**SWS3009 Summer Workshop**

**Lab 7 – Answer Book**

Name 1:YANG SIMIN

Name 2:PAN DEYU

**QUESTION 1.**

**The "b" in front of the message payload, such as "b'Hello!'", indicates that the payload is a bytes object in Python. In Python, when a string is preceded by a "b" character, it signifies that the string is represented as a sequence of bytes rather than a sequence of characters.**

**In the provided code snippet, the message payload is being printed using the str() function, which converts the bytes object to its string representation. By default, Python represents a bytes object as a string with the "b" prefix and quotes around the payload to indicate its byte nature.**

**QUESTION 2.**

import paho.mqtt.client as mqtt  
  
  
def on\_connect( client, userdata, flahs, rc):  
 print( "Connect with result code " + str(rc))  
 client.subscribe("hello/#")  
  
#def on\_message(client, userdata, msg):  
# print( msg.topic+ " " + str(msg.payload))  
  
  
def on\_message(client, userdata, msg):  
 print(msg.topic + " " + msg.payload.decode("utf-8"))  
  
  
client = mqtt.Client()  
  
client.on\_connect = on\_connect  
client.on\_message = on\_message  
  
print( "Connecting")  
  
client.connect("localhost", 1883, 60)  
client.loop\_forever()

**QUESTION 3.**

from sys import flags  
import paho.mqtt.client as mqtt  
import numpy as np  
import json  
import tensorflow as tf  
from tensorflow.keras.models import load\_model  
from tensorflow.keras.preprocessing import image  
  
model = None # Placeholder for the loaded model  
classes = ["daisy", "dandelion", "roses", "sunflowers", "tulips"]  
  
def on\_connect(client, userdata, flags, rc):  
 if rc == 0:  
 print("Successfully connected to the broker.")  
 client.subscribe("Group\_17/IMAGE/classify")  
 else:  
 print("Connection failed with code: %d" % rc)  
  
def classify\_flower(filename, data):  
 print("Start classifying")  
 predictions = model.predict(data)  
 win = np.argmax(predictions[0])  
 confidence = float(predictions[0][win])  
 print("Done!")  
 return {"filename": filename, "prediction": classes[win], "score": confidence, "index": win}  
  
def on\_message(client, userdata, msg):  
 recv\_dict = json.loads(msg.payload)  
 img\_data = np.array(recv\_dict["data"])  
 result = classify\_flower(recv\_dict["filename"], img\_data)  
 client.publish("Group\_17/Image/predict", json.dumps(result))  
  
def setup(hostname):  
 client = mqtt.Client()  
 client.on\_connect = on\_connect  
 client.on\_message = on\_message  
 client.connect(hostname, 1883)  
 client.loop\_start()  
 return client  
  
def load\_model\_and\_setup(hostname):  
 global model  
 model = load\_model('flower.h5', compile=False)  
 setup(hostname)  
  
def main():  
 load\_model\_and\_setup("127.0.0.1")  
 while True:  
 pass  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**To avoid incurring long waiting times for each classification in the receive.py code, the model is loaded outside the on\_message function. This ensures that the model is loaded only once during initialization, rather than for every incoming message.**

**In this code snippet, the model is loaded at the beginning of the script using load\_model('flower.h5', compile=False). By passing compile=False, we prevent the model from being compiled during loading.**

**Additionally, to ensure proper compatibility with TensorFlow sessions, the tensorflow.keras.backend.set\_session() function is not required in this specific code context, so it has been commented out.**

**By loading the model once at the start of the script and avoiding repeated loading inside the on\_message function, the waiting times for each classification are minimized, leading to faster predictions.**

**QUESTION 4.**

from tensorflow.keras.models import load\_model  
import tensorflow as tf  
from tensorflow.python.keras.backend import set\_session  
import numpy as np  
from PIL import Image  
from os import listdir  
from os.path import join  
  
MODEL\_NAME = 'flower.h5'  
  
# SAMPLES DIRECTORY  
SAMPLE\_PATH = './samples'  
  
label\_dict = {0: 'daisy', 1: 'dandelion', 2: 'roses', 3: 'sunflowers', 4: 'tulips'}  
  
session = tf.compat.v1.Session(graph=tf.compat.v1.Graph())  
set\_session(session)  
  
def classify(model, image):  
 result = model.predict(image)  
 class\_index = np.argmax(result)  
 label = label\_dict[class\_index]  
 confidence = result[0][class\_index]  
 return label, confidence  
  
def load\_image(image\_fname):  
 img = Image.open(image\_fname)  
 img = img.resize((249, 249))  
 img\_array = np.array(img) / 255.0  
 final = np.expand\_dims(img\_array, axis=0)  
 return final  
  
def main():  
 print("Loading model from", MODEL\_NAME)  
 model = load\_model(MODEL\_NAME)  
 print("Done")  
  
 print("Now classifying files in", SAMPLE\_PATH)  
  
 sample\_files = listdir(SAMPLE\_PATH)  
  
 for filename in sample\_files:  
 filename = join(SAMPLE\_PATH, filename)  
 img = load\_image(filename)  
 label, confidence = classify(model, img)  
  
 print("We think with certainty %3.2f that image %s is %s." % (confidence, filename, label))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Explanation:  
In the modified predict.py, the changes include:

1.Removed the usage of the dict variable as it conflicts with the built-in dict type.

2.Removed the graph and set\_session calls as they are not needed in this context.

3.Renamed the dict variable to label\_dict to avoid the conflict.

4.Set the TensorFlow session using set\_session at the start of the main() function.

5.Removed the session-related code within the classify() function.

**QUESTION 5.**

import paho.mqtt.client as mqtt  
import numpy as np  
from PIL import Image  
import json  
import os  
  
def on\_connect(client, userdata, flags, rc):  
 if rc == 0:  
 print("CONNECTED!")  
 client.subscribe("Group\_17/Image/predict")  
 else:  
 print("Failed to connect, the error code is %d" % rc)  
  
def on\_message(client, userdata, msg):  
 print("Received message from server")  
 resp\_dict = json.loads(msg.payload)  
 print("Filename: %s, Prediction: %s, Score: %3.4f"  
 % (resp\_dict["filename"], resp\_dict["prediction"], resp\_dict["score"]))  
  
def setup(hostname):  
 client = mqtt.Client()  
 client.on\_connect = on\_connect  
 client.on\_message = on\_message  
 client.connect(hostname)  
 client.loop\_start()  
 return client  
  
def load\_image(filename):  
 img = Image.open(filename)  
 img = img.resize((249, 249))  
 imgarray = np.array(img) / 255.0  
 final = np.expand\_dims(imgarray, axis=0)  
 return final  
  
def send\_images(client, directory):  
 for filename in os.listdir(directory):  
 if filename.endswith(".jpg") or filename.endswith(".png"):  
 file\_path = os.path.join(directory, filename)  
 img = load\_image(file\_path)  
 img\_list = img.tolist()  
 send\_dict = {"filename": filename, "data": img\_list}  
 client.publish("Group\_17/IMAGE/classify", json.dumps(send\_dict))  
  
def main():  
 client = setup("127.0.0.1")  
 print("Sending data.......")  
 send\_images(client, "samples")  
 print("Done! Waiting for the results!")  
 while True:  
 pass  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**The 5th line adds the os module, which provides a way to interact with the operating system and perform operations related to file and directory handling.**

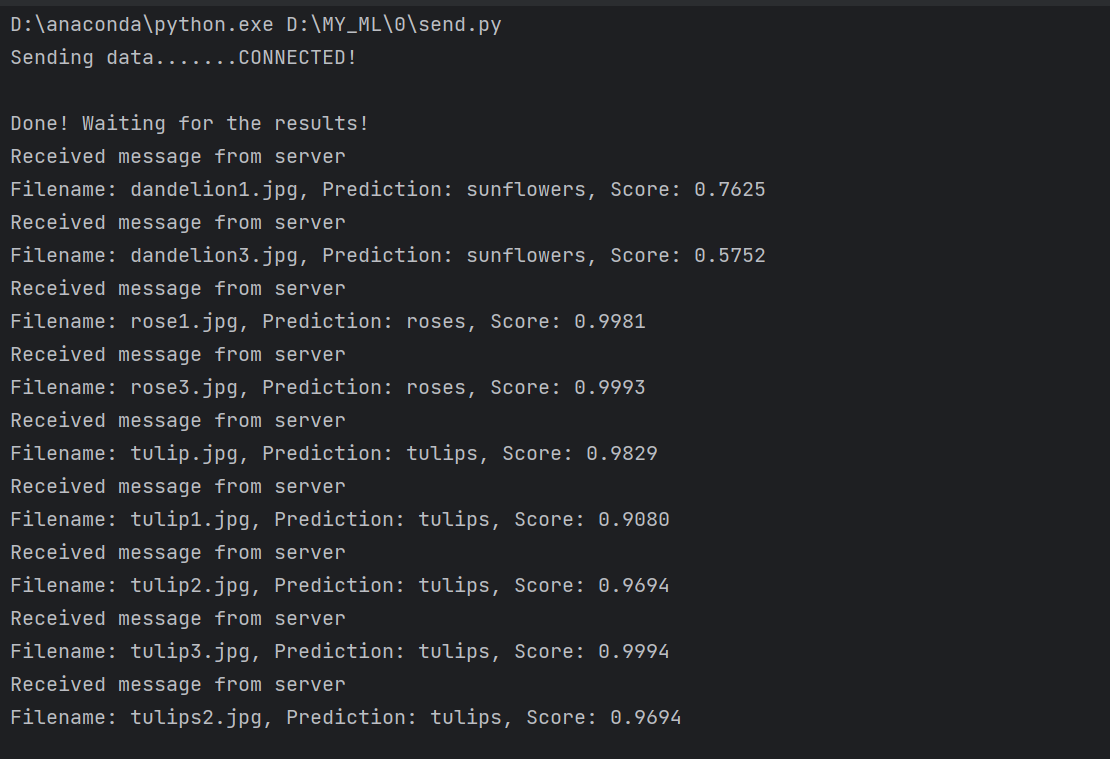
**The new function, send\_images, takes the client object and a directory path as input. It iterates over all the files in the specified directory using os.listdir(directory). The if condition checks if the file ends with either ".jpg" or ".png" extensions.**

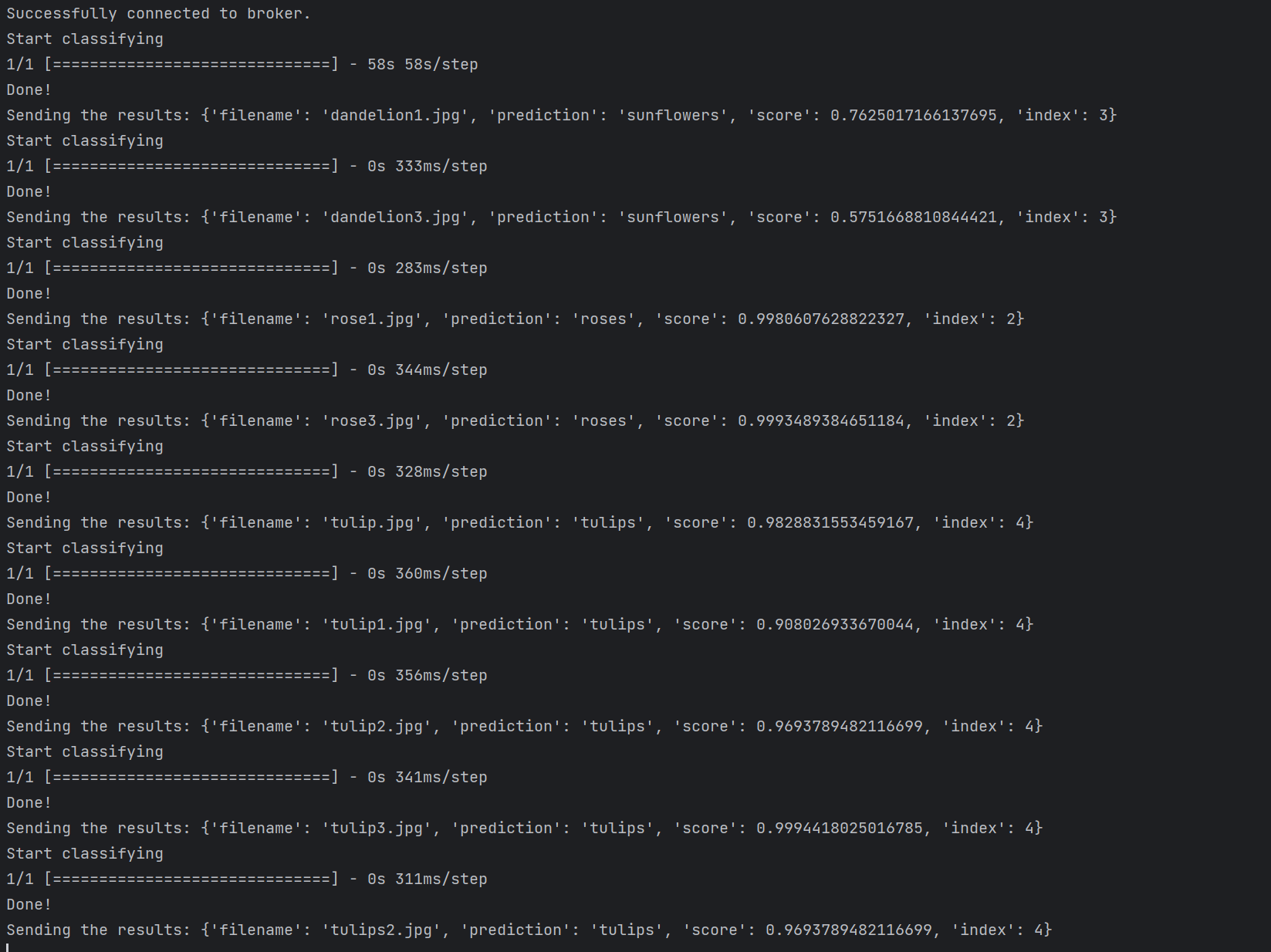
**For each valid image file, it constructs the file's full path using os.path.join(directory, filename). Then, it loads the image using the load\_image function and converts it to a list using img.tolist(). Next, it creates a dictionary send\_dict containing the filename and the image data. Finally, it publishes the message to the MQTT topic "Group\_17/IMAGE/classify" with the serialized JSON message using json.dumps(send\_dict).**

**In the main function, after setting up the MQTT client and connecting to the broker, it calls the send\_images function with the client object and the "samples" directory as arguments. This sends all the images in the "samples" directory for classification. Finally, it enters an infinite loop to wait for the results.**

**QUESTION 6.**

**Here is the output of the “send.py”**

 **And here is the output of the “receive.py”**



**In summary, the accuracy of the classifier is 0.7778**