UNIVERSITI TEKNOLOGI MARA

SEMESTER	**:**	**MARCH – AUGUST 2023**
COURSE	**:**	**IMAGE PROCESSING**
COURSE CODE	**:**	**CSC566**
PROJECT TITLE	**:**	**Tomato Leaf Disease Classification using Convolutional Neural Network (CNN)**

Group:	Group 1
Name, Student ID	1. Muhammad Nur Azree Bin Mohamad Nasir, 2021478032
	2. Awang Mohd Azrulazhim Bin Awang Zainol, 2021498172
	3. Nur Diana Bt Zulkefli, 2021889318
Class:	CSC2306C
Lecturer Name:	Ahmad Fikri Bin Anuar

ASSESSMENT

ITEMS	FULL MARKS	MARKS
PRESENTATION: 1. Project completeness and complexity 2. Content of presentation 3. Delivery skills	**30**	
REPORT: 1. Introduction 2. Objectives 3. Data Collection 3.1 Training dataset 3.2 Testing dataset 4. Flowchart 5. Model architecture 5.1 Input 5.2 Process 5.3 Output 5.4 Sample input and output 6. Source Code 7. Test and Evaluation 7.1 Accuracy rate 7.2 Learning rate 7.3 Error rate 7.4 Recall 7.5 Precision 8. Conclusion References	**70**	
PENALTY		
TOTAL MARKS	**100**	

TASK: TOMATO LEAF DISEASE CLASSIFICATION

1. Please refer to this link for image dataset:
   https://www.kaggle.com/datasets/cookiefinder/tomato-disease-multiple-sources

   Download the image dataset folder and choose at least **THREE (3)** different tomato leaf disease classes.

2. The example images are as follows:

Bacterial Spot

Leaf Mold

Spider Mites

3. Notice that each image contains different information, which can be utilized in order to classify the leaves into the three classification of diseases.

4. Perform any deep learning method (using Python) to classify the tomato leaf disease. The sample of steps could be followed from these videos (or any other resources):

   a. https://www.youtube.com/watch?v=iGWbqhdjf2s
   b. https://www.youtube.com/watch?v=jztwpsIzEGc
   c. https://www.youtube.com/watch?v=J1jhfAw5Uvo

5. Then, evaluate the accuracy rate for training and testing dataset.

6. Write a report by using the **INSTRUCTIONS** below.

INSTRUCTIONS

1. Use the project cover sheet as provided (**page 1 and 2**).

2. This is a group project. Each group consists of **3-5 members**.

3. General format:
    a. Font: Arial (for report) and Courier New (for source code)
    b. Font size: 11
    c. Table, Figure: Arial, 9
    d. Spacing: 1.15 (for report) and 1.0 (source code)

4. Special remarks:
    a. No marks will be given for late submission
    b. Any copied project (with NO EXCEPTION) will be given 0 (zero) mark.

5. Link for submission (p/s: please create your own group folder in the specific class folder): https://drive.google.com/drive/folders/1S6i1H6ALqyiMhiAttY3n1JsD67YWAGZz?usp=sharing

6. The submission should include:
    a. Recorded video presentation (for report and system demo)
    b. Report (.pdf file)
    c. System (.zip file)
    d. Dataset

7. Due date: **Wednesday, 12th July 2023, 6pm**.

ASSESSMENT INFORMATION

PRESENTATION (30 MARKS)

Presentation marks will be given based on the following criteria:

1. Project completeness and complexity
2. Content of presentation
3. Delivery skills

REPORT (70 MARKS)

For this mini project, you are required to experiment and write a program on the given topics using Python. Your project report should follow the requirements based on the given format. Include a report based on the format below:

1) Project report cover
As provided in page 1 – 2.

2) Table of content
As provided in page 2 – 3.

3) Introduction
Describe about the given project which reflect to your project title.

4) Objectives
Describe about the objectives of the project.

5) Data Collection
Describe and specify the total number of training and testing images that have been used, and the sources conducted for experimental data. The minimum dataset for training and testing images is **1000 images**.

You can use the ratio of training : testing = 70 : 30 or 80 : 20 or 90 : 10.
How to choose the best ratio?
It is based on the best results produced from your experiment based on the above ratio. You can also refer to papers and journals.

6) Flowchart
Explain and describe the flowchart or process flow of the methodology used in the experiment. The selected methods are based on the technique used in your experiment. Use and experiment the same flowchart and methods (source code) for all images.

7) Methods
Explain and describe all the methods involved during the project.

Example:

Machine Learning or Deep Learning Architecture

Explain all the processes involved in this architecture for each layer during your experiments.

a. Input
b. Feature extraction layer
c. Classification layer
d. Output
e. Sample input and output

8) Source Code

Write source code with proper comments.

9) Test and Evaluation

a. Accuracy rate
b. Learning rate
c. Error rate
d. Recall
e. Precision

The submitted report must be included all the processes (original image, process 1 until the last process, output) involved. The source code should work for all image datasets. All results must construct in the table. Discuss all the findings of the learning rate, accuracy rate and error rate.

Summarize and conclude all the findings from the experiments. Then, describe the future works.

10) References

Minimum references are 15 conference or journal papers.

CSC566: IMAGE PROCESSING
GROUP PROJECT

ASSESSMENT RUBRIC

RUBRIC	EXCELLENT (8 – 10)	GOOD (6 – 7)	SATISFACTORY (5)	POOR (1 – 4)	0
PRESENTATION (30 MARKS) 1. Project completeness and complexity 2. Content of presentation 3. Delivery skills	▪ High level of completeness and complexity achieved in solving the problem. ▪ Presenter has a smooth presentation flow and provides good explanations and/or elaboration, used time wisely.	▪ Moderate level of completeness and complexity achieved in solving the problem. ▪ Presenter provides explanations and/or elaboration, used time wisely.	▪ Fair level of completeness and complexity achieved in solving the problem. ▪ Presenter provides explanations and/or insufficient elaboration and use of time.	▪ Poor level of completeness and complexity achieved in solving the problem. ▪ There is no presentation flow. Goes over time limit or does not fully cover the topics.	No attempt.
REPORT (70 MARKS) 1. Introduction 2. Objectives and Project Significance 3. Data Collection 4. Flowchart 5. Model architecture 6. Source Code 7. Test and Evaluation 8. Conclusion References	▪ Working title that clearly reflects the project. ▪ Objectives - highly reflect the elements: specific, measurable, achievable, realistic and timeliness. ▪ Highly reflects the following elements: approach, methods, design and deliverables. ▪ Comprehensive examination and	▪ Working title that reflects the project. ▪ Objectives - clearly reflect the elements: specific, measurable, achievable, realistic and timeliness. ▪ Clearly reflects the following elements: approach, methods, design and deliverables. ▪ Detailed examination and explanation of the interaction	▪ Appropriate working title that reflects the project. ▪ Objectives - adequately reflect the elements: specific, measurable, achievable, realistic and timeliness. ▪ Adequately reflects the following elements: approach, methods, design and deliverables.	▪ Inappropriate working title that reflects the project. ▪ Objectives – does not reflect the elements: specific, measurable, achievable, realistic and timeliness. ▪ Poorly reflects the following elements: approach, methods, design and deliverables.	No attempt.

	explanation of the interaction between parameters and system function in the development. ▪ Comprehensive discussion of the result is articulated in an excellent manner.	between parameters and system function in the development. ▪ Detailed discussion of the result is articulated in a well manner.	▪ Appropriate examination and explanation of the interaction between parameters and system function in the development. ▪ Appropriate discussion of the result is articulated in a good manner.	▪ Incomplete examination and explanation of the interaction between parameters and system function in the development. ▪ Incomplete discussion of the result is articulated.	

ALL THE BEST STUDENTS!!!!!!!

1. INTRODUCTION

Malaysia is a rapidly developing nation, and agriculture was the foundation of the nation's early development. The field is encountering difficulties as a result of industrialization and globalization theories. Additionally, the younger generation needs to be made aware of the importance of agriculture and its necessity. In all areas nowadays, technology is crucial, yet in agriculture, we still rely on some antiquated practices. An incorrect diagnosis of a plant disease causes a significant loss in yield, productivity, cost, and product quality. The key to good cultivation is determining the plant's state. In the past, identification was done manually by skilled individuals, but environmental changes have made prediction more difficult.

2. OBJECTIVES

The objective of this project is:

- To identify the disease of tomato leaf

  This objective focuses on developing a system or approach that can accurately detect the presence of diseases in tomato leaves (Shoaib et al., 2023). According to Panthee & Chen (2010), tomato plants are susceptible to various diseases, such as early blight, leaf mold, bacterial spot, powdery mildew, and septoria leaf spot. The identification process involves using machine learning algorithms to analyze images of the tomato leaves and detect signs of disease, such as discoloration, spots, lesions, or other visual symptoms associated with specific diseases.

- To classify the disease of the tomato leaves based on the condition of the tomato leaves

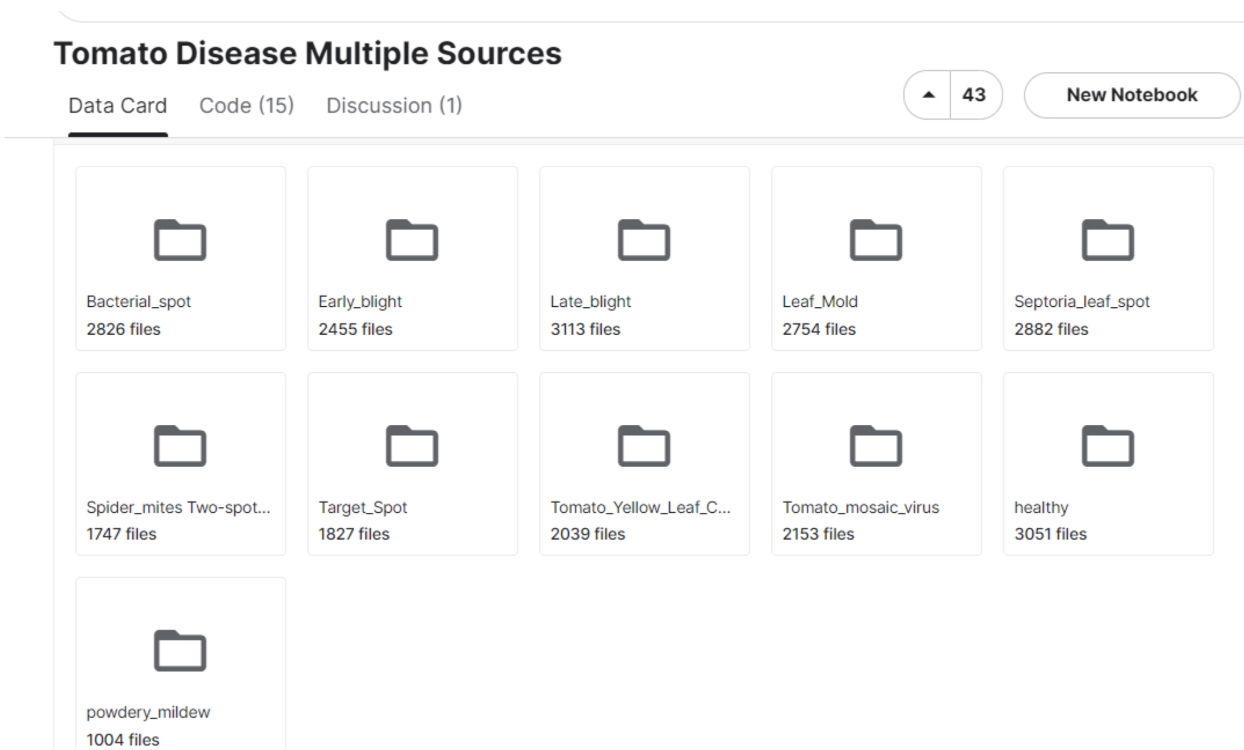
  Once the system has successfully identified that a tomato leaf is diseased, this objective aims to further categorize the specific disease affecting the leaf. Different diseases exhibit distinct symptoms and patterns on the leaves, and accurate classification of tomato leaf diseases is important for maintaining plant health and yields, and preventing the spread of diseases (Trivedi et al., 2021).

- To evaluate the accuracy of the approach method

  This objective involves assessing the performance and reliability of the disease identification and classification method or approach. Evaluation metrics, such as accuracy, precision, recall, F1-score, and confusion matrix, may be used to measure how well the system can correctly identify the presence of disease and how accurately it can classify the disease type based on the condition of the tomato leaves.

3. DATA COLLECTION

The dataset is obtained from Kaggle named Tomato Disease Multiple Sources by Qasim Khan. With 10 illnesses and 1 healthy class, there are almost 20k photos of tomato leaves. Both lab and in-the-wild situations are used to acquire images.



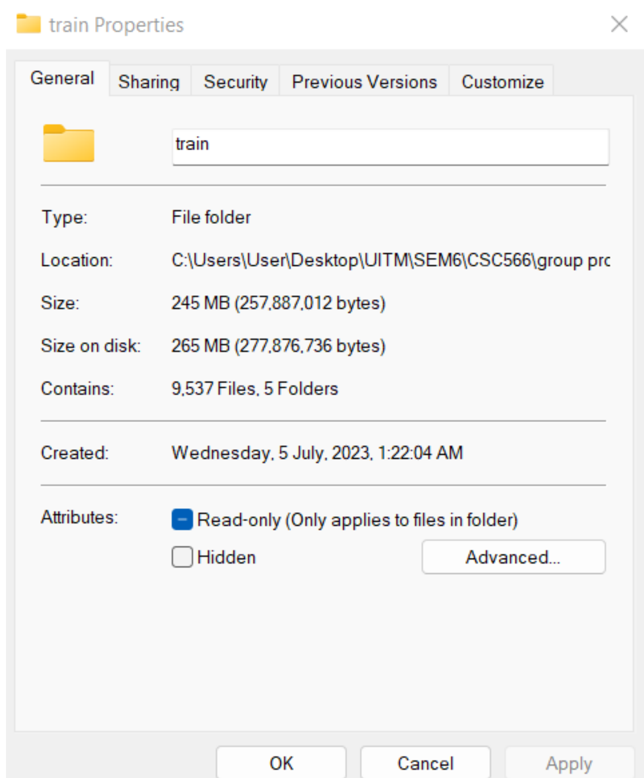
(Khan, 2022)

The objective is to create a simple model that can forecast tomato leaf disease and to use it offline in a mobile application. As the environment evolves, making predictions is getting harder. So, we can identify plant diseases using image processing techniques. In general, we can see disease symptoms on leaves, stems, flowers, etc., thus in this case we utilize leaves to identify diseased plants.

```
Found 11921 files belonging to 5 classes.
Using 9537 files for training.
Found 11921 files belonging to 5 classes.
Using 2384 files for validation.
```

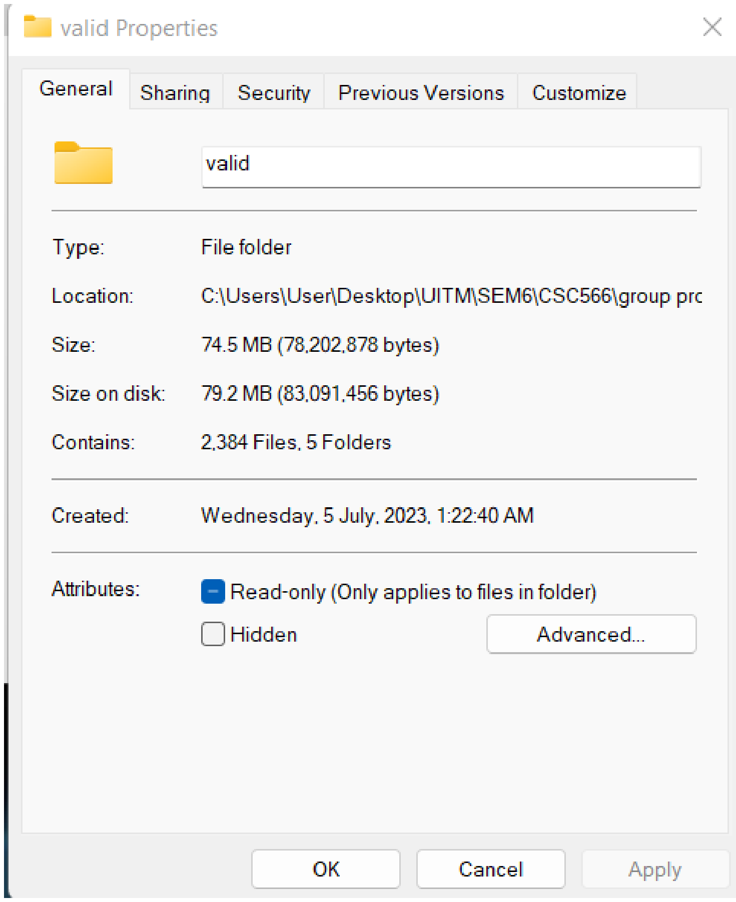
In this project, a total number of 11921 images belong to 5 classes, 9537 are for training while 2384 are for testing. The ratio of training : testing = 80 : 20.

3.1 TRAINING DATASET



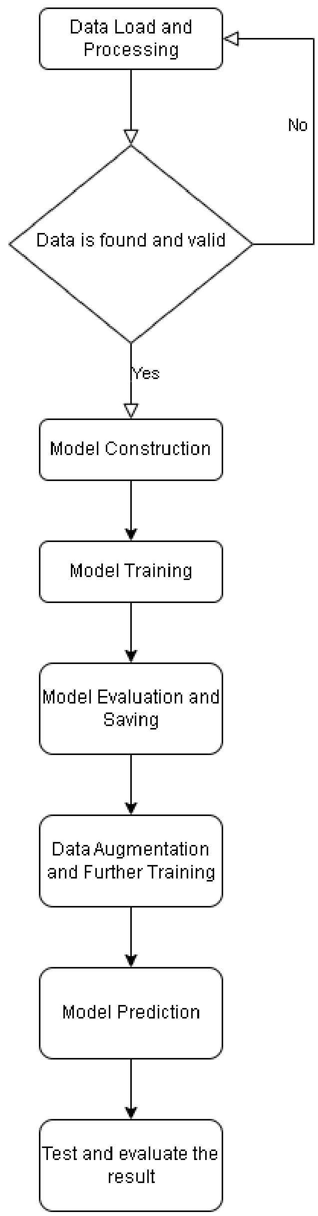
9537 images that are being used for training.

3.2 TESTING DATASET



2384 images that are being used for testing.

4. FLOWCHART



Data Load and Processing:

The experiment begins by specifying the directory, where the image data is located. If the image data is not found, the experiment will not be successful and we have to specify the correct directory for the experiment before proceeding to the next step.

Model Construction:

The CNN model is defined using the Sequential model from Keras. The model architecture is then constructed with Conv2D, MaxPooling2D, Flatten, and Dense layers. After that, the model is compiled with Adam optimizer, sparse categorical cross-entropy loss function, and accuracy metric.

Model Training:

The training dataset is passed to the model along with the validation dataset. The model is trained for a specified number of epochs, optimizing the model parameters to minimize the loss function. During training, the model performance and loss values are monitored and displayed.

Model Evaluation and Saving:

After training, the model's performance is evaluated using the validation dataset. The trained model is saved to the disk using the model.save method, creating a directory named 'saved_model' and saving the model in it.

Data Augmentation and Further Training:

Data augmentation techniques are applied to the training dataset. Random transformations are applied to augment the training images, increasing their diversity. A new CNN model is constructed, incorporating the data augmentation techniques. The model is compiled and trained again using the augmented training dataset.

Model Prediction:

An example image is loaded using the tf.keras.utils.load_img method. The image is preprocessed by resizing it to the specified input size. The trained model (new_model) is used to make predictions on the preprocessed image. The predicted class and the corresponding confidence score are displayed.

Test and evaluate the result:

Record and evaluate the result that we get from the model that we train.

5. METHOD ARCHITECTURE

Convolutional Neural Network (CNN)
Definition

CNN stands for Convolutional Neural Network. It is a specialized type of deep learning neural network designed for processing and analyzing visual data, such as images and videos. CNNs are widely used for various computer vision tasks, including image classification, object detection, segmentation, and more. CNN is designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers (Yamashita et al., 2018).

Fundamental elements

Convolutional Layers

The main building block of CNN is the convolutional layer (Dertat, 2017). These layers apply convolutional operations to the input data using a set of learnable filters (also known as kernels). The filters slide over the input data, extracting features by detecting patterns, edges, and textures. The outputs are referred to as feature maps.

Activation Function

An activation function, usually ReLU (Rectified Linear Unit), is applied after each convolution operation. It introduces non-linearity into the network and enables the model to learn complex relationships between input data and features. As a consequence, the usage of ReLU helps to prevent the exponential growth in the computation required to operate the neural network (Baeldung, 2023).

Pooling Layers

Pooling layers downsample the feature maps, reducing their spatial dimensions while retaining essential information. Max pooling is a common technique that retains the maximum value within a pool to capture the most important features.

Fully Connected Layers

After several convolutional and pooling layers, the feature maps are flattened and passed to fully connected layers. These layers perform high-level reasoning and decision-making based on the extracted features. They eventually output the predictions or classifications for the given input.

Softmax Layer

In the final layer of the CNN (often a Dense layer), a softmax activation function is used to produce probability scores for each class in a multi-class classification problem. It helps determine the class with the highest probability as the model's prediction.

5.1 INPUT

Input Layer:

- The input layer receives the image data as input
- The input images are expected to have a specific size defined by img_height and img_width
- The pixel values of the input images are normalized to the range [0, 1] using the layers.Rescaling layer

5.2 PROCESS

Feature Extraction Layer:

- The feature extraction layer consists of multiple layers.Conv2D and layers.MaxPooling2D layers
- Convolutional layers perform convolutions on the input images, extracting local patterns and features
- Activation functions (in this case, 'relu') are applied to introduce non-linearity and increase model expressiveness
- Max pooling layers downsample the feature maps, reducing their spatial dimensions and preserving the most prominent features
- The number of filters and filter sizes are specified for each convolutional layer, determining the complexity and capacity of the learned features

Classification Layer:

- After the feature extraction layers, the feature maps are flattened using the layers.Flatten layer
- Flattening converts the 3-dimensional feature maps into a 1-dimensional vector.

- The flattened features are then passed through one or more fully connected layers.Dense layers
- The dense layers perform computations to map the learned features to the desired output classes
- Activation functions (in this case, 'relu') are applied to the dense layers to introduce non-linearity
- The final dense layer has the number of units equal to the number of output classes, as defined by num_classes

5.3 OUTPUT

Output Layer:

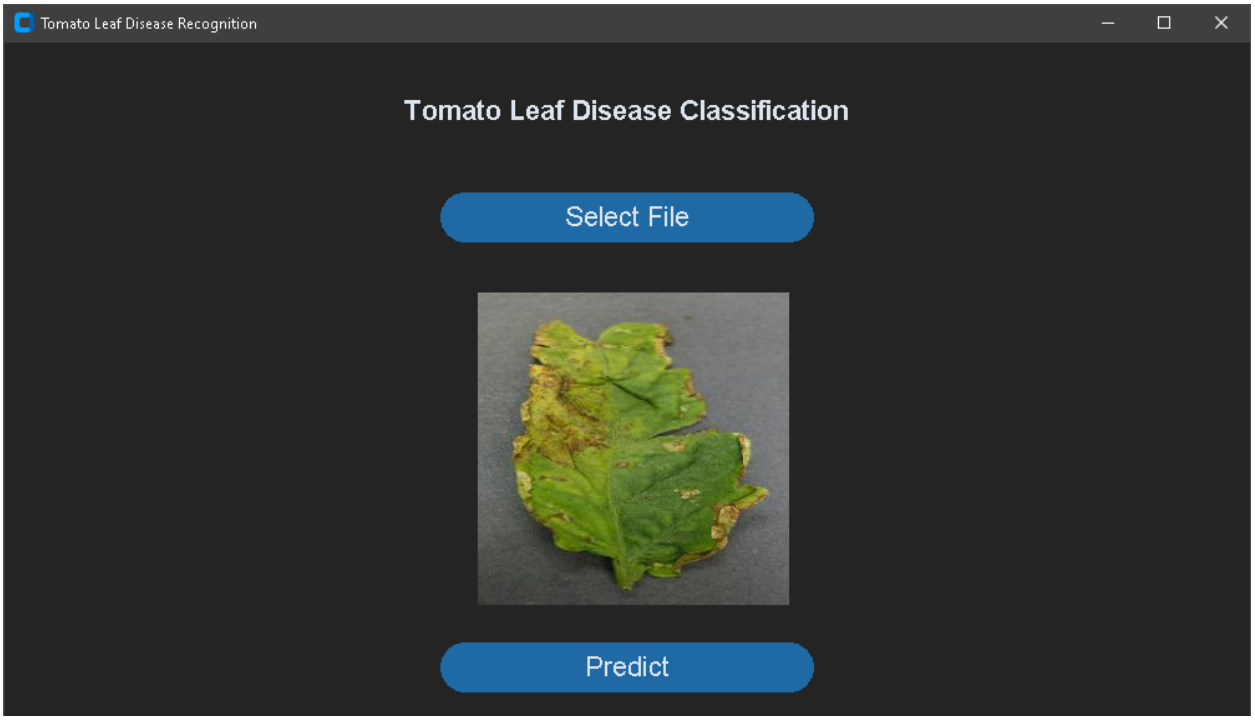
- The output layer is the final layer of the model
- It consists of the last layers.Dense layer with the number of units equal to the number of output classes
- The output layer produces logits (raw predictions) for each class
- These logits are not directly interpretable and need to be converted into probabilities using a softmax activation function

5.4 SAMPLE INPUT AND OUTPUT

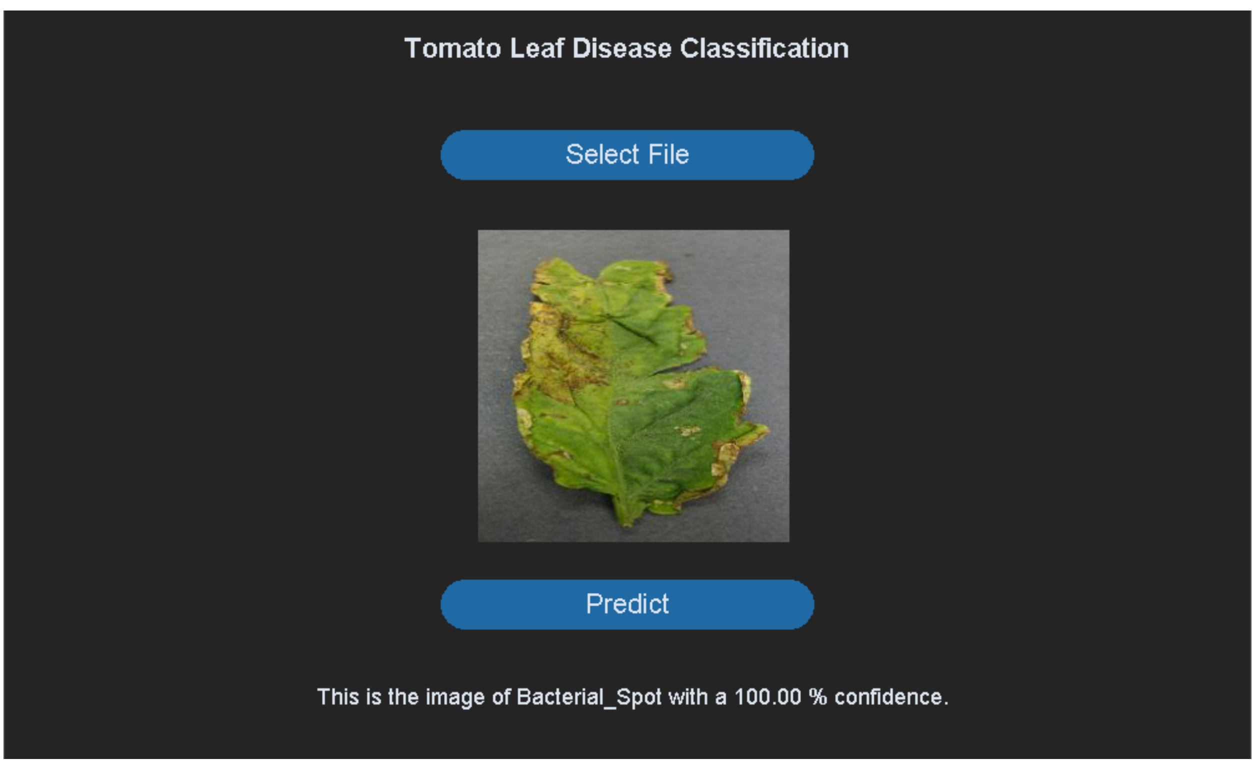
Sample Input and Output:

- For a sample input, an image with dimensions defined by img_height and img_width is provided to the model
- The input image is preprocessed by resizing it to the specified dimensions and normalizing its pixel values
- The model processes the input image through the layers, extracting features and making predictions
- The output of the model is a probability distribution over the classes
- For example, if there are 5 output classes, the output would be a vector of length 5, where each element represents the probability of the corresponding class
- The class with the highest probability can be considered as the predicted class for the given input image

Sample Input



Sample Output



6. SOURCE CODE

Training Model (Tomato_Leaf_Disease.ipynb)

```
# IMPORT THE LIBRARY
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import tempfile
from matplotlib import pyplot as plt

# Load the data
data_dir = "Tomato Leaf/train"

# Create a dataset
batch_size = 32 # batch size
img_height = 180 # image height
img_width = 180 # image width

# Create training dataset from images
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2, #validation split of 0.2
  subset="training",
  seed=123, #random seed for shuffling the dataset
  image_size=(img_height, img_width),
  batch_size=batch_size)

# Create validation dataset from images,
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2, #validation split of 0.2
  subset="validation",
  seed=123, #random seed for shuffling the dataset
```

```
  image_size=(img_height, img_width),
  batch_size=batch_size)

# Retrieve the class names from the train_ds dataset
class_names = train_ds.class_names
print(class_names) # print the class names

# Import the matplotlib library to plot
import matplotlib.pyplot as plt

# Create a figure with a size of 10x10
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1): # iterate over the first
batch of images
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1) # plot the figure
    plt.imshow(images[i].numpy().astype("uint8")) # show the figure
    plt.title(class_names[labels[i]]) # show the names for each
figure
    plt.axis("off")

# Prints the shape of the image and label batches
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break # exit the loop after printing

# Optimizes data loading and training performance
AUTOTUNE = tf.data.AUTOTUNE
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# Standardize the data
normalization_layer = layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x),
y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
```

```
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))

num_classes = 5 # number of classes

# Create a sequential model
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])

# Compile the model
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
, metrics=['accuracy'])

# Model summary
model.summary()

# Define a custom callback to track the learning rate
class LearningRateTracker(tf.keras.callbacks.Callback):
    def __init__(self):
        super(LearningRateTracker, self).__init__()
        self.learning_rates = []  # List to store learning rate
values

    def on_epoch_end(self, epoch, logs=None):
        optimizer = self.model.optimizer
        learning_rate = tf.keras.backend.get_value(optimizer.lr)  #
Get the learning rate
```

```
        self.learning_rates.append(learning_rate)  # Append the 
learning rate to the list

# Train and save the model
epochs = 10  # number of epochs
lr_tracker = LearningRateTracker()  # Create an instance of the 
custom callback
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    callbacks=[lr_tracker]  # Pass the callback to track the 
learning rate
)

!mkdir -p saved_model
model.save('datatrain/saved_model/my_model') # save the model

# Calculate evaluation metrics
train_loss, train_accuracy = model.evaluate(train_ds)
val_loss, val_accuracy = model.evaluate(val_ds)

predictions = np.argmax(model.predict(val_ds), axis=-1)
val_labels = np.concatenate([y for x, y in val_ds], axis=0)

error_rate = 1 - np.mean(predictions == val_labels)
recall = tf.keras.metrics.Recall()(val_labels, predictions).numpy()
precision = tf.keras.metrics.Precision()(val_labels, 
predictions).numpy()

# Print the evaluation metrics
print("Accuracy Rate - Train: {:.4f}, Validation: 
{:.4f}".format(train_accuracy, val_accuracy))
print("Learning Rate: {:.6f}".format(lr_tracker.learning_rates[-1]))
print("Error Rate: {:.4f}".format(error_rate))
print("Recall: {:.4f}".format(recall))
print("Precision: {:.4f}".format(precision))

# Load the saved model and view the summary
```

```
new_model =
tf.keras.models.load_model('datatrain/saved_model/my_model')
new_model.summary()

# Data augmentation using RandomFlip
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)

# Display the figure
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")

# Sequential model and its architecture
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
```

```
])

# Compile the model
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
, metrics=['accuracy'])

# Model summary
model.summary()

# Predict using the model that we train
img = tf.keras.utils.load_img(
   'Tomato Leaf/valid/powdery_mildew/powdery_mildew (10).JPG',
target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

# Print the result
print(
    "This image most likely belongs to {} with a {:.2f} percent
confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

Prediction (Main.py)

```
import customtkinter
from tkinter import filedialog
from PIL import Image
import numpy as np
import tensorflow as tf
from tensorflow import keras

import tensorflow as tf
print(tf.__version__)
```

```
# Create Model and load the saved model that we train
saved_model_dir = "datatrain/saved_model/my_model"
new_model = tf.saved_model.load(saved_model_dir)

batch_size = 32
img_height = 180
img_width = 180

# Specify the class names
class_names = ['Bacterial_Spot', 'Early_blight', 'Leaf_Mold',
'Powdery_mildew', 'Septoria_leaf_spot']

# Function for browse and display the image
def searchImage():
    global filename
    filename =
filedialog.askopenfilename(initialdir="/dataset",title="Select
Image",
    filetypes= (("JPG File","*.jpg*"),("PNG file","*.png"), ("All
Files", "*.*")))

    my_image =
customtkinter.CTkImage(light_image=Image.open(filename),
                                  size=(250, 250))
    image_label.configure(image=my_image)
    return filename

# Function to run the app to recognize the tomato leaf
def runApp():
    global filename
    img = keras.preprocessing.image.load_img(
    filename, target_size=(img_height, img_width))
    img_array = keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) # Create a batch
    input_tensor = tf.convert_to_tensor(img_array, dtype=tf.float32)
    predictions = new_model(input_tensor)
    score = tf.nn.softmax(predictions[0])
    Output = (
```

```
    "This is the image of {} with a {:.2f} %"" confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score)))
    print(Output)
    imagePredict.configure(text=Output)

# End of All Function

app = customtkinter.CTk()
customtkinter.set_appearance_mode("system")  # default
app.geometry("1000x800")
app.title("Tomato Leaf Disease Recognition")

labelTitle = customtkinter.CTkLabel(app,
fg_color="transparent",text="Tomato Leaf Disease Classification",
anchor="center", font=('Arial', 22, "bold"))
labelTitle.place(rely=0.05, relx=0.32)

buttonSelect = customtkinter.CTkButton(app, text="Select File",
font=('Arial', 22), width=300, height=40, corner_radius=20,
command=searchImage)
buttonSelect.place(rely=0.15, relx=0.35)

image_label = customtkinter.CTkLabel(app, text="")
image_label.place(rely=0.25, relx=0.38)

buttonPredict = customtkinter.CTkButton(app, text="Predict",
font=('Arial', 22), width=300, height=40, corner_radius=20,
command=runApp)
buttonPredict.place(rely=0.60, relx=0.35)

imagePredict = customtkinter.CTkLabel(app, text="", font=('Arial',
18))
imagePredict.place(rely=0.70, relx=0.25)

app.mainloop()
```

7. TEST AND EVALUATION

7.1 ACCURACY RATE

Training and Testing	Accuracy Rate	
	Train	Validation
90:10	0.9946 (99.46%)	0.8507 (85.07%)
80:20	0.9998 (99.98%)	0.8532 (85.32%)
70:30	0.9950 (99.50%)	0.8252 (82.52%)

7.2 LEARNING RATE

Training and Testing	Learning Rate
90:10	0.001000
80:20	0.001000
70:30	0.001000

7.3 ERROR RATE

Training and Testing	Error Rate
90:10	0.1493
80:20	0.1468
70:30	0.1748

7.4 RECALL

Training and Testing	Recall
90:10	0.9701
80:20	0.9591
70:30	0.9714

7.5 PRECISION

Training and Testing	Precision
90:10	0.9701
80:20	0.9718
70:30	0.9637

8. CONCLUSION

Based on the provided values, the 80:20 ratio (80% training data, 20% validation data) appears to perform the best among the three ratios.

On the training set, the 80:20 ratio had a high accuracy rate of 0.9998, and on the validation set, it had a rate of 0.8532. Although the accuracy rate for the 70:30 ratio was marginally lower on the training set (0.9950), it performed considerably worse on the validation set (0.8252). Similar to this, the 90:10 ratio performed similarly on the validation set (0.8507) while having a slightly lower accuracy rate on the training set (0.9946).

In addition, the 80:20 ratio showed precision and recall values that were greater or on par with those of the other ratios, at 0.9718 and 0.9591, respectively. The precision was marginally lower (0.9637) and recall was marginally greater (0.9714) for the 70:30 ratio. Similar to the 80:20 ratio in terms of precision (0.9701) but with a little poorer recall (0.9701).

Thus, the 80:20 ratio exhibits the best overall performance in terms of accuracy, precision, and recall based on the numbers provided.

9. REFERENCES

Baeldung. (2023, April 14). *How relu and dropout layers work in cnns*. Baeldung on Computer Science. https://www.baeldung.com/cs/ml-relu-dropout-layers#:~:text=The%20Rectified%20Linear%20Unit,-3.1.&text=A%20trained%20CNN%20has%20hidden,relevant%20for%20that%20particular%20input.

Dertat, A. (2017, November 13). *Applied deep learning - part 4: Convolutional Neural Networks*. Medium. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Shoaib, M., Shah, B., El-Sappagh, S., Ali, A., Ullah, A., Alenezi, F., Gechev, T., Hussain, T., & Ali, F. (2023). An advanced deep learning models-based plant disease detection: A review of recent research. *Frontiers in Plant Science*, *14*. https://doi.org/10.3389/fpls.2023.1158933

Panthee, D., & Chen, F. (2010). Genomics of Fungal Disease Resistance in Tomato. *Current Genomics*, *11*(1), 30–39. https://doi.org/10.2174/138920210790217927

Trivedi, N. K., Gautam, V., Anand, A., Aljahdali, H. M., Villar, S. G., Anand, D., Goyal, N., & Kadry, S. (2021). Early Detection and Classification of Tomato Leaf Disease Using High-Performance Deep Neural Network. *Sensors*, *21*(23), 7987. https://doi.org/10.3390/s21237987

Khan, Q. (2022, October 1). *Tomato disease multiple sources*. Kaggle. https://www.kaggle.com/datasets/cookiefinder/tomato-disease-multiple-sources?https%3A%2F%2Fwww.kaggle.com%2Fdatasets%2Fcookiefinder%2Ftomato-disease-multiple-sources%3Fresource=downloadresource

Tensorflow. (2023, May 23). *Basic classification: Classify images of clothing : Tensorflow Core*. TensorFlow. https://www.tensorflow.org/tutorials/keras/classification

Yamashita, R., Nishio, M., Do, R. K., & Togashi, K. (2018). Convolutional Neural Networks: An overview and application in Radiology. *Insights into Imaging*, *9*(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9