Text to Speech Support for Visually Impaired People

EEC 284: Final Project

Gaurav Kolhe, Sutej Kulkarni
Department of Electrical and Computer Engineering
University of California
Davis, CA, USA
{gskolhe, sgkulkarni}@ucdavis.edu

ABSTRACT

In today's world, machine learning has become a ubiquitous part of computer systems and applications. We rely more on recognizing and decoding patterns of data, rather than using instructions explicitly to perform a task. Using these patterns of data, we create a mathematical model that we call our 'training set' in order to make predictions work for us. We have leveraged this concept to create a system/application that can detect and communicate a visual text through sound.

1. Introduction

In our application, we send an image from our laptop wirelessly using Bluetooth. A python code from our laptop sends our image in JSON format to the HC-05 Bluetooth module which is connected to our CC3200 launchpad. The CC3200 using HTTP POST will send this data to Amazon Web Services(AWS), on which our ML algorithm running on the AWS side will decode this data and post the decode output image as our device's shadow on AWS. Using the HTTP GET, the CC3200 retrieves this data, and depending on the value obtained, we have a buzzer connected to our board, which will beep 'n' number of times, where 'n' is the number we had sent in our image from our laptop.

2. Objective

- To implement a text to speech support system for visually impaired people using data processing techniques and machine learning algorithms.
- To interface the HC-O5 Bluetooth module sensor and have the ability to perform two-way communication with AWS and implement a machine learning algorithm for processing image data.

Components

1. CC3200 LaunchPad (CC3200-LAUNCHXL)

TI CC3200 is used to process the data coming from the Bluetooth module. It packs the data in JPEG format.

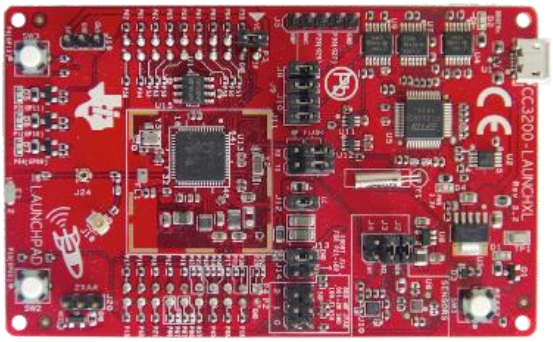


Figure 1. CC3200 Launchpad

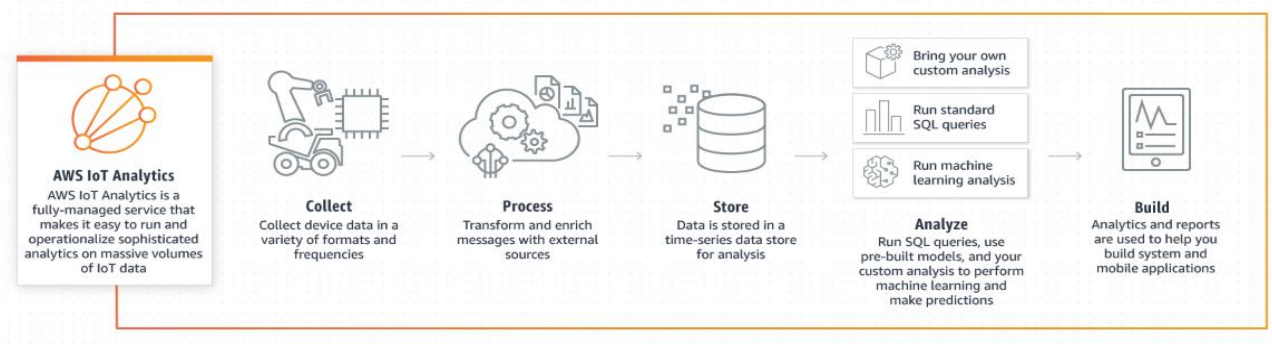


Figure 2. Overview of the IoT Analytics

2. HC-05 Bluetooth module

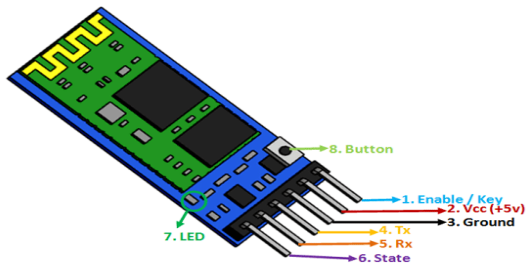


Figure 3. HC-05 Bluetooth module

The pins TX and RX are connected to pins PIN_01 and PIN_02 respectively on the CC3200 for communication.

3. USB-UART cable
4. Breadboard and wires

3. Background

Visually impaired people have a very difficult time navigating to different places and doing day to day tasks. Braille texts are not available at every location which makes it very difficult for them. We have implemented a Machine learning application on AWS, which takes an image as input and can decode, recognize and deliver an output message through audio, and we do this using the TI CC3200 Launchpad.

4. Approach

Figure 4. Algorithm for data processing on CC3200

Algorithm 1 : Processing on CC3200

```
1  : Input   ( JSON format )
2  : Output ( Label )
3  : While (1)
4  :      dataFromBluetooth = Bluetooth_Get ( )
         //Gets the data from   Bluetooth module using I2C.
5  :      http_post( dataFromBluetooth) //upload data to AWS
6  :       sleep(2);                 // sleep before going to http_get
7  :       receive = False;       //We haven't received lable
8  :       While(receive = False)
9  :                 received_data = http_get ( )
10:                 if 'label in received_data:
11:                       receive = True;
12:                 end
13:       end
14:       extracted_label = Parse(received_data)
15:       Ring_Buzzer(extracted_label)
16: end
```

In this section, we discuss the hardware and the software approach in detail. The whole process takes place in 3 steps, the first process captures the image and the custom script is used to convert it to a suitable format.

The next stage is on the CC3200 platform where we use the two-way communication and finally the AWS where we again have the custom script for the machine learning mechanism.

For the sensor, we have used the Bluetooth and for two-way communication, we have leveraged the MQTT where the CC3200 using MQTT and Rest API connects to the AWS and AWS forward this to the Python Notebook.

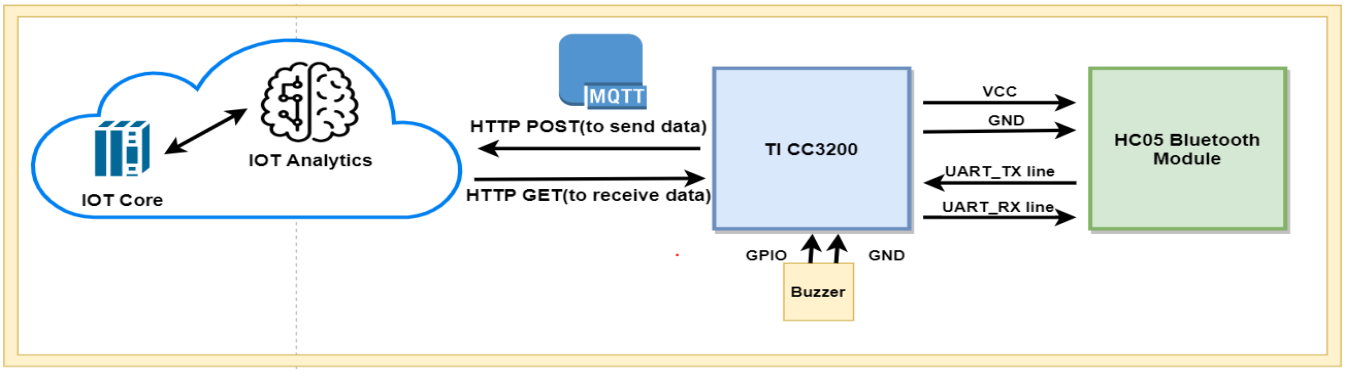


Figure 5. Block Diagram

The other way of connection is python sends the data which goes to the AWS shadow which is conveyed to the CC3200. We have also implemented machine learning on the data using the Convolutional Neural Network.

4.1 Implementation

In this work, we leveraged multiple components that we have discussed earlier. Figure 5 shows the hardware block diagram of our project. As per the very definition of the embedded devices, we use our CC3200 embedded programming module as the component that collects the data and conveys it into the physical world. It acts as a bridge between the software and the hardware domain. Due to lack of resources, not all the processing can be done by the CC3200 however, it can offload this to processors running on the cloud and communicate with it to get the desired results.

The first stage of our prototype is to get the image. We use the paint application on the windows to draw the number. The number is printed on the black background with white colors. Figure 4 shows the sample input that is given to our prototype.

Figure 6. The image of the data being feed into our prototype model

The image is processed before it is being sent over the Bluetooth. We have configured the HC05 module to send the data at 115200 baud. The data is sent to the CC3200 module in chunks of 28 pixels at once. The pixels in the time of sending are also embedded in the JSON format followed by encoding in the byte format. There is a delay of 2 seconds between subsequent payloads to avoid the payload clash at the receiver end, as CC3200 takes the time to process the data and save it. Continuously sending the data, results in a payload crash and the resulting data is invalid.

Upon receiving the data from the Bluetooth module, the CC3200 uses the Restful API to send the data to the AWS shadow. We leverage the HTTP get function which is the part of the API to upload the image as the JSON format over to the shadow. The amazon shadow is a JSON document that is used to store and retrieve current state information for a device. shadow to get and set the state of a device over MQTT or HTTP, regardless of whether the device is connected to the Internet. Each device's shadow is uniquely identified by the name of the corresponding thing, which in our case is XYZ.

The shadow is a part of the IoT core service provided by the AWS. Once the data is forwarded to the AWS shadow, the CC3200 sleeps for 2 seconds before it enters the HTTP get function. The HTTP get function is encased inside a while loop and it only exists when the shadow is updated and it has the keyword ‘label’, when the label is available, in the received payload, the execution gets out of the while loop and it extracts the label that is provided by the machine learning algorithm. The label value ranges from 0 to 9. The CC3200 then buzzes to repeat the number in audible format.

The pseudo algorithm for the processing done on the CC3200 is given in Algorithm 1. On the AWS side upon the reception of the data, we send the data to the IoT Analytics. The IoT Analytics consists of various things such as Channel, Pipeline, Data store and Dataset, and finally analyzation.

4.2 AWS IoT Analytics

In AWS IoT Analytics, the channel bridges the gap between our shadow and the machine learning code. The data is forwarded from the IoT analytics to the channel upon reception of the shadow from the CC3200. We have automated the flow of IoT analytics using the rules. The rule runs the IoT analytics when the shadow is received. The pipeline takes the data from the channel and it routes it to the processing unit. In this processing unit, we extract the received data from the JSON format. The received data is still in the byte representation, which is also converted to the integer format. The data here looks like an array of 28x28 where each pixel range from 0 to 255. The value 255 means the pixel is white while 0 means the pixel is black. The pixel value can vary between 0 to 255. After this processing, the data is stored into the dataset and upon storing this dataset another rule is a trigger that runs the machine learning algorithm for the prediction.

The python program is written in the IoT analytics is embedded inside the container, which executes on its own. It reads the recent data which is stored inside the dataset and it runs the machine learning algorithm to predict the label. Upon prediction of the label this label is put on the shadow, which is then read by the CC3200. To update the shadow, we use the IoT analytics SDK for Python. Using the Function shadow_get and shadow_post we can post and verify if the post went through or not.

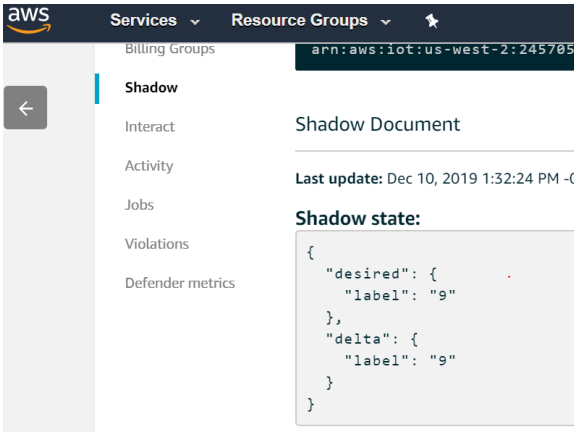


Figure 7: Shadow State on AWS

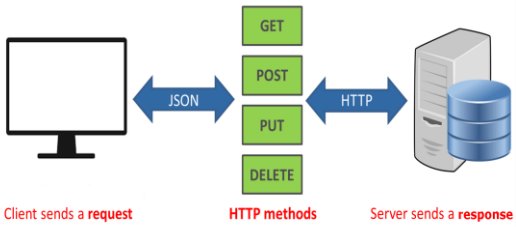


Figure 8: Network communication

4.3 Machine Learning

For the machine learning method, we leverage the Convolutional Neural network for the prediction of the labels. We used the MNIST database (Modified National Institute of Standards and Technology database) which contains the handwritten digits (0 to 9). For the training purpose, it has 60,000 training images and 10,000 testing images each of size 28x28. After the data is ready and preprocessed, we can feed it to the model. For this purpose, we are using the Keras sequential model and the architecture of the model is as follows.

The architecture here is 2 convolution layers followed by the pooling layer, a fully connected layer and softmax layer respectively. Multiple filters are used at each convolution layer, for different types of feature extraction. After both maxpooling and fully connected layers, dropout is introduced as regularization in our model to reduce the over-fitting problem. We also add the optimizer function, loss function and performance metrics to the architecture. we are using categorical_crossentropy loss function as it is a multi-class classification problem. Since all the labels carry similar weight, we prefer accuracy as a performance metric. A popular gradient descent technique called AdaDelta is used for the optimization of the model parameters. The model architecture and the curve of accuracy and loss function are demonstrated in figure x and figure y respectively.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 28)        280
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 28)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4732)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               605824
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 607,394
Trainable params: 607,394
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 9: Architecture of Neural Network

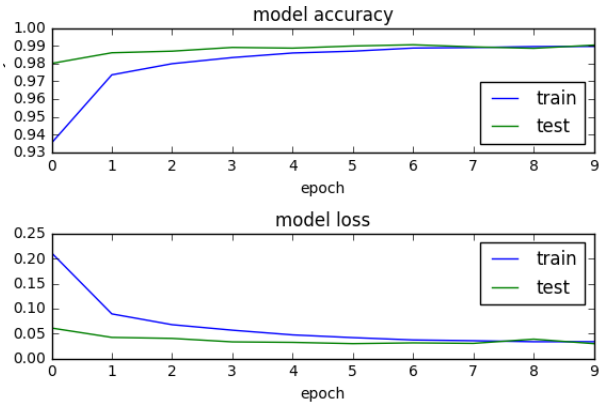
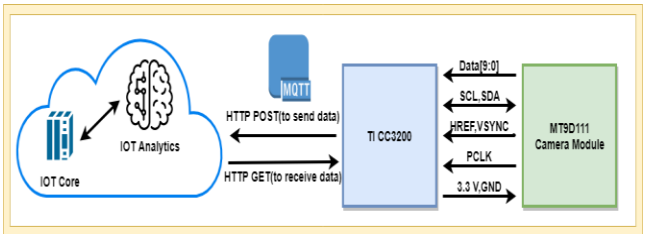


Figure 10: Performance of the ML Model

We save the model to the file to avoid the training phase, every time the shadow is updated the model is loaded, and prediction is performed. This significantly lowers the time for a prediction. As the training data here uses the number written on the black surface, we must follow the same convention to get accurate results.

5. Problems Faced



We were planning to use the camera module MT9D111 to take a picture of a text that we want to read and do processing on it and send it over the network to the cloud**.** But TI did not have enough resources to implement this application as they have discontinued their Websock camera application for the CC3200.

Listed below are some issues we faced while trying to interface this module:

- Conflicting pin mapping with I2C – The only I2C pin set that is available to us on the TI board which is essential to configure the internal camera registers(initialize the camera) does not get mapped during run time and the code breaks
- UART Boot is only restricted to few sets of pins – When we try to boot through UART, we get print logs only through one set of UART pins, but unfortunately, these pins are being used by our camera module

6. Future Scope

This prototype given the time and richer API could perform better. With the ability to add the camera and API to upload the bigger files to the S3 amazon cloud could give this board the capability to detect not only text but also the objects. The machine learning model such as Inception V2 or YOLO (you only look once) can be leveraged. It can help the visually impaired person in various ways such as, at the street crossing, if the pedestrian signal doesn't have an audible notification, the device can be used to detect the oncoming vehicle as well as the status of the signals.

7. Conclusion

In this work, we were able to successfully implement an ML algorithm on AWS using Python, achieve 2-way communication with AWS, run sophisticated analytics using IOT analytics on AWS and successfully implement a Text to Speech support system using the CC3200.

ACKNOWLEDGMENTS

We would like to thank Dr. Soheil Ghiasi and Kourosh for supporting us and providing us with all the necessary tools and the learning material to successfully implement this project.

REFERENCES

[1] Patricia S. Abril and Robert Plant, 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan, 2007), 36-44. DOI: https://doi.org/10.1145/1188913.1188915.

[2] Sten Andler. 1979. Predicate path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '79)*. ACM Press, New York, NY, 226-236. DOI:https://doi.org/10.1145/567752.567774

[3] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. DOI:https://doi.org/10.1007/3-540-09237-4.

[4] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY..

APPENDIX:

A.1 Python Code for Converting the image to 28x28 and connecting to Bluetooth.

In [ ]:
```
# Import all the libraries
from PIL import Image
from scipy.misc import imread
import imageio
import serial,time

# Connect to the bluetooth; the port for the bluetooth can be found from the Device manager.
ser = serial.Serial('COM4')
```

In [ ]:
```
# Open the image that needs to be converted into the 28x28 format
image = Image.open('try.jpg').convert('L')
image.thumbnail((28, 28), Image.ANTIALIAS)
image.save('resize_1.png')
img = imageio.imread('resize_1.png')
img.shape
```

A.2 Python Code to Send the data to the Bluetooth.

In [ ]:
```
# This cell is used to send the data to the bluetooth.
# Data is already embedded in the JSON format for the CC3200.
str_array = []
for i in img:
    for num in i:
        str_array.append(str(num))
index = 0
ser.write(str.encode('{\"state\": {\"desired\" : {\"pythonML\":\"'))
time.sleep(2)
for i in range (28):
    strTosend = []
    for i in range (28):
        strTosend.append(str(str_array[index]))
        index += 1
    print('x'.join(strTosend))
    strTosend = 'x'.join(strTosend)+'x'
    ser.write(str.encode(strTosend))
    time.sleep(2)
ser.write(str.encode('\"}}}\r\n#'))
print("Done Uploading the Data!")
```

A.3 Machine learning Code

In [ ]:
```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Link to Github project: https://github.com/sutejk/Visual_to_speech_support_system