Computer Vision Task Report
Prakriti Kharel

To develop a pipeline to count the number of passing cars in the video provided, I modified the code found on the following GitHub repo: https://github.com/theAIGuysCode/yolov4-deepsort

The code in the GitHub repo used YOLOv4 for object detection and the Deep SORT algorithm for object tracking. In this paper, first, I will explain these two algorithms as I currently understand them. Secondly, I will explain my journey with this project and lastly, I will explain how I modified the code from the Github repo above to accomplish the task assigned.

YOLOv4 - Object Detector

Before we start talking about YOLOv4 which is an object detector, let's briefly understand what Object Detection means and how it differs from classification and object localization.

Classification refers to assigning a label to an object, so it only tells us what class an object belongs to. With classification we can only identify one object per image (I guess you could identify more, but then the training set would be huge since it would have to account for all possible combinations of objects). Classification also does not tell us where the object is located in the image.

Object localization on the other hand involves drawing a bounding box around objects in an image, so it tells us where the object is located but not necessarily what the object is.

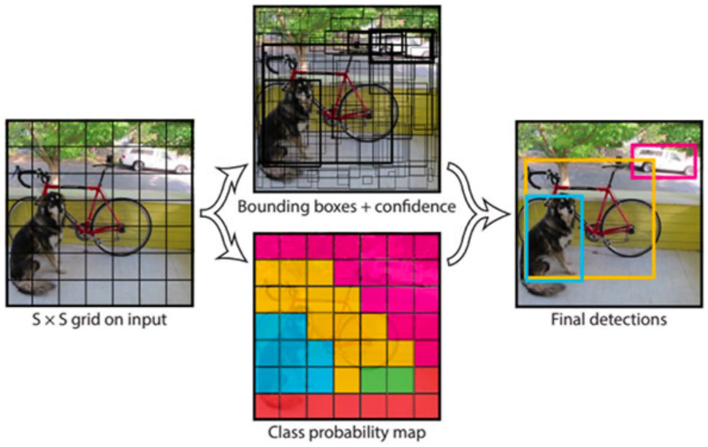
Object detection is the combination of those two: it tells what the objects are and where in the image/frame they are located. This is achieved by first using object localization to draw a bonding box or a mask around the object and then we use classification to each of those objects.

The framework that is considered to be the state of art framework for Object Detection for its ability to detect objects in real time with a pretty good accuracy is known as YOLO (You only look once).

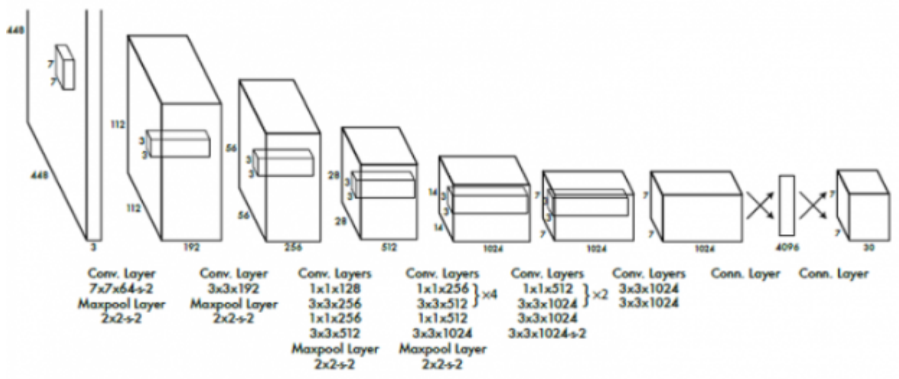
This first version of this framework was first proposed by Jospeh Redmon in 2016.

YOLO uses darknet (open source neural network framework written by Redmon himself) as the underlying framework for training. A single neural network is used to first split the input image into grid class (different regions), the network then predicts which cells are responsible for predicting the bounding around the objects. The network also predicts the class probabilities of each cell (the probability of each cell belonging to different classes) creating the class probabilities map.

The bounding box predictions along with the class probabilities map is used to predict the final bounding box and the class labels for the objects as seen in the image below.



The network only requires one forward propogation to predict the labels of the objects and the bounding box. The network only looks at the image once hence the name, You Only Look Once. This is the CNN architecture used.



It is used today over other methods such as Faster RCNN because it is extremely fast, it is able to detect multiple objects in real-time and it generalizes better than other methods.

The GitHub repo above uses version four of YOLO which was released in April 2020 which has an improvement in the speed and the accuracy (by using pretrained weights) than version one.

Deep SORT - Object Tracker

Due to the time constraint, I did not do as much research into Deep SORT as I did into YOLOv4, but here is what I understand.

As explained in the section above, the process of object detection consists of putting a bonding box around an object and classifying the object. This process only detects the objects in the given frame, once that object is no longer in the frame, there is no way to remember that object existed in a previous frame. For tasks such as counting cars, where you need memory of the objects that have exited in previous frames, we need some form of an object tracker the tracks the objects across the entire video. We need to use some sort of spatio-temporal features.

The most popular object tracking framework used today is Deep SORT. The SORT stands for Simple Real Time Tracker and Deep obviously means a deep machine learning model was used.

The SORT framework uses the Kalman Filter algorithm to track the objects. The Kalman Filter is an estimation algorithm which basically uses the position and the velocity of the objects to estimate where the object is going to be next.

This algorithm is then enhanced using a deep neural algorithm so that the algorithm tracks not only based on the position and the velocity of the object but also the deep features of every bonding box. For example, if you are tracking a person, we know that the person at position (x1, y2) in frame 1 is the same as the person at position (x19, y19) because of the velocity of the bounding box around the person but also because of the features of the person.

My Journey

When I first started working on this project, I discovered a library called PixelLib which is a library that allows for object detection with five lines of code. It uses instance segmentation (which is where the pixels of the objects are highlighted as opposed to object localization where a bounding box around the object is drawn) and MaskRCNN. With this, I was able to accomplish detecting cars that passed by. The only issue with this was that because it was a library, I could not modify the code to count the number of cars.

Next, I discovered YOLO for object detection, after a bit of research, I found the following GitHub repo: https://github.com/theAIGuysCode/yolov4-custom-functions

This repo used YOLOv4 for object detection and had a class where you could implement custom functions. At this point, I was unaware of object tracking. It took me a solid day to understand the code as I had never done object detection before. Then I begin the process of implementing a custom function that counted the number of cars. At this point, I was unaware of Deep SORT and object tracking.

My logic was that at each frame, I would check if a largest y value of the bounding box of the object was between a certain pixel value, if it was, I incremented the count. This required passing parameters between three different classes and it was a bit messy. Surprisingly, this worked pretty well, the key was making sure that the range where the counter would be incremented was the right range to ensure that the same vehicle was not counted twice.

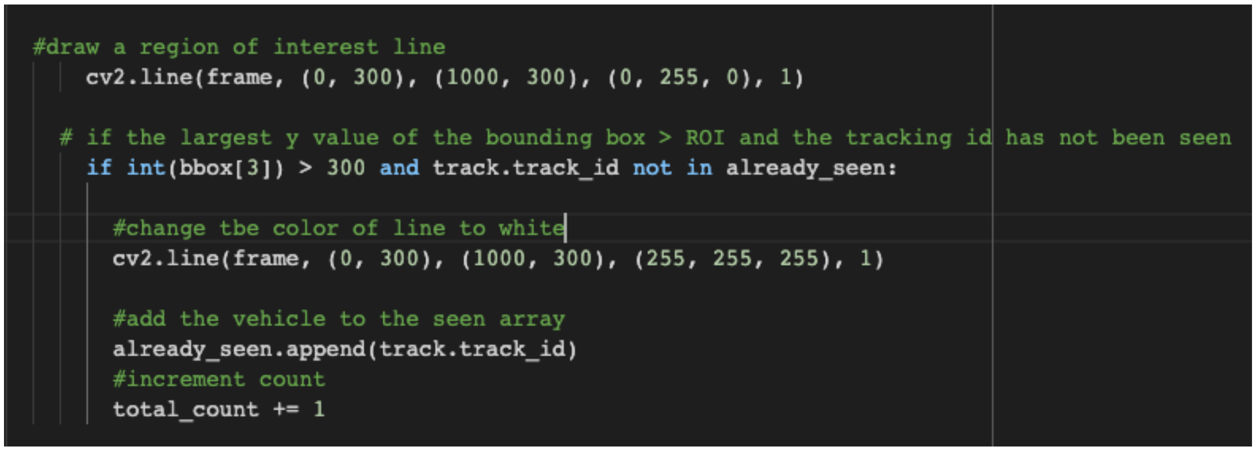
The issue though was that I knew this method would not generalize well because if the vehicle was stopped or was slow at the range where the counter is incremented, then the same vehicle would be counted multiple times since for multiple frames the largest y value of the bounding box would fall within the range.

After this failed attempt, I discovered Object Tracking and Deep Sort and a new GitHub repo that used YOLOv4 and DeepSort. I was able to understand the code pretty easily from my past failed experience .

With this new repo, I used the same logic as above to count cars, except because now I had a tracker which had a unique tracking ID with each object, I could ensure that the same object was not counted twice.

Instead of a range, I set one yvalue for my region of interest.

If the maximum yvalue of a bounding box is greater than the region of interest and if the tracking_id was not seen before, I would increment the count. This works for vehicles going all directions, since if the vehicle is coming from the other direction then the maximum y-value of the bounding box of the vehicle would already be greater than ROI, so as soon as it enters the frame, the counter would be incremented.



```
#draw a region of interest line
    cv2.line(frame, (0, 300), (1000, 300), (0, 255, 0), 1)

  # if the largest y value of the bounding box > ROI and the tracking id has not been seen
    if int(bbox[3]) > 300 and track.track_id not in already_seen:

      #change tbe color of line to white
      cv2.line(frame, (0, 300), (1000, 300), (255, 255, 255), 1)

      #add the vehicle to the seen array
      already_seen.append(track.track_id)
      #increment count
      total_count += 1
```

This worked pretty well. Adding the direction was simply a matter of printing the x,y pixel values of the vehicle.

For the given test video, the results video only counts 7 cars when in reality there is 8 cars that passes the ROI line, this is because the tracking ID for the second car and the tracking ID for the 8th car is the same. It should be unique but for some reason, the network thinks its the same car.

The colab attached was the colab that I created and used. I have pretty detailed instruction on the colab: https://colab.research.google.com/drive/1IM5CxQfIqAHJPXNN4iBl6m4GHta9heZl#scrollTo=6uh_wrL4c3re

In summary, first the GitHub repo is cloned and the requirements are installed, then I mounted the google drive so I could import and export the input and output videos quickly. I then downloaded the pre-trained weights (this was from the model that was trained on the Microsoft COCO dataset which has 80 different object classifications). Then I built and ran the YOLOv4 model, lastly, I simply ran the object_tracker.py script.

**** While testing, It is extremely important that after you clone the GitHub repo, you substitute the object_tracker.py class in the following location

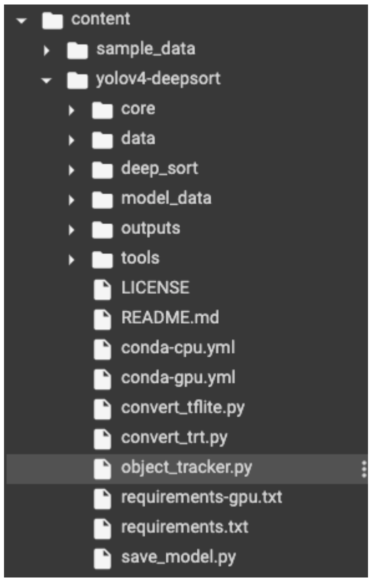
/content/yolov4-deepsort/

with the my object_tracker.py attached in this assignment as that is the object_tracker.py class that I modifie**d.**

**** After you download the weights, please move the yolov.weights file inside the data folder

/content/yolov4-deepsort/data

I could've simply modified the save_model.py file so that you wouldn't have to move the weights file but I felt like that would have been more confusing for you guys to test as it is easier to move the file inside the data folder.



```
content
  sample_data
  yolov4-deepsort
    core
    data
    deep_sort
    model_data
    outputs
    tools
    LICENSE
    README.md
    conda-cpu.yml
    conda-gpu.yml
    convert_tflite.py
    convert_trt.py
    object_tracker.py
    requirements-gpu.txt
    requirements.txt
    save_model.py
```

Final Thoughts

This was definitely a challenging task, but I absolutely love challenges which is probably the only reason I put in as much time and effort as I did into this project. To be honest, I forgot that I was doing this project as part of an interview process.

It is quite crazy to me how much I learnt in this short span of time (Parkinson's law is very real) about object detection and object tracking. I now understand (to some extent) the state of art algorithms for real-time object detection and object tracking, I have so many side project ideas in mind that I can accomplish with the knowledge that I have gained.

One thing I want to mention is that the code provided should also work for realtime data, run the following line, instead of the last line in the colab!

```
!python object_tracker.py --video 0 --output ./outputs/webcam.avi --model yolov4
```

Since this is for webcam, instead of cars, maybe try tracking the number of people in the frame. On line 163 of the object_tracker.py change the [‘car’] to [‘person’].
Pretty cool eh!

All in all, I had a ton of fun with this and I learnt a lot, thank you for this challenge.