**CS3237 Introduction to IoT**

**Lab 4**

**ANSWER BOOK**

|  |  |
| --- | --- |
| Student ID: A0184569L | Name: Le Hong Long |
| Student ID: A0243404R | Name: Chong Chee Kit |

**Question 1 (2 MARKS)**

The ‘b’ means the data type is byte (array), it is present because the data is sent as a byte stream and stored into ‘msg.payload’ as a byte array in python.

**Question 2 (3 MARKS)**

The modified code is pasted below:

import paho.mqtt.client as mqtt

def on\_connect(client, userdata, flags, rc):

print(f'Connected with result code {rc}')

client.subscribe('hello/#')

def on\_message(client, userdata, msg):

print(f'{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 (2 marks)**

How I avoid long model load delays is:

1. Global variables for session, graph (as a property of session) and model are saved

MODEL\_NAME = 'flowers.hd5'

session = tf.compat.v1.Session(graph=tf.compat.v1.Graph())

with session.graph.as\_default():

set\_session(session)

model = load\_model(MODEL\_NAME)

2. The same session and model (variables) are used for predictions, thus avoiding the cost of loading the model again

with session.graph.as\_default():

set\_session(session)

result = model.predict(data)

**Question 4 (2 mark)**

I ~~do~~/do not (cancel one) need to freeze weights because the weights are probably not optimal by default, even if the model was pretrained on a similar task. Thus, finetuning the full model will likely give better accuracy. However, by freezing some of the weights, I will not have to retrain the full model, thus I can make use of the learned features from the pretrained model. Also, it is cheaper (thus faster) to train only part of the model since there are less weights to update.

**Question 5 (4 marks)**

My predict function WITH explanation is: (Explanation is done via comments)

def classify\_flower(filename, data):

print('Start classifying')

# Set the current graph and session to the one with the model weights loaded, thus avoiding having to load the model again

with session.graph.as\_default():

set\_session(session)

# Do prediction

result = model.predict(data)

# 'win' is the index of the maximum output value (highest probability)

win = np.argmax(result).item()

# 'score' is the output value (probability) at index 'win'

score = result[0][win].item()

print('Done.')

# Return dictionary of results

return {

'filename': filename,

'prediction': classes[win], # Map index to class name

'score': score,

'index': win

}

**Question 6 (4 marks)**

Relevant code with explanation:

1. We modify the main() function of “send.py” to send all the images in the “samples” directory as follows: (Explanation in comments)

PATH = 'samples'

def main():

client = setup(HOSTNAME)

print('Sending data.')

# Loop through all images in PATH

for file in listdir(PATH):

# Get path to image

file\_path = join(PATH, file)

print(f'Sending image {file\_path}')

# Send image classification request

send\_image(client, file\_path)

print('Done. Waiting for results.')

while True:

pass

2. For “receive.py”, we will implement a ‘request\_queue’ which is a queue so that we can process the incoming requests one by one. We start by defining a global variable for the queue

from collections import deque

request\_queue = deque()

Next, we rename the on\_message() function to process\_request() but accepting only ‘client’ and ‘msg’ parameters (since we are not using ‘userdata’ anyway)

def process\_request(client, msg):

# Body same as the OLD on\_message()

The new on\_message() should only push the received request to the ‘request\_queue’

def on\_message(client, userdata, msg):

request\_queue.append((client, msg))

Finally, for the main() function, we modify the while loop to continuously check whether the ‘request\_queue’ has any request, if there is then we pop the request and pass it to the ‘process\_request’ function for processing, else sleep for 0.1s so that we don’t have to do “busy waiting” which would waste CPU cycles

from time import sleep

def main():

setup(HOSTNAME)

while True:

if request\_queue:

client, msg = request\_queue.popleft()

process\_request(client, msg)

else:

sleep(0.1)

**Question 7 (3 marks)**

My sample output:

1. Output for “send.py”:

Sending data.

Connected

Sending image samples/tulip2.jpg

Sending image samples/dandelion3.jpg

Sending image samples/daisy3.jpeg

Sending image samples/dandelion1.jpg

Sending image samples/sunflower3.jpeg

Sending image samples/tulip3.jpg

Sending image samples/sunflower1.jpeg

Sending image samples/sunflower2.jpeg

Sending image samples/tulip1.jpg

Sending image samples/rose1.jpg

Sending image samples/rose2.jpeg

Sending image samples/dandelion2.jpeg

Sending image samples/rose3.jpg

Sending image samples/daisy2.jpeg

Sending image samples/tulips2.jpg

Sending image samples/tulip.jpg

Done. Waiting for results.

Received message from server.

Filename: samples/tulip2.jpg, Prediction: tulips, Score: 0.9990

Received message from server.

Filename: samples/dandelion3.jpg, Prediction: dandelion, Score: 0.8666

Received message from server.

Filename: samples/daisy3.jpeg, Prediction: daisy, Score: 0.5174

Received message from server.

Filename: samples/dandelion1.jpg, Prediction: dandelion, Score: 0.9928

Received message from server.

Filename: samples/sunflower3.jpeg, Prediction: sunflowers, Score: 0.9987

Received message from server.

Filename: samples/tulip3.jpg, Prediction: tulips, Score: 0.9999

Received message from server.

Filename: samples/sunflower1.jpeg, Prediction: sunflowers, Score: 0.9996

Received message from server.

Filename: samples/sunflower2.jpeg, Prediction: sunflowers, Score: 0.9970

Received message from server.

Filename: samples/tulip1.jpg, Prediction: tulips, Score: 0.8291

Received message from server.

Filename: samples/rose1.jpg, Prediction: roses, Score: 0.9986

Received message from server.

Filename: samples/rose2.jpeg, Prediction: roses, Score: 0.9996

Received message from server.

Filename: samples/dandelion2.jpeg, Prediction: daisy, Score: 0.8788

Received message from server.

Filename: samples/rose3.jpg, Prediction: roses, Score: 0.9983

Received message from server.

Filename: samples/daisy2.jpeg, Prediction: daisy, Score: 0.3872

Received message from server.

Filename: samples/tulips2.jpg, Prediction: tulips, Score: 0.9990

Received message from server.

Filename: samples/tulip.jpg, Prediction: tulips, Score: 0.9986

2. Output for “receive.py” (Excluding tensorflow “INFO” and “WARNING” outputs):

Successfully connected to broker.

Start classifying

Done.

Sending results: {'filename': 'samples/tulip2.jpg', 'prediction': 'tulips', 'score': 0.9990081191062927, 'index': 4}

Start classifying

Done.

Sending results: {'filename': 'samples/dandelion3.jpg', 'prediction': 'dandelion', 'score': 0.866600513458252, 'index': 1}

Start classifying

Done.

Sending results: {'filename': 'samples/daisy3.jpeg', 'prediction': 'daisy', 'score': 0.5173771381378174, 'index': 0}

Start classifying

Done.

Sending results: {'filename': 'samples/dandelion1.jpg', 'prediction': 'dandelion', 'score': 0.9927653670310974, 'index': 1}

Start classifying

Done.

Sending results: {'filename': 'samples/sunflower3.jpeg', 'prediction': 'sunflowers', 'score': 0.9987271428108215, 'index': 3}

Start classifying

Done.

Sending results: {'filename': 'samples/tulip3.jpg', 'prediction': 'tulips', 'score': 0.9998666048049927, 'index': 4}

Start classifying

Done.

Sending results: {'filename': 'samples/sunflower1.jpeg', 'prediction': 'sunflowers', 'score': 0.9996241331100464, 'index': 3}

Start classifying

Done.

Sending results: {'filename': 'samples/sunflower2.jpeg', 'prediction': 'sunflowers', 'score': 0.9969670176506042, 'index': 3}

Start classifying

Done.

Sending results: {'filename': 'samples/tulip1.jpg', 'prediction': 'tulips', 'score': 0.8291115164756775, 'index': 4}

Start classifying

Done.

Sending results: {'filename': 'samples/rose1.jpg', 'prediction': 'roses', 'score': 0.9986279010772705, 'index': 2}

Start classifying

Done.

Sending results: {'filename': 'samples/rose2.jpeg', 'prediction': 'roses', 'score': 0.9996358156204224, 'index': 2}

Start classifying

Done.

Sending results: {'filename': 'samples/dandelion2.jpeg', 'prediction': 'daisy', 'score': 0.8787693977355957, 'index': 0}

Start classifying

Done.

Sending results: {'filename': 'samples/rose3.jpg', 'prediction': 'roses', 'score': 0.998250424861908, 'index': 2}

Start classifying

Done.

Sending results: {'filename': 'samples/daisy2.jpeg', 'prediction': 'daisy', 'score': 0.3871530294418335, 'index': 0}

Start classifying

Done.

Sending results: {'filename': 'samples/tulips2.jpg', 'prediction': 'tulips', 'score': 0.9990081191062927, 'index': 4}

Start classifying

Done.

Sending results: {'filename': 'samples/tulip.jpg', 'prediction': 'tulips', 'score': 0.9985779523849487, 'index': 4}

The accuracy of my classify is 15/16 ≈ 94%

**TOTAL: \_\_\_\_\_\_\_ / 20**