**The computer vision capstone project**

**OVERVIEW**

The project exists to provide New York City concrete examples of how computer vision can be used to count pedestrians. The code provided will give the user the ability to begin the project from the very beginning with scraping images from the web, to very end with visualizing the results. It will allow the user to use three techniques (blob detection, Histogram of oriented gradients, and convolutional neural networks) to train models for the purpose of counting pedestrians.

A combination of Linux, Unix, and Ubuntu machines were used to during the project. Furthermore, Python 2.7 and Lua the programming languages used in the project.

**Files**

The only data/files used in the project were the images scraped from the DOT video feeds. These can be found in the dot\_images folder on the project workspace. They are open source and they do not contain any confidential information.

**Process**

Our project took on multiple phases. The phases and the code folders are described below:

1. Scrape images from the NYV Department of Transportation (DOT) video feeds.
   1. scraper
2. Scrape the time and date off of the images
   1. get\_time
3. Rename the images with the extract time and date
   1. Get\_time
4. Create database
   1. db
5. Link the dot\_images folder to the database
   1. db
6. Design label tool and then label images
   1. LabelingTool
7. Create models for counting pedestrians—blob detection, Histogram of oriented gradients, and Convolutional Neural Networks (CNN)
   1. HOG
   2. nn
8. Build visualization to show results
   1. webpage

**Phase 1. Image Scaping**

Our project needed a lot of images to train the models with. In total we scraped ~500,000 images from the DOT traffic feeds. This took around 8 hours; but the code can be configured to run for as long as needed. The scraper folder in contains all the necessary code to run collect images; the vid\_scrape.py will initiate the scraping process. Multithreading was used to collect images at very close time intervals, otherwise the time between images at the same intersection would be too big.

**Phase 2. Scrape the time and date off the images**

In order to preserve the time and date of when the image was collected, a script was written to extract the pertinent information from the image. The run\_gettime.py file will do this and can be found in the get\_time folder. It should be run during the image scraping phase. The script is wrapped to the scraper script so no additional steps are necessary.

**Phase 3. Rename the images with the extract time and date**

Following section 2 the image would be renamed with the appropriate time and date that was extracted from the previous step. The run\_gettime.py file from get\_time folder is also in the scraper folder as well, so when the scraper script is running it will rename the image immediately.

**Phase 4. Create database**

This can be done by running the createddb.py script in the db folder.It will create three different tables—the camera table, image table, and label table. The tables are shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE I CAMERAS TABLE DATA DICTIONARY** | | | | |
| **Column Name** | **Optional** | **Format** | **Length** | **Description** |
| id | N | INT | - | Unique camera id (Serial) |
| name | Y | VARCHAR | 50 | Camera name |
| description | N | VARCHAR | 200 | Description of the camera |
| lat | Y | REAL | - | Latitude of the camera location |
| lon | Y | REAL | - | Longitude of the camera location |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE II IMAGES TABLE DATA DICTIONARY** | | | | |
| **Column Name** | **Optional** | **Format** | **Length** | **Description** |
| camera | N | INT | - | Reference id to **CAMERAS** **TABLE** |
| id | N | INT | - | Unique image id (Serial) |
| name | N | VARCHAR | 200 | Image name in the file system |
| year | N | INT | - | Year of the image scene taken |
| month | N | INT | - | Month of the image scene taken |
| day | N | INT | - | Day of the image scene taken |
| hour | N | INT | - | Hour of the image scene taken |
| minute | N | INT | - | Minute of the image scene taken |
| second | N | INT | - | Second of the image scene taken |
| date\_taken | N | TIMESTAMP | - | Timestamp of the image scene taken. |
| image\_path | N | VARCHAR | 500 | Image path in the file system |
| direction | Y | VARCHAR | 5 | Direction of the camera when taking image |
| labeled | N | BOOLEAN | - | Flag used for indicating wheter image is labeled or not. *‘True’* for labeled image, *‘False’* for unlabeled image |
| set\_type | Y | VARCHAR | 10 | Used for indicating whether image is in training set, validation set, or test set |
| ped\_count | Y | INT | - | Number of pedestrian in the image. Default value : 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE III LABELS TABLE DATA DICTIONARY** | | | | |
| **Column Name** | **Optional** | **Format** | **Length** | **Description** |
| image | N | INT | - | Reference id to **IMAGES** **TABLE** |
| id | N | INT | - | Unique camera id (Serial) |
| topx | N | INT | - | X coordinate value of top leftmost pixel of the bounding box |
| topy | N | INT | - | Y coordinate value of top leftmost pixel of the bounding box |
| botx | N | INT | - | X coordinate value of bottom rightmost pixel of the bounding box |
| boty | N | INT | - | Y coordinate value of bottom rightmost pixel of the bounding box |

**Some useful query commands for the database are:**

-- to count number of labeled images

select count(\*) from images where labeled=true;

-- to count number of labeled images not including crowd

select count(\*) from images where labeled=true and set\_type!='crowd';

-- to count number of positive patches

select count(\*) from labels where type='pos';

-- to count number of positive patches

select count(\*) from labels where type='neg';

-- count labels by camera

select camera, count(\*) from labels left join images on labels.image=images.id group by images.camera;

-- count positive labels by camera

select camera, count(\*) from labels left join images on labels.image=images.id where labels.type='pos' group by images.camera;

-- count negative labels by camera

select camera, count(\*) from labels left join images on labels.image=images.id where labels.type='neg' group by images.camera;

-- select heights and widths of positive labels

select (boty-topy) as height, (botx-topx) as width from labels where type='pos';

-- select heights and widths of negative labels

select (boty-topy) as height, (botx-topx) as width from labels where type='neg';

**Phase 5. Link the dot\_images folder to the database**

At this point the images are sitting in the dot\_images folder on compute, they then need to be put into a directory tree that links to the database. Using build\_files\_db.py script in the db will accomplish this.

The structure of the tree looks like:

PROJECT-DATAROOT:

   └───Camera Name

  └───Year

      └───Month

     └───Day

        └───Hour

            └───Minute

                 └───Image Name

**Phase 6. Design Label tool and start labeling images**

The labeling tool was created to generate training data. We needed positive examples of pedestrians, and negative examples of pedestrians. The labeling tool allows a user to draw a patch around a pedestrian (in blue) and a non-pedestrian (in red). For Mac users, you will need to install xquartz. You can do this at: https://www.xquartz.org/

Helpful instructions for the label tool:

1. Open a terminal window `ssh -X -L 1111:compute:22 NETID@gw.cusp.nyu.edu`

2. Open a new terminal window `ssh -X -p 1111 NETID@localhost`

3. Go to your cloned `hadive` repository folder

4.`git pull` to get the latest code

5. Go to `LabelingTool` folder, run `python LabelingTool\_TwoClick.py`

6. Start Labeling:

`d` for deleting last bounding box.

`q` for saving the labels, and loading next images

`r` for negative labeling

`b` for positive samples

`e` for escaping the crowd scene

`0` for quit from labeling tool (zero)

`c` for clearing all the patches

`t` If you don’t want to label a certain image press or in case you get some error when try to save it

The output of the label tool was a numpy file in it’s parent image’s folder.

**Phase 7. Create models for counting pedestrians—blob detection, Histogram of oriented gradients, and Convolutional Neural Networks (CNN)**

The Blob, HOG, and nn folders contains all the code to start running your blob detection, HOG, and CNN models, respectfully. Forewarning, for the CNN model, the programming language LUA was used. It can be downloaded from the following website: https://www.lua.org/download.html

**Phase 8. Build visualization to show results**

The webpage.py script will be in the webpage folder, and it when run it will take the coordinates of all the cameras used in the project (22) and display a histogram of the pedestrian counts for the previous 24 hours.

**Installation/configuration**

Some Python Libraries that were used in the scripts but are not very common and should be installed are:

1. Shutil
2. Psycopg2
3. Cv2
4. Numpy
5. Matplotlib
6. Glob
7. BeautifulSoup
8. vincent
9. folium