Report for Detect Pneumonia with Deep Learning

***Abstract*—The task is to train Machine Learning models with the given dataset, having as goal to detect Pneumonia. Our dataset contains x-ray images and their labels as train dataset and just x-ray images for test dataset. In project was used Convolutional Neural Network models and implemented methods to optimize the model's performance and finally the results.**

I. Data & problem description

The task is to train various Machine Learning models and make predictions detecting
Pneumonia. Starting with the given datasets, train dataset (train_images), test dataset(test_images) and labels_train csv which contains labels of the train dataset in the form of pairs filename, class_id. The train dataset contains 4627(JPEG) train images, out of which 1227 images belong to class 0: no disease, 2238 images belong to class 1 : bacterial pneumonia and 1207 images belong to class 2 : viral pneumonia. The test dataset contains 1168 images. These images are x-ray images taken from patients and our goal is to create a model in which we will achieve the highest accuracy as it is possible to predict as better as we can the labels for the test dataset.

The images come in various sizes and the quality is quite good, according to our datasets slightly the half of the images of the train dataset belong to class 1, bacterial pneumonia while the amounts of the images belonging to class 0 and class 2 are almost equal. Some examples of the images, with their labels in figure 1.

Class 0, No disease

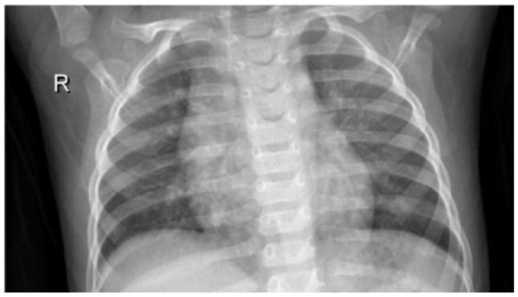


Figure 1.

Class 1, Bacterial pneumonia

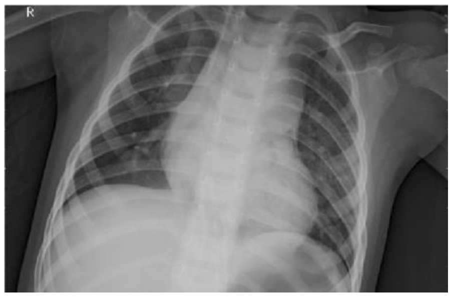


Figure 2.

Class 2, Viral pneumonia

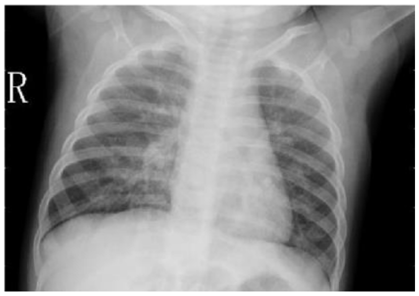


Figure 3.

II. Description of models used in each submission

A. Import the Libraries and run in Google Colab

Starting with importing the necessarily libraries like NumPy, Pandas and Keras in which we will use to create our model. Also, I import Google Drive because it is necessary to use Google Colab process power to run our model.

In all submission I run the models in Google Colab, using GPU or TPU accessed from Google. To do that, I had to mount the datasets on Google Drive to import them in the Google Colab. Afterwards, we create dataframes to store our data with their corresponding labels.

B. Pre-processing the data

In some models the pre-processing was different. I start by resizing the images in the same size and converting the color of them to gray. Submitting 29 predictions, in some of them the size of the images differs. In many of them the size was 100 x 100 in which we achieve higher speed execution for our model, but we lose some information. In the best submission I used 256 x 256 size in which the model performs better than resized pictures in 100 x 100 or 200 x 200. Also, in 2 submissions I also resize in 200 x 200, after cut the images trying to exclude the black frames from the X-ray and after resizing them again in 100 x 100. However, this method does not provide better results. Concluding this process, we save the images in NumPy arrays.

C. Split the data .

Continuing the process by splitting our dataset into X_train, X_val, Y_train and Y_val, setting the test size(validation) to 10%. Also, I reshape the numpy arrays to (-1, 256, 256, 1) to fit to the model, while I also normalize the train and validation data.

D. Implement some methods

After running many experiments with sparse categorical cross entropy, was wise to convert our data to categorical and use categorical cross entropy, so with the use of keras library, I converted the data to categorical because CNN model performs much better with categorical data in this project.

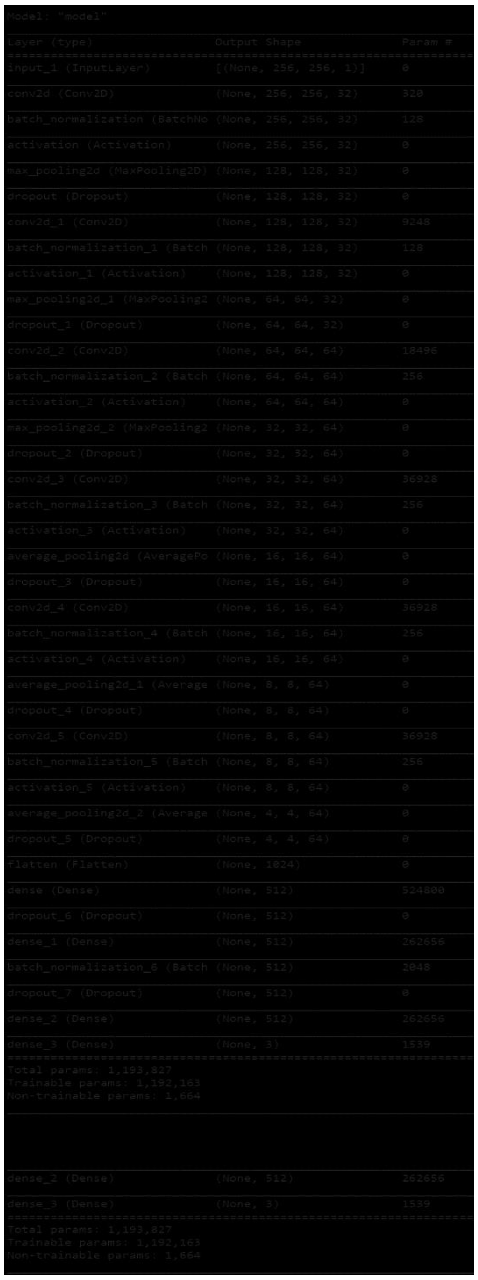
In addition, I used ImageDataGenerator which is a very famous method, that randomly modify the images, in many experiments were used many different transformation settings using ImageDataGenerator to our images, but the best settings so far were to randomly rotate 10 degrees, randomly zoom to 0.1, randomly shift images horizontally and vertically to 0.1 and randomly flip images horizontally.

Moreover, I used lr_schedule to prevent from overfitting while the lr change over the iterations of epoch.

E. The Model

The classification was done with the use of CNN (Convolutional Neural Network) model which is a class of deep learning neural networks. CNN method is very common in image recognition and classification because of his performance. The architecture of the model consists of some layers, like, input, hidden layers and output. The most important part and this that in most of the cases is modified for a better performance is the hidden layers. In all experiments was used Convolutional layers which are inside the hidden layers. Also, in all cases also, I used pooling, dropout and finally Dense layers.

In many submissions the model is changed by changing the setting of the CNN model. Trying out different combinations of Convolutional layers with different number of filters and kernel size. After 29 submissions the best performance model (figure 4) had 6 Convolutional layers with 32,32,64,64,64,64 set of filters. Finally, was used Flatten and 3 Dense layers with 512 neurons and 1 last Dense with 3 neurons as many as the classes.



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 256, 256, 1)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 256, 256, 32)      320
_________________________________________________________________
batch_normalization (BatchNo (None, 256, 256, 32)      128
_________________________________________________________________
activation (Activation)      (None, 256, 256, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 32)      0
_________________________________________________________________
dropout (Dropout)            (None, 128, 128, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 32)      9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 32)      128
_________________________________________________________________
activation_1 (Activation)    (None, 128, 128, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
activation_2 (Activation)    (None, 64, 64, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
activation_3 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
average_pooling2d (AveragePo (None, 16, 16, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
average_pooling2d_1 (Average (None, 8, 8, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 64)          0
_________________________________________________________________
average_pooling2d_2 (Average (None, 4, 4, 64)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 512)               524800
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 1539
=================================================================
Total params: 1,193,827
Trainable params: 1,192,163
Non-trainable params: 1,664
_________________________________________________________________
```

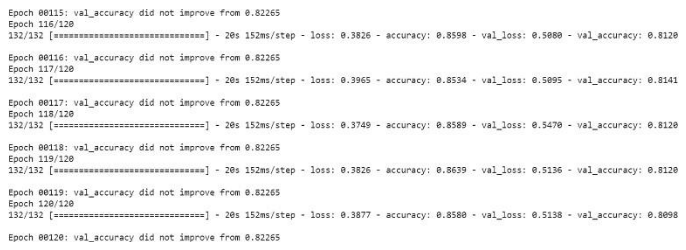
```
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 1539
=================================================================
Total params: 1,193,827
Trainable params: 1,192,163
Non-trainable params: 1,664
_________________________________________________________________
```

Figure 4.

The training and the testing of the model were performed in Google Colab with the following hardware, Python 3 Google Compute Engine backend (GPU) with 12GB RAM and 68.40 GB Disk. Worth mentioning that the GPUs available in Colab often include Nvidia K80s, T4s, P4s and P100s.

III. COMPARATIVE EXPERIMENTS AND RESULTS

In every different submission was tried different architecture of CNN model and different pre-processing on our data. Different number of convolutional layers with different number of filters with normal and categorical data. In all cases the goal is to prevent overfitting and success great performance on validation accuracy. The evaluation metric of this experiment is accuracy metric. Training our model, was set max epoch to 120 and batch size to 32. Also, I set the learning rate importing Learning Rate Scheduler from Keras library and using the lr schedule function for preventing overfitting. In the next figure 5 you can see the last results of our model training with the validation set accuracy to 82.265%.



```
Epoch 00115: val_accuracy did not improve from 0.82265
Epoch 116/120
132/132 [==============================] - 20s 152ms/step - loss: 0.3826 - accuracy: 0.8598 - val_loss: 0.5080 - val_accuracy: 0.8120

Epoch 00116: val_accuracy did not improve from 0.82265
Epoch 117/120
132/132 [==============================] - 20s 152ms/step - loss: 0.3965 - accuracy: 0.8534 - val_loss: 0.5095 - val_accuracy: 0.8141

Epoch 00117: val_accuracy did not improve from 0.82265
Epoch 118/120
132/132 [==============================] - 20s 152ms/step - loss: 0.3749 - accuracy: 0.8589 - val_loss: 0.5470 - val_accuracy: 0.8120

Epoch 00118: val_accuracy did not improve from 0.82265
Epoch 119/120
132/132 [==============================] - 20s 152ms/step - loss: 0.3826 - accuracy: 0.8639 - val_loss: 0.5136 - val_accuracy: 0.8120

Epoch 00119: val_accuracy did not improve from 0.82265
Epoch 120/120
132/132 [==============================] - 20s 152ms/step - loss: 0.3877 - accuracy: 0.8580 - val_loss: 0.5138 - val_accuracy: 0.8098

Epoch 00120: val_accuracy did not improve from 0.82265
```

Figure 5.

Also, I created a plot (figure 6) corresponding to train accuracy and validation accuracy for our best model which score 85.18% training accuracy and 82.265 % validation accuracy, while on Kaggle in-class competition it achieved 83.732 % accuracy score on Public
Leaderboard. According to the plot, we can see that after proximately the 45th epoch, the model becomes steadier and improving across the epoch, while in the 70th epoch it achieved the highest validation accuracy.

Finally, the prediction was done based on the test set after the model was trained. Lastly, every prediction was saved as csv for uploading it and checking the accuracy score that will finally achieve.

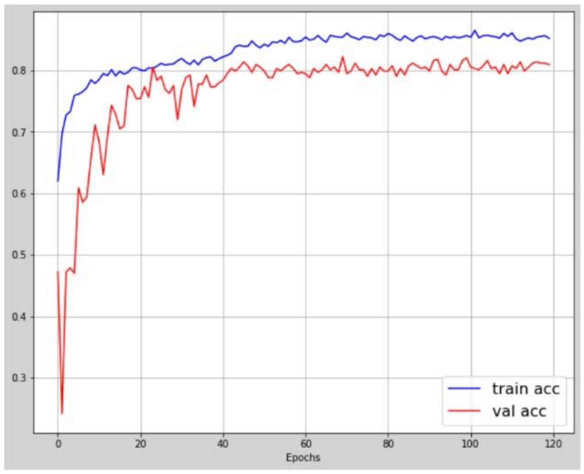


Figure 6.

IV. Conclusion

To conclude, while the goal of this project was to classify the X-ray images to three classes with the use of Machine Learning algorithm and predict right if someone has no disease, has bacterial pneumonia or viral pneumonia, thought would be wise to make use of the Convolutional Neural Network model and pre-process the image data before training the model and make predictions. After trying many different pre-processing methods and many different CNN architectures models the final model was the one with the best validation score. The results so far were good, however additional study and improvement required.