Final Report: The Average Joe Classifier

**Section One**

- Allison DeCicco: Allison was responsible for the User Interface design and the connection between the server and client side. This dealt with the HTML, CSS and node.js side of the project.
- Billy Miller: Billy was responsible for the machine learning part of the project. This involved data processing and cleaning to prepare it to be ingested by the machine learning model as well as the set up and training of the machine learning model. All of this was done using Python.
- Lian Showl: Lian was responsible for the annotation tool and labeling the images using bounding boxes to then be passed to the database. The annotation source code is from an open source package called Annotorious which primarily uses JavaScript and HTML.

**Section Two: Description**

Millions of cameras are installed in houses, businesses and on street corners, recording millions of images all day. When a crime is committed, law enforcement searches for the person or object they are looking for by manually reviewing hundreds of hours of video surveillance footage. However, with our product, The Average Joe Classifier, there is a way to drastically reduce the amount of time someone spends doing this task. Using this product, the officer could teach the system what he wants it to identify and upload the footage. The system will analyze the footage and label every time the specific object that it was told to find appears in the footage. This will drastically reduce the time spent doing this task and lead to solving crimes much faster. Tracking multiple objects through video is a vital issue in computer vision. It can be used in various video analysis scenarios, such as visual surveillance, sports analysis, robotic navigation, autonomous driving and medical visualization. The data that can be collected from object detection is a huge untapped potential for data analytics and could lead to advancements within many different fields.

The Average Joe Classifier will automatically identify a wide variety of objects for the user. Most cameras now track an object, but they do not identify what the object is. No product exists, as of now, that can label tracked objects. Current products for object detection have trained classifiers for specific objects. Object classifiers that currently exist are mainly for vehicles because of the abundance of traffic cameras, but it has never been expanded to have any person, whether he is a trained computer scientist or not, to classify whatever the user is interested in.

**Section Three: Website link**

**https://gw-cs-sd.github.io/sd-20-decicco-miller-showl/**

**Section Four: Libraries needed, APIs and packages used**

- Annotorious: https://annotorious.github.io/
- Node.js libraries: ejs, body-parser, mongoose, multer, express, multer gridfs-storage, gridfs-stream, method-override
- Python Libraries: Pymongo, Bson, Numpy, Gridfs, Opencv
- Darknet: http://pjreddie.com/darknet/

**Section Five: Technical Overview**

The Average Joe Classifier is a tool that takes a complicated idea such as Machine Learning and makes it simple for anyone to use. The goal is for the user to create an object classifier with less time commitment usually required and without requiring thousands of images. It is based off of the YOLO algorithm. In the paper *YOLOv3: An Incremental Improvement*, the author explains the advancements in the classifier that will make this algorithm superior to other choices. Yolo uses neural networks to classify the image. It contains 54 convolutional layers. When it is trained on a set of images, it uses weights to classify the image. Using YOLO, allows the user to have to submit less images and have to commit less time to creating a classifier. This makes our product user friendly.

There is a friendly user interface created using HTML. This is connected to our backend using node.js. We are using a MongoDB database. This allows for easy storage of the images the user is uploading.

**Section Six: "If I had to do this again"**

Allison: One thing I would do differently is take the initial set-up of the UI more seriously. We had to take a lot of time to reset up the UI as it got more complicated. I set up the initial interface with a template and did not have a complete understanding of the way the div tags were working. If I had taken time to understand the basic design, it would have been easier to update the UI and make it more complicated instead of having to relearn it.

Billy: One thing I would change if we did this project again is try to consolidate the number of different technologies used. We had 4 main components of the project (database, Node website, python scripts, and machine learning library) which led to a lot of communication between them which became difficult to manage at times. If we had done something as simple as using a python library, such as flask, for the website, instead of Node, that layer of communication would have been eliminated and would have streamlined the whole process.

Lian: Initially, we went full force into the project, doing what we could on what limited knowledge we had on the technologies we would be using. We feel we could have taken more time to map out and learn all the new technical aspects of the project early on to prevent problems in the future. Additionally, when researching the steps in how to implement our object

classifier project, I believe that grasping the high level picture is important, however, it was difficult to map the small in between steps when I only had the high level picture of what needed to happen. The small steps in finding an adaptable UI, finding a way to make it user-friendly enough for a typical user to navigate, to then connecting that to the database was harder in the long run when I was focusing on the big portions of the project (UI to Python to database).

**Section Seven: Instructions and what's next**

Setup: Need python3/pip3, node/npm, MySQL, wget (optional but easier)

1. git clone https://github.com/gw-cs-sd/sd-20-decicco-miller-showl.git
2. cd sd-20-decicco-miller-showl
3. pip3 -m venv env OR virtualenv env OR however you would create a python virtual environment
4. source env/bin/activate OR however you would activate your python virtual environment
5. pip3 install -r requirements.txt --user
6. cd app
7. npm install
8. cd darknet
9. make
    a. if running with GPU (required for training to not take a full day) change Makefile to GPU=1
10. wget https://pjreddie.com/media/files/yolov3-tiny.weights
    a. or click the link, download the file, and put it in the darknet directory
11. ./darknet partial cfg/yolov3-tiny.cfg yolov3-tiny.weights yolov3-tiny.conv.15 15
12. cd ..
13. Make sure MySQL is running locally (server is restarted frequently so that on will likely not be running)
14. add // to beginning of line 20 in app.js and remove // from beginning of line 21
15. add # to beginning of line 12 in darknet.py and remove # from beginning of line 11
16. npm start
17. navigate to localhost:1500

What works: Everything we set out to do with our project works. As far as the process of uploading, labeling, and retraining goes the process is solid. The thing that could be improved is the accuracy of the computer vision. The model will come out really well on high quality images from a fixed camera, but the quality will drop off quickly as the dataset gets more difficult. This is partially because we are trading some accuracy for more speed, but this is also partially the nature of computer vision models. The lower the image quality and the more variation between images the harder it will be to train. By using more sophisticated computer vision techniques this may be able to be improved.

Where to go from here: One idea we had to make the project a little better is to change the input and output of the system. Right now the user uploads a bunch of images and in the end they finish with labels for all of those images and a place to upload more images to be labeled with their model. The average person probably does not have a dataset of images to upload. They are far more likely to want to find someone or something in a video. So one way to improve upon the project would be to have the user upload a video instead of images, which the system would then split into frames to use as the training data. YOLO is then capable of labeling a video in real time so the output could be their video with live labels. We were not able to do this because of resource constraints (labeling of a video requires OpenCV which was not installed on the server we were using and is notoriously difficult to install so we were not allowed to add it).

With or without the incorporation of videos the best way to build on the Average Joe Classifier is to incorporate it into a product. Think of something like a Ring doorbell, but the user has the ability to go to the app/website, see their footage from their device, and create classifiers for objects commonly seen in the camera. This would allow them to get customized alerts in real time based on the models they have created and who/what their doorbell sees. This is just one example of product integration, but using the Average Joe Classifier in combination with another product would be a good next step.