

Traffic Flow Prediction by Image Recognition and Machine Learning

機械學習交通智能分析系統

Law Ho Wai, Adrian (Notstevve) (Leader)

Chau Tim Lok, Justin (timlok123)

Table of Content

Introduction	3
Details	4
1. Web-scraping and Cloud Computing	4
2. Image recognition	7
3. Data analytics	10
Discussion	13
Url to Github	13

Introduction

Currently, traffic statistics are collected by the Transport Department using physical counting stations. Meanwhile, third party companies provide this information by collecting geographical data from people's phones. Our design does not need complex equipment to be maintained, nor intrusive data collection methods. What we need can just be a static camera taking images periodically, and apply image recognition techniques. Meanwhile, we can provide traffic amount predictions (on a scale such as every half hour) using time series machine learning methods based on data collected over day(s). This allows users to pre-plan their travel based on predicted traffic

Details

1. Web-scraping and Cloud Computing

(The code used in this part is stored in *scrape_image.py*)

1. Scaping image from the webpage

https://static.data.gov.hk/td/traffic-snapshot-images/code/Traffic_Camera_Locations_En.xml



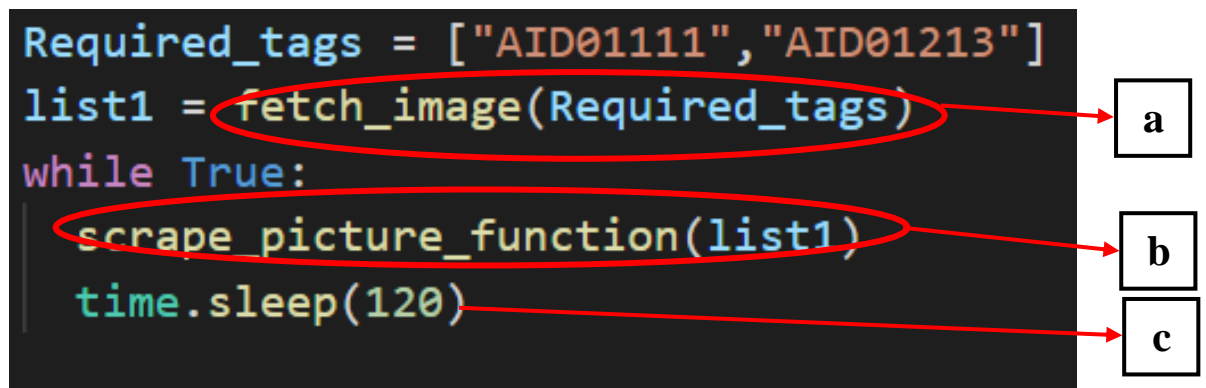
It is a subpage of data.gov.hk. Inside the webpage, the data is organized like a html file and in each blocks, there is an uniform resource locator (url), which links to image captured in a particular location.

As the url is static and the images update every 2 minutes, we therefore make use of it scrape the image using the url.

```
</image>
▼ <image>
  <key>AID01111</key>
  <region>Kowloon</region>
  <district>Kowloon City</district>
  <description>Hong Chong Road near The Hong Kong Polytechnic University - Northbound (1) [AID01111]</description>
  <easting>836693</easting>
  <northing>818352</northing>
  <latitude>22.30412026</latitude>
  <longitude>114.1809989</longitude>
  <url>https://tdcctv.data.one.gov.hk/AID01111.JPG</url>
</image>
```



2. Getting the page content and loading the images



2 locations, which are **Hong Chong Road near The Hong Kong Polytechnic University – Northbound(1) [AID01111]** and **Hong Chong Road near The Hong Kong Polytechnic University - Southbound [AID01213]** have been selected for analysis. The function of a), b), c) part of code are explained below,

a. The `fetch_image` function

```
def fetch_image(Required_tags):
    # Get page content
    url = "https://static.data.gov.hk/td/traffic-snapshot-images/code/Traffic_Camera_Locations_En.xml"

    page = requests.get(url)
    soup = BeautifulSoup(page.content, "html.parser")

    #load the data and pack them in tuple
    url_tags = soup.find_all("url")
    place_tags = soup.find_all("description")
    key_tags = soup.find_all("key")

    place_url_key_list = []

    for i in range(len(url_tags)):
        temp_url = (str(url_tags[i]).replace("<url>", "").replace("</url>", ""))
        temp_place = ((str(place_tags[i]).replace("<description>", "").replace("</description>", "").replace(" ", "_")))
        temp_key = ((str(key_tags[i]).replace("<key>", "").replace("</key>", "")))
        if str(temp_key) in Required_tags:
            place_url_key_list.append((temp_place, temp_url))

    return place_url_key_list
```

For the `fetch_image` function, it takes in the tags of 2 locations and it return a list that contains the **name of the locations** and the **urls**. The example list returned is shown below,

```
[('Hong_Chong_Road_near_The_Hong_Kong_Polytechnic_University_ - Northbound_(1)_[AID01111]', 'https://tdcctv.data.one.gov.hk/AID01111.JPG'), ...]
```

- Name of the location
- url

b. The `scrape_image_function`

```
def scrape_image_function(place_url_key_list):

    # Download the image and store it somewhere
    for count, i in enumerate(place_url_key_list):

        current_place = str(i[0])
        current_place_folder_name = current_place.split("_")[-1]
        image_url = str(i[1])
        t = time.localtime()
        current_time = time.strftime("%Y_%m_%d_%H_%M_%S",t)
        name_of_image = 'Index{count}_{place}_{time}.jpg'.format(count=count, place=current_place,time=current_time)

        #make a new directory and store it inside

        try:
            os.mkdir(os.path.join(os.getcwd(),current_place_folder_name))
        except:
            pass
        go_back = os.getcwd()
        os.chdir(os.path.join(os.getcwd(),current_place_folder_name))

        try:
            with open(name_of_image, 'wb') as f:
                f.write(requests.get(image_url).content)
                f.close()
        except:
            print("Index_{no} images goes wrong".format(no=count))

    #Return to the main_directory(loaded_img)a
    os.chdir(go_back)

print("Import completed")
```

For the **scrape_image_function**, it takes the list returned in part a) and store the image scraped in *folder [Tags]*.

📁 [AID01111]	✓	18/4/2022 11:28	檔案資料夾
📁 [AID01213]	✓	18/4/2022 11:29	檔案資料夾



- c. As the url updated every 2 minutes. It is not necessary to run the program every seconds and create redundant images. Therefore, `time.sleep(120)` is added so as to run the program in every 2 minutes.
3. Cloud Computing – Microsoft Azure: In order to streamline the process and avoid personal computers from running the image scraping script non-stop, we use cloud computing resources provided by Microsoft Azure. It is a Linux-based virtual machine, containing basic and essential packages such as python and git. By using the Linux command “nohup” on the python process, we are able to run the script non-stop without the need of personal computers (as long as the server is kept running).

```

33455 0.0 0.7 18396 6748 ? Ss Apr10 0:00 /lib/systemd/systemd --user
33456 0.0 0.5 170112 4872 ? S Apr10 0:00 (sd-pam)
35675 0.0 3.4 79728 31692 ? S Apr10 6:17 python3 new_scrape_image_use_this.py
40893 0.0 0.0 6040 456 ? Ss Apr11 0:00 ssh-agent -s
40913 0.0 0.0 6040 456 ? Ss Apr11 0:00 ssh-agent -s

```

The python process running since April 10

2. Image recognition

In order to fully utilize the image that we have obtained and reach of our goal of counting the number of cars in the image automatically, we use a technique called image recognition. In particular, we use OpenCV’s Haar Cascade Classifier, which is a machine learning based method for feature detection. The computer requires human-marked positive and negative images (in this case, image with cars and image without cars) to train the classifier.

This classifier works by feature extraction from the positive images, like edge and line features. To select the best features of the image, the system uses AdaBoost, to find the best threshold in classifying positive/negative components, and weights all these classifiers finally.

For an actual image, in order to save time, the classifiers stages are applied one by one, such that the particular region is only checked again by the next layer if it passes the previous layer. If the image passes all the layers, it is classified as a car.

References:

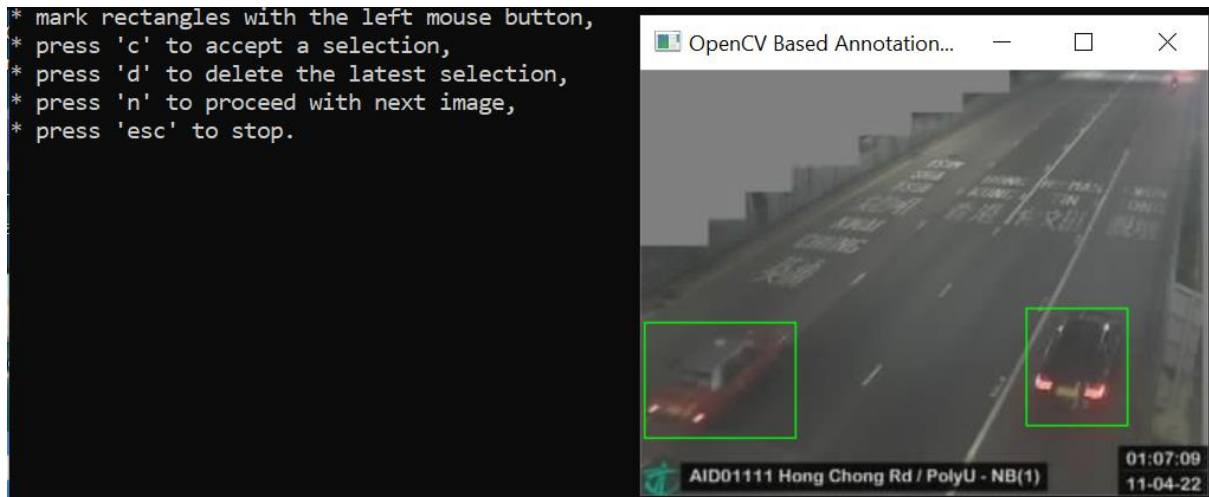
- https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
- <https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/>
- <https://learncodebygaming.com/blog/training-a-cascade-classifier>
- OpenCV download: <https://opencv.org/releases/>

Python OpenCV (opencv-contrib-python): pip install opencv-contrib-python

Using this command in command prompt installs opencv to the python environment using the pip installation package.

We have done the following steps for recognizing the cars in the image:

1. Collecting positive and negative images and classifying them.
 - a. **OpenCV 4.5.5 opencv_annotation.exe**: this software opens the photos in a file, letting the user to annotate the position of the object to be classified. Generates a .txt file for all photo paths and object location in the form (x y w h) (x position, y position, width, height)



Software to annotate object positions

- b. **TrainImgs/neg.py & TrainImgs/neg.txt**: neg.py runs through all the files in TrainImgs/neg and stores all the path of negative images in neg.txt
 - c. **OpenCV 3.4.11 opencv_createsamples.exe**: reads the .txt file from part (a), and appends all images together to a single .vec (vector) file. All images undergoes feature reduction to 20x20 size to decrease training time.
 - d. **OpenCV 3.4.11 opencv_traincascade.exe**: this file from OpenCV collects the negative images from part (b) and positive images from part (c), and trains on the dataset. 50 stages are used in the training. The result is an .xml file storing the features of the cars (TrainImgs/cascade/cascade.xml). (The file is not quite readable)


```
<?xml version="1.0"?>
<opencv_storage>
<cascade>
  <stageType>BOOST</stageType>
  <featureType>HAAR</featureType>
  <height>20</height>
  <width>20</width>
  <stageParams>
    <boostType>GAB</boostType>
    <minHitRate>9.9500000476837158e-01</minHitRate>
    <maxFalseAlarm>5.0000000000000000e-01</maxFalseAlarm>
    <weightTrimRate>9.4999999999999996e-01</weightTrimRate>
    <maxDepth>1</maxDepth>
    <maxWeakCount>100</maxWeakCount></stageParams>
  <featureParams>
    <maxCatCount>0</maxCatCount>
    <featSize>1</featSize>
    <mode>BASIC</mode></featureParams>
  <stageNum>50</stageNum>
  <stages>
    <!-- stage 0 -->
    <_>
      <maxWeakCount>4</maxWeakCount>
      <stageThreshold>-9.3634121119976044e-02</stageThreshold>
      <weakClassifiers>
        <_>
          <internalNodes>
            0 -1 514 -1.8898090720176697e-01</internalNodes>
          <leafValues>
```

cascade.xml

2. **ImageRecog/ImgRecog.py:** Applying the trained dataset onto real images, and count the number of cars
 - a. Check if the collected image is a proper image: is it an error image, or a duplicated image (sometimes the image is not updated from the government server due to issues) with a bitwise xor decision:

```
if not(np.bitwise_xor(ugly,img).any()) or
not(np.bitwise_xor(img,last).any()) or type(img)=="NoneType":
    print("identical/bad img:",filename)
```



Error Image

- b. Use the .xml file from part (1d), and apply the classification on the (converted to black and white) image, and add to a .csv file for the next model prediction part.

```
import cv2
import numpy as np
import os
import csv

i="[AID01111]"
fd=open(i+'.csv','w')
fd.write('"Datetime","count"')
fd.close()
for filename in os.listdir(i):
    img_fl = os.path.join(i,filename)
    class_file="cascade.xml"
    img=cv2.imread(img_fl)
    #converts file to black and white
    b_n_w=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    car_track=cv2.CascadeClassifier(class_file)
    cars=car_track.detectMultiScale(b_n_w)
    fd.write("{}{}\n".format(filename[-23:-4],len(cars)))
    fd.close()
```

3. Data analytics

(The code used in this part is stored in *time_series_anaylsis_new.py*)

Analyzing the data, we will be using the **Long short-term memory (LSTM)** to analyze the data collected in Image recognition.

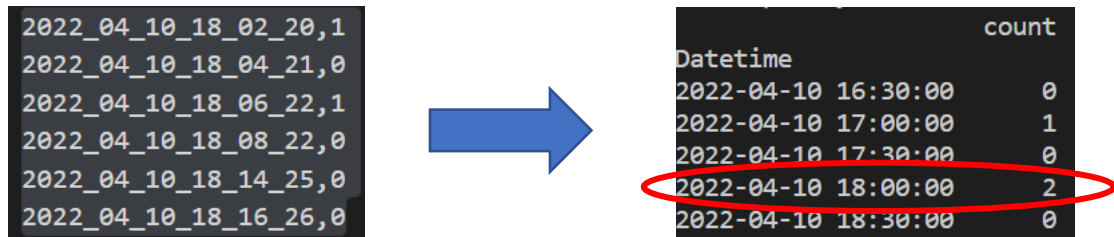
Long short-term memory (LSTM) is one kind of the recurrent neural network (RNN) architecture and it has connections that provides feedback. It can process entire sequences of data an it has various application like processing speech or handwriting.

We are using Tensorflow, which is a free an open-source python library for machine learning applications and building artificial intelligence, to build our LSTM model. \

We have done the following steps to analyse the result,

1. Resampling the data

Since the number of counts in csv is updated every 2 mins, in order to make the data less discrete, we group the counts that are in the same 30 mins.



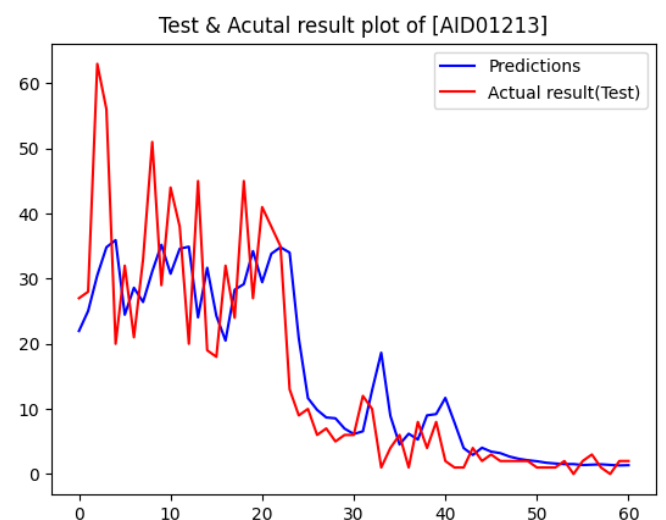
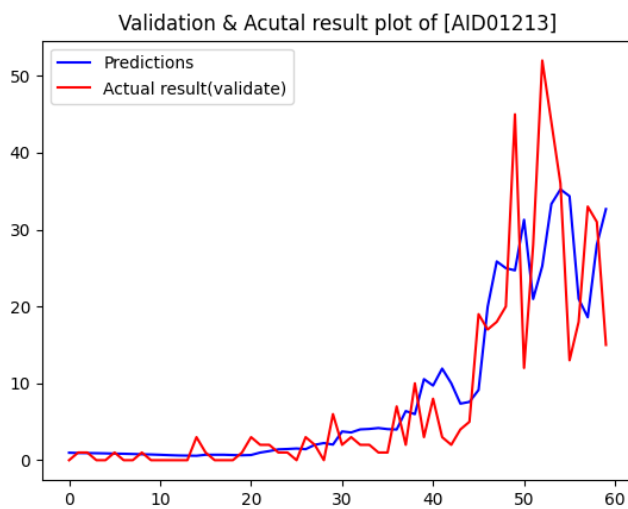
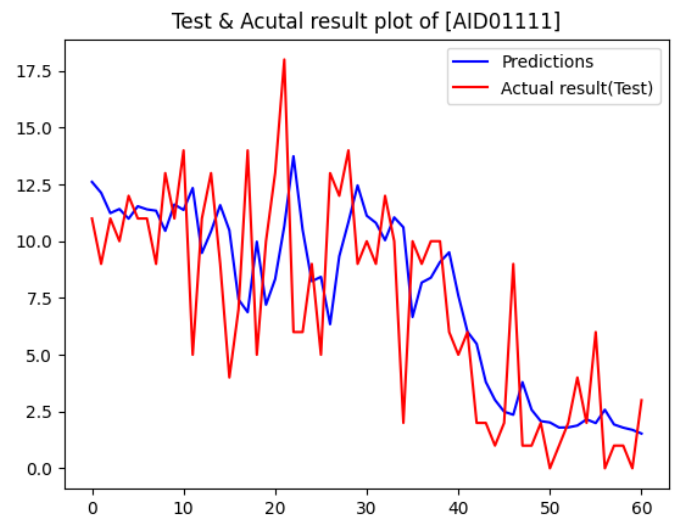
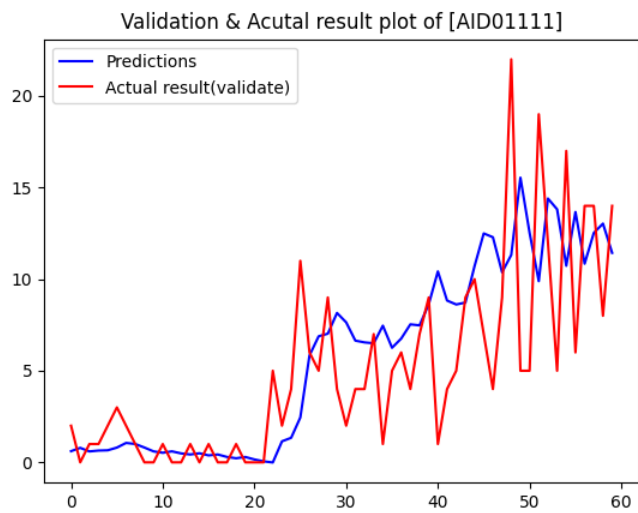
Datetime	count
2022-04-10 16:30:00	0
2022-04-10 17:00:00	1
2022-04-10 17:30:00	0
2022-04-10 18:00:00	2
2022-04-10 18:30:00	0

2. Training the model

```
model.fit(x_train,  
          y_train,  
          validation_data=(x_valid, y_valid),  
          epochs=epoch_no,  
          callbacks=[cp1])  
model.save('my_model.h5')
```

We split the data into 3 parts including training, validation, and testing. The data is then passed into the model and the trained model is saved as “my_model.h5” in the same folder as the *time_series_anaylsis_new.py*.

3. Using the model to predict the result and visualizing the data



As shown above, the plots on the left are comparing the predictions and the data in validation, while the plots on the right is comparing the prediction result and testing. The x-axis of all the plot is time, starting from 10th April, 2022 16:51, with 30 minutes as one unit. The y-axis is the number of cars count.

Since validation data are what we use to train the model and what we are concerning is using our model to predict the future, we should focus on the **2 plots on the right**.

Discussion

As we can see, in terms of the trend, the prediction curves follow the actual curves quite well in both plots, but the prediction **does not perform well on predicting peak and minimum values**. The possible reason could be the data points taken are too discrete as the images provided by the Hong Kong government only updated every 2 minutes. Therefore, rapid changes between data point are observed. Possible solution to this could be taking more data (i.e. collecting all the images in a month) and over a longer period, the data would tend to be smoother and the prediction are more likely to be more accurate.

Moreover, the inability to predict values with large jumps, is due to the inherent design of such predictive algorithms and the dataset. The dataset experiences stochastic behaviour due to its real-world nature, and it is almost impossible to predict such jumps. We would rather prevent the model from overfitting to the training dataset, as this causes the model failing to be applied on any other dataset and expecting a reasonable result. A possible solution is setting the collection period for one data point to be longer, in order to average out any short-term abnormalities. This, however has a trade-off in the form of less dense data outputs.

Url to Github

You can scan the following QRcode to access the code on our Github repository



<https://github.com/timlok123/Traffic-flow-prediction>

