National College of Ireland

Identifying Diseases In Mulberry Leaves That Affects Silk Production: A Deep Learning Approach

MSc Research Project
Data Analytics

Himanshu Ashok Duragkar
Student ID: X20210639

School of Computing
National College of Ireland

Supervisor: Qurrat Ul Ain

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Himanshu Ashok Duragkar
Student ID:	X20210639
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Qurrat Ul Ain
Submission Due Date:	15/12/2022
Project Title:	Identifying Diseases In Mulberry Leaves That Affects Silk Production: A Deep Learning Approach
Word Count:	557
Page Count:	9

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 speci ed in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	29th January 2023

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 o ce must be placed into the assignment box located outside the o ce.

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

Identifying Diseases In Mulberry Leaves That Affects Silk Production: A Deep Learning Approach

Himanshu Ashok Duragkar
X20210639

1 Hardware Requirements

The hardware speci cations used for this project were a 64-bit Windows 10 operating system. Intel 11th gen core with Intel iRISx graphics card and 8 GB of RAM (Figure 1)

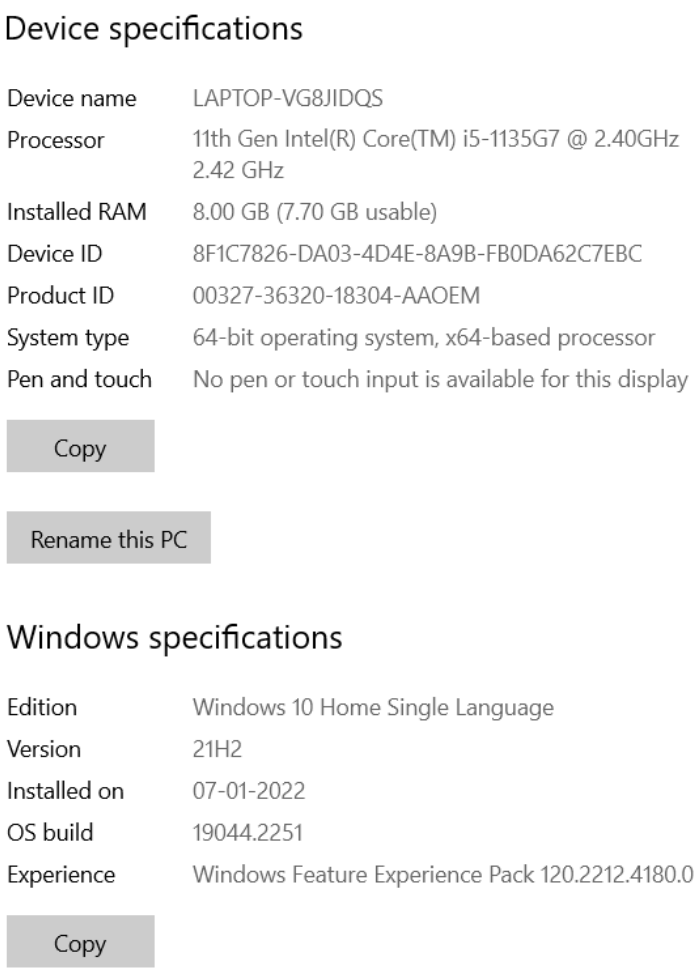

Figure 1: Hardware Requirements

2 Software Requirements

We wrote code in Python during the research. The Anaconda navigator platform's Jupyter Lab was used to develop Python codes. Because the system is 64-bit compatible and a 64-bit compatible application is being used, the first step is to install the Anaconda application (Figure 2).

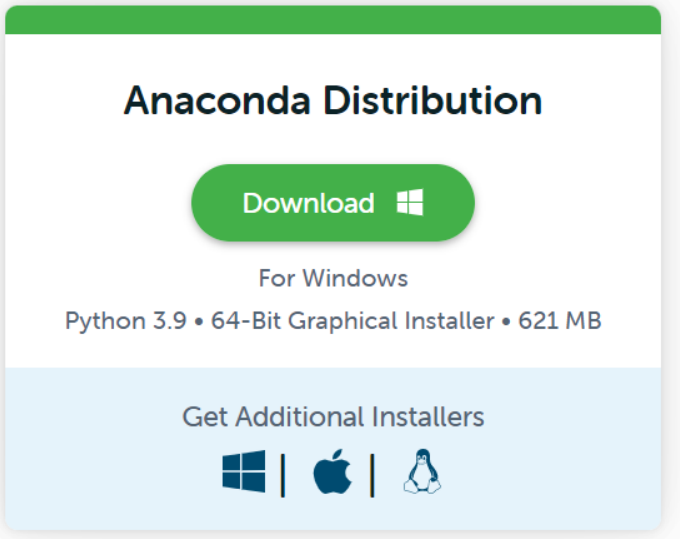

Figure 2: Anaconda navigator specification

After the installation of Anaconda navigator, Jupyter notebook can be launched from the navigator's home page itself (Figure 3).

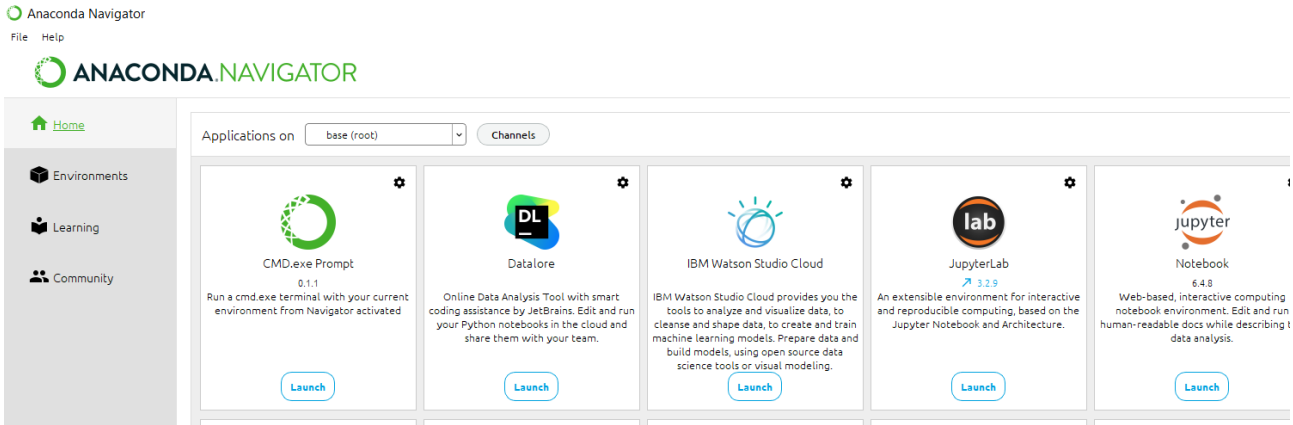

Figure 3: Anaconda navigator overview

3 Libraries required for Python

The following is a list of Python libraries that must be installed using the pip command at the Python environment's command prompt.

- Tensor ow.
- Keras.
- numpy.
- matplotlib.
- pandas.
- future.
- sklearn.
- itertools.

4 Dataset Description

- The mulberry leaves data set can be downloaded from the Multi-agent Intelligent Simulation Laboratory Research Unit website with help of the URL : `http://misl.it.msu.ac.th/?page_id=225`.
- The images come from various cultivators all over the world.

5 Data pre-processing

- Extract the zip le. divide the data into train, test and validate part into their respective folder each. For data to put into the model the images should be categorical i.e divided into two folder infected and non infected.
- Before the data is given input to the model, it is pre-processed with di erent image augmentation techniques (Figure 4).

```
def DataGenerator(training_batch, validation_batch, IMAGE_SIZE):
    generator = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   rescale=1./255,
                                   rotation_range=10,
                                   horizontal_flip=True,
                                   vertical_flip=True)

    generator.mean=np.array([103.939, 116.779, 123.68],dtype=np.float32).reshape(1,1,3)

    training_gen = generator.flow_from_directory("dataset/train data/",
                                            target_size=(IMAGE_SIZE, IMAGE_SIZE),
                                            color_mode='rgb',
                                            class_mode='categorical', #categorical because two of two present classes
                                                                      #infected and noninfected
                                            batch_size=training_batch)


    validation_gen = generator.flow_from_directory("dataset/val/",
                                           target_size=(IMAGE_SIZE, IMAGE_SIZE),
                                           color_mode='rgb',
                                           class_mode='categorical',
                                           batch_size=validation_batch)

    generator = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   rescale=1./255)

    generator.mean=np.array([103.939, 116.779, 123.68],dtype=np.float32).reshape(1,1,3)

    testing_gen = generator.flow_from_directory("dataset/test data/",
                                            target_size=(IMAGE_SIZE, IMAGE_SIZE),
                                            color_mode='rgb',
                                            class_mode='categorical',
                                            shuffle=False)

    return training_gen, validation_gen, testing_gen
```

Figure 4: Image data augmentation

6 Model Preparation

Two model is implemented in the proposed research:

6.1 VGG16 model for feature extraction and transfer learning

```
input_image = Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))

base = VGG16(include_top=False, weights='imagenet', input_tensor=input_image)

base.summary()
```

Figure 5: VGG16 model

6.2 Capsule neural network for image classification

This section consist of two parts. The first part is for creating the model and the second for model fitting.

```
class Capsule(Layer):



    def __init__(self,
                 num_capsule,
                 dim_capsule,
                 routings=3,
                 share_weights=True,
                 activation='squash',
                 **kwargs):
        super(Capsule, self).__init__(**kwargs)
        self.num_capsule = num_capsule
        self.dim_capsule = dim_capsule
        self.routings = routings
        self.share_weights = share_weights
        if activation == 'squash':
            self.activation = squash
        else:
            self.activation = activations.get(activation)

    def build(self, input_shape):
        input_dim_capsule = input_shape[-1]
        if self.share_weights:
            self.kernel = self.add_weight(
                name='capsule_kernel',
                shape=(1, input_dim_capsule,
                       self.num_capsule * self.dim_capsule),
                initializer='glorot_uniform',
                trainable=True)
        else:
            input_num_capsule = input_shape[-2]
            self.kernel = self.add_weight(
                name='capsule_kernel',
                shape=(input_num_capsule, input_dim_capsule,
                       self.num_capsule * self.dim_capsule),
                initializer='glorot_uniform',
                trainable=True)

    def call(self, inputs):


        if self.share_weights:
            hat_inputs = K.conv1d(inputs, self.kernel)
        else:
            hat_inputs = K.local_conv1d(inputs, self.kernel, [1], [1])
```

Figure 6: Capsule Model Creation

7 Evaluation of Implemented Methods

Initially, three experiments were run on the model. Experiment 1 to check weather the model is built properly or not (Figure 8)

```
#experiment 1 to chcek if the model is working correctly or not

model.fit_generator(train,
                    epochs=1,
                    validation_data=val,
                    validation_steps = len(val.classes)//validation_batch,
                    steps_per_epoch=120)

loss, acc = model.evaluate_generator(test, len(test))

#priniting data loss and accuracy
print ("Data Loss: {}".format(loss))
print ("Accuracy: {0:.2f} %".format(acc * 100))


test.reset()
```

Figure 7: Capsule Model t

```
#experiment 1 to chcek if the model is working correctly or not

model.fit_generator(train,
                    epochs=1,
                    validation_data=val,
                    validation_steps = len(val.classes)//validation_batch,
                    steps_per_epoch=120)

loss, acc = model.evaluate_generator(test, len(test))

#priniting data loss and accuracy
print ("Data Loss: {}".format(loss))
print ("Accuracy: {0:.2f} %".format(acc * 100))


test.reset()
```

Figure 8: Experiment 1

The second experiment increased the number of epochs to 10, with the same number of iterations per epoch (Figure 9).

```
#experiment 2 with increase in number of epochs

history=model.fit_generator(train,
                    epochs=10,
                    validation_data=val,
                    validation_steps = len(val.classes)//validation_batch,
                    steps_per_epoch=120)

loss, acc = model.evaluate_generator(test, len(test))


#priniting data loss and accuracy
print ("Data Loss: {}".format(loss))
print ("Accuracy: {0:.2f} %".format(acc * 100))


test.reset()
```

Figure 9: Experiment 2

To save a model or weights at some interval, a check point is created so the model or weights can be loaded later to continue the training from the saved state.

```
#creating a check point for exp2 so that model can be reused for further traing in the next experiment.
#used in conjunction with training using model.fit() to save a model or weights at some interval,
#so the model or weights can be loaded later to continue the training from the state saved.

lr=1e-4

checkpoint = ModelCheckpoint("weights.h5",
                             monitor='val_loss',
                             verbose=1,
                             save_best_only=True,
                             save_weights_only=False,
                             mode='min')

early = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode='min', restore_best_weights=True)

callback_list = [checkpoint, early]
```

Figure 10: Checkpoint

The third experiment increased the number of epochs to 15 with the same number of iterations per epoch, i.e., 120. (Figure 11)

```
#Experiment 3 with initailizing number of epochs to 15

model.compile(loss=margin_loss, optimizer=SGD(lr=lr, momentum=0.9), metrics=['accuracy'])

history = model.fit_generator(train,
                    epochs=15,
                    validation_data=val,
                    validation_steps = len(val.classes)//validation_batch,
                    steps_per_epoch=120,
                    callbacks=callback_list)

loss, acc = model.evaluate_generator(test, len(test))


print ("Data Loss: {}".format(loss))
print ("Accuracy: {0:.2f} %".format(acc * 100))


test.reset()
```

Figure 11: Experiment 3

The evaluation of the implemented model is addressed in this section for each experiment. In the initial stage, the accuracy and loss graph is plotted as a performance metric (Figure 12). The next classi cation report is printed for the implemented model (Figure 13). The last evaluation is done using a confusion matrix, where the model is evaluated using all the common metrics (Figure 14).

```
#Plotting data accuracy and loss with respect to the epochs

print(history.history.keys())

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Figure 12: Steps to display Accuracy and loss graphs

```
#Plotting the matrix

from sklearn.metrics import classification_report, confusion_matrix
Y_pred =  model.predict_generator(test)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix ')

cm = confusion_matrix(test.classes, y_pred)
plot_confusion_matrix(cm, classes, title='Confusion Matrix in terms of percentage')
plot_confusion_matrix(cm, classes,False, title='Confusion Matrix in terms of quantity')
```

Figure 13: Steps to display Confusion matrix

```
#plotting Classification report for exp 2

    from sklearn.metrics import classification_report
```

```
print("Model : Capsule neural netwrok")
print(classification_report(test.classes, y_pred))
```

Figure 14: Steps to display classi cation report

To check the results, an array is created that stores the actual results and the predicted results. A comparison is performed to check the classi ed results. (Figure 15) .