**Keck IoT Project**

Querying Parking Space Availability through Voice Recognition Program

By: Tim Nguyen, Irwan Winarto, Kadrian Rickman

Instructor: Mehmet Vurkaç

Date: 12/15/2018

ECEGR 4910 – Machine Learning

Seattle University: College of Science and Engineering

Table of Contents

[Introduction 3](#_Toc531458207)

[Equipment Error! Bookmark not defined.](#_Toc531458208)

[Method 4](#_Toc531458209)

[Procedure 1 - Setting up the Arduino D1 Mini Wi-Fi module (ESP8266) **Error! Bookmark not defined.**](#_Toc531458210)

[Procedure 2 - Setting up Ngrok tunnel as a public URL to access the Wi-Fi module **Error! Bookmark not defined.**](#_Toc531458211)

[Procedure 3 - Writing the computer vision program to recognize hand gestures **Error! Bookmark not defined.**](#_Toc531458212)

[Results 9](#_Toc531458213)

[Discussion and Conclusion 11](#_Toc531458214)

[References 11](#_Toc531458215)

[Appendix 13](#_Toc531458216)

# Introduction

In this project, we are introduced to the notion of Internet of Things (IoT) through implementing a “Parking Monitoring System” that utilizes a convolutional neural net to analyze availability of parking spots at the parking lot near Loyola building. The notion of IoT is introduced through implementation of communication between different computing components, which are interfacing with each other through the internet. The project involves setting up a local Raspberry Pi as a voice recognition device that accepts voice queries about parking spot(s) availability, a virtual machine on Amazon Elastic Compute Cloud (EC2) as a server for processing inference results for these queries, and a Raspberry Pi mounted at Loyola building to capture images of the desired parking lot.

# Method

As mentioned above, the project involves three steps: setting up the local Raspberry Pi to accept voice queries, configuring a server on EC2, and setting up the Raspberry Pi at Loyola building to capture picture and send back to EC2 server for processing.

## Setting up the Raspberry Pi to listen to voice queries, with Matrix Creator as audio input device

For this Raspberry Pi to register voice queries properly, we need to hook up a good audio input device to the Pi. We are choosing the Matrix Creator because the module has an array of 8 microphones that is capable of beam-forming for enhanced voice recognition ability. The module can easily be hooked up to the Pi through the GPIO header.

Once the devices are hooked up, we need to install all the dependencies required in order for the Pi to effectively interface with the Matrix Creator. This can be done by executing these shell commands from the Pi’s terminal:

>> curl https://apt.matrix.one/doc/apt-key.gpg | sudo apt-key add -

>> echo "deb https://apt.matrix.one/raspbian $(lsb\_release -sc) main" | sudo tee /etc/apt/sources.list.d/matrixlabs.list

>> sudo apt-get update

>> sudo apt-get upgrade

>> sudo apt-get install matrixio-malos

>> sudo reboot

…after the reboot…

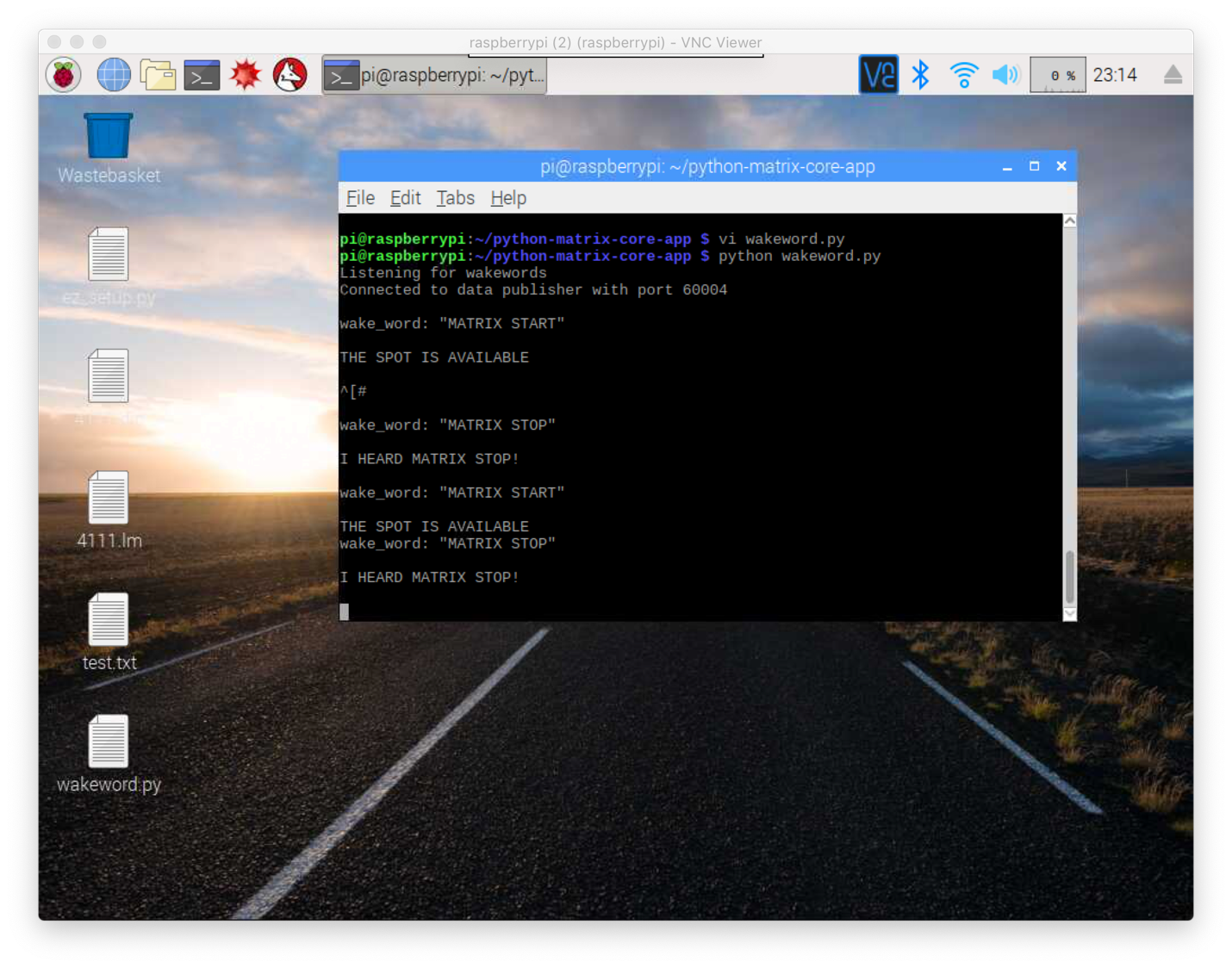
>> echo "deb http://download.opensuse.org/repositories/network:/messaging:/zeromq:/release-stable/Debian\_9.0/ ./" | sudo tee /etc/apt/sources.list.d/zeromq.list

>> wget https://download.opensuse.org/repositories/network:/messaging:/zeromq:/release-stable/Debian\_9.0/Release.key -O- | sudo apt-key add

Now that the Matrix kernel is installed, we can follow the sample code provided on: <https://matrix-io.github.io/matrix-documentation/matrix-core/python-examples/wakeword/>.



Before executing the code, we need to change the IP address on line #10 to the local IP of the Raspberry Pi. Once that is changed, we can now execute the code, which will make the Matrix actively listening to surrounding sound and will be activated to listen for further commands every time it hears the default wake word “Matrix Start.” Right now, the default behavior of the program is that it will print out the message “I HEARD MATRIX START!” if the wake word is detected. We will customize this behavior once we are successfully configured the EC2 server.

****

## Dealing with EC2 server

The first few steps are very straight forward. We register for an educational account on Amazon Web Services (AWS), which will grant us $100 credit for experiencing a variety of services provided by AWS. We then login to the AWS Console with the created educational account and navigate to EC2 Dashboard. From there, we initialize a “Amazon Deep Learning AMI, Ubuntu” instance on the “t2.small” grade of performance.

1. Setting up environment of the EC2 server

There are two additional required packages we need for the project that are not available by default on the EC2 instance, so we need to execute these commands from the terminal of the instance to install them:

>> pip install opencv-python

>> pip install paramiko

The first command is for installing OpenCV – Python version, which will come in handy when it comes to pre-processing images before feeding them into the CNN we are building. The second command is for installing Paramiko, a Python implementation of SSHv2 protocol that we will use to establish SSH connections to the Raspberry Pi in Loyola building.

1. Establishing connection to the Raspberry Pi at Loyola building to capture image and send back to server

In order to establish a SSH connection to the Raspberry Pi at Loyola, here is the code section we used for the task:

print('establishing an SSH session to 10.8.0.4...')

ssh\_client = paramiko.SSHClient()

ssh\_client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh\_client.connect(hostname='10.8.0.4', username='irwan', password='irwan')

print('connected to 10.8.0.4')

Now we that the code is within an SSH session to the Pi at Loyola, we just have to execute the shell command that will ask the Pi to take a picture:

print('image of parking lot being captured, please wait...')

stdin, stdout, stderr = ssh\_client.exec\_command("raspistill -o image.jpg")

sleep(8)

print('image captured')

And then transfer the image back to the EC2 server:

print('transferring the image back to EC2 instance...')

ftp\_client = ssh\_client.open\_sftp()

ftp\_client.get('/home/irwan/image.jpg', '/home/ubuntu/image.jpg')

ftp\_client.close()

print('the new image is successfully transferred!')

1. Process image with a CNN

Now that the desired image of the parking lot is taken and transferred back to EC2, we want to preprocess this image to be ready to feed it into the CNN. We use the code provided in the documentation of the project:

# Load and process image for the pretrained model

# read and correct colors (swap red and blue channels)

img = cv2.imread('image.jpg')

shape = img.shape

for i in range(shape[0]):

for j in range(shape[1]):

img[i, j, 0], img[i, j, 2] = img[i, j, 2], img[i, j, 0]

# crop an image of 100x100 image at (1604, 1694)

X\_LEFT = 1604

X\_RIGHT = X\_LEFT + 100

Y\_TOP = 1694

Y\_BOTTOM = Y\_TOP + 100

img = img[Y\_TOP:Y\_BOTTOM, X\_LEFT:X\_RIGHT]

img = cv2.resize(img, (225, 225))

# reformat for PyTorch

img = np.moveaxis(img, -1, 0)/255 - 0.5 # normalize values to [-1, 1]

img = torch.Tensor(img).unsqueeze(0)

And now we build the CNN. From the instructions given from the documentation, the structure of the CNN will be described as the code below:

class Net(nn.Module):

def \_\_init\_\_(self):

super(Net, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(in\_channels=3, out\_channels=16, kernel\_size=11, stride=4)

self.pool1 = nn.MaxPool2d(3, 2)

self.conv2 = nn.Conv2d(in\_channels=16, out\_channels=20, kernel\_size=5)

self.pool2 = nn.MaxPool2d(3, 2)

self.conv3 = nn.Conv2d(in\_channels=20, out\_channels=30, kernel\_size=3)

self.pool3 = nn.MaxPool2d(3, 2)

self.fc1 = nn.Linear(30\*3\*3, 48)

self.fc2 = nn.Linear(48, 2)

def forward(self, x):

# Max pooling over a (3, 2) window

x = self.conv1(x)

x = F.relu(x)

x = self.pool1(x)

x = self.conv2(x)

x = F.relu(x)

x = self.pool2(x)

x = self.conv3(x)

x = F.relu(x)

x = self.pool3(x)

x = x.view(-1, 270)

x = self.fc1(x)

x = F.relu(x)

x = self.fc2(x)

x = F.log\_softmax(x, dim=1)

return x

1. Flask webserver entry to the inference result

We follow the sample webserver code provided in the instruction document. The default behavior of the sample code was that any query to the EC2 public IPv4 address at port 5000 will get a return message of “Hello World!” The method that defines this behavior is the hello\_world(). As you can see from the code snippet below, we comment out the default return message, and replace it with the instantiation of the neural net we just defined in the last step. We then feed the processed image in to this net and assign the inference result to a variable called “x.” The inference result is in form of a tensor so we need to compare the first and second values stored in this tensor to determine if the desired parking is available or not. And from there, we return the corresponding message to inform querier the availability of the spot.

# Set up Flask server to return the inference result

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

# Initialize a neural, feed 'img' into it and get inference result

net = Net()

net.load\_state\_dict(torch.load('parking\_lot\_occupancy\_model.pth'))

x = net.forward(img)

if x[0][0] < x[0][1]:

return 'THE SPOT IS TAKEN!'

if x[0][0] > x[0][1]:

return 'THE SPOT IS AVAILABLE'

# return ‘Hello World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

## Modifying the wakeword engine in step 1 to query result from EC2 public IPv4

The default behavior of printing out “I HEARD MATRIX START” whenever the Matrix device detects the phrase “Matrix Start” is defined under the method wakeword\_data\_callback(). We modify this method so that every time the method is invoked, it makes a URL request to the public IPv4 address of the EC2 server at port 5000. This can easily be done by utilizing the package “urllib.” The modified method is as below:

## DATA UPDATE PORT ##

def wakeword\_data\_callback(data):

# Extract data

data = io\_pb2.WakeWordParams().FromString(data[0])

# Log data

print('{0}'.format(data))

# Run actions based on the phrase heard

# CHANGE TO YOUR PHRASE

if data.wake\_word == 'MATRIX START':

x = urllib.urlopen('http://ec2-18-191-226-114.us-east-2.compute.amazonaws.com:5000')

print(x.read())

# print ('I HEARD MATRIX START!\n')

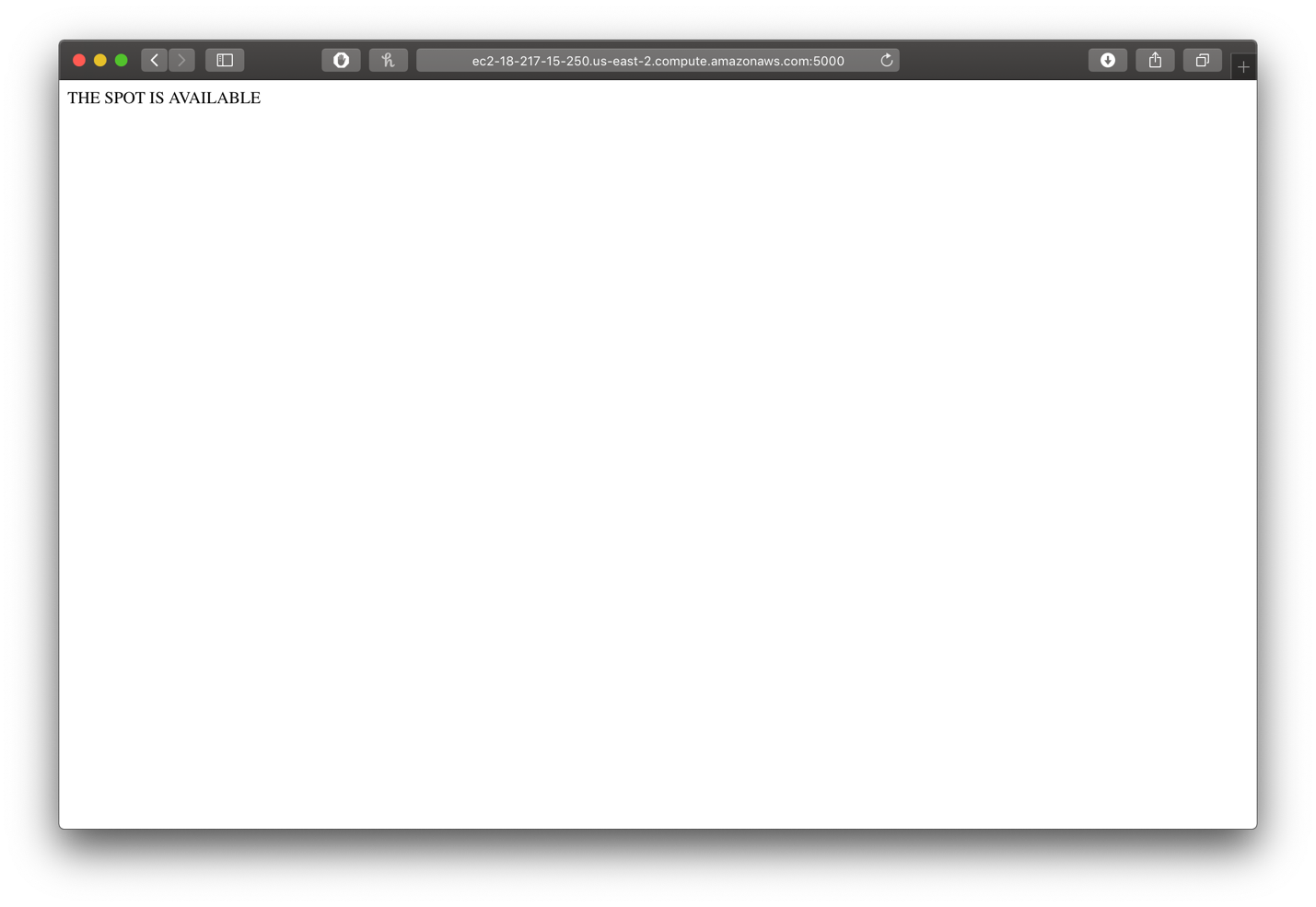
# CHANGE TO YOUR PHRASE

elif data.wake\_word == 'MATRIX STOP':

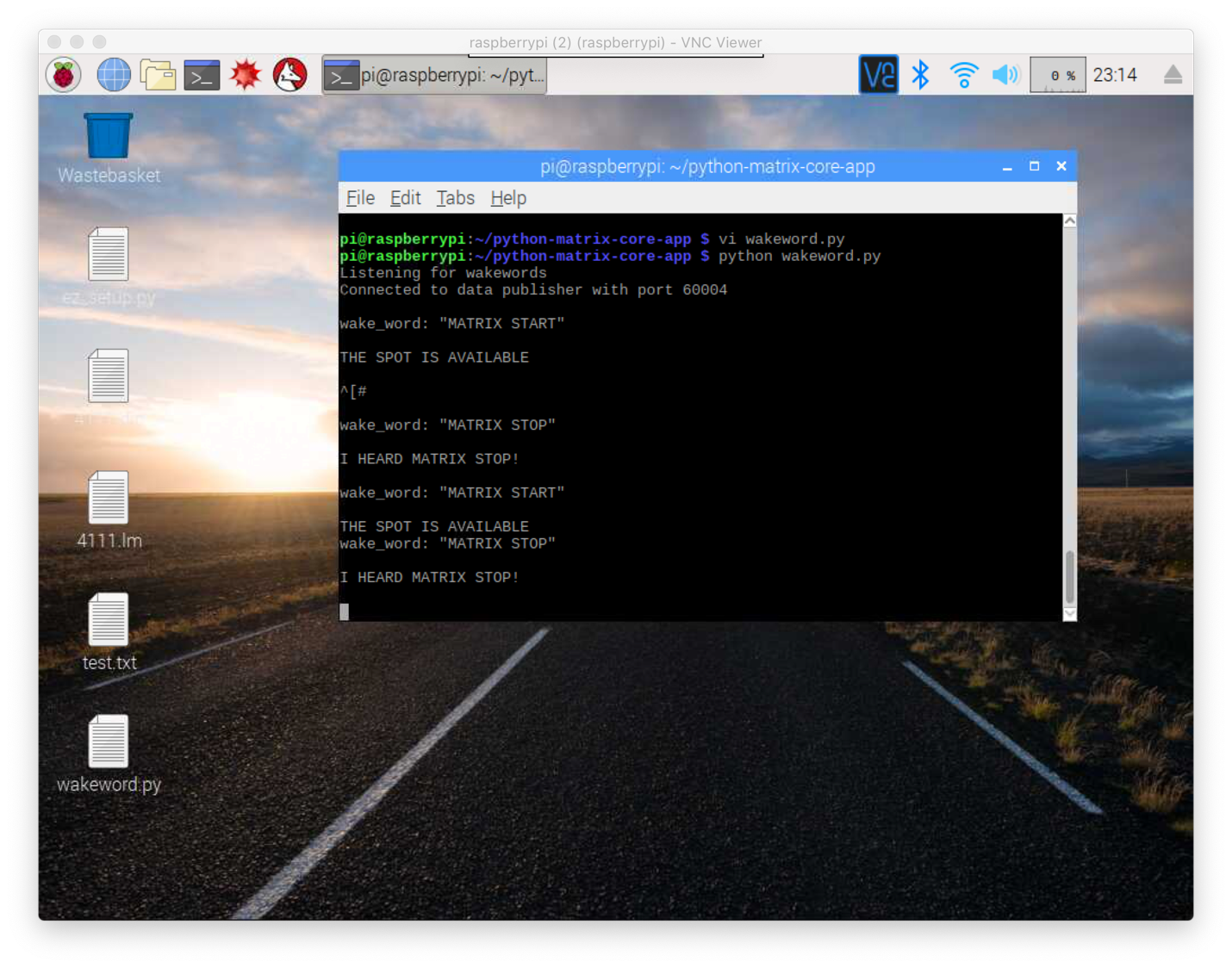
print ('I HEARD MATRIX STOP!\n')

# Results

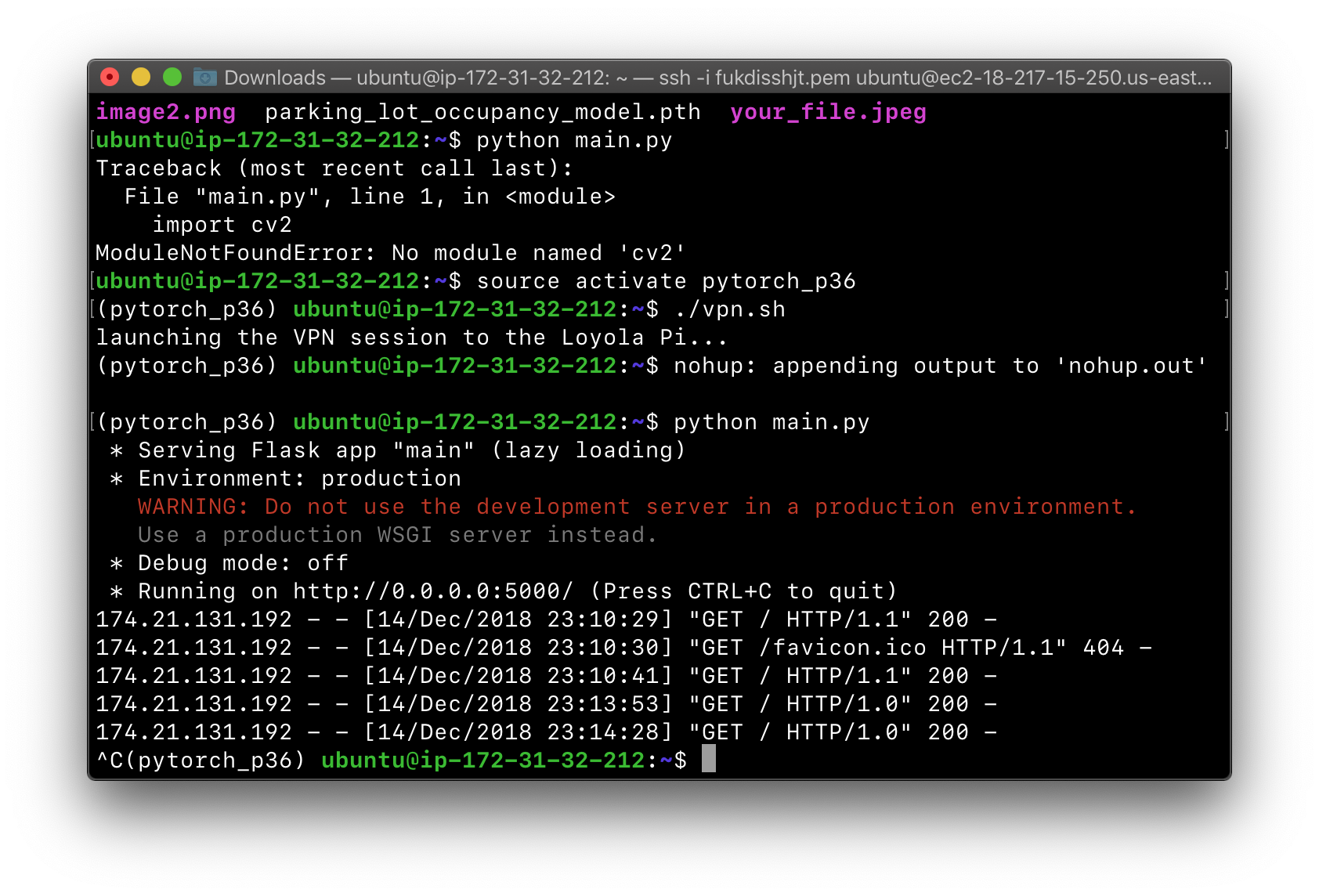
Once our setup is finished, we check the access to the public IPv4 address of the EC2 server one more time before we test the actual setup by accessing the public address through the browser. Here is the response:



That looks good. We then make a couple of queries by saying “Matrix Start” into the Matrix Creator, and got the feedback printed out to the console as follow:



Each time we make a query with the Matrix Creator, we can see that the Flask program running on the EC2 server print out the corresponding requests:



And that is the complete setup of the system.

# Discussion and Conclusion

Overall, the project well accomplished the main objective of introducing the notion of Internet of Things alongside experimenting a simple CNN for the task of identifying the availability of a parking spot.

# References

[1] Keck IoT Project Extra Info.pdf

[2] Keck-Project+Instructions,+Version+2.2.pdf

# Appendix

wakeword.py

## Set Initial Variables ##

import os # Miscellaneous operating system interface

import zmq # Asynchronous messaging framework

import time # Time access and conversions

import sys # System-specific parameters and functions

from matrix\_io.proto.malos.v1 import driver\_pb2 # MATRIX Protocol Buffer driver library

from matrix\_io.proto.malos.v1 import io\_pb2 # MATRIX Protocol Buffer io library

from multiprocessing import Process # Allow for multiple processes at once

from zmq.eventloop import ioloop # Asynchronous events through ZMQ

matrix\_ip = '10.133.0.81' # Local device ip

wakeword\_port = 60001 # Driver Base port

# Handy functions for connecting to the Data Update, & Error port

from utils import register\_data\_callback, register\_error\_callback

# Sphinx Knowledge Base files

LM\_PATH = '/home/pi/python-matrix-core-app/4111.lm'# Language Model File

DIC\_PATH = '/home/pi/python-matrix-core-app/4111.dic'# Dictation File

import urllib

## BASE PORT ##

def config\_socket():

# Define zmq socket

context = zmq.Context()

# Create a Pusher socket

socket = context.socket(zmq.PUSH)

# Connect Pusher to configuration socket

socket.connect('tcp://{0}:{1}'.format(matrix\_ip, wakeword\_port))

# Create a new driver config

config = driver\_pb2.DriverConfig()

# Language Model File

config.wakeword.lm\_path = LM\_PATH

# Dictation File

config.wakeword.dic\_path = DIC\_PATH

# Desired MATRIX microphone

config.wakeword.channel = 8

# Enable verbose option

config.wakeword.enable\_verbose = False

# Send driver configuration through ZMQ socket

socket.send(config.SerializeToString())

print ('Listening for wakewords')

## KEEP-ALIVE PORT ##

def ping\_socket():

# Define zmq socket

context = zmq.Context()

# Create a Pusher socket

ping\_socket = context.socket(zmq.PUSH)

# Connect to the socket

ping\_socket.connect('tcp://{0}:{1}'.format(matrix\_ip, wakeword\_port+1))

# Send a ping every 2 seconds

while True:

ping\_socket.send\_string('')

time.sleep(2)

## ERROR PORT ## (!Currently Experiencing False Errors!)

def wakeword\_error\_callback(error):

# Log error

print('{0}'.format(error))

## DATA UPDATE PORT ##

def wakeword\_data\_callback(data):

# Extract data

data = io\_pb2.WakeWordParams().FromString(data[0])

# Log data

print('{0}'.format(data))

# Run actions based on the phrase heard

# CHANGE TO YOUR PHRASE

if data.wake\_word == 'MATRIX START':

x = urllib.urlopen('http://ec2-18-191-226-114.us-east-2.compute.amazonaws.com:5000')

print(x.read())

# print ('I HEARD MATRIX START!\n')

# CHANGE TO YOUR PHRASE

elif data.wake\_word == 'MATRIX STOP':

print ('I HEARD MATRIX STOP!\n')

## Start Processes ##

if \_\_name\_\_ == '\_\_main\_\_':

# Initiate asynchronous events

ioloop.install()

# Send Base Port configuration

config\_socket()

# Start Error Port connection

#Process(target=register\_error\_callback, args=(wakeword\_error\_callback, matrix\_ip, wakeword\_port)).start()

# Start Data Update Port connection

Process(target=register\_data\_callback, args=(wakeword\_data\_callback, matrix\_ip, wakeword\_port)).start()

# Start Keep-alive Port connection

Process(target=ping\_socket).start()

main.py

import cv2

import numpy as np

import paramiko

from time import sleep

from flask import Flask

import torch

import torch.nn as nn

import torch.nn.functional as F

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Establishing an SSH session to the Pi in Loyola Hall to capture an image

# and load that image back to EC2

#print('establishing an SSH session to 10.8.0.4...')

#ssh\_client = paramiko.SSHClient()

#ssh\_client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

#ssh\_client.connect(hostname='10.8.0.4', username='irwan', password='irwan')

#print('connected to 10.8.0.4')

#

#print('image of parking lot being captured, please wait...')

#stdin, stdout, stderr = ssh\_client.exec\_command("raspistill -o image.jpg")

#sleep(8)

#print('image captured')

#

#print('transferring the image back to EC2 instance...')

#ftp\_client = ssh\_client.open\_sftp()

#ftp\_client.get('/home/irwan/image.jpg', '/home/ubuntu/image.jpg')

#ftp\_client.close()

#print('the new image is successfully transferred!')

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Load and process image for the pretrained model

# read and correct colors (swap red and blue channels)

img = cv2.imread('image.jpg')

shape = img.shape

for i in range(shape[0]):

for j in range(shape[1]):

img[i, j, 0], img[i, j, 2] = img[i, j, 2], img[i, j, 0]

# crop an image of 100x100 image at (1604, 1694)

X\_LEFT = 1604 #1974

X\_RIGHT = X\_LEFT + 100

Y\_TOP = 1694 #1752

Y\_BOTTOM = Y\_TOP + 100

img = img[Y\_TOP:Y\_BOTTOM, X\_LEFT:X\_RIGHT]

img = cv2.resize(img, (225, 225))

#cv2.imwrite('image1.jpg', img)

## reformat for PyTorch

img = np.moveaxis(img, -1, 0)/255 - 0.5 # normalize values to [-1, 1]

img = torch.Tensor(img).unsqueeze(0)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Define the neural net and load the pretrained weights (.pth file) into the

# model

class Net(nn.Module):

def \_\_init\_\_(self):

super(Net, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(in\_channels=3, out\_channels=16, kernel\_size=11, stride=4)

self.pool1 = nn.MaxPool2d(3, 2)

self.conv2 = nn.Conv2d(in\_channels=16, out\_channels=20, kernel\_size=5)

self.pool2 = nn.MaxPool2d(3, 2)

self.conv3 = nn.Conv2d(in\_channels=20, out\_channels=30, kernel\_size=3)

self.pool3 = nn.MaxPool2d(3, 2)

self.fc1 = nn.Linear(30\*3\*3, 48)

self.fc2 = nn.Linear(48, 2)

def forward(self, x):

# Max pooling over a (3, 2) window

x = self.conv1(x)

x = F.relu(x)

x = self.pool1(x)

x = self.conv2(x)

x = F.relu(x)

x = self.pool2(x)

x = self.conv3(x)

x = F.relu(x)

x = self.pool3(x)

x = x.view(-1, 270)

x = self.fc1(x)

x = F.relu(x)

x = self.fc2(x)

x = F.log\_softmax(x, dim=1)

return x

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Set up Flask server to return the inference result

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

# Initialize a neural, feed 'img' into it and get inference result

net = Net()

net.load\_state\_dict(torch.load('parking\_lot\_occupancy\_model.pth'))

x = net.forward(img)

if x[0][0] < x[0][1]:

return 'THE SPOT IS TAKEN!'

if x[0][0] > x[0][1]:

return 'THE SPOT IS AVAILABLE'

# return 'Your in MUAHAHAHAHAHAHAHAHAHAHAHAHAHA!'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)