Configuration Manual

MSc. Research Project
MSc. Data Analytics

Shubham Kathepuri
Student ID: x18127398

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shubham Balasaheb Kathepuri
Student ID:	X18127398
Programme:	MSc. Data Analytics **Year:** 2019-2020
Module:	MSc. Research Project
Lecturer:	Dr. Catherine Mulwa
Submission Due Date:	28th September 2020
Project Title:	Recognition and Classification of Fruits using Deep Learning Techniques
Word Count:	1556 **Page Count:** 27

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: Shubham Balasaheb Kathepuri

Date: 27th September 2020

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

Shubham Kathepuri
x18127398

1 Introduction

To reproduce the research "Recognition and Classification of Fruits using Deep Learning" this manual presents thorough information regarding the configuration of the system, hardware requirements, and software requirement to implement, execute and test the models used for the research successfully. Further, the manual presents a step-by-step guide in the following sections.

2 System Configuration

This section discusses the hardware and software requirements.

2.1 Hardware Requirements

The hardware requirements are summarised in Table 1. Figure 1 displays the system configuration.

Table 1 Hardware Configuration

Hardware	**Configuration**
System	Dell XPS 15
OS	Windows 10 x64
Hard Disk	256 GB
RAM	8 GB
Processor	Intel Core i7 8th Generation
GPU	Nvidia GTX 1050Ti

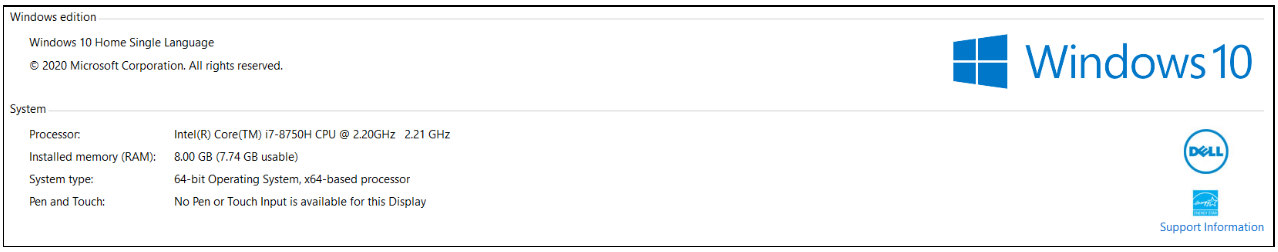

Figure 1 System Configuration

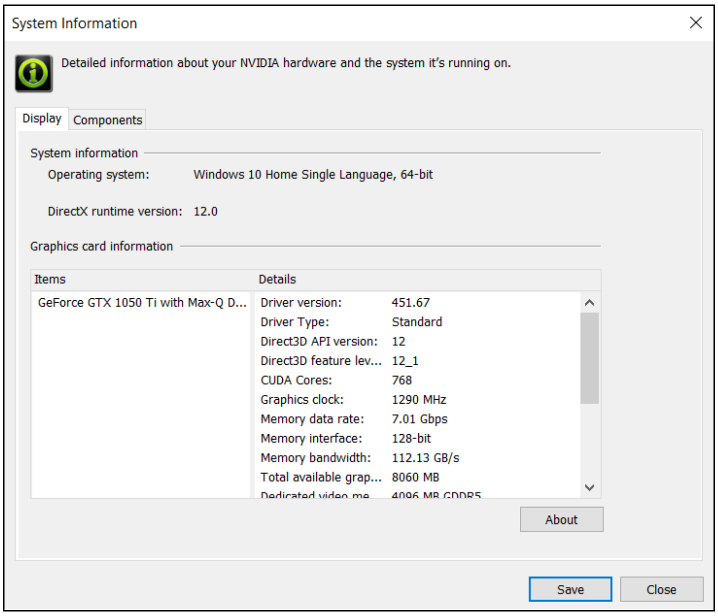

Figure 2 GPU Configuration

The GPU used for carrying out the project was Nvidia GTX 1050Ti (Figure 2).

2.2 Software Requirements

The list of software used for the research implementation is summarised in Table 2.

Table 2 Software Requirement

Software	Version
Python	3.7
Anaconda Navigator	1.9.12
Jupyter Notebooks	6.0.3
Google Chrome	84.0
Microsoft Excel	2020 Edition

2.2.1 Setting Anaconda Environment

The programming language used for the project was Python. To implement the project Anaconda Navigator environment was used as shown in Figure 3. Anaconda contains multiple applications that can be used for machine learning, development, and visualization. For this project, Jupyter Notebook was used for implementation purposes.

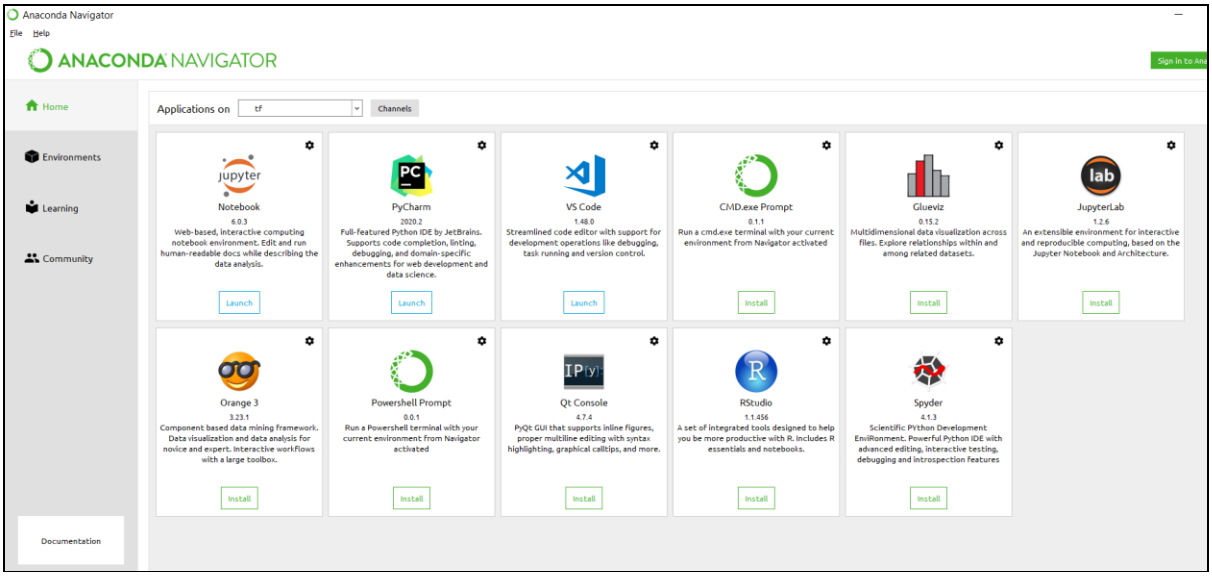

Figure 3 Anaconda Navigator Environment

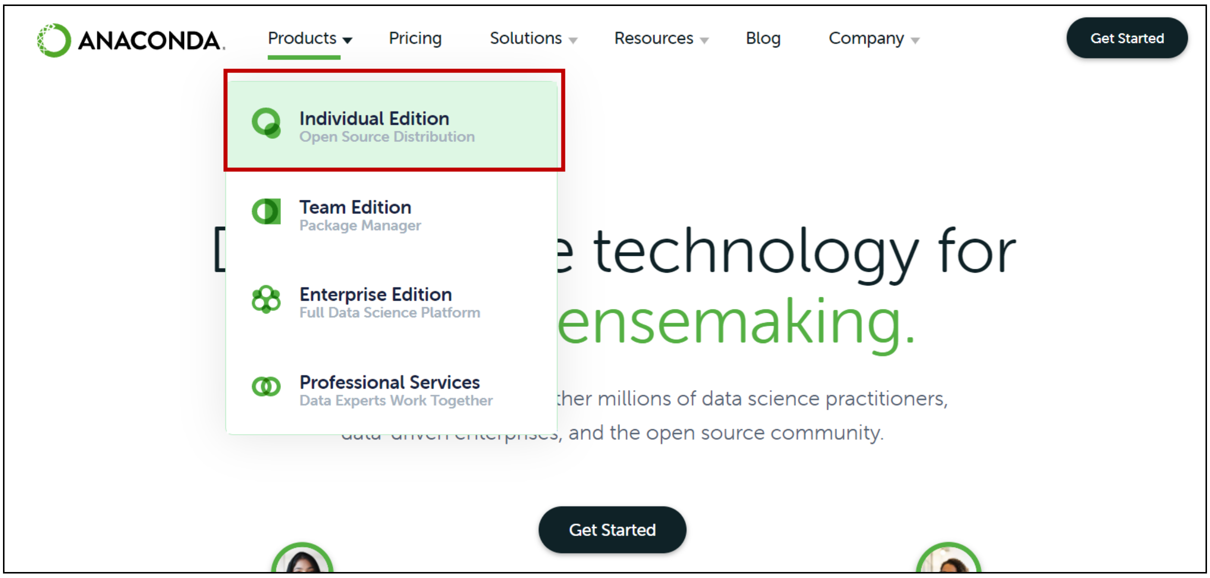

Figure 4 Downloading Anaconda

1. To download the Anaconda, visit the official website[1] and select the option shown in Figure 4.

[1] https://www.anaconda.com/products/individual

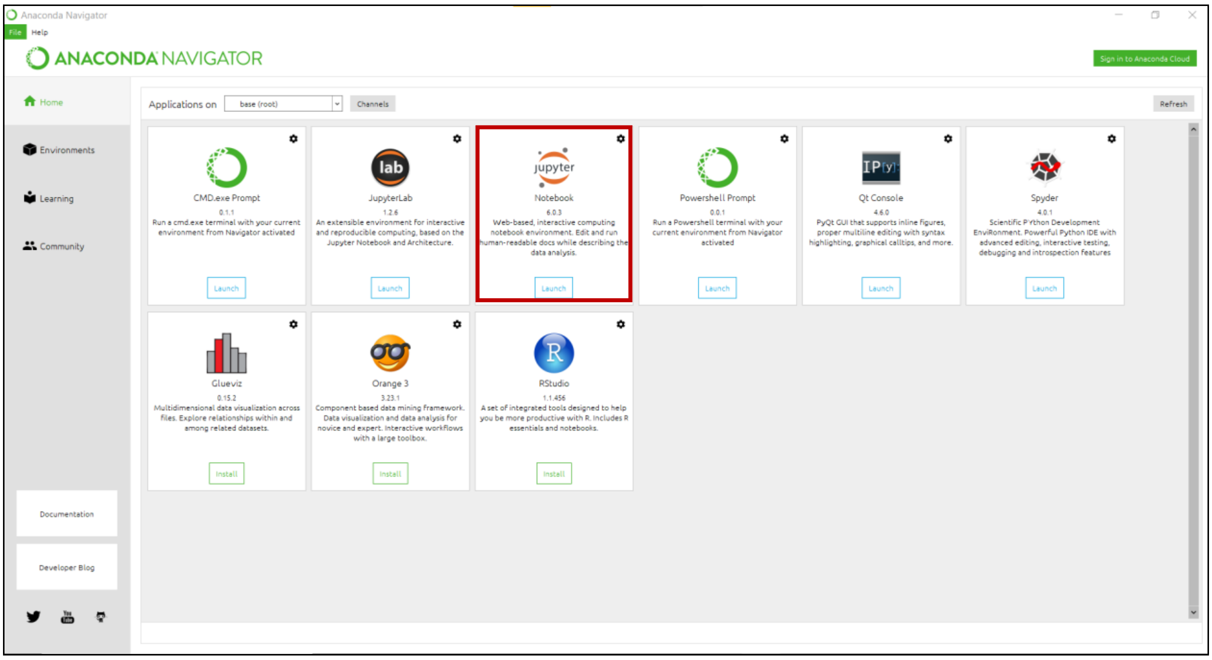

Figure 5 Install Jupyter Notebook

2. Install Jupyter Notebook from the Anaconda Navigator home screen as highlighted in Figure 5.

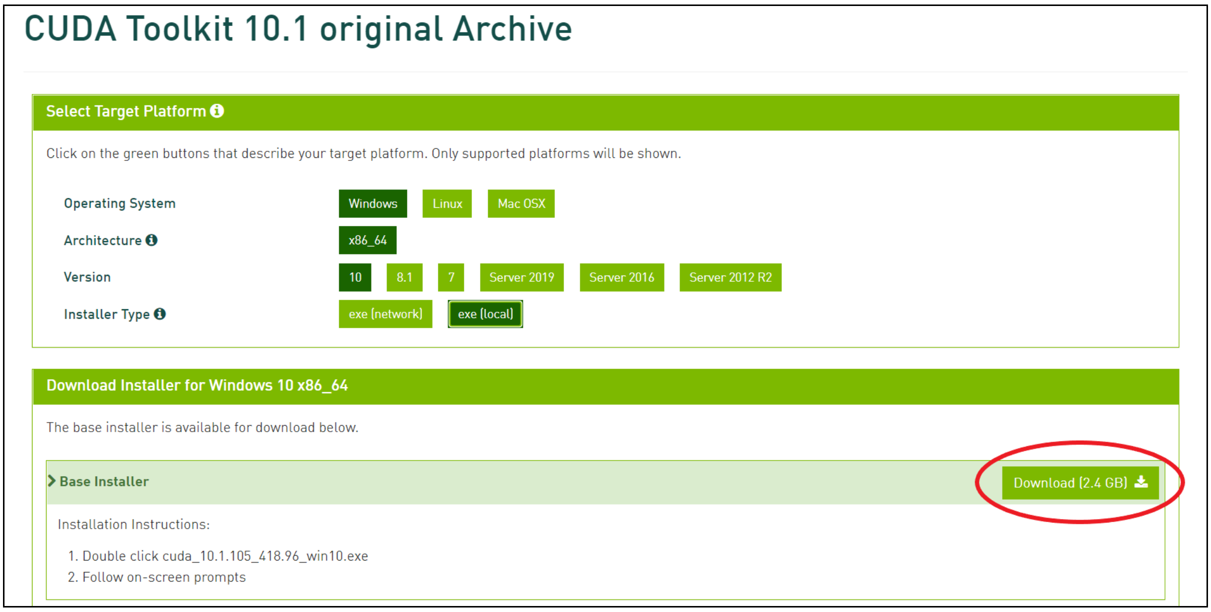

Figure 6 Install Cuda 10.1 for GPU

3. Install CUDA v.10.1 from the Nvidia Website[2] and cUDNN v.7.6.5 which compatible with the CUDA version from the archive option[3] (Figure 6).
4. To install TensorFlow for Anaconda environment, the command prompt of Anaconda was used and followed the instructions provided by the website[4].

2 https://developer.nvidia.com/cuda-10.1-download-archive
3 https://developer.nvidia.com/rdp/cudnn-archive#a-collapse765-101
4 https://docs.anaconda.com/anaconda/user-guide/tasks/tensorow/

3 Implementation

This section gives a step-by-step approach to reproduce the project which covers everything from data acquisition to model building, training, results, and visualizations.

3.1 Data Acquisition

The download of the data visit the GitHub[5] repository as highlighted in Figure 7.

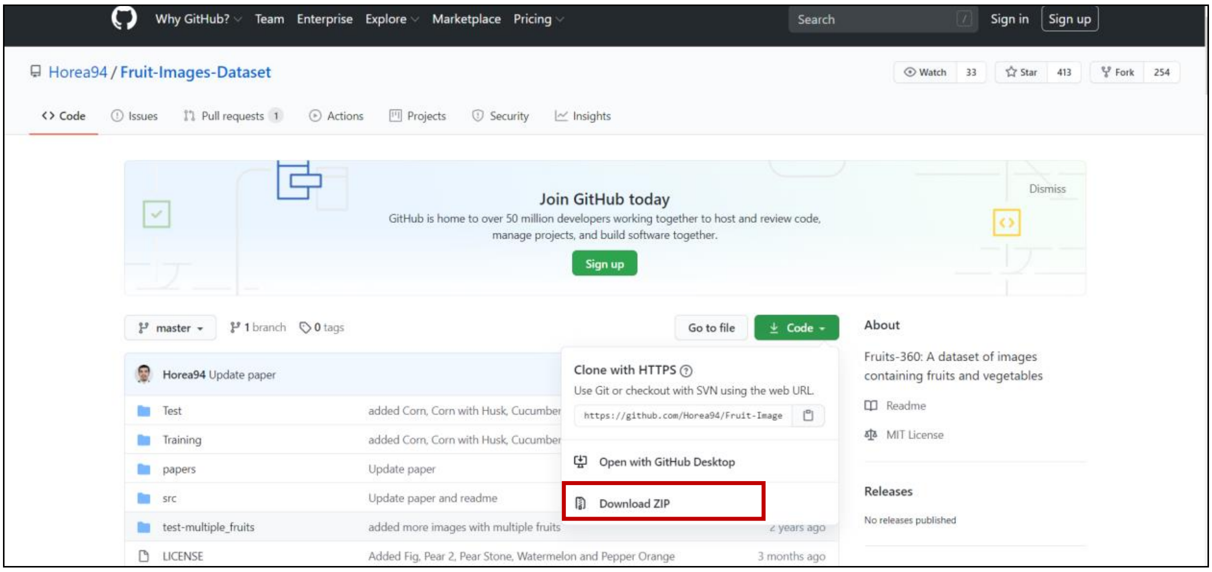

Figure 7 Data Acquisition

3.2 Data Preparation

The next step after data acquisition is data preparation. This step begins with importing the essential libraries required to implement the project (Figure 8).

Importing the essential libraries

```
In [66]: import os
import time
import random
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img
from os import listdir
from os.path import isfile, join
from glob import glob
from tqdm import tqdm
from IPython.display import display
from PIL import Image, ImageFile
from sklearn.datasets import load_files
from sklearn.metrics import classification_report, confusion_matrix
from keras.callbacks import ModelCheckpoint
from keras.preprocessing import image
from keras.utils import np_utils
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dropout, Flatten, Dense, Input
from keras.models import Sequential, Model, load_model
from keras.applications import ResNet50, VGG16
from keras.optimizers import Adam, SGD
```

Figure 8 Importing Essential Library

5 https://github.com/Horea94/Fruit-Images-Dataset

Seting up the dicrectories for train and test dataset

```
In [2]: # Root Directory
train_data_dir = 'C:/Users/Shubham/Downloads/NCI/Thesis/Fruit-Images-Dataset-master/Training'

#Validation Directory
test_data_dir = 'C:/Users/Shubham/Downloads/NCI/Thesis/Fruit-Images-Dataset-master/Test'
```

Figure 9 Directory Setup

To fetch the data, data directories are set up in Figure 9.

Defining Functions and Printing Category Names

```
In [3]: def show_directories(dir_list, path):
    print("######################################################################")
    print("Categories of Fruits")
    print("######################################################################")
    for index, dir in enumerate(dir_list):
        print (str(index+1) + ") ", dir)

def show_cwd_contents( path="." ):

    d_list = list()

    try:
        for f in os.listdir(path):
            if os.path.isfile(os.path.join(path, f)):
                f_list.append(f)
            else:
                if os.path.isdir(os.path.join(path, f)):
                    d_list.append(f)
    except:
        return

    show_directories(d_list, path)

if __name__ == "__main__":
    show_cwd_contents()
    show_cwd_contents(train_data_dir)
```

Figure 10 Function Definition for Displaying Categories

Functions are defined to print all the categories in dataset Figure 10.

Printing number of images per category of fruit

```
In [4]: print("######################################################################")
print("Image count per folder")
print("######################################################################")

path = train_data_dir
folders = ([name for name in os.listdir(path)
            if os.path.isdir(os.path.join(path, name))])
for folder in folders:
    contents = os.listdir(os.path.join(path,folder))
    if len(contents):
        print(folder,"=",len(contents))
```

Figure 11 Displaying Number of Images per Category

To print the number of images per category code shown in Figure 11 was used.

Displaying random image from each category for quality check and verification

```
In [5]: root_dir = train_data_dir
rows = 21
cols = 6
fig, ax = plt.subplots(rows, cols, frameon=False, figsize=(15, 25))
fig.suptitle('Random Image from Each Food Class', fontsize=20)
sorted_food_dirs = sorted(os.listdir(root_dir))
for i in range(rows):
    for j in range(cols):
        try:
            food_dir = sorted_food_dirs[i*cols + j]
        except:
            break
        all_files = os.listdir(os.path.join(root_dir, food_dir))
        rand_img = np.random.choice(all_files)
        img = plt.imread(os.path.join(root_dir, food_dir, rand_img))
        ax[i][j].imshow(img)
        ec = (0, .6, .1)
        fc = (0, .7, .2)
        ax[i][j].text(0, -20, food_dir, size=10, rotation=0,
                ha="left", va="top",
                bbox=dict(boxstyle="square",ec=ec, fc=fc))
plt.setp(ax, xticks=[], yticks=[])
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
fig.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/Dataset.jpg')
```

Figure 12 Quality Check and Category Verification

To verify the fruits in each category and check the quality of the images code shown in Figure 12 was used.

Loading the dataset into lists

```
In [58]: def load_dataset(path):
    data = load_files(path)
    fruit_files = np.array(data['filenames'])
    fruit_targets = np_utils.to_categorical(np.array(data['target']), 131)
    target_labels = np.array(data['target_names'])
    return fruit_files, fruit_targets, target_labels

# load train and test datasets
train_files, train_targets, train_labels = load_dataset(train_data_dir)
test_files, test_targets, test_labels = load_dataset(test_data_dir)

# load list of fruits names
fruit_names = [item[9:] for item in sorted(glob(train_data_dir + '/*'))]

# print statistics about the dataset
print('There are %d total fruit categories.' % len(fruit_names))
print('There are %s total fruit images.\n' % len(np.hstack([train_files, test_files])))
print('There are %d training fruit images.' % len(train_files))
print('There are %d test fruit images.'% len(test_files))
```

Figure 13 Loading Data and Displaying Statistics of Dataset

A user-defined function was used to load the dataset. The dataset statistics like the number of categories, the number of fruits in the train and test set, and the total number of images were print (Figure 13).

3.3 Data Pre-Processing

The next step after data preparation is data pre-processing.

Defining model parameters

```
In [7]: img_row = 100
img_height = 100
img_depth = 3
num_classes = 131
epochs = 5
batch_size=16
```

Figure 14 Define Model Parameters

Figure 14 shows the model parameters which were used for converting the images into an array, image augmentation, and model training.

Defining functions to convert the images into tensors using PIL library

```
In [56]: def path_to_tensor(img_path):
             # loads RGB image as PIL.Image.Image type
             img = image.load_img(img_path, target_size=(img_row, img_height))
             # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
             x = image.img_to_array(img)
             # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
             return np.expand_dims(x, axis=0)

         def paths_to_tensor(img_paths):
             list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
             return np.vstack(list_of_tensors)
```

Converting images into tensors

```
In [73]:
         ImageFile.LOAD_TRUNCATED_IMAGES = True

         # pre-process the data for Keras
         train_tensors = paths_to_tensor(train_files).astype('float32')/255

         100%|████████████████████████████████████████| 67692/67692 [01:38<00:00, 690.53it/s]

In [74]: test_tensors = paths_to_tensor(test_files).astype('float32')/255

         100%|████████████████████████████████████████| 22688/22688 [00:28<00:00, 796.93it/s]
```

Figure 15 Defining Function to Convert Images to Array (Tensors)

The images are converted into arrays Figure 15.

Image Augmentation

```
In [8]: from keras.preprocessing.image import ImageDataGenerator

        train_datagen = ImageDataGenerator(rescale=1./255,
                                           rotation_range=30,
                                           width_shift_range=0.3,
                                           height_shift_range=0.3,
                                           horizontal_flip=True,
                                           fill_mode='nearest',
                                           zoom_range = 0.2,
                                           validation_split=0.3)

        test_datagen = ImageDataGenerator(rescale=1./255)


        train_generator = train_datagen.flow_from_directory(train_data_dir,
                                                            target_size=(img_row, img_height),
                                                            batch_size =batch_size,
                                                            class_mode='categorical',
                                                            shuffle=True,
                                                            subset='training')

        validation_generator = train_datagen.flow_from_directory(train_data_dir,
                                                            target_size=(img_row, img_height),
                                                            batch_size=batch_size,
                                                            class_mode='categorical',
                                                            shuffle=False,
                                                            subset='validation')

        test_generator = test_datagen.flow_from_directory(test_data_dir,
                                                            target_size=(img_row, img_height),
                                                            batch_size=batch_size,
                                                            class_mode='categorical',
                                                            shuffle=False)
```

Figure 16 Image Augmentation

To answer the sub-RQ second set of training was created using ImageDataGenerator Library. The images were augmented using operations shown in Figure 16.

Printing Augmented Image

```
In [55]: print('Augmented Image')
         x_batch, y_batch = next(train_generator)
         i = random.randint(0,16)
         image = plt.figure(figsize=(4,4))
         fig= plt.imshow(x_batch[i])
         fig.axes.get_xaxis().set_visible(False)
         fig.axes.get_yaxis().set_visible(False)
         plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/Augmented_Image.jpg')
         plt.show()
```

Figure 17 Displaying Augmented Image

A sample of augmented images was printed in Figure 17.

Calculating Train and Validation Step for Augmented Models

```
In [ ]: train_step = train_generator.n//train_generator.batch_size
        val_step = validation_generator.n//validation_generator.batch_size
        print("Train step: ", train_step)
        print("Validation step: ", val_step)

        class_labels = validation_generator.class_indices
        class_labels = {v: k for k, v in class_labels.items()}
        classes = list(class_labels.values())
```

Figure 18 Calculating Train and Validation Step

Training and Validation steps were calculated for augmented models Figure 18.

3.4 Modeling

The next step after data pre-processing is modeling. In this step, three models were trained and tested (CNN, VGG16, ResNet50). Each of the models was trained using two sets of training data, image arrays and augmented data.

3.4.1 Base CNN

The base CNN model was developed from scratch which had the following configuration (Figure 19).

Base CNN

```
In [ ]: #Defining the architecture

        def base_cnn():
            model = Sequential()
            model.add(Conv2D(filters=16, kernel_size=2, padding='same',activation='relu',input_shape=(img_row,img_height,img_depth)))
            model.add(MaxPooling2D(pool_size=2))
            model.add(Conv2D(filters=32, kernel_size=2, padding='same',activation='relu'))
            model.add(MaxPooling2D(pool_size=2))
            model.add(Conv2D(filters=64, kernel_size=2, padding='same',activation='relu'))
            model.add(MaxPooling2D(pool_size=2))
            model.add(Conv2D(filters=128, kernel_size=2, padding='same',activation='relu'))
            model.add(MaxPooling2D(pool_size=2))
            model.add(Dropout(0.3))
            model.add(Flatten())
            model.add(Dense(500,activation='relu'))
            model.add(Dropout(0.4))
            model.add(Dense(131,activation='softmax'))
            model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
            model.summary()
            return model

In [ ]: model_base_cnn = base_cnn()

        WARNING:tensorflow:From C:\Users\Shubham\anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.m
        ax_pool is deprecated. Please use tf.nn.max_pool2d instead.

        Model: "sequential_1"
        _________________________________________________________________
        Layer (type)                 Output Shape              Param #
        =================================================================
        conv2d_1 (Conv2D)            (None, 100, 100, 16)      208
```

Figure 19 Base CNN Model Configuration

```
In [ ]: base_cnn_weight_path = 'C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.base_cnn.from_scratch.hdf5'

In [ ]: start = time.time()
        checkpointer = ModelCheckpoint(filepath=base_cnn_weight_path,
                                       verbose=1, save_best_only=True)

        base_cnn = model_base_cnn.fit(train_tensors, train_targets,
                  validation_split=0.3,
                  epochs=epochs, batch_size=batch_size, callbacks=[checkpointer], verbose=1)

        end = time.time()

        WARNING:tensorflow:From C:\Users\Shubham\anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.globa
        l_variables is deprecated. Please use tf.compat.v1.global_variables instead.

        Train on 47384 samples, validate on 20308 samples
        Epoch 1/5
        47384/47384 [==============================] - 127s 3ms/step - loss: 0.6977 - accuracy: 0.8102 - val_loss: 0.1773 - val_accurac
        y: 0.9455

        Epoch 00001: val_loss improved from inf to 0.17725, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.
        base_cnn.from_scratch.hdf5
        Epoch 2/5
        47384/47384 [==============================] - 142s 3ms/step - loss: 0.0913 - accuracy: 0.9728 - val_loss: 0.1472 - val_accurac
        y: 0.9562
```

Figure 20 Base CNN Model Training

The base CNN model was trained in Figure 20.

```
In [ ]: base_cnn_tt= end-start
        print('Time taken by the model to run: {}'.format(base_cnn_tt))

        Time taken by the model to run: 708.4216721057892

In [ ]: print(base_cnn.history.keys())

        # summarize history for accuracy
        plt.plot(base_cnn.history['accuracy'])
        plt.plot(base_cnn.history['val_accuracy'])
        plt.title('CNN - Model Accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/CNN_Graph-1.jpg')
        plt.show()

        # summarize history for loss
        plt.plot(base_cnn.history['loss'])
        plt.plot(base_cnn.history['val_loss'])
        plt.title('CNN - Model Loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/CNN_Graph-2.jpg')
        plt.show()
```

Figure 21 Computational Time, and Accuracy and Loss Plots

The computational time required, and accuracy and loss were printed in Figure 21.

```
In [ ]: cnn_train_accuracy=base_cnn.model.evaluate(train_tensors, train_targets)
        print('Train accuracy: %.4f%%' % (cnn_train_accuracy[1]*100))

        67692/67692 [==============================] - 100s 1ms/step
        Train accuracy: 99.9350%

In [ ]: cnn_test_accuracy=base_cnn.model.evaluate(test_tensors, test_targets)
        print('Test accuracy: %.4f%%' % (cnn_test_accuracy[1]*100))

        22688/22688 [==============================] - 31s 1ms/step
        Test accuracy: 96.9103%
```

Figure 22 Calculating Model Accuracy

The training and test accuracy were printed in Figure 22.

```
In [ ]: y_pred = model_base_cnn.predict(test_tensors)

        fig = plt.figure(figsize=(16,12))
        for i, idx in enumerate(np.random.choice(test_tensors.shape[0], size=16, replace=False)):
            ax = fig.add_subplot(4,4, i + 1, xticks=[], yticks=[])
            ax.imshow(np.squeeze(test_tensors[idx]))
            pred_idx = np.argmax(y_pred[idx])
            true_idx = np.argmax(test_targets[idx])
            ax.set_title("P:{}, A:{}".format(test_labels[pred_idx], test_labels[true_idx]),
                         color=("green" if pred_idx == true_idx else "red"))

        fig.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/CNN_Prediction.jpg')
```

Figure 23 Prediction on Random 16 Images

To verify the test accuracy the random 16 images were used for predictions and the results were visualized Figure 23.

```
In [24]: model = load_model(base_cnn_weight_path)

         #Confution Matrix and Classification Report
         Y_pred = model.predict_generator(test_generator, test_generator.n//test_generator.batch_size)
         y_pred = np.argmax(Y_pred, axis=1)

         target_names = list(class_labels.values())

         plt.figure(figsize=(30,30))
         cnf_matrix = confusion_matrix(test_generator.classes, y_pred)

         plt.imshow(cnf_matrix, interpolation='nearest')
         plt.colorbar()
         tick_marks = np.arange(len(classes))
         _ = plt.xticks(tick_marks, classes, rotation=90)
         _ = plt.yticks(tick_marks, classes)
         plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/CNN_CM.jpg')
```

Figure 24 Base CNN Confusion Matrix

The correct and incorrect predictions made by the model were visualized using confusion matrix Figure 24 and Figure 25.

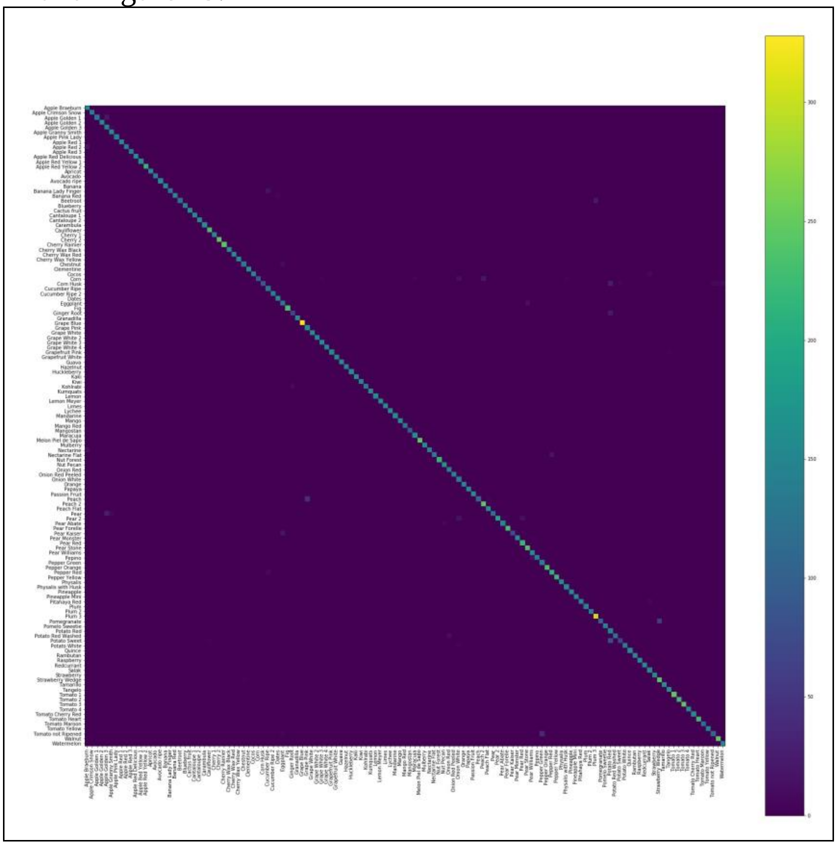

Figure 25 Base CNN Confusion Matrix

3.4.2 CNN with Augmentation

The augmented CNN model had the same configuration as the base model Figure 26.

CNN with Augmentation

```
In [ ]: def aug_cnn():
            model = Sequential()
            model.add(Conv2D(filters=16, kernel_size=2, padding='same',activation='relu',input_shape=(img_row,img_height,img_depth)))
            model.add(MaxPooling2D(pool_size=2))
            model.add(Conv2D(filters=32, kernel_size=2, padding='same',activation='relu'))
            model.add(MaxPooling2D(pool_size=2))
            model.add(Conv2D(filters=64, kernel_size=2, padding='same',activation='relu'))
            model.add(MaxPooling2D(pool_size=2))
            model.add(Conv2D(filters=128, kernel_size=2, padding='same',activation='relu'))
            model.add(MaxPooling2D(pool_size=2))
            model.add(Dropout(0.3))
            model.add(Flatten())
            model.add(Dense(500,activation='relu'))
            model.add(Dropout(0.4))
            model.add(Dense(131,activation='softmax'))
            model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
            model.summary()
            return model

In [ ]: model_aug_cnn = aug_cnn()
        aug_cnn_weight_path = 'C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.aug_cnn.from_scratch.hdf5'

        Model: "sequential_2"
        _________________________________________________________________
        Layer (type)                 Output Shape              Param #
        =================================================================
        conv2d_5 (Conv2D)            (None, 100, 100, 16)      208
        _________________________________________________________________
        max_pooling2d_5 (MaxPooling2 (None, 50, 50, 16)        0
        _________________________________________________________________
        conv2d_6 (Conv2D)            (None, 50, 50, 32)        2080
        _________________________________________________________________
        max_pooling2d_6 (MaxPooling2 (None, 25, 25, 32)        0
        _________________________________________________________________
        conv2d_7 (Conv2D)            (None, 25, 25, 64)        8256
        _________________________________________________________________
```

Figure 26 CNN with Augmentation Configuration

```
In [ ]: from keras.callbacks import ModelCheckpoint

        start = time.time()

        checkpoint = ModelCheckpoint(aug_cnn_weight_path,
                                     monitor="val_loss",
                                     mode="min",
                                     save_best_only = True,
                                     verbose=1)

        callbacks = [checkpoint]

        cnn_aug = model_aug_cnn.fit(train_generator,
                                    steps_per_epoch =train_step,
                                    epochs = epochs,
                                    callbacks = callbacks,
                                    validation_data = validation_generator,
                                    validation_steps = val_step)

        end = time.time()

        Epoch 1/5
        2964/2964 [==============================] - 297s 100ms/step - loss: 1.9573 - accuracy: 0.4467 - val_loss: 1.4070 - val_accurac
        y: 0.7678

        Epoch 00001: val_loss improved from inf to 1.40698, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.
        aug_cnn.from_scratch.hdf5
        Epoch 2/5
        2964/2964 [==============================] - 209s 70ms/step - loss: 0.6376 - accuracy: 0.7980 - val_loss: 0.0961 - val_accurac
        y: 0.8063

        Epoch 00002: val_loss improved from 1.40698 to 0.09610, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weig
        hts.aug_cnn.from_scratch.hdf5
        Epoch 3/5
        2964/2964 [==============================] - 237s 80ms/step - loss: 0.4609 - accuracy: 0.8610 - val_loss: 0.2359 - val_accurac
        y: 0.8842
```

Figure 27 CNN with Augmentation Model Training

The augmented CNN model was trained in Figure 27.

```
In [ ]: cnn_aug_tt= end-start
        print('Time taken by the model to run: {}'.format(cnn_aug_tt))

        Time taken by the model to run: 1202.9638531208038

In [ ]: print(cnn_aug.history.keys())

        # summarize history for accuracy
        plt.plot(cnn_aug.history['accuracy'])
        plt.plot(cnn_aug.history['val_accuracy'])
        plt.title('CNN with Augmentation - Model Accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/CNN_AUG_Graph-1.jpg')
        plt.show()

        # summarize history for loss
        plt.plot(cnn_aug.history['loss'])
        plt.plot(cnn_aug.history['val_loss'])
        plt.title('CNN with Augmentation - Model Loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/CNN_AUG_Graph-2.jpg')
        plt.show()
```

Figure 28 Computational Time, and Accuracy and Loss Plots

The computational time required, and accuracy and loss were printed in Figure 28.

```
In [ ]: cnn_aug_train_accuracy=cnn_aug.model.evaluate(train_generator)
        print('Train accuracy: %.4f%%' % (cnn_aug_train_accuracy[1]*100))

        2965/2965 [==============================] - 146s 49ms/step
        Train accuracy: 93.1916%

In [ ]: cnn_aug_test_accuracy=cnn_aug.model.evaluate(test_generator)
        print('Test accuracy: %.4f%%' % (cnn_aug_test_accuracy[1]*100))

        1418/1418 [==============================] - ETA:  - 25s 18ms/step
        Test accuracy: 92.5555%
```

Figure 29 CNN with Augmentation Accuracy

The training and test accuracy were printed in Figure 29.

```
In [ ]: y_pred = model_aug_cnn.predict(test_tensors)

        fig = plt.figure(figsize=(16,12))
        for i, idx in enumerate(np.random.choice(test_tensors.shape[0], size=16, replace=False)):
            ax = fig.add_subplot(4,4, i + 1, xticks=[], yticks=[])
            ax.imshow(np.squeeze(test_tensors[idx]))
            pred_idx = np.argmax(y_pred[idx])
            true_idx = np.argmax(test_targets[idx])
            ax.set_title("P:{}, A:{}".format(test_labels[pred_idx], test_labels[true_idx]),
                         color=("green" if pred_idx == true_idx else "red"))

        fig.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/CNN_AUG_Prediction.jpg')
```

Figure 30 Prediction on Random 16 Images

To verify the test accuracy the random 16 images were used for predictions and the results were visualized Figure 30.

```
[ ]: model = load_model(aug_cnn_weight_path)

     #Confution Matrix and Classification Report
     Y_pred = model.predict_generator(test_generator, test_generator.n//test_generator.batch_size)
     y_pred = np.argmax(Y_pred, axis=1)

     target_names = list(class_labels.values())

     plt.figure(figsize=(30,30))
     cnf_matrix = confusion_matrix(test_generator.classes, y_pred)

     plt.imshow(cnf_matrix, interpolation='nearest')
     plt.colorbar()
     tick_marks = np.arange(len(classes))
     _ = plt.xticks(tick_marks, classes, rotation=90)
     _ = plt.yticks(tick_marks, classes)
     plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/CNN_AUG_CM.jpg')
```

Figure 31 CNN with Augmentation Confusion Matrix

The correct and incorrect predictions made by the model were visualized using confusion matrix Figure 31 and Figure 32.

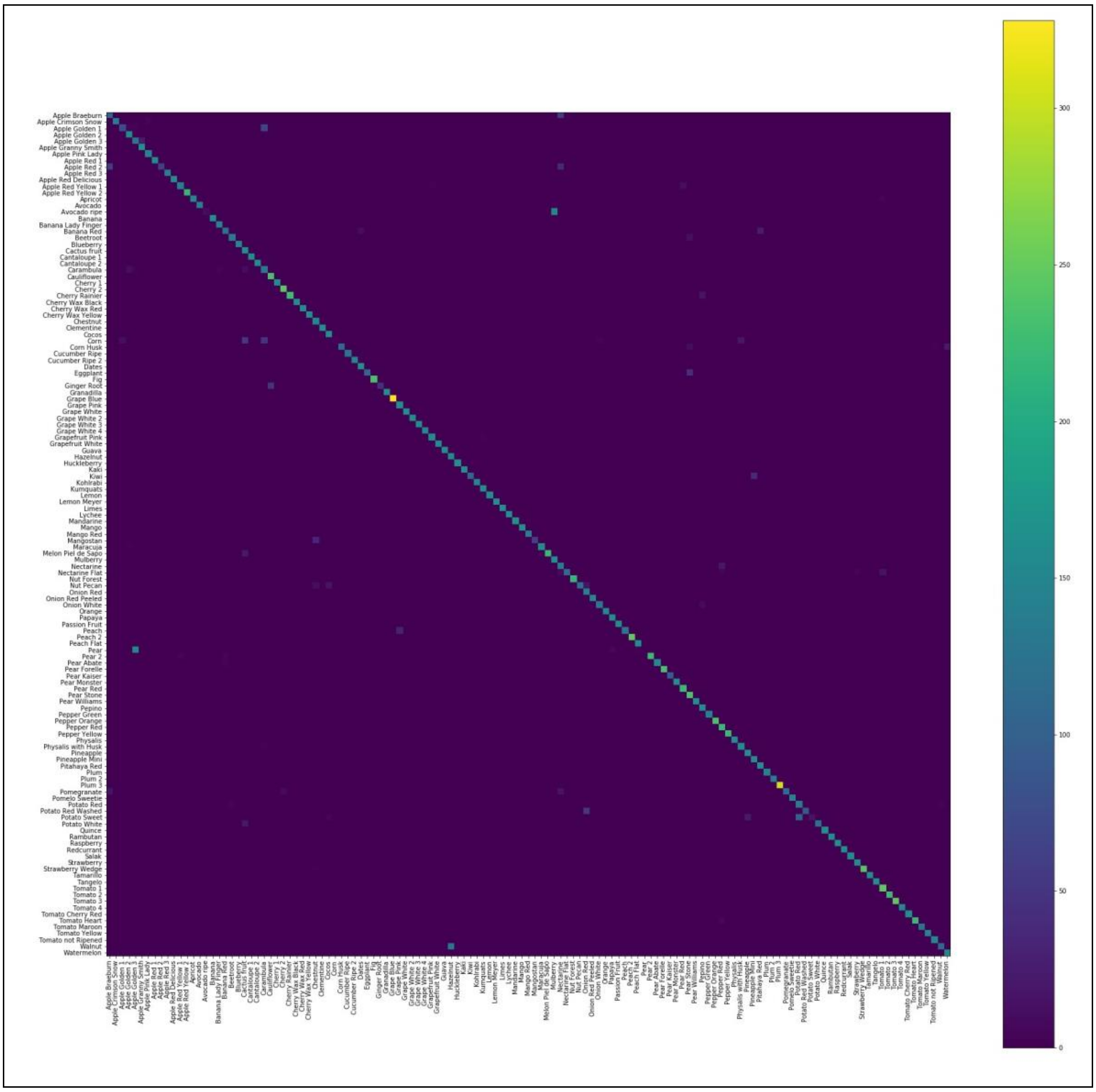

Figure 32 CNN with Augmentation Confusion Matrix

3.4.3 Base VGG16

The base VGG16 model was developed from scratch which had the following configuration (Figure 33).

Base VGG16

```
In [76]: #VGG16

def vgg16():
    model_vgg16_conv = VGG16(weights='imagenet', include_top=False)

    input_model = Input(shape=(img_row,img_height,img_depth))

    output_vgg16_conv = model_vgg16_conv(input_model)

    x = Flatten()(output_vgg16_conv)
    x = Dropout(0.3)(x)
    x = Dense(131, activation='softmax', name='predictions')(x)

    model = Model(input_model, x)

    for layer in model_vgg16_conv.layers[:]:
      layer.trainable = False

    model.compile(Adam(lr=1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
    model.summary()
    return model

In [77]: model_vgg = vgg16()

Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 100, 100, 3)       0
_________________________________________________________________
vgg16 (Model)                multiple                  14714688
_________________________________________________________________
flatten_3 (Flatten)          (None, 4608)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4608)              0
_________________________________________________________________
```

Figure 33 Base VGG16 Configuration

```
In [79]: base_vgg_weight_path = 'C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.base_vgg.from_scratch.hdf5'

In [ ]: from keras.callbacks import ModelCheckpoint

start= time.time()

checkpointer = ModelCheckpoint(filepath=base_vgg_weight_path,
                               verbose=1, save_best_only=True)

base_vgg = model_vgg.fit(train_tensors, train_targets,
          validation_split=0.3,
          epochs=epochs, batch_size=batch_size, callbacks=[checkpointer], verbose=1)

end = time.time()

Train on 47384 samples, validate on 20308 samples
Epoch 1/5
47384/47384 [==============================] - 373s 8ms/step - loss: 0.3916 - accuracy: 0.9272 - val_loss: 0.0292 - val_accurac
y: 0.9973

Epoch 00001: val_loss improved from inf to 0.02918, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.
base_vgg.from_scratch.hdf5
Epoch 2/5
47384/47384 [==============================] - 335s 7ms/step - loss: 0.0260 - accuracy: 0.9967 - val_loss: 0.0086 - val_accurac
y: 0.9995

Epoch 00002: val_loss improved from 0.02918 to 0.00857, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weig
hts.base_vgg.from_scratch.hdf5
Epoch 3/5
47384/47384 [==============================] - 266s 6ms/step - loss: 0.0150 - accuracy: 0.9969 - val_loss: 0.0103 - val_accurac
y: 0.9968
```

Figure 34 VGG16 Model Training

The base VGG16 model was trained in Figure 34.

```
In [ ]: base_vgg_tt= end-start
        print('Time taken by the model to run: {}'.format(base_vgg_tt))

        Time taken by the model to run: 1576.9526495933533

In [ ]: print(base_vgg.history.keys())

        # summarize history for accuracy
        plt.plot(base_vgg.history['accuracy'])
        plt.plot(base_vgg.history['val_accuracy'])
        plt.title('VGG - Model Accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/VGG_Graph-1.jpg')
        plt.show()


        # summarize history for loss
        plt.plot(base_vgg.history['loss'])
        plt.plot(base_vgg.history['val_loss'])
        plt.title('VGG - Model Loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/VGG_Graph-2.jpg')
        plt.show()
```

Figure 35 Computational Time, and Accuracy and Loss Plots

The computational time required, and accuracy and loss were printed Figure 35

```
In [ ]: vgg_train_accuracy=base_vgg.model.evaluate(train_tensors, train_targets)
        print('Train accuracy: %.4f%%' % (vgg_train_accuracy[1]*100))

        67692/67692 [==============================] - 225s 3ms/step
        Train accuracy: 99.9970%

In [ ]: vgg_test_accuracy=base_vgg.model.evaluate(test_tensors, test_targets)
        print('Test accuracy: %.4f%%' % (vgg_test_accuracy[1]*100))

        22688/22688 [==============================] - 102s 4ms/step
        Test accuracy: 96.3902%
```

Figure 36 VGG16 Accuracy

The training and test accuracy were printed in Figure 36.

```
In [83]: y_pred = model_vgg.predict(test_tensors)

         fig = plt.figure(figsize=(16,12))
         for i, idx in enumerate(np.random.choice(test_tensors.shape[0], size=16, replace=False)):
             ax = fig.add_subplot(4,4, i + 1, xticks=[], yticks=[])
             ax.imshow(np.squeeze(test_tensors[idx]))
             pred_idx = np.argmax(y_pred[idx])
             true_idx = np.argmax(test_targets[idx])
             ax.set_title("P:{}, A:{}".format(test_labels[pred_idx], test_labels[true_idx]),
                          color=("green" if pred_idx == true_idx else "red"))

         fig.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/VGG_Prediction.jpg')
```

Figure 37 VGG16 Prediction on Random 16 Images

To verify the test accuracy the random 16 images were used for predictions and the results were visualized Figure 37.

```
In [ ]: model = load_model(base_vgg_weight_path)

        #Confution Matrix and Classification Report
        Y_pred = model.predict_generator(test_generator, test_generator.n//test_generator.batch_size)
        y_pred = np.argmax(Y_pred, axis=1)

        target_names = list(class_labels.values())

        plt.figure(figsize=(30,30))
        cnf_matrix = confusion_matrix(test_generator.classes, y_pred)

        plt.imshow(cnf_matrix, interpolation='nearest')
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        _ = plt.xticks(tick_marks, classes, rotation=90)
        _ = plt.yticks(tick_marks, classes)
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/VGG_CM.jpg')
```

Figure 38 VGG16 Confusion Matrix

The correct and incorrect predictions made by the model were visualized using confusion matrix Figure 38 and Figure 39.

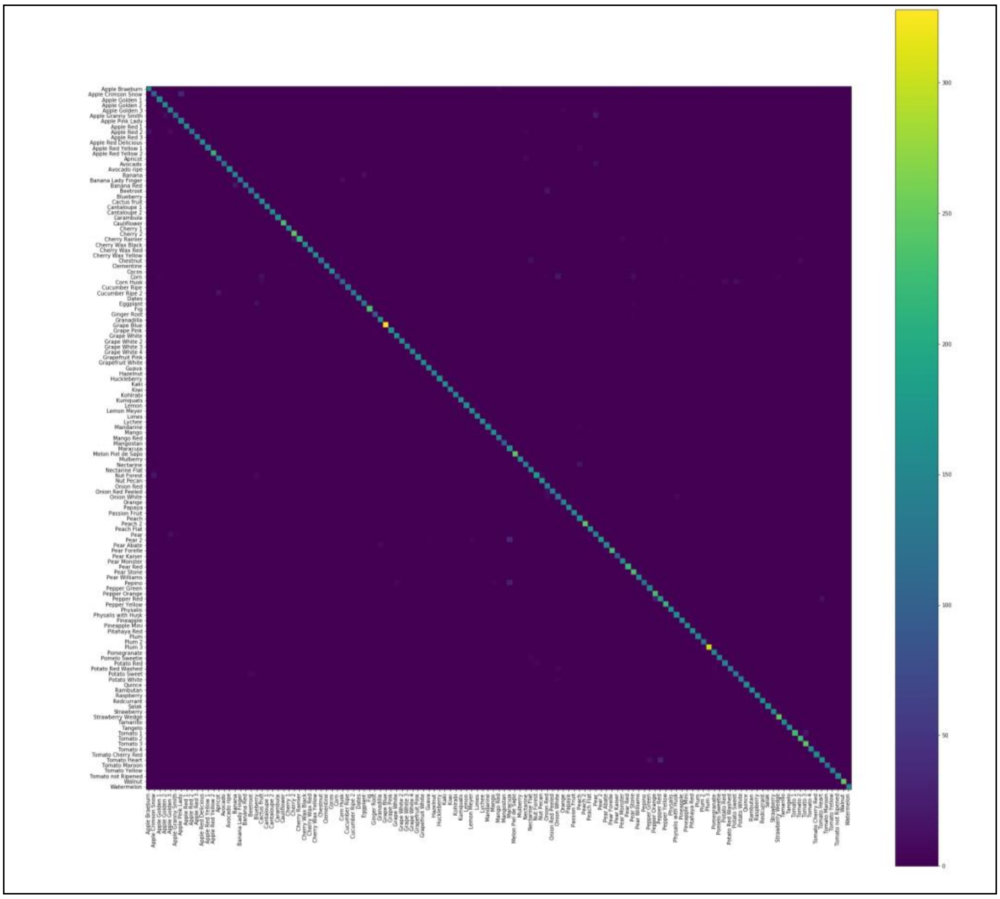

Figure 39 VGG16 Confusion Matrix

3.4.4 VGG16 with Augmentation

The augmented VGG16 model had the same configuration as the base model Figure 40.

VGG16 with augmentation

```
In [ ]: def aug_vgg16():
            model_vgg16_conv = VGG16(weights='imagenet', include_top=False)

            input_model = Input(shape=(img_row,img_height,img_depth))

            output_vgg16_conv = model_vgg16_conv(input_model)

            x = Flatten()(output_vgg16_conv)
            x = Dropout(0.3)(x)
            x = Dense(131, activation='softmax', name='predictions')(x)

            model = Model(input_model, x)

            for layer in model_vgg16_conv.layers[:]:
              layer.trainable = False

            model.compile(Adam(lr=1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
            model.summary()
            return model
```

```
In [ ]: model_aug_vgg = aug_vgg16()

        Model: "model_2"
        _________________________________________________________________
        Layer (type)                 Output Shape              Param #
        =================================================================
        input_4 (InputLayer)         (None, 100, 100, 3)       0
        _________________________________________________________________
        vgg16 (Model)                multiple                  14714688
        _________________________________________________________________
        flatten_4 (Flatten)          (None, 4608)              0
        _________________________________________________________________
        dropout_6 (Dropout)          (None, 4608)              0
        _________________________________________________________________
        predictions (Dense)          (None, 131)               603779
```

Figure 40 VGG16 with Augmentation Configuration

```
In [ ]: aug_vgg_weight_path = 'C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.aug_vgg.from_scratch.hdf5'

In [ ]: from keras.callbacks import ModelCheckpoint

        start = time.time()

        checkpoint = ModelCheckpoint(aug_vgg_weight_path,
                                     monitor="val_loss",
                                     mode="min",
                                     save_best_only = True,
                                     verbose=1)

        callbacks = [checkpoint]

        vgg_aug = model_aug_vgg.fit(
            train_generator,
            steps_per_epoch = train_step,
            epochs = epochs,
            callbacks = callbacks,
            validation_data = validation_generator,
            validation_steps = val_step)

        end = time.time()

        Epoch 1/5
        2964/2964 [==============================] - 254s 86ms/step - loss: 1.6225 - accuracy: 0.5881 - val_loss: 0.0293 - val_accurac
        y: 0.6712

        Epoch 00001: val_loss improved from inf to 0.02934, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.
        aug_vgg.from_scratch.hdf5
        Epoch 2/5
        2964/2964 [==============================] - 254s 86ms/step - loss: 0.7891 - accuracy: 0.7728 - val_loss: 8.1020e-04 - val_accu
        racy: 0.7272

        Epoch 00002: val_loss improved from 0.02934 to 0.00081, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weig
        hts.aug_vgg.from_scratch.hdf5
```

Figure 41 VGG16 with Augmentation Model Training

The augmented VGG16 model was trained in Figure 41.

```
In [ ]: vgg_aug_tt= end-start
        print('Time taken by the model to run: {}'.format(vgg_aug_tt))

        Time taken by the model to run: 1310.2697055339813

In [ ]: print(vgg_aug.history.keys())

        dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])

In [ ]: # summarize history for accuracy
        plt.plot(vgg_aug.history['accuracy'])
        plt.plot(vgg_aug.history['val_accuracy'])
        plt.title('VGG with Augmentation - Model Accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/VGG_AUG_Graph-1.jpg')
        plt.show()


        # summarize history for loss
        plt.plot(vgg_aug.history['loss'])
        plt.plot(vgg_aug.history['val_loss'])
        plt.title('VGG with Augmentation - Model Loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/VGG_AUG_Graph-2.png')
        plt.show()
```

Figure 42 Computational Time, and Accuracy and Loss Plots

The computational time required, and accuracy and loss were printed in Figure 42.

```
In [ ]: vgg_aug_train_accuracy=vgg_aug.model.evaluate(train_generator)
        print('Train accuracy: %.4f%%' % (vgg_aug_train_accuracy[1]*100))

        2965/2965 [==============================] - 181s 61ms/step
        Train accuracy: 89.6346%

In [ ]: vgg_aug_test_accuracy=vgg_aug.model.evaluate(test_generator)
        print('Test accuracy: %.4f%%' % (vgg_aug_test_accuracy[1]*100))

        1418/1418 [==============================] - 69s 49ms/step
        Test accuracy: 86.0146%
```

Figure 43 VGG16 with Augmentation Accuracy

The training and test accuracy were printed in Figure 43.

```
In [ ]: y_pred = model_aug_vgg.predict(test_tensors)

        fig = plt.figure(figsize=(16,12))
        for i, idx in enumerate(np.random.choice(test_tensors.shape[0], size=16, replace=False)):
            ax = fig.add_subplot(4,4, i + 1, xticks=[], yticks=[])
            ax.imshow(np.squeeze(test_tensors[idx]))
            pred_idx = np.argmax(y_pred[idx])
            true_idx = np.argmax(test_targets[idx])
            ax.set_title("P:{}, A:{}".format(test_labels[pred_idx], test_labels[true_idx]),
                         color=("green" if pred_idx == true_idx else "red"))
        fig.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/VGG_AUG_Prediction.jpg')
```

Figure 44 VGG16 with Augmentation Prediction on Random 16 Images

To verify the test accuracy the random 16 images were used for predictions and the results were visualized Figure 44.

```python
In [ ]: model = load_model(aug_vgg_weight_path)

        #Confution Matrix and Classification Report
        Y_pred = model.predict_generator(test_generator, test_generator.n//test_generator.batch_size)
        y_pred = np.argmax(Y_pred, axis=1)

        target_names = list(class_labels.values())

        plt.figure(figsize=(30,30))
        cnf_matrix = confusion_matrix(test_generator.classes, y_pred)

        plt.imshow(cnf_matrix, interpolation='nearest')
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        _ = plt.xticks(tick_marks, classes, rotation=90)
        _ = plt.yticks(tick_marks, classes)
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/VGG_AUG_CM.jpg')
```

Figure 45 VGG16 with Augmentation Confusion Matrix

The correct and incorrect predictions made by the model were visualized using confusion matrix Figure 45 and Figure 46.

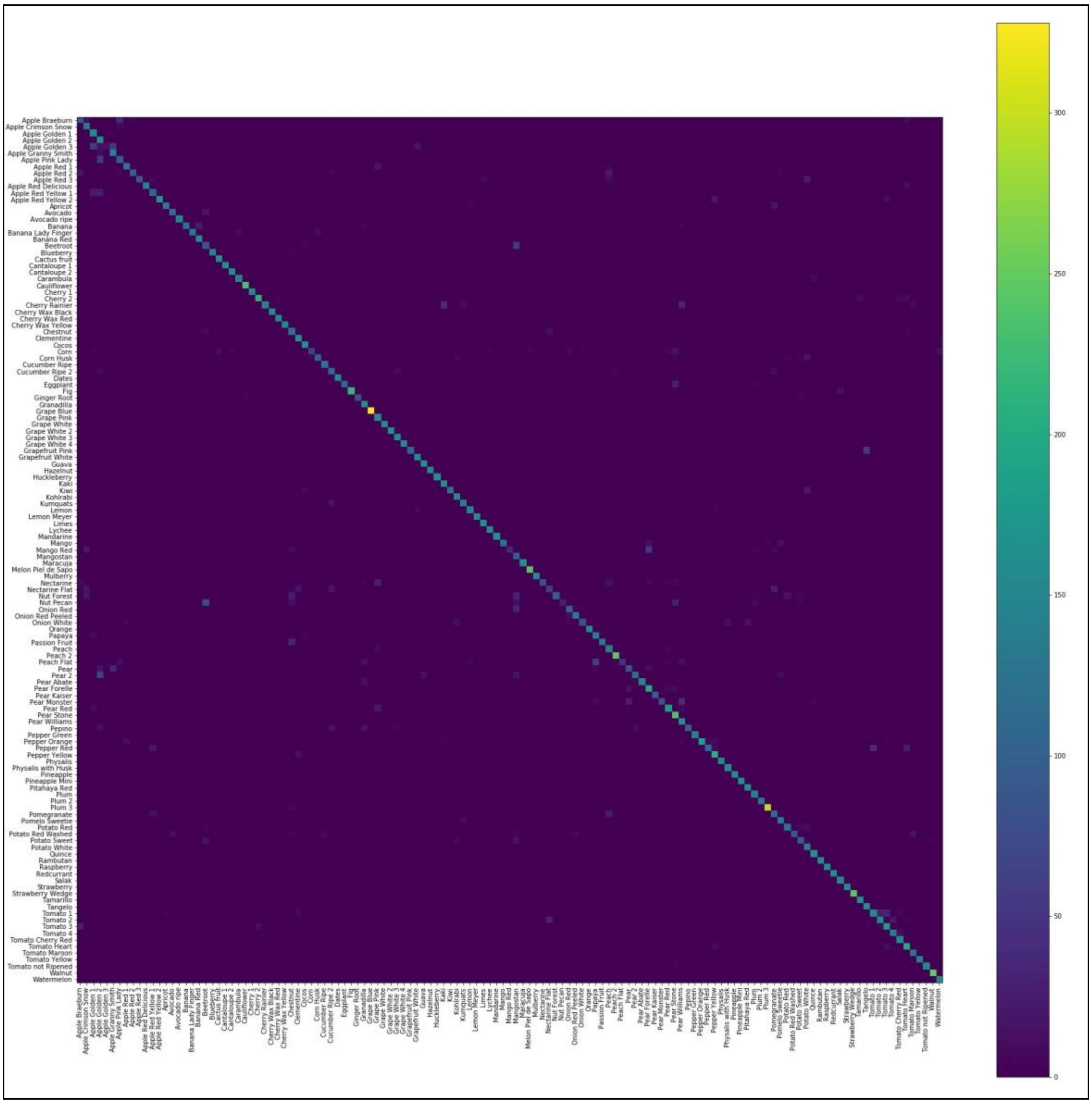

Figure 46 VGG16 with Augmentation Confusion Matrix

3.4.5 Base ResNet50

The base ResNet50 model was developed from scratch which had the following configuration (Figure 47).

Base ResNet50

```
In [64]: # Resnet50

def resnet50():
    model_resnet50_conv = ResNet50(weights='imagenet', include_top=False)

    input_model = Input(shape=(img_row,img_height,img_depth))

    output_resnet50_conv = model_resnet50_conv(input_model)

    x = Flatten()(output_resnet50_conv)
    x = Dropout(0.3)(x)
    x = Dense(131, activation='softmax', name='predictions')(x)

    model = Model(input_model, x)

    for layer in model_resnet50_conv.layers[:]:
      layer.trainable = False

    model.compile(SGD(lr=1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
    model.summary()
    return model

In [67]: model_resnet = resnet50()

Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 100, 100, 3)       0
_________________________________________________________________
resnet50 (Model)             multiple                  23587712
_________________________________________________________________
```

Figure 47 Base ResNet50 Configuration

```
In [68]: resnet_weight_path = 'C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.resnet_SGD.hdf5'

In [ ]: from keras.callbacks import ModelCheckpoint

start = time.time()

checkpointer = ModelCheckpoint(filepath=resnet_weight_path,
                               verbose=1, save_best_only=True)

base_resnet = model_resnet.fit(train_tensors, train_targets,
          validation_split=0.3,
          epochs=epochs, batch_size=batch_size, callbacks=[checkpointer], verbose=1)

end = time.time()

Train on 47384 samples, validate on 20308 samples
Epoch 1/5
47384/47384 [==============================] - 425s 9ms/step - loss: 0.3248 - accuracy: 0.9287 - val_loss: 0.0910 - val_accurac
y: 0.99520.92

Epoch 00001: val_loss improved from inf to 0.09103, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.
resnet_SGD.hdf5
Epoch 2/5
47384/47384 [==============================] - 364s 8ms/step - loss: 0.0269 - accuracy: 0.9946 - val_loss: 0.1034 - val_accurac
y: 0.9959

Epoch 00002: val_loss did not improve from 0.09103
Epoch 3/5
47384/47384 [==============================] - 277s 6ms/step - loss: 0.0166 - accuracy: 0.9969 - val_loss: 0.0859 - val_accurac
y: 0.9959

Epoch 00003: val_loss improved from 0.09103 to 0.08587, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weig
hts.resnet_SGD.hdf5
Epoch 4/5
47384/47384 [==============================] - 301s 6ms/step - loss: 0.0112 - accuracy: 0.9980 - val_loss: 0.0890 - val_accurac
y: 0.9959
```

Figure 48 Base ResNet Model Training

The base ResNet50 model was trained in Figure 48.

```
In [ ]: base_resnet_tt= end-start
        print('Time taken by the model to run: {}'.format(base_resnet_tt))

        Time taken by the model to run: 1740.072785615921

In [ ]: print(base_resnet.history.keys())

        dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])

In [ ]: # summarize history for accuracy
        plt.plot(base_resnet.history['accuracy'])
        plt.plot(base_resnet.history['val_accuracy'])
        plt.title('ResNet - Model Accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/ResNet_Graph-1-1.jpg')
        plt.show()


        # summarize history for loss
        plt.plot(base_resnet.history['loss'])
        plt.plot(base_resnet.history['val_loss'])
        plt.title('ResNet - Model Loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/ResNet_Graph-2-1.jpg')
        plt.show()
```

Figure 49 Computational Time, and Accuracy and Loss Plots

The computational time required, and accuracy and loss were printed in Figure 49.

```
In [ ]: resnet_train_accuracy=base_resnet.model.evaluate(train_tensors, train_targets)
        print('Train accuracy: %.4f%%' % (resnet_train_accuracy[1]*100))

        67692/67692 [==============================] - 267s 4ms/step
        Train accuracy: 99.6174%

In [ ]: resnet_test_accuracy=base_resnet.model.evaluate(test_tensors, test_targets)
        print('Test accuracy: %.4f%%' % (resnet_test_accuracy[1]*100))

        22688/22688 [==============================] - 95s 4ms/step
        Test accuracy: 96.9191%
```

Figure 50 Base ResNet50 Accuracy

The training and test accuracy were printed in Figure 50.

```
In [82]: y_pred = model_resnet.predict(test_tensors)

         fig = plt.figure(figsize=(16,12))
         for i, idx in enumerate(np.random.choice(test_tensors.shape[0], size=16, replace=False)):
             ax = fig.add_subplot(4,4, i + 1, xticks=[], yticks=[])
             ax.imshow(np.squeeze(test_tensors[idx]))
             pred_idx = np.argmax(y_pred[idx])
             true_idx = np.argmax(test_targets[idx])
             ax.set_title("P:{}, A:{}".format(test_labels[pred_idx], test_labels[true_idx]),
                          color=("green" if pred_idx == true_idx else "red"))

         fig.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/ResNet_Prediction-1.jpg')
```

Figure 51 Base ResNet50 Prediction on Random 16 Images

To verify the test accuracy the random 16 images were used for predictions and the results were visualized Figure 51.

```
In [ ]: model = load_model(resnet_weight_path)

        #Confution Matrix and Classification Report
        Y_pred = model.predict_generator(test_generator, test_generator.n//test_generator.batch_size)
        y_pred = np.argmax(Y_pred, axis=1)

        target_names = list(class_labels.values())

        plt.figure(figsize=(30,30))
        cnf_matrix = confusion_matrix(test_generator.classes, y_pred)

        plt.imshow(cnf_matrix, interpolation='nearest')
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        _ = plt.xticks(tick_marks, classes, rotation=90)
        _ = plt.yticks(tick_marks, classes)
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/ResNet_CM-1.jpg')
```

Figure 52 Base ResNet50 Confusion Matrix

The correct and incorrect predictions made by the model were visualized using confusion matrix Figure 52 and Figure 53.

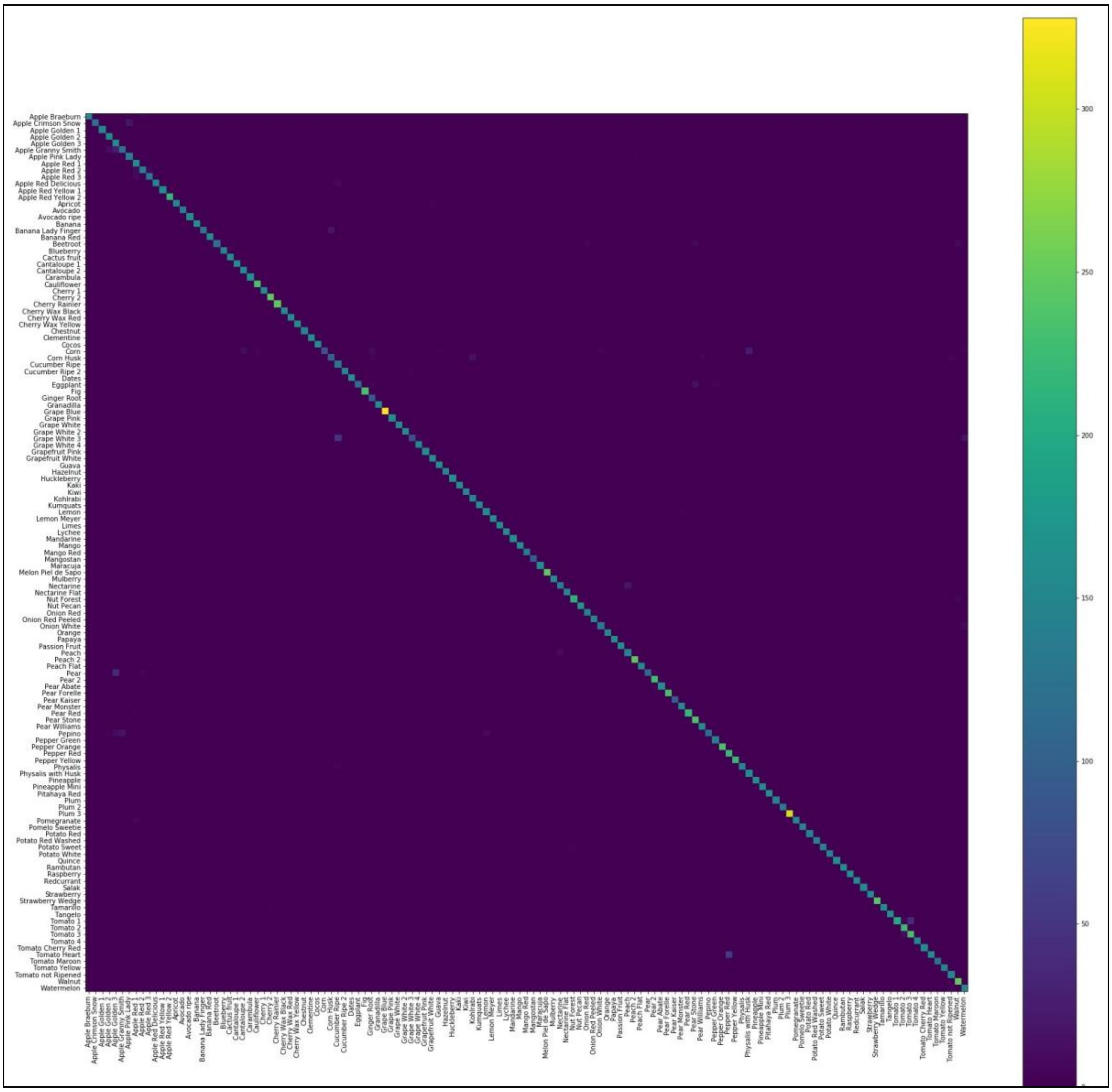

Figure 53 Base ResNet50 Confusion Matrix

3.4.6 ResNet50 with Augmentation

The augmented ResNet50 model was trained in Figure 54.

ResNet50 with augmentation

```
In [ ]: def aug_resnet50():
            model_resnet50_conv = ResNet50(weights='imagenet', include_top=False)

            input_model = Input(shape=(img_row,img_height,img_depth))

            output_resnet50_conv = model_resnet50_conv(input_model)

            x = Flatten()(output_resnet50_conv)
            x = Dropout(0.3)(x)
            x = Dense(131, activation='softmax', name='predictions')(x)

            model = Model(input_model, x)

            for layer in model_resnet50_conv.layers[:]:
              layer.trainable = False

            model.compile(optimizer= SGD(lr=1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
            model.summary()
            return model

In [ ]: model_aug_resnet = aug_resnet50()
        aug_resnet_weight_path = 'C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.aug_resnet_SDG.hdf5'

        C:\Users\Shubham\anaconda3\lib\site-packages\keras_applications\resnet50.py:265: UserWarning: The output shape of `ResNet50(inc
        lude_top=False)` has been changed since Keras 2.2.0.
          warnings.warn('The output shape of `ResNet50(include_top=False)` '

        Model: "model_7"
        _________________________________________________________________
        Layer (type)                 Output Shape              Param #
        =================================================================
        input_14 (InputLayer)        (None, 100, 100, 3)       0
```

Figure 54 ResNet50 with Augmentation Configuration

```
[ ]: from keras.callbacks import ModelCheckpoint

     start = time.time()

     checkpoint = ModelCheckpoint(aug_resnet_weight_path,
                                  monitor="val_loss",
                                  mode="min",
                                  save_best_only = True,
                                  verbose=1)

     callbacks = [checkpoint]

     resnet_aug = model_aug_resnet.fit(
         train_generator,
         steps_per_epoch =train_step,
         epochs = epochs,
         callbacks = callbacks,
         validation_data = validation_generator,
         validation_steps = val_step)

     end = time.time()

     Epoch 1/5
     2964/2964 [==============================] - 297s 100ms/step - loss: 1.1354 - accuracy: 0.7179 - val_loss: 0.0271 - val_accurac
     y: 0.8512

     Epoch 00001: val_loss improved from inf to 0.02706, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weights.
     aug_resnet_SDG.hdf5
     Epoch 2/5
     2964/2964 [==============================] - 301s 102ms/step - loss: 0.3488 - accuracy: 0.8948 - val_loss: 0.0014 - val_accurac
     y: 0.8949

     Epoch 00002: val_loss improved from 0.02706 to 0.00144, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weig
     hts.aug_resnet_SDG.hdf5
     Epoch 3/5
     2964/2964 [==============================] - 300s 101ms/step - loss: 0.2457 - accuracy: 0.9248 - val_loss: 3.4270e-05 - val_acc
     uracy: 0.9069

     Epoch 00003: val_loss improved from 0.00144 to 0.00003, saving model to C:/Users/Shubham/Downloads/NCI/Thesis/saved-models/weig
     hts.aug_resnet_SDG.hdf5
```

Figure 55 ResNet50 with Augmentation Model Training

The augmented ResNet50 model was trained in Figure 55.

```
In [ ]: resnet_aug_tt= end-start
        print('Time taken by the model to run: {}'.format(resnet_aug_tt))

        Time taken by the model to run: 1492.1311955451965

In [ ]: print(resnet_aug.history.keys())

        dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])

In [ ]: # summarize history for accuracy
        plt.plot(resnet_aug.history['accuracy'])
        plt.plot(resnet_aug.history['val_accuracy'])
        plt.title('ResNet with Augmentation - Model Accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/ResNet_AUG_Graph-1-1.jpg')
        plt.show()

        # summarize history for loss
        plt.plot(resnet_aug.history['loss'])
        plt.plot(resnet_aug.history['val_loss'])
        plt.title('ResNet with Augmentation - Model Loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/ResNet_AUG_Graph-2-1.jpg')
        plt.show()
```

Figure 56 Computational Time, and Accuracy and Loss Plots

The computational time required, and accuracy and loss were printed in Figure 56.

```
In [ ]: resnet_aug_train_accuracy=resnet_aug.model.evaluate(train_generator)
        print('Train accuracy: %.4f%%' % (resnet_aug_train_accuracy[1]*100))

        2965/2965 [==============================] - 182s 61ms/step
        Train accuracy: 97.2842%

In [ ]: resnet_aug_test_accuracy=resnet_aug.model.evaluate(test_generator)
        print('Test accuracy: %.4f%%' % (resnet_aug_test_accuracy[1]*100))

        1418/1418 [==============================] - 75s 53ms/step
        Test accuracy: 95.2177%
```

Figure 57 ResNet50 with Augmentation Accuracy

The training and test accuracy were printed in Figure 57.

```
In [ ]: y_pred = model_aug_resnet.predict(test_tensors)

        fig = plt.figure(figsize=(16,12))
        for i, idx in enumerate(np.random.choice(test_tensors.shape[0], size=16, replace=False)):
            ax = fig.add_subplot(4,4, i + 1, xticks=[], yticks=[])
            ax.imshow(np.squeeze(test_tensors[idx]))
            pred_idx = np.argmax(y_pred[idx])
            true_idx = np.argmax(test_targets[idx])
            ax.set_title("P:{}, A:{}".format(test_labels[pred_idx], test_labels[true_idx]),
                         color=("green" if pred_idx == true_idx else "red"))

        fig.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/ResNet_AUG_Prediction-1.jpg')
```

Figure 58 ResNet50 with Augmentation Prediction on Random 16 Images

To verify the test accuracy the random 16 images were used for predictions and the results were visualized Figure 58.

```
In [ ]: model = load_model(aug_resnet_weight_path)

        #Confution Matrix and Classification Report
        Y_pred = model.predict_generator(test_generator, test_generator.n//test_generator.batch_size)
        y_pred = np.argmax(Y_pred, axis=1)

        target_names = list(class_labels.values())

        plt.figure(figsize=(30,30))
        cnf_matrix = confusion_matrix(test_generator.classes, y_pred)

        plt.imshow(cnf_matrix, interpolation='nearest')
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        _ = plt.xticks(tick_marks, classes, rotation=90)
        _ = plt.yticks(tick_marks, classes)
        plt.savefig('C:/Users/Shubham/Downloads/NCI/Thesis/Report Pictures/ResNet_AUG_CM-1.jpg')
```

Figure 59 ResNet50 with Augmentation Confusion Matrix

The correct and incorrect predictions made by the model were visualized using confusion matrix Figure 59 and Figure 60.

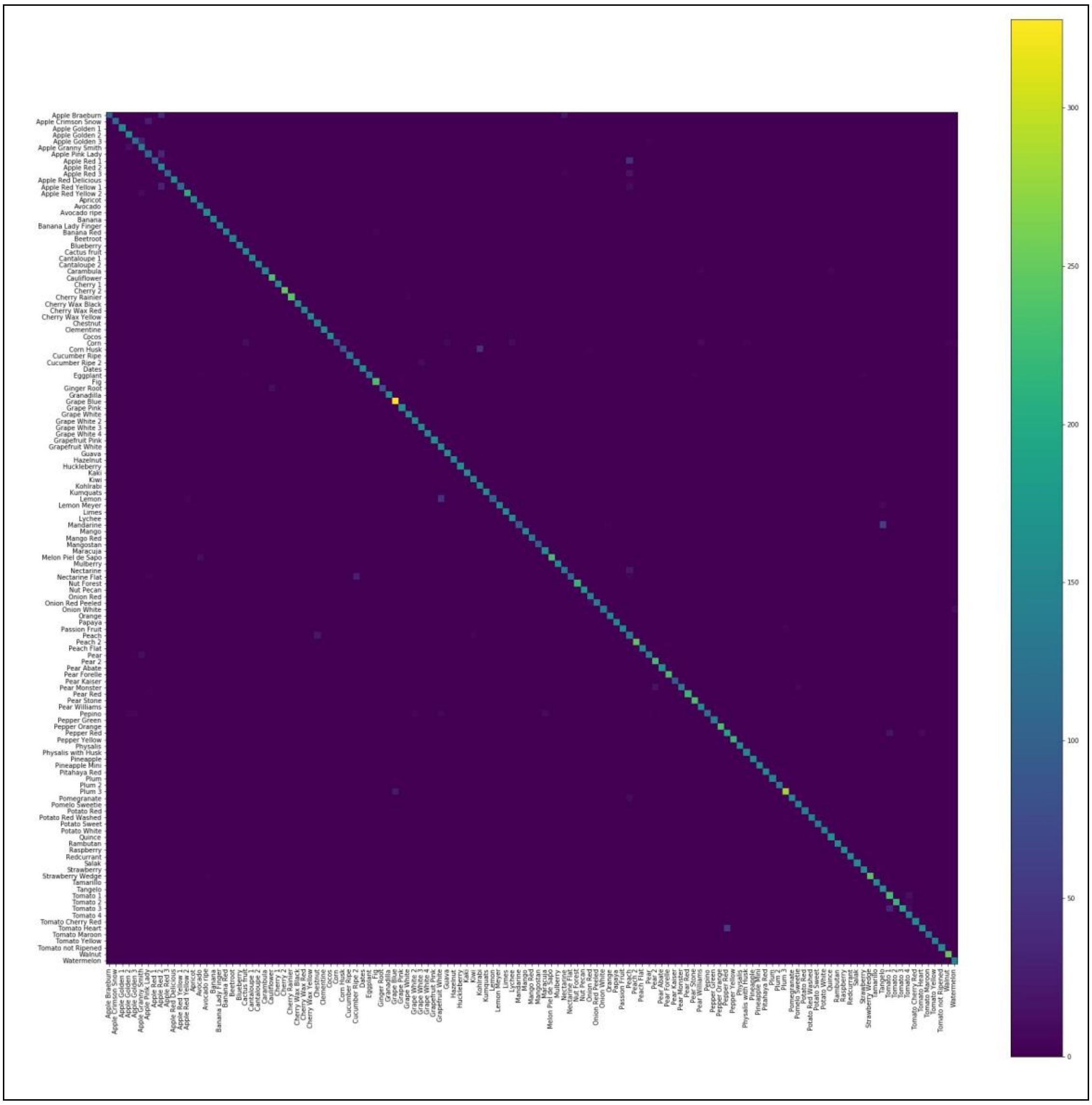

Figure 60 ResNet50 with Augmentation Confusion Matrix

	A	B	C	D	E
1		Computational Time			
2	Models	Base	Augmented		
3	CNN	11.8	20.05		
4	VGG16	26.28	21.83		
5	ResNet50	29	24.87		
6					
7					
8					

Figure 61 Visualising Computation Time

Microsoft Excel was used to visualize and compare the computation time taken by each model Figure 61.

References

Kang, H. and Chen, C. (2020) ‘Fast implementation of real-time fruit detection in apple orchards using deep learning’, Computers and Electronics in Agriculture, 168, p.105108, ScienceDirect. doi: 10.1016/j.compag.2019.105108

Rojas-Aranda J.L., Nunez-Varela J.I., Cuevas-Tello J.C., Rangel-Ramirez G. (2020) ‘Fruit Classification for Retail Stores Using Deep Learning’, in *2020 Pattern Recognition*, MCPR 2020. Lecture Notes in Computer Science, vol 12088. Springer, Cham. https://doi.org/10.1007/978-3-030-49076-8_1.