Configuration Manual

Machine Learning Based Approach in Detection and Classification of Tomato Plant Leaf Diseases
MSc in Data Analytics

Rajath Ramakrishna
Student ID: x18130721

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rajath Ramakrishna
Student ID:	X18130721
Programme:	MSc in Data Analytics **Year:** 2019-2020
Module:	MSc Research Project
Lecturer:	Dr. Muhammad Iqbal
Submission Due Date:	12/12/2019
Project Title:	Machine Learning Based Approach in Detection and Classification of Tomato Plant Leaf Diseases

Word Count: **Page Count:**

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: Rajath Ramakrishna

Date: 12/12/2019

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

Rajath Ramakrishna
Student ID: x18130721

1 Introduction

The configuration manual outlines the steps and procedures to be followed while running the implemented scripts for the current research project. This will allow for smooth running of the code without any glitches. This also includes information about the machine's hardware configuration in which the scripts are executed and provides the same minimum recommended configuration. Following these steps would help replicate the project's results. This can then be analysed and is straightforward to conduct future research.

2 System Specification

2.1 Hardware configuration

As mentioned below, the hardware specification of the computer on which the work was performed is:
Processor: Intel Core i5 – 8250U CPU @ 1.60GHz
RAM: 8 GB
Storage: 256GB SSD
Operating System: 64-bit operating system, Windows 10 Home

2.2 Software configuration

Software used to implement the project is Python based Jupyter notebook IDE (Integrated Development Environment) which is available within the Anaconda package. Forthcoming sections illustrates the steps to be followed in executing the developed scripts.

3 Downloads and Installation

- **Python**

The research project is carried out using Python since it has a significant number of libraries and supporting models of machine and deep learning. It also comes with several modules that help make it easier to pre-process and alter images, making it easier to use and implement. Therefore, the fundamental requirement on the computer running the script is to have downloaded on it the latest version of Python. This can be done by proceeding to the python website[1] download page and installing the downloadable installer of the specified version based on the OS of the machine running it. Fig 1 illustrates the screenshot of the website from where the latest version is installed. Once downloaded, file must be installed by following the installation instructions.

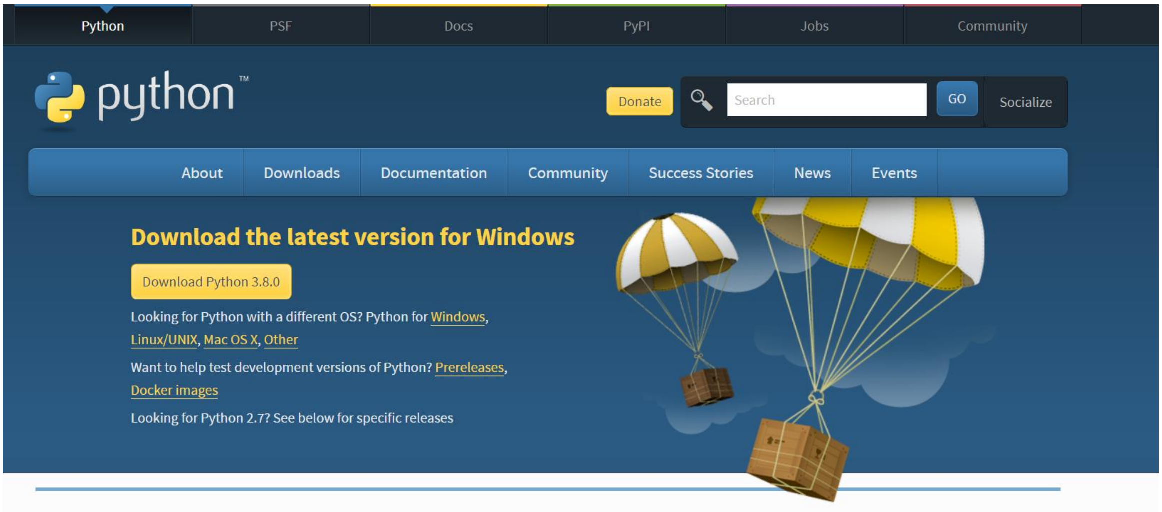


Figure 1. Download Page of Python

After the installation is successful it can be verified in windows command prompt by using 'python –version' command. It provides the version number of the Python installed.

- ## Anaconda

Anaconda is the next package to be installed. It provides various python-based IDEs that are user-friendly that can be used for code development and results visualization. Jupyter Notebook and Spyder are the most common of the IDE's available in Anaconda Navigator on installation. Anaconda can be downloaded from the official website[2]. Downloadable installer is shown in Fig 2. Package comes for different OS hence the OS specific installer needs to be downloaded

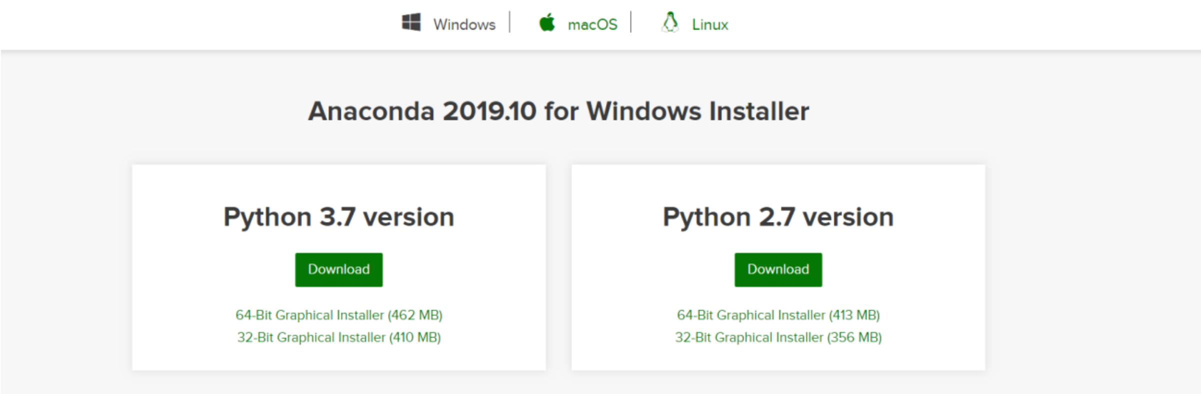


Figure 2. Download page of Anaconda Installer

[1] https://www.python.org/downloads/
[2] https://www.anaconda.com/distribution/

As shown in Fig 3, upon successful installation of Anaconda Navigator, it will display multiple IDE that can be selected for development. Of which Jupyter IDE is used in this research project.

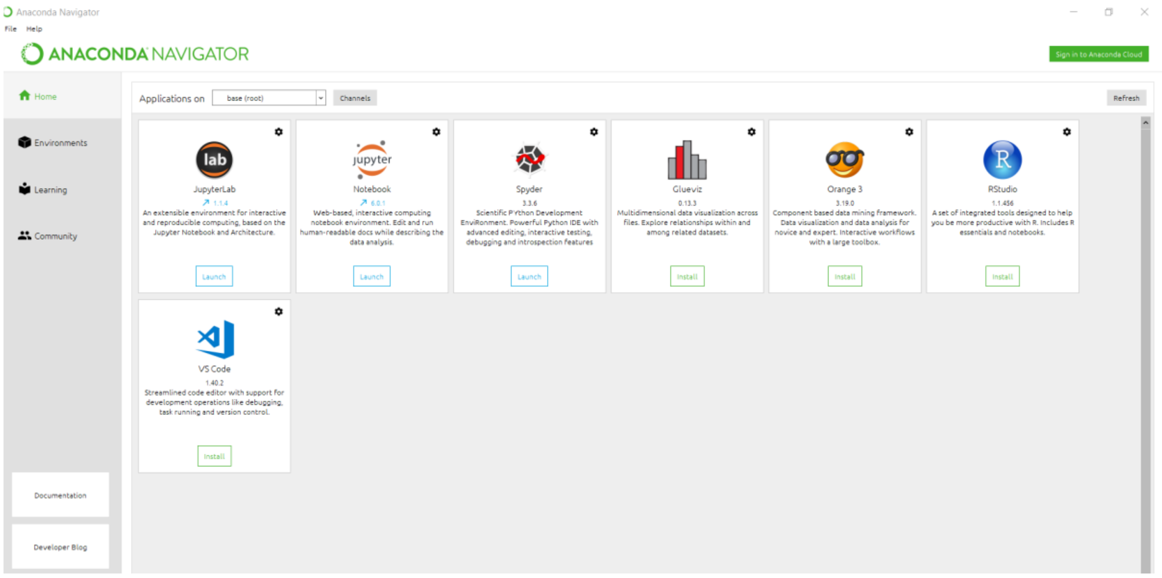


Figure 3. Anaconda Navigator

▪ Data Source

This research uses the colored images of tomato plant leaves from the publicly accessible PlantVillage dataset from GitHub[3]. This open repository contains color, segmented and grayscale images of several plants. Thus, only colored images of tomato plants must be downloaded, and the other images must be removed from the downloaded folder.

▪ Project Development

Jupyter notebook must be launched from the installed navigator. As shown in Fig 4. new tab is opened in the browser

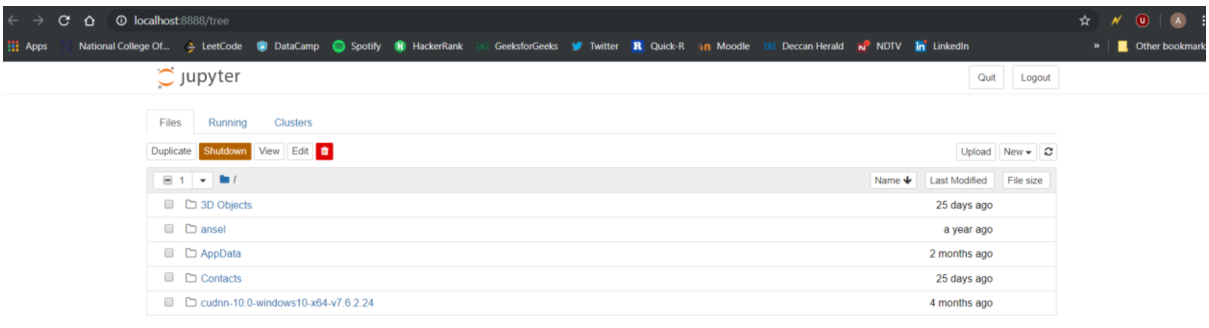


Figure 4. Home page of Jupyter Notebook

[3] https://github.com/spMohanty/PlantVillage-Dataset

Coding can be started by clicking the new icon on the top left corner and selecting Python 3 which launches a new page for coding. As the project is being built using machine and deep learning techniques based on transfer learning, additional python libraries also need to be installed when required. These can be installed using pip install command in the windows command prompt as shown in Fig. 5

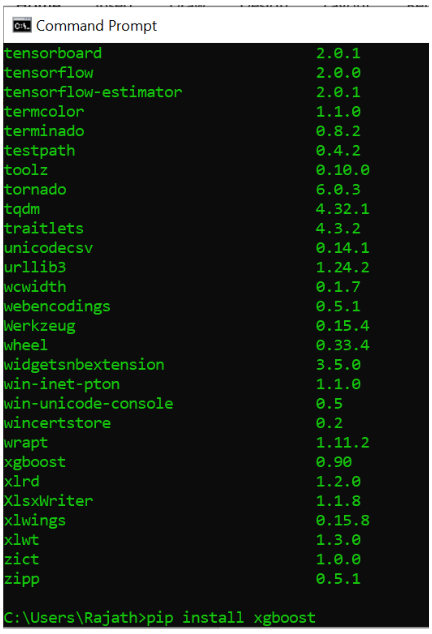


Figure 5. Command prompt for Python library installations

Firstly, few of the standard libraries that are required to build image classification models include

- TensorFlow 2.0.0
- Keras 2.3.1
- Keras-Applications 1.0.8
- Keras-Preprocessing 1.1.0
- Numpy 1.16.5
- Scikit-Image 0.16.2
- Scikit-Learn 0.21.3
- Sklearn 0.0
- Opencv-contrib-python 4.1.1.26
- Matplotlib 3.1.1

Once the coding has been successfully completed, the script can be executed by running the jupyter notebook command as shown in Fig. 6 or by running the code individually by blocks. If there are errors present in the code, they will be shown below the code block that can then be used to debug.

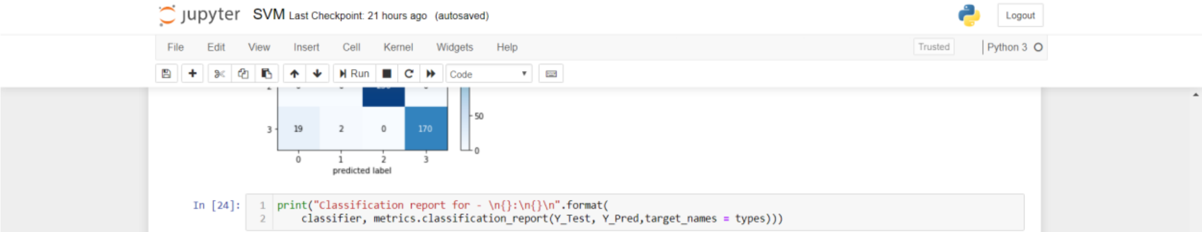


Figure 6. Running the script

Model Development: Basic CNN

```
In [1]: import numpy as np
        import pickle
        import cv2
        import random
        from sklearn.preprocessing import LabelEncoder
        from os import listdir
        from sklearn.preprocessing import LabelBinarizer
        from keras.models import Sequential
        from keras.layers.normalization import BatchNormalization
        from keras.layers.convolutional import Conv2D
        from keras.layers.convolutional import MaxPooling2D
        from keras.layers.core import Activation, Flatten, Dropout, Dense
        from keras.layers.pooling import GlobalAveragePooling2D
        from keras import backend as K
        from keras.preprocessing.image import ImageDataGenerator
        from keras.optimizers import Adam, SGD, Adamax, Nadam
        from keras.preprocessing import image
        from keras.preprocessing.image import img_to_array
        from sklearn.preprocessing import MultiLabelBinarizer
        from sklearn.model_selection import train_test_split
        from skimage.io import imread,imshow
        import matplotlib.pyplot as plt
        from sklearn.metrics import roc_curve, auc,precision_recall_curve,confusion_matrix
        from mlxtend.plotting import plot_confusion_matrix
        from datetime import datetime
        from sklearn import metrics

        Using TensorFlow backend.
```

```
In [2]: def timer(start_time=None):
            if not start_time:
                start_time = datetime.now()
                return start_time
            elif start_time:
                thour, temp_sec = divmod((datetime.now() - start_time).total_seconds(), 3600)
                tmin, tsec = divmod(temp_sec, 60)
                print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round(tsec, 2)))
```

Figure 7

- The first block shows the list of libraries installed for the implementation of this model
- Second block shows timer function implemented to calclulate the training time of all the models

```
In [3]: EPOCHS = 25
        INIT_LR = 0.01
        BS = 32
        default_image_size = tuple((128, 128))
        image_size = 0
        directory_root = 'C://NCI/Sem 3/tomato plant disease dataset/PlantVillage/'
        width=128
        height=128
        depth=3
```

```
In [4]: images=[]
        def convert_image_to_array(image_dir):
            try:
                image = cv2.imread(image_dir)
                if image is not None :
                    image = cv2.resize(image, default_image_size)
                    images.append(image)
                    return img_to_array(image)
                else :
                    return np.array([])
            except Exception as e:
                print(f"Error : {e}")
                return None
```

Figure 8

This block shows how the image data is converted to numpy arrays for building models

```
In [5]: image_list, label_list = [], []
        try:
            print("[INFO] Loading images ...")
            plant_disease_folder_list = listdir(directory_root)
            print (plant_disease_folder_list)

            for plant_disease_folder in plant_disease_folder_list:
                print(f"[INFO] Processing {plant_disease_folder} ...")
                plant_disease_image_list = listdir(f"{directory_root}/{plant_disease_folder}/")


                for image in plant_disease_image_list[:]:
                    image_directory = f"{directory_root}/{plant_disease_folder}/{image}"
                    if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True:
                        image_list.append(convert_image_to_array(image_directory))
                        label_list.append(plant_disease_folder)
            print("[INFO] Image loading completed")
        except Exception as e:
            print(f"Error : {e}")

        [INFO] Loading images ...
        ['Tomato_healthy', 'Tomato_Late_blight', 'Tomato_Leaf_Mold', 'Tomato_mosaic_virus', 'Tomato_Septoria_leaf_spot']
        [INFO] Processing Tomato_healthy ...
        [INFO] Processing Tomato_Late_blight ...
        [INFO] Processing Tomato_Leaf_Mold ...
        [INFO] Processing Tomato_mosaic_virus ...
        [INFO] Processing Tomato_Septoria_leaf_spot ...
        [INFO] Image loading completed
```

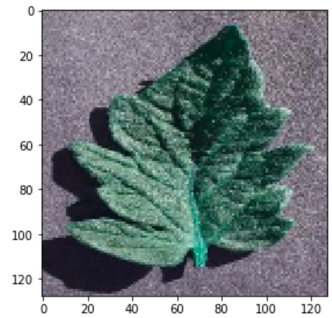
```
In [6]: image_size = len(image_list)
        print(image_size)

        6595
```

Figure 9

This block shows the images being loaded into the model

```
In [7]: imshow(images[600])
Out[7]: <matplotlib.image.AxesImage at 0x1b37f3d4748>
```



```
In [8]: label_binarizer = LabelBinarizer()
        image_labels = label_binarizer.fit_transform(label_list)
        pickle.dump(label_binarizer,open('label_transform.pkl', 'wb'))
        n_classes = len(label_binarizer.classes_)

In [9]: print(label_binarizer.classes_)

        ['Tomato_Late_blight' 'Tomato_Leaf_Mold' 'Tomato_Septoria_leaf_spot'
         'Tomato_healthy' 'Tomato_mosaic_virus']
```

Figure 10

This block shows the image of sample tomato leaf and the following block is used to encode the class names.

```
In [10]: value = []
Tomato_mosaic_virus = Tomato_healthy = Tomato_Late_blight = Tomato_Leaf_Mold = Tomato_Septoria_leaf_spot = 0

for i in range(len(label_list)):
    if (label_list[i] == 'Tomato__Tomato_mosaic_virus'):
        Tomato_mosaic_virus = Tomato_mosaic_virus + 1
    elif (label_list[i] == 'Tomato_healthy'):
        Tomato_healthy = Tomato_healthy + 1
    elif (label_list[i] == 'Tomato_Late_blight'):
        Tomato_Late_blight = Tomato_Late_blight + 1
    elif (label_list[i] == 'Tomato_Leaf_Mold'):
        Tomato_Leaf_Mold = Tomato_Leaf_Mold + 1
    else:
        TTomato_Septoria_leaf_spot = Tomato_Septoria_leaf_spot + 1

value=[Tomato_mosaic_virus,Tomato_healthy,Tomato_Late_blight,Tomato_Leaf_Mold,Tomato_Septoria_leaf_spot]
types = ('mosaic_virus','Healthy','Late_blight','Leaf_Mold','Septoria_leaf_spot','Two_spotted_spider_mite')
y_pos = np.arange(5)

plt.figure(figsize=(10,10))
plt.bar(y_pos, value, align='center', alpha=0.5)
plt.xticks(y_pos, types)
plt.ylabel('Number of Images')
plt.title('Data Distribution')

plt.show()
```

Figure 11

This is used to plot bar chart which shows the total number of images in each class

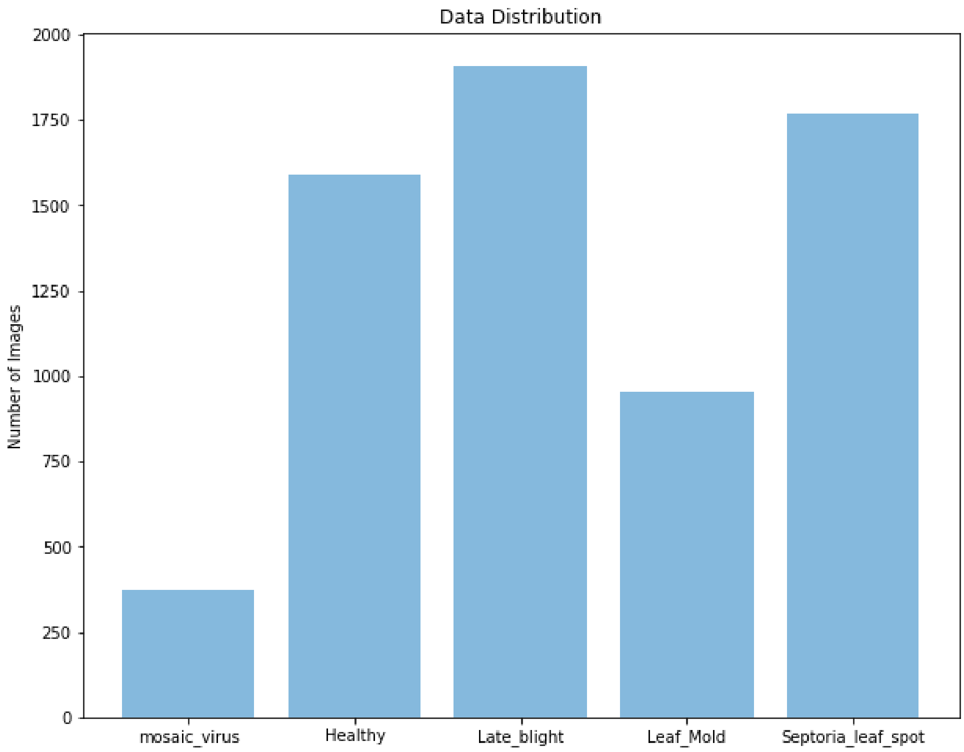


Figure 12

This is the bar chart plotted showing the total number of images in each class.

```
In [11]: np_image_list = np.array(image_list, dtype=np.float16) / 255.0
```

```
In [12]: all_indexes = list(range(len(images)))
         random_indexes = random.sample( all_indexes, 4)
```

```
In [13]: j = 1
         plt.figure( figsize=(15, 6))
         for i in random_indexes:
             plt.subplot(2, 2, j);
             plt.grid(False)
             plt.imshow(images[i])
            # plt.title(tomato_dataset.target_names[tomato_dataset.target[i]])
             j = j + 1
         plt.show()
```

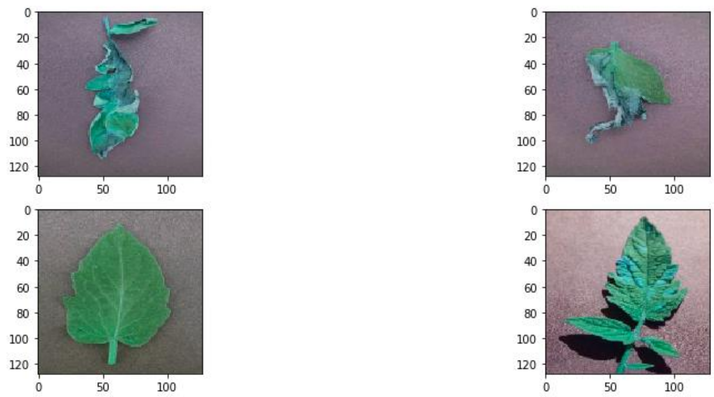


Figure 13

This block displays the random set of tomato leaf images

```
In [14]: print("[INFO] Spliting data to train, test")
         x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state = 42)

         [INFO] Spliting data to train, test

In [15]: y_test.shape

Out[15]: (1319, 5)

In [16]: aug = ImageDataGenerator(
             rotation_range=25, width_shift_range=0.1,
             height_shift_range=0.1, shear_range=0.2,
             zoom_range=0.2,horizontal_flip=True,
             fill_mode="nearest")
```

Figure 14

In this block, the data is split into test and train in the ratio of 80:20.

```
In [22]: model = Sequential()
         inputShape = (height, width, depth)
         chanDim = -1
         if K.image_data_format() == "channels_first":
             inputShape = (depth, height, width)
             chanDim = 1

         model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape, activation = "relu"))
         model.add(BatchNormalization())
         model.add(MaxPooling2D(pool_size=(2, 2)))


         model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape, activation = "relu"))
         model.add(BatchNormalization())
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(64, (3, 3), padding="same",input_shape=inputShape, activation = "relu"))
         model.add(BatchNormalization())
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(64, (3, 3), padding="same",input_shape=inputShape, activation = "relu"))
         model.add(BatchNormalization())
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(128, (3, 3), padding="same",input_shape=inputShape, activation = "relu"))
         model.add(BatchNormalization())
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(GlobalAveragePooling2D())
         model.add(Dense(units=256, activation = "relu"))
         model.add(Dropout(0.25))

         model.add(Dense(units=128, activation = "relu"))
         model.add(Dropout(0.25))

         model.add(Dense(units=n_classes, activation = "softmax"))
```

Figure 15

This block shows the CNN model with 5 convolution layers being built

```
In [23]: model.summary()

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 128, 128, 32)      896
_________________________________________________________________
batch_normalization_5 (Batch (None, 128, 128, 32)      128
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
batch_normalization_6 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
batch_normalization_7 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_8 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_9 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
global_average_pooling2d_2 ( (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
```

Figure 16

The above block gives the summary of different layers

```
In [24]: model.compile(loss= "binary_crossentropy",optimizer = 'Adam',metrics=["accuracy"])

In [25]: start_time = timer(None)

In [26]: history= model.fit_generator(
             aug.flow(x_train,y_train,batch_size=BS),
             validation_data=(x_test,y_test),
             steps_per_epoch =len(x_train) // BS,
             epochs=EPOCHS, verbose=1
         )

Epoch 1/25
164/164 [==============================] - 152s 925ms/step - loss: 0.1941 - accuracy: 0.9215 - val_loss: 1.1388 -
val_accuracy: 0.7252
Epoch 2/25
164/164 [==============================] - 153s 933ms/step - loss: 0.1131 - accuracy: 0.9573 - val_loss: 3.2918 -
val_accuracy: 0.7180
Epoch 3/25
164/164 [==============================] - 154s 936ms/step - loss: 0.0873 - accuracy: 0.9683 - val_loss: 0.3067 -
val_accuracy: 0.8978
Epoch 4/25
164/164 [==============================] - 154s 936ms/step - loss: 0.0823 - accuracy: 0.9706 - val_loss: 1.8857 -
val_accuracy: 0.7741
Epoch 5/25
164/164 [==============================] - 154s 938ms/step - loss: 0.0558 - accuracy: 0.9785 - val_loss: 1.8174 -
val_accuracy: 0.7806
Epoch 6/25
164/164 [==============================] - 154s 941ms/step - loss: 0.0518 - accuracy: 0.9811 - val_loss: 0.1719 -
val_accuracy: 0.9456
Epoch 7/25

In [27]: timer(start_time)

Time taken: 1 hours 4 minutes and 53.58 seconds.
```

Figure 17

This block shows how the model is trained on the training data

In [28]:
```python
Y_pred= model.predict(x_test)
```

In [29]:
```python
y_pred = np.argmax(Y_pred,axis=1)
```

In [30]:
```python
y_test_end = label_binarizer.inverse_transform(y_test)
```

In [31]:
```python
label = LabelEncoder()
y_encoded = label.fit_transform(y_test_end)
```

In [ ]:
```python
def plot_accuracy(hist):
    plt.plot(hist['accuracy'])
    plt.plot(hist['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train',
                'test'],
               loc='upper left')
    plt.show()
```

In [ ]:
```python
plot_accuracy(history.history)
```

In [ ]:
```python
def plot_loss(hist):
    plt.plot(hist['loss'])
    plt.plot(hist['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train',
                'test'],
               loc='upper left')
    plt.show()
```

Figure 18

This block shows how the model is being evaluated

In [32]:
```python
cm = confusion_matrix(y_encoded, y_pred)
print(cm)
```

```
[[301   0  21  66   1]
 [  0  94  11  61   4]
 [  1   1 297  42   7]
 [  0   0   0 338   0]
 [  0   0   0  34  40]]
```

In [33]:
```python
fig, ax = plot_confusion_matrix(conf_mat=cm,
                                show_absolute=True,
                                show_normed=False,
                                colorbar=True,
                               )
plt.show()
```

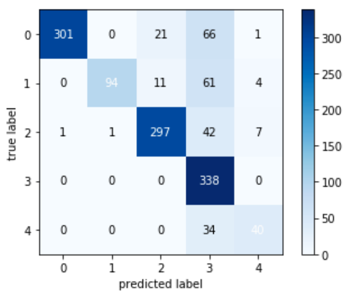


Figure 19

This block generates the confusion matrix for the implemented model

```
In [34]: print("Classification report for - \n{}:\n{}\n".format(
             model, metrics.classification_report(y_encoded, y_pred)))

Classification report for -
<keras.engine.sequential.Sequential object at 0x000001B3A2247EB8>:
              precision    recall  f1-score   support

           0       1.00      0.77      0.87       389
           1       0.99      0.55      0.71       170
           2       0.90      0.85      0.88       348
           3       0.62      1.00      0.77       338
           4       0.77      0.54      0.63        74

    accuracy                           0.81      1319
   macro avg       0.86      0.74      0.77      1319
weighted avg       0.86      0.81      0.81      1319
```

Figure 20

This block generates the classification report to evaluate the model using multiple evaluation metrics

Following blocks of code shows the models defined for various algorithms

Support Vector Machine:

```
In [13]: support_vector = svm.SVC(kernel='linear', probability=True)
         support_vector.fit(x_train, y_train)

Out[13]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
             kernel='linear', max_iter=-1, probability=True, random_state=None,
             shrinking=True, tol=0.001, verbose=False)
```

Figure 21

Random Forest:

```
In [13]: random_forest  = RandomForestClassifier(n_estimators=100, probability=True)
         random_forest.fit(x_train, y_train)

Out[13]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=None, max_features='auto', max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=100,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)
```

Figure 22

VGG16:

```
In [17]: #base_model=MobileNet(weights='imagenet',include_top=False) #imports the mobilenet model and discards the last 1000
         base_model=VGG16(include_top=False, weights='imagenet')

         x=base_model.output
         x=GlobalAveragePooling2D()(x)
         x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and class
         x=Dense(1024,activation='relu')(x) #dense layer 2
         x=Dense(512,activation='relu')(x) #dense layer 3
         preds=Dense(5,activation='softmax')(x) #final layer with softmax activation
```

Figure 23

VGG19:

```
In [17]: #base_model=MobileNet(weights='imagenet',include_top=False) #imports the mobilenet model and discards the last 1000
         base_model=VGG19(include_top=False, weights='imagenet')

         x=base_model.output
         x=GlobalAveragePooling2D()(x)
         x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and class
         x=Dense(1024,activation='relu')(x) #dense layer 2
         x=Dense(512,activation='relu')(x) #dense layer 3
         preds=Dense(5,activation='softmax')(x) #final layer with softmax activation
```

Figure 24

XGBoost:

```
In [15]: classifier = XGBClassifier(probability=True)
         classifier.fit(x_train, y_train)

Out[15]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                       colsample_bynode=1, colsample_bytree=1, gamma=0,
                       learning_rate=0.1, max_delta_step=0, max_depth=3,
                       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                       nthread=None, objective='multi:softprob', probability=True,
                       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                       seed=None, silent=None, subsample=1, verbosity=1)
```

Figure 25

All the scripts of the various models implemented in this project are uploaded as part of ICT Solution.