YASHVARDHAN DAS **U63030800**

CS 542 CLASS CHALLENGE REPORT

I. DESCRIPTION OF MODEL ARCHITECTURES

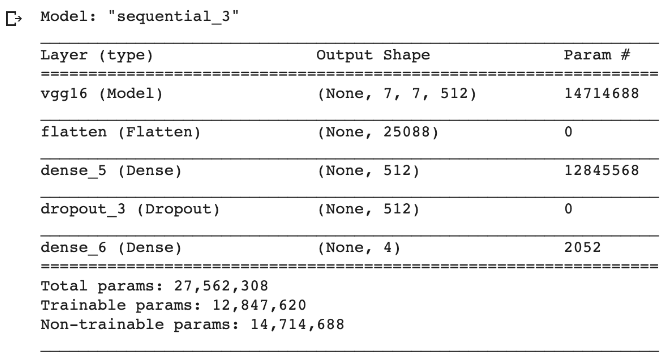
- For **Task 1 (Binary Classification)**, the details of the constructed Neural Network layers are explained as follows -
  - The very first layer added to the model includes that of the pre-trained network layer of **VGG16**. Its parameters include the weights' valueS to be initialised with those of **imagenet** and the respective input-shape of images is 224 * 224 * 3. The output dimension of this layer is 7 * 7 * 512
  - The second layer that is added sequentially is the Flatten layer.
  - The third layer added includes a Dense layer with 256 number of neurons and its activation function is the ReLU function.
  - The final layer added is the output Dense layer with the activation set to the **Sigmoid** function.
  - I initially added a **dropout** layer after the third layer but upon tuning and testing the model, I got to understand that the model gave better results without its usage.
- For **Task 2 (Multinomial Classification),** the details of the constructed Neural Network layers for the **First Model (Model 1)** are explained as follows -
  - The first layer added to the model includes that of the pre-trained network layer of **VGG16**. Its parameters include the weights value to be initialised with those of **imagenet** and the respective input-shape of images is 224 * 224 * 3. The output dimension of this layer is 7 * 7 * 512
  - The second layer that is added sequentially is the Flatten layer.
  - The third layer added includes a Dense layer with 512 number of neurons and its activation function is the **ReLU** function.
  - The fourth layer is a **dropout** layer with a respective percentage of **25%**.
  - The final output layer is a Dense layer with its activation function set to **Softmax** function.
- For **Task 2 (Multinomial Classification),** the details of the constructed Neural Network layers for the **Second Model (Model 2)** are explained as follows -

- The first layer added to the model includes that of the pre-trained network layer of **InceptionResNetV2**. Its parameters include the weights value to be initialised with those of **imagenet** and the respective input-shape of images is 224 * 224 * 3. The output dimension of this layer is 5 * 5 * 1536
- The second layer that is added sequentially is the Flatten layer.
- The third layer added includes a Dense layer with 512 number of neurons and its activation function is the ReLU function.
- The fourth layer is a **dropout** layer with a respective percentage of **30%**.
- The final output layer is a Dense layer with its activation function set to Softmax function.

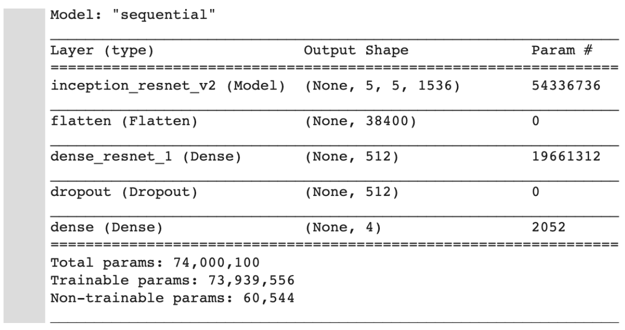
II. DESCRIPTION OF OPTIMISERS, LOSS FUNCTIONS, PARAMETERS AND REGULARIZATIONS

- For all the 3 models encompassing the 2 tasks, I have used the **adam** optimiser.
- The loss function used for **Task1** includes that of **binary_crossentropy** while for **Task2** it includes that of **categorical_crossentropy.**
- Regularizations are specifically used in **both the models** of **Task 2**. The parameter used in this regularization is the **bias_regularizer** with the **L2 norm loss**. No regularization is employed in **Task 1.**
- The rest of the parameters that have been used along these layers are - **include_top = False** and, **name** for simplicity of naming specific intermediate layers.

III. COMPARISON OF DIFFERENT ARCHITECTURES FOR THE SECOND TASK



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               12845568
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 4)                 2052
=================================================================
Total params: 27,562,308
Trainable params: 12,847,620
Non-trainable params: 14,714,688
_________________________________________________________________
```



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_resnet_v2 (Model)  (None, 5, 5, 1536)        54336736
_________________________________________________________________
flatten (Flatten)            (None, 38400)             0
_________________________________________________________________
dense_resnet_1 (Dense)       (None, 512)               19661312
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 4)                 2052
=================================================================
Total params: 74,000,100
Trainable params: 73,939,556
Non-trainable params: 60,544
_________________________________________________________________
```

Fig. 1 - Comparison of model architectures for Task 2

- Given above the screenshots of the summaries of the 2 different neural network architectures I have used for Task2.
- **VGGNet** was born out of the need to reduce the number of parameters in the CONV layers and improve on training time. **VGG16** has a total of 138 million parameters. The important point to note here is that all the conv kernels are of size 3x3 and maxpool kernels are of size 2x2 with stride of two.
- In an Image classification task, the size of salient feature can considerably vary within the image frame. Hence, deciding on a fixed kernel size is rather difficult. Lager kernels are preferred for more global features that are distributed over large area of the image, on the other hand smaller kernels provide good results in detecting area specific features that are distributed across the image frame. For effective recognition of such variable sized feature, we need kernels of different sizes. That is what **Inception Resnet V2** does. Instead of simply going deeper in terms of number of layers, it goes wider. Multiple kernels of different sizes are implemented within the same layer. Inception Resnet V2 increases the network space from which the best network is to be chosen via training. Each inception module is able to capture salient features at different levels.
- The first model ("sequential_3") is composed of the pre-trained model of **VGG16** as the initial layer. The second model ("sequential") contains the pre-trained model of **Inception Resnet V2** as the first layer.
- The similarities between both these architectures include the usage of 512 units (neurons) for the respective intermediate dense layers and the employing the same regularizer ,i.e, bias regularizer with L2 norm loss.
- One key difference between the architectures includes the usage of **different rates of dropouts** - 0.25 for Model 1 and 0.30 for Model 2. These rates have been used eventually after tuning with other rates and getting the best approximated results.
- The testing accuracy of Model 1 is about **78%** while that of Model 2 is around **64%**.
- The loss rate of Model 1 is about **0.73** while that of Model 2 is around **0.90**.
- For model 1, the parameter tuning for better accuracy involved **reducing the dropout rate** from 0.30 to 0.25, **changing the regularizer** from kernel type to bias type and **increasing the number of nodes** in the dense layer from 256 to 512.

- Similarly, the parameter tuning for achieving for more accurate results for Model 2 involved **fluctuating the dropout rates** from 0.25 to 0.35 and **finally choosing ReLU function** as the activation function in the intermediate dense layer.

IV. PLOT AND COMMENT ON THE ACCURACY AND THE LOSS FOR BOTH TASKS

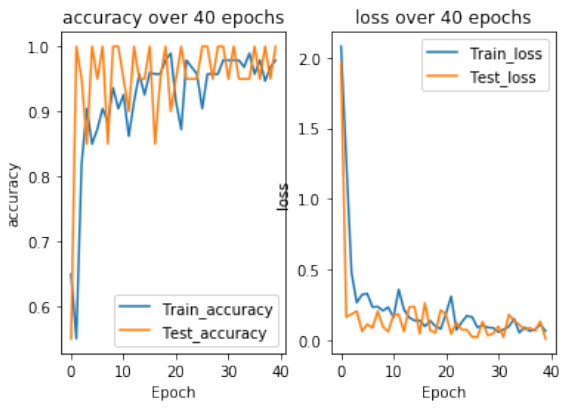


Fig. 2 - Accuracy & Loss Plots for Task 1

- As shown in the figure above, we can observe that the accuracies for both training and testing data increase continuously upto a certain point and then start fluctuating within a specific range. At the last epoch, these two values have an acceptable level of difference owing to proper usage of regularizations and dropouts.
- For the loss graph, the training and testing losses decrease continuously upto a certain level and after that, we observe minor fluctuations between two specific levels. As was in the case of accuracy plots, at the last epoch these two values have an acceptable level of difference.

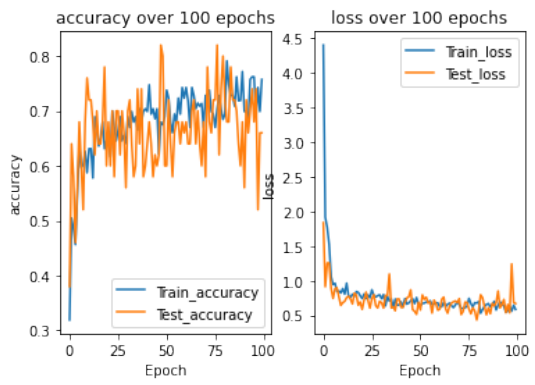


Fig. 3 - Accuracy & Loss Plots for Task 2 (Model 1)

- As shown in the above plot, there is a proportionate fluctuation in both the training and testing accuracies. The overall testing accuracy after completion of all 100 epochs is 78%.
- Similar observation can be inferred from the loss graph. The testing loss after the complete classification stood at 0.72. As was in the case of accuracy plots, at the last epoch these two values have an acceptable level of difference.

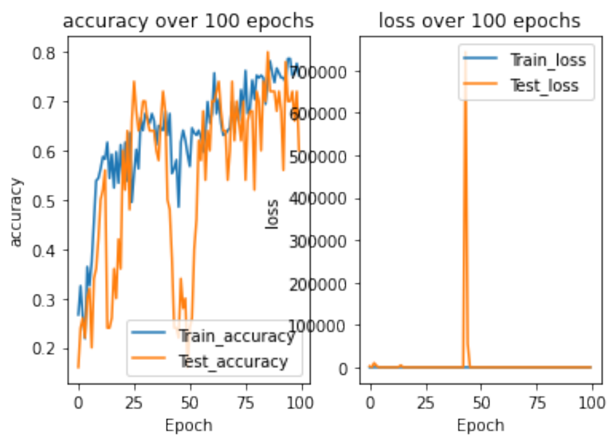


Fig. 4 - Accuracy & Loss Plots for Task 2 (Model 2)

- For this second model of **Task 2**, there is a case of overfitting in the model at some point as observed from the huge down-spike of test accuracy. This corresponds to a similar rise in test loss as shown in the adjacent graph. At the later stages of the epochs, the difference in the parameters in the test and train data becomes proportionate owing to proper usage of regularizations and dropouts.
- The overall testing accuracy after completion of all 100 epochs is around 64%, while the overall loss rate stands at 0.90.

V. PLOT AND COMMENT ON THE TSNE VISUALIZATIONS

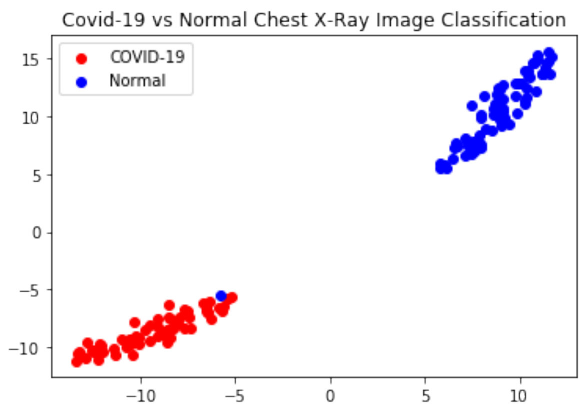


Fig. 5 - TSNE Plot of Task 1

- As observed from the above plot, there is a stark difference and segregation between the 2 classes of images. There is, however, a single output of misclassification observed.
- The parameters used in the TSNE function used include n_components=2 and random_state=4.

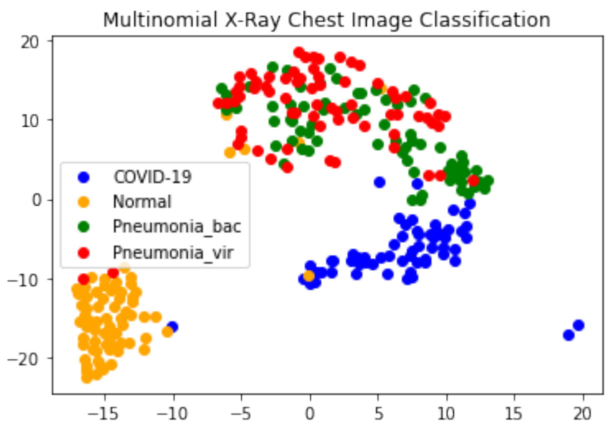


Fig. 6 - TSNE Plot of Task 2 (Model - 1)

- The above constructed TSNE plot shows a clear demarcation between Normal cases and the rest of the 3 types of cases. Between these 3 cases, there is also a considerable level of separation between the Covid-19 class the remaining classes, with the exceptions of some misclassified outcomes.
- However, there is a significant overlap between the class types of Pneumonia_bac and Pneumonia_vir. This scenario may have arisen owing to the fact that the these 2 classes of diseases have lots of similarities in the types of symptoms and the respective carriers. Both these disease types fall under the category of Pneumonia with only the difference arising between further sub-categories.

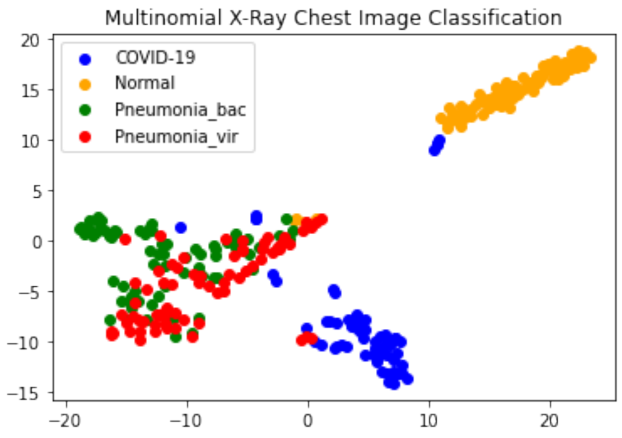
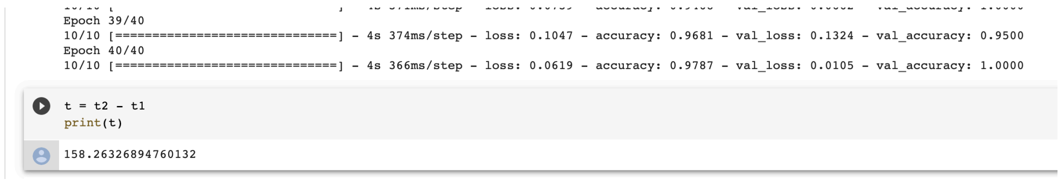


Fig. 7 - TSNE Plot of Task 2 (Model - 2)

- The above constructed TSNE plot shows a clear demarcation between Normal cases, Covid-19 cases and the rest of the cases, with the exceptions of some misclassified outcomes.
- However, again, there is a significant overlap between the class types of Pneumonia_bac and Pneumonia_vir. This scenario may have arisen owing to the fact that the these 2 classes of diseases have lots of similarities in the type of symptoms and the respective carriers. Both these disease types fall under the category of Pneumonia with only the difference arising between further sub-categories.

VI. GPU vs CPU TRAINING TIME COMPARISON

- I ran the CPU task in Google Colab while the GPU task was executed on the BU's Shared Computing Cluster (SCC). My .ipynb notebook that exists in the folder of named "yashvdas" is the one that used the GPU of the SCC. I used 2 CPU cores and 1 GPU core.
- For the CPU implementation, I had already tried implementing the code with Google Colab before and hence I took the screenshot by then.
- The implementation of all the epochs in GPU took about 158 seconds, i.e, just around 2.5 minutes.
- The implementation of all the epochs in CPU took about 2451 seconds, i.e, just around 41 minutes.

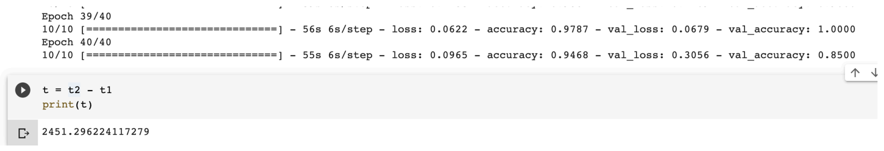


```
Epoch 39/40
10/10 [==============================] - 4s 374ms/step - loss: 0.1047 - accuracy: 0.9681 - val_loss: 0.1324 - val_accuracy: 0.9500
Epoch 40/40
10/10 [==============================] - 4s 366ms/step - loss: 0.0619 - accuracy: 0.9787 - val_loss: 0.0105 - val_accuracy: 1.0000
```

```
t = t2 - t1
print(t)
```

```
158.26326894760132
```

GPU's TIME CONSUMPTION



```
Epoch 39/40
10/10 [==============================] - 56s 6s/step - loss: 0.0622 - accuracy: 0.9787 - val_loss: 0.0679 - val_accuracy: 1.0000
Epoch 40/40
10/10 [==============================] - 55s 6s/step - loss: 0.0965 - accuracy: 0.9468 - val_loss: 0.3056 - val_accuracy: 0.8500
```

```
t = t2 - t1
print(t)
```

```
2451.296224117279
```

CPU's TIME CONSUMPTION