CS512 Assignment 4: Report

Kajol Tanesh Shah
Department of Computer Science
Illinois Institute of Technology
April 6, 2022

Abstract

In this assignment, we have performed binary classification and multi-class classification on MNIST and CIFAR10 datasets respectively using Convolutional Neural Networks and implemented residual and inception blocks along with it. We have also done hyperparameter tuning as per the model requirement to improve accuracy. To do these operations, we have used the Keras, OpenCV libraries in python and other libraries like Numpy, Matplotlib, etc. were also used.

1. Problem Statement

In this assignment, our aim was to perform binary classification on MNIST dataset and multi-class classification of CIFAR10 dataset. In the MNIST dataset, the handwritten digits had labels from 0-9 which were to be converted into odd/even labels and then a network was to be constructed with multiple convolution, pooling, dropout and fully connected layers to perform binary classification. The CIFAR10 dataset consists of 10 image categories and for this, a similar basic convolution neural network was to be constructed and the model's performance was to be tested. Inception blocks and Residual blocks were also to be added and then tested for the performance of the model.

2. Proposed solution

Using the Keras, OpenCV libraries and other supporting libraries like numpy, matplotlib, etc. we have implemented the program to do classification of the images.

Part 1: Binary Classification

1)Firstly, MNIST dataset was loaded from keras.datasets and split into train/test/validation subsets. After that, the image array was reshaped and normalized by dividing it by 255.
2)The dataset consisted of labels 0-9 but as our aim was to perform binary classification, we converted the digit labels into odd/even respectively
3)After that, we constructed a CNN model with pooling, dropout and fully connected layers using keras and added a flatten layer too
4)For the model, we used SGD optimizer with 0.001 learning rate and used binary_crossentropy loss function and used 'accuracy' as the evaluation metric
5)We then trained the model on the training data and validated it against the validation data and observed the accuracy and loss

6)After that, we plotted the training and validation loss as a function of epochs using matplotlib.pyplot

Part 2: Hyperparameter tuning

In this step, we implemented different variations of the model created in Part 1. Here, we changed the network architecture, receptive field, stride, optimizer, loss function, learning rate, number of epochs, added normalization layers and different weight initializers

Part 3: Inference

Here, we used our pre-trained CNN to classify a handwritten digit into odd/even. Here, using OpenCV, we did some preprocessing steps on the image like resizing it, transforming it into grayscale, then to a binary image using GaussianBlur() and adaptiveThreshold() functions. We then reshaped it to provide it as input to our CNN model

Part 4: Multiclass Classification

1)Here, we loaded the CIFAR10 dataset from keras.datasets and built a basic CNN model with various convolution, pooling, normalization and dense layers and used softmax activation function
2)After that, we tested the model and tuned hyperparameters to improve accuracy
3)We also added inception model and residual blocks and tested the performance of the models

3. Implementation details

Some of the problems and design issues that were faced are as mentioned below:

- At first, I was not getting good accuracy with the MNIST model in Part 1. I tried changing various parameters but it did not work. After changing the activation function of the last layer from softmax to sigmoid, the accuracy of my model improved
- When I added multiple convolution layers, I was getting error as the dimensions were transformed to negative due to pooling. So, I reduced a couple of layers and this fixed my issue
- I tried displaying the images in separate windows using cv2.imshow function but this function does not work in google colab. The alternative for this function is cv2_imshow but even using this, I was not able to display the images in separate windows. As the image displayed by cv2_imshow was very small, I used matplotlib.pyplot.imshow() for it
- I faced issues while coding the logic for residual and inception blocks as I was not able to integrate these blocks with my basic model. So, I created separate models for both Residual and Inception blocks

4. Results

Part 1:

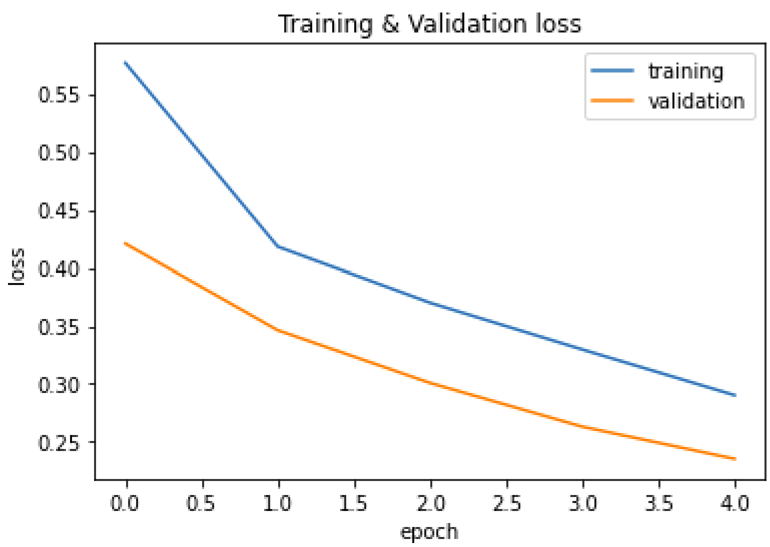
Basic Model with 2 convolution layers, pooling, dropout and 2 fully connected layers

Training and validation loss

```
[42] history = mmodel.fit(train_images, train_labels, epochs = 5, batch_size=128, validation_data=(val_images, val_labels))

Epoch 1/5
430/430 [==============================] - 46s 104ms/step - loss: 0.5764 - accuracy: 0.7140 - val_loss: 0.4207 - val_accuracy: 0.8073
Epoch 2/5
430/430 [==============================] - 53s 124ms/step - loss: 0.4182 - accuracy: 0.8091 - val_loss: 0.3464 - val_accuracy: 0.8545
Epoch 3/5
430/430 [==============================] - 44s 103ms/step - loss: 0.3700 - accuracy: 0.8371 - val_loss: 0.3009 - val_accuracy: 0.8801
Epoch 4/5
430/430 [==============================] - 46s 106ms/step - loss: 0.3297 - accuracy: 0.8576 - val_loss: 0.2632 - val_accuracy: 0.9023
Epoch 5/5
430/430 [==============================] - 52s 122ms/step - loss: 0.2903 - accuracy: 0.8801 - val_loss: 0.2354 - val_accuracy: 0.9143
```

```
[44] histm = mmodel.evaluate(test_images, test_labels)

157/157 [==============================] - 2s 10ms/step - loss: 0.2126 - accuracy: 0.9244
```

```
Text(0, 0.5, 'loss')
```



As we can see, the model achieved a good accuracy of 92% and the training and validation loss decreased over time as the number of epochs increased

Part 2: Hyperparameter tuning

Model 2 - In this model, I added more convolutional layers, added dropout=0.2, used ‘he_normal’ weight initializer and Batch Normalization layers, Flatten and Dense layers. After giving more convolutional layers, increasing the number of epochs to 10, using different optimizer=’adam’ and changing the learning rate to 0.01, the model’s accuracy increased by 8%. This model performed the best out of all the other model variations with an accuracy of 99.26% on test set

```
[48] history2 = m2model.fit(train_images, train_labels, epochs = 10, batch_size=128, validation_data=(val_images, val_labels))
```

```
Epoch 1/10
430/430 [==============================] - 79s 181ms/step - loss: 0.1696 - accuracy: 0.9339 - val_loss: 0.0698 - val_accuracy: 0.9751
Epoch 2/10
430/430 [==============================] - 73s 169ms/step - loss: 0.0746 - accuracy: 0.9734 - val_loss: 0.0352 - val_accuracy: 0.9876
Epoch 3/10
430/430 [==============================] - 70s 164ms/step - loss: 0.0562 - accuracy: 0.9797 - val_loss: 0.0345 - val_accuracy: 0.9880
Epoch 4/10
430/430 [==============================] - 70s 164ms/step - loss: 0.0465 - accuracy: 0.9838 - val_loss: 0.0263 - val_accuracy: 0.9911
Epoch 5/10
430/430 [==============================] - 71s 165ms/step - loss: 0.0410 - accuracy: 0.9858 - val_loss: 0.0270 - val_accuracy: 0.9911
Epoch 6/10
430/430 [==============================] - 71s 164ms/step - loss: 0.0351 - accuracy: 0.9883 - val_loss: 0.0260 - val_accuracy: 0.9922
Epoch 7/10
430/430 [==============================] - 70s 164ms/step - loss: 0.0319 - accuracy: 0.9889 - val_loss: 0.0265 - val_accuracy: 0.9917
Epoch 8/10
430/430 [==============================] - 70s 163ms/step - loss: 0.0301 - accuracy: 0.9893 - val_loss: 0.0239 - val_accuracy: 0.9923
Epoch 9/10
430/430 [==============================] - 70s 162ms/step - loss: 0.0268 - accuracy: 0.9905 - val_loss: 0.0226 - val_accuracy: 0.9927
Epoch 10/10
430/430 [==============================] - 70s 162ms/step - loss: 0.0236 - accuracy: 0.9917 - val_loss: 0.0241 - val_accuracy: 0.9923
```

```
[49] histm2 = m2model.evaluate(test_images, test_labels)
```

```
157/157 [==============================] - 2s 13ms/step - loss: 0.0248 - accuracy: 0.9926
```

Model 3

In this case, we changed the stride of the filter to 2 and used a different kernel size which is 5x5 and gave epochs as 8. This model also performed well with an accuracy of 99%

```
history3 = m3model.fit(train_images, train_labels, epochs = 8, batch_size=128, validation_data=(val_images, val_labels))
```

```
Epoch 1/8
430/430 [==============================] - 20s 45ms/step - loss: 0.1807 - accuracy: 0.9261 - val_loss: 0.0580 - val_accuracy: 0.9788
Epoch 2/8
430/430 [==============================] - 18s 41ms/step - loss: 0.0726 - accuracy: 0.9745 - val_loss: 0.0378 - val_accuracy: 0.9860
Epoch 3/8
430/430 [==============================] - 25s 59ms/step - loss: 0.0518 - accuracy: 0.9817 - val_loss: 0.0361 - val_accuracy: 0.9873
Epoch 4/8
430/430 [==============================] - 18s 41ms/step - loss: 0.0426 - accuracy: 0.9852 - val_loss: 0.0322 - val_accuracy: 0.9884
Epoch 5/8
430/430 [==============================] - 18s 41ms/step - loss: 0.0349 - accuracy: 0.9878 - val_loss: 0.0302 - val_accuracy: 0.9903
Epoch 6/8
430/430 [==============================] - 17s 40ms/step - loss: 0.0279 - accuracy: 0.9903 - val_loss: 0.0267 - val_accuracy: 0.9911
Epoch 7/8
430/430 [==============================] - 24s 57ms/step - loss: 0.0256 - accuracy: 0.9910 - val_loss: 0.0265 - val_accuracy: 0.9911
Epoch 8/8
430/430 [==============================] - 18s 42ms/step - loss: 0.0224 - accuracy: 0.9918 - val_loss: 0.0257 - val_accuracy: 0.9909
```

```
histm3 = m3model.evaluate(test_images, test_labels)
```

```
157/157 [==============================] - 1s 5ms/step - loss: 0.0346 - accuracy: 0.9908
```

Model 4

In this case, a different loss function ‘hinge’ was used and ‘random_normal’ weight initializer was used. The accuracy of this model was a little less as compared to other models but was still good which is 95%

```
history4 = m4model.fit(train_images, train_labels, epochs = 5, batch_size=128, validation_data=(val_images, val_labels))
```

```
Epoch 1/5
430/430 [==============================] - 19s 42ms/step - loss: 0.6200 - accuracy: 0.8728 - val_loss: 0.5433 - val_accuracy: 0.9493
Epoch 2/5
430/430 [==============================] - 17s 40ms/step - loss: 0.5727 - accuracy: 0.9190 - val_loss: 0.5352 - val_accuracy: 0.9578
Epoch 3/5
430/430 [==============================] - 17s 41ms/step - loss: 0.5743 - accuracy: 0.9173 - val_loss: 0.5442 - val_accuracy: 0.9482
Epoch 4/5
430/430 [==============================] - 20s 47ms/step - loss: 0.5662 - accuracy: 0.9253 - val_loss: 0.5328 - val_accuracy: 0.9600
Epoch 5/5
430/430 [==============================] - 18s 42ms/step - loss: 0.5682 - accuracy: 0.9234 - val_loss: 0.5382 - val_accuracy: 0.9544
```

Model 5

In this model, the receptive field was changed by using dilation_rate = 2. Also, Layer normalization was used. This model also performed well with an accuracy of 98%

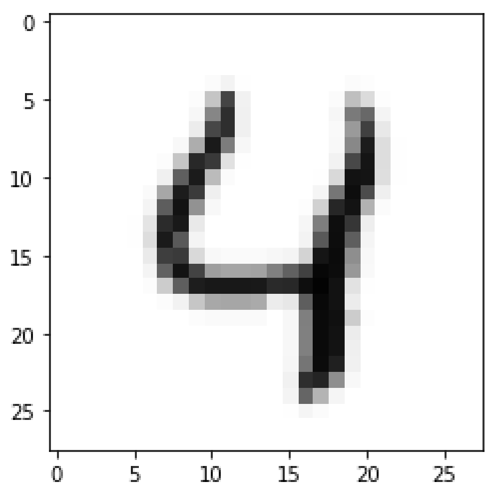
```
history5 = m5model.fit(train_images, train_labels, epochs = 5, batch_size=128, validation_data=(val_images, val_labels))

Epoch 1/5
430/430 [==============================] - 173s 400ms/step - loss: 0.3834 - accuracy: 0.9035 - val_loss: 0.0797 - val_accuracy: 0.9729
Epoch 2/5
430/430 [==============================] - 185s 431ms/step - loss: 0.0865 - accuracy: 0.9700 - val_loss: 0.0577 - val_accuracy: 0.9817
Epoch 3/5
430/430 [==============================] - 199s 462ms/step - loss: 0.0726 - accuracy: 0.9755 - val_loss: 0.0481 - val_accuracy: 0.9831
Epoch 4/5
430/430 [==============================] - 189s 440ms/step - loss: 0.0557 - accuracy: 0.9817 - val_loss: 0.0405 - val_accuracy: 0.9862
Epoch 5/5
430/430 [==============================] - 182s 423ms/step - loss: 0.0512 - accuracy: 0.9827 - val_loss: 0.0375 - val_accuracy: 0.9870
```

Out of these 5 models, Model 2 performed the best on test data with an accuracy of 99% as seen before.

Part 3: Inference

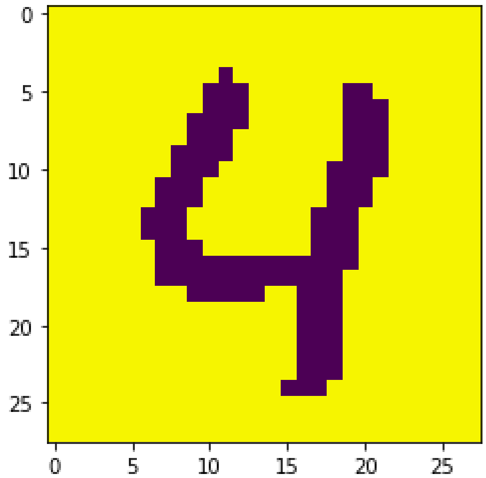
Handwritten digit 4

```
<matplotlib.image.AxesImage at 0x7fe574e9c710>
```



Binary Image

<matplotlib.image.AxesImage at 0x7fe5



Prediction

```
[80] bpred = mmodel.predict(bimage)
     print(bpred)
```

```
[[0.00335884]]
```

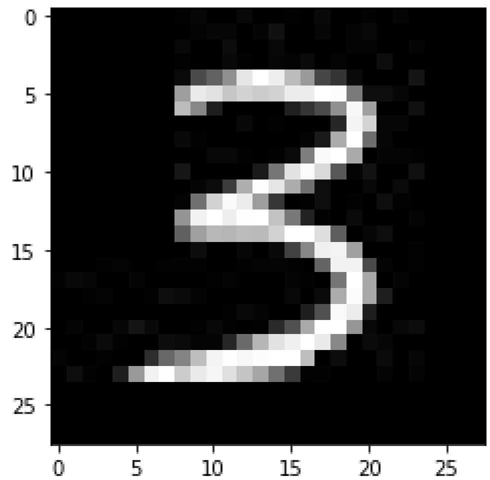
```
[81] if bpred < 0.5:
       print("even number")
     else:
       print("odd number")
```

```
even number
```

The handwritten digit was correctly classified into even number

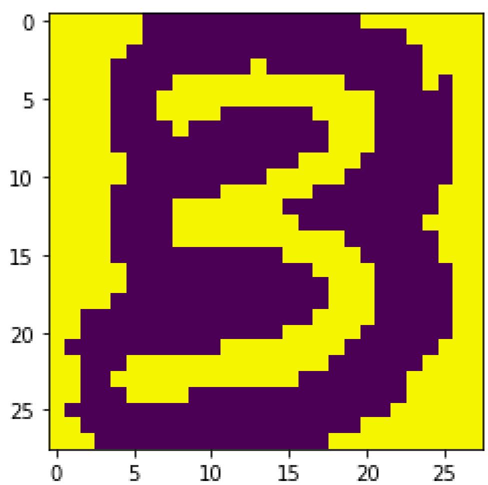
Handwritten digit 3

```
<matplotlib.image.AxesImage at 0x7f
```



Binary image

```
<matplotlib.image.AxesImage at 0x
```



Correctly classified as odd number

```
[113] bpred = mmodel.predict(bimage)
      print(bpred)

      [[0.967953]]
```

```
if bpred < 0.5:
  print("even number")
else:
  print("odd number")

odd number
```

Part 4: Multi-class classification

Basic model - got 72% accuracy on test set

```
[91] c1history = c1model.fit(ctrain_images, ctrain_labels, epochs = 8, batch_size=128, validation_data=(ctest_images, ctest_labels))

Epoch 1/8
391/391 [==============================] - 124s 317ms/step - loss: 1.5187 - accuracy: 0.4568 - val_loss: 1.5496 - val_accuracy: 0.4239
Epoch 2/8
391/391 [==============================] - 118s 302ms/step - loss: 1.1811 - accuracy: 0.5865 - val_loss: 1.1557 - val_accuracy: 0.5855
Epoch 3/8
391/391 [==============================] - 118s 303ms/step - loss: 1.0441 - accuracy: 0.6326 - val_loss: 1.0939 - val_accuracy: 0.6207
Epoch 4/8
391/391 [==============================] - 116s 297ms/step - loss: 0.9473 - accuracy: 0.6663 - val_loss: 1.0007 - val_accuracy: 0.6510
Epoch 5/8
391/391 [==============================] - 119s 305ms/step - loss: 0.8856 - accuracy: 0.6917 - val_loss: 0.9062 - val_accuracy: 0.6833
Epoch 6/8
391/391 [==============================] - 114s 291ms/step - loss: 0.8276 - accuracy: 0.7100 - val_loss: 0.8856 - val_accuracy: 0.6887
Epoch 7/8
391/391 [==============================] - 116s 296ms/step - loss: 0.7820 - accuracy: 0.7237 - val_loss: 0.8091 - val_accuracy: 0.7201
Epoch 8/8
391/391 [==============================] - 114s 291ms/step - loss: 0.7460 - accuracy: 0.7395 - val_loss: 0.7963 - val_accuracy: 0.7278
```

```
[92] predc1 = c1model.evaluate(ctest_images, ctest_labels)
     print("Accuracy is ", predc1[1])

313/313 [==============================] - 6s 18ms/step - loss: 0.7963 - accuracy: 0.7278
Accuracy is  0.7278000116348267
```

Model 2- changed learning rate and epochs - did not perform well

```
from keras.backend import learning_phase
from tensorflow.keras.optimizers import Adam
ad = Adam(learning_rate=0.1)
c2model.compile(optimizer=ad, loss = 'categorical_crossentropy', metrics=['accuracy'])
```

```
c2history = c2model.fit(ctrain_images, ctrain_labels, epochs = 100, batch_size=128, validation_data=(ctest_images, ctest_labels))

Epoch 1/100
391/391 [==============================] - 184s 471ms/step - loss: 1.7156 - accuracy: 0.4097 - val_loss: 1.7724 - val_accuracy: 0.4334
Epoch 2/100
391/391 [==============================] - 181s 462ms/step - loss: 1.7077 - accuracy: 0.4365 - val_loss: 2.0112 - val_accuracy: 0.3683
Epoch 3/100
391/391 [==============================] - 180s 460ms/step - loss: 1.6985 - accuracy: 0.4568 - val_loss: 2.3847 - val_accuracy: 0.4601
Epoch 4/100
391/391 [==============================] - 189s 483ms/step - loss: 1.6445 - accuracy: 0.4938 - val_loss: 2.1060 - val_accuracy: 0.4636
Epoch 5/100
391/391 [==============================] - 180s 460ms/step - loss: 1.6774 - accuracy: 0.5072 - val_loss: 1.5159 - val_accuracy: 0.5448
Epoch 6/100
391/391 [==============================] - 194s 495ms/step - loss: 1.4151 - accuracy: 0.5516 - val_loss: 7.7901 - val_accuracy: 0.5588
Epoch 7/100
391/391 [==============================] - 181s 464ms/step - loss: 1.4834 - accuracy: 0.5509 - val_loss: 1.2989 - val_accuracy: 0.5706
Epoch 8/100
391/391 [==============================] - 180s 461ms/step - loss: 1.4369 - accuracy: 0.5632 - val_loss: 1.3287 - val_accuracy: 0.5885
Epoch 9/100
391/391 [==============================] - 180s 462ms/step - loss: 1.8120 - accuracy: 0.5332 - val_loss: 2.7580 - val_accuracy: 0.4783
Epoch 10/100
```

Model 3 - Implemented using Inception block and achieved accuracy of 63.85%

```
c3history = c3model.fit(ctrain_images, ctrain_labels, epochs = 8, batch_size=128, validation_data=(ctest_images, ctest_labels))
```

```
Epoch 1/8
391/391 [==============================] - 614s 2s/step - loss: 1.6406 - accuracy: 0.4278 - val_loss: 1.3560 - val_accuracy: 0.5127
Epoch 2/8
391/391 [==============================] - 606s 2s/step - loss: 1.2288 - accuracy: 0.5646 - val_loss: 1.2142 - val_accuracy: 0.5736
Epoch 3/8
391/391 [==============================] - 607s 2s/step - loss: 1.0691 - accuracy: 0.6262 - val_loss: 1.1807 - val_accuracy: 0.5851
Epoch 4/8
391/391 [==============================] - 603s 2s/step - loss: 0.9581 - accuracy: 0.6684 - val_loss: 1.1117 - val_accuracy: 0.6077
Epoch 5/8
391/391 [==============================] - 602s 2s/step - loss: 0.8682 - accuracy: 0.7011 - val_loss: 1.0783 - val_accuracy: 0.6218
Epoch 6/8
391/391 [==============================] - 609s 2s/step - loss: 0.7922 - accuracy: 0.7304 - val_loss: 1.0935 - val_accuracy: 0.6201
Epoch 7/8
391/391 [==============================] - 602s 2s/step - loss: 0.7198 - accuracy: 0.7565 - val_loss: 1.0555 - val_accuracy: 0.6359
Epoch 8/8
391/391 [==============================] - 601s 2s/step - loss: 0.6583 - accuracy: 0.7801 - val_loss: 1.0483 - val_accuracy: 0.6385
```

```
predc3 = c3model.evaluate(ctest_images, ctest_labels)
print("Accuracy is ", predc3[1])
```

```
313/313 [==============================] - 32s 104ms/step - loss: 1.0483 - accuracy: 0.6385
Accuracy is  0.6384999752044678
```

Model 4 - Implemented using Residual block and got an accuracy of 73.85 %

```
c4history = c4model.fit(ctrain_images, ctrain_labels, epochs = 8, batch_size=128, validation_data=(ctest_images, ctest_labels))
```

```
Epoch 1/8
391/391 [==============================] - 457s 1s/step - loss: 1.9640 - acc: 0.2608 - val_loss: 1.5287 - val_acc: 0.4293
Epoch 2/8
391/391 [==============================] - 454s 1s/step - loss: 1.4866 - acc: 0.4584 - val_loss: 1.2018 - val_acc: 0.5600
Epoch 3/8
391/391 [==============================] - 439s 1s/step - loss: 1.2296 - acc: 0.5601 - val_loss: 1.0895 - val_acc: 0.6047
Epoch 4/8
391/391 [==============================] - 424s 1s/step - loss: 1.0693 - acc: 0.6228 - val_loss: 1.4909 - val_acc: 0.5227
Epoch 5/8
391/391 [==============================] - 412s 1s/step - loss: 0.9508 - acc: 0.6664 - val_loss: 0.9105 - val_acc: 0.6782
Epoch 6/8
391/391 [==============================] - 418s 1s/step - loss: 0.8513 - acc: 0.7041 - val_loss: 0.8251 - val_acc: 0.7072
Epoch 7/8
391/391 [==============================] - 413s 1s/step - loss: 0.7749 - acc: 0.7352 - val_loss: 0.8392 - val_acc: 0.7120
Epoch 8/8
391/391 [==============================] - 413s 1s/step - loss: 0.7085 - acc: 0.7578 - val_loss: 0.7558 - val_acc: 0.7385
```

Out of the 4 models, the residual block model performed well with 73.85% accuracy on test data and after that, the basic model performed well

References

https://www.tensorflow.org/