MODEL

```
# -*- coding: utf-8 -*-
"""Another copy of Copy of FinalDraft.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1Cb0lJ0N2O9DXFyK3xO7Ac0_1lAvD9-YC
"""

# Import necessary libraries
import os
import numpy as np
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense , Dropout
import matplotlib.pyplot as plt

from google.colab import drive

# Mount Google Drive
```

```
drive.mount('/content/drive')


# Define paths to training and testing datasets
train_path = '/content/drive/MyDrive/tomato/train'
test_path = '/content/drive/MyDrive/tomato/val'


train_gen =
ImageDataGenerator(rescale=(1./255),horizontal_flip=True,shear_range=0.2,zo
om_range = 0.2)
test_gen = ImageDataGenerator(rescale=(1./255))  #--> (0 to 255) convert to (0
to 1)


train = train_gen.flow_from_directory( '/content/drive/MyDrive/tomato/train',
                    target_size=(120, 120),
                    class_mode='categorical',
                    subset='training',
                    batch_size=9)
test = test_gen.flow_from_directory('/content/drive/MyDrive/tomato/val',
                   target_size=(120, 120),
                    class_mode='categorical',
                    batch_size=9)


train.class_indices


# CNN model
from tensorflow.keras.layers import
Convolution2D,MaxPooling2D,Flatten,Dense,BatchNormalization,GlobalAverag
ePooling2D,Activation
```

```
from tensorflow.keras.models import Sequential

model = Sequential()

# Block 0
model.add(Conv2D(64, (5, 5), strides=1, padding="same", input_shape=(120,
120, 3)))
model.add(BatchNormalization())
model.add(Activation("relu"))

# Block 1
model.add(Conv2D(64, (5, 5), strides=1, padding="same"))
model.add(MaxPooling2D((4, 4)))
model.add(BatchNormalization())
model.add(Activation("relu"))

# Block 2
model.add(Conv2D(128, (3, 3), strides=2, padding="same"))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.1))  # Adjust dropout rate

# Block 3
model.add(Conv2D(256, (7, 7), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
```

```
model.add(Dropout(0.2))  # Adjust dropout rate

# Block 4
model.add(Conv2D(512, (3, 3), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.25))  # Adjust dropout rate

# Block 5
model.add(Conv2D(512, (3, 3), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.15))  # Adjust dropout rate

# Global Average Pooling
model.add(GlobalAveragePooling2D())

# Fully connected layers
model.add(Dense(1024, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.3))  # Adjust dropout rate
model.add(Dense(512, activation="relu"))
model.add(BatchNormalization())
model.add(Dense(256, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.4))  # Adjust dropout rate
```

```
# Output layer
model.add(Dense(9, activation='softmax'))

model.summary()

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['acc
uracy'])

#performing early stopping to avoid overfitting
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor = 'val_accuracy', mode = 'max',
patience = 20, verbose = 1, restore_best_weights = True)

history = model.fit(train,batch_size=10,validation_data=test,epochs=50)

model.save('/content/drive/MyDrive/Colab Notebooks/FinalDraft.h5')

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model on the validation generator
val_results = model.evaluate(test)

# Extract the metrics from the evaluation results
val_loss = val_results[0]
val_accuracy = val_results[1]

print(f'Validation Loss: {val_loss}')
print(f'Validation Accuracy: {val_accuracy}')

# Predict on the validation generator
val_pred = model.predict(test)
```

```
# Convert predicted probabilities to class labels
val_pred_classes = np.argmax(val_pred, axis=1)

# Assuming your validation labels are one-hot encoded
val_true_classes = test.classes

# Calculate metrics
accuracy = accuracy_score(val_true_classes, val_pred_classes)
precision = precision_score(val_true_classes, val_pred_classes,
average='weighted')
recall = recall_score(val_true_classes, val_pred_classes, average='weighted')
f1 = f1_score(val_true_classes, val_pred_classes, average='weighted')

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')

# Confusion matrix
conf_matrix = confusion_matrix(val_true_classes, val_pred_classes)
print('Confusion Matrix:')
print(conf_matrix)

# Plot confusion matrix
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
```

```
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

from sklearn.metrics import  confusion_matrix



# Calculate sensitivity and specificity for each class
sensitivity_per_class = np.diag(conf_matrix) / np.sum(conf_matrix, axis=1)
specificity_per_class = np.diag(conf_matrix) / np.sum(conf_matrix, axis=0)

for i in range(len(sensitivity_per_class)):
    print(f'Class {i} - Sensitivity (Recall): {sensitivity_per_class[i]}, Specificity: {specificity_per_class[i]}')

import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

import matplotlib.pyplot as plt

# Load the pre-trained model
model = load_model('/content/drive/MyDrive/FinalDraft.h5')

# Load and preprocess the input image
```

```
img_path =
'/content/drive/MyDrive/tomato/train/Tomato___Tomato_Yellow_Leaf_Curl_V
irus/cfbac9ed-82d2-4ccd-9a73-c97c0f92b2e2___UF.GRC_YLCV_Lab 02814.JPG'
img = image.load_img(img_path, target_size=(120, 120))


plt.imshow(img)
plt.title('Input Image')
plt.show()


img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0


# Make prediction
predictions = model.predict(img_array)
print(predictions)


class_labels = list(train.class_indices.keys())
# Map predictions to class labels
predicted_class_index = np.argmax(predictions)
predicted_class_label = class_labels[predicted_class_index]


# Show the predicted class label
print(f'Predicted Class: {predicted_class_label}')
```

WEBSITE INTEGRATION CODE

App.py

```
from __future__ import division, print_function

import os

import numpy as np

from keras.models import load_model

from keras.preprocessing import image

from keras.applications.imagenet_utils import preprocess_input


# Flask utils

from flask import Flask, request, render_template

from werkzeug.utils import secure_filename

from gevent.pywsgi import WSGIServer


app = Flask(__name__)


# Model saved with Keras model.save()

MODEL_PATH = "C:/Users/gpree/OneDrive/Desktop/prediction/FinalDraft.h5"


# Load your trained model

model = load_model(MODEL_PATH)

print('Model loaded. Check http://127.0.0.1:5000/')


# Ensure the 'uploads' directory exists

UPLOAD_FOLDER = os.path.join(os.path.dirname(__file__), 'uploads')
```

```
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def model_predict(img_path, model):
    img = image.load_img(img_path, target_size=(120, 120)) # -----------

    """Preprocessing the image
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x /= 255.0  #Match the rescale factor used during training
    x = preprocess_input(x, mode='caffe')
    preds = model.predict(x)"""

    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0
    img_array = preprocess_input(img_array)
    # Make prediction
    predictions = model.predict(img_array)

    return predictions

@app.route('/', methods=['GET'])
def index():
```

```
    # Main page
    return render_template('index.html')


@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']


        # Save the file to temporary
        file_path = os.path.join(app.config['UPLOAD_FOLDER'],
secure_filename(f.filename))
        f.save(file_path)
        class_indices = {'Tomato___Bacterial_spot': 0,
                 'Tomato___Early_blight': 1,
                 'Tomato___Late_blight': 2,
                 'Tomato___Leaf_Mold': 3,
                 'Tomato___Septoria_leaf_spot': 4,
                 'Tomato___Target_Spot': 5,
                 'Tomato___Tomato_Yellow_Leaf_Curl_Virus': 6,
                 'Tomato___Tomato_mosaic_virus': 7,
                 'Tomato___healthy': 8}
        # Make prediction
        predictions = model_predict(file_path, model)


        # Get the predicted class index
        predicted_class_index = np.argmax(predictions)
```

```
        class_labels = list(class_indices.keys())
        predicted_class_label = class_labels[predicted_class_index]

        return predicted_class_label
    return None


if __name__ == '__main__':
    app.run(debug=True)
```

WEBSITE

Base.html

```html
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>DL Model Prediction</title>
    <link
href="https://cdn.bootcss.com/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet">
```

```
    <script
src="https://cdn.bootcss.com/popper.js/1.12.9/umd/popper.min.js">
</script>

    <script
src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>

    <script
src="https://cdn.bootcss.com/bootstrap/4.0.0/js/bootstrap.min.js"><
/script>

    <link href="{{ url_for('static', filename='css/main.css') }}"
rel="stylesheet">

</head>

<center>

<body>

        <div class="container">

            <h1 class="navbar-brand" href="#">Plant Disease
Prediction</h1>


        </div>

    <div class="container">

        <div id="content" style="margin-top:2em">{% block content
%}{% endblock %}</div>

    </div>

</body>


<footer>
```

```
   <script src="{{ url_for('static', filename='js/main.js') }}"
type="text/javascript"></script>
</footer>
   </center>

</html>
```

Index.html

```
{% extends "base.html" %} {% block content %}

<center>

<h2>Plant Disease Prediction using Deep Learning</h2>

<div>
   <form id="upload-file" method="post" enctype="multipart/form-
data">
      <label for="imageUpload" class="upload-label">
         Click to Upload your Picture
      </label>
      <input type="file" name="file" id="imageUpload" accept=".png,
.jpg, .jpeg">
   </form>
```

```
    <div class="image-section" style="display:none;">
        <div class="img-preview">
            <div id="imagePreview">
            </div>
        </div>
        <div>
            <button style="background-color:grey ;font-size : 20px;
border-color:white; height:50px;width:200px; color:white" class="btn
btn-primary btn-lg " id="btn-predict">Predict the Disease</button>
        </div>
    </div>

    <div class="loader" style="display:none;"></div>

    <h3 id="result">
        <span> </span>
    </h3>

</div>

</center>

{% endblock %}
```

index.html

Success.py

```
# Import necessary libraries
import os
import numpy as np
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense , Dropout
import matplotlib.pyplot as plt

from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Define paths to training and testing datasets
train_path = '/content/drive/MyDrive/tomato/train'
test_path = '/content/drive/MyDrive/tomato/val'

train_gen =
ImageDataGenerator(rescale=(1./255),horizontal_flip=True,shear_ran
ge=0.2,zoom_range = 0.2)
```

```
test_gen = ImageDataGenerator(rescale=(1./255))  #--> (0 to 255) convert to (0 to 1)


train = train_gen.flow_from_directory('/content/drive/MyDrive/tomato/train',
                                      target_size=(120, 120),
                                      class_mode='categorical',
                                      subset='training',
                                      batch_size=9)
test = test_gen.flow_from_directory('/content/drive/MyDrive/tomato/val',
                                    target_size=(120, 120),
                                      class_mode='categorical',
                                      batch_size=9)


train.class_indices


# CNN model
from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense,BatchNormalization,GlobalAveragePooling2D,Activation
from tensorflow.keras.models import Sequential


model = Sequential()
```

```
# Block 0
model.add(Conv2D(64, (5, 5), strides=1, padding="same",
input_shape=(120, 120, 3)))
model.add(BatchNormalization())
model.add(Activation("relu"))

# Block 1
model.add(Conv2D(64, (5, 5), strides=1, padding="same"))
model.add(MaxPooling2D((4, 4)))
model.add(BatchNormalization())
model.add(Activation("relu"))

# Block 2
model.add(Conv2D(128, (3, 3), strides=2, padding="same"))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.1))  # Adjust dropout rate

# Block 3
model.add(Conv2D(256, (7, 7), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.2))  # Adjust dropout rate
```

```
# Block 4
model.add(Conv2D(512, (3, 3), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.25))  # Adjust dropout rate

# Block 5
model.add(Conv2D(512, (3, 3), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.15))  # Adjust dropout rate

# Global Average Pooling
model.add(GlobalAveragePooling2D())

# Fully connected layers
model.add(Dense(1024, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.3))  # Adjust dropout rate
model.add(Dense(512, activation="relu"))
model.add(BatchNormalization())
model.add(Dense(256, activation="relu"))
model.add(BatchNormalization())
```

```
model.add(Dropout(0.4))  # Adjust dropout rate

# Output layer
model.add(Dense(9, activation='softmax'))

model.summary()



model.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

#performing early stopping to avoid overfitting
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor = 'val_accuracy', mode =
'max', patience = 20, verbose = 1, restore_best_weights = True)

history =
model.fit(train,batch_size=10,validation_data=test,epochs=50)



model.save('/content/drive/MyDrive/Colab
Notebooks/FinalDraft.h5')

import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model on the validation generator
val_results = model.evaluate(test)
```

```
# Extract the metrics from the evaluation results
val_loss = val_results[0]
val_accuracy = val_results[1]

print(f'Validation Loss: {val_loss}')
print(f'Validation Accuracy: {val_accuracy}')

# Predict on the validation generator
val_pred = model.predict(test)

# Convert predicted probabilities to class labels
val_pred_classes = np.argmax(val_pred, axis=1)

# Assuming your validation labels are one-hot encoded
val_true_classes = test.classes

# Calculate metrics
accuracy = accuracy_score(val_true_classes, val_pred_classes)
precision = precision_score(val_true_classes, val_pred_classes,
average='weighted')
recall = recall_score(val_true_classes, val_pred_classes,
average='weighted')
f1 = f1_score(val_true_classes, val_pred_classes, average='weighted')
```

```
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')

# Confusion matrix
conf_matrix = confusion_matrix(val_true_classes, val_pred_classes)
print('Confusion Matrix:')
print(conf_matrix)

# Plot confusion matrix
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

from sklearn.metrics import  confusion_matrix



# Calculate sensitivity and specificity for each class
sensitivity_per_class = np.diag(conf_matrix) / np.sum(conf_matrix,
axis=1)
```

```
specificity_per_class = np.diag(conf_matrix) / np.sum(conf_matrix, axis=0)

for i in range(len(sensitivity_per_class)):
    print(f'Class {i} - Sensitivity (Recall): {sensitivity_per_class[i]}, Specificity: {specificity_per_class[i]}')



import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Load the pre-trained model
model = load_model('/content/drive/MyDrive/FinalDraft.h5')

# Load and preprocess the input image
img_path = '/content/drive/MyDrive/tomato/train/Tomato__Tomato_Yellow_Leaf_Curl_Virus/cfbac9ed-82d2-4ccd-9a73-c97c0f92b2e2__UF.GRC_YLCV_Lab 02814.JPG'
img = image.load_img(img_path, target_size=(120, 120))

plt.imshow(img)
plt.title('Input Image')
```

```
plt.show()

img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

# Make prediction
predictions = model.predict(img_array)
print(predictions)

class_labels = list(train.class_indices.keys())
# Map predictions to class labels
predicted_class_index = np.argmax(predictions)
predicted_class_label = class_labels[predicted_class_index]

# Show the predicted class label
print(f'Predicted Class: {predicted_class_label}')
```

<u>Workingmodel.py</u>

```
# Import necessary libraries
import os
import numpy as np
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
```

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense , Dropout
import matplotlib.pyplot as plt


# In[ ]:


from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')


# In[ ]:


# Define paths to training and testing datasets
train_path = '/content/drive/MyDrive/tomato/train'
test_path = '/content/drive/MyDrive/tomato/val'
```

```
# In[ ]:



train_gen =
ImageDataGenerator(rescale=(1./255),horizontal_flip=True,shear_ran
ge=0.2,zoom_range = 0.2)
test_gen = ImageDataGenerator(rescale=(1./255))  #--> (0 to 255)
convert to (0 to 1)


# In[ ]:



train = train_gen.flow_from_directory(
'/content/drive/MyDrive/tomato/train',
                                      target_size=(120, 120),
                                      class_mode='categorical',
                                      subset='training',
                                      batch_size=9)
test =
test_gen.flow_from_directory('/content/drive/MyDrive/tomato/val',
                                    target_size=(120, 120),
                                      class_mode='categorical',
                                      batch_size=9)
```

```
# In[ ]:


train.class_indices


# In[ ]:


# CNN model
from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense,BatchNormalization,GlobalAveragePooling2D,Activation
from tensorflow.keras.models import Sequential


# In[ ]:


model = Sequential()

# Block 0
```

```
model.add(Conv2D(64, (5, 5), strides=1, padding="same",
input_shape=(120, 120, 3)))
model.add(BatchNormalization())
model.add(Activation("relu"))

# Block 1
model.add(Conv2D(64, (5, 5), strides=1, padding="same"))
model.add(MaxPooling2D((4, 4)))
model.add(BatchNormalization())
model.add(Activation("relu"))

# Block 2
model.add(Conv2D(128, (3, 3), strides=2, padding="same"))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.1))  # Adjust dropout rate

# Block 3
model.add(Conv2D(256, (7, 7), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.2))  # Adjust dropout rate
```

```
# Block 4
model.add(Conv2D(512, (3, 3), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.25))  # Adjust dropout rate

# Block 5
model.add(Conv2D(512, (3, 3), strides=2, padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.15))  # Adjust dropout rate

# Global Average Pooling
model.add(GlobalAveragePooling2D())

# Fully connected layers
model.add(Dense(1024, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.3))  # Adjust dropout rate
model.add(Dense(512, activation="relu"))
model.add(BatchNormalization())
model.add(Dense(256, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.4))  # Adjust dropout rate
```

```
# Output layer
model.add(Dense(9, activation='softmax'))

model.summary()


# In[ ]:


model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])


# In[ ]:


#performing early stopping to avoid overfitting
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor = 'val_accuracy', mode = 'max', patience = 20, verbose = 1, restore_best_weights = True)


# In[ ]:
```

```
history =
model.fit(train,batch_size=10,validation_data=test,epochs=50)


# In[ ]:


model.save('/content/drive/MyDrive/Colab
Notebooks/FinalDraft.h5')


# In[ ]:


import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score


# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


# In[ ]:


# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()


# In[ ]:


# Evaluate the model on the validation generator
val_results = model.evaluate(test)
```

```
# Extract the metrics from the evaluation results
val_loss = val_results[0]
val_accuracy = val_results[1]

print(f'Validation Loss: {val_loss}')
print(f'Validation Accuracy: {val_accuracy}')



# In[ ]:



# Predict on the validation generator
val_pred = model.predict(test)

# Convert predicted probabilities to class labels
val_pred_classes = np.argmax(val_pred, axis=1)

# Assuming your validation labels are one-hot encoded
val_true_classes = test.classes

# Calculate metrics
accuracy = accuracy_score(val_true_classes, val_pred_classes)
```

```
precision = precision_score(val_true_classes, val_pred_classes, average='weighted')

recall = recall_score(val_true_classes, val_pred_classes, average='weighted')

f1 = f1_score(val_true_classes, val_pred_classes, average='weighted')


print(f'Accuracy: {accuracy}')

print(f'Precision: {precision}')

print(f'Recall: {recall}')

print(f'F1 Score: {f1}')



# In[ ]:



# Confusion matrix

conf_matrix = confusion_matrix(val_true_classes, val_pred_classes)

print('Confusion Matrix:')

print(conf_matrix)


# Plot confusion matrix

plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()
```

```
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()


# In[ ]:


from sklearn.metrics import  confusion_matrix


# Calculate sensitivity and specificity for each class
sensitivity_per_class = np.diag(conf_matrix) / np.sum(conf_matrix, axis=1)
specificity_per_class = np.diag(conf_matrix) / np.sum(conf_matrix, axis=0)

for i in range(len(sensitivity_per_class)):
    print(f'Class {i} - Sensitivity (Recall): {sensitivity_per_class[i]}, Specificity: {specificity_per_class[i]}')


# In[ ]:
```

```
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Load the pre-trained model
model = load_model('/content/drive/MyDrive/FinalDraft.h5')

# Load and preprocess the input image
img_path = '/content/drive/MyDrive/tomato/train/Tomato__Tomato_Yellow_Leaf_Curl_Virus/cfbac9ed-82d2-4ccd-9a73-c97c0f92b2e2___UF.GRC_YLCV_Lab 02814.JPG'
img = image.load_img(img_path, target_size=(120, 120))

plt.imshow(img)
plt.title('Input Image')
plt.show()

img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

# Make prediction
predictions = model.predict(img_array)
```

```python
print(predictions)

class_labels = list(train.class_indices.keys())
# Map predictions to class labels
predicted_class_index = np.argmax(predictions)
predicted_class_label = class_labels[predicted_class_index]

# Show the predicted class label
print(f'Predicted Class: {predicted_class_label}')



# In[ ]:
```

Main.css

```css
.img-preview {
   width: 256px;
   height: 256px;
   position: relative;
   border: 5px solid #F8F8F8;
   box-shadow: 0px 2px 4px 0px rgba(0, 0, 0, 0.1);
   margin-top: 1em;
   margin-bottom: 1em;
}
```

```
.img-preview>div {
    width: 100%;
    height: 100%;
    background-size: 256px 256px;
    background-repeat: no-repeat;
    background-position: center;
}

input[type="file"] {
    display: none;
}

.upload-label{
    display: inline-block;
    padding: 12px 30px;
    background: #39D2B4;
    color: #fff ;
    font-size: 1em;
    transition: all .4s;
    cursor: pointer;
}

.upload-label:hover{
    background: #34495E;
```

```
    color: #39D2B4;
}

.loader {
    border: 8px solid #f3f3f3; /* Light grey */
    border-top: 8px solid #3498db; /* Blue */
    border-radius: 50%;
    width: 50px;
    height: 50px;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

#result{
    display: inline-block;
    padding: 5px 20px;
    background: Tomato;
    color: #fff ;
    }
```

Main.jss

```
.img-preview {
   width: 256px;
   height: 256px;
   position: relative;
   border: 5px solid #F8F8F8;
   box-shadow: 0px 2px 4px 0px rgba(0, 0, 0, 0.1);
   margin-top: 1em;
   margin-bottom: 1em;
}

.img-preview>div {
   width: 100%;
   height: 100%;
   background-size: 256px 256px;
   background-repeat: no-repeat;
   background-position: center;
}

input[type="file"] {
   display: none;
}
```

```
.upload-label{
    display: inline-block;
    padding: 12px 30px;
    background: #39D2B4;
    color: #fff ;
    font-size: 1em;
    transition: all .4s;
    cursor: pointer;
}

.upload-label:hover{
    background: #34495E;
    color: #39D2B4;
}

.loader {
    border: 8px solid #f3f3f3; /* Light grey */
    border-top: 8px solid #3498db; /* Blue */
    border-radius: 50%;
    width: 50px;
    height: 50px;
    animation: spin 1s linear infinite;
}
```

```css
@keyframes spin {
   0% { transform: rotate(0deg); }
   100% { transform: rotate(360deg); }
}

#result{
   display: inline-block;
   padding: 5px 20px;
   background: Tomato;
   color: #fff ;
   }
```