Machine Learning Engineer Nanodegree

Capstone Project

Ahmed Wael
March 12nd, 2019

I. Definition

Project Overview

Pneumonia is a widely known in lung infection that affect the air sacs. While not very acute if diagnosed in the early stages, it can be fatal if not. According to American Thoracic Society, pneumonia is the world leading cause of death among children under 5 years old, killing over 2,400 children a day in 2015, and accountable for 16% of all children under 5 years old deaths. Moreover, nearly 1 million adults get hospitalized and 50,000 of them die every year in the US alone from misdiagnosis or being diagnosed very late. This number gets higher in developing countries such as Egypt as 90% of deaths of pneumonia's deaths are in these countries.

Knowing that the treatment of this infection is fairly available and cheap, the threat of pneumonia lies in discovering it in later stages or being misdiagnosed for normal fever. The diagnosis is usually done using X-Ray chest image performed by a physician. By collecting enough images of X-Ray, a machine learning algorithm can be used to train on these images and classify future images in nearly no time if this person has pneumonia or not.

The dataset the was used to train is available on Kaggle with the name 'Chest X-Ray Images (Pneumonia). The dataset is available for public use and can be easily download from the website. It consists of 5,865 labeled images (JPEG) split into 2 categories which are normal and infected X-Ray images. Furthermore, the dataset is organized into 3 folders which are train, test, and val. According to the data scientist who uploaded the dataset, this dataset was collected and preprocessed from Guangzhou, Women and Children's Medical Center as part of patients' routine clinical care. It was screened for quality control by removing unreadable or low-quality images. After that, the labeling was performed by two expert physicians, and validated by a third one.

Problem Statement

The goal is to create a machine learning model that has X-Ray image as input and returns the class of it as an output, by being either normal or having pneumonia infection. The tasks involved are the following:

1. Download the dataset generically, by using a single command.
2. Explore the dataset statistically and visually.
3. Preprocess the dataset to remove any outliers (if any), resize all images, and link each image to its label.
4. Perform basic data augmentation to have more training examples (double the size).
5. Use k-fold validation technique to decrease overfitting.
6. Train and test the dataset on a benchmark model.
7. Experimenting with different models and fine-tuning.
8. Output statistics and visualizations of the different models.

The chosen machine learning algorithm that will be used is Convolution Neural Network (CNN), as it has been proven that it works better that any other algorithm on image dataset.

Metrics

While accuracy is the standard metric that is used to evaluate any machine learning model, it is not the most important metric in this case. The most important metric is recall, because the model should be more sensitive to false negatives – the sick patients that were diagnosed as healthy-, more than the false positives – the healthy people that were diagnosed as sick-. More metrics can be used such as precision, but in this case, the most important metric is recall.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{All\ the\ examples}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

II. Analysis

Data Exploration

The dataset consists of 5,856 labeled images. The images vary in size and the number of pixels, which can be a huge problem for the CNN model. Also, while the dataset contains a fair number of images to train the model, the dataset split is very unbalanced. The number of infected or sick patients is triple the number of normal healthy people. While this is normal in medical datasets because of the nature of the origin of them, this can be very misleading if accuracy or precision metrices are used.

Also, the number of training images is 5216, while the number of testing images is 624, and the number of validation images is 16. If the dataset is used as it is, the validation accuracy will not be statistically sufficient and the model will be very misleading and inconsistent.

Moreover, while the images appear to be in gray level, they are RGB when plotted. Choosing to work with the RGB images as they are or converting them to gray level will be discussed in the methodology.

Finally, note that the aspect ratio was set to auto for the sake of visualization here. However, the images vary in size.

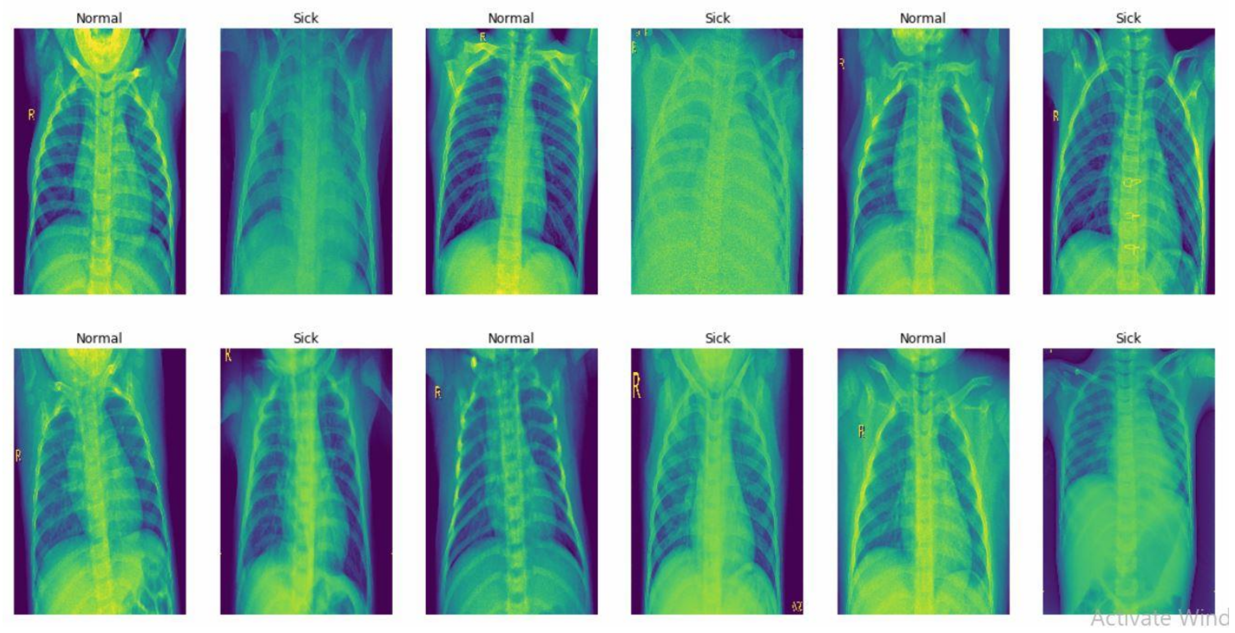


Figure 1: Few plotted X-Ray images

Exploratory Visualization

The bar plot below shows how the classes are distrusted among the images. This clearly agrees with the unbalancing issues that was discussed in the previous section.

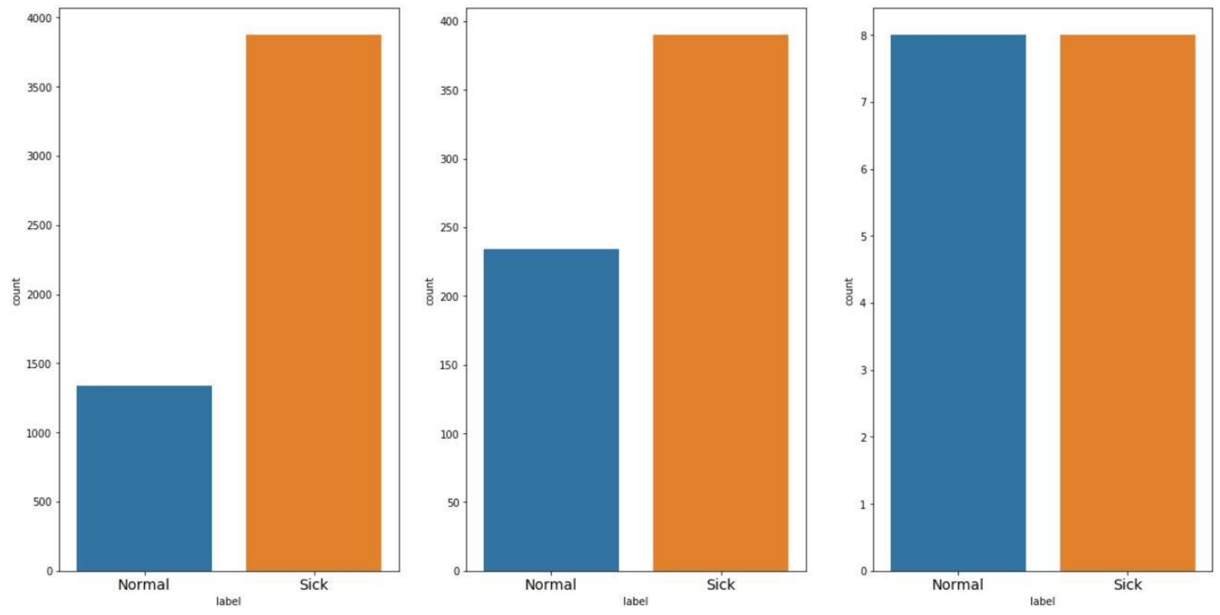


Figure 2: Condition Vs counts for each dataset folder

It can be seen that the most of the dataset lies in the training folder while there are nearly no validation images. Also, it is clear that the training set is very biased towards the sick than the healthy people.

Algorithms and Techniques

The machine learning classifier that was used is a Convolutional Neural Network, which is a state-of-the-art algorithm used extensively with image datasets. The only drawback of CNN is that it needs a large amount of training data when compared with other machine learning algorithms. However, this is not a problem here as the dataset is large enough and thus has a high capacity that the CNN requires.

The algorithm outputs a probability from 0 to 1 for each class by first convert the labels from [0,1] to be [01,10] in order to use the SoftMax activation function in the final layer of the CNN. This outperforms using the labels as they are with a sigmoid activation function as because there are two outputs which are independent of one another .

The following parameters can be tuned to optimize the classifier:

1. Training Parameters:
    a. Number of epochs (how many times the whole dataset was trained)
    b. Batch size (as training the dataset as a whole is memory inefficient and can be some times impossible)
    c. Optimizer (Adam and RMSprop are the most used optimizers, so one of them will be used here).
    d. Learning rate (how fast to learn)
    e. Weight decay (prevents overfitting as it is a regularization technique)
2. Neural Network architecture:
    a. Number of layers
    b. Layer types (convolutional, normalization, pooling, fully-connected)
    c. Layer parameters (number of neurons, activation function, padding, filter size)

The parameters specified above are obtained using experimenting with different combinations. However, there are some heuristics that can be used. For example, the number of epochs can be from 20 to 100 or even more, but with the use of early stopping technique, the model will stop training if a specified number of epochs passed and the model is not getting better results. Also, the learning rate is usually between 0.0001 and 0.01, but also with the use of weight decay, the learning rate can be dynamic so it would decrease after each epoch.

The architecture specified above can vary a lot and it is a matter of design. This design should take into respect that the number of layers should be big enough to capture more complex features in the image, hence the name *deep learning*. On the other hand, increasing the number of neurons in each layer can increase overfitting as if fails to generalize well [ ] . Also, the activation function that is used in most of the current research is ReLU with the exception of the final layer with a SoftMax layer is used. This is because ReLU is a piece-wise activation function, which is more prone to overfitting than sigmoid, and the final layer should be the output probabilities, hence the use of SoftMax. Finally, the layers are usually convolutional at the beginning with pooling and normalization in the middle, and the last few layers are few fully connected layers with normalization layers in between.

Benchmark

An ideal benchmark model for this problem is a shallow neural network with one or two hidden layers. The default hyperparameters will be used in this model.

The architecture of the benchmark model is as following:

1. Two Conv2D layers with ReLU activation function, 32 filters and a filter size of (3,3) to capture the most basic features that any human eye can see.
2. One Global Average Pooling layer to decrease the dimensionality.
3. One dense layer with 2 neurons corresponding to output, with a SoftMax activation function.
4. The optimizer used is Adam with no hyperparameters tuning.
5. The loss is categorical cross entropy which is used with categorical inputs when one-hot encoded.
6. The metric is both the categorical accuracy and the recall.

By using the specified model, the results obtained show how much it is difficult for the human eye (*the benchmark model*) to detect the infection as the results are as following:

1. Accuracy: 72.9%
2. Recall: 73%
3. Precision: 100% (which is misleading as all the images were classified as positives).

III. Methodology

Data Preprocessing

The preprocessing done in the first part of the notebook consists of the following steps:

1. Download the dataset using *wget* command as the training was done on a google cloud instance.
2. Unzip the downloaded dataset using *zipfile* library.
3. Get all the directories paths of the dataset folder, which are six in total as there are two per each sub-dataset. This was done using *Path* library.

4. Get the absolute path for all the images and label each image with its corresponding class.
5. Append all the dataset into one data frame.
6. Loop through the images and resize all of them to be (256,256)
7. Normalize all the images by dividing each one by 255 to have pixels values from 0 to 1. This would help reducing the weights.
8. Perform basic data augmentation by flipping the images from left to right so the dataset would double from 5,800 to nearly 1,100.
9. Encode the label of each image by using one hot encoding.
10. Split the dataset to be a training sub-dataset and test sub-dataset, with a ratio of 75% to 25%, thus getting rid of the imbalance problem of the different sub-dataset in the beginning.

Implementation

For the implementation process, the classifier was trained on the processed training data, and validated on the validation data. The training validation procedure was done as following:

1. All the dataset was saved into one data frame.

2. The dataset was split using *sklearn* split_train_test function, with the test set equal 25% of the whole dataset.

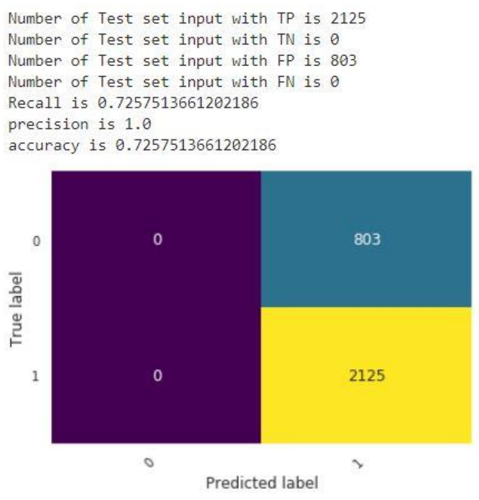
3. A loop was implemented which lasted k times (k-folds), where each iteration the training data was shuffled and split using the same *sklearn* function, but this time it is split to training and validation. Each iteration, the model is trained on the training dataset and validated on the validation set which is 10% of the training data. Finally, the best weights are saved so they can be used for testing and plotting.

4. The test dataset is used after training and validating to check if the model generalizes well or not. Accuracy, precision, and recall are all implemented using a helper function which predicts the output on the test set, and counts the number of true positives, true negatives, false positives, false negatives, and thus get the metrics values. Finally, a confusion matrix is plotted for visualization.

5. The model history is used to plot the metrics vs the epochs and the train accuracy vs the validation accuracy. This is important to check if the model is underfitting or overfitting.

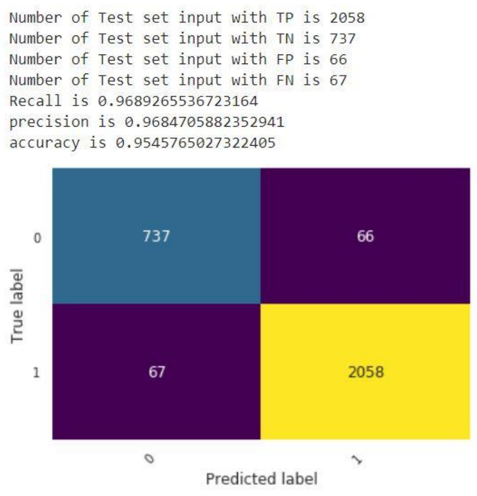
Refinement

While the benchmark model achieved 73% on recall and 73% on accuracy, the first model achieved nearly 73% on both metrics also as shown below.



```
Number of Test set input with TP is 2125
Number of Test set input with TN is 0
Number of Test set input with FP is 803
Number of Test set input with FN is 0
Recall is 0.7257513661202186
precision is 1.0
accuracy is 0.7257513661202186
```

The second model achieved 96.8% on recall as shown below.



```
Number of Test set input with TP is 2058
Number of Test set input with TN is 737
Number of Test set input with FP is 66
Number of Test set input with FN is 67
Recall is 0.9689265536723164
precision is 0.9684705882352941
accuracy is 0.9545765027322405
```

This was successfully done using the following:

1. Fine tuning the optimizer as eventually Adam optimizer was used.

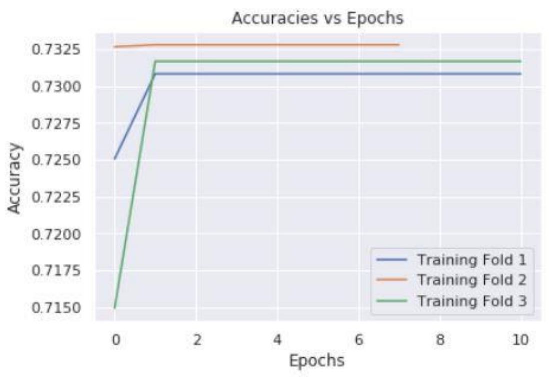
2. The optimizer was used with a weight decay which prevents overfitting as the weight decay increase when the training and validation losses diverges too much.

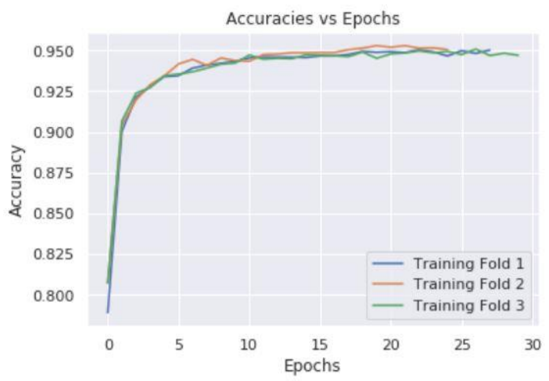
3. By adding dropout layer in the end of the networks, which prevents overfitting and makes the model reach better minima.

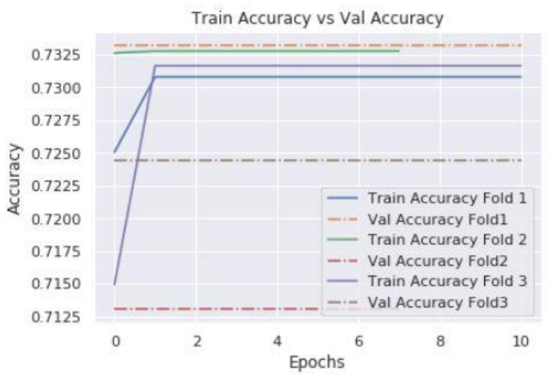
4. Adding a dynamic learning rate to converge faster.

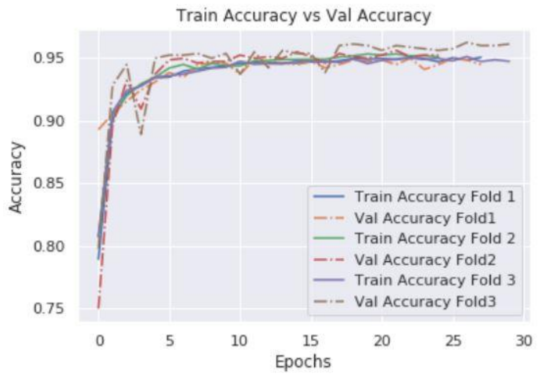
All the numbers used for hyperparameters are heuristics at the beginning and it is tuned slowly until it reaches the capacity of the model. After that, the best solution is to increase the model, and thus increase the capacity.

The following plots are obtained from the three different models experimented with, the first model, and the second and final model.









It’s clear that the second model achieve higher accuracy and generalizes much better than the first one.

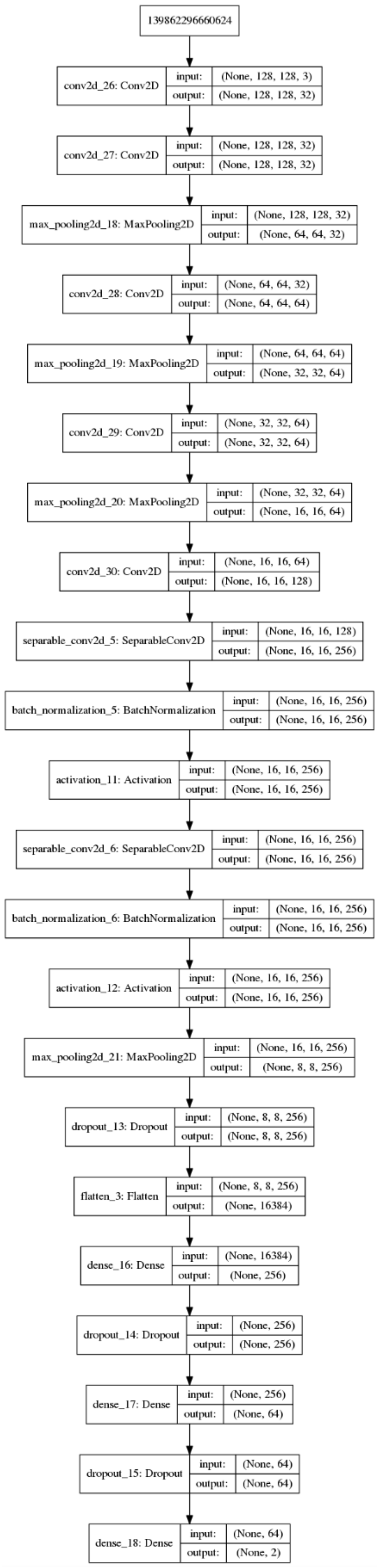
IV. Results

Model Evaluation and Validation

The validation and data splitting procedure were explained in the data preprocessing section. Plotting the training and validation losses for each epoch shows if the model is underfitting, overfitting, or fitting the data perfectly.

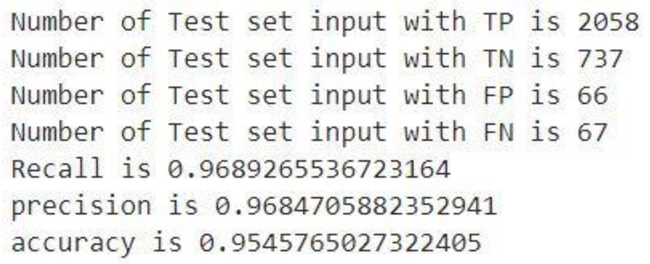
The final model used is shown below. The details of the model are as following:

1. The shape of the filter of the convolutional layer is 3*3.
2. Same padding was used along the network.
3. The number of filters for the convolutional layer start from 32 to 128, in order to capture more complex features.
4. Max Pooling layers are used between every one or two convolutional layers in order to reduce the number of parameters.
5. Separable convolutional layers are used after a stack of 3-4 conv and max pool, as they perform very well in other image classification problems.
6. Batch normalization was used before each activation function of the separable convolutional layer, in order to normalize and prevent weights explosion.
7. Flatten layer is added to act as a fully connected layer that will be feed to the dense layers.
8. Three dense layers was used with drop-out layers in between to prevent overfitting.
9. The final dense layer has a SoftMax activation function opposed to the ReLU activation function used along the network. This is because it is the output layer, with only two neurons.
10. The network with trained for 30 epochs on 3 different folds.

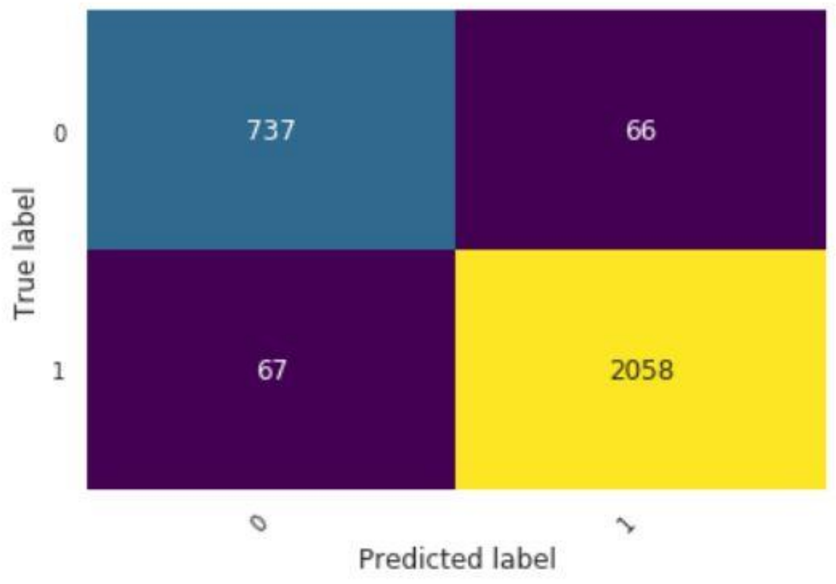

139862296660624
conv2d_26: Conv2D | input: | (None, 128, 128, 3) | output: | (None, 128, 128, 32)
conv2d_27: Conv2D | input: | (None, 128, 128, 32) | output: | (None, 128, 128, 32)
max_pooling2d_18: MaxPooling2D | input: | (None, 128, 128, 32) | output: | (None, 64, 64, 32)
conv2d_28: Conv2D | input: | (None, 64, 64, 32) | output: | (None, 64, 64, 64)
max_pooling2d_19: MaxPooling2D | input: | (None, 64, 64, 64) | output: | (None, 32, 32, 64)
conv2d_29: Conv2D | input: | (None, 32, 32, 64) | output: | (None, 32, 32, 64)
max_pooling2d_20: MaxPooling2D | input: | (None, 32, 32, 64) | output: | (None, 16, 16, 64)
conv2d_30: Conv2D | input: | (None, 16, 16, 64) | output: | (None, 16, 16, 128)
separable_conv2d_5: SeparableConv2D | input: | (None, 16, 16, 128) | output: | (None, 16, 16, 256)
batch_normalization_5: BatchNormalization | input: | (None, 16, 16, 256) | output: | (None, 16, 16, 256)
activation_11: Activation | input: | (None, 16, 16, 256) | output: | (None, 16, 16, 256)
separable_conv2d_6: SeparableConv2D | input: | (None, 16, 16, 256) | output: | (None, 16, 16, 256)
batch_normalization_6: BatchNormalization | input: | (None, 16, 16, 256) | output: | (None, 16, 16, 256)
activation_12: Activation | input: | (None, 16, 16, 256) | output: | (None, 16, 16, 256)
max_pooling2d_21: MaxPooling2D | input: | (None, 16, 16, 256) | output: | (None, 8, 8, 256)
dropout_13: Dropout | input: | (None, 8, 8, 256) | output: | (None, 8, 8, 256)
flatten_3: Flatten | input: | (None, 8, 8, 256) | output: | (None, 16384)
dense_16: Dense | input: | (None, 16384) | output: | (None, 256)
dropout_14: Dropout | input: | (None, 256) | output: | (None, 256)
dense_17: Dense | input: | (None, 256) | output: | (None, 64)
dropout_15: Dropout | input: | (None, 64) | output: | (None, 64)
dense_18: Dense | input: | (None, 64) | output: | (None, 2)

Final Model Arch.

To check that the model performs well, it was tested on the test set and the results are shown below.



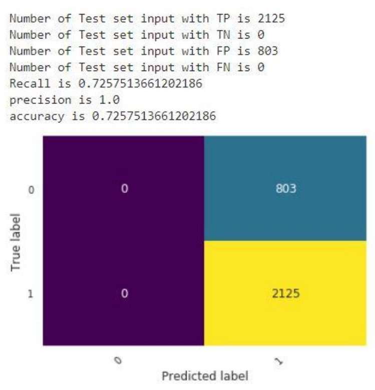
```
Number of Test set input with TP is 2058
Number of Test set input with TN is 737
Number of Test set input with FP is 66
Number of Test set input with FN is 67
Recall is 0.9689265536723164
precision is 0.9684705882352941
accuracy is 0.9545765027322405
```



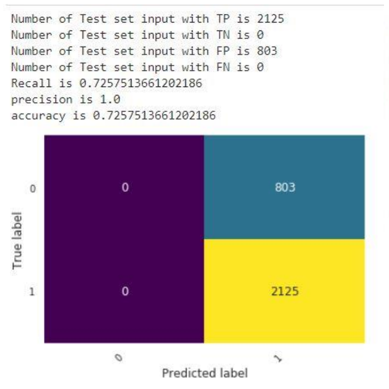
Justification

Putting the statistics and the visualization of the three models beside each other’s, shows how much the final model outperforms the benchmark model, especially on recall, which the most important metric.

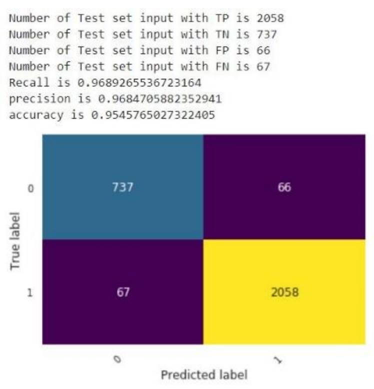
The final model reaches a recall of 96.89% which is nearly perfect.



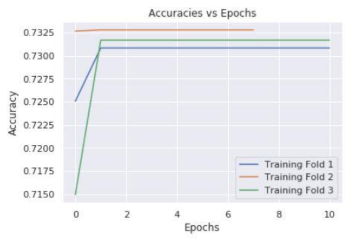
```
Number of Test set input with TP is 2125
Number of Test set input with TN is 0
Number of Test set input with FP is 803
Number of Test set input with FN is 0
Recall is 0.7257513661202186
precision is 1.0
accuracy is 0.7257513661202186
```

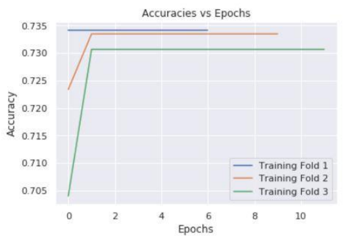


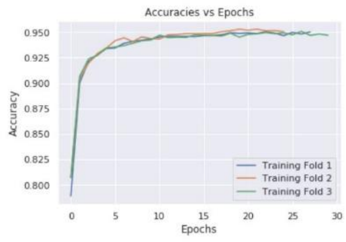
```
Number of Test set input with TP is 2125
Number of Test set input with TN is 0
Number of Test set input with FP is 803
Number of Test set input with FN is 0
Recall is 0.7257513661202186
precision is 1.0
accuracy is 0.7257513661202186
```

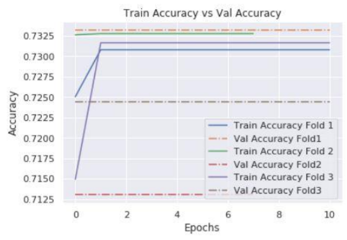


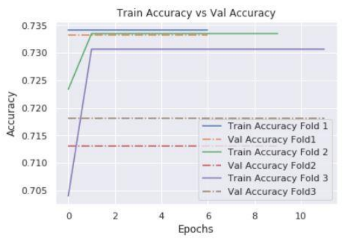
```
Number of Test set input with TP is 2058
Number of Test set input with TN is 737
Number of Test set input with FP is 66
Number of Test set input with FN is 67
Recall is 0.9689265536723164
precision is 0.9684705882352941
accuracy is 0.9545765027322405
```

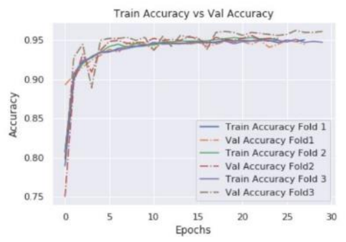








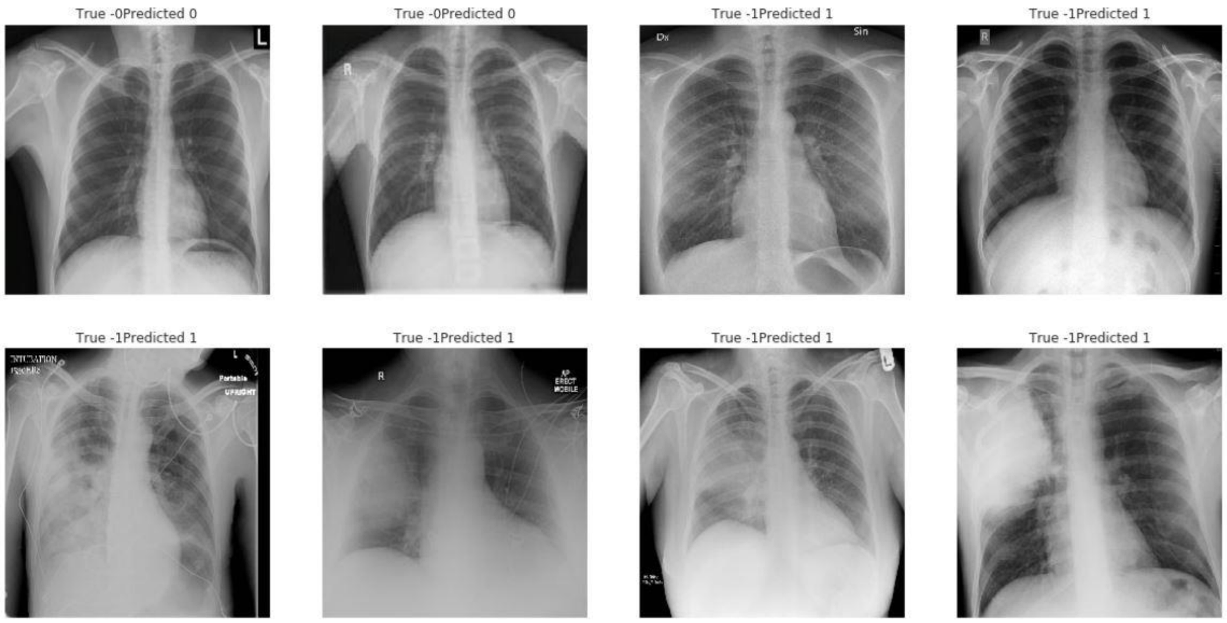




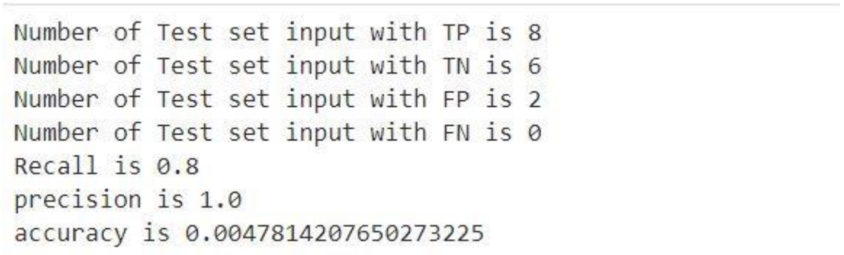
V. Conclusion

Free-Form Visualization

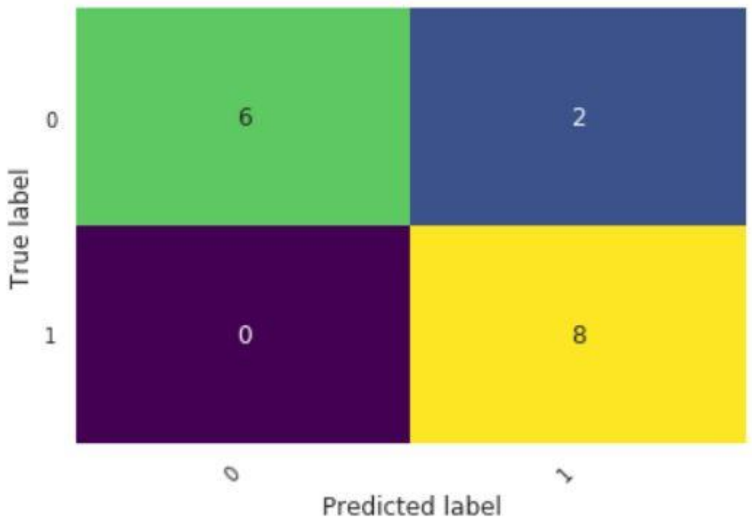
Getting some random images from the and labeling them; the final model predicts them mostly correctly, with 80% recall and 100% precision. However, these numbers are not reflective as only 8 images were obtained. Also, there is a chance that the images are corrupted or noisy.



Collected data tested on the CNN



```
Number of Test set input with TP is 8
Number of Test set input with TN is 6
Number of Test set input with FP is 2
Number of Test set input with FN is 0
Recall is 0.8
precision is 1.0
accuracy is 0.0047814207650273225
```



Confusion Matrix for the web images

Reflection

The process used for this project can be summarized using the following steps:

1. The problem was found and reported in the proposal.
2. The dataset was download (using a single line of code) and preprocessed on two stages.
3. The dataset was thoroughly explored and analyzed before training.
4. The classifier was training on the data for 30 epochs using 3 different models.
5. The final model was further tuned and tested on a statistically sufficient test set.

The most interesting part for me was to work on google cloud instance. This is my first time doing so, and it was extremely hard at the beginning because of the lack of proper documentation and because it is a new style different from what I used to.

Also, I had lost all my work on three separate occasions. One of them is deleting the instance by mistake after nearly finishing all the code, and I had to rewrite it again from scratch. I think this was beneficial as I learnt many things throughout the journey and made me link my work with GitHub.

Finally, the most beneficial part for me was working on a machine learning cycle from the very start to the very beginning. This made me ask more questions and dive deeper in the field of AI.

Improvement

A very important improvement that can be made to this project is using transfer learning as initial weights for the model. Although the model finally gets decent results, it could have reached these results faster if transfer learning was used.

Also, more advanced data augmentation can be used to generate more data for the model. This is always a good thing as it increases the model capacity.