EE 452 - COMPUTER VISION

Assignment I Report

Instructor - Dr. Muhammad Farhan

Author - Kuldeep Dileep (kl04008)

Contents

1 Question 1

1.1 Part a

Figure 1.1 shows few samples from both classes of Malaria dataset i.e. uninfected and parasitized. From the figure we can also identify their class labels i.e. if labelis 0 then the sample belongs to parasitized class otherwise, if label is 1 then the sample belongs to uninfected class. From the figure one can tell that the major difference is that an uninfected cell/organism has a uniform color throughout whereas, parasitized cell/organism has purple colored spots in them.

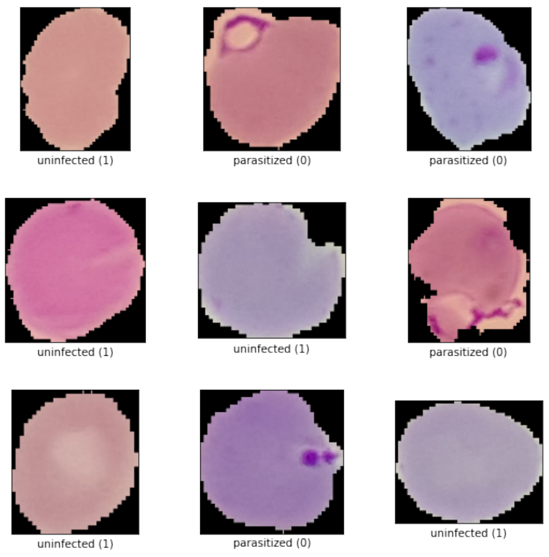


Figure 1.1: Samples from Malaria dataset

1.2 Part b

Binary classification is performed on dataset where there are only 2 classes that the samples could be classified in. Typically, there is only one class that is normal class and the other class becomes abnormal. In a binary classification, classes can be represented in binary form where

normal class could be labeled 1 and abnormal as 0.
We are using binary classification for this task because there are only 2 classes namely, uninfected and parasitized given labels 1 and 0 respectively.

1.3 Part c

Following are the pre-processing techniques that I have applied on the malaria dataset and the rationale behind them.

1. Resize: Once when the Malaria dataset was loaded, I observed that the samples in it had varying shape size which would have been incompatible with the constant input shape of any model. Moreover, due to the large size of the dataset my session on colab kept crashing because of insufficient RAM. So to resolve these issues I reshaped all images to (96, 96, 3) because that was the size that the RAM could handle.

2. Normalization: It is an image pre-processing technique where we change the range of image's pixel intensities from [0, 255] to [0, 1]. It helps in contrast stretching which enhances the contrast of image which enables classifiers to extract better features for better classification.

3. Data type conversion: I converted data type of samples from uint8 to float32 because float values give more precise information than an integer value. Moreover, after normalization float value tend to have more range between 0 and 1 and therefore, they give a better representation of integer values ranged between 0 and 255.

1.4 Part e

Once that the dataset was prepared then with the help of sklearn I split the dataset into 70% training and 30% test dataset. Then I used sklearn's built-in function of Logistic Regression and achieved test accuracy of 65.69%. To improve this I performed various techniques but with kfold cross validation at k = 10 classifier's test accuracy improved to 69%. Which is close enough to the required threshold of 70%. I decided the value of k after going through various literature available.

2 Question 2

Cifar-10 dataset contains 60,000 colored images of 32x32 dimension each belonging to 10 different classes which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Figure 2.1 shows 10 sample images from each class. When the dataset is loaded into python notebook it results in training and test set containing 50,000 and 10,000 RGB images respectively.

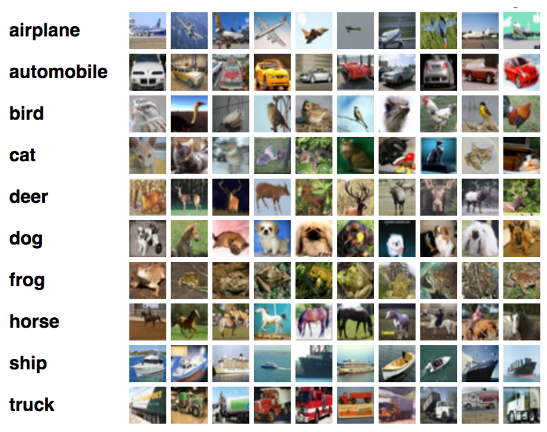


Figure 2.1: Samples from Cifar-10 dataset

Preprocessing is a set of techniques applied on images to enhance their appearance and spatial information so that better features could be extracted which will result in better performance of classifier. In general following are some of the pre-processing techniques applied on images:

1. Converting RGB image to grayscale: When color information is not required so to save computational power and memory image is converted to grayscale.

2. Resizing image: Some classifier such as CNN has a constant input size so there is a need to scale images to the required dimensions. Moreover, due to memory limitation images could be resized to dimension where the features aren't lost.

3. Data augmentation: It involves scaling, rotation and other transformation technique in order to increase the size of dataset and to introduce variation so that a more general classifier is achieved.

4. Histogram equalization/normalization: This is a contrast enhancement technique which helps in enhancing the appearance and features of the image which improves the performance of the classifier.

5. Image filtering: Image filtering is performed to suppress irrelevant information and enhance relevant and important features such as low pass filter suppresses noise and high pass filter extracts edges.

3 Question 3

In this task we were expected to develop an algorithm from scratch that would apply filter on image. Filters are applied with or without padding. When padding is performed the filtered image has the same dimension as the original image on the contrary, when padding is not performed the size of the image reduces. To perform zero-padding I computed padding factor from the dimension of filter and then increased the number of rows and columns of the input image by 2 times the padding factor whereas, the dimension of output image is same as the original input image. When padding is not applied the dimension of output image is reduced by 2 times the padding factor.
Next, I iterated through image and if the pixel in the image is a valid pixel i.e. when the center of the filter is placed on that pixel then the filter completely overlaps and doesn't exceed the boundaries of the image. Then I sliced that region of the image and computed element wise multiplication and summed them all. Lastly, I assigned the resulting number at the respective

position of the output image. This process continues until the filter slides over all the valid pixels and results in a filtered output image.

Figure 3.1 shows the resulting filtered image on the right when 9x9 averaging filter was applied on the original image on the left without padding. When the same filter was applied on a gray scale image it resulted in the filtered image on the right in figure 3.2. On the other hand, in figure 3.3 and 3.4 padding was also performed along with filteration. I have also tested sobel filter for edge detection and figure 3.5 and 3.6 show the resultant edge map image of RGB and gray scale image respectively.

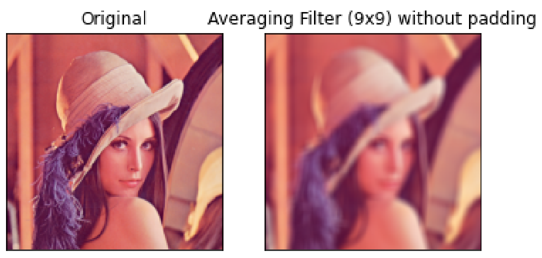


Figure 3.1:

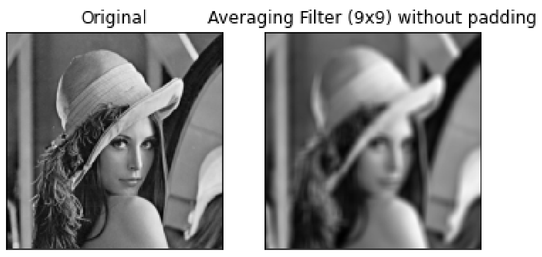


Figure 3.2:

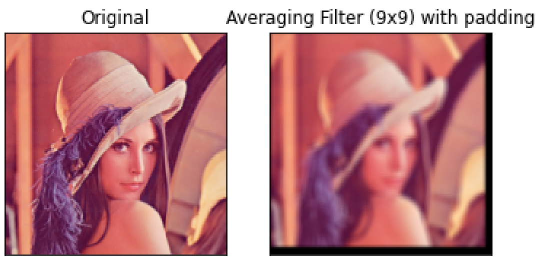


Figure 3.3:

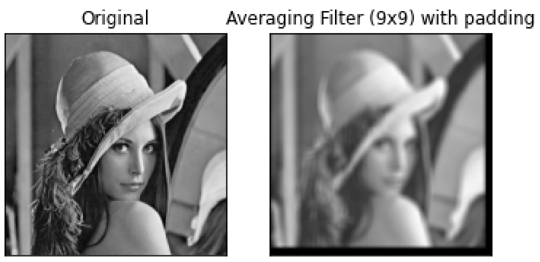


Figure 3.4:

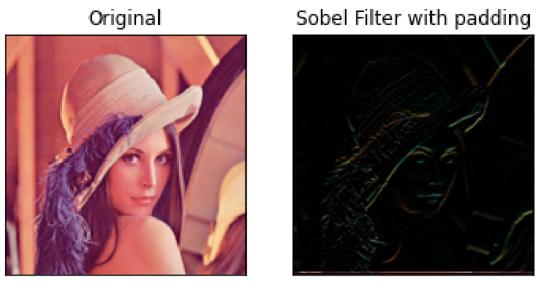


Figure 3.5:

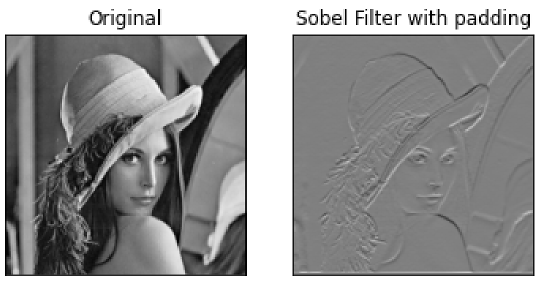


Figure 3.6:

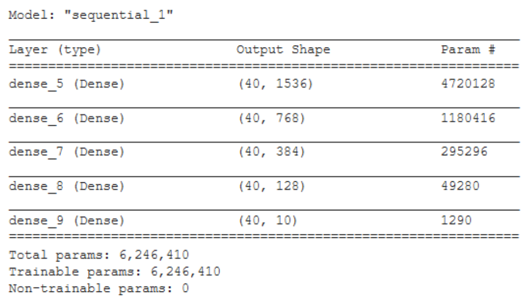
4 Question 4

Histogram of Oriented Gradient (HOG) is a feature descriptor in which histogram of oriented gradient are used as features. These gradients are useful in extracting information about the object in the image such as edges. It gives the information of the magnitude of edges and their orientation. The remaining not so useful information is suppressed.
To extract hog features I used skimage's builtin function hog which returns feature descriptor and hog image. With this function I computed feature descriptor for all the images in training and test dataset and fed that feature descriptor in a linear SVM and achieved an accuracy of 55%.

5 Question 5

5.1 Part a

I initialized a sequential model as I had to stack layers in series. I had tried different architectures by varying number of layers, adding dropout layers at varying dropout rate and varying number of neurons but figure 5.1 shows the summary of my final feedforward Neural Network's architecture that gave best results. I have used 4 hidden layers with input nodes of 1536, 768, 384, and 128 respectively for each layer. I have used ReLu as the activation function. I had tried various optimization function such as Stochastic Gradient Descent (SGD) at varying learning rate and momentum, RMSProp and adam. But with SGD at 0.01 learning rate, 50 epochs and batch size of 40 I achieved the maximum test accuracy of 57.5% as shown in figure 5.4. Figure 5.2 and 5.3 show the learning curves of the model.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (40, 1536)                4720128
_________________________________________________________________
dense_6 (Dense)              (40, 768)                 1180416
_________________________________________________________________
dense_7 (Dense)              (40, 384)                 295296
_________________________________________________________________
dense_8 (Dense)              (40, 128)                 49280
_________________________________________________________________
dense_9 (Dense)              (40, 10)                  1290
=================================================================
Total params: 6,246,410
Trainable params: 6,246,410
Non-trainable params: 0
```

Figure 5.1: Summary of feedforward Neural Network architecture

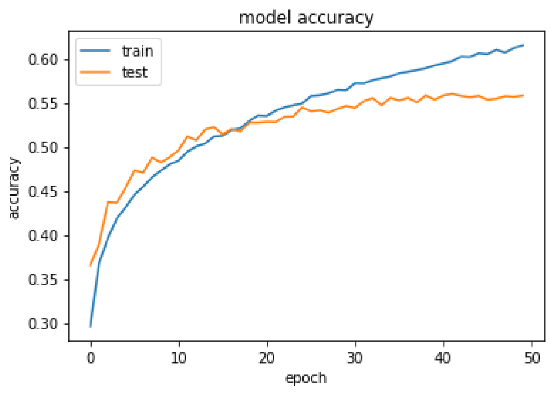


Figure 5.2: Epochs vs Accuracy plot

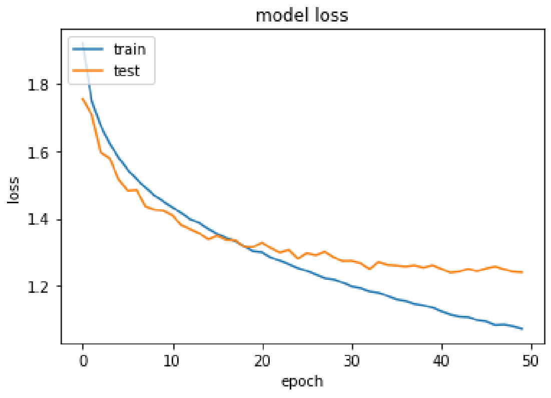
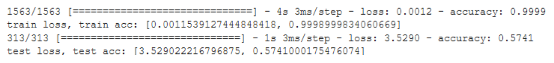


Figure 5.3: Epochs vs Loss plot

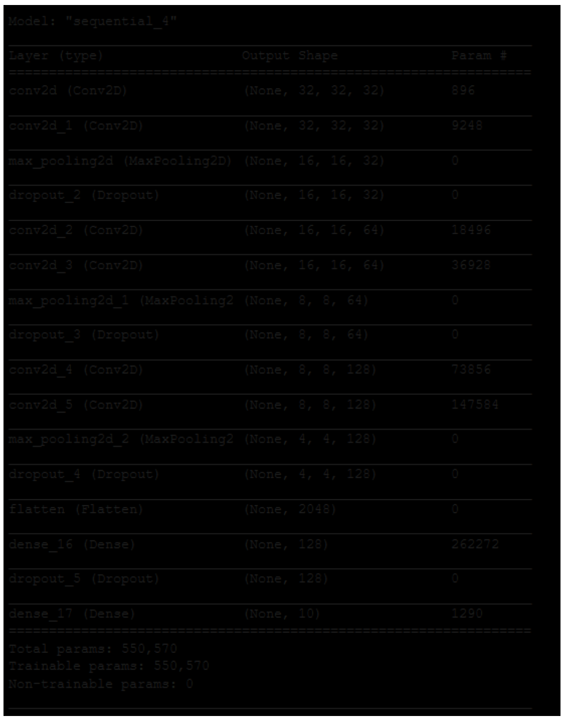


```
1563/1563 [==============================] - 4s 3ms/step - loss: 0.0012 - accuracy: 0.9999
train loss, train acc: [0.0011539127444848418, 0.9998999834060669]
313/313 [==============================] - 1s 3ms/step - loss: 3.5290 - accuracy: 0.5741
test loss, test acc: [3.529022216796875, 0.5741000175476074]
```

Figure 5.4: Result of feedforward Neural Network

5.2 Part b

I initialized a sequential model as I had to stack layers in series. I added an input layer with input size according to our data set i.e. (32, 32, 3) and then I added 5 Conv2D layers with 32, 64, 64, 128, and 128 number of neurons along with Maxpooling and dropout layer. The summary of the model is shown in figure 5.5. I have used ReLu as the activation function. I had tried various optimization function such as Stochastic Gradient Descent (SGD) at varying learning rate and momentum, RMSProp and adam. But with SGD where learning rate is 0.001, momentum is 0.9 and 50 epochs I was able to achieve the maximum test accuracy of 83% as shown in figure 5.8. Figure 5.6 and 5.7 show the learning curves of the model.



```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense_16 (Dense)             (None, 128)               262272
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1290
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.5: Summary of CNN architecture

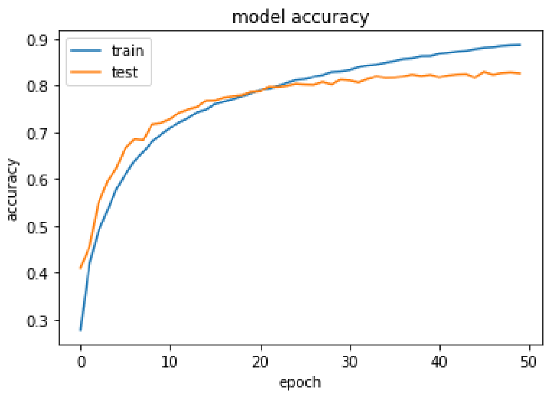


Figure 5.6: Epochs vs Accuracy plot

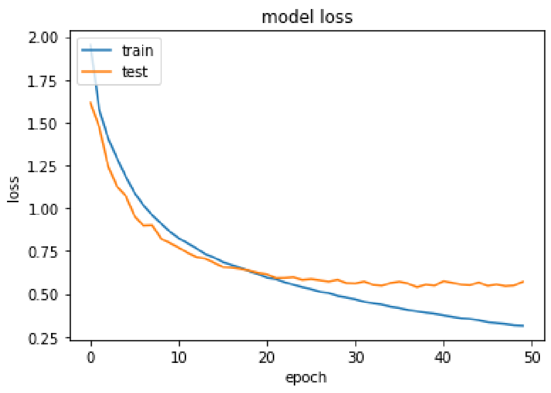
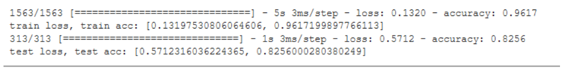


Figure 5.7: Epochs vs Loss plot

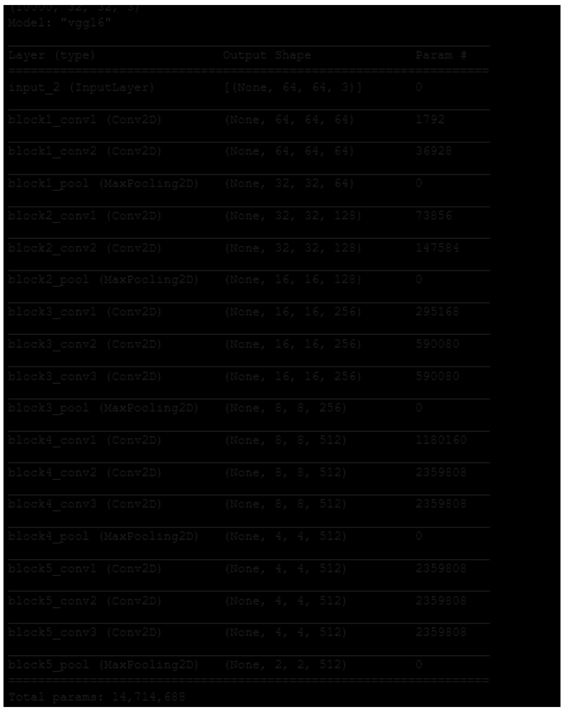


```
1563/1563 [==============================] - 5s 3ms/step - loss: 0.1320 - accuracy: 0.9617
train loss, train acc: [0.13197530806064606, 0.9617199897766113]
313/313 [==============================] - 1s 3ms/step - loss: 0.5712 - accuracy: 0.8256
test loss, test acc: [0.5712316036224365, 0.8256000280380249]
```

Figure 5.8: Result of CNN

5.3 Part c

VGG-16 is a CNN model with 16 layers where 13 layers are convolutional layers and 3 are fully connected layers. In this part of the assignment I have stacked a pre-trained VGG-16 model on 2 fully connected layers with 512 and 256 number of neurons. I have used ReLu as the activation function. I had tried various optimization function such as Stochastic Gradient Descent (SGD) at varying learning rate and momentum, RMSProp and adam. But with adam where learning rate is 0.001 and 5 epochs I was able to achieve the maximum test accuracy of 71% as shown in figure 5.12. Figure 5.10 and 5.11 show the learning curves of the model.



```
(10000, 32, 32, 3)
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
=================================================================
Total params: 14,714,688
```

Figure 5.9: Summary of architecture

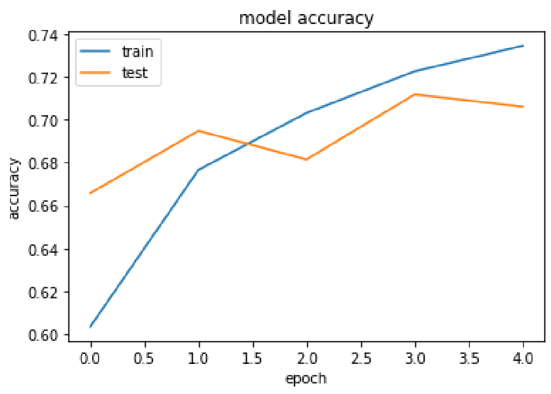


Figure 5.10: Epochs vs Accuracy plot

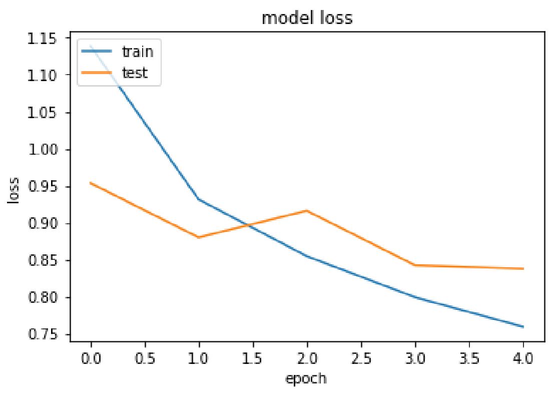
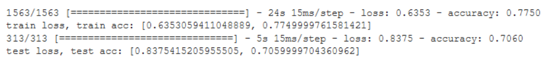


Figure 5.11: Epochs vs Loss plot



```
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6353 - accuracy: 0.7750
train loss, train acc: [0.6353059411048889, 0.7749999761581421]
313/313 [==============================] - 5s 15ms/step - loss: 0.8375 - accuracy: 0.7060
test loss, test acc: [0.837541520595505, 0.7059999704360962]
```

Figure 5.12: Result

6 Conclusion

I had a great time working on this assignment. It gave me a good exposure to various fundamental concepts of deep learning. It gave me the chance to gain hands-on experience over various fundamental libraries involved when working with deep learning models. I found model tuning to improve accuracy the most difficult and inefficient process since, there are numerous parameters involved and I wasn't clear on how different parameters affect the performance of the model. Overall it was a good learning experience.