SAPIENZA
UNIVERSITÀ DI ROMA

Deep Learning for COVID images classification

Course: Deep Learning And Applied Artificial Intelligence 2020
Professor: Emanuele Rodola
Student: Anxhelo Diko
Matricola: 1867167

1)Problem Description

The problem i am facing in this project has to do with COVID-19 CT image classification and it is an open challenge, so to better describe the problem i am referring to the challenge description on this link https://covid-ct.grand-challenge.org/CT-diagnosis-of-COVID-19/ which says the following: Coronavirus disease 2019 (COVID-19) has infected more than 1.3 million individuals all over the world and caused more than 106,000 deaths. One major hurdle in controlling the spreading of this disease is the inefficiency and shortage of medical tests. To mitigate the inefficiency and shortage of existing tests for COVID-19, we propose this competition to encourage the development of effective Deep Learning techniques to diagnose COVID-19 based on CT images.
The problem in this challenge is to classify each CT image into positive COVID-19 (the image has clinical findings of COVID-19) or negative COVID-19 (the image does not have clinical findings of COVID-19). It's a binary classification problem based on CT images.

2) Dataset

The dataset contains CT images of COVID and non COVID patients. The COVID images are in total 349. The training COVID images are extracted from papers about COVID using tools for extracting information about the pdf papers. Because of the way the images are extracted they are not of good quality and do not have all the characteristics and information that a CT image can give but based on different studies with experts it was proven that they can still be usable to identify the disease.

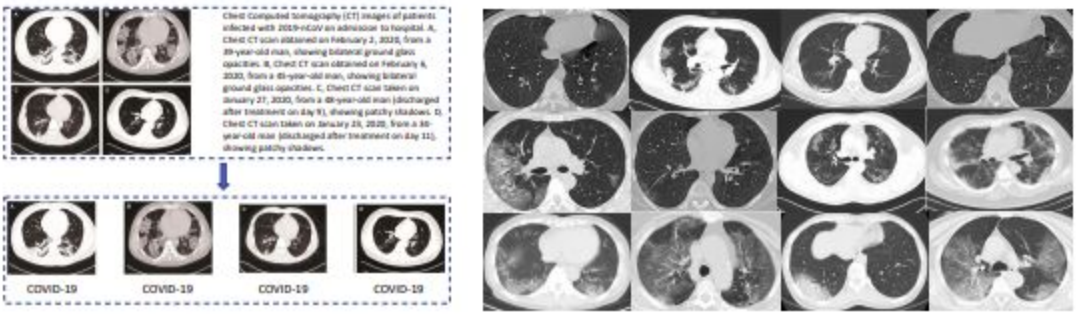

Fig. 1: Example of how images are extracted from pdf

The images of COVID are different in size, the height starts from 153 and goes up to 1853 and the width from 124 up to 1485. In the original dataset some of the images are also associated with metadata about the patients but in the challenge we have only the images. Test and validation sets for COVID images are taken from hospitals and are of a better quality. In the

dataset are also included non COVID images taken from public CT dataset like LUNA, MedPix, Radiopaedia and PubMed Central. Together with the data we were given files with image split for train, test and validation. More information about the dataset can be found here https://arxiv.org/pdf/2003.13865.pdf.

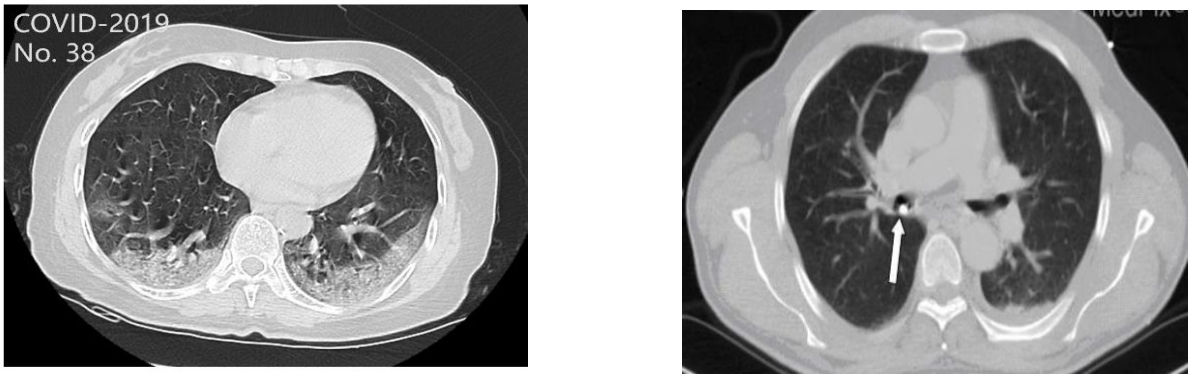

Fig 2. Example of COVID (left) and non COVID images from the dataset

3 Setting up the pipeline

Before starting to actually train the model a crucial step was building the pipeline (end to end training and evaluation) and being sure it works before going to the real training. This step is of course done after data exploration and understanding it well. In this step it was more suitable to use a simple architecture and try to get something meaningful. I tried to train it and see some meaningful values and plot the loss to identify if we are doing it right.

4 Overfit to prove that the model can learn

Before proceeding with model selection and fine tuning I tried to overfit a single batch of the model and visualize images, ground truth and predicted labels of that batch to see if there was a bug or not at that phase. The results were the following:

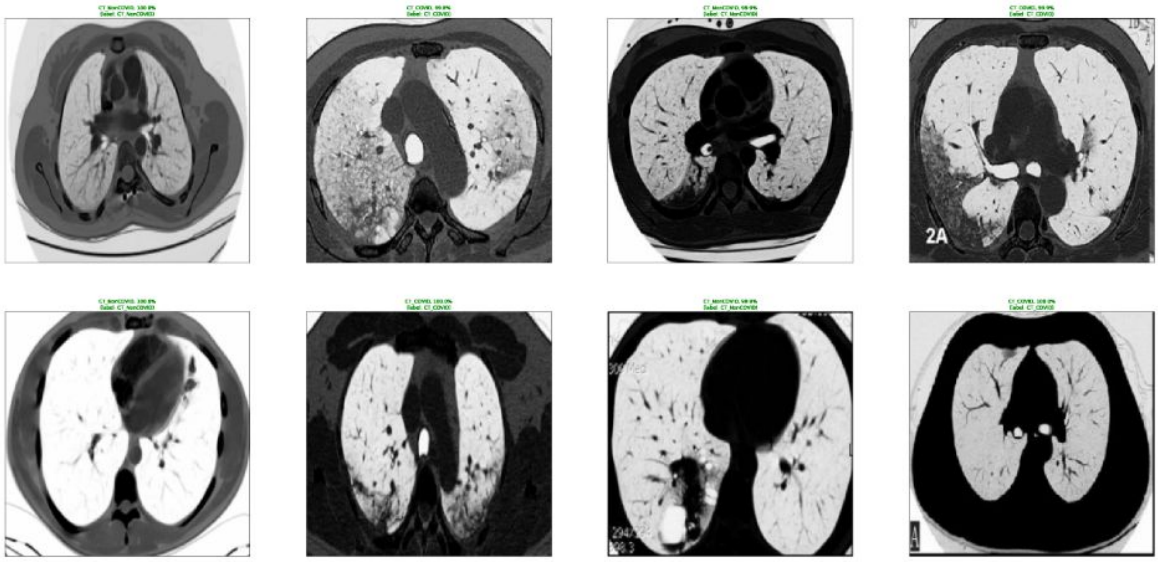

Fig. 1: Predictions during the overfitting phase

The image is not clear but the green text above each image of the batch shows that the predicted label and the ground truth matches each other. So until now there was no bug and can freely proceed to the next step.

5 Data augmentation

This part was crucial, data augmentation was very important for making my models learn. I tried different augmentation techniques like smoothing (gaussian), rotation, cropping, etc during the training. I changed them a lot until I found something that really gave a boost to the learning process. The final data augmentation techniques that i used were:

a) Random Resized Crop (only for training images): This way i can feed more different examples to the network meanwhile the semantic information is not lost. This was far more useful than Center Crop.

b) Random Horizontal Flip (with p=0.7 , default 0.5)

c) Random Vertical Flip (with p=0.7 , default 0.5)

d) Random Rotation

e) Color Jitter (randomly changes the brightness and contrast of the image)

For normalization i tried two different statistics, the one of the dataset and the one of imagenet since i will be using a pretrained network and fine tune it. The one giving better results obviously was the normalization with dataset statistics.

6) Main architecture

Choosing the main architecture took a lot of testing. The architectures that I tried were ResNet18, ResNet50, AlexNet, DenseNet121, DenseNet161 and DenseNet169. I started with the smaller networks since it is more memory efficient but in the end the results they were producing were lower compared to the bigger architectures and in the end the chosen architecture was DenseNet169 (also proven to work in COVID classification from many papers but not on this dataset).

7) Regularization

Densenet architecture already includes batch normalization and i tried to add dropout but it resulted not in a good idea, maybe my case was different or drop out and batch normalization does not really work together.

8) Training

The training phase of the network went through three subphases which I will explain separately. The first subphase was supervised and consisted in fine tuning the DenseNet up to the best result it could give. The second phase was self-supervised using contrastive learning from the fine tuned network of the previous step and the third phase was the addition of classification layers to the self-supervised architecture and training those layers to transform the network from a feature extraction network into a classifier.

First Subphase

As stated before the first phase consists in fine tuning the DenseNet architecture trying to get the best result out of it. In the fine tuning phase I tried to fine tune different layers until I got some good results. Started from the bottom layers and went up to denseblock3 and denseblock4. So in total i fine tuned the classifier part, denseblock4 and denseblock3 with a batch size of 128 (after a lot of testing going higher would give a bad result). The optimizer i used was Stochastic Gradient Descent (SGD), i also tried RMSProp which seems to learn far more faster than SGD but SGD would converge better producing better results but very very slow compared to RMSProp. The loss function was CrossEntropy. The starting learning rate was 0.007 and with the learning rate scheduler it was multiplied by 0.5 every 400 epochs. The total number of epochs were 3000 and the plotted statistics are as follow:

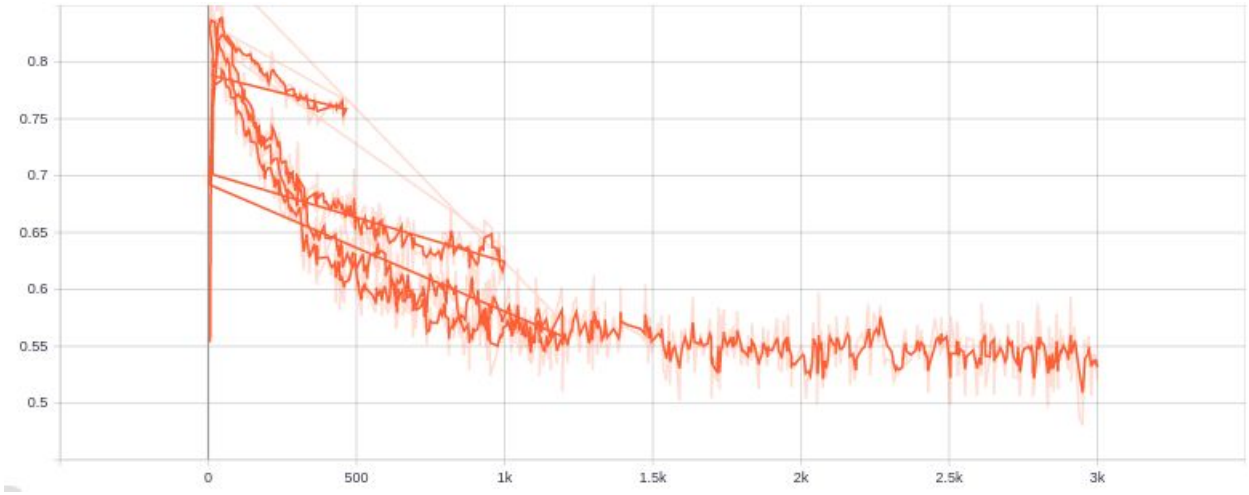

Fig 2. Training loss for 400, 1000, 1200 and 3000 epochs

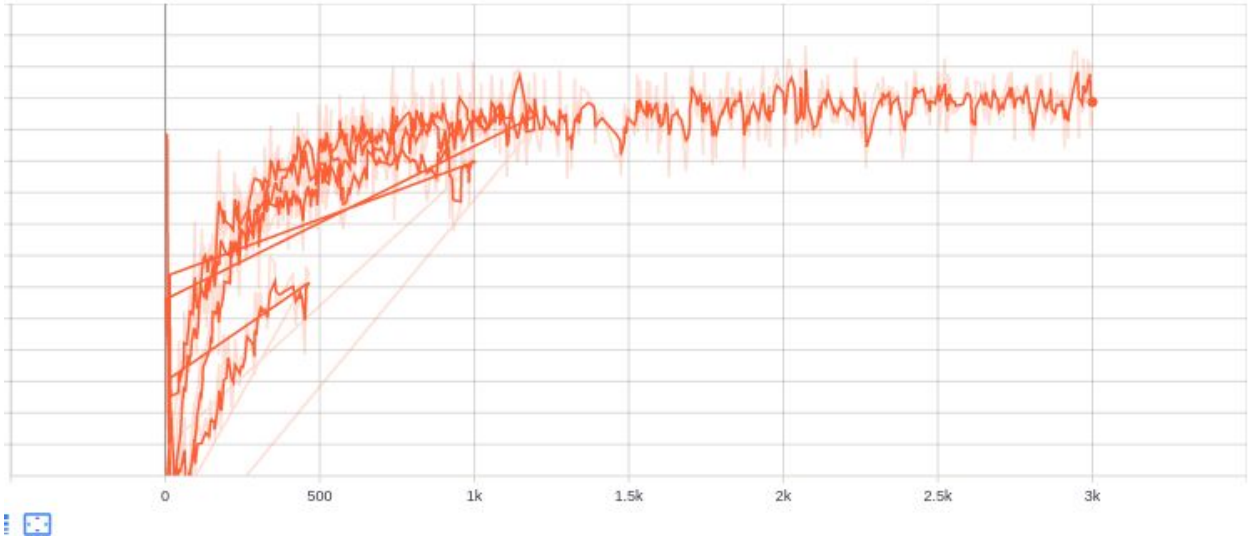

Fig 3. Training AUC (88% for the 3000 epochs run).

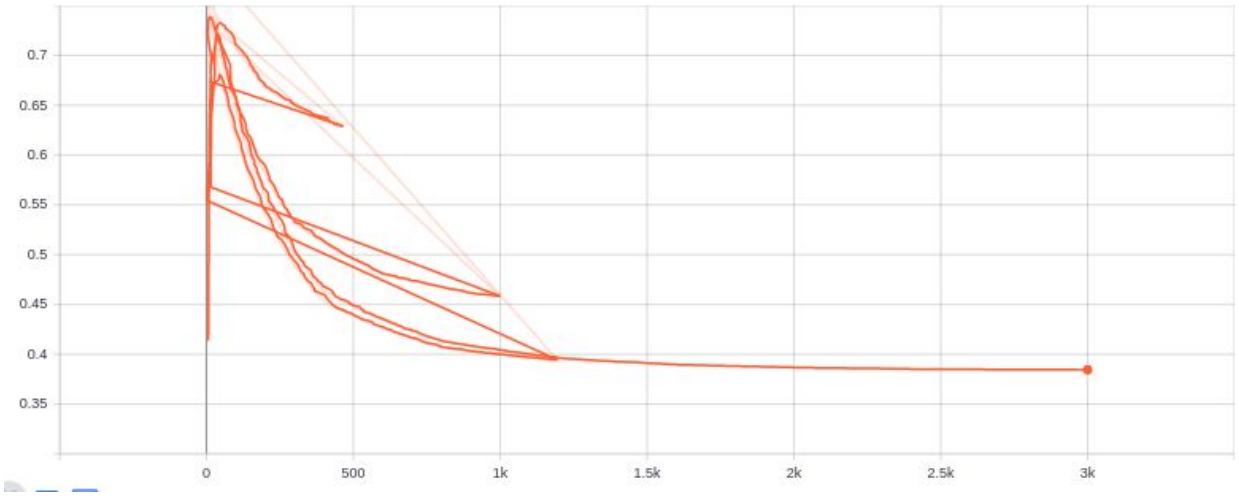

Fig 4 : Validation Loss 400, 1000, 1200 and 3000 epochs

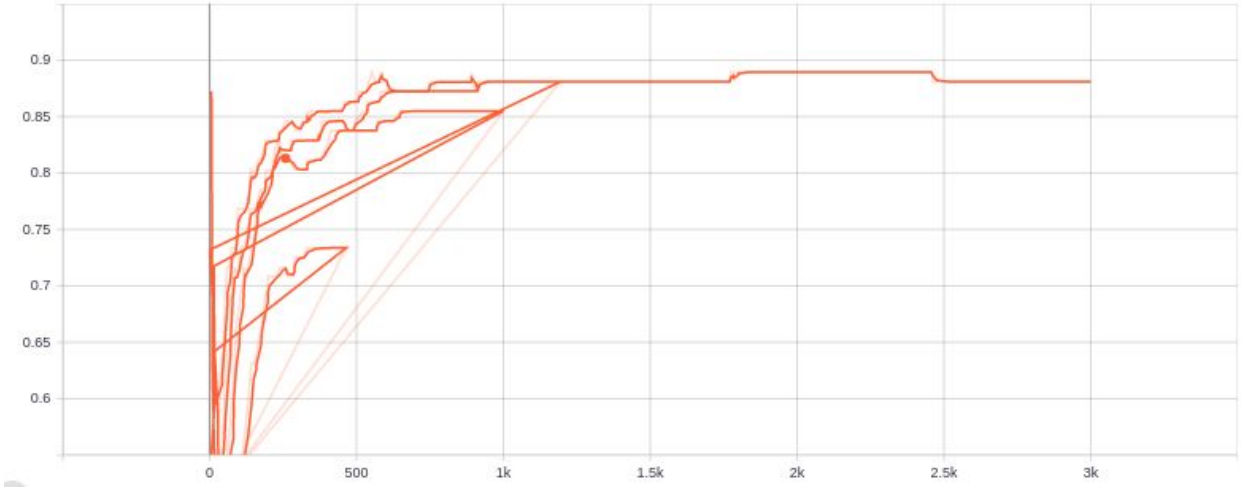

Fig 5: Validation AUC 400, 1000, 1200 and 3000 epochs.

The testing accuracy of this phase was 89%, f1-score of 90% and AUC 89%.

Second Phase

In the second phase I went for contrastive learning. The base network was the one I fine tuned in the previous subphase and removed the classifier to use it as a feature extractor and also added 3 linear layers and as a final result it produces embeddings (l2 normalized) of length 512 units. The loss function used was triplet loss that takes triplets of images (anchor, positive and negative). The approaches for choosing the triplets were two, totally randomly (which produced a lot of hard triplets) and selecting an anchor, creating the positive with an aggressive augmentation (not as the one used for validation) and randomly selecting the negative image. In this phase i was limited from the hardware, i could not go further than 32 images for batch size if i would fine tune more than denseblock4 and only with denseblock4 i could go up to 64 images

per batch which actually for certain tasks is low. The network was trained for 300 epoch with learning rate 0.0005 and SGD as optimizer. The used margin for the triplet loss was 0.5.

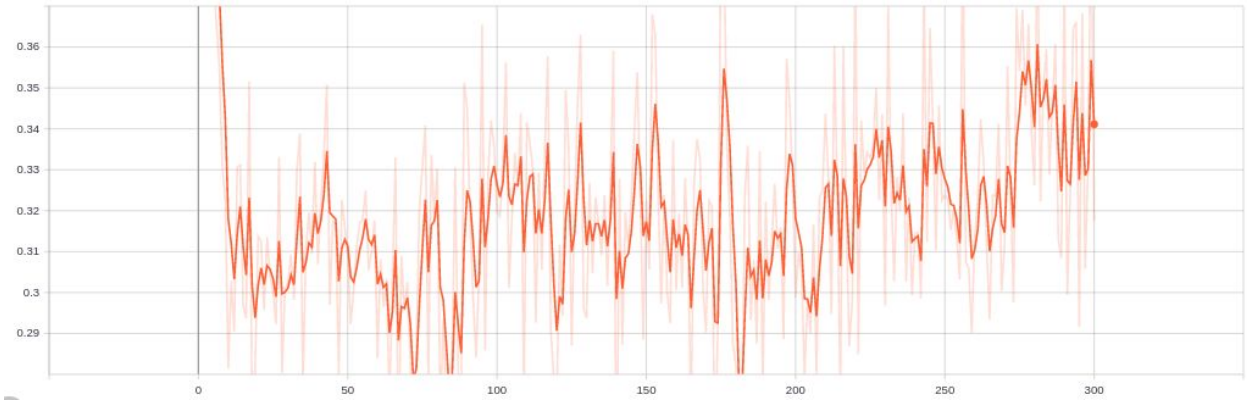

Fig 6. Training loss TriNet

Fig 7. Validation loss TriNet

From the images we can see that the training loss is very unstable meanwhile the one in validation is more stable, the cause of this in my opinion might be because in training we have more hard tripletes because everything is selected randomly and for evaluation the positive image is generated from the augmentation of the anchor.

Third phase

The third phase now is straight forward, i added another linear layer to the previously trained network and trained that layer also fine tuned some of the bottom layers which helped in achieving better results. The optimizer again was SGD, CrossEntropy as a criterion, 220 epochs and learning rate of 0.0003.

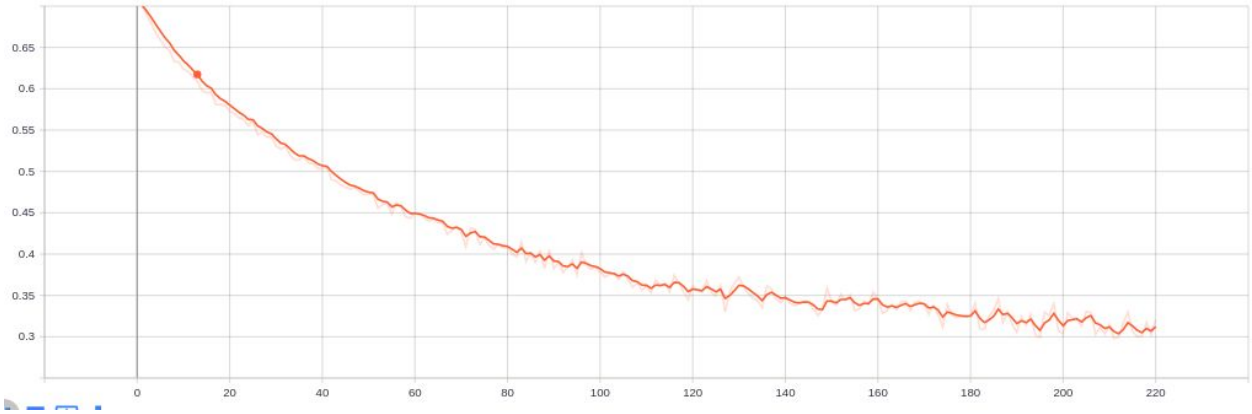

Fig 8: Training loss final

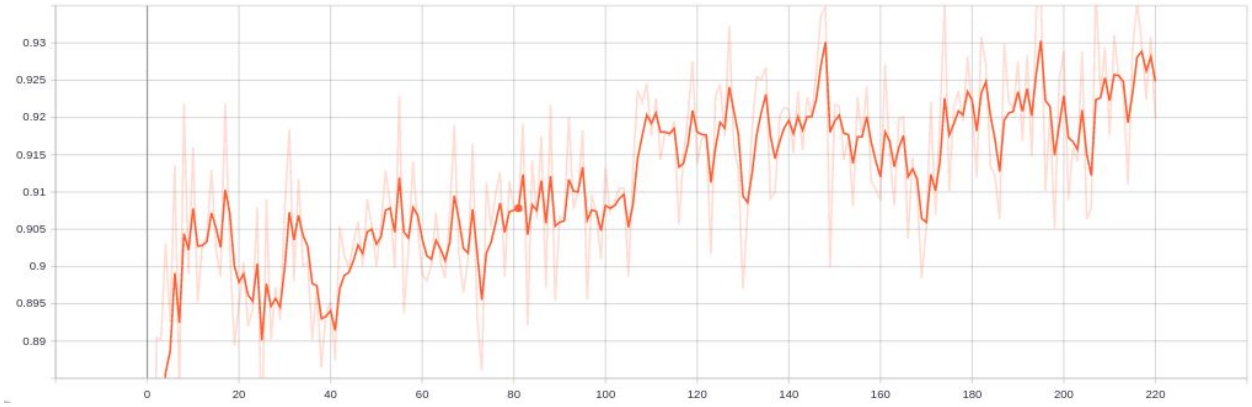

Fig 9: Training AUC

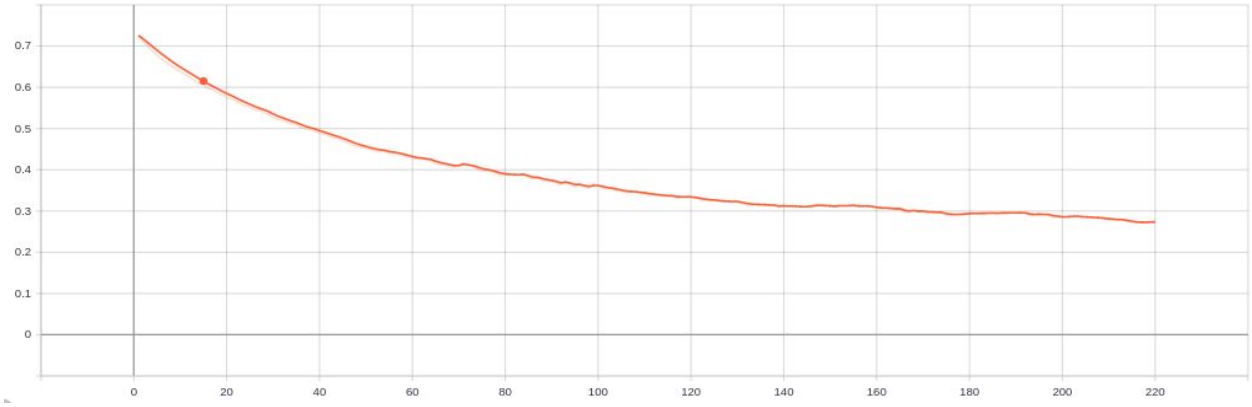

Fig 10: Validation loss

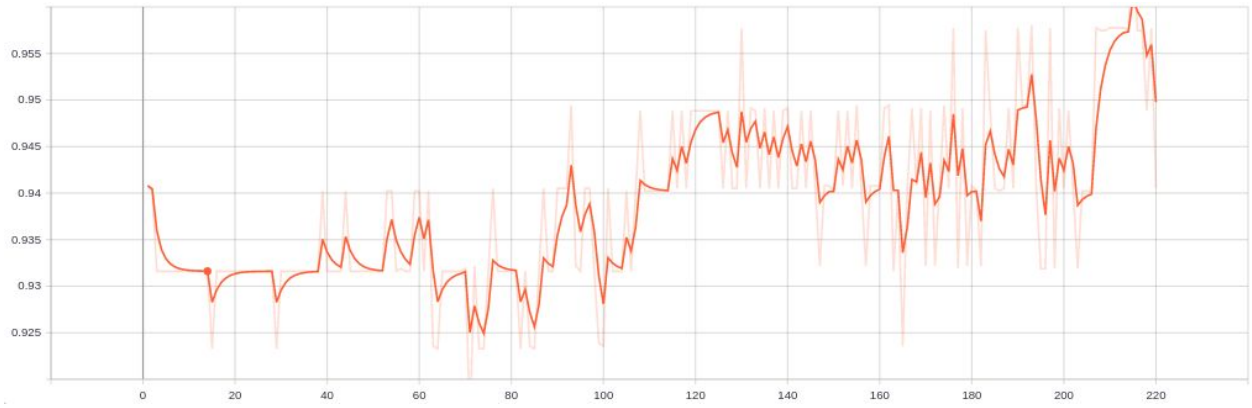

Fig. 11: Validation AUC

Final test results were the following:

```
Test Accuracy: 0.9408866995073891
TEST F1-score: 0.9411764705882353
TEST AUC: 0.9421768707482994
```

9) Hyperparameters

I did not use any standard hyperparameter tuning strategy like grid search or random search because it was very costly and would require many days to find the best parameters for all the three training phases so I went with the approach of manually trying different values.