856 - val\_loss: 2.2818 - val\_accuracy: 0.6470 | Test Accuracy: 0.616216242313385  Test Loss: 2.8867790699005127 |
| RESTNET50 MODEL | Kernel pre trained, dataset used option b, 50 epochs | loss: 0.4297 - accuracy: 0.8380 - val\_loss: 0.7030 - val\_accuracy: 0.7444 1:38 | Test Accuracy: 0.7531531453132629  Test Loss: 0.702957272529602 |
| UNet model | Kernel pre trained, dataset used option b, 50 epochs | loss: 0.2403 - accuracy: 0.8837 - val\_loss: 1.0417 - val\_accuracy: 0.7366 | Test Accuracy: 0.7333333492279053  Test Loss: 0.9194253087043762 |
| RestNet50 | Kernel pre trained, dataset used option b, 100 epochs |  | Test Accuracy: 0.7675675749778748  Test Loss: 0.8955003619194031 |
| UNet model | Kernel pre trained, dataset used option b, 100 epochs | loss: 0.2172 - accuracy: 0.8939 - val\_loss: 1.0509 - val\_accuracy: 0.7511 | Test Accuracy: 0.7297297120094299  Test Loss :  1.1975 |

# **Conclusion**

This project provided us with the opportunity to apply our acquired knowledge in a practical context. We demonstrated proficiency in image analysis using Keras, employing both standard and pre-trained models. Our approach included modifying these models to achieve improved accuracy on the test dataset. Key tasks involved preprocessing images, utilizing iterators, and leveraging data visualization to present results. We also implemented callbacks, re-trained models, and showcased our findings using Streamlit.

Understanding why certain models outperform others can be challenging. However, we are skilled in evaluating models and selecting the most suitable one for each specific data analysis task. The highest test accuracy we achieved was 82.74%, using a simplified version of the VGG model. This model was trained for 12 epochs on a dataset of 14,000 images, employing data augmentation techniques on a balanced dataset.

Of course, with unlimited GPU time, we could obtain even better results. However, given the substantial size of the dataset and our limited GPU resources, it was not feasible to train all models on the full dataset extensively.