National College of Ireland

An Implementation of Deep Learning Techniques to Detect Tomato Leaf Diseases – Configuration Manual

MSc Research Project

MSc in Data Analytics

Manikanta Dinesh Gudivada

Student ID: x18191851

School of Computing

National College of Ireland

Supervisor: Vladimir Milosavljevic

Student Name:	Gudivada Manikanta Dinesh
Student ID:	x18191851
Programme:	MSc Data Analytics
Year:	2019 - 2020
Module:	Research Project
Supervisor:	Vladimir Milosavljevic
Submission Due Date:	17th August 2020
Project Title:	An Implementation of Deep Learning Techniques to Detect Tomato Leaf Diseases – Configuration Manual
Word Count:	3265
Page Count:	25

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.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	G.M. DINESH
Date:	17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	Q
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	Q
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.	Q

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):	

1. Introduction

What exactly this report is for and why it is required? This is the first thing I want to explain this is the overall setup report where all of my coding stuff, configurations of the systems, code implementation using different languages were explained in detail and as well whole libraries and major code part is explained to understand how exactly I approached my problem.

This document gives an overall picture of the codes how I have implemented the models. The major information relates to the graphs and outputs and how I have predicted the diseases. The main important this is that no information which is presented here is not explained in report work. In simple words, any individual who does not have an idea of this research domain will get full clarity of what I have implemented in my thesis by looking into this manual report.

In brief, my project is all about farming where many farmers were committing suicides because of huge loss in their crops due to plant diseases. So, I have considered detecting the Tomato Plant Diseases. So, I have considered 17000 images with 9 diseased leaf classes and 1 healthy class. I have detected images using Deep Learning and Transfer Learning methods which are very much useful in predicting image data. I have used 4 different models (Dense Net, Le Net, Mobile Net and CNN).

2. Implemented specifications in experimenting the Predictions and Results

2.1. Hardware requirements

In this section I have given the details of what hardware configurations is required in implementing the code.

- Laptop Model: HP Performs very good for executing the code with its memory storage.
- Operating System used: Windows 10 Operating System is Convenient with 64 bites.
- Internal Processor: INTEL Core i5 processor helped me in running the code very fast with 8th Generation.
- Memory requirement: Minimum 8 GB ram is compulsory, and I have used 8 GB.

2.2. Software requirements

This section projects the software requirements which to be installed in implementing the written code. I have used python language to implement all the codes using multiple predefined libraries. To run the python code, I have used Anaconda Navigator's Jupyter Notebook and to be faster and more accurate I have used Google Collab where the GPU is very fast all the codes run in the online cloud.

- Python: Version 3.7.3 is the latest version used for this project with all inbuilt libraries.
- Jupyter Notebook: 6.1.2 – This is where we run the code by creating multiple notebooks.
- Anaconda Navigator: 1.9.7 – This is a python platform where we have to open Python.

```
isort                    4.3.21                      py37_0
itsdangerous             1.1.0                       py37_0
jdcal                    1.4.1                         py_0
jedi                     0.13.3                      py37_0
jinja2                   2.10.1                      py37_0
joblib                   0.13.2                      py37_0
jpeg                     9d                       he774522_0      conda-forge
json5                    0.8.4                         py_0
jsonschema               3.0.1                       py37_0
jupyter                  1.0.0                       py37_7
jupyter_client           5.3.1                         py_0
jupyter_console          6.0.0                       py37_0
jupyter_core             4.5.0                         py_0
jupyterlab               1.0.2                    py37hf63ae98_0
jupyterlab_server        1.0.0                         py_0
keras                    2.3.1                            0      anaconda
keras-applications       1.0.8                         py_0      anaconda
keras-base               2.3.1                       py37_0      anaconda
keras-preprocessing      1.1.0                         py_1      anaconda
keyring                  18.0.0                      py37_0
kiwisolver               1.1.0                    py37ha925a31_0
krb5                     1.16.1                    hc04afaa_7
lazy-object-proxy        1.4.1                    py37he774522_0
libarchive               3.3.3                      h0643e63_5
libblas                  3.8.0                       14_mkl      conda-forge
libcblas                 3.8.0                       14_mkl      conda-forge
libclang                 9.0.1              default_hf44288c_0      conda-forge
libcurl                  7.65.2                    h2a8f88b_0
libiconv                 1.15                       h1df5818_7
liblapack                3.8.0                       14_mkl      conda-forge
liblapacke               3.8.0                       14_mkl      conda-forge
liblief                  0.9.0                      ha925a31_2
libmklml                 2019.0.5                         0      anaconda
libopencv                4.3.0                       py37_1      conda-forge
```

Figure 1

```
requests                 2.22.0                      py37_0
retrying                 1.3.3                       py37_2
rope                     0.14.0                        py_0
rsa                      4.6                         pypi_0      pypi
ruamel_yaml              0.15.46                  py37hfa6e2cd_0
scikit-image             0.15.0                   py37ha925a31_0
scikit-learn             0.21.2                   py37h6288b17_0
scipy                    1.2.1                    py37h29ff71c_0
seaborn                  0.9.0                       py37_0
send2trash               1.5.0                       py37_0
setuptools               41.0.1                      py37_0
simplegeneric            0.8.1                       py37_2
simplejson               3.17.0                   py37he774522_0      anaconda
```

Figure 2: The above figures 1 & 2 are the list of the libraries used in python.

The above-mentioned figures give information about my libraries used in the program. Mainly I have used Keras, Tensorflow, Matplotlib, Seaborn, Pandas and Numpy etc were the major libraries used for the python. The libraries are mainly the predefined codes which are already the code is been set as an

inbuild in the libraries whereby importing those as packages we can directly use the features of the libraries such as plotting graphs figures and calculations everything can be done easily.
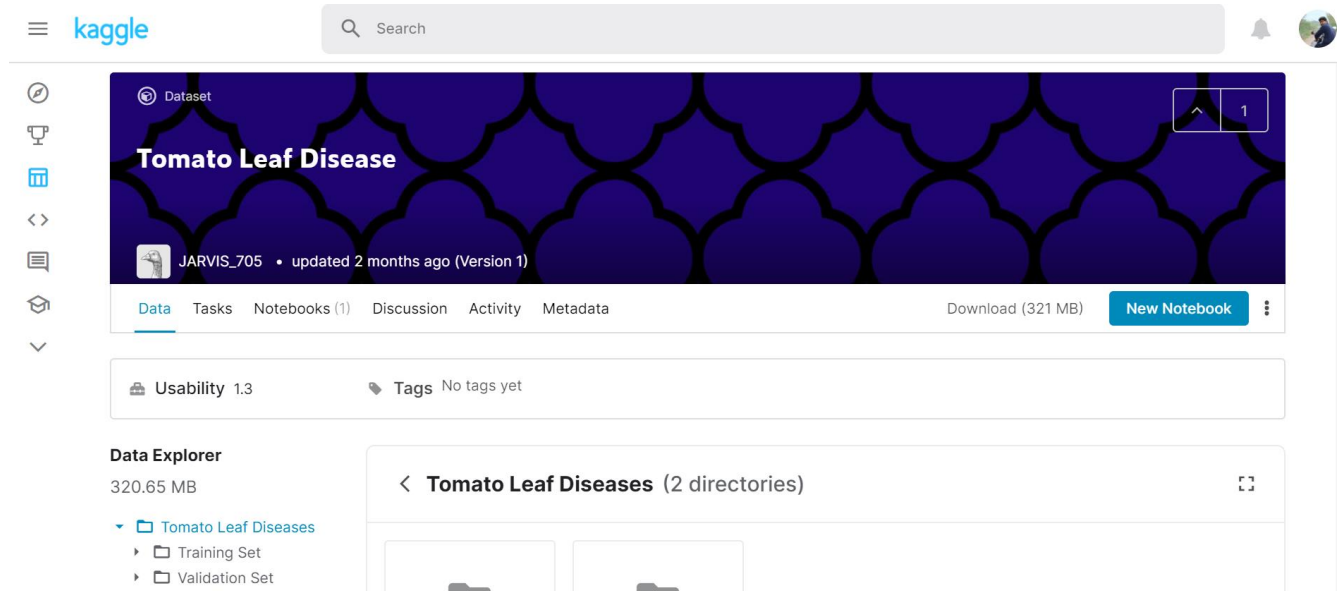
2.3. Online Source for Data Collection

Figure 3: Kaggle Repository

The **Figure 3** gives the insight of the dataset where I have collected the dataset from. In detail about the dataset the dataset consists of 17000 Tomato Leaf Diseased Leaf images which are divided into validation and Training sets. In each set there are 9 different diseased folders and 1 healthy leaf folder. All the folders contains a balanced number of images with 1000 images for each folder in training set and 700 images each in validation set. This is all about the data information I have gathered for my research which is available in open source platform and no measures to be taken for security purpose.

3. Implementation & Results for Deep Learning Models

3.1. Pre-Processing Steps

This section gives the detailed information of my approach in finding predictions of Tomato Leaf Disease using python language in Google Collaboratory platform. Before performing all these models firstly, we must import the data to the google drive using google. Once after storing all this data in the google, we can directly view the data.
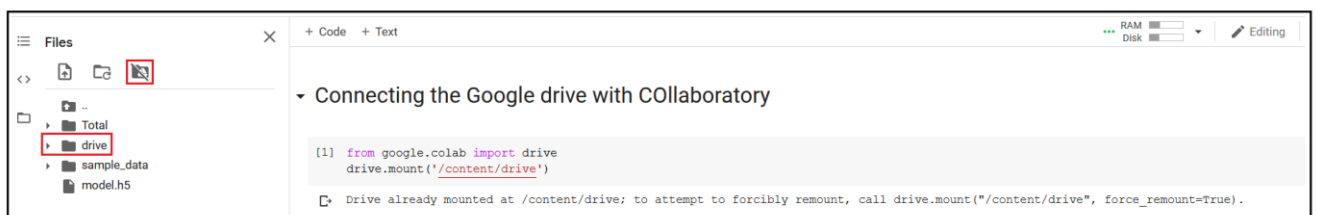
Figure 4: Connecting to Google Drive

3

As we already stored all the data into Google Drive now, we are mounting the Google Drive to the collab platform as we can see that the drive was connected to the platform.
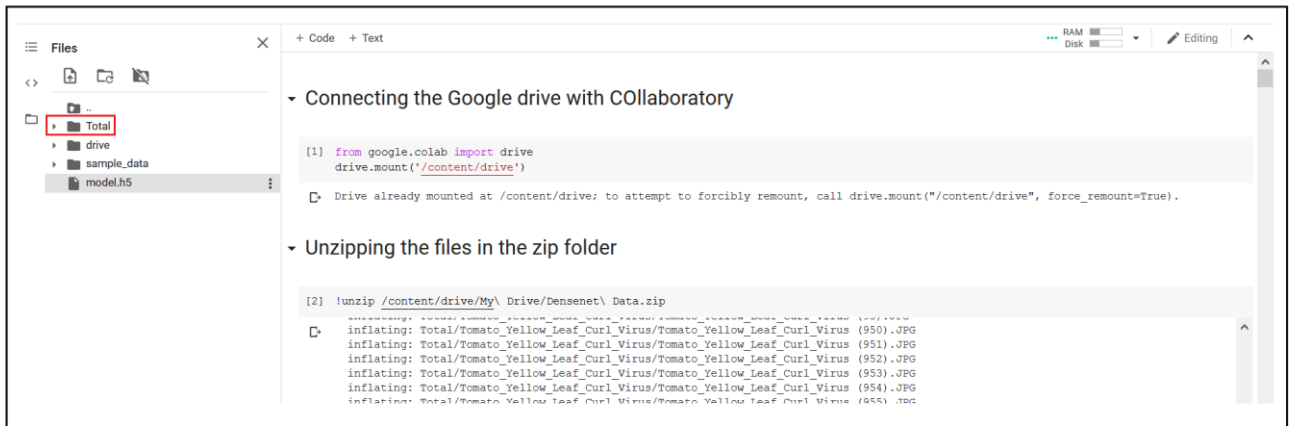
Figure 5: Extracting the files from the zip folder

Figure 5 gives an image of how I have unzipped the files from the folder and stored it in the system.

1. These are the basic two steps which are same for all the 4 models.
2. Once after mounting and unzipping the files I have loaded the data.

Figure 6: Loading Data to the model.

In the above **Figure 6** we can observe the data path is stored in a variable and all the folders stored in total have been extracted as is mentioned there are total of 10 classes.

Figure 7: Packages imported for the model evaluation

Figure 7 provides information about the imported packages from Keras and different other libraries used for the model.

3.2.Performance of Exploratory Data Analysis

Figure 8: EDA Analysis

In **Figure 8** we can observe that the data analysis is been evaluated where all the folders have stored in the diseases_types and the other code is used for just knowing the system is understanding whether the folders were containing images and reading those images or not. Where this is a single image from the dataset.

```python
#Canny edge detection by resizing

default_image_size = tuple((128,128))

def img_to_np(DIR,flatten=True):
    cv_img=mpimg.imread(DIR,0)
    cv_img=cv2.resize(cv_img,default_image_size)
    img = np.uint8(cv_img)
    if(flatten):
        img=img.flatten()
        return img
```

```python
#Appending all the folders and image files in the dataset for providing the Graphical Representation of Tomato Diseases

TRAIN_DIR="/content/Total/"

index=0
data={}
for FOLDER in os.listdir(TRAIN_DIR):
    for image_dir in os.listdir(TRAIN_DIR+FOLDER):
        if index not in data:
            data[index]=[]
        try:
            data[index].append(img_to_np(TRAIN_DIR+FOLDER+"/"+image_dir))
        except:
            print("Error to load the image "+TRAIN_DIR+FOLDER+"/"+image_dir)
    index=index+1
```

Figure 9: List Directory

Figure 9 gives the information of loading all the images to make the system understand and read all the images present in the dataset around 17000 should be created into a listing directory and then we have to convert those images into Num-py arrays. From the second block of the same diagram, we can make sure that the images were getting stored in the Folder and using for a loop all the images were being read by the image directory. Finally, they have appended all the data into the data variable by adding the folder and the images present in the folder.

```python
#Declaring the limit size for the graphs

CLASS_LIMIT=500
colors=["darkorange","crimson","dimgray","royalblue","darkgreen","gold","coral","darkviolet","hotpink","midnightblue"]
for index_class in range(len(data)):
    index=0
    for arr in data[index_class]:
        plt.hist(arr,color=colors[index_class],alpha=0.5)
        if(index>CLASS_LIMIT):
            plt.title(disease_types[index_class])
            plt.show()
            break
        index=index+1
```

Figure 10: Printing Folder Data

Figure 10 is all about drawing the histogram for all the folder data which is around 1700 in all the 10 folders and visualizing the data in the form of histogram graphs which are present in the below diagrams.
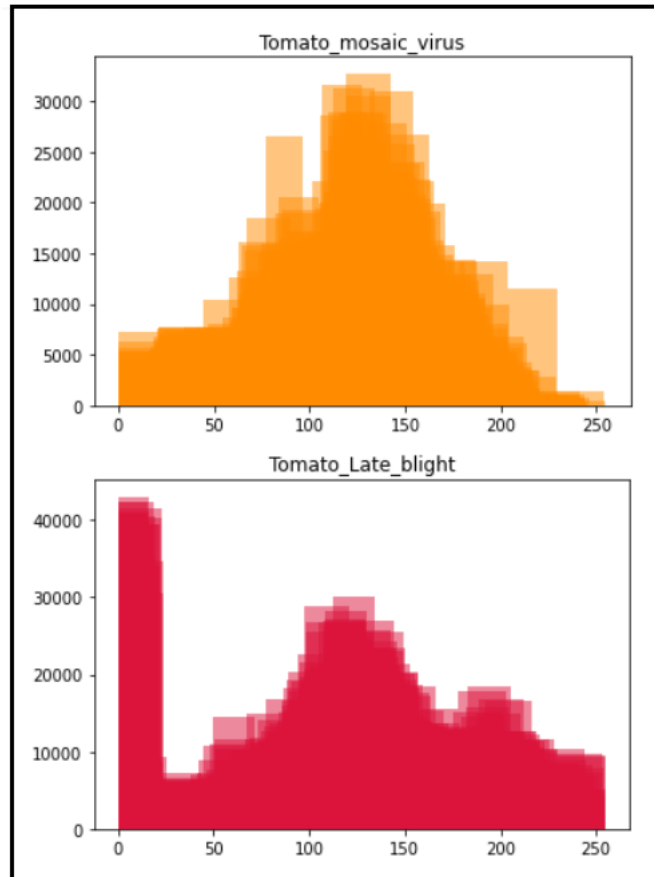
```
[ ]  # Information for the Dataset Folders
     labels = os.listdir(DATASET_DIR)
     print("Number of Labels:", len(labels))

     total = 0
     for lb in os.scandir(DATASET_DIR):
         print('folder: {} images: {}'.format(lb.name, len(os.listdir(lb))))
         total += len(os.listdir(lb))
     print('Total images:', total)

 ⊏→  Number of Labels: 10
     folder: Tomato_Yellow_Leaf_Curl_Virus images: 1700
     folder: Tomato_Late_blight images: 1700
     folder: Tomato_healthy images: 1700
     folder: Tomato_mosaic_virus images: 1700
     folder: Tomato_Target_Spot images: 1700
     folder: Tomato_Early_blight images: 1700
     folder: Tomato_Spider_mites images: 1700
     folder: Tomato_Bacterial_spot images: 1700
     folder: Tomato_Leaf_Mold images: 1700
     folder: Tomato_Septoria_leaf_spot images: 1700
     Total images: 17000
```

Figure 11: Data Folder Information.

This is to make a system to understand the folders and how many images were present in each folder. Scan Directory is a function using scan directory I am storing the data into total which is initialized as 0.

7

TRAIN.TAIL()

```
[ ]  #Printing the tail values which are stored by the system from the dataset
     train_data = []
     for defects_id, sp in enumerate(disease_types):
         for file in os.listdir(os.path.join(DATASET_DIR, sp)):
             train_data.append(['{}/{}'.format(sp, file), defects_id, sp])

     train = pd.DataFrame(train_data, columns=['File', 'DiseaseID','Disease Type'])
     train.tail()
```

	File	DiseaseID	Disease Type
16995	Tomato_Early_blight/Tomato_Early_blight (421).JPG	9	Tomato_Early_blight
16996	Tomato_Early_blight/Tomato_Early_blight (520).JPG	9	Tomato_Early_blight
16997	Tomato_Early_blight/Tomato_Early_blight (454).JPG	9	Tomato_Early_blight
16998	Tomato_Early_blight/Tomato_Early_blight (508).JPG	9	Tomato_Early_blight
16999	Tomato_Early_blight/Tomato_Early_blight (912).JPG	9	Tomato_Early_blight

Figure 12: Tail Values

The above code train.tail() is used to print the last 5 values in the data.

Train.columns

train.info

train.dtypes

train.count()

train.describe()

train.head()

train.hist()

Referring to the **Figure 12** as it is mentioned that the tail values are given in the same way to find all the different formats of the data, we have used the above formulae for the different kinds of evaluations.

As here the above-mentioned implementation techniques such as Mounting, Unzipping, Loading and EDA analysis are the same steps done in all the models and used as pre-processing steps for training the data.

3.3. Model Performance

3.3.1. Dense Net Model

```
[ ]  #Reshaping the images for the better model performance
     IMAGE_SIZE = 64
     def read_image(filepath):
         return cv2.imread(os.path.join(TRAIN_DIR, filepath)) # Loading a colored image as a default flag

     # Resizing the image to the target size which is given as 64
     def resize_image(image, image_size):
         return cv2.resize(image.copy(), image_size, interpolation=cv2.INTER_AREA)


[ ]  #To find the shape of the Train Folder in the Dataset
     X_train = np.zeros((train.shape[0], IMAGE_SIZE, IMAGE_SIZE, 3))
     for i, file in tqdm(enumerate(train['File'].values)):
         image = read_image(file)
         if image is not None:
             X_train[i] = resize_image(image, (IMAGE_SIZE, IMAGE_SIZE))

     # Normalization of the train data
     X_Train = X_train / 255.
     print('Train Shape: {}'.format(X_Train.shape))

 ⊡   17000it [00:20, 847.58it/s]
     Train Shape: (17000, 64, 64, 3)
```

Figure 13: Data Storage for Training the Model

From the **Figure 13,** I have declared the Image size to 64 and copied the resized images and then the images were stored into X_Train and it is divided by the dimension to set all the images in one format. At the end all the 17000 images were trained and stored to the model with input shape. Later I have divided this data by created 10 class labels.

```
[ ]  BATCH_SIZE = 128

     # Split the train and validation sets
     X_train, X_val, Y_train, Y_val = train_test_split(X_Train, Y_train, test_size=0.2, random_state=SEED)
```

Figure 14: Data Division

Based on Figure 14 we can understand that the data is divided for training and testing purpose. Where the division is of 0.2 were 80% for training and 20% for testing.

9

```
# Performing Model Using 50 Epochs
EPOCHS = 50
SIZE=64
N_ch=3

def build_densenet():
    densenet = DenseNet121(weights='imagenet', include_top=False)
    input = Input(shape=(SIZE, SIZE, N_ch))
    x = Conv2D(3, (3, 3), padding='same')(input)
    #x = Maxpool2D(3, strides = 2, padding='same')(x)
    x = densenet(x)
    x = GlobalAveragePooling2D()(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

    # multi output
    output = Dense(10, activation = 'softmax', name='root')(x)

    # model
    model = Model(input,output)

    optimizer = Adam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=0.1, decay=0.0)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    model.summary()

    return model
```

Figure 15: Dense Net – 121

In **Figure 15** the Dense Net 121 model is defined for the Tomato Disease Prediction here in the above figure we have created a dense net model using image net dataset using transfer learning and all the required pooling layers and activations functions were used for the model. Once after defining the model I have compiled the model before fitting to the model.

```
#Printing the model for identifying the DenseNet Parameters
model = build_densenet()
annealer = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=5, verbose=1, min_lr=1e-3)
checkpoint = ModelCheckpoint('model.h5', verbose=1, save_best_only=True)
```

```
Model: "functional_5"

Layer (type)                    Output Shape             Param #
=================================================================
input_6 (InputLayer)            [(None, 64, 64, 3)]      0

conv2d_2 (Conv2D)               (None, 64, 64, 3)        84

densenet121 (Functional)        (None, None, None, 1024) 7037504

global_average_pooling2d_2 (    (None, 1024)             0

batch_normalization_4 (Batch    (None, 1024)             4096

dropout_4 (Dropout)             (None, 1024)             0

dense_2 (Dense)                 (None, 256)              262400

batch_normalization_5 (Batch    (None, 256)              1024

dropout_5 (Dropout)             (None, 256)              0

root (Dense)                    (None, 10)               2570
=================================================================
Total params: 7,307,678
Trainable params: 7,221,470
Non-trainable params: 86,208
_____
```

Figure 16: Model Summary

In the above code it is giving the summary of all the parameters and the model parameters.

```
[ ]  from skimage import io
     from keras.preprocessing import image

     disease_types = ['Tomato_mosaic_virus',
                      'Tomato_Late_blight',
                      'Tomato_Septoria_leaf_spot',
                      'Tomato_Bacterial_spot',
                      'Tomato_Spider_mites',
                      'Tomato_Yellow_Leaf_Curl_Virus',
                      'Tomato_Leaf_Mold',
                      'Tomato_Target_Spot',
                      'Tomato_healthy',
                      'Tomato_Early_blight']

     img = image.load_img('/content/Total/Tomato_Late_blight/Tomato_Late_blight (1086).JPG')
     show_img=image.load_img('/content/Total/Tomato_Late_blight/Tomato_Late_blight (1086).JPG', grayscale=False, target_size=(224, 224))

     x = image.img_to_array(img)
     x = np.expand_dims(x, axis = 0)
     #x = np.array(x, 'float32')
     x /= 255

     disease = model.predict(x)
     print(disease[0])

     plt.imshow(show_img)
     plt.show()

     a=disease[0]
     ind=np.argmax(a)

     print('Prediction:',disease_types[ind])
```
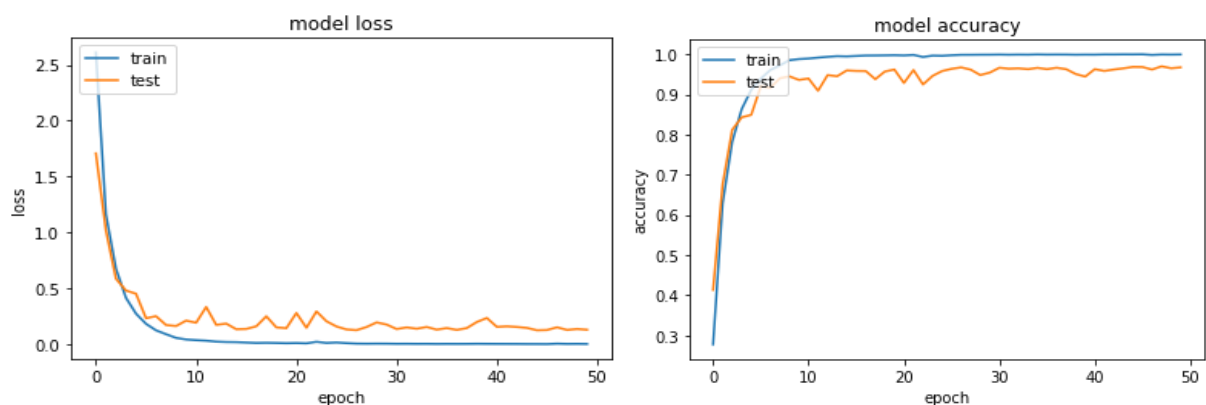
Figure 17: Prediction Module

The above model has predicted the Tomato Disease. I have passed the image of the test set and evaluated using the model which is already compiled and fitted by Dense Net and using the algorithm the system predicted the correct prediction of disease.

- **Evaluation Metrics Results**

Figures 18 & 19: Graphs for Model Loss and Accuracy

These graphs give us the information about how accurate the model evaluated the images internally by each epoch up to 50 epochs. The model loss is decreased to 0.2 approximately for both train and test consistently and accuracy have increased to 0.9 approx. where there is no overfitting of the data and the model performance is very good and predicted the correct outputs.

```
from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics

print ("Accuracy : ", metrics.accuracy_score(Y_true, Y_pred)*100)
```
Accuracy : 96.73529411764706

```
# Precision
from sklearn.metrics import precision_score

precision_score(Y_true, Y_pred, average=None)
```
array([1.   , 0.95689655, 0.95014663, 0.9622093 , 0.95364238,
       0.9719888 , 0.97337278, 0.95481928, 0.99132948, 0.95844875])

```
# Recall

from sklearn.metrics import recall_score
recall_score(Y_true, Y_pred, average=None)
```
array([0.99698795, 0.93802817, 0.96716418, 0.97067449, 0.9632107 ,
       0.99142857, 0.97916667, 0.95195195, 0.98847262, 0.93010753])

```
# F1_score
from sklearn.metrics import f1_score

f1_score(Y_true, Y_pred, average=None)
```
array([0.9984917 , 0.94736842, 0.95857988, 0.96642336, 0.95840266,
       0.98161245, 0.97626113, 0.95338346, 0.98989899, 0.94406548])

```
from sklearn.metrics import r2_score

r2_score(Y_true, Y_pred)
```
0.9130904054952927

Figure 20: Results

The above figure provides us the Evaluation metrics results for the model. By considering this result we can assure that the Dense Net model has performed very good with 96% Accuracy.

```
#MAE L1 loss function - Should be close to 0

from sklearn.metrics import mean_absolute_error
mean_absolute_error(Y_true, Y_pred)
```
0.1361764705882353

```
#MSE L2 loss function - Should be close to 0

from sklearn.metrics import mean_squared_error
mean_squared_error(Y_true, Y_pred)
```
0.7338235294117647

```
from sklearn.metrics import classification_report

print(classification_report(Y_true, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       332
           1       0.96      0.94      0.95       355
           2       0.95      0.97      0.96       335
           3       0.96      0.97      0.97       341
           4       0.95      0.96      0.96       299
           5       0.97      0.99      0.98       350
           6       0.97      0.98      0.98       336
           7       0.95      0.95      0.95       333
           8       0.99      0.99      0.99       347
           9       0.96      0.93      0.94       372

    accuracy                           0.97      3400
   macro avg       0.97      0.97      0.97      3400
weighted avg       0.97      0.97      0.97      3400
```

Figure 21: Error Results

These above results give us the information for all the evaluation techniques in the classification report. As well the errors for the model were perfectly good with good values where the error values should be equal to zero **(Huang *et al.*, 2017)**.

3.3.2. Le Net Model:

```
[ ]  #Initializing the values for the model.

     EPOCHS = 50
     SIZE = 60
     N_ch = 3


#Creating the Le_net Model function

def LeNet_Model():
    model = Sequential()# creating a sequential model

    model.add(Convolution2D(20, 5, 5, padding="same",input_shape=(60, 60,3)))
    model.add(Dropout(0.2))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    # second layer is setting to CONV - RELU - POOL
    model.add(Convolution2D(50, 5, 5, padding="same"))
    model.add(Dropout(0.2))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=(2, 2))) # specifying the dimensions in particular order.

    # Adding the set of FC - RELU layers to the model Le Net
    model.add(Flatten())
    model.add(Dense(500))
    model.add(Dropout(0.2))
    model.add(Activation("relu"))

    # Activating the model with Softmax Classifier
    model.add(Dense(10)) # 10 categories is 10 classes
    model.add(Dropout(0.2))
    model.add(Activation("softmax"))

    return model
```

Figure 22: Le Net Model

Figure 22 provides the of the parameters, layers, functions used for the Le Net model. Here in the model, we have used two convolutional layers and then we have used different activation functions to activate the model.

```
[ ]  # Compiling the model before fitting with cost and optimization method

     model.compile(loss='categorical_crossentropy',
             optimizer="Adam",
             metrics=['accuracy'])
```

Figure 23: Model Compilation

This step is common for any Deep Learning model where before fitting the model we must compile the model and then we must train the loss functions involved in the model and the optimizer improves the model performance.

```
# Fits the model on batches with real-time data augmentation
# It took around 20 Min to complete all the 100 Epochs

hist = model.fit(datagen.flow(X_train, Y_train, batch_size=BATCH_SIZE),# Here we are fitting the model with divided data
             steps_per_epoch=X_train.shape[0] // BATCH_SIZE,
             epochs=EPOCHS,
             verbose=2,
             callbacks=[annealer, checkpoint],
             validation_data=(X_val, Y_val))
```

Figure 24: Fitting the model.

The above figure is mentioning about the model fitting where the training and testing data is giving to the model with an epoch of 50 and by-passing batch size of 128.

```
# In this step we are prediciting the model by passing the saving model which we already fitted with Le Net

from skimage import io
from keras.models import load_model
from keras.preprocessing import image
from keras.applications.mobilenet import preprocess_input

model = load_model('Lenet.h5')

disease_types = ['Tomato_mosaic_virus',
                 'Tomato_Late_blight',
                 'Tomato_Septoria_leaf_spot',
                 'Tomato_Bacterial_spot',
                 'Tomato_Spider_mites',
                 'Tomato_Yellow_Leaf_Curl_Virus',
                 'Tomato_Leaf_Mold',
                 'Tomato_Target_Spot',
                 'Tomato_healthy',
                 'Tomato_Early_blight']

show_img=image.load_img('/content/Total/Tomato_Early_blight/Tomato_Late_blight (1004).JPG', grayscale=False, target_size=(60, 60))
img = image.load_img('/content/Total/Tomato_Early_blight/Tomato_Late_blight (1004).JPG', target_size=(60, 60))

x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
img_data = preprocess_input(x)

classes = model.predict(img_data)

a=classes[0]
ind=np.argmax(a)
print(ind)
print('Prediction:',disease_types[ind])

plt.imshow(show_img)
plt.show()
```
```
1
Prediction: Tomato_Late_blight
```

Figure 25: Model Prediction for Le Net

In this step the model is trained and fitted by compiling and later step is to predict the disease. Here we are passing the test image for the model and loading the model to predict the outcome by using model.predict we can find the output as Tomato Late Blight.

```
[ ]  from keras.utils import plot_model

     plot_model(model, to_file='model1.png', show_shapes=True, rankdir='TB', expand_nested=True)
```

Figure 26: Model Plotting

Here we are just plotting the model which is evaluated for understanding all the parameters used for the model in a detailed way.

▪ **Evaluation Results**


```
[ ] from sklearn.metrics import accuracy_score
    import sklearn.metrics as metrics

    print ("Accuracy : ", metrics.accuracy_score(Y_true, Y_pred)*100)

    Accuracy :  86.94117647058823

[ ] # Precision
    from sklearn.metrics import precision_score

    precision_score(Y_true, Y_pred, average=None)

    array([0.96904025, 0.95065789, 0.99583333, 0.88825215, 0.74193548,
           0.99013158, 0.92429022, 0.78289474, 0.79587156, 0.77605322])

[ ] # Recall

    from sklearn.metrics import recall_score
    recall_score(Y_true, Y_pred, average=None)

    array([0.94277108, 0.81408451, 0.71343284, 0.90909091, 0.92307692,
           0.86      , 0.87202381, 0.71471471, 1.        , 0.94086022])

[ ] # F1_score
    from sklearn.metrics import f1_score

    f1_score(Y_true, Y_pred, average=None)

    array([0.95572519, 0.87708649, 0.83130435, 0.89855072, 0.82265276,
           0.9204893 , 0.89739663, 0.74725275, 0.88633461, 0.85054678])

[ ] from sklearn.metrics import r2_score

    r2_score(Y_true, Y_pred)

    0.6833286077585997
```

Figure 27: Model Results

These results provide the good results with good accuracy of 86% and the precision for all the classes were very good and all the other techniques have also evaluated.


```
[ ] #MAE L1 loss function - Should be close to 0

    from sklearn.metrics import mean_absolute_error
    mean_absolute_error(Y_true, Y_pred)

    0.5032352941176471

[ ] #MAE L2 loss function - Should be close to 0

    from sklearn.metrics import mean_squared_error
    mean_squared_error(Y_true, Y_pred)

    2.6738235294117647

[ ] from sklearn.metrics import classification_report

    print(classification_report(Y_true, Y_pred))

                  precision    recall  f1-score   support

               0       0.97      0.94      0.96       332
               1       0.95      0.81      0.88       355
               2       1.00      0.71      0.83       335
               3       0.89      0.91      0.90       341
               4       0.74      0.92      0.82       299
               5       0.99      0.86      0.92       350
               6       0.92      0.87      0.90       336
               7       0.78      0.71      0.75       333
               8       0.80      1.00      0.89       347
               9       0.78      0.94      0.85       372

        accuracy                           0.87      3400
       macro avg       0.88      0.87      0.87      3400
    weighted avg       0.88      0.87      0.87      3400
```

Figure 28: Model Results 2 for Le Net

In the second results section, there are few more techniques where the absolute and mean errors were calculated and are equal to zero. The whole report for the metrics was also given in the figure above.
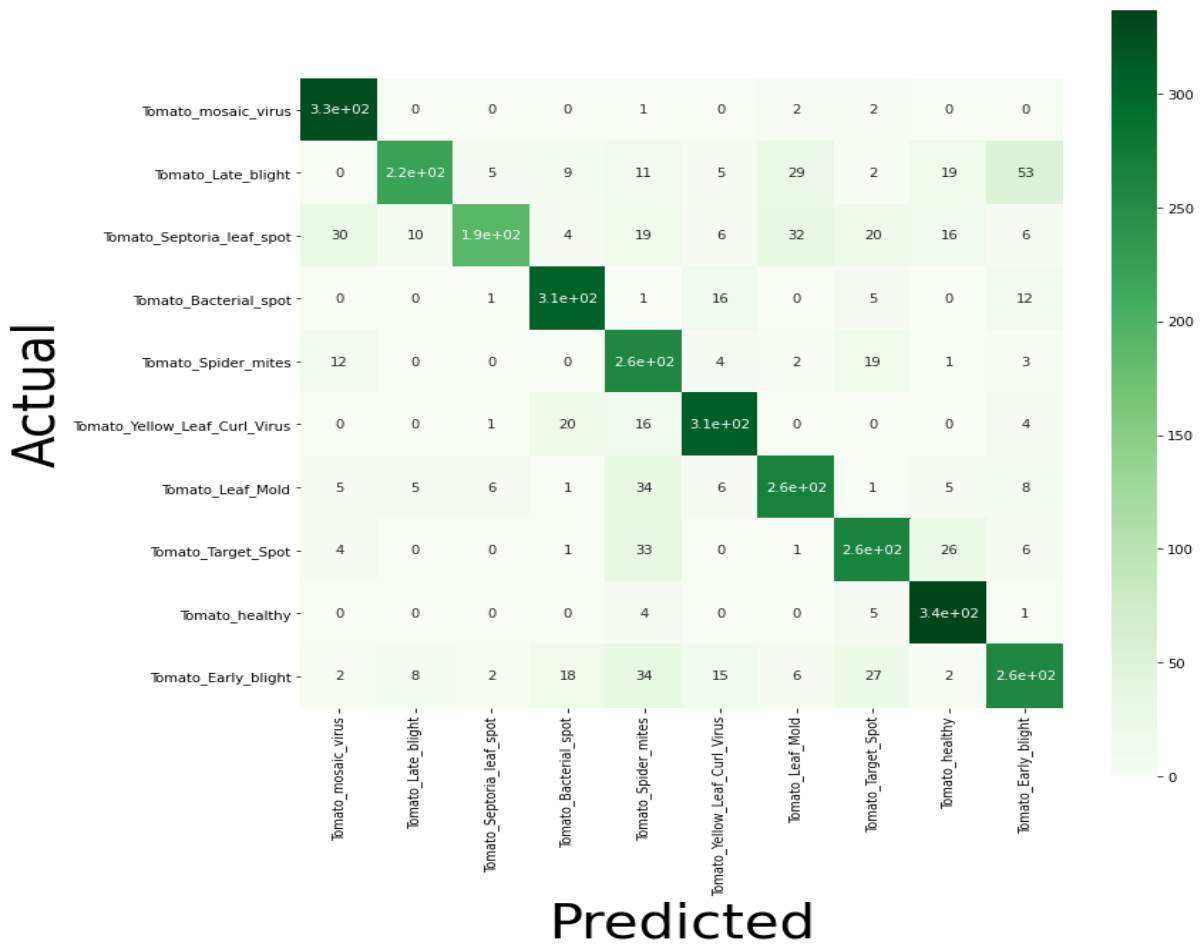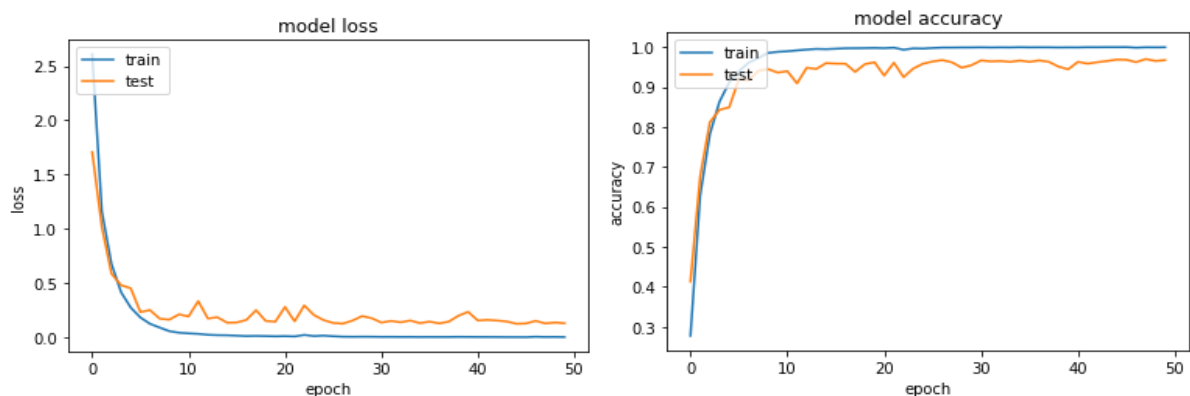
15

Figure 29: Confusion Matrix

This matrix is used for understanding the values which the model have predicted properly considering all the images training images were 1360 and remaining were 340 out of that almost all the classes have predicted the proper results with good accuracy.

These graphs were used to understand the values that happened between each epoch in a sequence manner. If we observe the train and test for the model loss was decreased gradually and increased accuracy and stood at 0.8 approximately which gives the good model performance.

3.3.3. CNN Outputs

```
[18] TRAIN_DIR='/content/Total/'

     #Reshaping the images for the better model performance
     IMAGE_SIZE = 64
     def read_image(filepath):
         return cv2.imread(os.path.join(TRAIN_DIR, filepath)) # Loading a colored image as a default flag

     # Resizing the image to the target size which is given as 64
     def resize_image(image, image_size):
         return cv2.resize(image.copy(), image_size, interpolation=cv2.INTER_AREA)

[19] #To find the shape of the Train Folder in the Dataset
     X_train = np.zeros((train.shape[0], IMAGE_SIZE, IMAGE_SIZE, 3))
     for i, file in tqdm(enumerate(train['File'].values)):
         image = read_image(file)
         if image is not None:
             X_train[i] = resize_image(image, (IMAGE_SIZE, IMAGE_SIZE))

     # Normalization of the train data
     X_Train = X_train / 255.
     print('Train Shape: {}'.format(X_Train.shape))

[→]  17000it [00:26, 647.03it/s]
     Train Shape: (17000, 64, 64, 3)

[26] Y_train = train['DiseaseID'].values
     print('Before shape value:',Y_train.shape)

     Y_train = to_categorical(Y_train, num_classes=10)
     print('shape value after adding the classes:',Y_train.shape)

[→]  Before shape value: (17000,)
     shape value after adding the classes: (17000, 10)
```

Figure 30: CNN Model Initialization

This model is one of the famous models which is used by many authors and this model helps in detecting the diseases. In the above figure same as other models I have read all the 17000 images and stored in the X_Train variable and with a shape 64 and depth 3. Later I have created 10 class variables for predicting the outputs.

```
[ ] EPOCHS = 50
    N_ch=3
    INIT_LR = 1e-3

    width=64
    height=64
    depth=3
```

Figure 31: Epochs Declaration

In the above figure, I have initialized the model with 50 epochs and height, width and depth are also pre-defined for the CNN model.

```
model = Sequential()
inputShape = (height, width, depth)
chanDim = -1

if K.image_data_format() == "channels_first":
    inputShape = (depth, height, width)
    chanDim = 1

model.add(Conv2D(64, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation("softmax"))
```

Figure 32: CNN Model Architecture

This is the major part of the code for the CNN where after diving the data and stored with all the dimensions I have created 5 convolutional layers for better results. If we increase the layers the level of understanding for the model will be increased. All the max-pooling and different functions have also used for the model evaluation.

```
[ ]  # Fits the model on batches with real-time data augmentation
     #It took around 20 Min to complete all the 50 Epochs

     hist = model.fit(X_train, Y_train, batch_size=BATCH_SIZE,
                      steps_per_epoch=X_train.shape[0] // BATCH_SIZE,
                      epochs=EPOCHS,
                      verbose=2,
                      callbacks=[annealer, checkpoint],
                      validation_data=(X_val, Y_val))
```

Figure 33: CNN Model Fitting

After creating all the layers and initializing the shapes for each layer we are fitting the model. While fitting the model the X and Y train values were given for the model and the X and Y validation were given for the results sections with a batch size of 128. Where at once the model will take 128 images instead of taking one on one image.

- **Evaluation Metrics Outputs for CNN**

```
[ ]  from sklearn.metrics import accuracy_score
     import sklearn.metrics as metrics

     print ("Accuracy : ", metrics.accuracy_score(Y_true, Y_pred)*100)

     Accuracy :  88.29411764705883

[ ]  # Precision
     from sklearn.metrics import precision_score

     precision_score(Y_true, Y_pred, average=None)

     array([0.95614035, 0.91554054, 0.99259259, 0.97222222, 0.82315113,
            0.97965116, 0.88121547, 0.93913043, 0.63837638, 0.91556728])

[ ]  # Recall

     from sklearn.metrics import recall_score
     recall_score(Y_true, Y_pred, average=None)

     array([0.98493976, 0.76338028, 0.8       , 0.92375367, 0.85618729,
            0.96285714, 0.94940476, 0.64864865, 0.99711816, 0.9327957 ])

[ ]  # F1_score
     from sklearn.metrics import f1_score

     f1_score(Y_true, Y_pred, average=None)

     array([0.97032641, 0.83256528, 0.88595041, 0.94736842, 0.83934426,
            0.97118156, 0.91404011, 0.76731794, 0.7784027 , 0.9241012 ])
```

Figure 34: CNN Results Section 1

In section 1 results of CNN, we have got Accuracy, Precision, Recall and F1 Score values where the precision and Recall values were calculated for all the classes with a good outcome.

```
    from sklearn.metrics import r2_score

    r2_score(Y_true, Y_pred)

    0.7225163006715438

[ ]  #MAE L1 loss function - Should be close to 0

     from sklearn.metrics import mean_absolute_error
     mean_absolute_error(Y_true, Y_pred)

     0.44882352941176473

[ ]  #MAE L2 loss function - Should be close to 0

     from sklearn.metrics import mean_squared_error
     mean_squared_error(Y_true, Y_pred)

     2.342941176470588

[ ]  from sklearn.metrics import classification_report

     print(classification_report(Y_true, Y_pred))

                  precision    recall  f1-score   support

               0       0.96      0.98      0.97       332
               1       0.92      0.76      0.83       355
               2       0.99      0.80      0.89       335
               3       0.97      0.92      0.95       341
               4       0.82      0.86      0.84       299
               5       0.98      0.96      0.97       350
               6       0.88      0.95      0.91       336
               7       0.94      0.65      0.77       333
               8       0.64      1.00      0.78       347
               9       0.92      0.93      0.92       372

        accuracy                           0.88      3400
```

Figure 35: CNN Section 2 Results

In section 2 of CNN evaluation results, we have calculated the MSE and MAE along with the R2 Score where the values should be equal to zero.
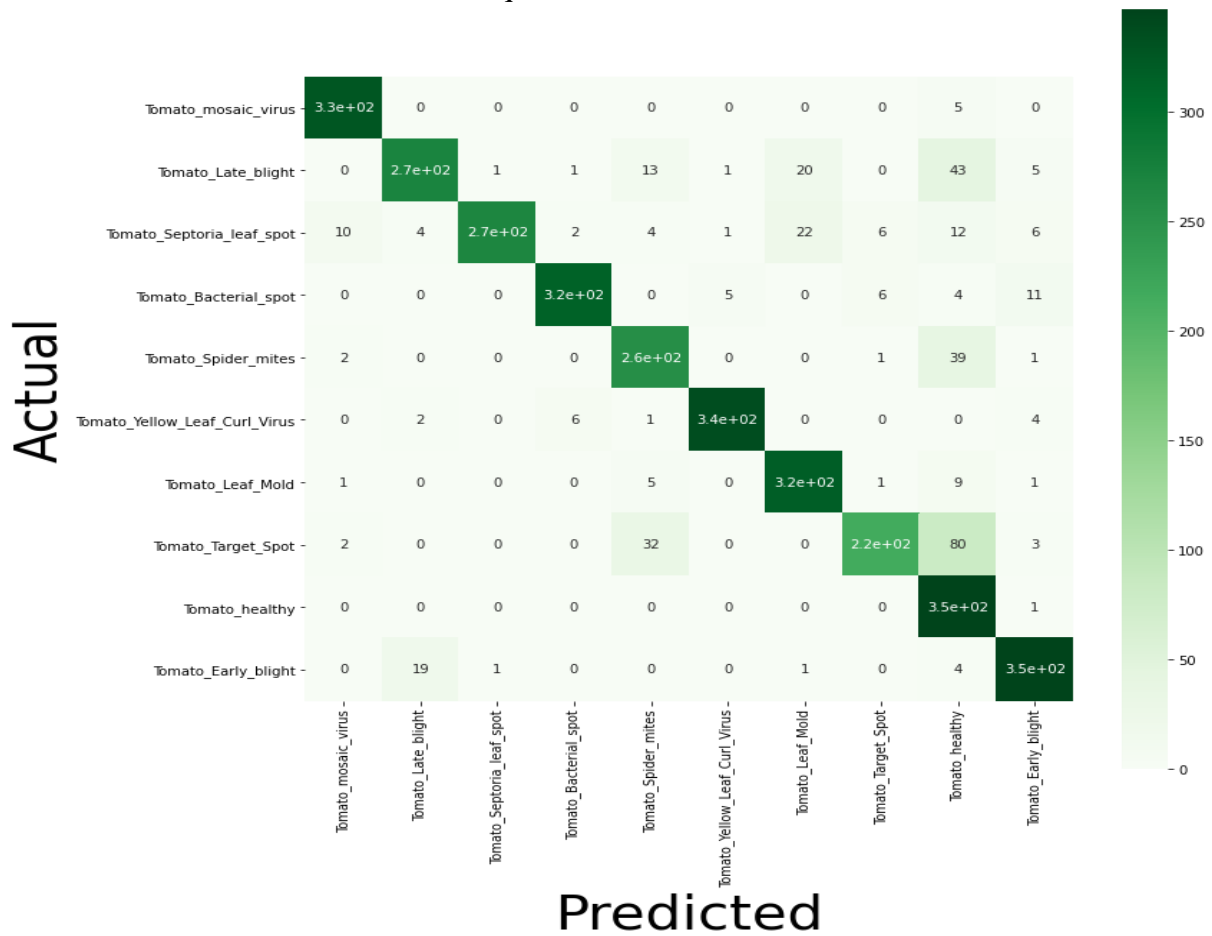

Figure 3: Confusion Matrix for CNN Model

If we observe the graph the actual values and predicted values were showing in the graphs the values which are predicted correctly has shown in dark green with a good number of predictions and the remaining 1 or 2 values were not predicted correctly by the model. On average, all the images in all the classes are showing good results.
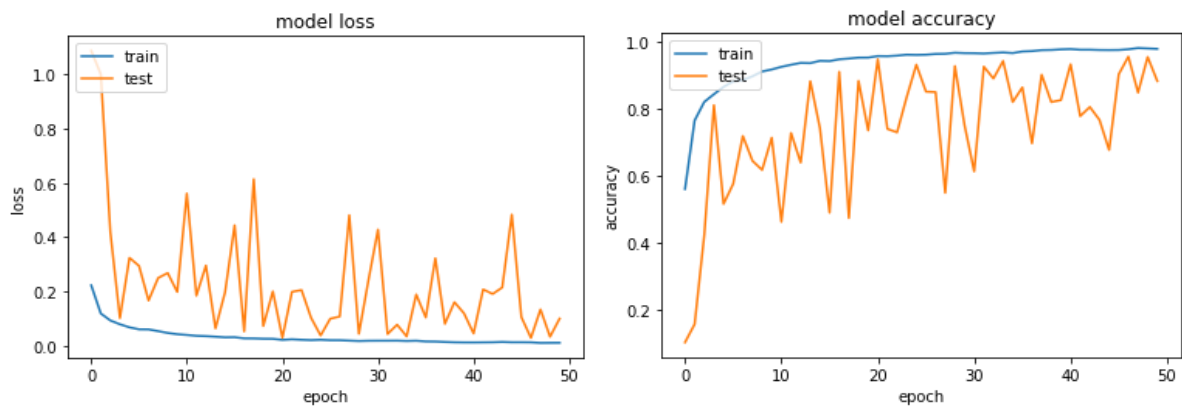
These graphs give us the loss and accuracy outcomes for both train and test models for 50 epochs. If we consider the loss values, it is decreased to 0 and accuracy increased to 1 which is a good prediction outcome for CNN model (**Khan** *et al.*, **2020**).

3.3.4. Mobile Net Outputs

The mobile net model is also trained based on the transfer learning where this model is trained in a different way where the images were already divided into two different folders for training and testing purpose and while training the model we have given the model train path and test path separately.

```
[ ] IMAGE_SIZE = [224, 224]

    train_path = '/content/Research Dataset/Train'
    valid_path = '/content/Research Dataset/Test'

    # adding the preprocessing layer to the very first layer of MobileNet Model
    den = MobileNet(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

    #Image net is the name of the compinetiong where all these models were saved
    # Don't train existing weights
    for layer in den.layers:
        layer.trainable = False

[→] Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf_no_top.h5
    17227776/17225924 [==============================] - 0s 0us/step

[ ] # Base layer of the Model
    x = Flatten()(den.output)
    prediction = Dense(len(Tomato_Leaf_Diseases_Train), activation='softmax',  kernel_regularizer=regularizers.l2(0.0001))(x)
    #Using L2 regularizer to avoid overfitting.

[ ] # creating a model object
    model = Model(inputs=den.input, outputs=prediction)

    # viewing the structure of the model
    model.summary()
```

Figure 37: Mobile Net Transfer Learning Technique

The above code gives us the correct information about how I have trained the model separately for both training and testing purpose. I have used the image net data and considered the mobile net to detect tomato diseases. Here I am training the mobile net model with an image size of 224. In the other steps once after collecting the data the model was flattened and created a model objective for the model. Finally checking the summary values of the model.

```
[ ] # Compiling the model with cost and optimization method
    model.compile(
      loss='categorical_crossentropy',
      optimizer='adam',
      metrics=['accuracy']
    )

[ ] #Dividing the data for training and testing purpose
    training_set = train_datagen.flow_from_directory('/content/Research Dataset/Train',
                                                      target_size = (224, 224),
                                                      batch_size = 32,
                                                      class_mode = 'categorical')

    test_set = test_datagen.flow_from_directory('/content/Research Dataset/Test',
                                                target_size = (224, 224),
                                                batch_size = 32,
                                                class_mode = 'categorical')

[→] Found 10000 images belonging to 10 classes.
    Found 7000 images belonging to 10 classes.

[ ] my_callbacks = [tf.keras.callbacks.EarlyStopping(patience=2)]

[ ] #Fitting the Model with 25 Epochs
    result = model.fit(training_set,
                       validation_data=test_set,
                       epochs=100,
                       steps_per_epoch=len(training_set),
                       validation_steps=len(test_set) ,callbacks=my_callbacks)
```
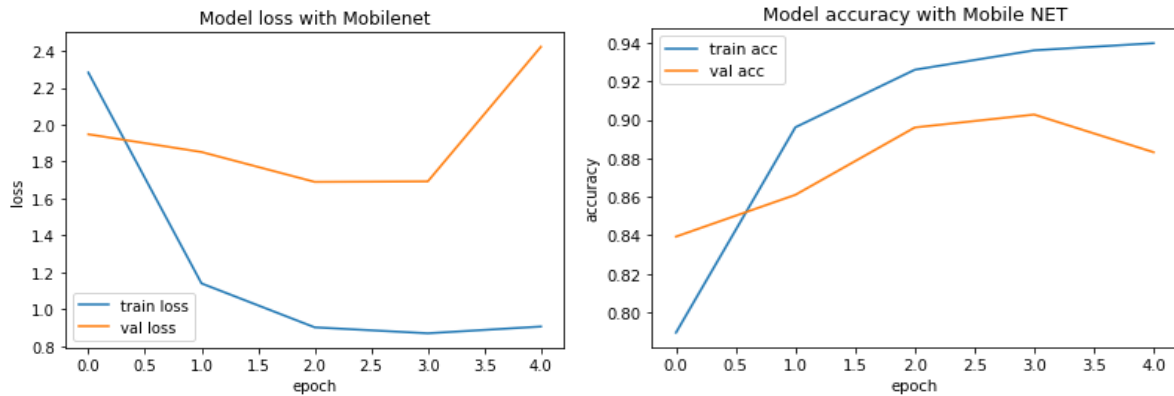
Figure 38: Model Fit for Mobile Net

After checking all the parameters of the mobile net, we have compiled the model and used callbacks function where this will help in overfitting the data. Finally, we ran the model by giving the training set images to the model and storing the test set image values in the validation results.

- **Evaluation Metrics Graphs**

The above graphs are showing up to only 5 epochs values because the model is not learning anything from there as all the images were been fitted into the model. There is no increase in the val_accuracy of the model **(Sinha and El-Sharkawy, 2019)**.
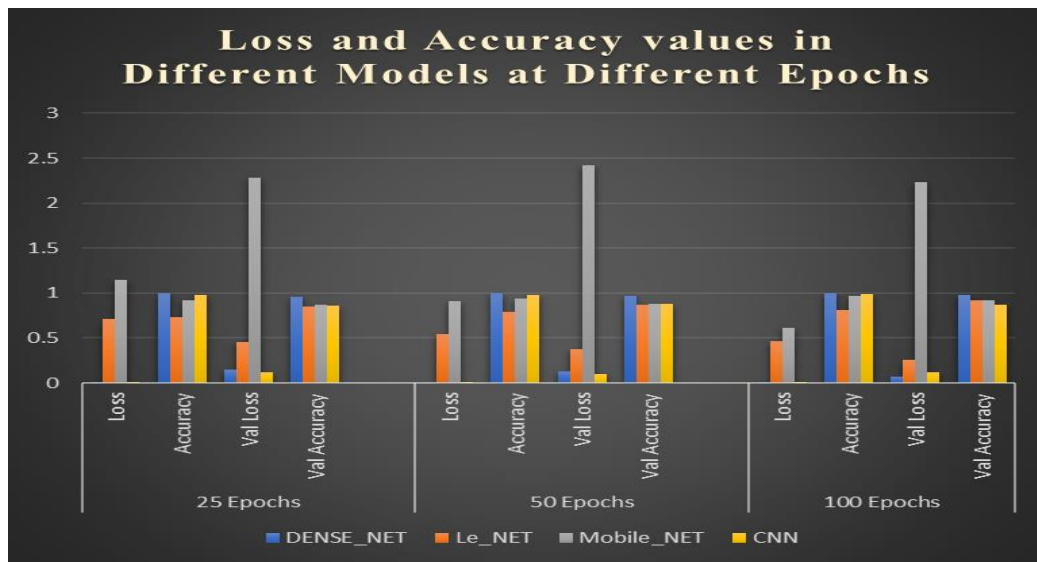
Comparing the Models

Figure 39: Over all Epochs Values

This graph gives detailed information on which model has how much accuracy and loss values at different epoch stages. If we observe in all the epochs (25, 50, 100) the Val Accuracy and Accuracy values are high for Dense Net. The Loss and Val Loss is very low for Dense Net. Which means out of all the models Dense Net is providing the better results **(Kumar and Vani, 2019)**.
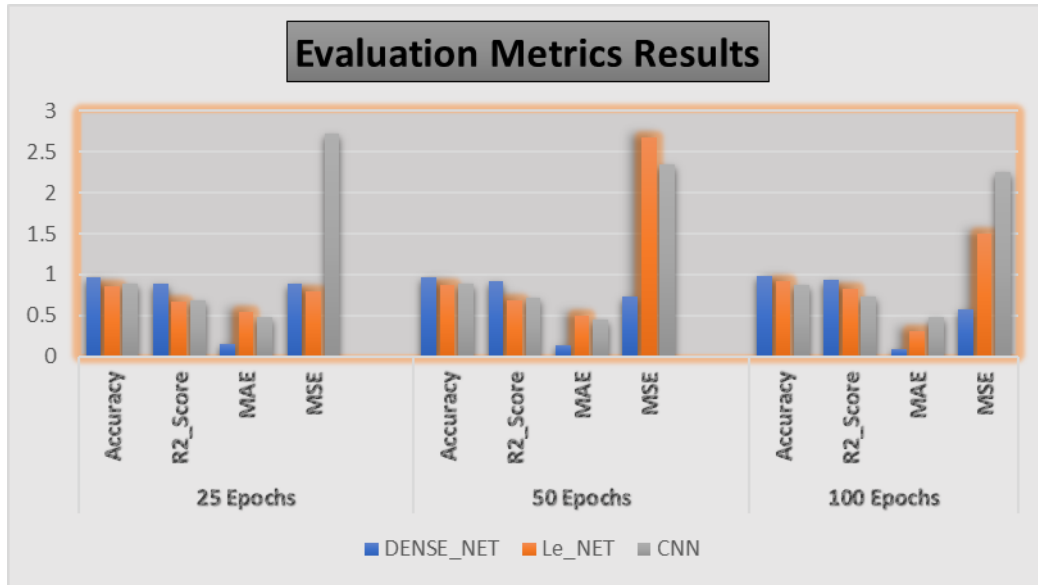
Figure 40: Over all Metrics Results

On overall, the Accuracy and R2-Score are high for Dense Net. Absolute and Square Error-values are low for Dense Net which is nearly equal to zero which means they are providing good results with no error values. Comparatively, Le Net and CNN both are almost equal for R2-Score and Error-values are differentiated.

4. Conclusion

To conclude the model this whole report is useful for individuals to understand clearly what I have done to achieve my output. So, this whole report is submitted with all the codes, graphs and components used to complete this project with a detailed explanation. All these codes have implemented in Google Collaboratory platform using GPU memory which will increase the speed of the model and helped me to get outputs in less than one hour.

5. References

Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K. Q. (2017) 'Densely connected convolutional networks', *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. IEEE, 2017-Janua, pp. 2261–2269. doi: 10.1109/CVPR.2017.243.

Khan, A., Sohail, A., Zahoora, U. and Qureshi, A. S. (2020) 'A survey of the recent architectures of deep convolutional neural networks', *Artificial Intelligence Review*, pp. 1–70. doi: 10.1007/s10462-020-09825-6.

Kumar, A. and Vani, M. (2019) 'Image Based Tomato Leaf Disease Detection', *2019 10th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2019*. IEEE, pp. 1–6. doi: 10.1109/ICCCNT45670.2019.8944692.

Sinha, D. and El-Sharkawy, M. (2019) 'Thin MobileNet: An Enhanced MobileNet Architecture', *2019 IEEE 10th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2019*, pp. 0280–0285. doi: 10.1109/UEMCON47517.2019.8993089.