In [ ]:
```
# Import the libraries
from keras.models import model_from_json
import tensorflow as tf
```

In [ ]:
```
# Load the model and verify it
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
# load json and create model
json_file = open('model_mnist.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json, custom_objects={'softmax_v2'
: tf.nn.softmax})
# load weights into new model
loaded_model.load_weights("model_mnist.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
score = loaded_model.evaluate(x_test, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

In [ ]:
```
# FUnction to connect to the AWS Waypoint

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
import time
import numpy as np

# A random programmatic shadow client ID.
SHADOW_CLIENT = "myShadowClient007"

# The unique hostname that AWS IoT generated for
# this device.
HOST_NAME = "azwputk3qmzds-ats.iot.us-west-2.amazonaws.com"

# The relative path to the correct root CA file for AWS IoT,
# that you have already saved onto this device.
ROOT_CA = "C:/Users/Administrator/Desktop/284_CERTI/AmazonRootCA1.pem"

# The relative path to your private key file that
# AWS IoT generated for this device, that you
# have already saved onto this device.
PRIVATE_KEY = "C:/Users/Administrator/Desktop/connect_device_package/CC3200_Th
ing_gksk.private.key"

# The relative path to your certificate file that
# AWS IoT generated for this device, that you
# have already saved onto this device.
CERT_FILE = "C:/Users/Administrator/Desktop/connect_device_package/CC3200_Thin
g_gksk.cert.pem"

# A programmatic shadow handler name prefix.
SHADOW_HANDLER = "CC3200_Thing_gksk"

# Automatically called whenever the shadow is updated.
def myShadowUpdateCallback(payload, responseStatus, token):
    print('UPDATE: $aws/things/' + SHADOW_HANDLER +
    '/shadow/update')
    print("payload = " + payload)
    print("responseStatus = " + responseStatus)
    print("token = " + token)

# Create, configure, and connect a shadow client.
myShadowClient = AWSIoTMQTTShadowClient(SHADOW_CLIENT)
myShadowClient.configureEndpoint(HOST_NAME, 8883)
myShadowClient.configureCredentials(ROOT_CA, PRIVATE_KEY,
  CERT_FILE)
myShadowClient.configureConnectDisconnectTimeout(10)
myShadowClient.configureMQTTOperationTimeout(5)
myShadowClient.connect()

# Create a programmatic representation of the shadow.
myDeviceShadow = myShadowClient.createShadowHandlerWithName(
  SHADOW_HANDLER, True)
```

In [ ]:
```
final = []
do_ml = False
done = True
prediction = [1]
def machine_learning(payload, responseStatus, token):
    # If the payload contains the pythonML keyword we extract the word
    if 'pythonML' in payload:
        print("Detected")
        payload = payload[payload.find('"pythonML":"')+12:]
        payload = payload[:payload.find('"')]
        pixels = payload.split('x')
        pixels = pixels[:784]
        if len(pixels) == 784:
            final_pixel = []
            for pixel in pixels:
                final_pixel.append(pixel[pixel.find(":")+1:])
            global final
            final = np.asarray(final_pixel)
            final = final.reshape(1,28,28,1)
            global do_ml
            do_ml = True
        else:
            print("problem!")

    # if shadow is not updated successfully we upload again
    elif 'No shadow exists with name' in payload:
        print("Updating shadow again!")
        global done
        done = False

    # If no data is recived from the CC3200 we wait.
    else:
        done = True
        print("waiting for the CC3200 to upload message!")


while(1):
    myDeviceShadow.shadowGet(machine_learning,5)
    while(not done):
        time.sleep(2)
        myDeviceShadow.shadowGet(machine_learning,5)
        time.sleep(2)
        myDeviceShadow.shadowUpdate(
            '{"state":{"desired":{"label":'+ '"' +str(prediction[0]) + '"' +
'}}}',
            myShadowUpdateCallback, 5)

    # if the do_ml flag is set, we do the Machine learning on the received dat
a
    if do_ml:
        # We used the loaded model to predict the classes.
        prediction = loaded_model.predict_classes(final)
        try:
            # we delete the shadow and update is with the new label.
            myDeviceShadow.shadowDelete(myShadowUpdateCallback, 5)
            time.sleep(2)
```

```
            myDeviceShadow.shadowUpdate(
            '{"state":{"desired":{"label":'+ '"' +str(prediction[0]) + '"' +
'}}}',
            myShadowUpdateCallback, 5)
        except:
            print("Trying to post!")
        do_ml = False
        done = False

    time.sleep(2)
```