import tensorflow as tf

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.metrics import confusion\_matrix

import numpy as np

import matplotlib.pyplot as plt

import cv2

# Set the directories for the training and validation datasets

train\_dir = '/Users/prarabdhkumar/Downloads/apple\_disease\_classification/Train'

val\_dir = '/Users/prarabdhkumar/Downloads/apple\_disease\_classification/Test'

# Set the parameters for the image preprocessing

img\_width, img\_height = 224, 224

batch\_size = 32

# Create data generators to preprocess the images

train\_datagen = ImageDataGenerator(rescale=1./255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True)

val\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(train\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode='categorical')

val\_generator = val\_datagen.flow\_from\_directory(val\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode='categorical')

# Define the CNN architecture

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_width, img\_height, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dense(4, activation='softmax')

])

model.summary()

# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

epochs = 50

history = model.fit(train\_generator,

epochs=epochs,

validation\_data=val\_generator)

# Save the trained model

model.save('fruitdisesae.keras')

# Plot the accuracy and loss curves

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(epochs)

plt.figure(figsize=(8, 8))

plt.subplot(2, 1, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy')

plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()

import numpy as np

from sklearn.metrics import confusion\_matrix

from sklearn.utils.multiclass import unique\_labels

import matplotlib.pyplot as plt

# Generate predictions and compute the confusion matrix

Y\_pred = model.predict(val\_generator)

y\_pred = np.argmax(Y\_pred, axis=1)

cm = confusion\_matrix(val\_generator.classes, y\_pred)

# Define class labels

class\_names = ['blotch', 'scab', 'rot','healthy']

# Plot the confusion matrix

fig, ax = plt.subplots()

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

ax.figure.colorbar(im, ax=ax)

# Set the x-axis and y-axis labels

ax.set(xticks=np.arange(cm.shape[1]),

yticks=np.arange(cm.shape[0]),

xticklabels=class\_names, yticklabels=class\_names,

xlabel='Predicted label', ylabel='True label')

# Rotate the x-axis labels to vertical orientation

plt.setp(ax.get\_xticklabels(), rotation=90, ha="right",

rotation\_mode="anchor")

# Loop over the data and annotate each cell with its value

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

ax.text(j, i, format(cm[i, j], 'd'),

ha="center", va="center",

color="white" if cm[i, j] > cm.max() / 2. else "black")

# Add a title to the plot

ax.set\_title("Confusion matrix")

fig.tight\_layout()

plt.show()

from sklearn.metrics import classification\_report

# Generate predictions

Y\_pred = model.predict(val\_generator)

y\_pred = np.argmax(Y\_pred, axis=1)

# Get the class labels

class\_labels = list(val\_generator.class\_indices.keys())

# Generate the classification report

report = classification\_report(val\_generator.classes, y\_pred, target\_names=class\_labels)

print(report)

# Get the class labels

class\_labels = list(val\_generator.class\_indices.keys())

# Get the class-wise accuracy values

class\_wise\_accuracy = cm.diagonal() / cm.sum(axis=1)

# Create a bar chart

plt.bar(class\_labels, class\_wise\_accuracy)

plt.title('Accuracy by Class')

plt.xlabel('Class')

plt.ylabel('Accuracy')

plt.show()

# Evaluate the model on the validation data

\_, accuracy = model.evaluate(val\_generator)

# Print the validation accuracy

print('Validation Accuracy: %.2f' % (accuracy\*100))

# Get the accuracy from the history object

accuracy = history.history['accuracy'][-1]

# Print the accuracy to the console

print("Accuracy: {:.2f}%".format(accuracy \* 100))