National College of Ireland

Configuration Manual

MSc Research Project
MSc. Data Analytics

Manish Bajpai
Student ID: x19229887

School of Computing
National College of Ireland

Supervisor: Dr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Manish Bajpai		
Student ID:	x19229887		
Programme:	MSc. Data Analytics	**Year:**	2021/22
Module:	Research Project		
Lecturer:	Dr. Hicham Rifai		
Submission Due Date:	31/01/2022		
Project Title:	Handwritten Signature Verification using Deep Learning Technique in Conjunction with Image Processing		
Word Count:	1962	**Page Count**: 14	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Manish Bajpai
Date:	30/01/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Manish Bajpai
x19229887

1 Introduction

This configuration manual contains all the necessary steps to build a VGGNet model to recognize offline handwritten signature images. It includes all the steps and the methodology opted for data gathering to the model building phase and evaluation with the specific evaluation metrics. It contains a code abstract along with the and steps to reproduce the Model building phase.

2 Hardware and Software Configurations

Table 1 shows the required hardware specification required to build a VGGNet model for recognition of the handwritten signature images and Table 2 shows the software required for this case study. This section will give the Google colaboratory setup and all the steps performed. Steps performed to setup Colab has been discussed below.

Table 1: Hardware Configurations Adopted

Host Machine	HPE EliteBook with i5 Processor
RAM	8 GB
GPU	Google Colaboratory integrated GPU with 80 GB free storage and 13 GB RAM

Table 2: Software Employed

Programming language	Python
Cloud environment	Google Collaboratory
Browser	Google chrome

3 Colaboratory Setup

This section will give the Google colaboratory setup and all the steps performed. Step perform to setup Colab has been discussed below.

1. Google drive is mounted from which colab will be set up. Figure 1 shows the mounting of google drive in Google Colab.

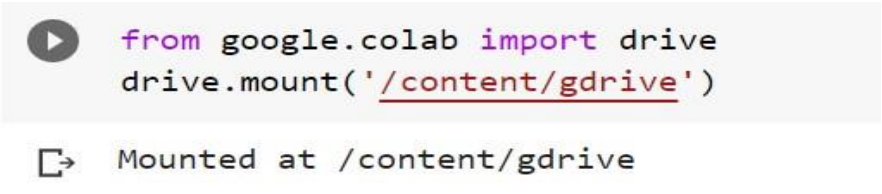

Figure 1: Mounting of Google Drive

2. In the next step provide the path in which content of dataset will be stored. Figure 2 shows the path where data is being stored directly from the Kaggle website.

```
[ ] import os
    os.environ['KAGGLE_CONFIG_DIR'] = "/content/gdrive/My Drive/Final Thesis"
```

Figure 2: Path to store image Dataset

3. Next and the most important step in exporting data directly to Google colaboratory is to download the Kaggle.Json file which will be placed in the folder path mention above. Figure 3 shows the API token in the Kaggle website from where Kaggle. Json file will be downloaded.

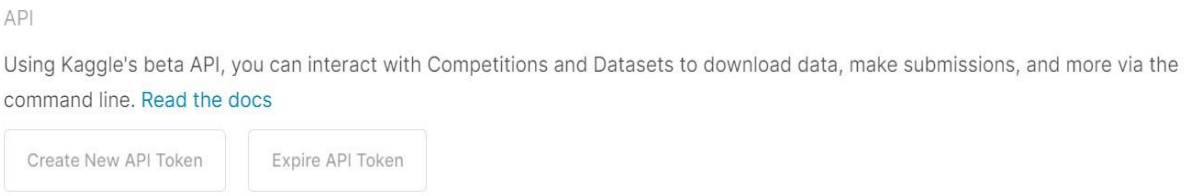

Figure 3: Importing Kaggle JSON

4. 4th step involves downloading all the images from the Kaggle dataset to the Google drive path mention in step 2.

5. There is a command to download folders which is shown below in Figure 4, as by using the Kaggle API command, the entire process gets automated as the entire dataset can be directly imported to Google drive and from there the data can be fetched. Figure 4 shows the command for downloading the image dataset directly from the Kaggle website through API.

```
!kaggle datasets download -d robinreni/signature-verification-dataset
```

Figure 4: Downloading handwritten Signature Images

6. The next process is to unzip the images. Figure 5 shows the unzipping command.

```
[ ] !unzip \*.zip
```

Figure 5: Unzipping the folder

4 Training and Test Images

The images are divided into two groups training and testing folders. During this phase, the necessary libraries, such as NumPy and Pandas, are installed. A directory for training and testing has been created. Figure 6 splitting of the training and testing folder.

```
[ ] import numpy as np # linear algebra
    import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

[ ] train_dir="/content/gdrive/My Drive/Final Thesis/sign_data/train/"
    test_dir="/content/gdrive/My Drive/Final Thesis/sign_data/test/"
```

Figure 6: Splitting the directories for training and Test set images

5 Image-preprocessing

In image pre-processing various step has been performed in this case study which has been discussed below.

1. First the images obtained from the dataset are assigned to their class labels. In this case study, two class labels are present the Genuine and Forge handwritten signature image. Also, feature vector formation was done as a part of pre-processing. The same thing has been employed in the test dataset. For feature vector formation Cv2 library is utilized along with **imread modules.glob library** is imported for the specific pattern path and it returns the value if a pattern exists. In this section Resizing of the images has been also done as a part of pre-processing. Figure 7 shows Resizing and feature vector formation.

```
import cv2
import os
import glob

train_data = []
train_labels = []

for per in os.listdir('/content/gdrive/My Drive/Final Thesis/sign_data/train/'):
    for data in glob.glob('/content/gdrive/My Drive/Final Thesis/sign_data/train/'+per+'/*.*'):
        img = cv2.imread(data)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (SIZE,SIZE))
        train_data.append([img])
        if per[-1]=='g':
            train_labels.append(np.array(1))
        else:
            train_labels.append(np.array(0))

train_data = np.array(train_data)/255.0
train_labels = np.array(train_labels)
```

Figure 7: Feature vector formation in the Training set

2. Same steps have been employed in the test set where the feature vector formation is done as shown in below Figure 8.

```
test_data = []
test_labels = []

for per in os.listdir('/content/gdrive/My Drive/Final Thesis/sign_data/test/'):
    for data in glob.glob('/content/gdrive/My Drive/Final Thesis/sign_data/test/'+per+'/*.*'):
        img = cv2.imread(data)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (SIZE,SIZE))
        test_data.append([img])
        if per[-1]=='g':
            test_labels.append(np.array(1))
        else:
            test_labels.append(np.array(0))

test_data = np.array(test_data)/255.0
test_labels = np.array(test_labels)
```

Figure 8: Feature vector formation in the Test set

3. Training data is reshaped with the help of reshape function in python. Figure 9 shows reshaping operation on images.

```
[ ]  train_data.shape

     (1649, 1, 224, 224, 3)

[ ]  train_data = train_data.reshape(-1, SIZE,SIZE, 3)
     test_data = test_data.reshape(-1, SIZE,SIZE, 3)

[ ]  train_data.shape

     (1649, 224, 224, 3)
```

Figure 9: Reshape of the Training set

4. Reshuffling has been performed on the images to reduce bias also reducing the training time for the model. It also reduces the cross entropy loss and aid in achieving better performance for the model. Shuffling has been performed in both training and test set. Figure 10 shows shuffle operation images

```
[ ] from sklearn.utils import shuffle
    train_data,train_labels = shuffle(train_data,train_labels)
    test_data,test_labels = shuffle(test_data,test_labels)
```

Figure 10: Reshuffling Test and Train set of images

6 Importing all the necessary libraries for building VGGNet Model

For building the VGGNet model mainly Keras and TensorFlow library has been used.

1. Various libraries from Tensorflow and keras like optimizer , adam, application, and Dropout have been imported. Figure 11 shows the import of necessary libraries.

```
import tensorflow as tf
from keras.models import Sequential, Model, load_model
from tensorflow.keras import applications
from keras import optimizers
from keras.layers import Dropout, Flatten, Dense
from tensorflow.keras.optimizers import Adam
```

Figure 11: Importing the libraries

7 Modelling of VGG16

VGG16 neural network has been utilized to recognize handwritten signature images. The advantage of using VGG16 is it has 13 convolution layers and 3 fully connected layers which help in achieving the better performance of the model as compared to other state of art. Figure 12 shows the VGGNet Model.

1. First base model of VGG16 has been created and Imagenet weights have been used for the VGG16 model. Also, the Application library has been imported from Tensorflow.
2. The next step is to add a sequential layer by using add function. VGGG16 is a pre-trained model which consists of 13 convolution layers and 3 dense layers or fully connected layers.

3. After the Sequential layer, the output of the convolution layer is flattened with the help of the flattened layer. This layer aims to convert images to a one-dimensional array.

4. After flattening the 1-dimensional array output goes directly to the Dense layer which is also called a fully connected layer. The most important thing in the dense layer is the use of the softmax function to bring non-linearity, so that two label classes Genuine and forge can be easily separable.

```
base_model = applications.VGG16(weights='imagenet', include_top=False, input_shape=(224,224,3))
base_model.summary()

add_model = Sequential()
add_model.add(Flatten(input_shape=base_model.output_shape[1:]))
add_model.add(Dense(256, activation='relu'))
add_model.add(Dense(2, activation='softmax'))
```

Figure 12: Building base model with layers

5. The next step is to add hyperparameters utilized for building the model. For this case study, Cross entropy loss is used as this particular research has two class labels. Adam Optimizer has been used for stochastic gradient. Figure 13 shows the Model compilation.

```
model = Model(inputs=base_model.input, outputs=add_model(base_model.output))
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
metrics=['accuracy'])

model.summary()
```

Figure 13: Hyper Parameter Utilized

6. Callback and Early stopping have been used to train the VGG16 Model. Call back help in adjusting the weight of the layers by learning parameter. The main of Early stopping is to stop the model from running on the number of mentioned epochs if the value of the parameter set is not changing. In this case study, an Early stop has been made on the parameter validation loss. Figure 14 shows the callback and Earlystop method.

```
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping, ReduceLROnPlateau, TensorBoard
earlyStopping = EarlyStopping(monitor='val_loss',
                              min_delta=0,
                              patience=3,
                              verbose=1)

early_stop=[earlyStopping]
```

Figure 14: Callback and Early stop modules in VGG16

7. The fit function also referred to as batch size has been set to 32 as shown in Figure 15. The splitting of the data is done in a 70:30 ratio in training and test set.

```
EPOCHS = 10
BS = 64
progess = model.fit(train_data,train_labels, batch_size=32,epochs=EPOCHS, callbacks=early_stop,validation_split=.3)
```

Figure 15: Fitting the VGG16 Model

8 Experiment 1 setup along with Evaluation

The first experiment setup was done by changing the learning hyperparameter value to 0.0004 and seeing the model behavior in terms of evaluation metrics of the training set and test set. The VGG16 model has been tested on 3,5 and 10 number epochs.

From Figure 16 it's visible the training set accuracy is very less and it's around 69 percent.

```
Epoch 1/10
37/37 [==============================] - 65s 1s/step - loss: 0.8789 - accuracy: 0.5069 - val_loss: 0.6928 - val_accuracy: 0.5414
Epoch 2/10
37/37 [==============================] - 32s 860ms/step - loss: 1.6263 - accuracy: 0.5243 - val_loss: 0.6918 - val_accuracy: 0.5414
Epoch 3/10
37/37 [==============================] - 32s 856ms/step - loss: 0.6917 - accuracy: 0.5364 - val_loss: 0.6912 - val_accuracy: 0.5414
Epoch 4/10
37/37 [==============================] - 32s 855ms/step - loss: 0.6912 - accuracy: 0.5364 - val_loss: 0.6906 - val_accuracy: 0.5414
Epoch 5/10
37/37 [==============================] - 32s 857ms/step - loss: 0.6909 - accuracy: 0.5364 - val_loss: 0.6904 - val_accuracy: 0.5414
Epoch 6/10
37/37 [==============================] - 32s 857ms/step - loss: 0.6909 - accuracy: 0.5364 - val_loss: 0.6902 - val_accuracy: 0.5414
Epoch 7/10
37/37 [==============================] - 32s 856ms/step - loss: 0.6908 - accuracy: 0.5364 - val_loss: 0.6900 - val_accuracy: 0.5414
Epoch 8/10
37/37 [==============================] - 32s 857ms/step - loss: 0.6906 - accuracy: 0.5364 - val_loss: 0.6899 - val_accuracy: 0.5414
Epoch 9/10
37/37 [==============================] - 32s 858ms/step - loss: 0.6906 - accuracy: 0.5364 - val_loss: 0.6899 - val_accuracy: 0.5414
Epoch 10/10
37/37 [==============================] - 32s 857ms/step - loss: 0.6906 - accuracy: 0.5364 - val_loss: 0.6898 - val_accuracy: 0.5414
```

Figure 16: Training set accuracy

Also, the layer diagram of the VGG16 model is shown in Figure 17. This Figure shows all the layers involved in VGG16. Layers start with the input layers and then a stack of convolutional layers started along with the pooling layer. The main aim of the pooling layer is to reduce the size of the handwritten signature images keeping all features of the image enacted.

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Figure 17: layers of VGG16

Predict variable is constructed in which test data prediction is estimated with the help of predict function. Figure 18 shows the same.

```
pred = model.predict(test_data)
```

Figure 18: Prediction Test data

From the scikit learn evaluation metrics has been called such as Classification report module which provides Precision, Recall, and accuracy of the test data, which gives the overall accuracy of the model. Figure 19 showcases the accuracy achieved in the test set. Two graphs have been plotted between training accuracy and validation accuracy With the help history module has been used to record validation and training accuracy. The same has been shown in figure 20. It shows there is a big gap between training and test accuracy which shows the model is not efficient in terms of evaluation metrics.

```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
print(classification_report(np.argmax(pred,axis=1), test_labels))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67       500
           1       0.00      0.00      0.00         0

    accuracy                           0.50       500
   macro avg       0.50      0.25      0.34       500
weighted avg       1.00      0.50      0.67       500
```

Figure 19: Evaluation Metrics on Test data

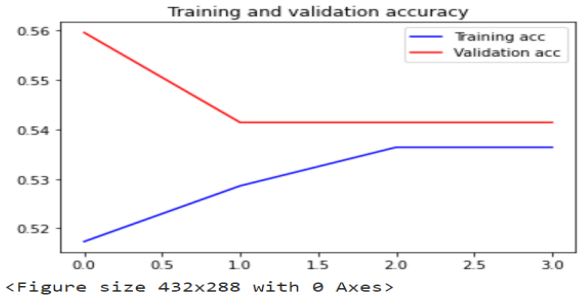

Figure 20: Training accuracy and validation accuracy

The second graph is plotted between Validation loss and Training loss. Figure 21 demonstrates the same. It shows in the training set the loss is increasing which is not good for the efficient model. For the validation set, the loss remains constant therefore the line is parallel.

```
[ ] acc = progess.history['accuracy']
    val_acc = progess.history['val_accuracy']
    loss = progess.history['loss']
    val_loss = progess.history['val_loss']

    epochs = range(len(acc))

    plt.plot(epochs, acc, 'b', label='Training acc')
    plt.plot(epochs, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.figure()
```

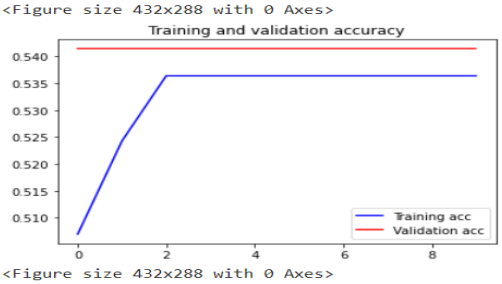

Figure 21: Training loss and validation loss

9 Experiment Setup 2 with evaluation metrics

The second experiment Setup was done as the efficiency of the previous VGG16 model was not good. So in the second setup, the new convolution layer was added and dropout was also added to increase the efficacy of the VGG16 model. In this setup extra dense layer has also been added. The learning parameter was set to 0.0008. Figure 22 shows the code for adding an extra convolutional layer along with the dense layer.

```
base_model = applications.VGG16(weights='imagenet', include_top=False, input_shape=(224,224,3))
base_model.summary()

add_model = Sequential()
add_model.add(Convolution2D(64, (kernel_size, kernel_size), input_shape=input_shape))
add_model.add(Activation('relu'))
add_model.add(MaxPooling2D(pool_size=(2, 2)))
add_model.add(Flatten(input_shape=base_model.output_shape[1:]))
add_model.add(Dense(256, activation='relu'))
add_model.add(Dense(2, activation='softmax'))
add_model.add(Dense(2, activation='softmax'))
add_model.add(Dropout(0.5))

model = Model(inputs=base_model.input, outputs=add_model(base_model.output))
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=0.0008),
metrics=['accuracy'])
```

Figure 22: Block code for adding extra Convolution layers

The accuracy was achieved by the VGG16 model after changing the learning parameter. It is seen that after adding an extra convolutional layer and adding dropout the overall accuracy of the training set didn't improve. Figure 23 showcases the same.

```
EPOCHS = 10
BS = 64
progess = model.fit(train_data,train_labels, batch_size=32,epochs=EPOCHS, callbacks=early_stop,validation_split=.3)
```

```
Epoch 1/10
37/37 [==============================] - 79s 1s/step - loss: 3.4717 - accuracy: 0.5009 - val_loss: 2.2822 - val_accuracy: 0.5556
Epoch 2/10
37/37 [==============================] - 39s 1s/step - loss: 0.9369 - accuracy: 0.5113 - val_loss: 0.6896 - val_accuracy: 0.5556
Epoch 3/10
37/37 [==============================] - 40s 1s/step - loss: 0.6917 - accuracy: 0.5303 - val_loss: 0.6886 - val_accuracy: 0.5556
Epoch 4/10
37/37 [==============================] - 39s 1s/step - loss: 0.6930 - accuracy: 0.5269 - val_loss: 0.6992 - val_accuracy: 0.4404
Epoch 5/10
37/37 [==============================] - 39s 1s/step - loss: 0.6870 - accuracy: 0.5511 - val_loss: 0.6842 - val_accuracy: 0.5556
Epoch 6/10
37/37 [==============================] - 39s 1s/step - loss: 0.6872 - accuracy: 0.5425 - val_loss: 0.6923 - val_accuracy: 0.5131
Epoch 7/10
37/37 [==============================] - 39s 1s/step - loss: 0.6952 - accuracy: 0.5477 - val_loss: 1.8395 - val_accuracy: 0.4242
Epoch 8/10
37/37 [==============================] - 39s 1s/step - loss: 1.5780 - accuracy: 0.4948 - val_loss: 0.6917 - val_accuracy: 0.5556
Epoch 00008: early stopping
```

Figure 23: Accuracy of the Training set

Also, the accuracy achieved in the test set is not satisfactory. The same has been demonstrated in the graph between validation and Test accuracy which is shown in Figure 24.

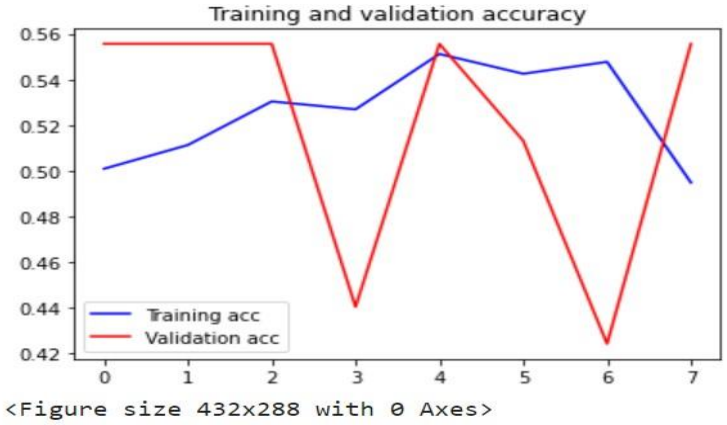

Figure 24: Accuracy of Training and Validation accuracy

10 Experiment Setup 3 and Optimized Model for a handwritten signature image

The third experiment was the final setup done and its optimized VGG16 model with the better efficiency achieved than the rest of the model obtained from experiments 1 and 2 respectively. During this experiment, it was noticed adding an extra convolution layer didn't improve the performance of the model and even the dropout value also didn't add that much to the performance of the model. Therefore these were removed from the final and Optimized model. The learning rate was set to 1e-4. Figure 25 showcase the change of learning parameters in the final code version.

```
model = Model(inputs=base_model.input, outputs=add_model(base_model.output))
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
metrics=['accuracy'])
```

Figure 25: Code change for changing learning parameter

The accuracy of the final model achieved is better than the 1 and 2 models in the training as well as in the test set. As the number of Epochs increases the validation accuracy is increased. Figure 26 demonstrates the same through the output.

```
progess = model.fit(train_data,train_labels, batch_size=32,epochs=EPOCHS, callbacks=early_stop,validation_split=.3)

Epoch 1/10
37/37 [==============================] - 41s 1s/step - loss: 0.7371 - accuracy: 0.5286 - val_loss: 0.6564 - val_accuracy: 0.5919
Epoch 2/10
37/37 [==============================] - 39s 1s/step - loss: 0.6350 - accuracy: 0.6343 - val_loss: 0.4546 - val_accuracy: 0.7758
Epoch 3/10
37/37 [==============================] - 39s 1s/step - loss: 0.4022 - accuracy: 0.8154 - val_loss: 0.2906 - val_accuracy: 0.8768
Epoch 4/10
37/37 [==============================] - 39s 1s/step - loss: 0.1585 - accuracy: 0.9367 - val_loss: 0.1138 - val_accuracy: 0.9495
Epoch 5/10
37/37 [==============================] - 40s 1s/step - loss: 0.1053 - accuracy: 0.9627 - val_loss: 0.1928 - val_accuracy: 0.9414
Epoch 6/10
37/37 [==============================] - 39s 1s/step - loss: 0.0652 - accuracy: 0.9714 - val_loss: 0.1061 - val_accuracy: 0.9556
Epoch 7/10
37/37 [==============================] - 39s 1s/step - loss: 0.0284 - accuracy: 0.9931 - val_loss: 0.1055 - val_accuracy: 0.9616
Epoch 8/10
37/37 [==============================] - 39s 1s/step - loss: 0.0208 - accuracy: 0.9913 - val_loss: 0.0978 - val_accuracy: 0.9677
Epoch 9/10
37/37 [==============================] - 39s 1s/step - loss: 0.0136 - accuracy: 0.9974 - val_loss: 0.0781 - val_accuracy: 0.9717
Epoch 10/10
37/37 [==============================] - 39s 1s/step - loss: 0.0437 - accuracy: 0.9870 - val_loss: 0.2088 - val_accuracy: 0.9475
```

Figure 26: Accuracy of the Training set

Figure 27 showcases the accuracy of the test set along with the other evaluation metrics which have been called of with the aid of the Scikit learn library in python.it seems the test set has also achieved better accuracy.

```
[53] from sklearn.metrics import confusion_matrix, classification_report
     from sklearn.metrics import accuracy_score
     print(classification_report(np.argmax(pred,axis=1), test_labels))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	253
1	1.00	1.00	1.00	247
accuracy			1.00	500
macro avg	1.00	1.00	1.00	500
weighted avg	1.00	1.00	1.00	500

Figure 27: Accuracy of the Test set

Below Fig 28 demonstrates the layers in the VGG16 Model that includes 13 Convolutional layers and 3 fully connected layers.

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

Figure 28: Layers of VGG16

11 Conclusion

The final model obtained from experiment setup 3 is the better model in terms of performance and higher evaluation metrics. Figure 29 shows case the also the validity and training loss function is decreasing as the number of epochs increases. That's the reason for the better performance of the model as the loss function is decreasing.

```
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

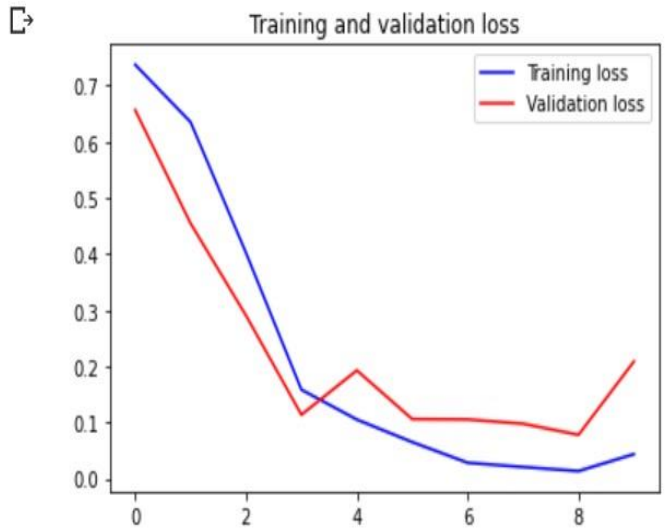

Figure 29: Accuracy of the Test set