Deep Learning in Production

as

SMART CITY WITH ARTIFICIAL INTELLIGENCE

Syed Owais Chishti

Elacsoft -Pakistan

Author Details

Data Scientist, www.elacsoft.cf.

https://github.com/elacsoft/DataScience.

Abstract

This is a note that describes how to involve Artificial Intelligence or Machine Learning with IoT (Internet of Things) to solve your daily life problems, like leaving work or other important stuff just to grab some vegetables and dairy products. Also, corner to corner cameras will be installed for surveillance and security with the help of deep learning and computer vision.

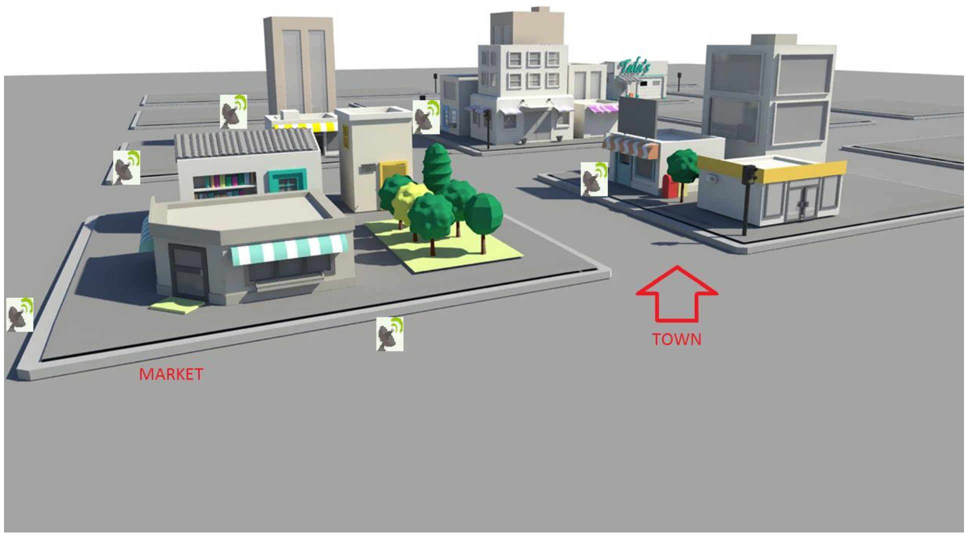
This note tries to present you some small but much useful technologies which can make your city a "Smart City":

- Consider the routers distance placed by internet supplier in your area, at the same distance own server-based units will be installed so that use internet service for ordering and to view your logs without any charges.
- Cameras with computer vision techniques will be installed for surveillance and security, also it recognizes every person or object enters or leaves the town or city with the help of data collected from each resident. Also, in case of any missing the cameras will help to find the person or object.
- As now in modern era everyone has android devices, each android device will contain very user-friendly android application to order stuff of choice.

Keywords: Deep Learning, Machine Learning, Artificial Intelligence, Computer Vision, Internet of Things.

Introduction

The idea in this note is inspired by the me, due to developing and programming I always have very little free time thus people like these routines know well the importance of time and energy. The scope of this note is to provide the easiness and self-security to every resident of town or city. Also, this note leads towards modern deep learning models to production, how to detect objects in live video and save it in cloud so that anyone can access it simply by login in to their local TCP/IP portal also user-friendly Android app allows every single person to access any stuff within territory. From now on we learn about algorithms, codes, and techniques used to develop this whole scenario.



TCP/IP Servers

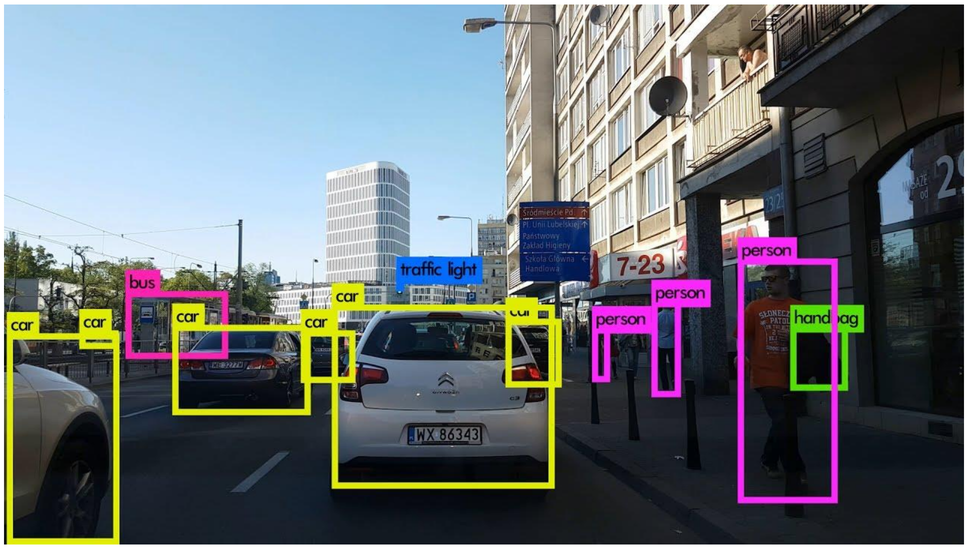
TCP/IP (Transmission Control Protocol/Internet Protocol) **TCP/IP**, or the Transmission Control Protocol/Internet Protocol, is a suite of communication protocols used to interconnect network devices on the internet. **TCP/IP** can also be used as a communications protocol in a private network (an intranet or an extranet). Nowadays many controllers have built-in server functions like Node-MCU (actually a Wi-fi module with built-in GPIOs), so that no one has to pay a single penny to use services offered. These units(Node-MCUs) will be placed corner to corner at particular distance also they will be inter-connected to avoid distance communication gap. A user sends a request from Android App connects to near-by unit and propagates through other units till it reaches to specified IP address, different IP addresses are assigned to different shop keepers respectively and statically.

Moreover, these units upload pictures taken by cameras to local server which can be accessed by any one in that town with id, password allotted to each person resident of that town, this server is also under vision of security agencies for surveillance.

Object Detection

Here comes the interesting part for AI lovers, Computer Vision with deep learning. For sure CNN- Convolutional Neural Networks are used to detect objects in images and videos so far but moving forward we are using some advance features of CNN like e.g. Capsule Networks and YOLO (you only look ones). Let's discuss a how YOLO algorithm works with some python code, also after few pages I will share C++ pseudo code for Node-MCU.

R-CNN vs YOLO Algorithm.



We take usual CNN and repurpose it as object detector, use any existing classifier VGG net, inception these are huge CNN trained on huge datasets. by google etc. We can take object in question in image and slightly rotate the classifier to every object we have consider in boxes. Thus, we it becomes multi classifier and we can keep what about classifier is very focused on but it is computational very expensive approach. Solution is R-CNN. Before proceeding to CNN, it uses process called selective search, to create a set of bounding boxes also known as region proposals in image. Selective process looks the image in different windows of random sizes. and for different boxes it groups the pixels for say texture, color or intensity to detect object. No. of boxes or windows depends on user by some threshold, and then proceed it to CNN, finally SVM to classify the image in box. Then we run the run the box in linear regression model to output tidier coordinates for the box once the object has been classified.

It is proven effective object detection approach, but due to improvements required in R-CNNs different proposals are submitted.

R-CNN: https://arxiv.org/abs/1311.2524

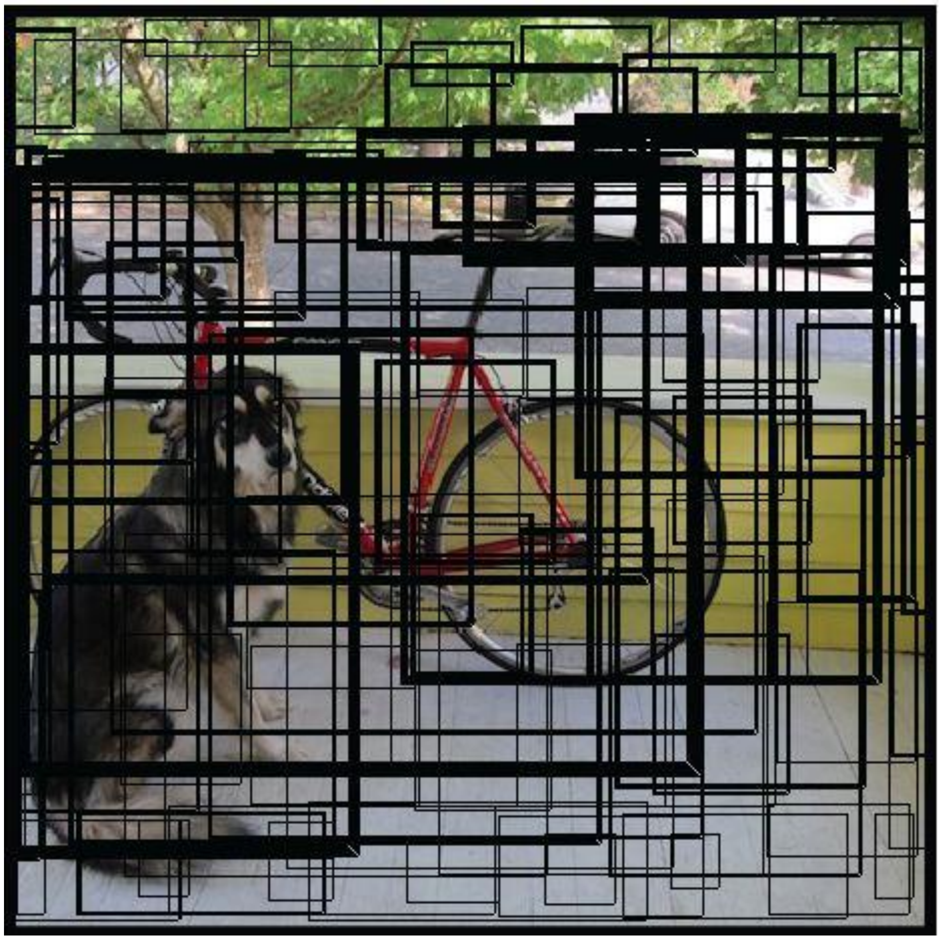
Fast R-CNN: https://arxiv.org/abs/1504.08083

Faster R-CNN: https://arxiv.org/abs/1506.01497

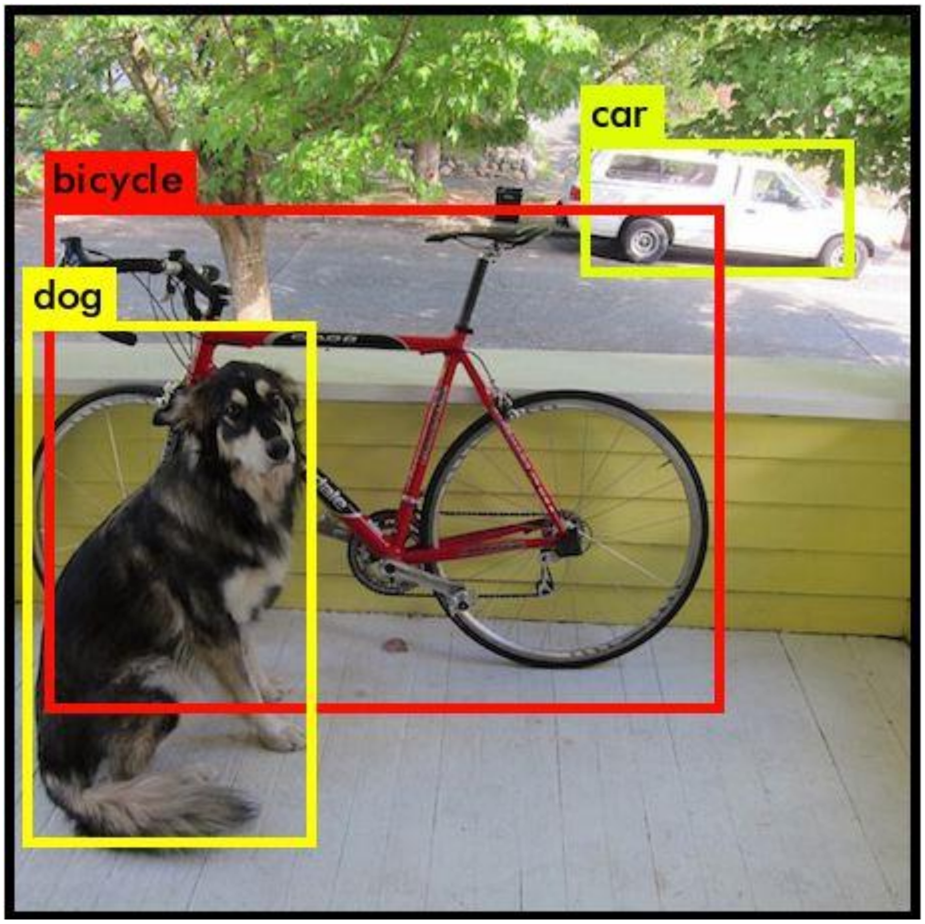
Mask R-CNN: https://arxiv.org/abs/1703.06870

Now, talk about YOLO, it is state if the art object detection technique and uses totally different approach.

YOLO looks the image only once as name implies and divides the image in to 13x13 cells, each of these cell is responsible to predict 5 bounding boxes. YOLO also provides a confidence level which describes that if the cell grabs any object with level of certainty. Notice the thickness of boxes around dog and bicycle indicates level of confidence.



Since there are 13×13 = 169 grid cells and each cell predicts 5 bounding boxes, we end up with 845 bounding boxes in total. It turns out that most of these boxes will have very low confidence scores, so we only keep the boxes whose final score is 30% or more (you can change this threshold depending on how accurate you want the detector to be).



From the 845 total bounding boxes we only kept these three because they gave the best results. But note that even though there were 845 separate predictions, they were all made at the same time — the neural network just ran once. And that's why YOLO is so powerful and fast. This neural network only uses standard layer types: convolution with a 3×3 kernel and max-pooling with a 2×2 kernel. No fancy stuffs. There is no fully-connected layer in YOLOv2. This neural network only uses standard layer types: convolution with a 3×3 kernel and max-pooling with a 2×2 kernel. No fancy stuffs. There is no fully-connected layer in YOLOv2. The very last convolutional layer has a 1×1 kernel and exists to reduce the data to the shape 13×13×125. This 13×13 should look familiar: that is the size of the grid that the image gets divided into.

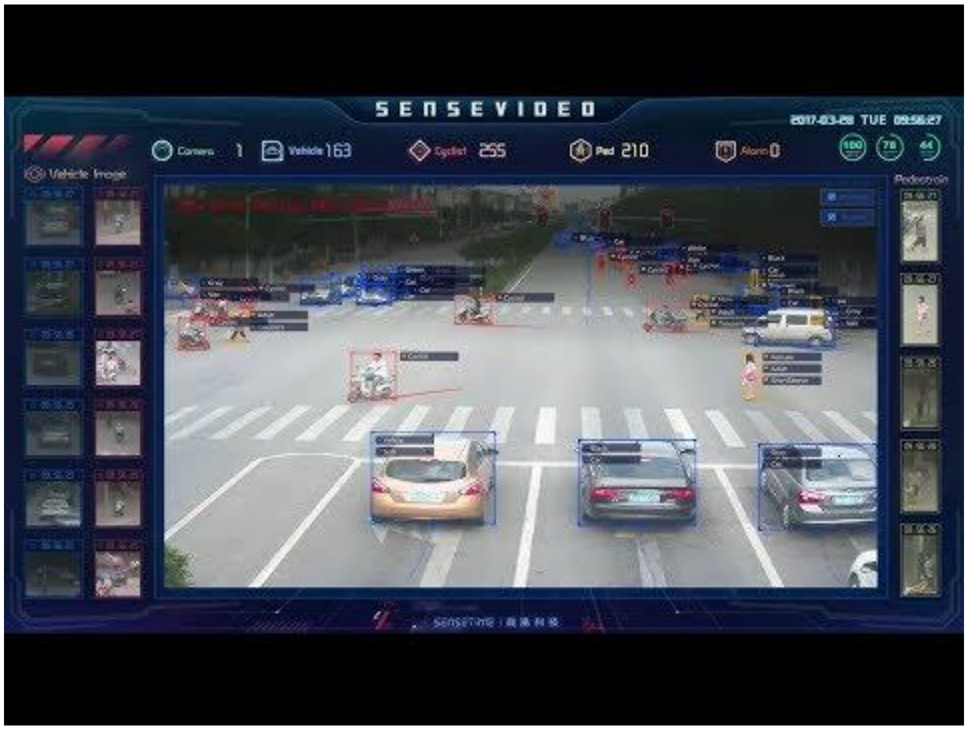
So, we end up with 125 channels for every grid cell. These 125 numbers contain the data for the bounding boxes and the class predictions. Why 125? Well, each grid cell predicts 5 bounding boxes and a bounding box is described by 25 data elements:

- x, y, width, height for the bounding box's rectangle.
- the confidence scores.
- the probability distribution over the classes.

Read about YOLO more in Paper here https://arxiv.org/pdf/1612.08242v1.pdf.

Some of Work done before this note.

China has equipped itself with state of art surveillance system.



This is the real footage from China's current CCTV and real time analysis system,

On the very top of the window, yellow character with 118215 is the number of pedestrians, pink with 69965 is non-motorized vehicles, blue with 48763 is cars/trucks. Showing the total amount of different subjects captured by this camera within a certain time frame. The! mark with zero count is warnings. On the actual video, the small yellow bracket surrounding pedestrians first shows gender, then adult/kids, then one or two items for clothing. Some items are omitted when no clear identification. The red/pink bracket for non-motorized vehicle only shows non-motorized vehicle, but it will try to analyze the person on that vehicle when possible, like they do with pedestrians (this function is shown around the 0:06 mark where the women on bike is near the center. The blue bracket for cars/trucks shows the color of the vehicle and the type of vehicle (sedan, van, truck, etc.), possibly capturing license plate numbers as well but not shown. (China's been known for having system with auto/semi-auto plate ID).

Actual video: https://youtu.be/aE1kA0Jy0Xg

Codes:

1) Pseudo code for Node-MCU:

```
wifi.setmode(wifi.STATION)
wifi.sta.config("YOUR_NETWORK_NAME","YOUR_NETWORK_PASSWORD")
print(wifi.sta.getip())
led1 = 3
led2 = 4
gpio.mode(led1, gpio.OUTPUT)
gpio.mode(led2, gpio.OUTPUT)
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        local buf = "";
        local _, _, method, path, vars = string.find(request, "([A-Z]+) (.+)?(.+) HTTP");
        if(method == nil)then
            _, _, method, path = string.find(request, "([A-Z]+) (.+) HTTP");
        end
        local _GET = {}
        if (vars ~= nil)then
            for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
                _GET[k] = v
            end
        end
        buf = buf.."<h1> ESP8266 Web Server</h1>";
        buf = buf.."<p>GPIO0 <a href=\"?pin=ON1\"><button>ON</button></a>&nbsp;<a
href=\"?pin=OFF1\"><button>OFF</button></a></p>";
        buf = buf.."<p>GPIO2 <a href=\"?pin=ON2\"><button>ON</button></a>&nbsp;<a
href=\"?pin=OFF2\"><button>OFF</button></a></p>";
        local _on,_off = "",""
        if(_GET.pin == "ON1")then
              gpio.write(led1, gpio.HIGH);
        elseif(_GET.pin == "OFF1")then
              gpio.write(led1, gpio.LOW);
        elseif(_GET.pin == "ON2")then
              gpio.write(led2, gpio.HIGH);
        elseif(_GET.pin == "OFF2")then
              gpio.write(led2, gpio.LOW);
        end
        client:send(buf);
        client:close();
        collectgarbage();
    end)
end)
```

2) Python Codes for video to image and YOLO detection, here we are using OpenCV a Python Library for Computer Vision:

OpenCV link: https://opencv.org/releases.html

Download link: https://sourceforge.net/projects/opencvlibrary/files/opencv-win/3.4.0/opencv-3.4.0-vc14_vc15.exe/download

```
import sys

import argparse

import cv2

print(cv2.__version__)

def extractImages(pathIn, pathOut):

  count = 0  vidcap = cv2.VideoCapture(pathIn)

  success,image = vidcap.read()

  success = True

  while success:

    vidcap.set(cv2.CAP_PROP_POS_MSEC,(count*1000))

    # added this line

    success,image = vidcap.read()

    print ('Read a new frame: ', success)

    cv2.imwrite(pathOut + "\\frame%d.jpg" % count, image)  # save frame as JPEG file

count = count + 1

if __name__=="__main__":

print("aba")

a = argparse.ArgumentParser()

a.add_argument("--pathIn", help="path to video")

a.add_argument("--pathOut", help="path to images")

args = a.parse_args()

print(args)

extractImages(args.pathIn, args.pathOut)
```

YOLO detection code:

Explore code from my GitHub Repository: https://github.com/elacsoft/YOLO

Server Codes and Size challenges:

The problem is that as we are using TCP/IP server and all the data is saved on a local machine rather than a cloud storage. Thus, this note emphasizes on some techniques to solve this issue.

Size of our local machine server will be 100 GB (Giga Bytes) and let's consider few points below in according to the server size.

- As discussed above video is converted to images for YOLO detection, as YOLO looks the image only once thus the image just converted yet is passed to YOLO and after its analysis the image is deleted but its data is recorded in database.

Let's discuss video size just as recorded by camera:

- Video Format: MPEG-2 3.7Mbps fixed rate.
- Resolution: 720x486.
- Frames per Second: 14
- Video Length: 24 hours
- Size: 39 GB

After 24 hours the video is compressed to save space.

- Format: same as original.
- Resolution: 460x458
- FPS = 5 low end video.
- Video Length: 24 Hours
- Size: 632 KB (Kilo Bytes)

All compressed videos are saved for one month of duration and occupied only 632*30 = 18960 KB or 18.96 MB, then at the start of every month all previous videos are deleted except those which are required in some cases.

Let's see how to create server on different Operating Systems:

- Windows:

For Windows OS simply download XAMPP PHP server, open XAMPP control panel and START both MySQL and Apache.

Place all your server files in C:\XAMPP\htdocs directory. Run local host and port number specified on XAMPP control panel besides Apache as "localhost:port" in your browser to access your data in htdocs directory.

See, https://www.apachefriends.org/index.html

- Linux:

For Linux use Terminal and Enter following command to create Apache server and enable Php for it.

```
>>> apt-get install apache2 apache2-doc apache2-utils.
```

```
>>> apt-get install libapache2-mod-php5 php5 php-pear php5-xcache.
```

Your server is created in /var/www/html directory.

Then use "ifconfig" in terminal to get your local IP address, and in browser use only this IP address to access data in /var/www/html directory. Port is 80 by default.

Delete image after it is processed by YOLO:

To delete image after it is analyzed by YOLO, make some changes in Python code written above after this line of code,

```
cv2.imwrite(pathOut + "\\frame%d.jpg" % count, image)

os.system("Run YOLO commands here")

del (pathOut + "\\frame%d.jpg" % count, image).
```

Database:

For the sake of simplicity and to save money and space this note emphasize to use Python based SQLite3 Database, which is easy to use and space efficient.

Databases offer, typically, a superior method of high-volume data input and output over a typical file such as a text file. SQLite is a "light" version that works based on SQL syntax. SQL is a programming language in itself but is a very popular database language. Many websites use MySQL, for example. SQLite truly shines because it is extremely lightweight. Setting up an SQLite database is nearly instant, there is no server to set up, no users to define, and no permissions to concern yourself with. For this reason, it is often used as a developmental and prototyping database, but it can and is used in production. The main issue with SQLite is that it winds up being much like any other flat-file, so high volume input/output, especially with simultaneous queries, can be problematic and slow therefore, only admin where server is placed will have access to it, no need to put it in server directory. You may then ask, what really is the difference between a typical file and SQLite. First, SQLite will let you structure your data as a database, which can easily be queried, so you get that functionality both with adding new content and calling upon it later. Each table would likely need its own file if you were doing plain files, and SQLite is all in one. SQLite is also going to be buffering your data. A flat file will require a full load before you can start querying the full dataset, SQLite files don't work that way. Finally, edits do not require the entire file to be re-saved, it's just that part of the file. This improves performance significantly. Alright great, let's dive into some SQLite and let's demonstrate how to create, read, write few tables in Relational SQLite3 Database.

```
import sqlite3
import time
import random
import datetime
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib import style
```

```
style.use('fivethirtyeight')
 inp = input("Enter Your SQL Choice (R)egular, (A)ppend, (r)ead, (V)isualize, (U)pdate or (D)elete\n")
 connect = sqlite3.connect('SmartTownDataBase.db')
c = connect.cursor()
#Create table
def create_table():
    c.execute('CREATE TABLE IF NOT EXISTS stuffToPlot(unix REAL, datestamp TEXT, key TEXT, description
REAL)')
def data_entry():
    c.execute("INSERT INTO stuffTOPlot VALUES(12345, '2018-14-02', 'Image1', "No abnormal activity
detected.......")")
    connect.commit()
    c.close()
    connect.close()
#create_table()
#data_entry()
def dynamic_data_entry():
    unix = time.time()
    date = str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))
    keyword = 'Python'
    value = random.randrange(0,10)
    c.execute("INSERT INTO stuffToPlot (unix, datestamp, imageNumber, description) VALUES(?,?,?,?)",
     (unix, date, key, data))
    connect.commit()
#read data
def read_from_db():
    c.execute('SELECT * FROM stuffToPlot')
    #c.execute('SELECT keyword, unix FROM stuffToPlot WHERE unix > 1452618731')
    #data = c.fetchall()
    #print(data)
    #print('\n{}'.format(data[0][3]))
    for row in c.fetchall():
        print(row)
if inp == 'r':
    print('Reading Data from Data Base')
    read_from_db()
if inp == 'R':
    for i in range(10):
        dynamic_data_entry()
        time.sleep(1)
#Append new data
if inp == 'A':
    count = int(input('How many entries do you have? '))
    for i in range(count):
        unix = input('Enter Unix: ')
        date = input('Enter Date: ')
        keyword = input('Enter Keyword: ')
        value = input('Enter Value: ')
        c.execute("INSERT INTO stuffToPlot (datestamp, imageNumber, description) VALUES(?,?,?,?)",
         (unix, date, key, data))
        connect.commit()
#Visualizing Data
def graphs():
    c.execute('SELECT unix, value FROM stuffToPlot')
    dates = []
    values = []
    for row in c.fetchall():
        #print(row[0])
        #print(datetime.datetime.fromtimestamp(row[0]))
        dates.append(datetime.datetime.fromtimestamp(row[0]))
        values.append(row[1])
    plt.plot_date(dates, values, '-')
    plt.show()
#Updates and deletes
def update():
    c.execute('SELECT * FROM stuffToPlot')
    [print(row) for row in c.fetchall()]
    c.execute('UPDATE stuffToPlot SET description is = NULL WHERE description = 0)
    connect.commit()
    c.execute('SELECT * FROM stuffToPlot')
    [print(row) for row in c.fetchall()]
def delete():
    c.execute('DELETE FROM stuffToPlot WHERE description is = NULL')
    connect.commit()
    print(50*'#')
    c.execute('SELECT * FROM stuffToPlot')
    [print(row) for row in c.fetchall()]

if inp == 'D':
    delete()
if inp == 'U':
    del_update()
if inp == 'V':
    graphs()
c.close()
connect.close()
```

Size Compression Code:

Use this code:

```
os.system('ffmpeg -y -i myvideo.avi -vf scale=320:240 video_360p.avi')
```

It requires "ffmpeg" support download from this link:

https://drive.google.com/file/d/1nyhvCVcw89hLU9fYdq6wmnNq7UhUhY8r/view

Conclusion:

The idea in this note is to put your efforts made in Deep Learning in to production, if such kind of society, town or city is made it will be very helpful for residents and commercials. All the codes written above may have tendency of amendments because this note is an idea not the final project report. If anyone tries to work on this idea, can contact me for further elaborations, help for design of Android Application and help in changing performance of codes and algorithms. However, being an author of this note I believe that this idea is very cost effective and easy to implement.

Bibliography

1. Convolutional Neural Nets: https://cs.nju.edu.cn/wujx/paper/CNN.pdf
2. China image link: https://i.ytimg.com/vi/aE1kA0Jy0Xg/hqdefault.jpg
3. OpenCV video/image python code: https://stackoverflow.com/questions/33311153/python-extracting-and-saving-video-frames
4. Video size description links: https://stackoverflow.com/questions/27559103/video-size-calculation.

https://toolstud.io/video/filesize.php/imagewidth=460&imageheight=458&framerate=20&timeduration=24&timeunit=hours