Vehicle count model

March 26, 2021

1 Vehicle count model- Using Unsupervised Learning

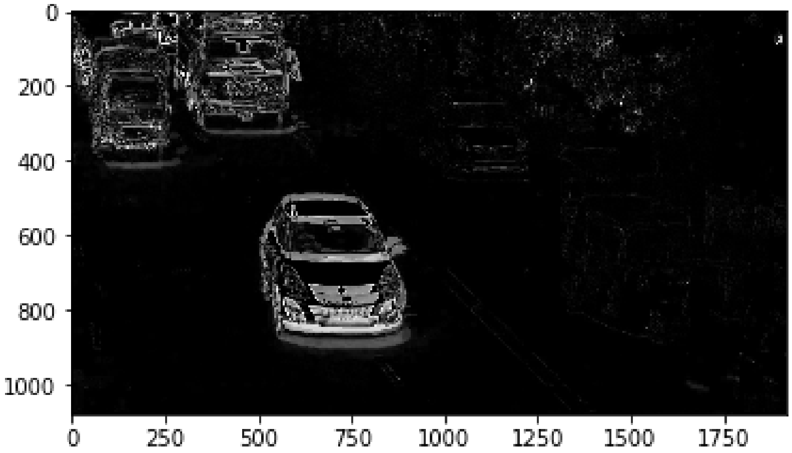
In this task, we are going to use an unsupervised learning model to detect objects that are in motion based on the difference in the frames of the given dataset (which is a video). The input will be location of the video and two coordinates to decide a crossing line for the vehicle in motion.

1. Understanding the Model

1) In this model, we are going to split the video into multiple frames and use the concept of frame differencing. The technique involves manupulating the frames of the video such that a relevant feature can be obtained from the difference between the frames.

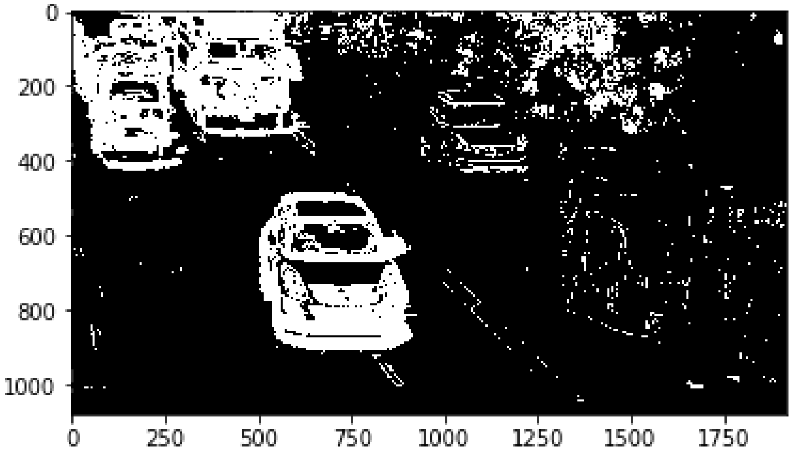
The Original Frame

2) Then we will apply threshold to alter the pixels of the frame differenced images. (It is usefull to provide better results). Low threshold will yield more accuracy for night detection while higher threshold will yield better accuracy for daylight detection.



Frame after Image differencing

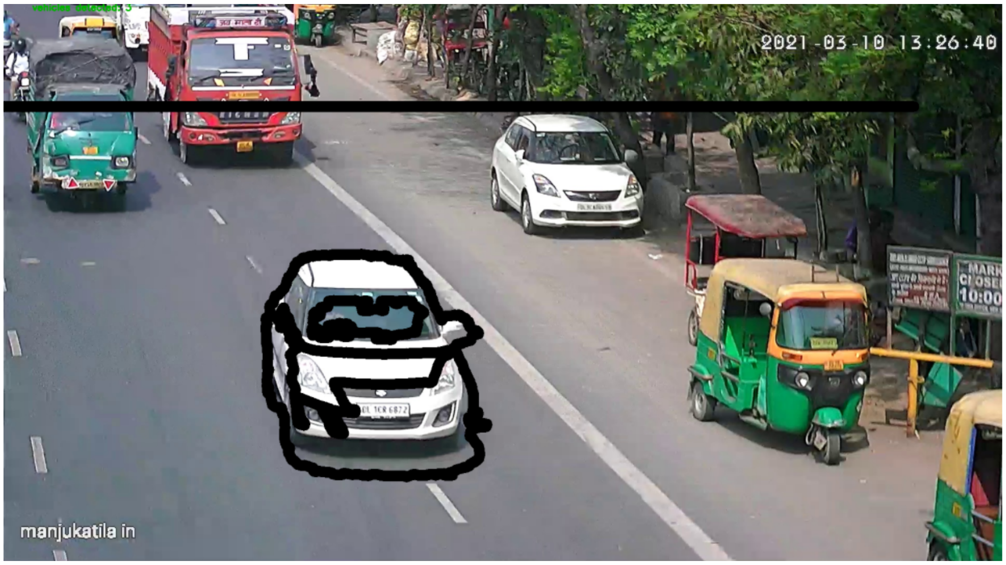
3) The we dilate the manupulated frames so that



Frame after thresholding, dilating, and image differencing-

4) Now we will apply contours over the highlighted parts of the frame and filter the counter to only those that satisfy our criteria (such as it should lie after the crossing line, the size of the contour). We will also add the crossing line to the frame as well.

Final image- [The black line denotes the crossing line and the contour denotes the detected vehicle in motion]



Final image- [The black line denotes the crossing line and the contour denotes the detected vehicle in motion]

The given dataset is:
(https://drive.google.com/drive/folders/1ALJVtlTSzU9b7B75DYFJHWfVINeufbmH?usp=sharing)

2. Why this model and alternatives

This model is based on unsupervised learning method, which is much better idea when there is no given dataset for the vehicle that is being detected. The model focus on contrast of the image under greyscale and therefore it can be more effective as compared to usual supervised learning models .Such supervised learning models have the drawback of showing poor result due to change in color in the video, which can occur when the lightings shift from daylight to night). In such cases of variation in color, unsupervised learning can increase accuracy (by setting the threshold and contour validation condition properly)

Common alternative is : CNN (convolutional neural network) can be combined with configurations of multiple number of suitable layers and training dataset of more than 5000 images (at least, lowering the value will make the model overfit the data which is not good for better accuracy). CNN can be used with OpenCv (Computation cost will increase much rapidly) or YOLO object detection (much faster but needs dataset for training the model)

3. Coding- Building the algorith

We are going to use the following python **libraries**:

[9]:
```
import os #For processsing
from os.path import isfile, join
import re #Will be used as an aid in sorting the frames to easily␣
 ↪recompile them into output video.
import cv2 # opencv library
import matplotlib.pyplot as plt #To show and test the manupulated frames␣
 ↪in the console
import numpy as np #Creating Matrix
import time #For time related calculation - Not necessity for the␣
 ↪algorithm.
```

Splitting the video into frames (image files):

[10]:

```
def video_to_frames(location_input, location_output): #Convert Video to␣
 ↪frame
    try:

        os.mkdir( location_output)

    except OSError:

        pass

    startoftime = time.time() # To get the time taken to complete the␣
 ↪process

    capture = cv2.VideoCapture(location_input) #Capturing the video feed

    vid_length = int(capture.get(cv2.CAP_PROP_FRAME_COUNT)) - 1 #Number␣
 ↪of frame
    print ("Number of frames: ", vid_length) #Print number of frame
    counter = 0
    print ("Converting video..\n") #Print the process
    # Start converting the video
    while cap.isOpened():

        rete, frame = cap.read()

        cv2.imwrite( location_output + "/%#05d.jpg" % (counter+1), frame)
        counter = counter + 1

        if (counter > (vid_length-1)):

            endoftime = time.time()

            cap.release()

            print ("Done extracting frames.\n%d frames extracted" %␣
 ↪counter) #Final Results
            print ("It took %d seconds forconversion." % (endoftime-␣
 ↪startoftime)) #time taken by the process
            break

if __name__=="__main__": #Main call to the function
```

```
    location_input = '/locIn.mp4' #Enter input location
    location_output = '/locOut' #Enter output location
    video_to_frames(location_input, location_output)
```

```
Number of frames:  930
Converting video..

Done extracting frames.
930 frames extracted
It took 66 seconds forconversion.
```

Importing the frames to the model

[19]:

```
Input_Loc='/Input_Location_folder_address' #Location of folder where the␣
 ↪frames are stored

collected_frames = os.listdir(Input_Loc) #Collecting names of the frame␣
 ↪in the folder
collected_frames.sort(key=lambda f: int(re.sub('\D', '', f))) #To sort␣
 ↪them in order
collected_images=[] #To collect the images from the folder

for i in col_frames: #Iterate over the name of the frames from the folder
    img = cv2.imread('E:/Whatsapp data/whatsapp data/Car Track counter/
 ↪DATA/Frame/'+i) #To read frames

    collected_images.append(img) #Adding the frames in the␣
 ↪collected_images=[] list
```

Pre-Processing

[20]:

```
font = cv2.FONT_HERSHEY_DUPLEX #Fonts to write the number of detected␣
 ↪vehicles.
```

[21]:

```
Framekernel = np.ones((4,4),np.uint8) #Kernal for the dilation.
```

[22]:

```
pathIn = "E:/Whatsapp data/whatsapp data/Car Track counter/Output/Out/"␣
 ↪#Manupulated output for frames

for i in range(len(col_images)-1): #Iteration for Frame Differencing by␣
 ↪greyscalling and finding difference in the frames
    num=int(0)

    GreyFrameA = cv2.cvtColor(collected_images[i], cv2.COLOR_BGR2GRAY)
    GreyFrameB = cv2.cvtColor(collected_images[i+1], cv2.COLOR_BGR2GRAY)
    FrameDiffer = cv2.absdiff(GreyFrameB , GreyFrameA )
    rete, Framethreshold = cv2.threshold(FrameDiffer, 20, 255, cv2.
↪THRESH_BINARY) #Applying image thresholding to differentiated frames.


    Framedilated = cv2.dilate(Framethreshold ,Framekernel,iterations = 1)␣
↪#Applying dilation based on the defined Kernel
```

Processing Data

[25]:

```
"""We will create contours based on the difference in frames and validate␣
↪the correct required contours"""
    contour, hierarchies = cv2.findContours(Framedilated.copy(), cv2.
↪RETR_TREE,cv2.CHAIN_APPROX_NONE)

    # shortlist contours appearing in the detection zone
    valid_contour = []

    for runner in contour:
        x,y,w,h = cv2.boundingRect(runner) #Creates bounding rectangles␣
↪for the contours
        if (x <= 1750 ) & (x>= 0) & (y >= 400) & (cv2.contourArea(runner)␣
↪>= 400*20): #Criteria for contour selection
            if (y >= 420) & (cv2.contourArea(runner) < 390):
                break
            valid_contour.append(runner) #Validated contours

    """"Image processing to draw line and validated contours on the␣
↪frame"""
    drawframe = col_images[i].copy()
    cv2.drawContours(drawframe, valid_contour, -1, (0,0,0), 20) ␣
↪#Thickness=20
    cv2.line(dmy, (0, 200),(1750,200),(0, 0, 0),20)

    """Text on the frame"""
    cv2.putText(dmy, "vehicles detected: " + str(len(valid_cntrs)), (55,␣
↪15), font, 0.6, (0, 180, 0), 2)

    """exporting the frames"""
    cv2.imwrite(pathIn+str(i)+'.png',drawframe )

    """or we can directly manupulate the data and compile the frames into␣
↪video. Here we first store the frames and later
    combine them, not recommended for Larger files"""
```

Export [Compiling the frames into a video]

[26]:

```
frame_folder = '/Frame/Adress' #Add the image folder
name_video = 'video_name.avi' #Name of the video

images = [img for img in os.listdir(frame_folder) if img.endswith(".png")]

images.sort(key=lambda f: int(re.sub('\D', '', f))) #to sort the frame so␣
 ↪that they compile in correct order
frame = cv2.imread(os.path.join(frame_folder, images[0]))

height, width, layers = frame.shape #to get the dimension for the video

video = cv2.VideoWriter(name_video, 0, 12, (width,height)) #frame rate=␣
 ↪12 fps

for image in images:
    video.write(cv2.imread(os.path.join(frame_folder, image)))

cv2.destroyAllWindows()
video.release()
```

The overall algorithm can be converted into:

[28]:

```
import os
from os.path import isfile, join
import re
import cv2
import matplotlib.pyplot as plt
import numpy as np
import time



def video_output(frame_folder, name_video ): #Convert frame to video

    images = [img for img in os.listdir(frame_folder) if img.endswith(".
 ↪png")]

    images.sort(key=lambda f: int(re.sub('\D', '', f))) #to sort the␣
 ↪frame so that they compile in correct order
    frame = cv2.imread(os.path.join(frame_folder, images[0]))

    height, width, layers = frame.shape #to get the dimension for the␣
 ↪video

    video = cv2.VideoWriter(name_video, 0, 12, (width,height)) #frame␣
 ↪rate= 12 fps
```

```
    for image in images:
        video.write(cv2.imread(os.path.join(frame_folder, image)))

    cv2.destroyAllWindows()
    video.release()

#######################################################################################

def video_process(Input_Loc,pathIn):

    collected_frames = os.listdir(Input_Loc) #Collecting names of the␣
 ↪frame in the folder
    collected_frames.sort(key=lambda f: int(re.sub('\D', '', f))) #To␣
 ↪sort them in order
    collected_images=[] #To collect the images from the folder
    font = cv2.FONT_HERSHEY_DUPLEX #Fonts to write the number of detected␣
 ↪vehicles.
    for i in collected_frames: #Iterate over the name of the frames from␣
 ↪the folder
        img = cv2.imread(Input_Loc+i) #To read frames

        collected_images.append(img) #Adding the frames in the␣
 ↪collected_images=[] list

        for i in range(len(collected_images)-1):


            GreyFrameA = cv2.cvtColor(collected_images[i], cv2.
 ↪COLOR_BGR2GRAY)
            GreyFrameB = cv2.cvtColor(collected_images[i+1], cv2.
 ↪COLOR_BGR2GRAY)
            FrameDiffer = cv2.absdiff(GreyFrameB , GreyFrameA )


            rete, Framethreshold = cv2.threshold(FrameDiffer, 20, 255,␣
 ↪cv2.THRESH_BINARY)
            Framekernel = np.ones((4,4),np.uint8) #Kernal for the␣
 ↪dilation.
            Framedilated = cv2.dilate(Framethreshold␣
 ↪,Framekernel,iterations = 1)


            contour, hierarchies = cv2.findContours(Framedilated.copy(),␣
 ↪cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
```

```
            # shortlist contours appearing in the detection zone
            valid_contour = []

            for runner in contour:
                x,y,w,h = cv2.boundingRect(runner) #Creates bounding␣
 ↪rectangles for the contours
                if (x <= 1750 ) & (x>= 0) & (y >= 400) & (cv2.
 ↪contourArea(runner) >= 400*20): #Criteria for contour selection
                    if (y >= 420) & (cv2.contourArea(runner) < 390):
                        break
                    valid_contour.append(runner) #Validated contours


            drawframe = collected_images[i].copy()
            cv2.drawContours(drawframe, valid_contour, -1, (0,0,0), 20) ␣
 ↪#Thickness=20
            cv2.line(drawframe, (0, 200),(1750,200),(0, 0, 0),20)
            cv2.putText(drawframe, "vehicles detected: " +␣
 ↪str(len(valid_contour)), (55, 15), font, 0.6, (0, 180, 0), 2)
            cv2.imwrite(pathIn+str(i)+'.png',drawframe )

    """or we can directly manupulate the data and compile the frames into␣
 ↪video. Here we first store the frames and later
    combine them, not recommended for Larger files"""

#######################################################################################

def video_to_frames(location_input, location_output): #Convert Video to␣
 ↪frame
    try:

        os.mkdir( location_output)

    except OSError:

        pass

    startoftime = time.time() # To get the time taken to complete the␣
 ↪process

    capture = cv2.VideoCapture(location_input) #Capturing the video feed

    vid_length = int(capture.get(cv2.CAP_PROP_FRAME_COUNT)) - 1 #Number␣
 ↪of frame
    print ("Number of frames: ", vid_length) #Print number of frame
    counter = 0
    print ("Converting video..\n") #Print the process
    # Start converting the video
    while capture.isOpened():

        rete, frame = capture.read()

        cv2.imwrite( location_output + "/%#05d.jpg" % (counter+1), frame)
        counter = counter + 1

        if (counter > (vid_length-1)):

            endoftime = time.time()

            capture.release()

            print ("Done extracting frames.\n%d frames extracted" %␣
 ↪counter) #Final Results
            print ("It took %d seconds forconversion." % (endoftime-␣
 ↪startoftime)) #time taken by the process
            break

#######################################################################################

if __name__=="__main__": #Main call to the function

    location_input = '/locIn.mp4' #Enter input location of video data
    location_output = '/locOut' #Enter output location for splitted frames
    output_frames='/loc' #Enter location to save the processed frames
    name_video='Name.avi' #Enter the name for output of the video.

    video_to_frames(location_input, location_output)
    video_process(location_output,output_frames)
    video_output(output_frames, name_video)
```

Conclusion:

1) The model can be made more accurate by applying more logical contour conditions.
2) The model should have different threshold value for better accuracy at night and at day.
3) The model do not need training dataset and yet it shows good results.