Social Spaces CS 404 Project

Seth Lunders
lund4272@vandals.uidaho.edu

ABSTRACT

The *Social Spaces* project is designed to allow users to see the activity in a social space over time. It uses machine vision to identify moving objects in the scene, then machine learning to classify those objects. Using Flask and ChartJS, the data collected is displayed in a simple chart to the user.

1 INTRODUCTION

The design requirements for the project included some form of machine vision, machine learning, and something else such as API calls to add complexity. I wanted to create something that could potential be useful outside of this class, so I came up with a website to track activity in social spaces. This could be used by businesses, governments, or individuals to track when and how spaces are used. This information could be used to inform decisions on what to do to improve those spaces.

Tools and Software

I used git + GitHub, to keep track of files and changes to the project. I used Flask for the web server back-end, and CSS for styling the HTML on the front-end. I used a CSV file to store the data, as it is easy to set up and get working with Flask and ChartJS. I used Visual Studio Code for writing the code. For the machine vision sections, OpenCV was used. The neural network was implemented in Keras. Overleaf was used for this report.
Here I will outline some of the helpful features of each of the tools I used.

git & GitHub:
Git was extremely useful for the product. It helped us keep track of files and changes in case I broke something.

Flask:
Flask was the most important part for display the data to the user. It creates the web server for displaying the site.

HTML, CSS, & Jinja: Flask uses Jinja as the templating engine. Since it's just a single page project, this wasn't terribly necessary but it was straightforward to use. It uses basic HTML and CSS, while allowing for some Python-like code.

ChartJS
ChartJS is a JavaScript library for making charts. I followed this guide https://towardsdatascience.com/flask-and-chart-js-tutorial-i-d33e05fba845 to learn how to set up a chart to read from a CSV.

Visual Studio Code
All of the programming for the site was done in Visual Studio Code. The code formatting, linting, and syntax highlighting were all nice to have.

Overleaf
Overleaf is an online LaTeX editor and viewer. It what was used to write and format this report.

OpenCV
OpenCV was used for a few things. First, to read from the webcam. I was also able to use its image processing functions to isolate moving objects in the image, to be fed into the Neural Network.

TensorFlow/Keras
TensorFlow, via Keras, made making a neural network fairly straightforward. It reads the images captured by OpenCV and attempts to classify them as human or non-human.

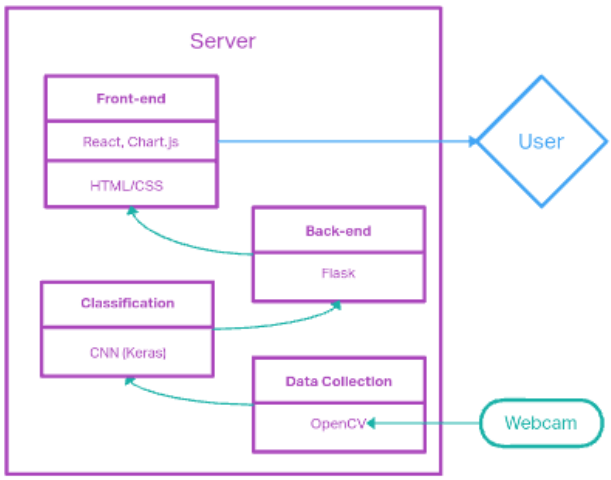
Google Colaboratory
Google Colab was very helpful in training the neural network. My laptop that I would normally use (that has a GPU) is out of commission, so using Colab reduced training times by about 10x.

Firefox & Brave
I tested the site in both Firefox and Brave (a chromium-based browser).

Linux and Miniconda
I developed on Linux with VS Code. I used Miniconda to create a virtual environment to install all the python libraries needed.

2 DESIGN & METHODS

Overview:
The image below shows an overview of the application design and flow of information. The Webcam captures images of moving objects via OpenCV. These images are fed to the neural network to classify. The website, via Flask, then can display this information to the user. In this section I'll go a bit more in depth into each of these steps.



OpenCV:
Here are the steps for capturing the moving objects with OpenCV:

- Open Webcam.
- While program is running, read frames
- Preprocess frame (blur, grayscale)
- Close image
- Use OpenCV's background remover
- Close the image again. This seemed to give me the best results.
- Use OpenCV's getConnectedComponentsWithStats function to find a list of objects in the image.
- Sort the list by size of object to get the most prominent ones.
- Every specified amount of time, a separate script tells this one to save the cropped images of each object visible.

CNN/Keras:
The neural network uses both fully connected and convolutional layers to classify images. Here is the summary of the model.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_5 (Rescaling)     (None, 256, 256, 3)       0

 conv2d_9 (Conv2D)           (None, 256, 256, 32)      896

 max_pooling2d_9 (MaxPooling  (None, 128, 128, 32)     0
 2D)

 dropout_6 (Dropout)         (None, 128, 128, 32)      0

 conv2d_10 (Conv2D)          (None, 126, 126, 64)      18496

 max_pooling2d_10 (MaxPoolin  (None, 63, 63, 64)       0
 g2D)

 dropout_7 (Dropout)         (None, 63, 63, 64)        0

 conv2d_11 (Conv2D)          (None, 61, 61, 128)       73856

 max_pooling2d_11 (MaxPoolin  (None, 30, 30, 128)      0
 g2D)

 dropout_8 (Dropout)         (None, 30, 30, 128)       0

 flatten_3 (Flatten)         (None, 115200)            0

 dense_12 (Dense)            (None, 32)                3686432

 dropout_9 (Dropout)         (None, 32)                0

 dense_13 (Dense)            (None, 64)                2112

 dropout_10 (Dropout)        (None, 64)                0

 dense_14 (Dense)            (None, 128)               8320

 dropout_11 (Dropout)        (None, 128)               0

 dense_15 (Dense)            (None, 8)                 1032

=================================================================
Total params: 3,791,144
Trainable params: 3,791,144
Non-trainable params: 0
```

It achieved around 80% accuracy on the 5 classes it was trained on. I believe this is partly because it classifies cats as dogs most of the time. After I noticed this, I didn't have time to retrain it, but I believe just taking cats out would improve the accuracy.
After the OpenCV script saves the images of the objects to a folder, it tells the neural network to classify those images. The predictions are stored to the CSV that the web server will use.

Flask:
As I'm writing this paper, I realize that Flask wasn't strictly necessary to create a viewable website. I *could've* just used plain HTML & CSS, as the website isn't dynamic. However, it does open

up the possibility for adding dynamic functionality to the site in the future. At the very least, it was something I wanted to learn more about anyways.

Front-End:
ChartJS was very helpful for creating a nice-looking graph of the data. I just had to specify the CSV file to read from and what columns were used for the X and Y axes. With more time, I'd like to add in a filter functionality, so users could look at different spans of time.

3 Conclusion and Future Developments

Due to the wide variety of technologies to learn and the limited amount of time, each component has things that could be improved on.

First of all, the OpenCV portion does not detect still objects. This means that though it could detect activity on something like a street or crosswalk, it would have a harder time tracking people in, say, a coffee shop. A better way of identifying people in the image would be useful, possibly something like YOLO could be used.

For the machine learning portion, a lot of the trouble it has is just that the OpenCV portion doesn't always identify objects well. Furthermore, it doesn't have a 'negative' class. It can identify humans, dogs, cats (as dogs mostly), cars and motorcycles if given a decent photo, but it can't identify a picture *without* those things. This means if the OpenCV code mistakenly captures an image of something with nothing in it, it will still classify it as one of the 5 classes it was trained on.

For the Flask portion of the project, it would be nice to implement some functionality that made use of what you can do in Flask. For example, it would be great if there was a way for other users to create an account and start uploading activity for a space they want to track.

Finally, it would be nice if there were some more useful options on the chart. Currently, it displays all the data available. It could be useful, especially as the time tracked increases, to look at counts on a hour by hour, day by day, week by week, or even month by month basis.

Conclusion

There's still a lot of room for improvement on the site, but still I'm pretty happy with how it's turned out. Though I haven't had time to implement all the functionality *and* make it look nice, I've learned a lot about how I could integrate Machine Vision, Machine Learning, and dynamic web applications with each other and even other technologies in future projects.

Here is how the final UI turned out: