Pollution Under Control Authentication (PUCA)

A Synopsis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Bachelor's in Technology

In

Computer Science and Engineering

by

Abhishek Shukla (1707710002)

Yash Mishra (1707710062)

Udityanshu Singh (1707710057)

Paras Aggarwal (1707710036)

CSE DEPARTMENT

Dr. Kedar Nath Modi Institute of Engineering and Technology

(AKTU)

August-7, 2021

PUCA

TIME TO INNOVATE

PROJECT REPORT

PUCA is an innovative idea to regulate freely and smoothly regulate Transport Department and Regional Transport Office. And most importantly to make our environment clean and livable and to ccontibute towards **CLEAN INDIA-HELTHY INDIA**

August-7, 2021

By

Abhishek Shukla
Udityanshu Singh
Paras Agarwal
Yash Mishra
The Innovative Team

Project Introduction

The objective of this project is to validate the document issued by the Regional Transport Offices (RTO) of Government of India. This Algorithm helps in solving the problem of spoofing the authority and regulates the system smoothly, efficiently and more accurately.

This Algorithm is designed using Python, Computer Vison and its inbuilt modules. Let’s discuss this project further in brief.

A few years ago, the creation of the software and hardware image processing systems was mainly limited to the development of the user interface, which most of the programmers of each firm were engaged in. The situation has been significantly changed with the advent of the Windows operating system when the majority of the developers switched to solving the problems of image processing itself. However, this has not yet led to the cardinal progress in solving typical tasks of recognizing faces, car numbers, road signs, analyzing remote and medical images, etc. Each of these "eternal" problems is solved by trial and error by the efforts of numerous groups of the engineers and scientists. As modern technical solutions are turn out to be excessively expensive, the task of automating the creation of the software tools for solving intellectual problems is formulated and intensively solved abroad. In the field of image processing, the required tool kit should be supporting the analysis and recognition of images of previously unknown content and ensure the effective development of applications by ordinary programmers. Just as the Windows toolkit supports the creation of interfaces for solving various applied problems.

Object recognition is to describe a collection of related computer vision tasks that involve activities like identifying objects in digital photographs. Image classification involves activities such as predicting the class of one object in an image. Object localization is referring to identifying the location of one or more objects in an image and drawing an abounding box around their extent. Object detection does the work of combines these two tasks and localizes and classifies one or more objects in an image. When a user or practitioner refers to the term “object recognition “, they often mean “object detection “. It may be challenging for beginners to distinguish between different related computer vision tasks.

So, we can distinguish between these three computer vision tasks with this example:

Image Classification: This is done by Predict the type or class of an object in an image.

Input: An image which consists of a single object, such as a photograph.

Output: A class label (e.g., one or more integers that are mapped to class labels).

Object Localization: This is done through, Locate the presence of objects in an image and indicate their location with a bounding box.

Input: An image which consists of one or more objects, such as a photograph.

Output: One or more bounding boxes (e.g., defined by a point, width, and height).

Object Detection: This is done through, Locate the presence of objects with a bounding box and types or classes of the located objects in an image.

Input: An image which consists of one or more objects, such as a photograph.

Output: One or more bounding boxes (e.g., defined by a point, width, and height), and a class label for each bounding box.

One of the further extensions to this breakdown of computer vision tasks is object segmentation, also called "object instance segmentation" or "semantic segmentation," where instances of recognized objects are indicated by highlighting the specific pixels of the object instead of a coarse bounding box. From this breakdown, we can understand that object recognition refers to a suite of challenging computer vision tasks.

For example, image classification is simply straight forward, but the differences between object localization and object detection can be confusing, especially when all three tasks may be just as equally referred to as object recognition.

Humans can detect and identify objects present in an image. The human visual system is fast and accurate and can also perform complex tasks like identifying multiple objects and detect obstacles with little conscious thought. The availability of large sets of data, faster GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy. We need to understand terms such as object detection, object localization, loss function for object detection and localization, and finally explore an object detection algorithm known as "You only look once" (YOLO).

Image classification also involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is always more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all these problems are referred to as object recognition.

Object recognition refers to a collection of related tasks for identifying objects in digital photographs. Region-based Convolutional Neural Networks, or R-CNNs, is a family of techniques for addressing object localization and recognition tasks, designed for model performance. You Only Look Once, or YOLO is known as the second family of techniques for object recognition designed for speed and real-time use.

A. Video Processing

Overview

- Excited by the idea of smart cities? You'll love this tutorial on building your own vehicle detection system
- We'll first understand how to detect moving objects in a video before diving into the implementation part
- We'll be using OpenCV and Python to build the automatic vehicle detector

Introduction

I love the idea of smart cities. The thought of automated smart energy systems, electrical grids, one-touch access ports – it's an enthralling concept! Honestly, it's a dream for a data scientist and I'm delighted that a lot of cities around the world are moving towards becoming smarter.

One of the core components of a smart city is automated traffic management. And that got me thinking – could I use my data science chops to build a vehicle detection model that could play a part in smart traffic management?

Think about it – if you could integrate a vehicle detection system in a traffic light camera, you could easily track a number of useful things simultaneously:

- How many vehicles are present at the traffic junction during the day?
- What time does the traffic build up?
- What kind of vehicles are traversing the junction (heavy vehicles, cars, etc.)?
- Is there a way to optimize the traffic and distribute it through a different street?

And so on. The applications are endless!

Us humans can easily detect and recognize objects from complex scenes in a flash. Translating that thought process to a machine, however, requires us to learn the art of object detection using computer vision algorithms.

So, in this article, we will be building an automatic vehicle detector and counter model. Here's a taste of what you can expect:

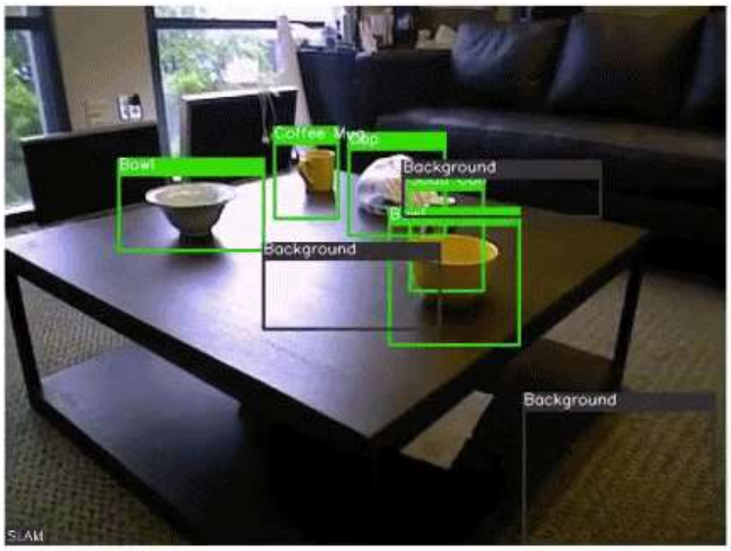
Excited? Let's turn on the ignition and take this for a spin!

Table of Contents

The Idea Behind Detecting Moving Objects in Videos

Object detection is a fascinating field in computer vision. It goes to a whole new level when we're dealing with video data. The complexity rises up a notch, but so do the rewards!

We can perform super useful high-value tasks such as surveillance, traffic management, fighting crime, etc. using object detection algorithms. Here's a GIF demonstrating the idea:



There are a number of sub-tasks we can perform in object detection, such as counting the number of objects, finding the relative size of the objects, or finding the relative distance between the objects. All these sub-tasks are important as they contribute to solving some of the toughest real-world problems.

If you're looking to learn about object detection from scratch, I recommend these tutorials:

- A Step-by-Step Introduction to the Basic Object Detection Algorithms
- Real-Time Object Detection using SlimYOLOv3
- Other Object Detection Articles and Resources

Let's look at some of the exciting real-world use cases of object detection.

Real-World Use Cases of Object Detection in Videos

Nowadays, video object detection is being deployed across a wide range of industries. The use cases range from video surveillance to sports broadcasting to robot navigation.

Here's the good news – the possibilities are endless when it comes to future use cases for video object detection and tracking. Here I've listed down some of the interesting applications:

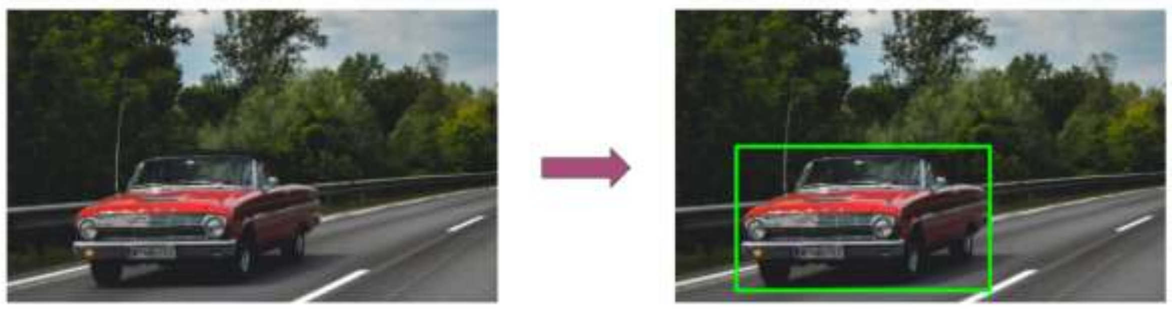
1. Crowd counting
2. Vehicle number plate detection and recognition
3. Ball tracking in Sports
4. Robotics
5. Traffic management (an idea we'll see in this article)

Essential Concepts you should know about Video Object Detection

There are certain key concepts you should know before getting started with building a video detection system. Once you are familiar with these basic concepts, you would be able to build your own detection system for any use case of your choice.

So, how would you like to detect a moving object in a video?

Our objective is to capture the coordinates of the moving object and highlight that object in the video. Consider this frame from a video below:



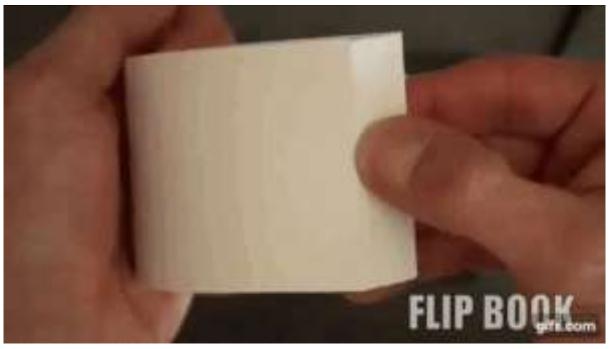
We would want our model to detect the moving object in a video as illustrated in the image above. The moving car is detected and a bounding box is created surrounding the car.

There are multiple techniques to solve this problem. You can train a deep learning model for object detection or you can pick a pre-trained model and fine-tune it on your data. However, these are supervised learning approaches and they require labeled data to train the object detection model.

In this article, **we will focus on the unsupervised way of object detection in videos, i.e., object detection without using any labeled data**. We will use the technique of **frame differencing.** Let's understand how it works!

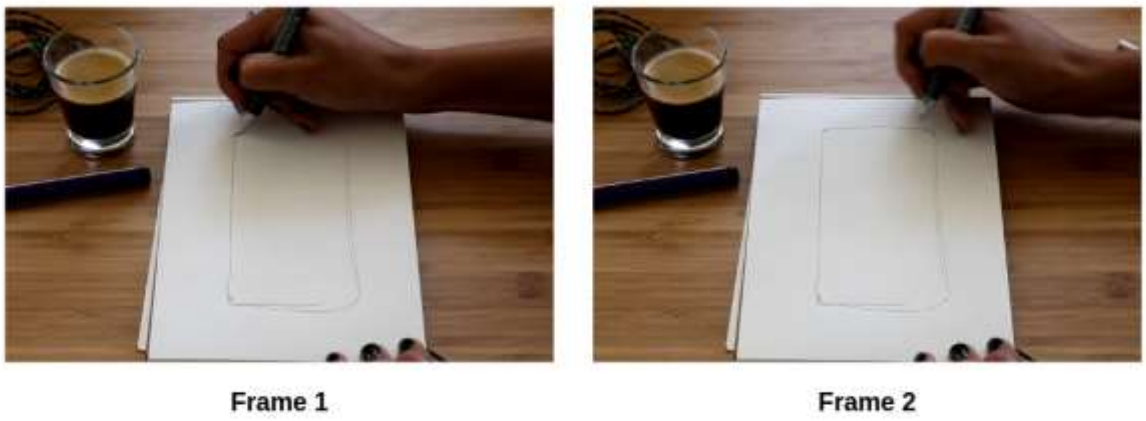
Frame Differencing

A video is a set of frames stacked together in the right sequence. So, when we see an object moving in a video, it means that the object is at a different location at every consecutive frame.



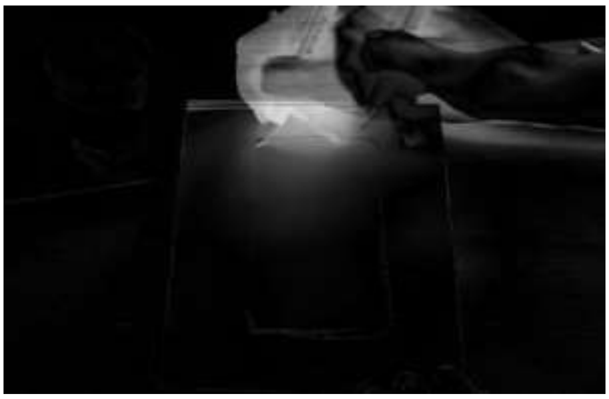
If we assume that apart from that object nothing else moved in a pair of consecutive frames, then the pixel difference of the first frame from the second frame will highlight the pixels of the moving object. Now, we would have the pixels and the coordinates of the moving object. This is broadly how the frame differencing method works.

Let's take an example. Consider the following two frames from a video:



Can you spot the difference between the two frames?

Yes – it is the position of the hand holding the pen that has changed from frame 1 to frame 2. The rest of the objects have not moved at all. So, as I mentioned earlier, to locate the moving object, we will perform frame differencing. The result will look like this:

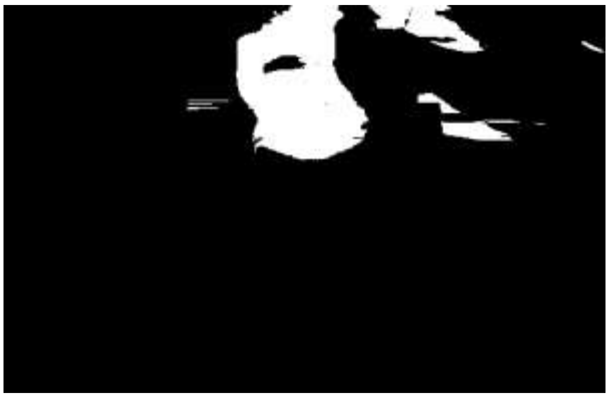


You can see the highlighted or the white region where the hand was present initially. Apart from that, the notepad is also highlighted a bit along its edges. This could be due to the change in the illumination by the movement of the hand. It is advisable to get rid of unwanted detection of stationary objects. Therefore, we would need to perform certain image pre-processing steps on the frames.

Image Thresholding

In this method, the pixel values of a grayscale image are assigned one of the two values representing black and white colors based on a threshold. So, if the value of a pixel is greater than a threshold value, it is assigned one value, else it is assigned the other value.

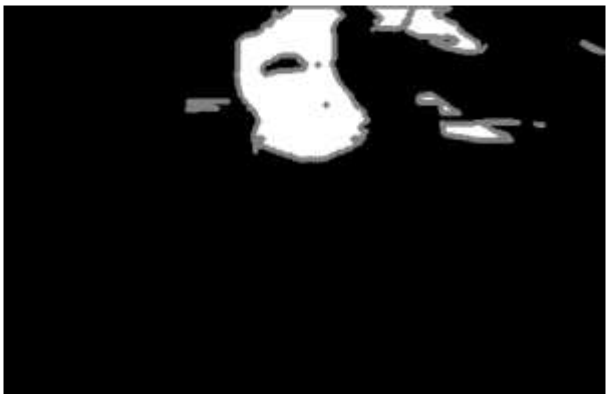
In our case, we will apply image thresholding on the output image of the frame differencing in the previous step:



You can see that a major part of the unwanted highlighted area has gone. The highlighted edges of the notepad are not visible anymore. The resultant image can also be called as a binary image as there are only two colors in it. In the next step, we will see how to capture these highlighted regions.

Finding Contours

The contours are used to identify the shape of an area in the image having the same color or intensity. Contours are like boundaries around regions of interest. So, if we apply contours on the image after the thresholding step, we would get the following result:



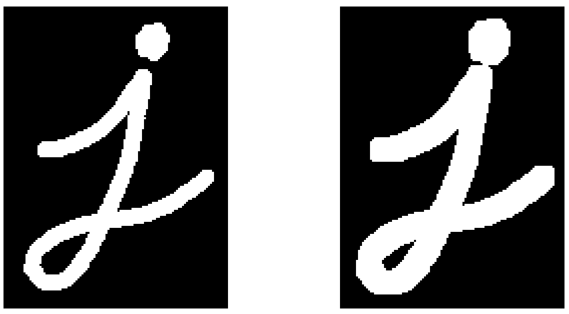
The white regions have been surrounded by grayish boundaries which are nothing but contours. We can easily get the coordinates of these contours. This means we can get the locations of the highlighted regions.

Note that there are multiple highlighted regions and each region is encircled by a contour. In our case, the contour having the maximum area is the desired region. Hence, it is better to have as few contours as possible.

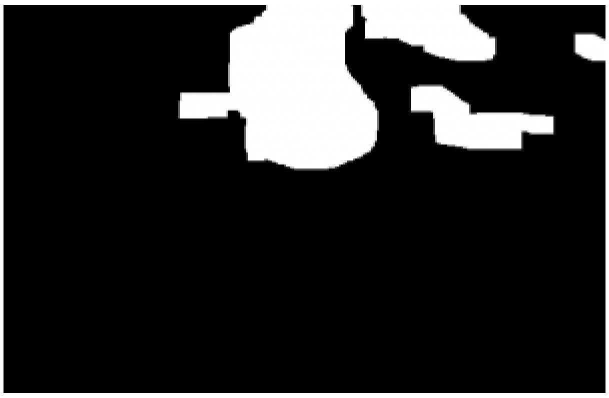
In the image above, there are still some unnecessary fragments of the white region. There is still scope of improvement. The idea is to merge the nearby white regions to have fewer contours and for that, we can use another technique known as image dilation.

Image Dilation

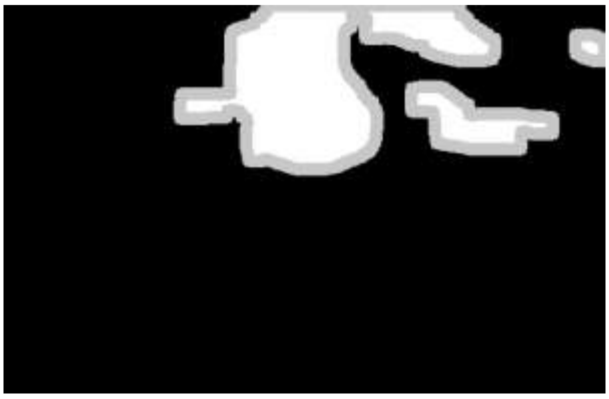
This is a convolution operation on an image wherein a kernel (a matrix) is passed over the entire image. Just to give you intuition, the image on the right is the dilated version of the image on the left:



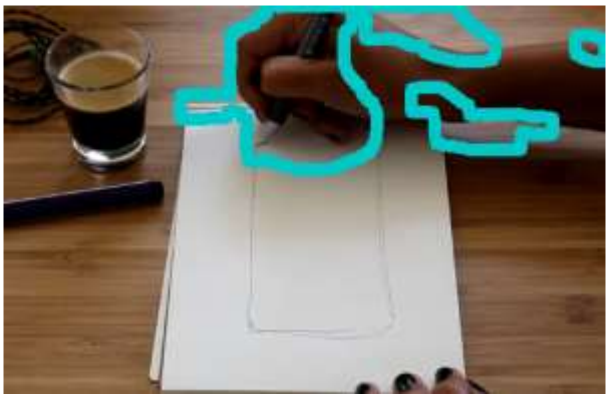
So, let’s apply image dilation to our image and then we will again find the contours:



It turns out that a lot of the fragmented regions have fused into each other. Now we can again find the contours in this image:



Here, we have only four candidate contours from which we would select the one with the largest area. You can also plot these contours on the original frame to see how well the contours are surrounding the moving object:



Build a Vehicle Detection System using OpenCV and Python

We are all set to build our vehicle detection system! We will be using the computer vision library OpenCV (version – 4.0.0) a lot in this implementation. Let's first import the required libraries and the modules.

Import Libraries

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

view raw **obj_detect_import_lib.py** hosted with by **GitHub**

Import Video Frames

Please download the frames of the original video from this link.

Keep the frames in a folder named “frames” inside your working directory. From that folder, we will import the frames and keep them in a list:

```
# get file names of the frames
col_frames = os.listdir('frames/')

# sort file names
col_frames.sort(key=lambda f: int(re.sub('\D', '', f)))

# empty list to store the frames
col_images=[]

for i in col_frames:
    # read the frames
    img = cv2.imread('frames/'+i)
    # append the frames to the list
    col_images.append(img)
```

view raw **obj_detect_read_frames.py** hosted with by **GitHub**

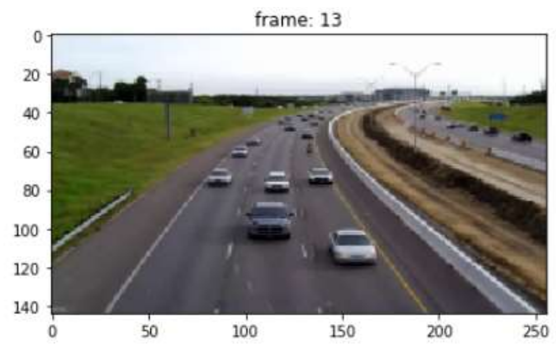
Data Exploration

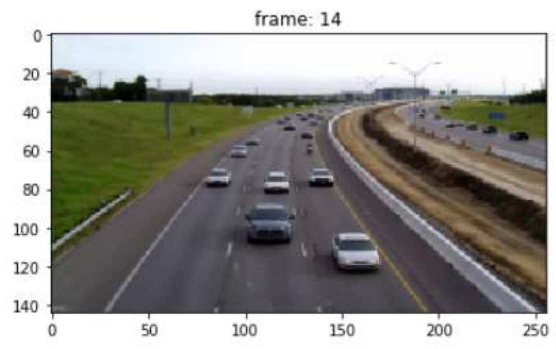
Let’s display two consecutive frames:

```
# plot 13th frame
i = 13

for frame in [i, i+1]:
    plt.imshow(cv2.cvtColor(col_images[frame], cv2.COLOR_BGR2RGB))
    plt.title("frame: "+str(frame))
    plt.show()
```

view raw **obj_detect_display.py** hosted with by **GitHub**



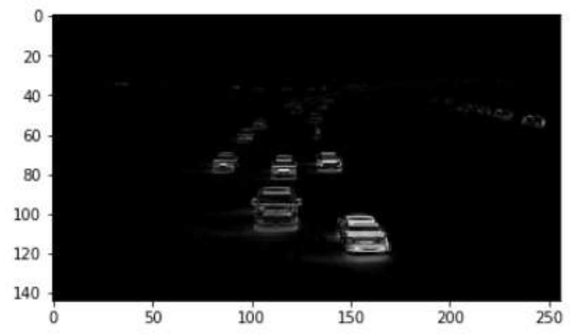


It is hard to find any difference in these two frames, isn't it? As discussed earlier, taking the difference of the pixel values of two consecutive frames will help us observe the moving objects. So, let's use the technique on the above two frames:

```
# convert the frames to grayscale
grayA = cv2.cvtColor(col_images[i], cv2.COLOR_BGR2GRAY)
grayB = cv2.cvtColor(col_images[i+1], cv2.COLOR_BGR2GRAY)

# plot the image after frame differencing
plt.imshow(cv2.absdiff(grayB, grayA), cmap = 'gray')
plt.show()
```

view rawobj_detect_frame_diff.py hosted with by GitHub



Now we can clearly see the moving objects in the 13th and 14th frames. Everything else that was not moving has been subtracted out.

Image Pre-processing

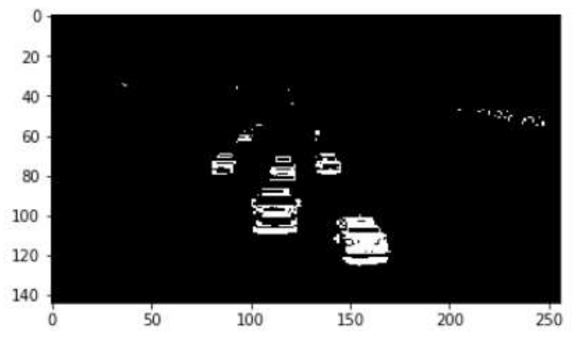
Let’s see what happens after applying thresholding to the above image:

```
diff_image = cv2.absdiff(grayB, grayA)

# perform image thresholding
ret, thresh = cv2.threshold(diff_image, 30, 255, cv2.THRESH_BINARY)

# plot image after thresholding
plt.imshow(thresh, cmap = 'gray')
plt.show()
```

view rawobj_detect_threshold.py hosted with by GitHub

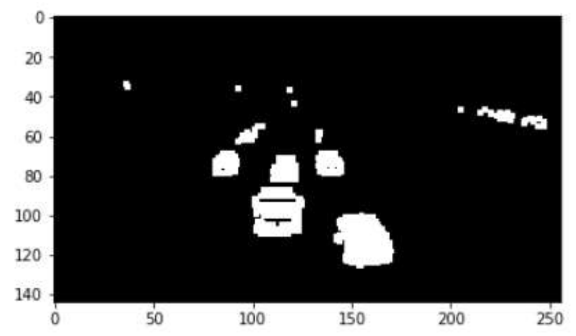


Now, the moving objects (vehicles) look more promising and most of the noise (undesired white regions) are gone. However, the highlighted regions are a bit fragmented. So, we can apply image dilation over this image:

```
# apply image dilation
kernel = np.ones((3,3),np.uint8)
dilated = cv2.dilate(thresh,kernel,iterations = 1)

# plot dilated image
plt.imshow(dilated, cmap = 'gray')
plt.show()
```

view rawobj_detect_dilation.py hosted with by GitHub



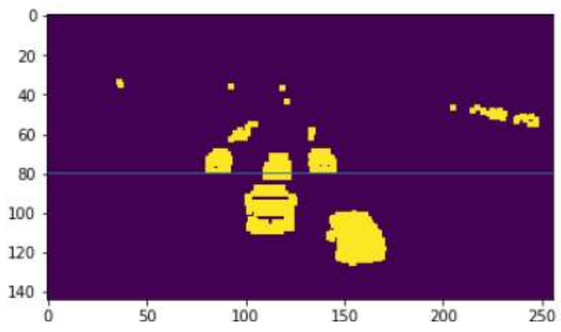
The moving objects have more solid highlighted regions. Hopefully, the number of contours for every object in the frame will not be more than three.

However, we are not going to use the entire frame to detect moving vehicles. We will first select a zone, and if a vehicle moves into that zone, then only it will be detected.

So, let me show you the zone that we will be working with:

```
# plot vehicle detection zone
plt.imshow(dilated)
cv2.line(dilated, (0, 80),(256,80),(100, 0, 0))
plt.show()
```

view rawobj_detect_zone.py hosted with by **GitHub**



The area below the horizontal line y = 80 is our vehicle detection zone. We will detect any movement that happens in this zone only. You can create your own detection zone if you want to play around with the concept.

Now let's find the contours in the detection zone of the above frame:

```
# find contours


contours, hierarchy = cv2.findContours(thresh.copy(),cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
```

The code above finds all the contours in the entire image and keeps them in the variable '*contours*'. Since we have to find only those contours that are present in the detection zone, we will apply a couple of checks on the discovered contours.

The first check is whether the top-left y-coordinate of the contour should be >= 80 (I am including one more check, x-coordinate <= 200). The other check is that the area of the contour should be >= 25. You can find the contour area with the help of the *cv2.contourArea()* function.

```
valid_cntrs = []


for i,cntr in enumerate(contours):
    x,y,w,h = cv2.boundingRect(cntr)
    if (x <= 200) & (y >= 80) & (cv2.contourArea(cntr) >= 25):
        valid_cntrs.append(cntr)


# count of discovered contours
```

```
len(valid_cntrs)
```

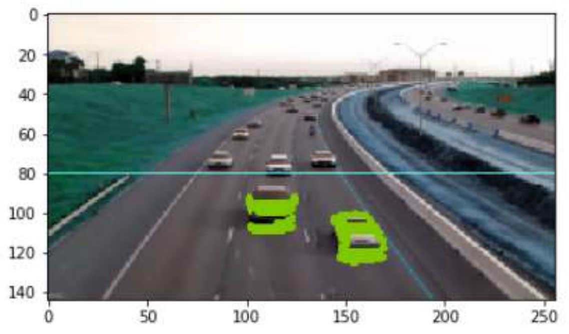
view rawobj_detect_valid_cntrs.py hosted with by GitHub

Next, let's plot the contours along with the original frame:

```
dmy = col_images[13].copy()

cv2.drawContours(dmy, valid_cntrs, -1, (127,200,0), 2)
cv2.line(dmy, (0, 80),(256,80),(100, 255, 255))
plt.imshow(dmy)
plt.show()
```

view rawobj_detect_plot_cntrs.py hosted with by GitHub



Cool! Contours of only those vehicles that are inside the detection zone are visible. This is how we will detect vehicles in all the frames.

Vehicle Detection in Videos

It's time to apply the same image transformations and pre-processing operations on all the frames and find the desired contours. Just to reiterate, we will follow the below steps:

1. Apply frame differencing on every pair of consecutive frames
2. Apply image thresholding on the output image of the previous step
3. Perform image dilation on the output image of the previous step
4. Find contours in the output image of the previous step
5. Shortlist contours appearing in the detection zone
6. Save frames along with the final contours

```
7# kernel for image dilation

kernel = np.ones((4,4),np.uint8)

# font style
font = cv2.FONT_HERSHEY_SIMPLEX

# directory to save the ouput frames
pathIn = "contour_frames_3/"

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

    # frame differencing
    grayA = cv2.cvtColor(col_images[i], cv2.COLOR_BGR2GRAY)
    grayB = cv2.cvtColor(col_images[i+1], cv2.COLOR_BGR2GRAY)
    diff_image = cv2.absdiff(grayB, grayA)

    # image thresholding
    ret, thresh = cv2.threshold(diff_image, 30, 255, cv2.THRESH_BINARY)

    # image dilation
    dilated = cv2.dilate(thresh,kernel,iterations = 1)

    # find contours
    contours, hierarchy = cv2.findContours(dilated.copy(), cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)

    # shortlist contours appearing in the detection zone
    valid_cntrs = []
    for cntr in contours:
        x,y,w,h = cv2.boundingRect(cntr)
        if (x <= 200) & (y >= 80) & (cv2.contourArea(cntr) >= 25):
            if (y >= 90) & (cv2.contourArea(cntr) < 40):
                break
            valid_cntrs.append(cntr)

    # add contours to original frames
    dmy = col_images[i].copy()
    cv2.drawContours(dmy, valid_cntrs, -1, (127,200,0), 2)

    cv2.putText(dmy, "vehicles detected: " + str(len(valid_cntrs)), (55, 15), font, 0.6, (0, 180, 0), 2)
    cv2.line(dmy, (0, 80),(256,80),(100, 255, 255))
    cv2.imwrite(pathIn+str(i)+'.png',dmy)
```

Video Preparation

Here, we have added contours for all the moving vehicles in all the frames. It’s time to stack up the frames and create a video:

```
# specify video name


pathOut = 'vehicle_detection_v3.mp4'





# specify frames per second


fps = 14.0
```

Next, we will read the final frames in a list:

```
frame_array = []


files = [f for f in os.listdir(pathIn) if isfile(join(pathIn, f))]
```

```
files.sort(key=lambda f: int(re.sub('\D', '', f)))

for i in range(len(files)):
    filename=pathIn + files[i]

    #read frames
    img = cv2.imread(filename)
    height, width, layers = img.shape
    size = (width,height)

    #inserting the frames into an image array
    frame_array.append(img)
```

view rawobj_detect_read_final_frames.py hosted with by **GitHub**

Finally, we will use the below code to make the object detection video:

```
out = cv2.VideoWriter(pathOut,cv2.VideoWriter_fourcc(*'DIVX'), fps, size)

for i in range(len(frame_array)):
    # writing to a image array
    out.write(frame_array[i])
    out.release()
```

view rawobj_detect_export_video.py hosted with by GitHub

Congratulations on building your own vehicle object detection!

Detecting moving objects in Video

In this notebook, we will demonstrate techniques to detect moving objects in a video from a CCTV camera. The camera will be in a fixed position and does not move.

We will do the following:

1. Estimate the background with median filering
2. Show how to remove the background from a picture
3. Blurring and thresholding techniques
4. Detection of countours
5. Implementing the above process on a video

Import libraries

In [1]:

```
import numpy as np
import cv2

%matplotlib inline
from matplotlib import pyplot as plt

np.random.seed(42)
```

Define a routine to fix a color issue to ensure colors are seen correctly in matplotlib

In [2]:

```
#Routine to fix
def fixColor(image):
    return(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

Input

In [18]:

```
#Take a look at the input video
from IPython.display import Video
#Video("images/overpass.mp4", embed=True)

#Replace with your own video
```

Extract background in a video.

We identify a background we will have to use filtering techniques. An easy way to do this is to take a few random frames from the video and finding the median of it. We do this by finding the median of every pixel. This works because most of the time there is no vehicle passing over the road. The median will end filtering all moving objects in the video.

Note that this works if the camera is static and not moving.
In the section we capture the video in a video stream and get 30 random frames. These frames are saved in the array frames

In [4]:

```
video_stream = cv2.VideoCapture('images/overpass.mp4')

# Randomly select 30 frames
frameIds = video_stream.get(cv2.CAP_PROP_FRAME_COUNT) * np.random.uniform(size=30)

# Store selected frames in an array
frames = []
for fid in frameIds:
    video_stream.set(cv2.CAP_PROP_POS_FRAMES, fid)
    ret, frame = video_stream.read()
    frames.append(frame)

video_stream.release()
```

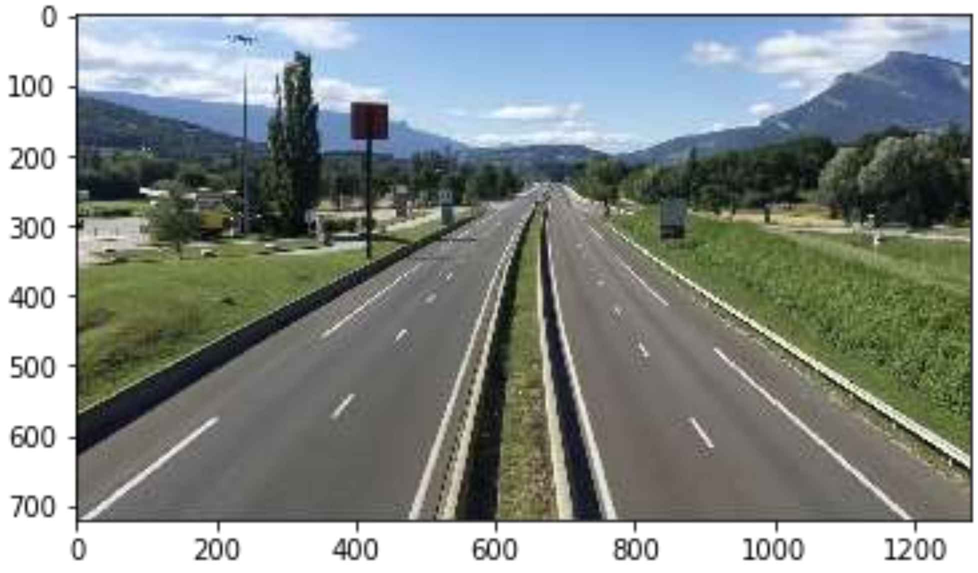
We will now calculate the median and average frames for saved frames. We will use the median as it remove outliers better.

In [5]:

```
# Calculate the median along the time axis
medianFrame = np.median(frames, axis=0).astype(dtype=np.uint8)
plt.imshow(fixColor(medianFrame))
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x7f71575ded68>
```

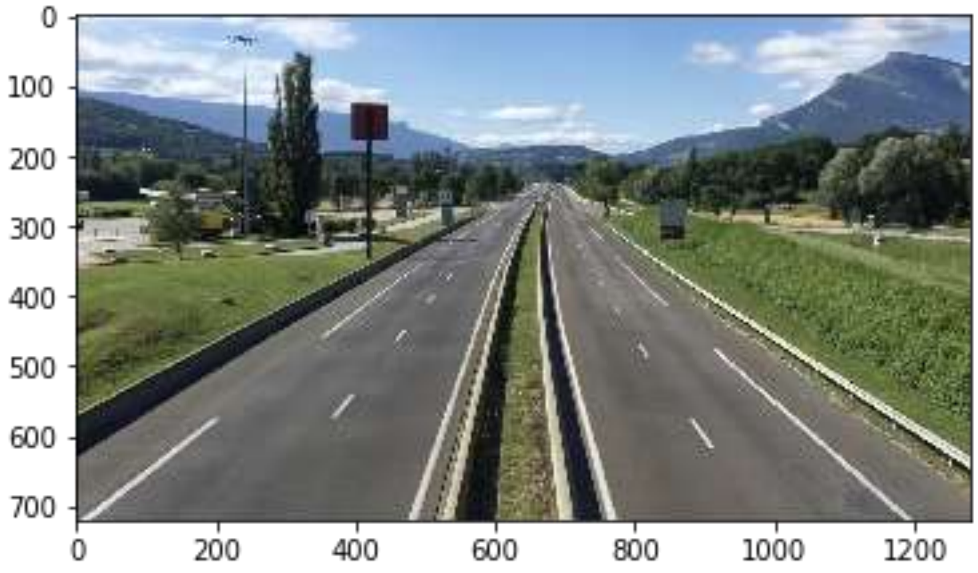


In [6]:

```
# Calculate the average along the time axis
avgFrame = np.average(frames, axis=0).astype(dtype=np.uint8)
plt.imshow(fixColor(avgFrame))
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x7f7157500908>
```



Processing a frame

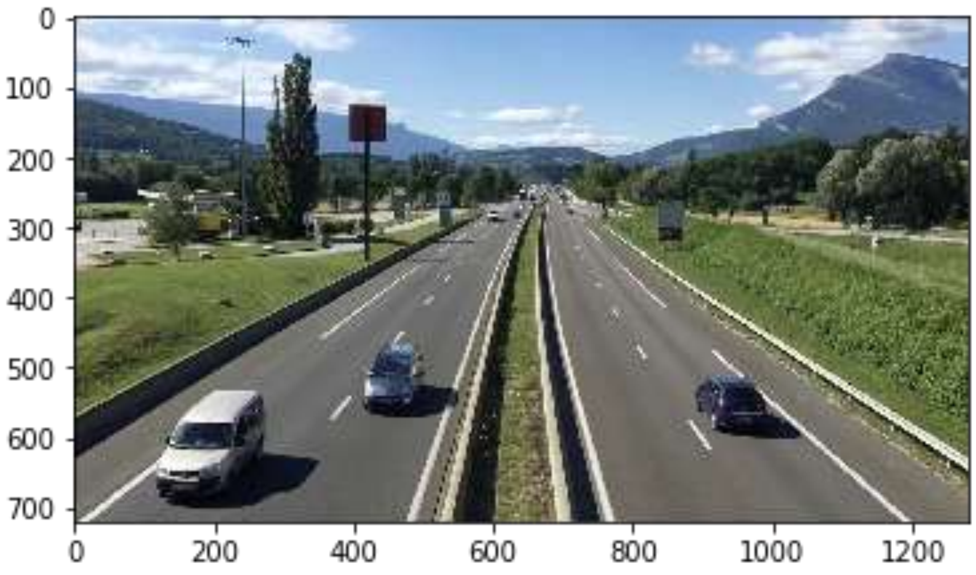
We will now show how to process a single photo explain all the steps needed. We will take the first frame from our frames array. We can see a few cars in the picture here.

In [7]:

```
sample_frame=frames[0]
plt.imshow(fixColor(sample_frame))
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x7f715401b0b8>
```



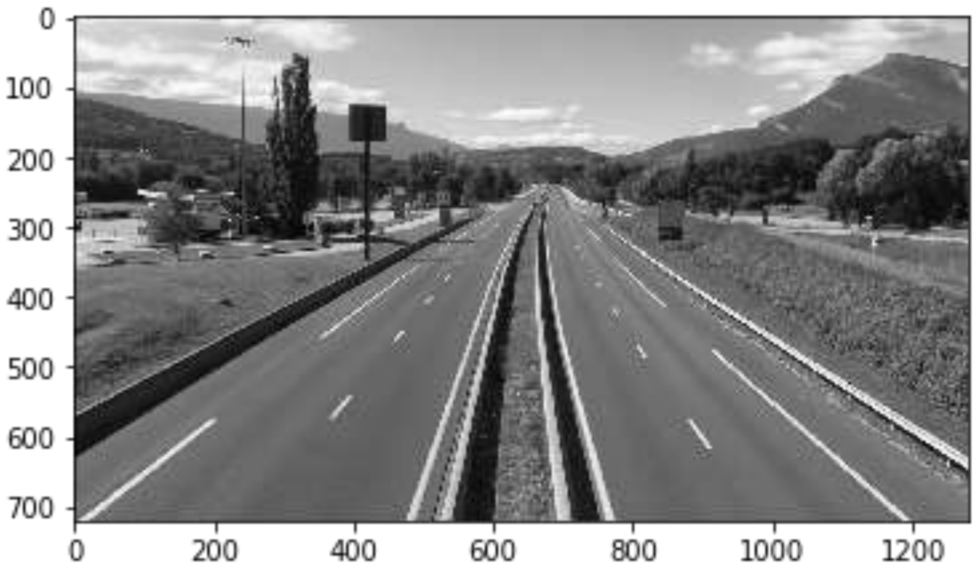
For identifying moving objects it is better to work with grayscale images. We will convert both the median image and sample image to grayscale

In [8]:

```
grayMedianFrame = cv2.cvtColor(medianFrame, cv2.COLOR_BGR2GRAY)
plt.imshow(fixColor(grayMedianFrame))
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x7f714dff6518>
```

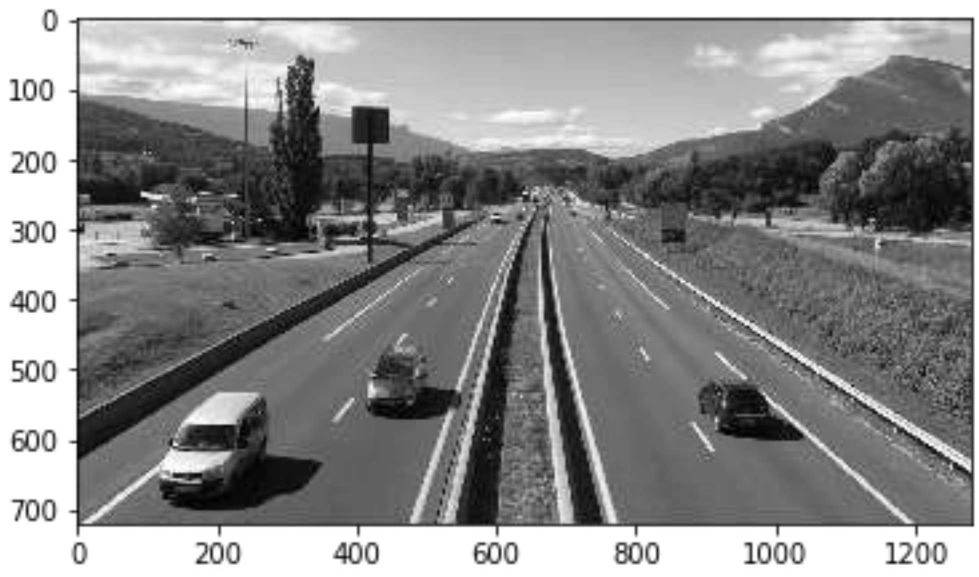


In [9]:

```
graySample=cv2.cvtColor(sample_frame, cv2.COLOR_BGR2GRAY)
plt.imshow(fixColor(graySample))
```

Out[9]:

```
<matplotlib.image.AxesImage at 0x7f714df66048>
```



Remove background

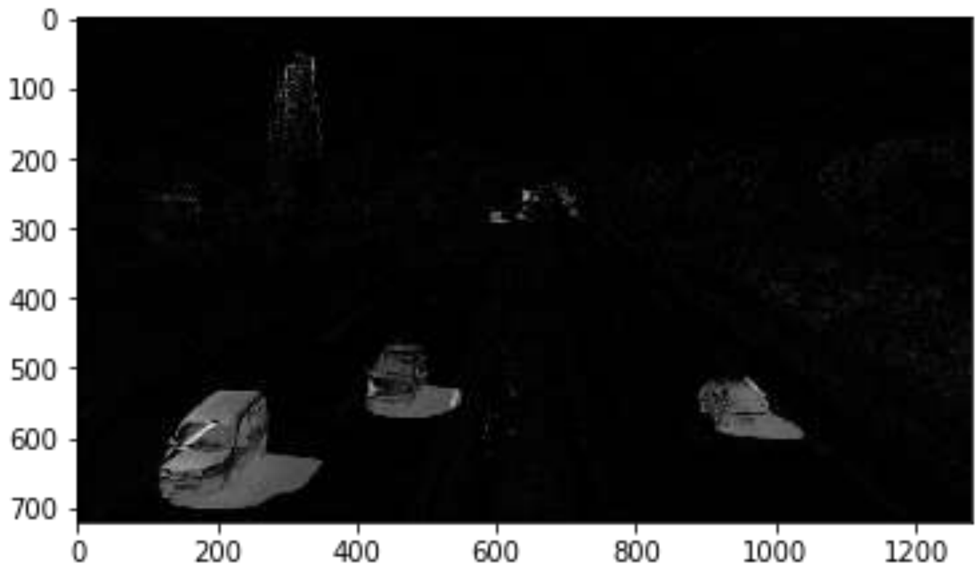
Remove the background from our sample. We can now see a ghost image with cars and background removed.

In [10]:

```
dframe = cv2.absdiff(graySample, grayMedianFrame)
plt.imshow(fixColor(dframe))
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x7f714df3a7b8>
```



Blurring

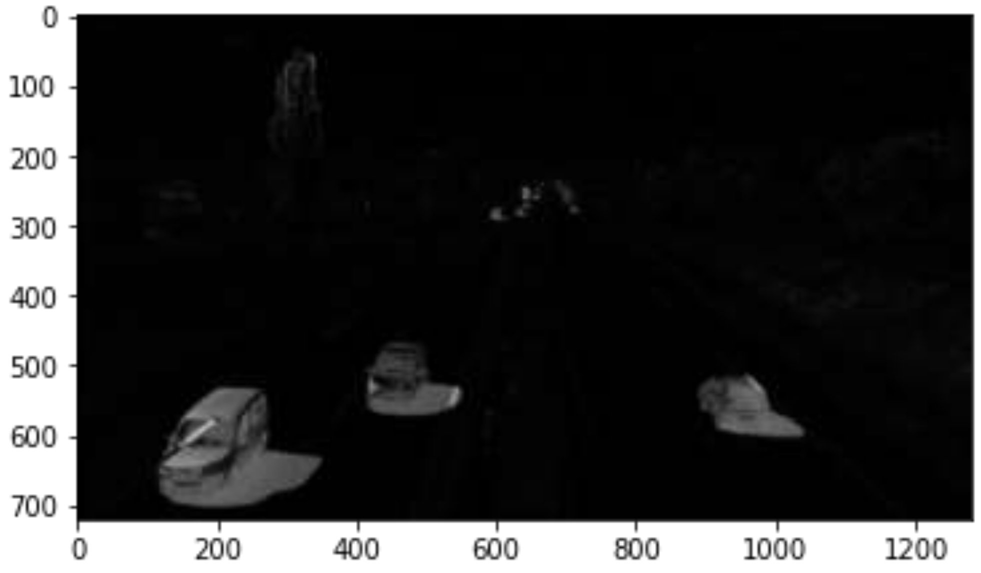
We will run Gaussian blurring to reduce noise and enable easier identification of edges

In [11]:

```
blurred = cv2.GaussianBlur(dframe, (11,11), 0)
plt.imshow(fixColor(blurred))
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x7f714de9ecc0>
```



Thresholding

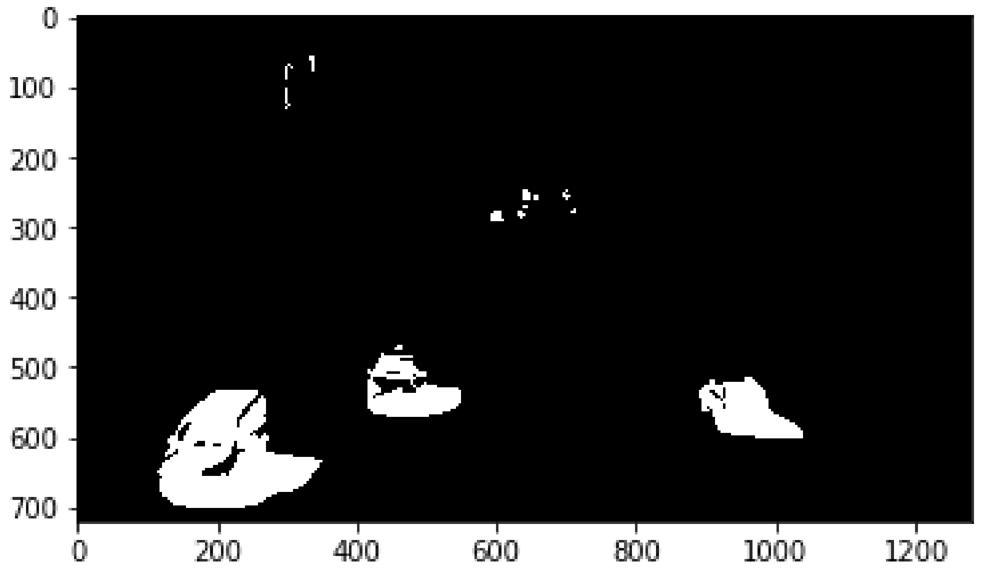
We will now run a threshold to clear bring out the objects left here. We will use OTSU thresholding which automatically figure out the correct thresold levels.

In [12]:

```
ret, tframe= cv2.threshold(blurred,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plt.imshow(fixColor(tframe))
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x7f714de0e1d0>
```



Contour and Bounding Boxes

We will create contours using the thresholded pictures

In [13]:

```
(cnts, _) = cv2.findContours(tframe.copy(), cv2.RETR_EXTERNAL,
                                            cv2 .CHAIN_APPROX_SIMPLE)
```

We will now create bounding boxes for the contours identified and show them on our sample images. We will disregard items like moving clouds in the top of our picture

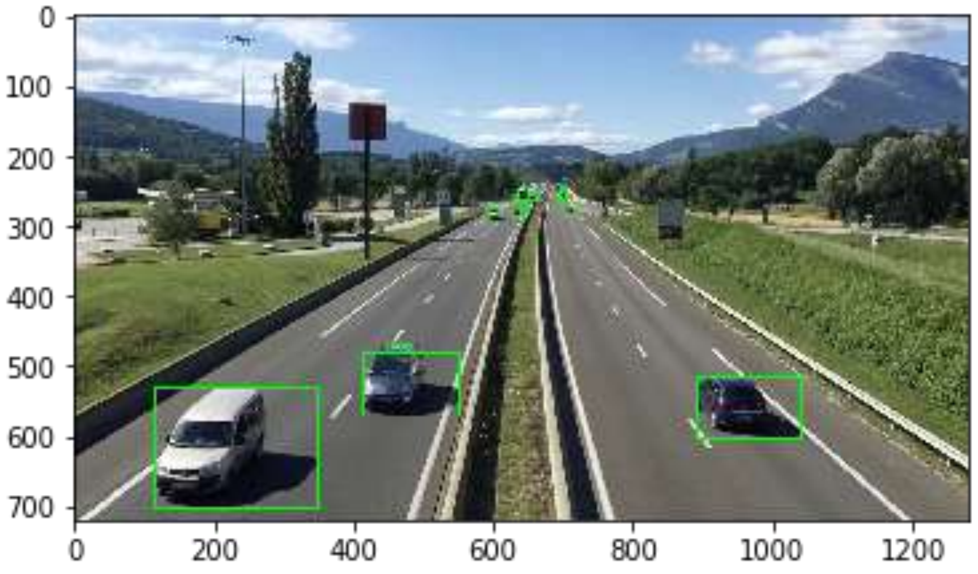
In [14]:

```
for cnt in cnts:
    x,y,w,h = cv2.boundingRect(cnt)
    if y > 200:  #Disregard item that are the top of the picture
        cv2.rectangle(sample_frame,(x,y),(x+w,y+h),(0,255,0),2)

plt.imshow(fixColor(sample_frame))
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x7f714ddedd30>
```



Putting in together for processing video

We will first declares the output video which will be create. We use in-built CV2 create an MP4 video, with 30 fps and frame size 640*480

In [15]:

```
writer = cv2.VideoWriter("output.mp4",
                         cv2.VideoWriter_fourcc(*"MP4V"), 30,(640,480))
```

In [16]:

```
#Create a new video stream and get total frame count
video_stream = cv2.VideoCapture('images/overpass.mp4')
total_frames=video_stream.get(cv2.CAP_PROP_FRAME_COUNT)
total_frames
```

Out[16]:

```
812.0
```

In [17]:

```
frameCnt=0
while(frameCnt < total_frames-1):
    frameCnt+=1
    ret, frame = video_stream.read()

    # Convert current frame to grayscale
    gframe = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Calculate absolute difference of current frame and
    # the median frame
    dframe = cv2.absdiff(gframe, grayMedianFrame)
    # Gaussian
    blurred = cv2.GaussianBlur(dframe, (11, 11), 0)
    #Thresholding to binarise
    ret, tframe= cv2.threshold(blurred,0,255,
                               cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    #Identifying contours from the threshold
    (cnts, _) = cv2.findContours(tframe.copy(),
                                 cv2.RETR_EXTERNAL, cv2 .CHAIN_APPROX_SIMPLE)
    #For each contour draw the bounding bos
    for cnt in cnts:
        x,y,w,h = cv2.boundingRect(cnt)
        if y > 200: # Disregard items in the top of the picture
            cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

    writer.write(cv2.resize(frame, (640,480)))

#Release video object
video_stream.release()
writer.release()
```

In [ ]:

2. Background :

The aim of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image. Generally, only a small number of instances of the object are present in the image, but there is a very large number of possible locations and scales at which they can occur and that need to somehow be explored. Each detection of the image is reported with some form of pose information. This is as simple as the location of the object, a location and scale, or the extent of the object defined in terms of a bounding box. In some other situations, the pose information is more detailed and contains the parameters of a linear or non-linear transformation. For example for face detection in a face detector may compute the locations of the eyes, nose and mouth, in addition to the bounding box of the face. An example of a bicycle detection in an image that specifies the locations of certain parts is shown in Figure 1. The pose can also be defined by a three-dimensional transformation specifying the location of the object relative to the camera. Object detection systems always construct a model for an object class from a set of training examples. In the case of a fixed rigid object in an image, only one example may be needed, but more generally multiple training examples are necessary to capture certain aspects of class variability

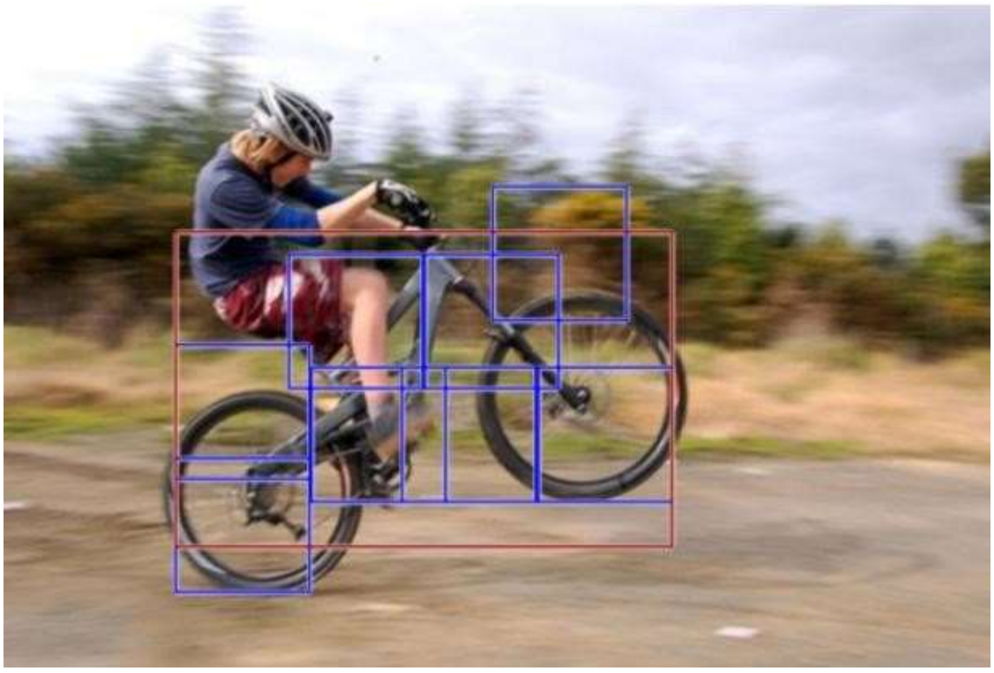


Figure 1

Convolutional implementation of the sliding windows Before we discuss the implementation of the sliding window using convents, let us analyze how we can convert the fully connected layers of the network into convolutional layers. Fig. 2 shows a simple convolutional network with two fully connected layers each of shape.

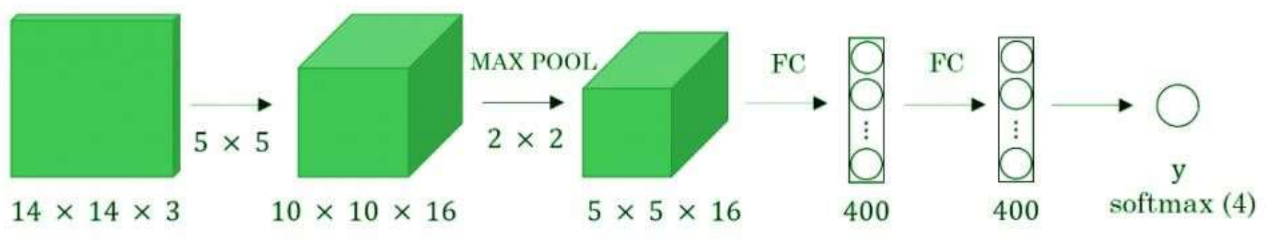


Figure 2: simple convolution network

A fully connected layer can be converted to a convolutional layer with the help of a 1D convolutional layer. The width and height of this layer is equal to one and the number of filters are equal to the shape of the fully connected layer. An example of this is shown in Fig 3.

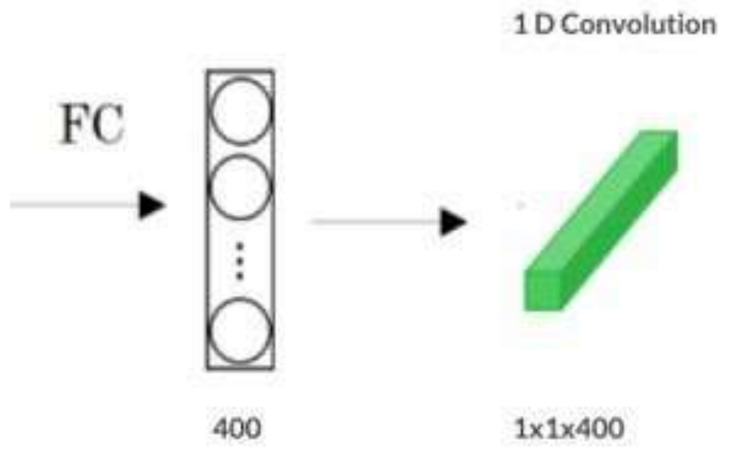


Figure 3

We can apply the concept of conversion of a fully connected layer into a convolutional layer to the model by replacing the fully connected layer with a 1-D convolutional layer. The number of filters of the 1D convolutional layer is equal to the shape of the fully connected layer. This representation is shown in Fig 4. Also, the output softmax layer is also a convolutional layer of shape (1, 1, 4), were 4 is the number of classes to predict.

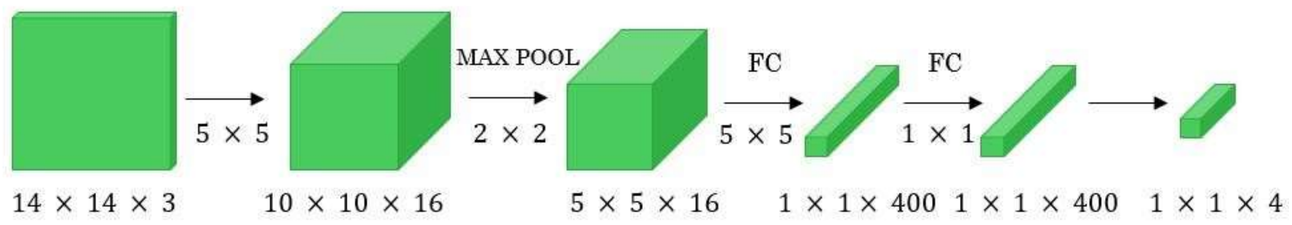


Figure 4

Now, let's extend the above approach to implement a convolutional version of the sliding window. First, let us consider the ConvNet that we have trained to be in the following representation (no fully connected layers).

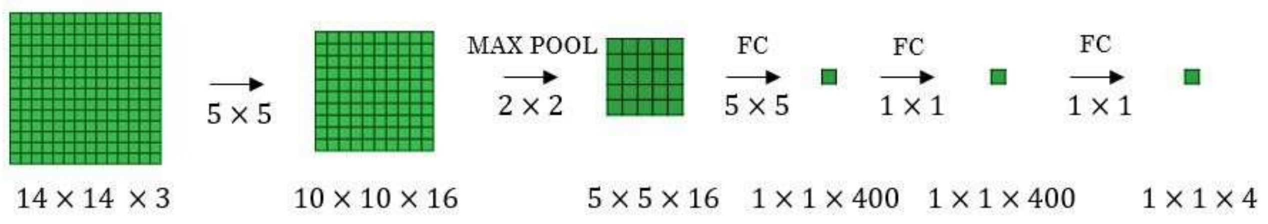


Figure 5

Let's assume the size of the input image to be 16 × 16 × 3. If we are using the sliding window approach, then we would have passed this image to the above ConvNet four times, where each time the sliding window crops the part of the input image matrix of size 14 × 14 × 3 and pass it through the ConvNet. But instead of this, we feed the full image (with shape 16 × 16 × 3) directly into the trained Convent (see Fig. 6). These results will give an output matrix of shape 2 × 2 × 4. Each cell in the output matrix represents the result of the possible crop and the classified value of the cropped image. For example, the left cell of the output matrix (the green one) in Fig. 6 represents the result of the first sliding window. The other cells in the matrix represent the results of the remaining sliding window operations.

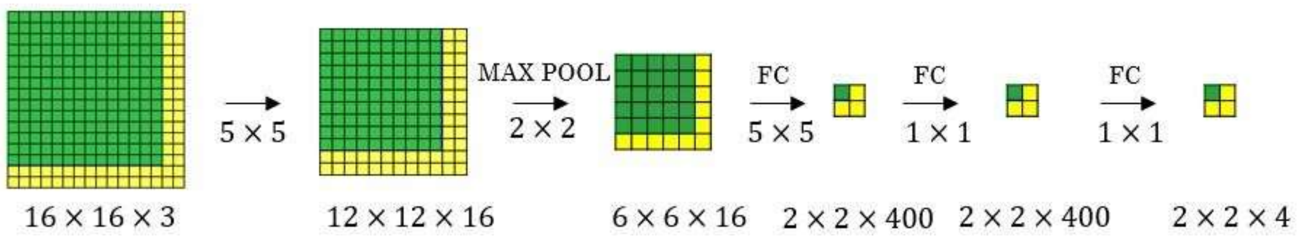


Figure 6

The stride of the sliding window is decided by the number of filters used in the Max Pool layer. In the example above, the Max Pool layer has two filters, and for the result, the sliding window moves with a stride of two resulting in four possible outputs to the given input. The main advantage of using this technique is that the sliding window runs and computes all values simultaneously. Consequently, this technique is really fast. The weakness of this technique is that the position of the bounding boxes is not very accurate.

A better algorithm that tackles the issue of predicting accurate bounding boxes while using the convolutional sliding window technique is the YOLO algorithm. YOLO stands for you only look once which was developed in 2015 by Joseph Redmon, Santosh Divola, Ross Girshick, and Ali Farhadi. It is popular because it achieves high accuracy while running in real-time. This algorithm requires only one forward propagation pass through the network to make the predictions.

This algorithm divides the image into grids and then runs the image classification and localization algorithm (discussed under object localization) on each of the grid cells. For example, we can give an input image of size 256 × 256. We place a 3 × 3 grid on the image (see Fig. 7).

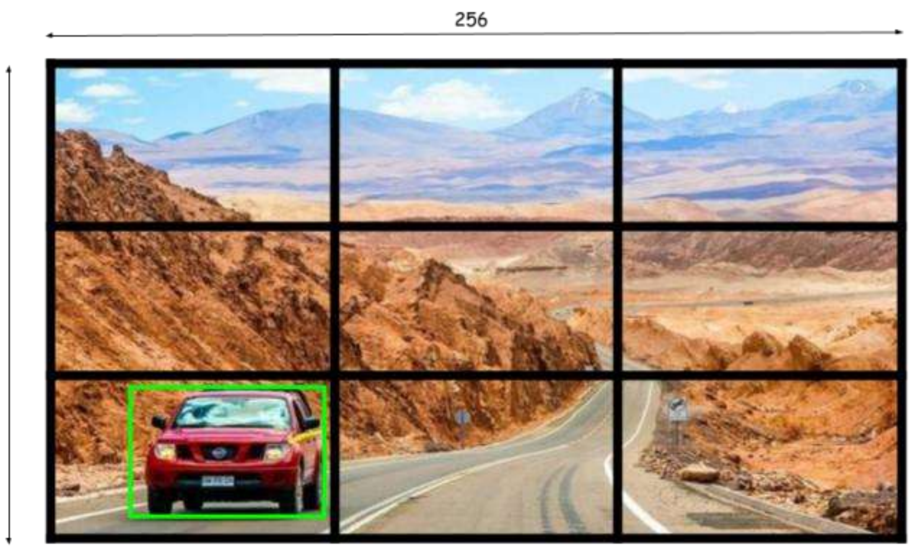


Figure 7

Next, we shall apply the image classification and localization algorithm on each grid cell. In the image each grid cell, the target variable is defined as

Yi,j=[pcbxbybhbwc1c2c3c4]T(6)

Do everything once with the convolution sliding window. Since the shape of the target variable for each grid cell in the image is 1 × 9 and there are 9 (3 × 3) grid cells, the final output of the model will be:

$$Final\ Output = \underbrace{3 \times 3}_{\text{Number of grid cells}} \times \underbrace{9}_{\text{Output label for each grid cell}}$$

The advantages of the YOLO algorithm are that it is very fast and predicts much more accurate bounding boxes. Also, in practice to get the more accurate predictions, we use a much finer grid, say 19 × 19, in which case the target output is of the shape 19 × 19 × 9.

3. LITERATURE SURVEY:

In various fields, there is a necessity to detect the target object and also track them effectively while handling occlusions and other included complexities. Many researchers (Almeida and Gutting 2004, Hsiao-Ping Tsai 2011, Nicolas Papadakis and Auer lie Bugeau 2010) attempted for various approaches in object tracking. The nature of the techniques largely depends on the application domain. Some of the research works which made the evolution to proposed work in the field of object tracking are depicted as follows.

3.1 OBJECT DETECTION

Object detection is an important task, yet challenging vision task. It is a critical part of many applications such as image search, image auto-annotation and scene understanding, object tracking. Moving object tracking of video image sequences was one of the most important subjects in computer vision. It had already been applied in many computers vision fields, such as smart video surveillance (Arun Hampapur 2005), artificial intelligence, military guidance, safety detection and robot navigation, medical and biological application. In recent years, a number of successful single-object tracking system appeared, but in the presence of several objects, object detection becomes difficult and when objects are fully or partially occluded, they are obtruded from the human vision which further increases the problem of detection. Decreasing illumination and acquisition angle. The proposed MLP based object tracking system is made robust by an optimum selection of unique features and also by implementing the Adaboost strong classification method.

3.1.1 Background Subtraction

The background subtraction method by Horprasert et al (1999), was able to cope with local illumination changes, such as shadows and highlights, even globe illumination changes. In this method, the background model was statistically modelled on each pixel. Computational color mode, include the brightness distortion and the chromaticity distortion which was used to distinguish shading background from the ordinary background or moving foreground objects. The background and foreground subtraction method used the following approach. A pixel was modelled by a 4-tuple [Ei, si, ai, bi], where Ei- a vector with expected color value, si - a vector with the standard deviation of color value, ai - the variation of the brightness distortion and bi was the variation of the chromaticity distortion of the ith pixel. In the next step, the difference between the background image and the current image was evaluated. Each pixel was finally classified into four categories: original background, shaded background or shadow, highlighted background and moving foreground object. Liyuan Li et al (2003), contributed a method for detecting foreground objects in non-stationary complex environments containing moving background objects. A Bayes decision rule was used for classification of background and foreground changes based on inter-frame color co-occurrence statistics. An approach to store and fast retrieve color cooccurrence statistics was also established. In this method, foreground objects were detected in two steps. First, both the foreground and the background changes are extracted using background subtraction and temporal differencing. The frequent background changes were then recognized using the Bayes decision rule based on the learned color co-occurrence statistics. Both short-term and long-term strategies to learn the frequent background changes were used. An algorithm focused on obtaining the stationary foreground regions as said by Álvaro Bayona et al (2010), which was useful for applications like the detection of abandoned/stolen objects and parked vehicles. This algorithm mainly used two steps. Firstly, a sub-sampling scheme based on background subtraction techniques was implemented to obtain stationary foreground regions. This detects foreground changes at different time instants in the same pixel locations. This was done by using a Gaussian distribution function. Secondly, some modifications were introduced on this base algorithm such as thresh holding the previously computed subtraction. The main purpose of this algorithm was reducing the amount of stationary foreground detected.

3.1.2 Template Matching

Template Matching is the technique of finding small parts of an image which match a template image. It slides the template from the top left to the bottom right of the image and compares for the best match with the template. The template dimension should be equal to the reference image or smaller than the reference image. It recognizes the segment with the highest correlation as the target. Given an image S and an image T, where the dimension of S was both larger than T, output whether S contains a subset image I where I and T are suitably similar in pattern and if such I exists, output the location of I in S as in Hager and Bellhumear (1998). Schweitzer et al (2011), derived an algorithm which used both upper and lowers bound to detect ‘k’ best matches. Euclidean distance and Walsh transform kernels are used to calculate match measure. The positive things included the usage of priority queue improved quality of decision as to which bound-improved and when good matches exist inherent cost was dominant and it improved performance. But there were constraints like the absence of good matches that lead to queue cost and the arithmetic operation cost was higher. The proposed methods dint use queue thereby avoiding the queue cost rather used template matching. Visual tracking methods can be roughly categorized in two ways namely, the feature-based and region-based method as proposed by Ken Ito and Shigeyuki Sakane (2001). The feature-based approach estimates the 3D pose of a target object to fit the image features the edges, given a 3D geometrical model of an object. This method requires much computational cost. Region-based can be classified into two categories namely, parametric method and view-based method. The parametric method assumes a parametric model of the images in the target image and calculates optimal fitting of the model to pixel data in a region. The view-based method was used to find the best match of a region in a search area given the reference template. This has the advantage that it does not require much computational complexity as in the feature-based approach

4. Existing Methods:

4.1 ResNet

To train the network model in a more effective manner, we herein adopt the same strategy as that used for DSSD (the performance of the residual network is better than that of the VGG network). The goal is to improve accuracy. However, the first implemented for the modification was the replacement of the VGG network which is used in the original SSD with ResNet. We will also add a series of convolution feature layers at the end of the underlying network. These feature layers will gradually be reduced in size that allowed prediction of the detection results on multiple scales. When the input size is given as 300 and 320, although the ResNet–101 layer is deeper than the VGG–16 layer, it is experimentally known that it replaces the SSD's underlying convolution network with a residual network, and it does not improve its accuracy but rather decreases it.

4.2 R-CNN

To circumvent the problem of selecting a huge number of regions, Ross Girshick et al. proposed a method where we use the selective search for extract just 2000 regions from the image and he called them region proposals. Therefore, instead of trying to classify the huge number of regions, you can just work with 2000 regions. These 2000 region proposals are generated by using the selective search algorithm which is written below.

Selective Search:

1. Generate the initial sub-segmentation, we generate many candidates' regions
2. Use the greedy algorithm to recursively combine similar regions into larger ones
3. Use generated regions to produce the final candidate region proposals

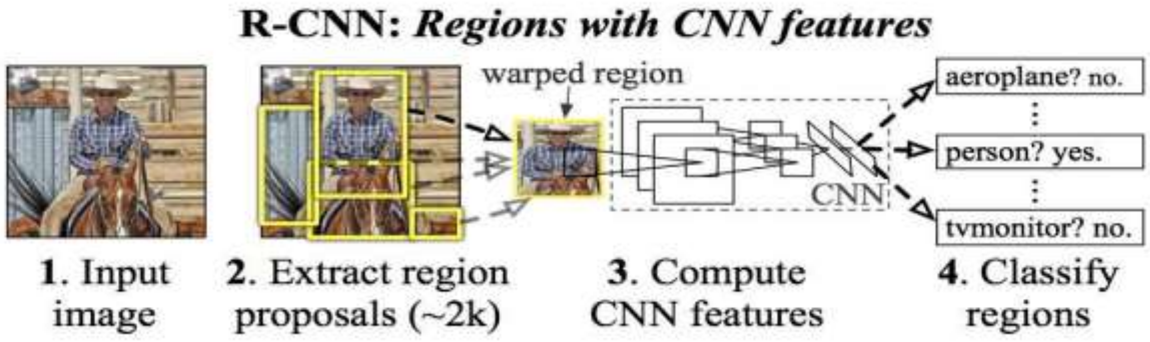


Figure 8

These 2000 candidate regions which are proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. The CNN plays a role of feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM for the classify the presence of the object within that candidate region proposal. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values for increasing the precision of the bounding box. For example, given the region proposal, the algorithm might have predicted the presence of a person but the face of that person within that region proposal could have been cut in half.

Therefore, the offset values which is given help in adjusting the bounding box of the region proposal.

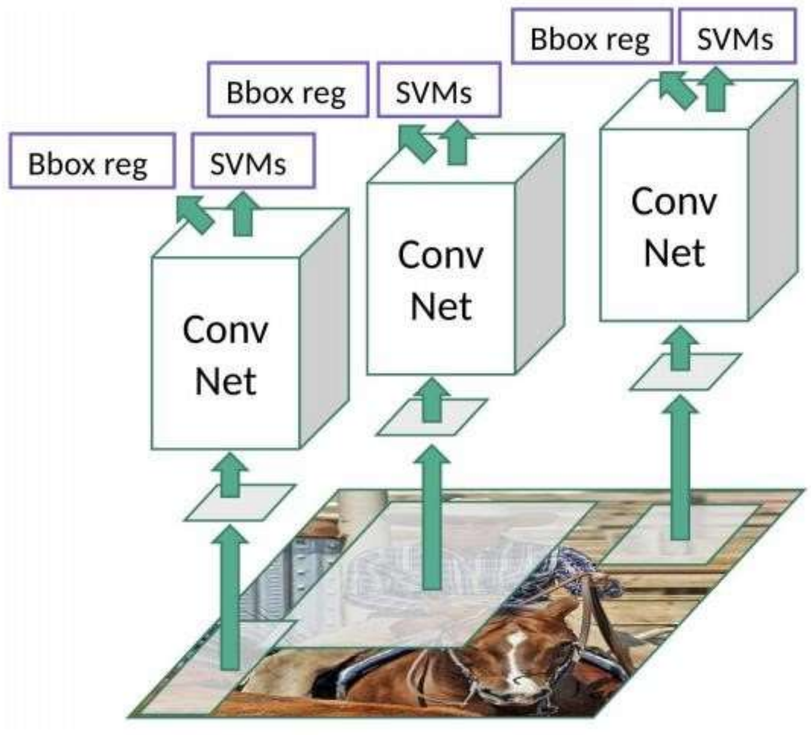


Figure 9: R-CNN

4.2.1 Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.

- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

4.3 Fast R-CNN

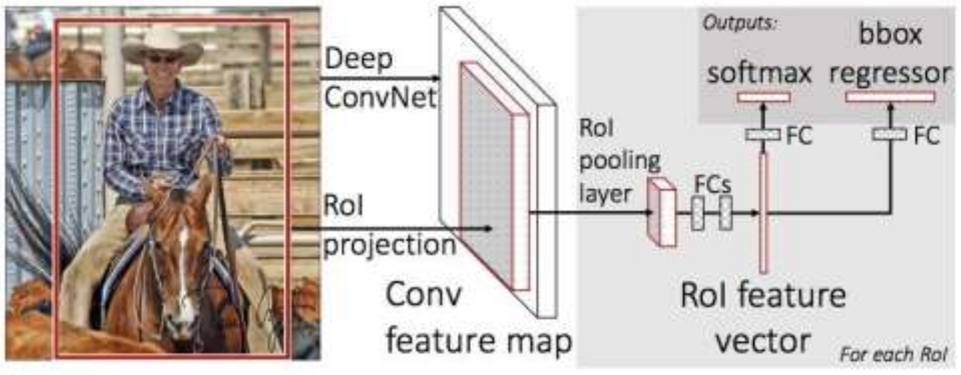


Figure 10: Fast R-CNN

The same author of the previous paper(R-CNN) solved some of the drawbacks of R-CNN to build a faster object detection algorithm and it was called Fast R-CNN. The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map, we can identify the region of the proposals and warp them into the squares and by using an RoI pooling layer we reshape them into the fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, we can use a SoftMax layer to predict the class of the proposed region and also the offset values for the bounding box.

The reason “Fast R-CNN” is faster than R-CNN is because you don’t have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is always done only once per image and a feature map is generated from it.

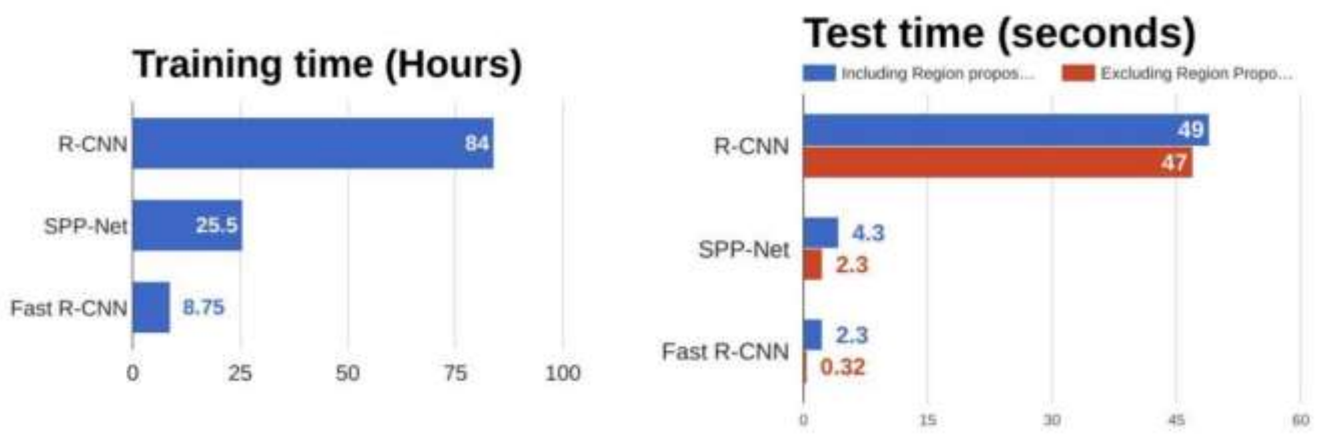


Figure 11: Comparison of object detection algorithms

From the above graphs, you can infer that Fast R-CNN is significantly faster in training and testing sessions over R-CNN. When you look at the performance of Fast R-CNN during testing time, including region proposals slows down the algorithm significantly when compared to not using region proposals. Therefore, the region which is proposals become bottlenecks in Fast R-CNN algorithm affecting its performance.

4.4 Faster R-CNN

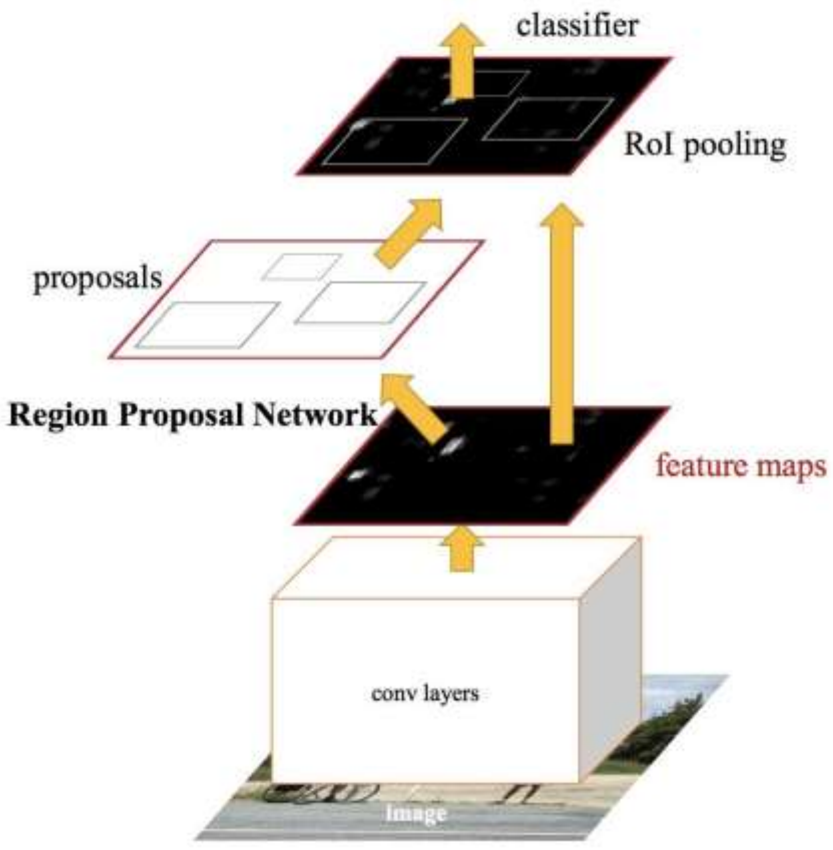


Figure 12: Faster R-CNN

Both of the above algorithms (R-CNN & Fast R-CNN) uses selective search to find out the region proposals. Selective search is the slow and time-consuming process which affect the performance of the network.

Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using the selective search algorithm for the feature map to identify the region proposals, a separate network is used to predict the region proposals. The predicted the region which is proposals are then reshaped using an RoI pooling layer which is used to classify the image within the proposed region and predict the offset values for the bounding boxes.

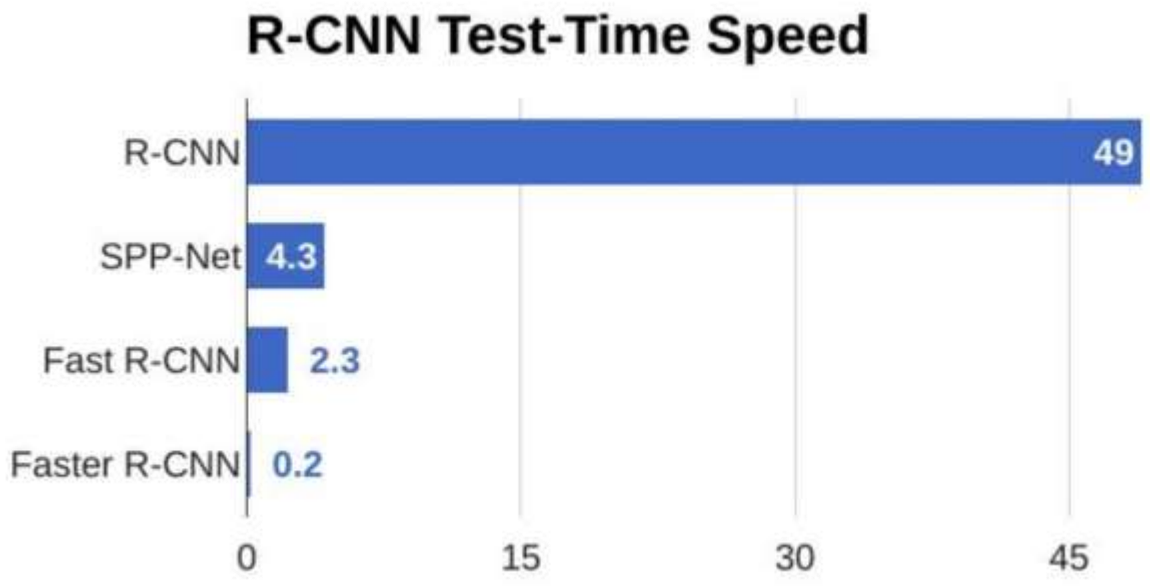


Figure13: Comparison of test-time speed of object detection algorithms

From the above graph, you can see that Faster R-CNN is much faster than its predecessors. Therefore, it can even be used for real-time object detection.

4.5 YOLO — You Only Look Once

All the previous object detection algorithms have used regions to localize the object within the image. The network does not look at the complete image. Instead, parts of the image which has high probabilities of containing the object. YOLO or You Only Look Once is an object detection algorithm much is different from the region-based algorithms which seen above. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.

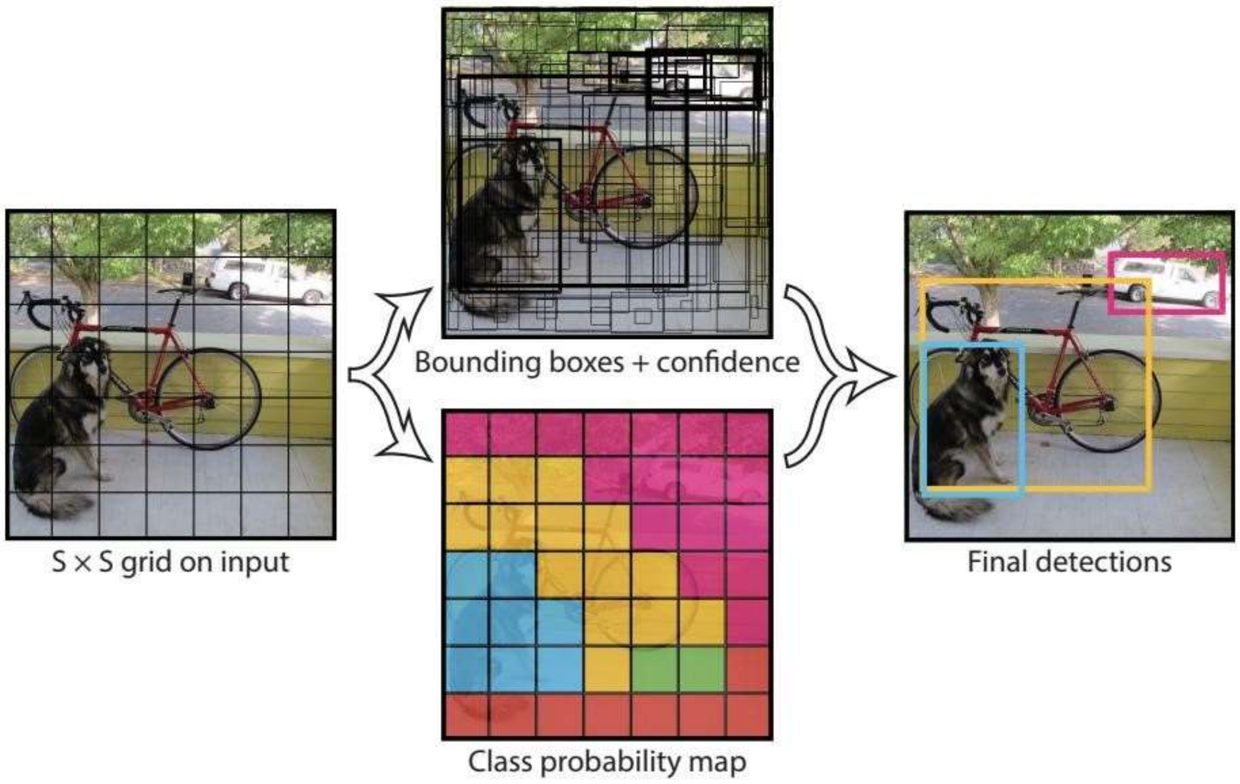


Figure 14

YOLO works by taking an image and split it into an SxS grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network gives an output a class probability and offset values for the bounding box. The bounding boxes have the class probability above a threshold value is selected and used to locate the object within the image.

YOLO is orders of magnitude faster (45 frames per second) than any other object detection algorithms. The limitation of YOLO algorithm is that it struggles with the small objects within the image, for example, it might have difficulties in identifying a flock of birds. This is due to the spatial constraints of the algorithm.

4.6 SDD:

The SSD object detection composes of 2 parts:

1. Extract feature maps, and

2. Apply convolution filters to detect objects.

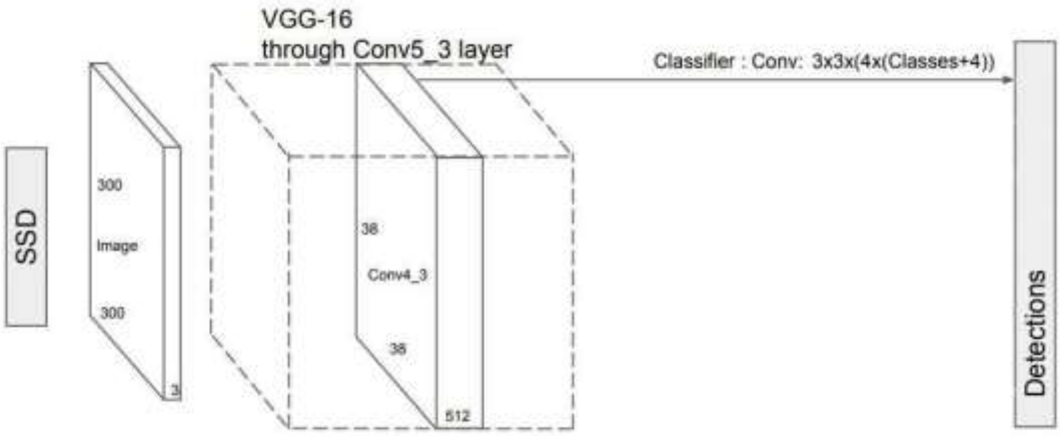


Figure 15

SSD uses VGG16 to extract feature maps. Then it detects objects using the Conv4_3 layer. For illustration, we draw the Conv4_3 to be 8 × 8 spatially (it should be 38 × 38). For each cell in the image (also called location), it makes 4 object predictions.

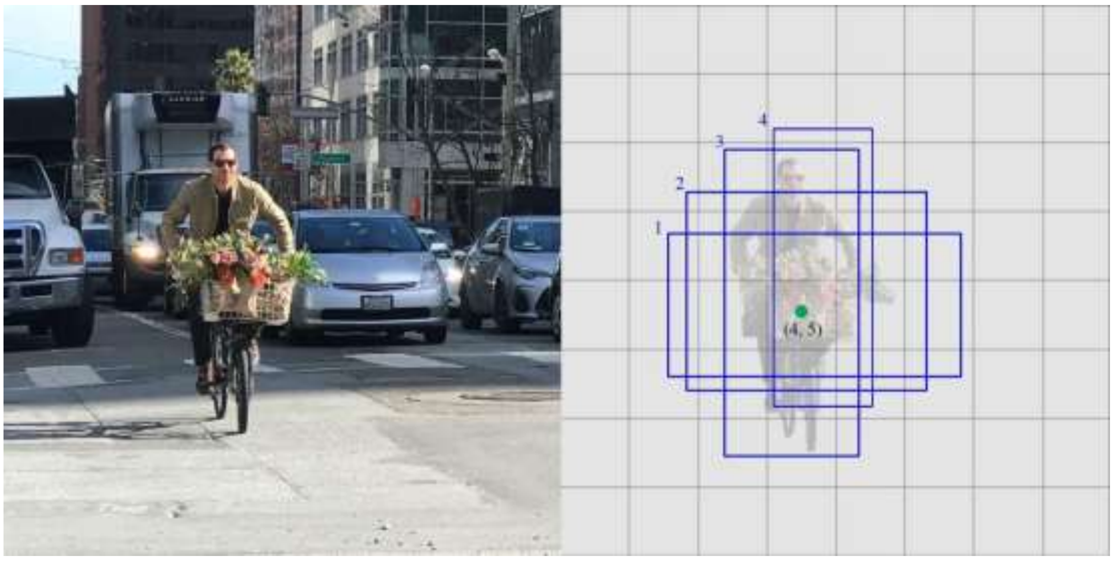


Figure 16

Each prediction composes of a boundary box and 21 scores for each class (one extra class for no object), and we pick the highest score as the class for the bounded object. Conv4_3 makes total of 38 × 38 × 4 predictions: four predictions per cell regardless of the depth of feature maps. An expected, many predictions contain no object. SSD reserves a class "0" to indicate

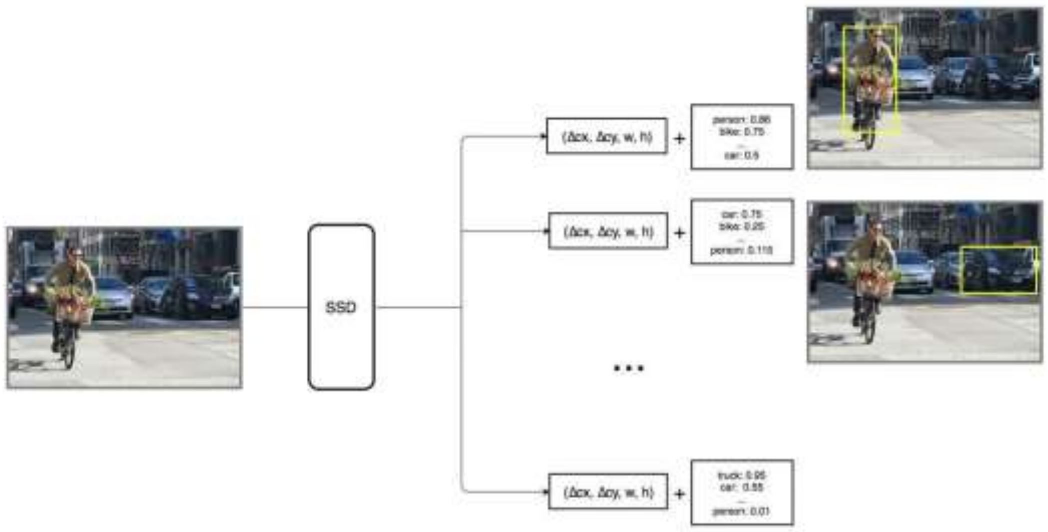


Figure 17

SSD does not use the delegated region proposal network. Instead, it resolves to a very simple method. It computes both the location and class scores using small convolution filters. After extraction the feature maps, SSD applies 3 × 3 convolution filters for each cell to make predictions. (These filters compute the results just like the regular CNN filters.) Each filter gives outputs as 25 channels: 21 scores for each class plus one boundary box.

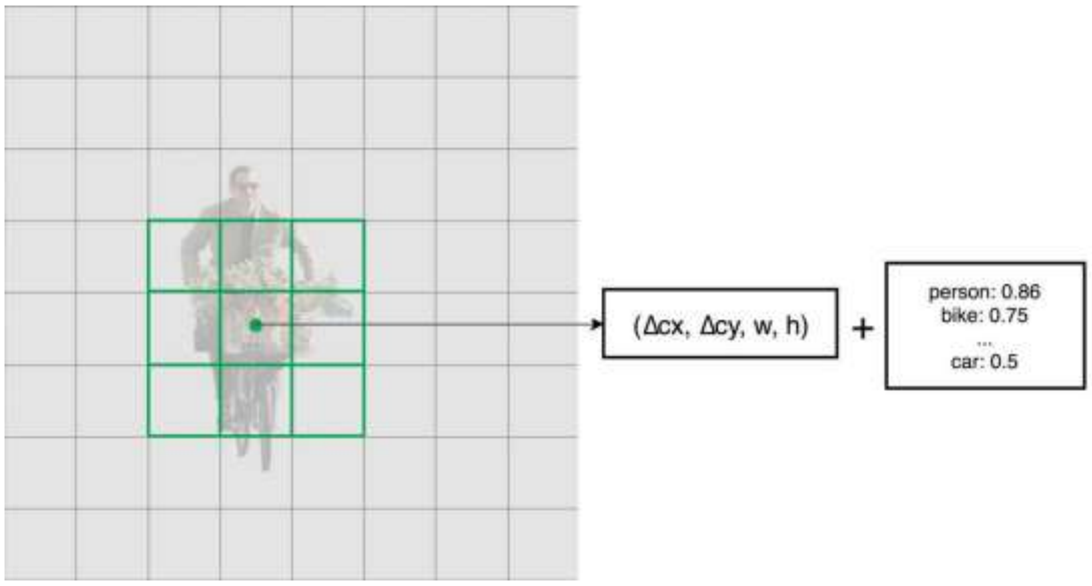


Figure 18

Beginning, we describe the SSD detects objects from a single layer. Actually, it uses multiple layers (multi-scale feature maps) for the detecting objects independently. As CNN reduces the spatial dimension gradually, the resolution of the feature maps also decreases. SSD uses lower resolution layers for the detect larger-scale objects. For example, the 4× 4 feature maps are used for the larger-scale object.

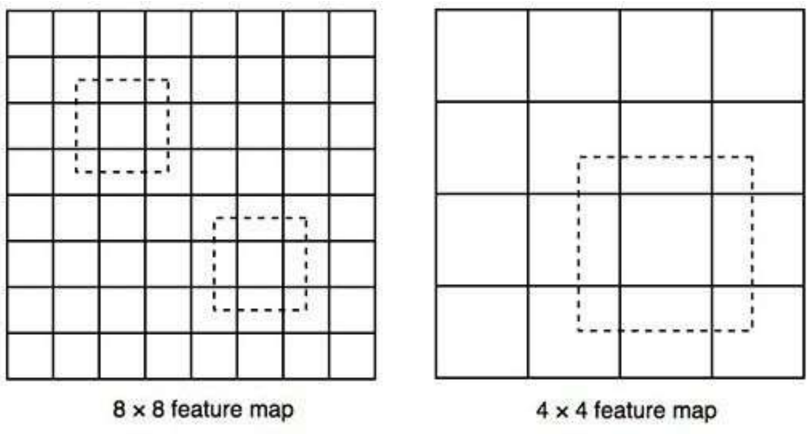


Figure 19

SSD adds 6 more auxiliary convolution layers to image after VGG16. Five of these layers will be added for object detection. In which three of those layers, we make 6 predictions instead of 4. In total, SSD makes 8732 predictions using 6 convolution layers.

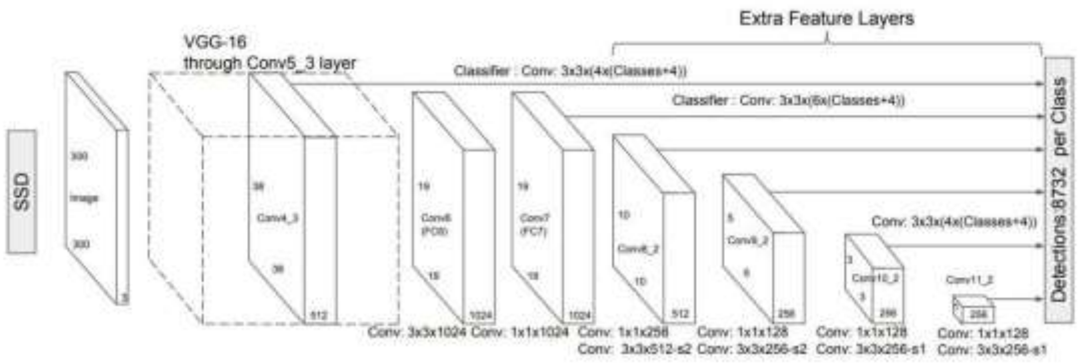


Figure 20

Multi-scale feature maps enhance accuracy. The accuracy with different number of feature map layers is used for object detection.

Prediction source layers from:						mAP use boundary boxes?		# Boxes
38×38	19×19	10×10	5×5	3×3	1×1	Yes	No	
✔	✔	✔	✔	✔	✔	74.3	63.4	8732
✔	✔	✔				70.7	69.2	9864
	✔					62.4	64.0	8664

Table 1

4.7 MANet:

Target detection is fundamental challenging problem for long time and has been a hotspot in the area of computer vision for many years. The purpose and objective of target detection is, to determine if any instances of a specified category of objects exist in an image. If there is an object to be detected in a specific image, target detection returns the spatial positions and the spatial extent of the instances of the objects (based on the use a bounding box, for example). As one of cornerstones of image understanding and computer vision, target and object detection forms the basis for more complex and higher-level visual tasks, such as object tracking, image capture, instance segmentation, and others. Target detection is also widely used in areas such as artificial intelligence and information technology, including machine vision, automatic driving vehicles, and human–computer interaction. In recent times, the method automatic learning of represented features from data based on deep learning has effectively improved performance of target detection. Neural networks are foundation of deep learning. Therefore, design of better neural networks has become a key issue toward improvement of target detection algorithms and performance. Recently developed object detectors that has been based on convolutional neural networks (CNN) has been classified in two types: The first is two-stage detector type, such as Region-Based CNN (R–CNN), Region-Based Full Convolutional Networks (R–FCN), and Feature Pyramid Network (FPN), and the other is single-stage detector, such as the You Only Look Once (YOLO), Single-shot detector (SSD), and the RetinaNet. The former type generates a series of candidate frames as samples of data, and then classifies the samples based on a CNN; the latter type does not generate candidate frames but directly converts the object frame positioning problem into a regression processing problem.

To maintain Realtime speeds without sacrificing precision in various object detectors described above, Liu et al proposed the SSD which is faster than YOLO and has a comparable accuracy to that of the most advanced region-based target detectors. SSD combines regression idea of YOLO with the anchor box mechanism of Faster R–CNN, predicts the object region based on the feature maps of the different convolution layers, and outputs discretized multiscale and multi proportional default box coordinates. The convolution kernel predicts frame coordinates compensation of a series of candidate frames and the confidence of each category. The local feature maps of multiscale area are used to obtain results for each position in the entire image. This maintains the fast characteristics of YOLO algorithm and also ensures that the frame positioning effect is similar to that is induced by the Faster R–CNN. However, SSD directly and independently uses two layers of the backbone VGG16 and four extra layers obtained by a convolution with stride 2 to construct feature pyramid but lacks strong contextual connections.

To solve these problems, A single-stage detection architecture, commonly referred to as MANet, which aggregates feature information at different scales. MANet achieves 82.7% map on the PASCAL VOC 2007 test.

5. SYSTEM REQUIREMENT:

Install Python on your computer system

1. Install Image AI and its dependencies like TensorFlow, NumPy, OpenCV, etc.

2. Download the Object Detection model file (Retinanet)

5.1 Steps to be followed: -

1) Download and install Python version 3 from official Python Language website

https://python.org

2) Install the following dependencies via pip:

I. TensorFlow:

TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning application such as neural networks. It is used for both research and production by Google.

TensorFlow is developed by the Google Brain team for internal Google use. It is released under the Apache License 2.0 on November 9,2015.

TensorFlow is Google Brain's second-generation system.1st Version of TensorFlow was released on February 11, 2017.While the reference implementation runs on single devices, TensorFlow can run on multiple CPU’s and GPU (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on various platforms such as64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

The architecture of TensorFlow allows the easy deployment of computation across a variety of platforms (CPU’s, GPU’s, TPU’s), and from desktops - clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.

pip install TensorFlow -command

ii. NumPy:

NumPy is library of Python programming language, adding support for large, multi-dimensional array and matrices, along with large collection of high-level mathematical function to operate over these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several developers. In 2005 Travis Oliphant created NumPy by incorporating features of computing Numarray into Numeric, with extension modifications. NumPy is open-source software and has many contributors.

pip install NumPy -command

iii. SciPy:

SciPy contain modules for many optimizations, linear algebra, integration, interpolation, special function, FFT, signal and image processing, ODE solvers and other tasks common in engineering.

SciPy abstracts majorly on NumPy array object, and is the part of the NumPy stack which include tools like Matplotlib, pandas and SymPy, etc., and an expanding set of scientific computing libraries. This NumPy stack has similar uses to other applications such as MATLAB, Octave, and Scilab. The NumPy stack is also sometimes referred as the SciPy stack.

The SciPy library is currently distributed under BSDlicense, and its development is sponsored and supported by an open community of developers. It is also supported by NumFOCUS, community foundation for supporting reproducible and accessible science.

pip install scipy -command

iv. OpenCV:

OpenCV is a library of programming functions mainly aimed on real time computer vision. originally developed by Intel, it is later supported by Willow Garage then Itseez. The library is a cross-platform and free to use under the open-source BSD license.

pip install OpenCV-python -command

v. Pillow:

Python Imaging Library is a free Python programming language library that provides support to open, edit and save several different formats of image files. Windows, Mac OS X and Linux are available for this.

pip install pillow -command

vi. Matplotlib:

Matplotlib is a Python programming language plotting library and its NumPy numerical math extension. It provides an object-oriented API to use general-purpose GUI toolkits such as Tkinter, wxPython, Qt, or GTK+ to embed plots into applications.

pip install matplotlib - command

vii. H5py:

The software h5py includes a high-level and low-level interface for Python's HDF5 library. The low interface expected to be complete wrapping of the HDF5 API, while the high-level component uses established Python and NumPy concepts to support access to HDF5 files, datasets and groups.

A strong emphasis on automatic conversion between Python (NumPy) datatypes and data structures and their HDF5 equivalents vastly simplifies the process of reading and writing data from Python.

pip install h5py

viii. Keras

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

pip install keras

ix. Image AI:

Image AI provides API to recognize 1000 different objects in a picture using pre-trained models that were trained on the ImageNet-1000 dataset. The model implementations provided are SqueezeNet, ResNet, InceptionV3 and DenseNet.

pip3 install image Ai --upgrade

3) Download the RetinaNet model file that will be used for object detection using following link

https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/resnet50_coco_best_v2.0.1.h5

Copy the RetinaNet model file and the image you want to detect to the folder that contains the python file.

6. METHODOLOGY:

6.1 SqueezeNet:

SqueezeNet is name of a DNN for computer vision. SqueezNet is developed by researchers at DeepScale, University of California, Berkeley, and Stanford University together. In SqueezeNet design, the authors goal is to create a smaller neural network with few parameters that can more easily fit into memory of computer and can more easily be transmitted over a computer network.

SqueezeNet is originally released in 2016. This original version of SqueezeNet was implemented on top of the Caffe deep learning software framework. The open-source research community ported

SqueezeNet to a number of other deep learning frameworks. And is released in additions, in 2016,

Eddie Bell released a part of SqueezeNet for the Chainer deep learning framework. in 2016, Guo Haria released a part of SqueezeNet for the Apache MXNet framework. 2016, Tammy Yang released a port of SqueezeNet for the Keras framework. In 2017, companies including Baidu, Xilnx, Imagination Technologies, and Synopsys demonstrated Squeezed Net running on low-power processing platforms such as smartphones, FPGAs, and custom processors.

SqueezeNet ships as part of the source code of a number of deep learning frameworks such as PyTorch, Apache MXNet, and Apple CoreML.In addition, 3rd party developers have created implementation of SqueezeNet that are compatible with frameworks such as TensorFlow. Below is summary of frameworks that support SqueezeNet.

DNN Model	**Application**	**Original Implementation**	**Other Implementations**
SqueezeDet	Object Detection on Images	TensorFlow	Caffe, Keras
SqueezeSeg	Semantic Segmentation of LIDAR	TensorFlow	
SqueezeNext	Image Classification	Caffe	TensorFlow, Keras, PyTorch
SqueezeNAS	Neural Architecture Search for Semantic Segmentation	PyTorch	

Table 2

6.2 InceptionV3:

Inception v3 is widely used as image recognition model that has showed to obtain accuracy of greater than 78.1% on the ImageNet dataset. The model is the culmination of many ideas developed by researchers over years. It is based on “Rethinking the Inception Architecture Computer Vision” by Szegedy

The model is made of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers. Batch norm is used more throughout the model and applied to activation inputs. Loss is computed via SoftMax.

A high-level diagram of the model is shown below:

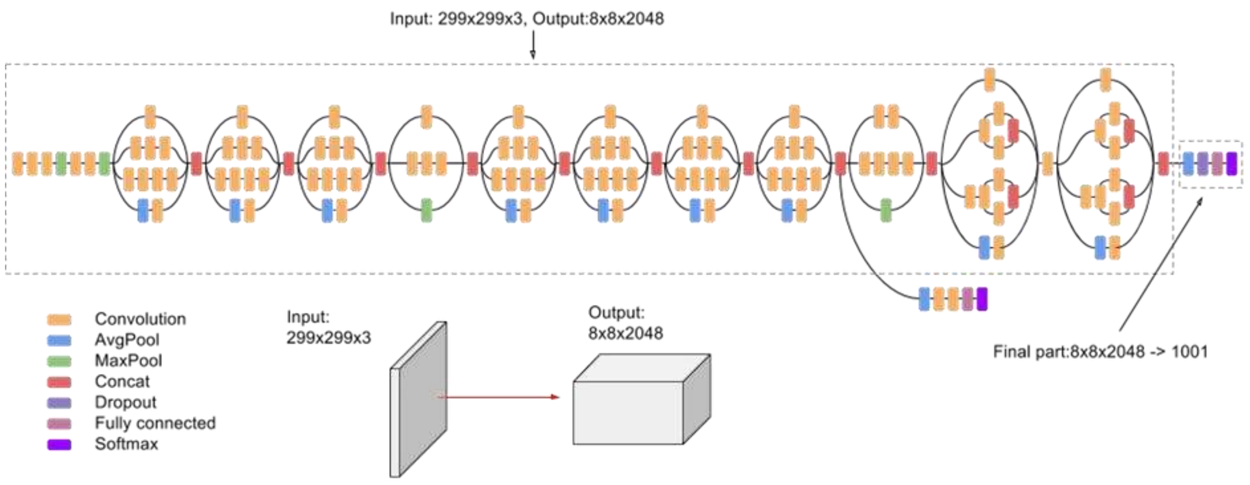
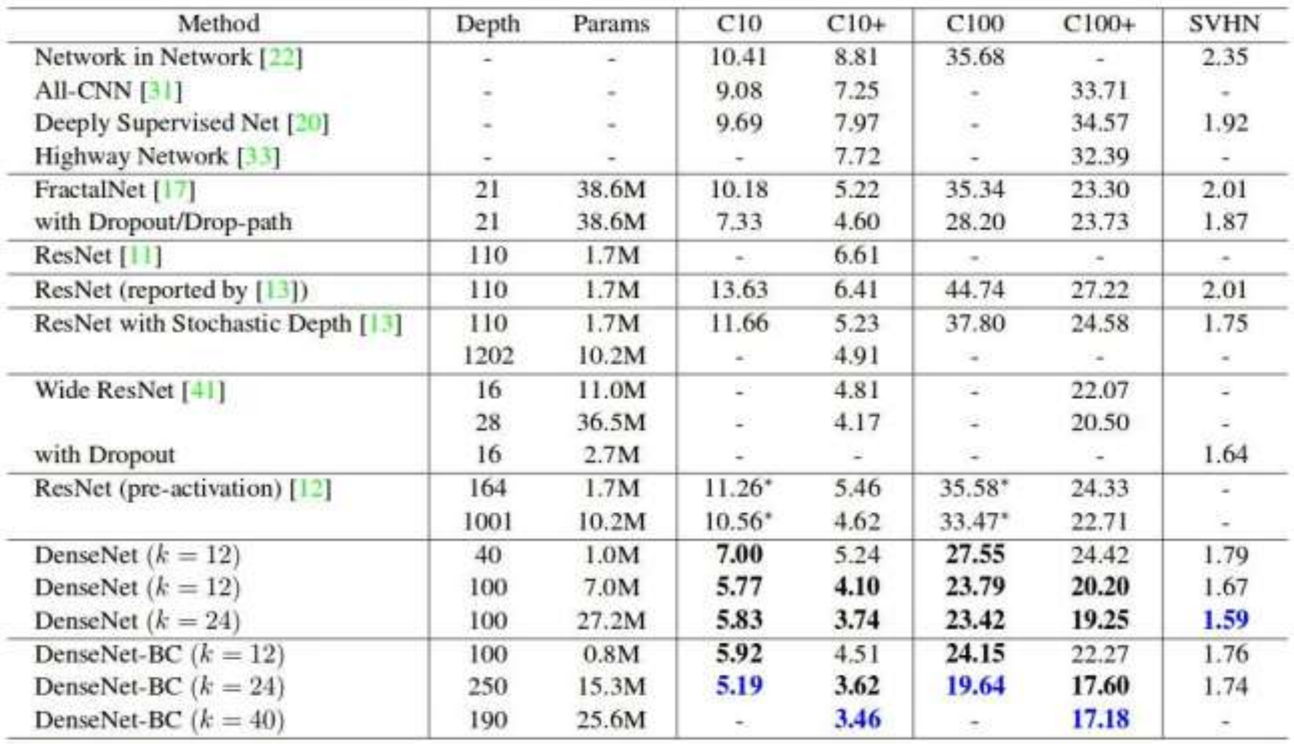


Figure 21

6.3 DenseNet:

DenseNet stands for Densely Connected Convolutional Networks it is one of the latest neural networks for visual object recognition. It is similar to ResNet but has some fundamental differences.

With all improvements DenseNets have one of the lowest error rates on CIFAR/SVHN datasets:



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [31]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [33]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [41]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

Table 3

Error rates on various datasets

And for ImageNet dataset DenseNets require fewer parameters than ResNet with same accuracy:

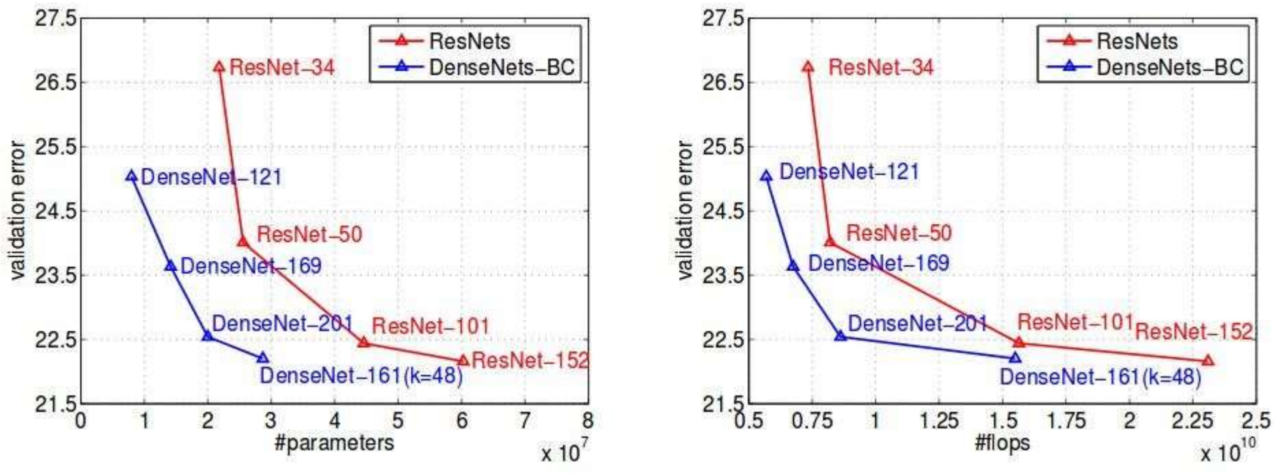


Figure 22

Comparison of the DenseNet and ResNet Top-1 error rates on the ImageNet classification dataset as a function of learned parameters and flops during test-time

This post assumes past knowledge of neural networks and convolutions. This mainly focus on two topics:

- Why does dense net differ from another convolution networks?
- What are the difficulties during the implementation of DenseNet in TensorFlow?

If you know how DenseNets works and interested only in TensorFlow implementation feel free to jump to the second chapter or check the source. If you are not familiar with any of these topics to attain knowledge

Compare DenseNet with other convolution networks available

Usually, Convolution networks work such a way that

We have an initial image, say having a shape of (29, 34, 31). After we apply set of convolution or pooling filters on it, squeezing dimensions of width and height and increasing features dimension.

So, the output from the Li layer is input to the Li+1 layer.

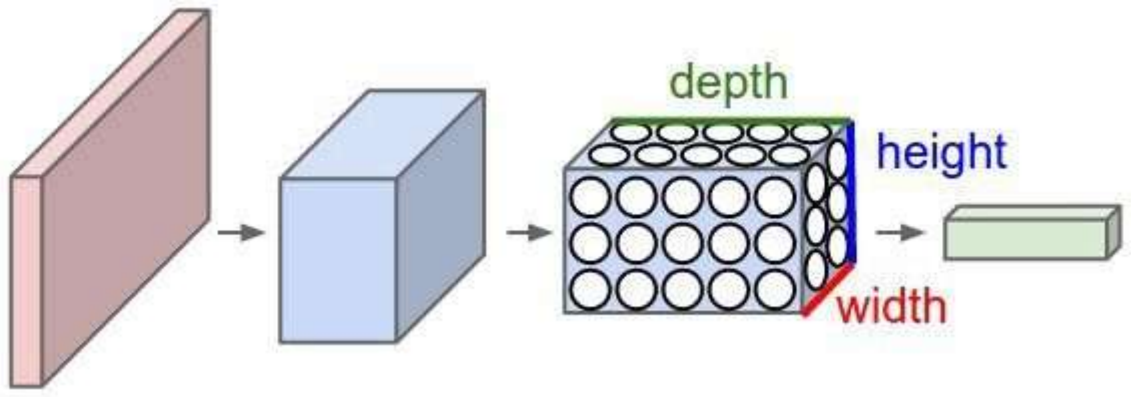


Figure 23

ResNet architecture is proposed for Residual connection, from previous layers to the present layer. input to Li layer is obtain by addition of outputs from previous layers.

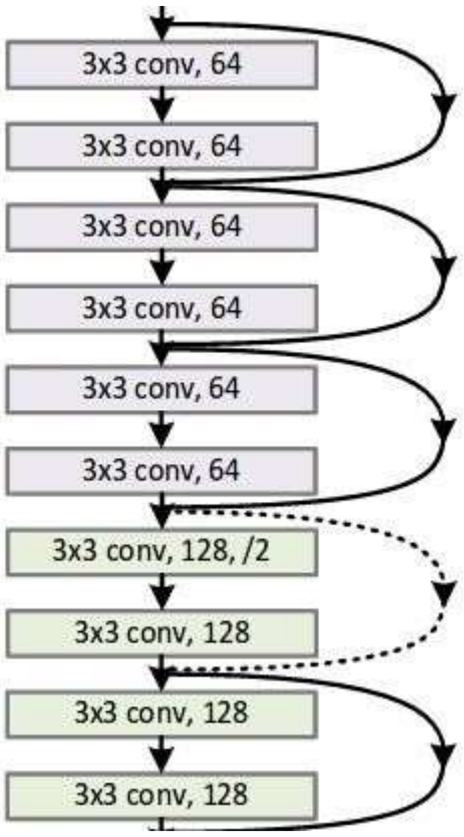


Figure 24

In contrast, DenseNet paper proposes concatenating outputs from the previous layers instead of using the summation.

So, let's imagine we have an image with shape (28, 28, 3). First, we spread image to initial 24 channels and receive the image (28, 28, 24). Every next convolution layer will generate k=12 features, and remain width and height the same.

The output from L_i layer will be (28, 28, 12).

But input to the L_{i+1} will be (28, 28, 24+12), for L_{i+2} (28, 28, 24 + 12 + 12) and so on.

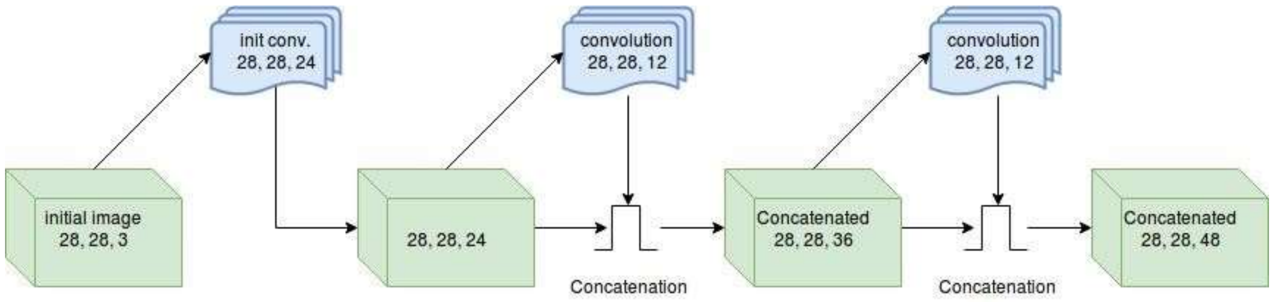


Figure 25: Block of convolution layers with results concatenated

After a while, we receive the image with same width and height, but with plenty of features (28, 28, 48).

All these N layers are named Block in the paper. There's also batch normalization, nonlinearity and dropout inside the block.

To reduce the size, DenseNet uses transition layers. These layers contain convolution with kernel size = 1 followed by 2x2 average pooling with stride = 2. It reduces height and width dimensions but leaves feature dimension the same. As a result, we receive the image with shapes (14, 14, 48).

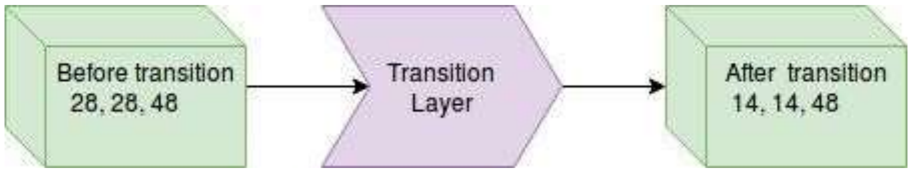


Figure 26: Transition layer

Now we can again pass the image through the block with N convolutions.

With this approach, DenseNet improved a flow of information and gradients throughout the network, which makes them easy to train.

Each layer has direct access to the gradients from the loss function and the original input signal, leading to an implicit deep supervision.

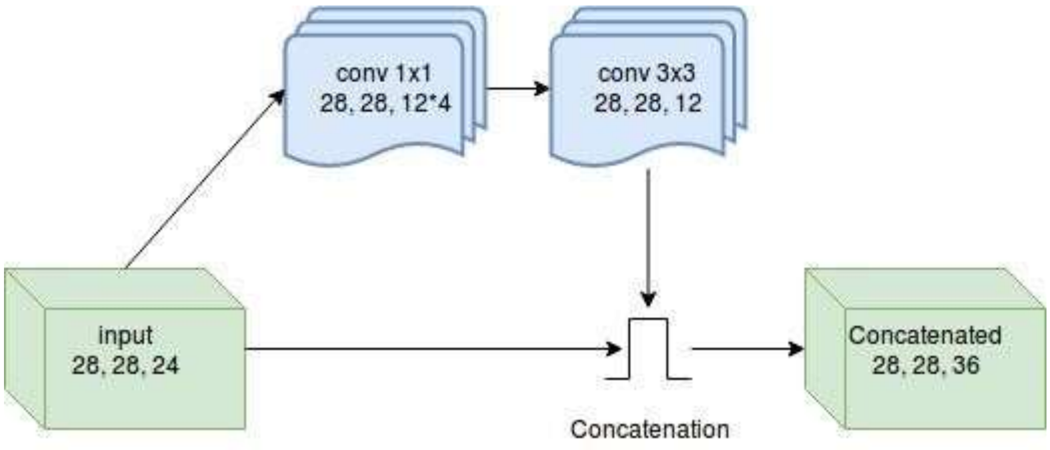


Figure 27

Also at transition layers, not only width and height will be reduced but features also. So, if we have image shape after one block (28, 28, 48) after transition layer, we will get (14, 14, 24).

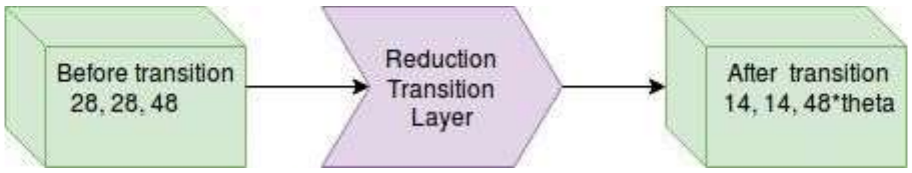


Figure 28

Where theta — some reduction values, in the range (0, 1).

When using bottleneck layers with DenseNet, maximum depth will be divided by 2. It means that if y ouhave 16 3x3 convolution layers with depth 20 previously (some layers are transition layers), you will now have 8 1x1 convolution layers and 8 3x3 convolutions. Last, but not least, about data preprocessing. In the paper per channel normalization was used. With this approach, every image channel should be reduced by its mean and divided by its standard deviation. In many implementations was another normalization used — just divide every image pixel by 255, so we have pixels values in the range [0, 1].

Note about NumPy implementation of per channel normalization. By default, images provided with data type unit. Before any manipulations, it is advised to convert the images to any float representation.

Because otherwise, a code will fail without any warnings or errors.

7. RESULTS AND DISCUSSION:

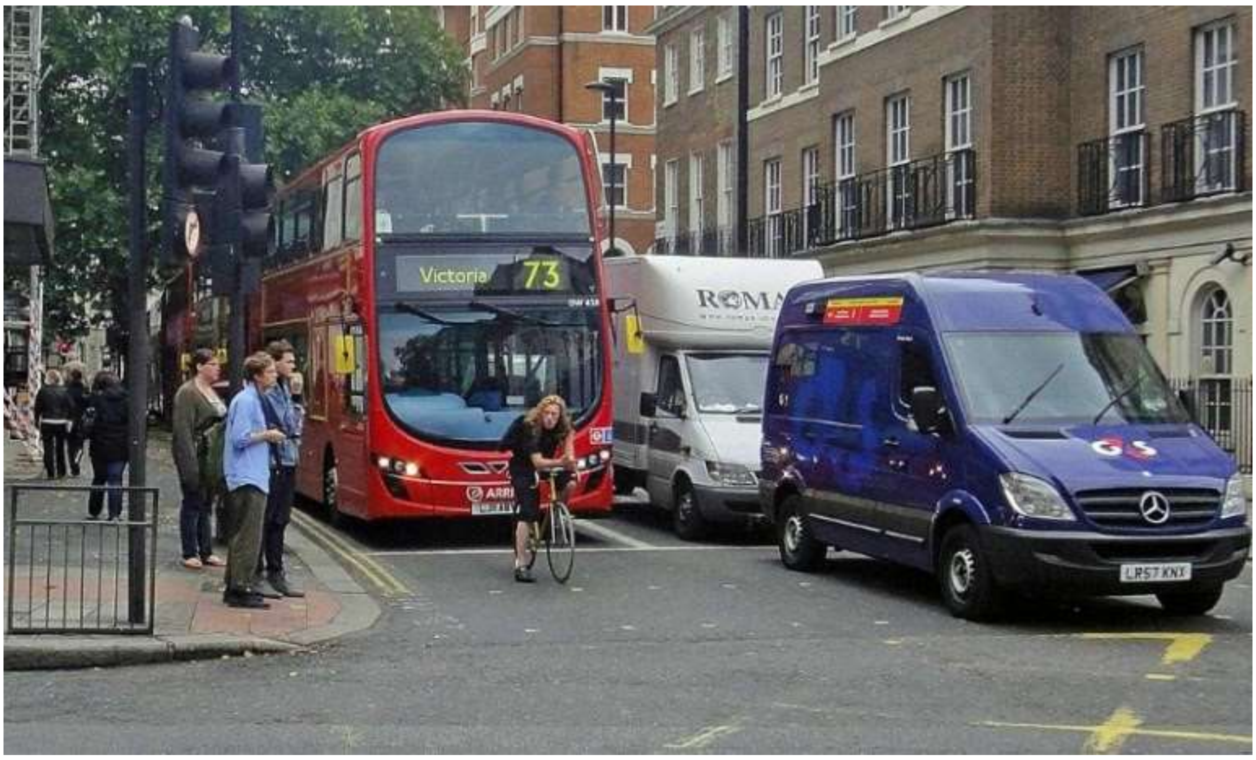


Figure 29: Before Detection:

This is a sample image we feed to the algorithm and expect our algorithm to detect and identify objects in the image and label them according to the class assigned to it.

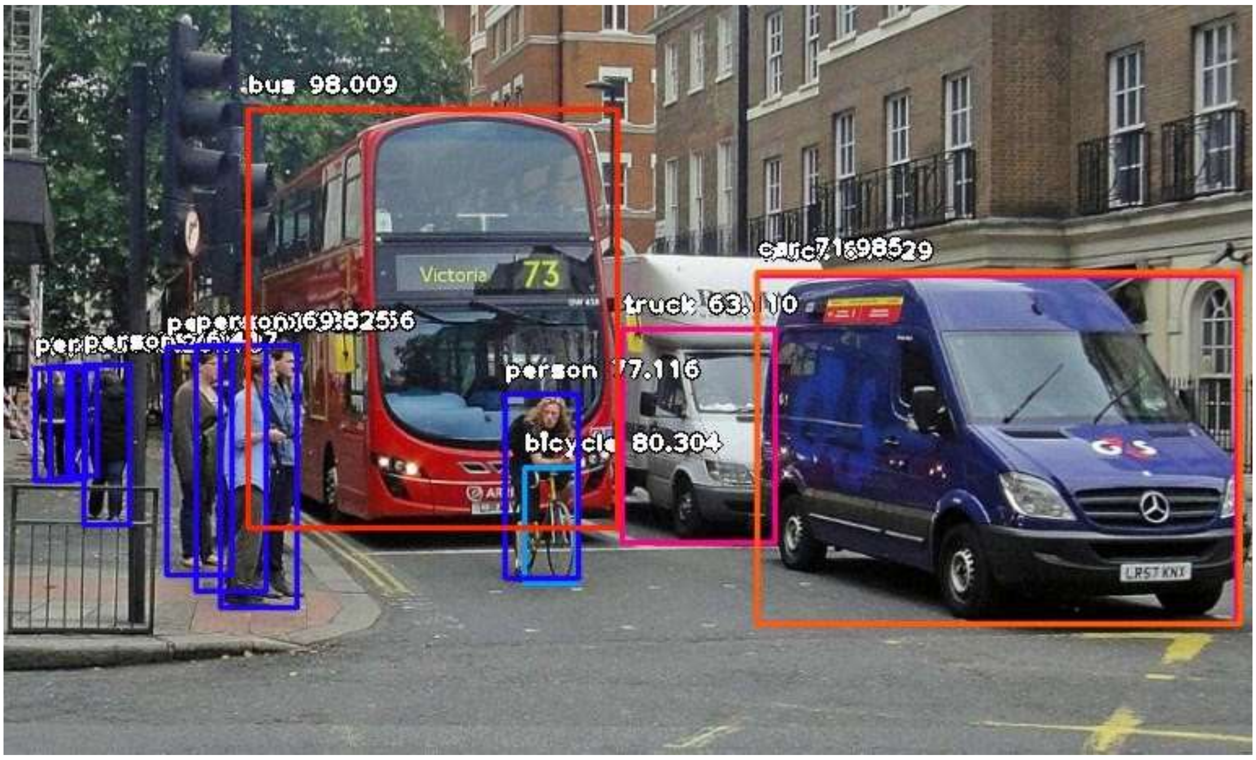
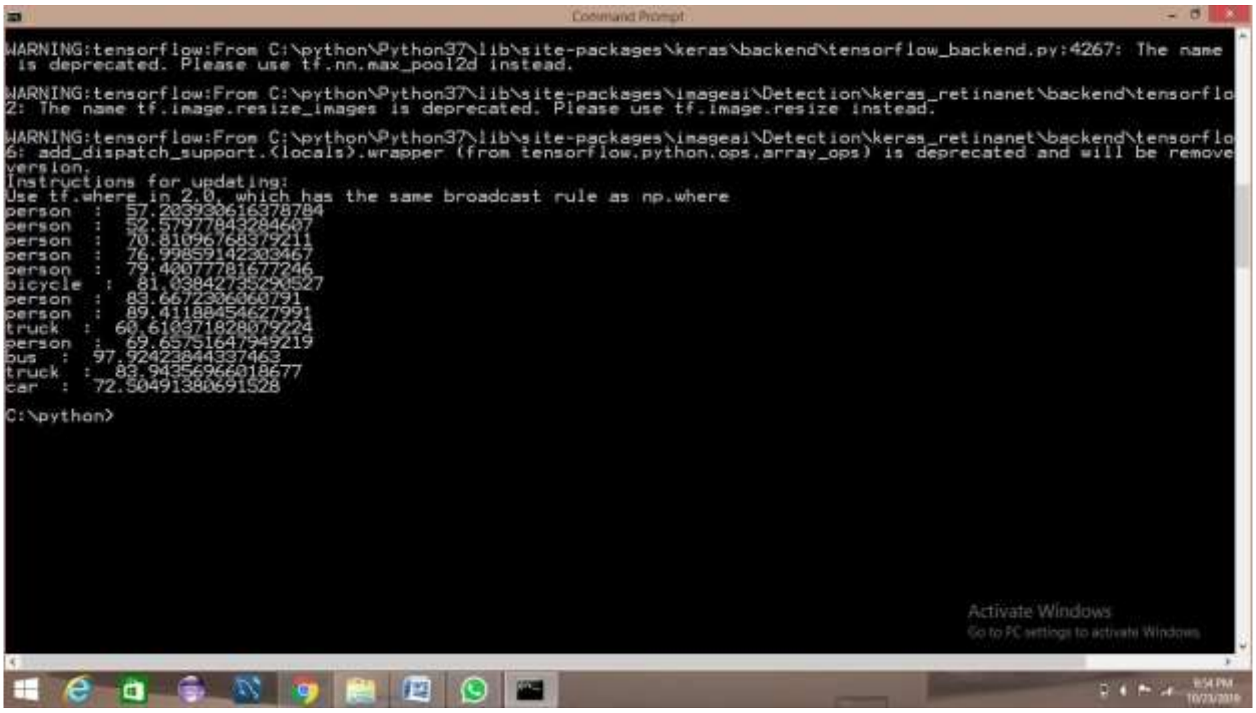


Figure 30: After Detection

As expected, our algorithm identifies the objects by its classes Ans assigns each object by its tag and has dimensions on detected image.



```
WARNING:tensorflow:From C:\python\Python37\lib\site-packages\keras\backend\tensorflow_backend.py:4267: The name
 is deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From C:\python\Python37\lib\site-packages\imageai\Detection\keras_retinanet\backend\tensorflo
2: The name tf.image.resize_images is deprecated. Please use tf.image.resize instead.

WARNING:tensorflow:From C:\python\Python37\lib\site-packages\imageai\Detection\keras_retinanet\backend\tensorflo
6: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be remove
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
person  :  57.203930616378784
person  :  52.57977843284607
person  :  70.81096768379211
person  :  76.99859142303467
person  :  79.40077781677246
bicycle  :  81.03842735290527
person  :  83.66723060600791
person  :  89.41188454627991
truck  :  60.610371828079224
person  :  69.65751647949219
bus  :  97.92423844337463
truck  :  83.94356966018677
car  :  72.50491380691528

C:\python>
```

Figure 31: ***Console result for above image:***

Image AI provides many more features useful for customization and production capable deployments for object detection tasks. Some of the features supported are:

- Adjusting Minimum Probability: By default, objects detected with a probability percentage of less than 50 will not be shown or reported. You can increase this value for high certainty cases or reduce the value for cases where all possible objects are needed to be detected.

- Custom Objects Detection: Using a provided Custom Object class, you can tell the detection class to report detections on one or a few numbers of unique objects.

- Detection Speeds: You can reduce the time it takes to detect an image by setting the speed of detection speed to "fast", "faster" and "fastest".

- Input Types: You can specify and parse in file path to an image, NumPy array or file stream of an image as the input image

- Output Types: You can specify that the detectObjectsFromImage function should return the image in the form of a file or NumPy array

7.1 Detection Speed: -

Image AI now provides detection speeds for all object detection tasks. The detection speeds allow you to reduce the time of detection at a rate between 20% - 80%, and yet having just slight changes but accurate detection results. Coupled with lowering the minimum-percentage-probability parameter, detections can match the normal speed and yet reduce detection time drastically. The available detection speeds are **"normal"**(default), **"fast"**, **"faster", "fastest"** and **"flash"**. All you need to do is to state the speed mode you desire when loading the modeling the code.

```
detector. load Model (detection speed="fast")
```

Hiding/Showing Object Name and Probability: -

Image AI provides options to hide the name of objects detected and / or the percentage probability from being shown on the saved/returned detected image. Using the detectObjectsFromImage () and detectCustomObjectsFromImage () functions, the parameters display-object-name and display-percentage-probability can be set to True of False individually.

```
detections=detector. detectObjectsFromImage (input_image=os. path. Join(execution_path,"image3.jpg")
,                output_image_path=os.                path.join(execution_path,"image3new_nodetails.jpg"),
minimum_percentage_probability=30,
display_percentage_probability=False, display_object_name=False)
```

8. CONCLUSION:

By using this thesis and based on experimental results we are able to detect object more precisely and identify the objects individually with exact location of an object in the picture in x, y axis. This paper also provides experimental results on different methods for object detection and identification and compares each method for their efficiencies.

9. FUTURE ENCHANCEMENTS

The object recognition system can be applied in the area of surveillance system, face recognition, fault detection, character recognition etc. The objective of this thesis is to develop an object recognition system to recognize the 2D and 3D objects in the image. The performance of the object recognition system depends on the features used and the classifier employed for recognition. This research work attempts to propose a novel feature extraction method for extracting global features and obtaining local features from the region of interest. Also, the research work attempts to hybrid the traditional classifiers to recognize the object. The object recognition system developed in this research was tested with the benchmark datasets like COIL100, Caltech 101, ETH80 and MNIST. The object recognition system is implemented in MATLAB 7.5

It is important to mention the difficulties observed during the experimentation of the object recognition system due to several features present in the image. The research work suggests that the image is to be preprocessed and reduced to a size of 128 x 128. The proposed feature extraction method helps to select the important feature. To improve the efficiency of the classifier, the number of features should be less in number. Specifically, the contributions towards this research work are as follows,

- An object recognition system is developed, that recognizes the two-dimensional and three-dimensional objects.
- The feature extracted is sufficient for recognizing the object and marking the location of the object. x the proposed classifier is able to recognize the object in less computational cost.
- The proposed global feature extraction requires less time, compared to the traditional feature extraction method.
- The performance of the SVM-kNN is greater and promising when compared with the BPN and SVM.

- The performance of the One-against-One classifier is efficient.

- Global feature extracted from the local parts of the image.
- Local feature PCA-SIFT is computed from the blobs detected by the Hessian-Laplace detector.
- Along with the local features, the width and height of the object computed through projection method is used.

The methods presented for feature extraction and recognition are common and can be applied to any application that is relevant to object recognition.

The proposed object recognition method combines the state-of-art classifier SVM and k-NN to recognize the objects in the image. The multiclass SVM is used to hybridize with the k-NN for the recognition. The feature extraction method proposed in this research work is efficient and provides unique information for the classifier.
The image is segmented into 16 parts, from each part the Hu's Moment invariant is computed and it is converted into Eigen component. The local feature of the image is obtained by using the Hessian-Laplace detector. This helps to obtain the objects feature easily and mark the object location without much difficulty.

As a scope for future enhancement,

- Features either the local or global used for recognition can be increased, to increase the efficiency of the object recognition system.
- Geometric properties of the image can be included in the feature vector for recognition. 150 Using
- unsupervised classifier instead of a supervised classifier for recognition of the object.
- The proposed object recognition system uses grey-scale image and discards the colour information. The colour information in the image can be used for recognition of the object. Colour based object recognition plays vital role in Robotics

Although the visual tracking algorithm proposed here is robust in many of the conditions, it can be
- made more robust by eliminating some of the limitations as listed below:

- In the Single Visual tracking, the size of the template remains fixed for tracking. If the size of the object reduces with the time, the background becomes more dominant than the object being tracked. In this case the object may not be tracked.
- Fully occluded object cannot be tracked and considered as a new object in the next frame.
- Foreground object extraction depends on the binary segmentation which is carried out by applying threshold techniques. So, blob extraction and tracking depend on the threshold value.
- Splitting and merging cannot be handled very well in all conditions using the single camera due to the loss of information of a 3D object projection in 2D images.

- For Night time visual tracking, night vision mode should be available as an inbuilt feature in the CCTV camera.

To make the system fully automatic and also to overcome the above limitations, in future, Multiview tracking can be implemented using multiple cameras. Multi view tracking has the obvious advantage over single view tracking because of wide coverage range with different viewing angles for the objects to be tracked.

In this thesis, an effort has been made to develop an algorithm to provide the base for future applications such as listed below.

- In this research work, the object Identification and Visual Tracking has been done through the use of ordinary camera. The concept is well extendable in applications like Intelligent Robots, Automatic Guided Vehicles, Enhancement of Security Systems to detect the suspicious behaviour along with detection of weapons, identify the suspicious movements of enemies on boarders with the help of night vision cameras and many such applications.

- In the proposed method, background subtraction technique has been used that is simple and fast. This technique is applicable where there is no movement of camera. For robotic application or automated vehicle assistance system, due to the movement of camera, backgrounds are continuously changing leading to implementation of some different segmentation techniques like single Gaussian mixture or multiple Gaussian mixture models.

- Object identification task with motion estimation needs to be fast enough to be implemented for the real time system. Still there is a scope for developing faster algorithms for object identification. Such algorithms can be implemented using FPGA or CPLD for fast execution

10.References

1. Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse, part-based representation. IEEE Trans. Pattern Anal. Mach. Intel. 26,1475–1490. doi:10.1109/TPAMI.2004.108
2. Alexe, B., Deselaers, T., and Ferrari, V. (2010). “What is an object?” in Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on (San Francisco, CA: IEEE), 73–80. doi:10.1109/CVPR.2010.5540226
3. Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. Int. Compute. Vis. 1, 333–356. doi:10.1007/BF00133571
4. Andreopoulos, A., and Tsotsos, J. K. (2013). 50 years of object recognition: directions forward. Compute. Vis. Image Underst. 117, 827–891. doi:10.1016/j.cviu.2013.04.005
5. Azizpour, H., and Laptev, I. (2012). “Object detection using strongly-superviseddeformable part models,” in Computer Vision-ECCV 2012 (Florence: Springer),836–849.
6. Azzopardi, G., and Petkov, N. (2013). Trainable cosfire filters for keypoint detectionand pattern recognition. IEEE Trans. Pattern Anal. Mach. Intell. 35, 490–503.doi:10.1109/TPAMI.2012.106
7. Azzopardi, G., and Petkov, N. (2014). Ventral-stream-like shape representation: from pixel intensity values to trainable object-selective cosfire models. Front.Comput. Neurosci. 8:80. doi:10.3389/fncom.2014.00080
8. Benbouzid, D., Busa-Fekete, R., and Kegl, B. (2012). “Fast classification using sparsedecision dags,” in Proceedings of the 29th International Conference on MachineLearning (ICML-12), ICML ‘12, eds J. Langford and J. Pineau (New York, NY:Omnipress), 951–958.
9. Bengio, Y. (2012). “Deep learning of representations for unsupervised and transfer learning,” in ICML Unsupervised and Transfer Learning, Volume 27 of JMLRProceedings, eds I. Guyon, G. Dror, V. Lemaire, G. W. Taylor, and D. L. Silver (Bellevue: JMLR.Org), 17–36.
10. Bourdev, L. D., Maji, S., Brox, T., and Malik, J. (2010). “Detecting peopleusing mutually consistent poselet activations,” in Computer Vision – ECCV2010 – 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part VI, Volume 6316 of

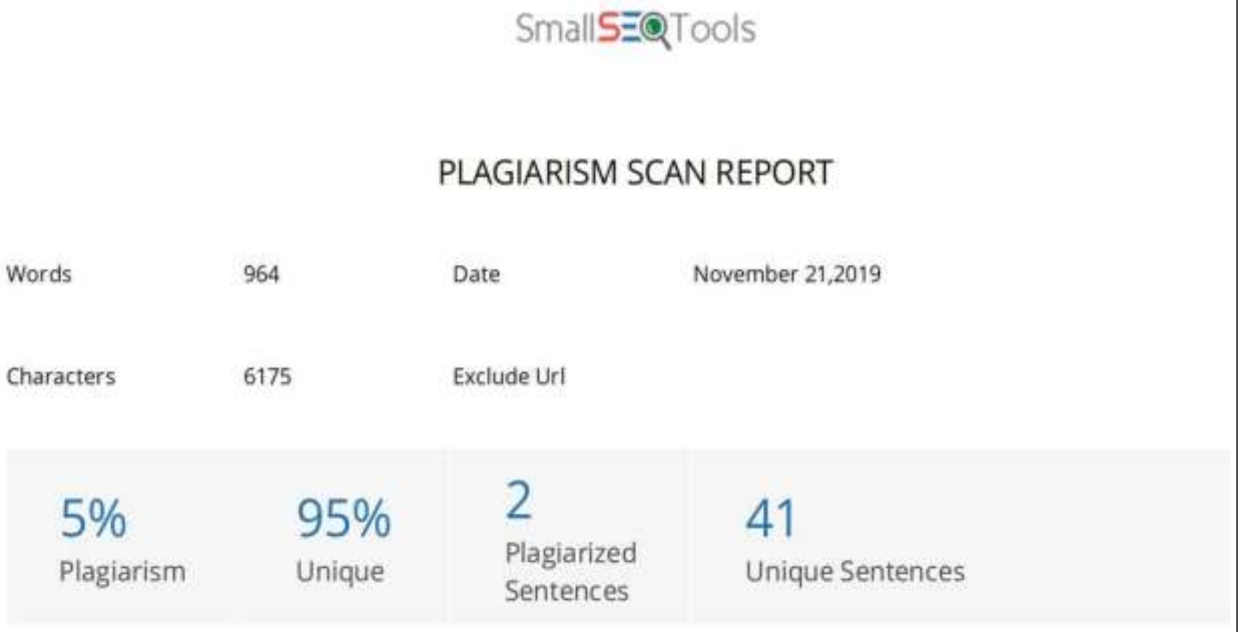
Lecture Notes in Computer Science, eds K. Daniilidis, P. Maragos, and N. Paragios

(Heraklion: Springer), 168–181.

11. Bourdev, L. D., and Malik, J. (2009). “Pose lets: body part detectors trained using 3dhuman pose annotations,” in IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 – October 4, 2009 (Kyoto: IEEE),1365–1372.
12. Cadena, C., Dick, A., and Reid, I. (2015). “A fast, modular scene understanding sys-tem using context-aware object detection,” in Robotics and Automation (ICRA),2015 IEEE International Conference on (Seattle, WA).
13. Correa, M., Hermosilla, G., Verschae, R., and Ruiz-del-Solar, J. (2012). Humandetection and identification by robots using thermal and visual information indomestic environments. J. Intell. Robot Syst. 66, 223–243. doi:10.1007/s10846-011-9612-2
14. Dalal, N., and Triggs, B. (2005). “Histograms of oriented gradients for humandetection,” in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEEComputer Society Conference on,

    Vol. 1 (San Diego, CA: IEEE), 886–893. doi:10.1109/CVPR.2005.177
15. Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). “Scalable object detec-tion using deep neural networks,” in Computer Vision and Pattern Recognition Frontiers in Robotics and AI www.frontiersin.org November 2015

11. PLAGIARISM ANALYSIS:

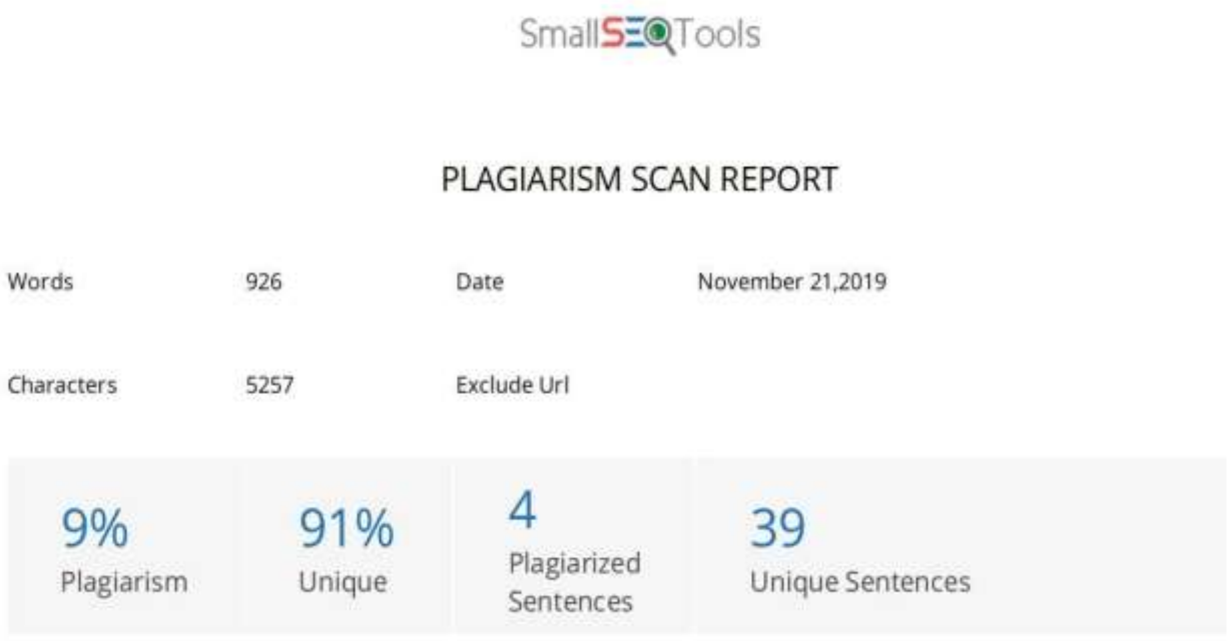


SmallSEOTools

PLAGIARISM SCAN REPORT

Words	964	Date	November 21,2019
Characters	6175	Exclude Url	

5% Plagiarism	95% Unique	2 Plagiarized Sentences	41 Unique Sentences

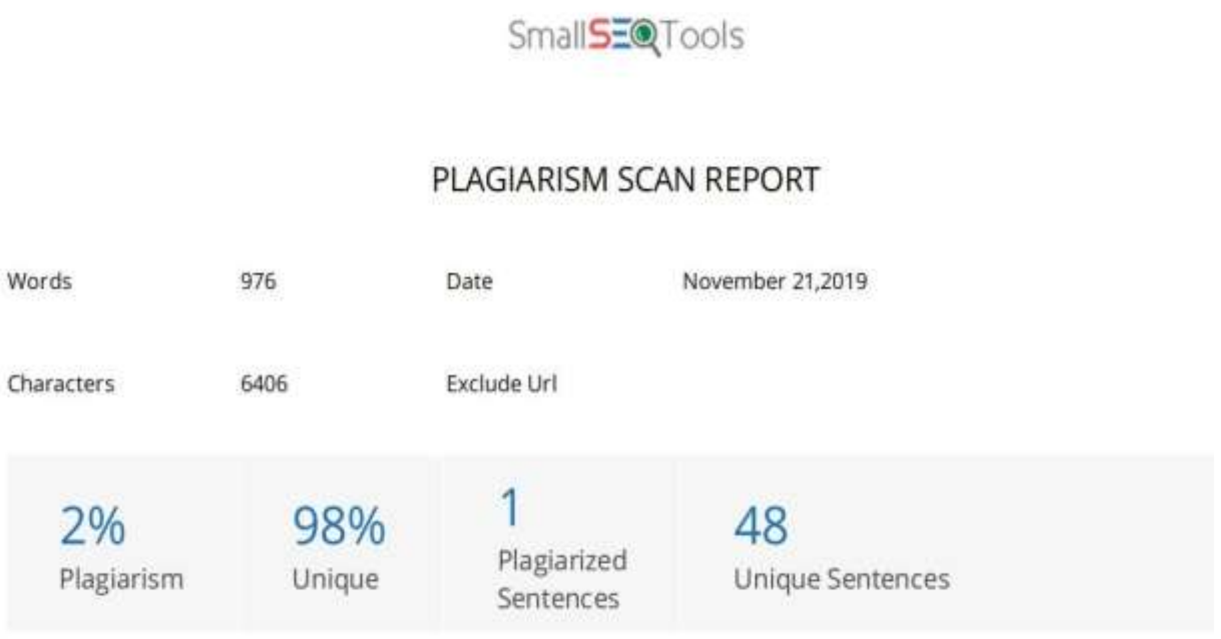


SmallSEOTools

PLAGIARISM SCAN REPORT

Words	926	Date	November 21,2019
Characters	5257	Exclude Url	

9% Plagiarism	91% Unique	4 Plagiarized Sentences	39 Unique Sentences

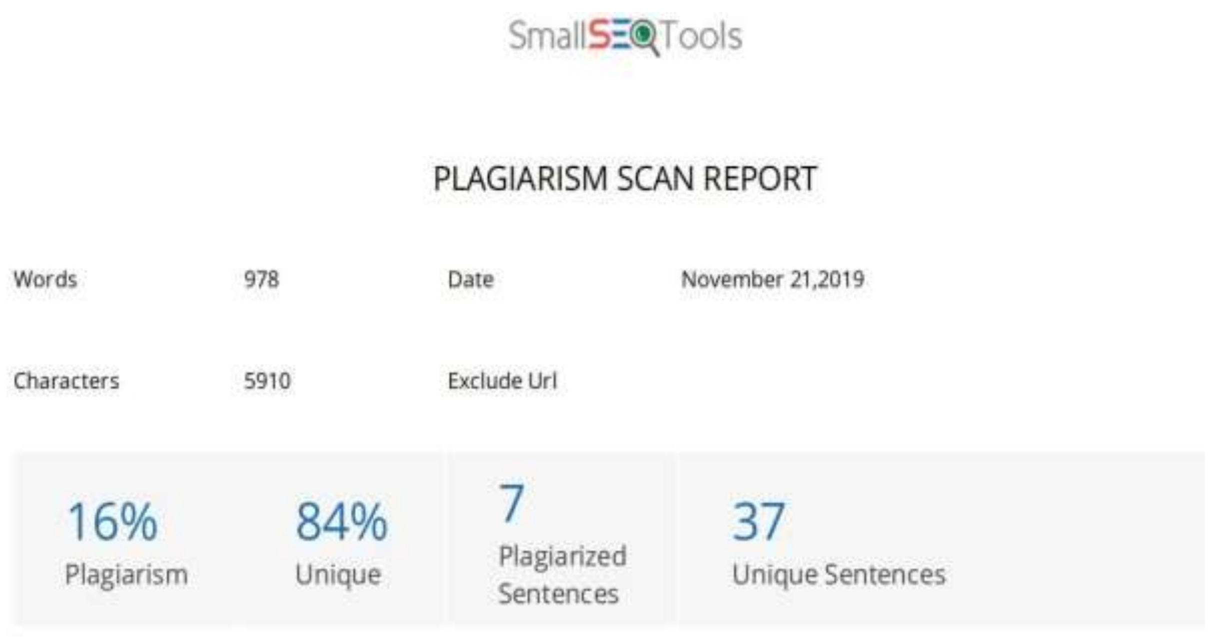


SmallSEOTools

PLAGIARISM SCAN REPORT

Words	976	Date	November 21,2019
Characters	6406	Exclude Url	

2% Plagiarism	98% Unique	1 Plagiarized Sentences	48 Unique Sentences

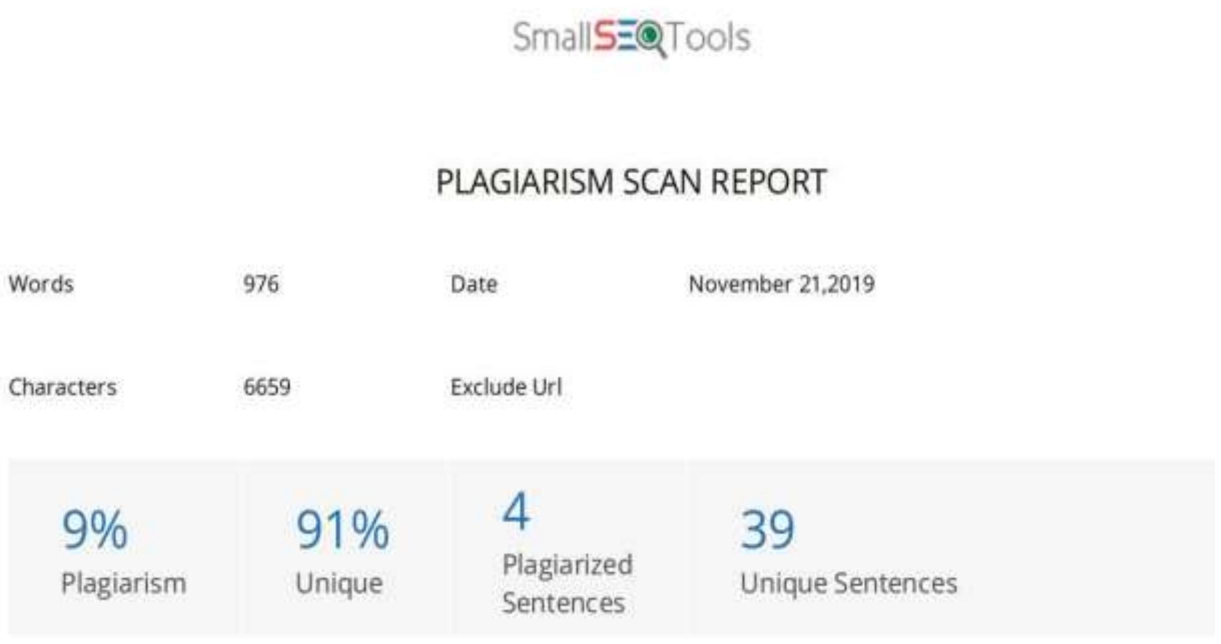


SmallSEOTools

PLAGIARISM SCAN REPORT

Words	978	Date	November 21,2019
Characters	5910	Exclude Url	

16% Plagiarism	84% Unique	7 Plagiarized Sentences	37 Unique Sentences

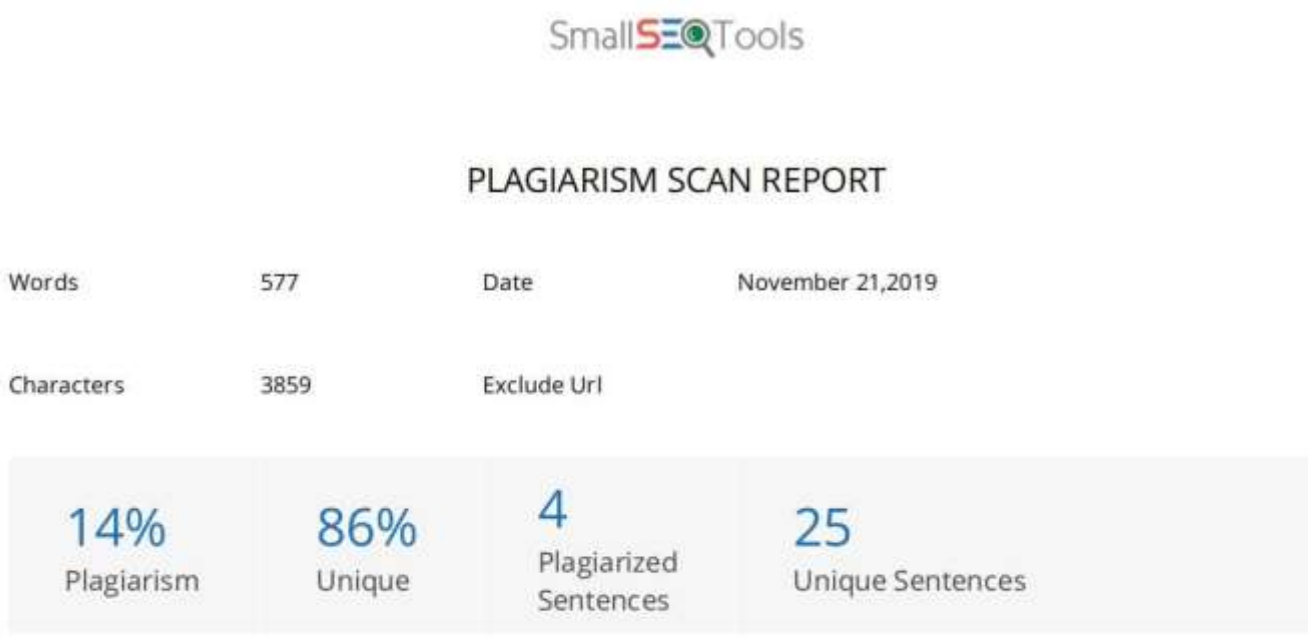


SmallSEOTools

PLAGIARISM SCAN REPORT

Words	976	Date	November 21,2019
Characters	6659	Exclude Url	

9% Plagiarism	91% Unique	4 Plagiarized Sentences	39 Unique Sentences



SmallSEOTools

PLAGIARISM SCAN REPORT

Words	577	Date	November 21,2019
Characters	3859	Exclude Url	

14% Plagiarism	86% Unique	4 Plagiarized Sentences	25 Unique Sentences