HAND SIGNALS WITH VISUAL DATA

PROGRESS REPORT 2

[CSC 59866 – DD1]
FALL 2019

BY
AYUSHYA AMITABH
CHRIS PANICAN
GERRY XU
OMAR ELNAGDY

Table of Contents

INTRODUCTION

“Hand Signals with Visual Data” is the title of our proposed project that will work to create a software to allow for gesture based controls on the end user’s computer with the aid of deep learning to help recognize the user’s hand as well as the number of fingers being held up by the user.

This report will cover the progress made by the team towards our final project. Through the course of this progress report we will also discuss the division of roles through the team, our achievements, ongoing progress, general project organization, the challenges we have faced, and our future goals.

Our project source code is being managed on GitHub and can be found at:

https://github.com/cpanican/capstone

Team Contacts

Ayushya Amitabh .... aamitab000@citymail.cuny.edu

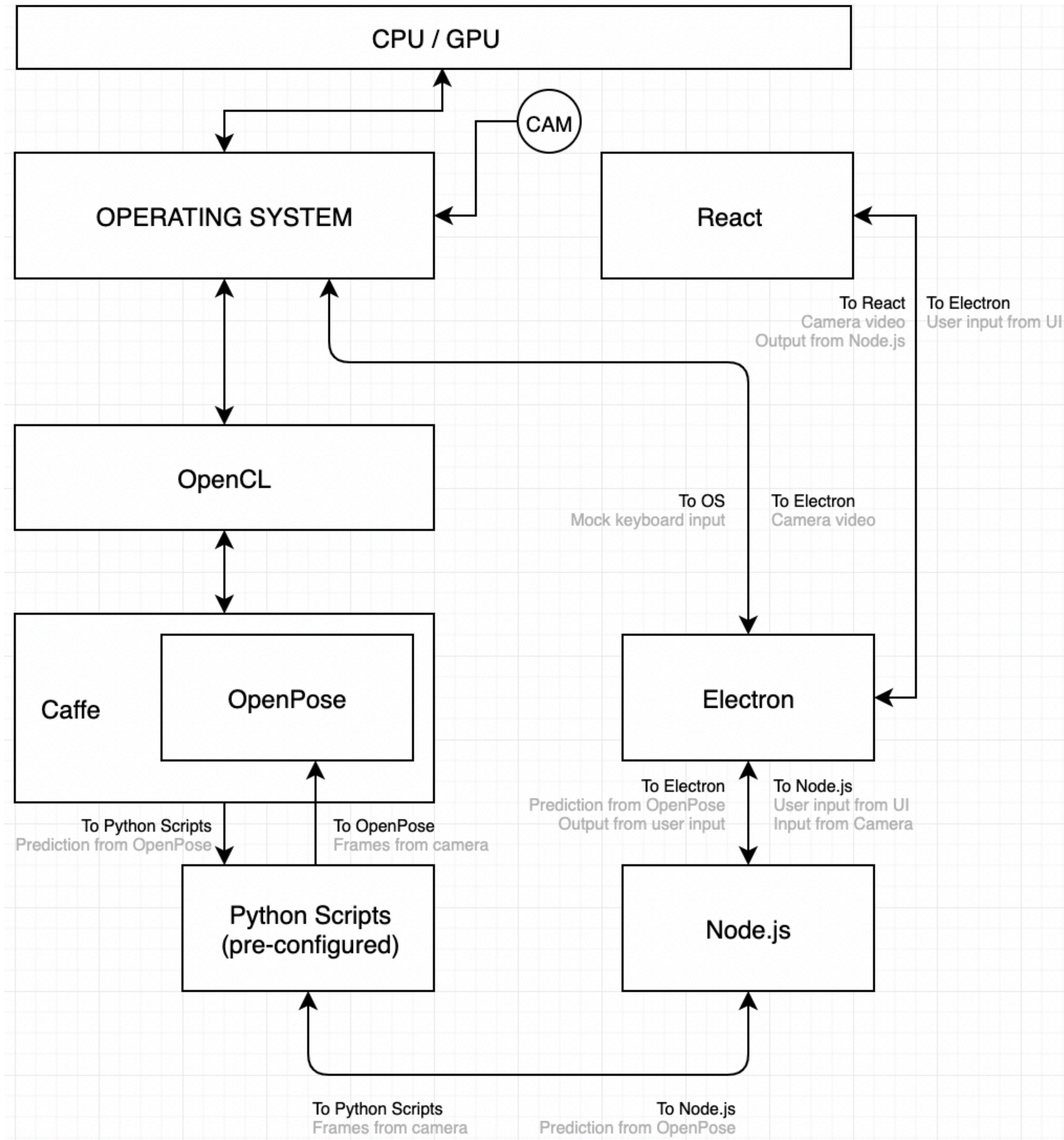
Chris Panican .... cpanica000@citymail.cuny.edu

Gerry Xu .... gxu000@citymail.cuny.edu

Omar Elnagdy .... oelnagd000@citymail.cuny.edu

ARCHITECTURE

The diagram below presents a very basic form of our software architecture:



In the first weeks of development our team has separately worked toward developing the client-side app and the machine learning model. To start off, let's take a closer look at the progress made towards our client-side app.

CLIENT-SIDE APP

As described in the architecture above, we are using a combination of Node.js along with the Electron Framework built on top of Node.js, and React library to manage our front-end DOM. In the first few weeks of development we have established code standards, based off of Airbnb's Es-Lint configurations which can be found on their GitHub JavaScript Packages.

In accordance with Electron's developer guide, our app establishes two main processes – Main and Renderer. The Main process is in charge of handling the communication between the operating system and the Renderer process. The Main process is written using Node.js but will be capable of invoking other scripts if necessary. The Renderer process manages the GUI, although it does have it's own utilities structure which allows for a modular approach to communication between the Renderer and Main processes.

The communication between the Main and Renderer process is done using instances of an Inter Process Communicator (IPC) which is packaged with the Electron framework. The IPC Main is invoked in the Main process while the IPC Renderer is selectively called in the Renderer process.

Below is an example of event listeners being registered in the Main process.

```
/*
File: app/public/electron.js
*/

  app.on('ready', () => {
  // EVENT LISTENERS
      ipcMain.on('get-window', getWindow);
      ipcMain.on('create-settings', createSettings);
      ipcMain.on('save-model', (event, arg) => {
  // …..
      createWindow();
  });
```

Here is an example of the event listener being triggered through the Renderer process.

```
/*
File: app/src/Utilities/Server.js
*/

  getDateModel = async (d) => new Promise((resolve, reject) => {
    ipc.on('save-model-done', (e, reply) => {
      resolve(true);
    });
    ipc.send('save-model', {
      url: this.__url + ENDPOINTS.DOWNLOAD_DATE.url(d),
      method: ENDPOINTS.DOWNLOAD_DATE.method,
    });
  })
```

The invocation of an IPC event allows for replies from the listener in the Main process, this enables a two-way channel between the IPC Main and IPC Renderer. These IPC tunnels are responsible for downloading models and will handle simulating key presses.

SERVER

Our server is also written in Node.js – the main purpose of our server is to provide downloadable models to the client app, this allows for version control of the models and gives the user control over which version they want to use. Our server uses Express.js, a Node.js library that simplifies hosting a REST API using Node.js. The project structure for an active server is the following:

```
.server
+-- index.js                                    - server root
|   +-- uploads/                                - folder containing uploads
|   |   +-- status.json                         - maintains model version info
|   |   +-- {filename}-{datestring}.{extension} - format of saved models
```

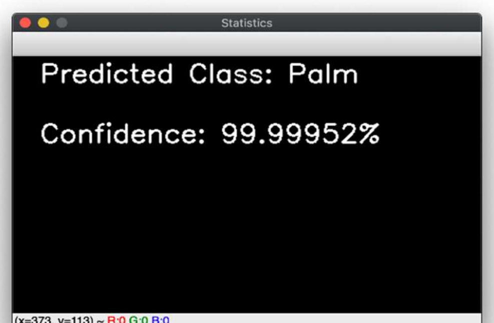
The uploads directory stores the uploaded models and handles the version listing, in order to achieve this, our REST API establishes the following endpoints:

Method	URL	Purpose
GET	/	Checks if server is alive
GET	/upload	UI for uploading new model versions
PUT	/upload	Uploads file with unique name and updates server status
GET	/download/list	Gets list of available versions and lastest version
GET	/download/:date	Downloads specific version, latest if date is invalid
GET	/download	Downloads latest version

DEEP LEARNING

In our project, we will utilize deep learning by gathering visual data from a webcam and use that data to trigger commands in the user's machine. Our group picked CNN because of its ability to process images. Other neural networks, such as RNN, cannot excel in this task.

In the past few weeks of development, our group has made enough progress to show a functioning product. The most notable accomplishment our team has made so far is that we were able to create a Convolutional Neural Network to detect different hand gestures. The program is able to use the user's webcam as input, then it will output a predicted gesture. Shown below are screenshots of our model's prediction input and output.



For our current model we are using Digit/Number classification similar to what we used in the MNIST database classification.

To generate our dataset, we are using a script that continuously extracts frames from a video stream, similar to what the end usage would be using a computers built-in webcam. These frames are then fed into our model trainer which is defined as:

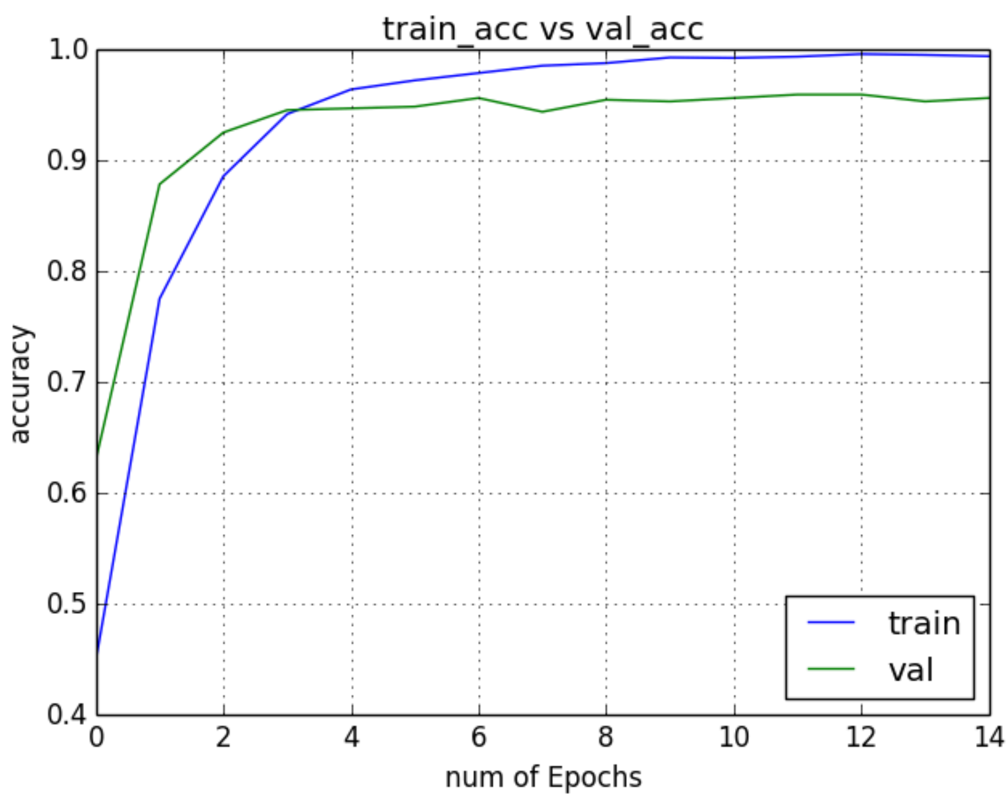
```
model = Sequential()
model.add(Conv2D(nb_filters, (nb_conv, nb_conv),
                    padding='valid',
                    input_shape=(img_channels, img_rows, img_cols)))
convout1 = Activation('relu')
model.add(convout1)
model.add(Conv2D(nb_filters, (nb_conv, nb_conv)))
convout2 = Activation('relu')
model.add(convout2)
model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

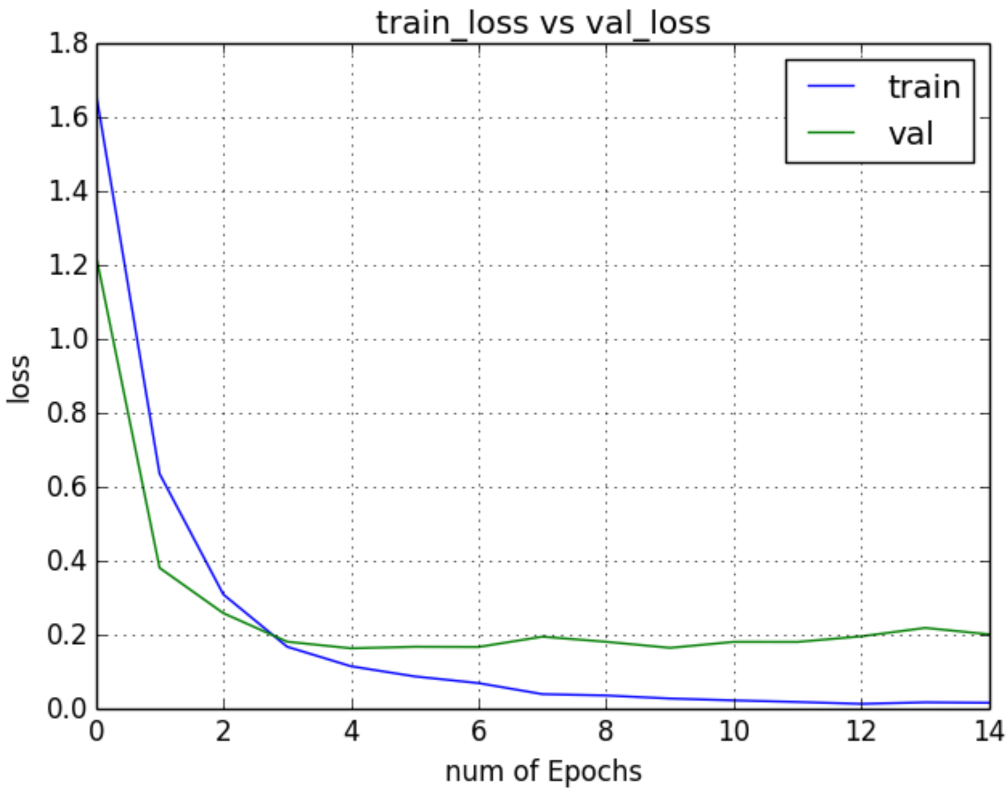
Our model has following twelve layers:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 198, 198)      320
_________________________________________________________________
activation_1 (Activation)    (None, 32, 198, 198)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 196, 196)      9248
_________________________________________________________________
activation_2 (Activation)    (None, 32, 196, 196)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 98, 98)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 98, 98)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 307328)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               39338112
```

```
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645
_________________________________________________________________
activation_4 (Activation)    (None, 5)                 0
=================================================================
```

On training our model for fifteen epochs, we were able to achieve the following accuracy and loss:



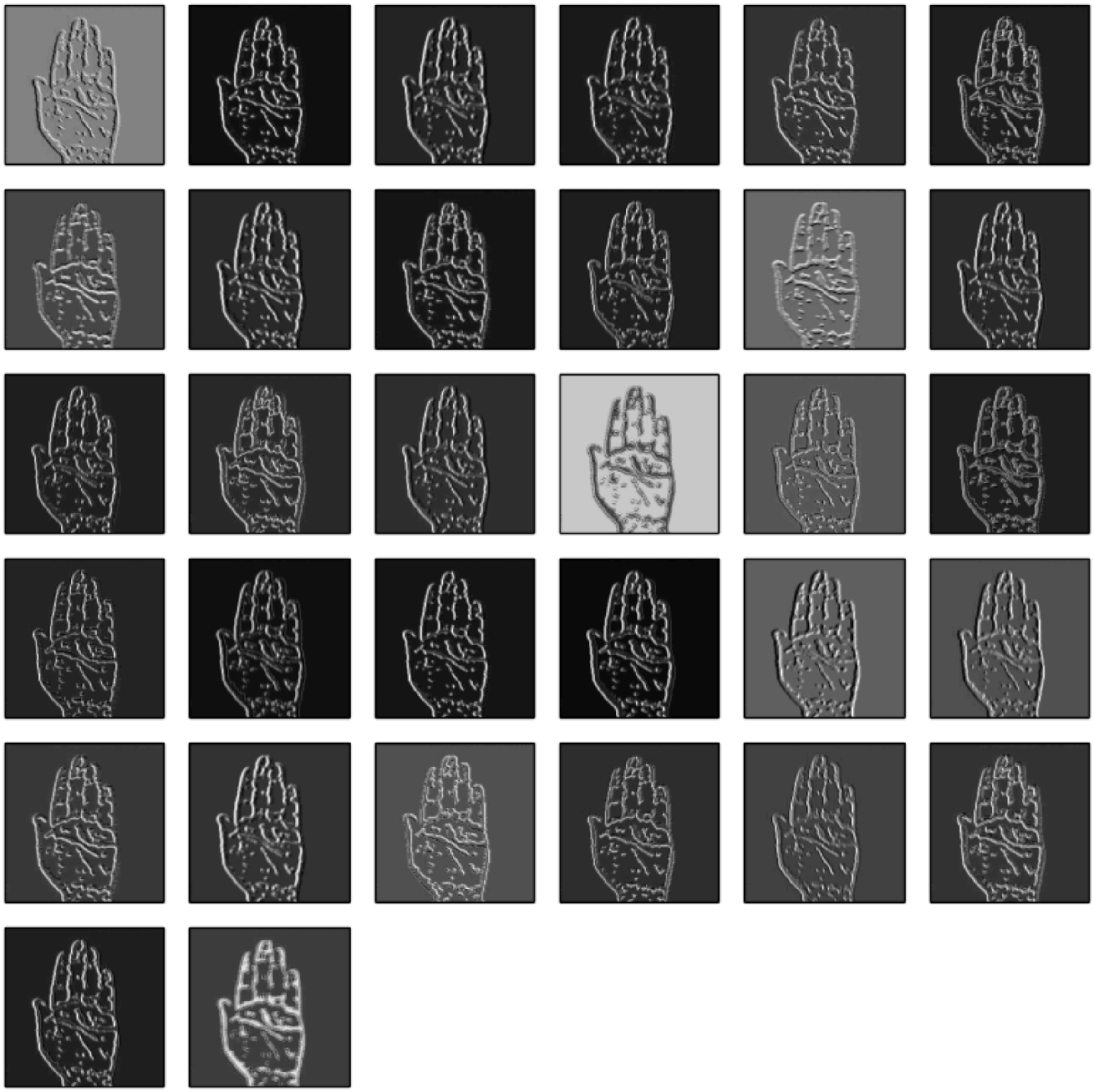


CNN is good in detecting edges and that's why it's useful for image classification kind of problems. In order to understand how the neural net is understanding the different gesture input it's possible to visualize the layer feature map contents.

Using Keras we can visualize the layers:

```
layer = model.layers[layerIndex]

get_activations = K.function([model.layers[0].input, K.learning_phase()],
[layer.output,])
activations = get_activations([input_image, 0])[0]
output_image = activations
```

Here are the visualizations generated for the palm gesture:



CONCLUSION

Challenges

Our team is aware of the fair amount of hurdles we are currently facing in our road ahead. One major issue that we are trying to resolve is having a more optimized model in our program. There will be some overlaps between different classes because we have to account to multiple variables when collecting the data. Users might have different kinds of hardware or permissions that may block us from gathering images. Other notable challenge is moving gestures. Right now, our program can only detect static images but our main goal is to detect movements as well.

Future Goals

Our team plans to tackle the following to-do's after accomplishing our current tasks. We believe that after finishing this list, then we will have minimum viable product:

1. Ability to do actions on user's computer
2. Better gesture detection with camera
    a. Motion detection
    b. Background elimination
3. Easy to use and intuitive interface
4. Documentation

REFERENCES

Deng, L.; Yu, D. (2014).
"Deep Learning: Methods and Applications" (PDF).
Foundations and Trends in Signal Processing.
https://www.microsoft....DeepLearning-NowPublishing-Vol7-SIG-039.pdf

OpenPose
https://github.com/CMU-Perceptual-Computing-Lab/openpose

Handtrack.js
https://github.com/victordibia/handtrack.js/
Library for prototyping real-time hand detection

Gesture Recognition
https://github.com/asingh33/CNNGestureRecognizer
https://github.com/avidLearnerInProgress/hand-gesture-recognition \
Projects with examples for gestures recognition

TensorFlow Examples
https://github.com/tensorflow/examples
General examples from TensorFlow on gesture classification and object recognition

Kaggle Dataset
https://www.kaggle.com/.../hand-gesture-recognition-database-with-cnn/
Database is composed by 10 different hand-gestures

Project Repository
https://github.com/cpanican/capstone
GitHub repository containing the source code for our project