CPS FINAL REPORT

Austin White and Sierra Wyllie

Description of content

1) An **explanation of how to get your code to run and user manual**

Description :

Our project will use Machine - Learning to identify a target individual as they move around in a live camera feed. We will identify the person through facial recognition, and use Convolutional Neural Networks to track the person as they move from frame to frame. Users can choose to identify a face from a .jpg image, webcam stream (so long as the provided url fetches a frame), and from their laptop camera.

The server must be running in order for this to work, and it likely is. If so, simply Shift + Enter the first cell of GUI.nb.
If not, there's quite a process to go through.

For Mac:
go to a terminal window and type:
ssh Austin@161.6.5.177 -p 6240
it will prompt you for a password, just type a space " " and hit enter.
You are now connected to node1.

For Windows:
Press WINDOWS + R and type "cmd" then hit enter/ok. Type:
ssh Austin@161.6.5.177:6240
it will prompt you for a password, just type a space " " and hit enter.
You are now connected to node1.

Then, type "math" , and the wolfram Kernel for Linux should pop up with an input field.
Copy and paste each cell from Server.nb into it, and the server should now be running.

Go back to the GUI file and run it as previously stated.

2) A **detailed description**

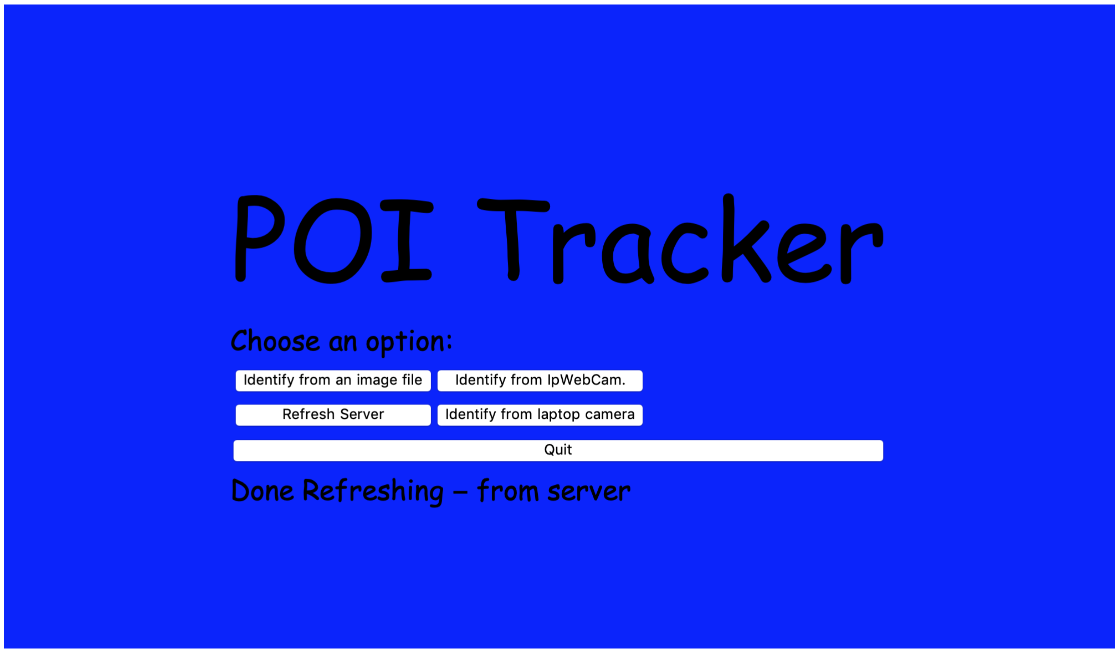
```
In[ ]:= SetDirectory[NotebookDirectory[]]
Out[ ]= /Users/sierrawyllie/Desktop/WhiteWyllieFinalReport/untitled folder
```

Description :

Our project will use Machine - Learning to identify a person as they move around in a live camera feed. We will identify the person through a Convolutional Neural Net. Users can choose to identify a face from a .jpg image, webcam stream (so long as the provided url fetches a frame), and from their laptop camera.

In[ ]:=
```
Import["menu.png"]
```

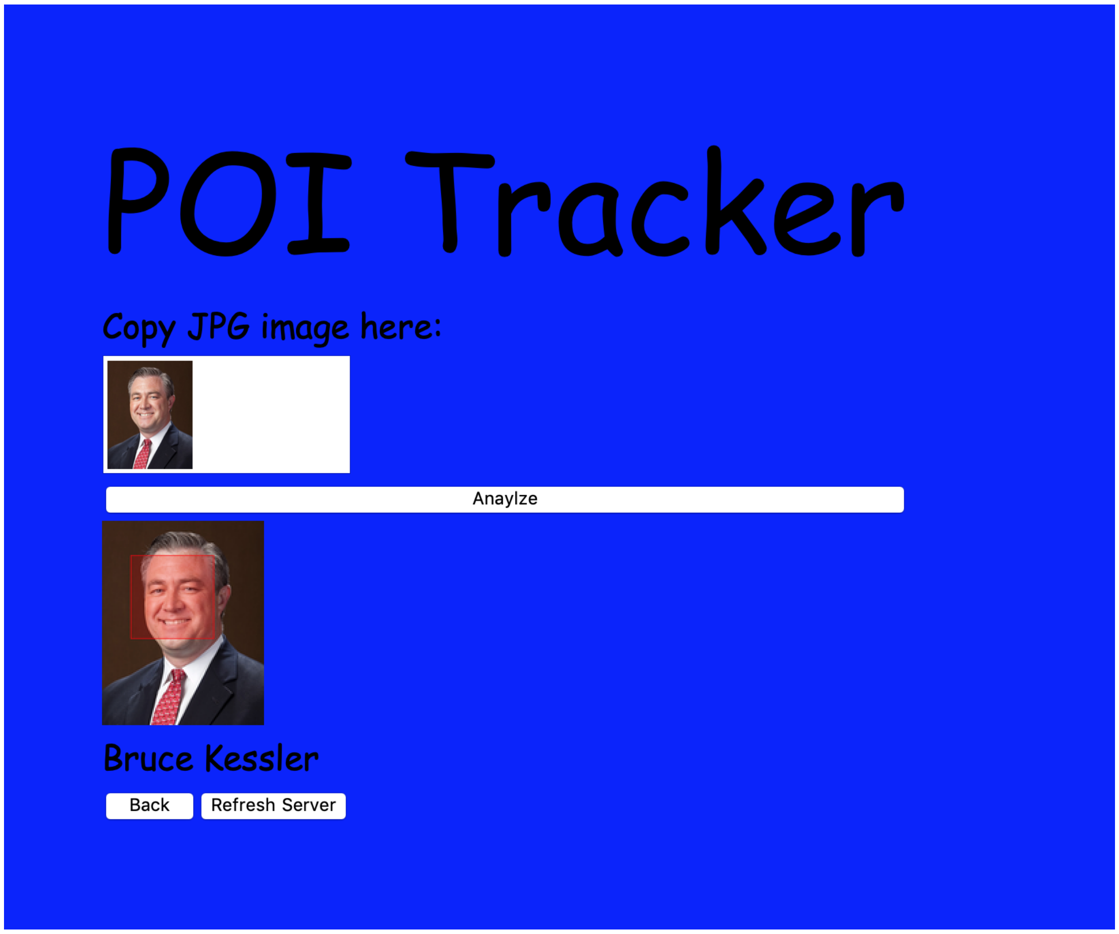
Out[ ]=


POI Tracker
Choose an option:
Identify from an image file
Identify from IpWebCam.
Refresh Server
Identify from laptop camera
Quit
Done Refreshing - from server

Above is the main menu for our GUI.nb.

In[ ]:=
```
Import["jpg.png"]
```

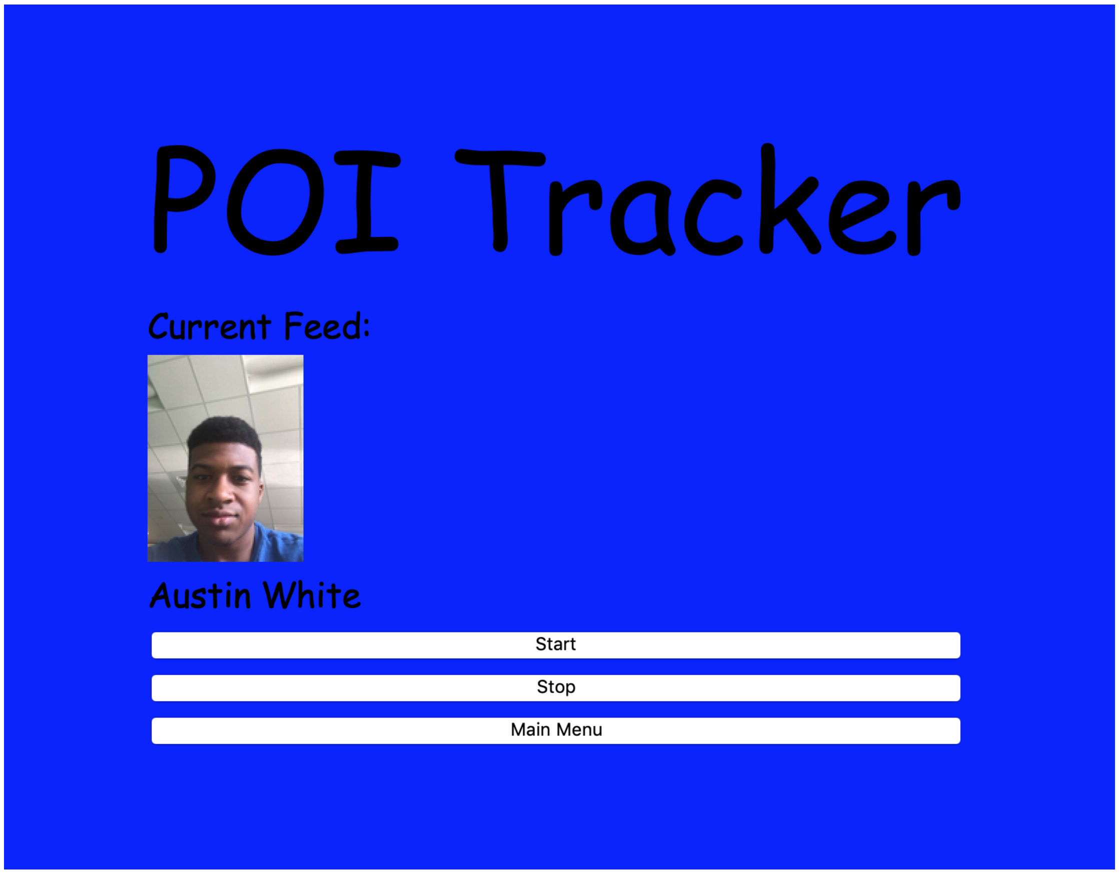
Out[ ]=



The above is an example of a successful identification of Dr. Kessler through an image we found on the Google.

In[ ]:=
```
Import["ipwebcam.png"]
```

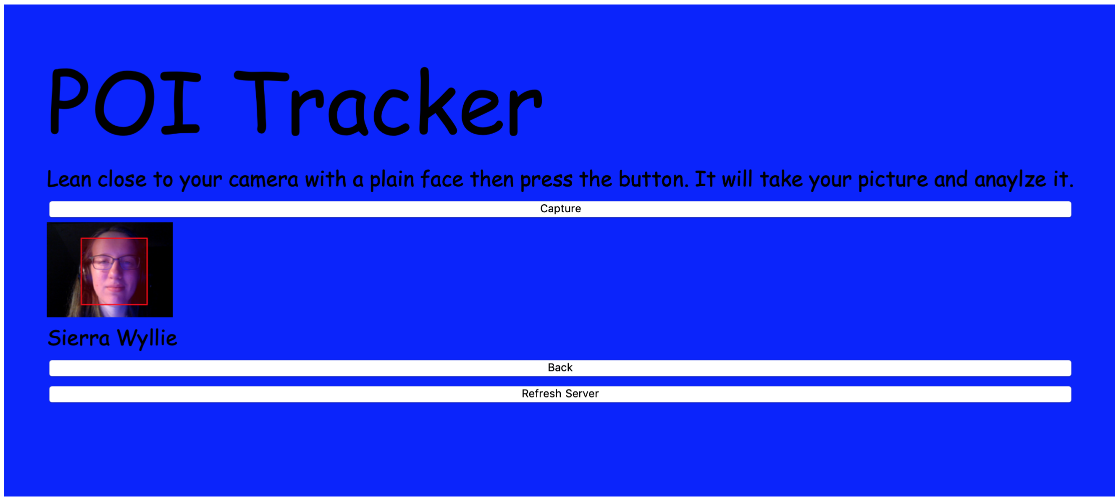
Out[ ]=



The above is a successful identification of Austin from the IPWebCam app on his iPhone.

In[ ]:=
```
Import["Screen Shot 2019-05-09 at 12.41.20 AM.png"]
```

Out[ ]=



Lean close to your camera with a plain face then press the button. It will take your picture and anaylze it.

The above is a successful identification of Sierra from her laptop camera on her mac.

3) An **explanation of algorithms/formulas**

AI viewpoint:

(netowrks and identification)

Our Artificial Intelligence consists of a Convoluted Inception Neural Network (CINN) trained via Siamese triplet loss. Much of the internal functions of CINN lie beyond the scope of this assignment. However, a basic network understanding, training methodology and practical use of the network as an identifier is reasonable to overview.

Our network is formally known as OpenFace Face Recognition Net Trained on CASIA-WebFace and FaceScrub Data, so we call it CINN for short.

CINN is a convoluted neural network that takes input images of size 96x96x3 (RGB) and computes various tensors (often 4-dimensional) to output a 128 bit vector representing the image. The network is heavily influenced by the GoogLeNet Inception Network; in which the image is “projected” against the inside of a sphere in order to recognize features about that image (like a person’s eye, for instance). All neural networks are intelligent, and therefore learn. CINN has to learn roughly 3.5 million parameters, with an additional bout of hyperparameters that the coder chooses to help optimize network performance. These normal parameters are computed via backpropagation, which occurs as a result of the loss function. This loss function generally compares the output of the neural net with the real/correct answer, which is stored in a database. Therefore, by evaluating the network on the database’s “training set”, the network begins to learn the parameters. In our case, the training scenario is quite a bit more complicated.

During training, the network is a Triplet, in which three identical networks evaluate a picture and compare results. Two of the networks evaluate a picture of the same person, and the third evaluates a picture of a different person. The networks evaluating on the same person are call the anchor and positive, whereas the network evaluating the other person’s image is called the negative. The distance between the two output vectors of the anchor and positive networks should be some margin (alpha) smaller than the distance between the anchor and negative output vectors. This is the essence of the triplet loss function, which is mathematically conducted via L-2 Squared Normalization. This process occurs repeatedly until the network’s parameters are well tuned. The network’s parameters are saved after every batch, which occurs at roughly every 200 triplet comparisons. The training is also conducted over epochs, each epoch represents a full cycle through the training set. As such, a higher number of epochs should yield a greater network accuracy. This process generally occurs on the scale of days, which presents additional challenges in networking. Unfortunately, Mathematica 11 and 12 are ill suited for complex training mechanisms. We can currently train a simple network architecture (such as LeNet-5), but are having difficulty with training CINN under the same circumstances. In essence, our network is failing to converge, as loss remains constant at our hyper-parameter alpha. We are certain, however, that our training setup is functional. Mathematica’s limited Computer Vision has caused us to create additional networks that “tag” onto CINN to help train, most notably for creating the triplet system and our triplet loss function. This is not a common training method in any other Computer Vision/ Machine Learning language, but is has so far worked to train LeNet-5. We have, however, been unable to use our own Mathematica-trained network, and instead use OpenFace Face Recognition Net Trained on CASIA-WebFace and FaceScrub Data, which was created and trained by Wolfram in python/OpenCV, then manually converted to Mathematica.

Once the network is trained, we evaluate its parameters on the testing set, which the network has never seen before. The testing is done via a Siamese system, which holds two identical neural networks. Each network is fed a different image of the same person, and must compute output vectors that are within a certain threshold distance of each other. We grade its accuracy over how many pairs the network says are the same person.
In a general application scenario, there is only one network. When this network is fed an image of a person's face (assuming that person is stored in the database), the network computes the vector for that image. It then computes the vectors of all the images in the database, and identifies the individual by finding the closest cluster of images. We decided to expediate this process by storing pre-computed feature vectors in our database, so that we only have to calculate the distance between vectors to make an identification. The closer the distance, the higher likelihood of a matching person.
Identification:
Our ID system (in Server.nb) runs once an image is received. It computes the vector for that image using the Neural Net, then begins looping a random sample of 50 pictures per each individual in the class dataset. The euclidean distance between the vectors are computed for each of these selected class dataset images, and if the distance is less than .6, that person's name is added to a list which holds potential matches. The name with the most repetitions on the list in selected as the final name, and sent back to the local application/GUI. If there are no distance matches, it returns the String "Unknown".

Networking:

Node1 is a server machine so it needs to be constantly active and always listening for a TCP connection on port 1234. The TCP connection works by first initiating a three way handshake. A program (Mathematica) opens a port (1234) and is listening for an attempted connection. When our GUI sends a connection request to node1 on port 1234, the port on the machine running our GUI and port 1234 on node1 will connect using TCP protocol. This socket-to-socket connection is the only way for us to quickly transfer the picture data from the GUI to node1.

4) A description of any files and/or file structure

Our project is extremely structured. We use a PostgreSQL database to store our class dataset which holds a table with the names, id number, and file paths of each individual in the dataset. (We store file paths instead of BYTEA/BLOB rigs of the images because we can store the entire dataset both locally (on laptops) and on node1 (Cloud lab), and because fetching the file is faster than rendering from BYTEA). PostgreSQL is a server/database software hosted by the machine (local or node1), and Mathematica interfaces with it via sockets in port :5432 (usually).
The server will run a kernel code (stored in Server.nb) that allows the server to constantly await input from local applications and maintain communication. The kernel code also identifies the image based on the picture it is given and comparisons to the PostgreSQL dataset, then sends the name back to the GUI.

5) A description of any major data structures

Other than the SQL database, our project makes relatively little use of other structures. During the creation of the database in database.m (which is a one-time part of the project) file paths are pulled from a directory on node1 into a list, then iteratively added into the SQL database to create our class data set.

6) An overview of how your code approaches the problem (with list of functions and their info)

An updated flowchart of our pipeline and approach to the project is below. Essentially, the only services run locally are the user interactions and connection to node1, which does all the heavy lifting. The start application occurs when the program is launched, where the user then must turn on their IPWebCam, then input the IP to connect the application to the webserver that hosts the IPWebCam, which is a third party webserver. The images gathered are then sent to CINN, which will FindFaces, and if there are faces, attempt to identify them. Node1 will then alter the original frame to create a box around the person's face and display their name above that box. This process continues until the user stops it.
We originally decided to use node1 as a network training "muscle machine" that simply executes the CINN network. We then considered the costs of hosting the class dataset of faces on local computers and decided to move as much as possible to node1.

Major preparatory functions :

aaaa.m -a node1 script that creates the dataset in PostgreSQL (psql). Unfortunately, the name of this file must be alphabetically first in our file structure.

Server - not really a notebook file, it begins the server for the application, and must be manually run via wolfram command line kernel on node1. No parameters, no returns.

Major Application Function

GUI.nb - the entire GUI, its sockets, and all the user interactions. No parameters needed, nothing returned.

7) An explanation of externally used code

Our neural network that we use for identification is a wolfram-created neural net provided on their Wolfram Neural Net Repository as OpenFace Face Recognition Net Trained on CASIA-WebFace and FaceScrub Data (we call it CINN for short). This network is unaltered. It is the only outside code we use.

8) A description of the limitations

The analyze from an image only works for .jpeg files, and the stream requires a URL that shows an individual frame as a .jpg, which is one feature of IPWebCam.
The entire project must be internet connected, since it must communicate with the server.
There is one bug, on Mathematica's side, which we cannot figure out, and have notified Wolfram about. FindFaces is the built in function we use to identify the face within the frame. Mathematica sometimes throws an error that says "Image" is not a property of FindFaces. Even though it is. Even though it says so in documentation. We don't pretend to understand this. If you see it, try restarting your kernel, and running the GUI again. If it still doesn't work, try switching to Apple.

We actually ran into several errors where Mathematica refused to accept previously acceptable properties for functions, especially in FindFaces and Scheduled Tasks. We did not update our Mathematica to 12, so we could not find any reason for these weird bugs.

9) A list of references

Courses:

Stanford Machine Learning with Andrew Ng
https://www.coursera.org/learn/machine-learning/
Deep Learning Specialization by deeplearning.ai with Andrew Ng
https://www.coursera.org/specializations/deep-learning
CS 180, 221, 371

Papers/Books:

Computer Networking (Fifth Edition) by Kurose and Ross
Going Deeper with Convolutions by {Christian Szegedy1, Wei Liu2, Yangqing Jia1, Pierre Sermanet1, Scott Reed3,Dragomir Anguelov1, Dumitru Erhan1,Vincent Vanhoucke1, Andrew Rabinovich4} (Were the CINN inspiration came from)
Learning PostgreSQL 11 by Juba and Volkov
Deep Learning with Python by Francois Chollet

Code:

Wolfram Documentation Center
PostgreSQL documentation https://www.postgresql.org/docs/
The neural network we use:
https://resources.wolframcloud.com/NeuralNetRepository/resources/OpenFace-Face-

Recognition-Net-Trained-on-CASIA-WebFace-and-FaceScrub-Data

Consults:

- Dr. Wang
- Dr. Michael Galloway
- Ismael Abumuhfouz
- Dr. Uta Ziegler

Software:

- Mathematica (unfortunately)
- PostgreSQL (Linux and macOS)
- postgres.app (GUI for macOS, not required to run SQL database)
- WinSCP (for windows terminal functions, used only as a development tool)
- WolframScript (very unfortunately)
- vim or nano (used to edit/view .wls scripts)

Signed Statements are included in the READMES folder.