Assignment 1 Real-time Object Detection and Classification James O’Rourke 15334121

Pipeline Design:

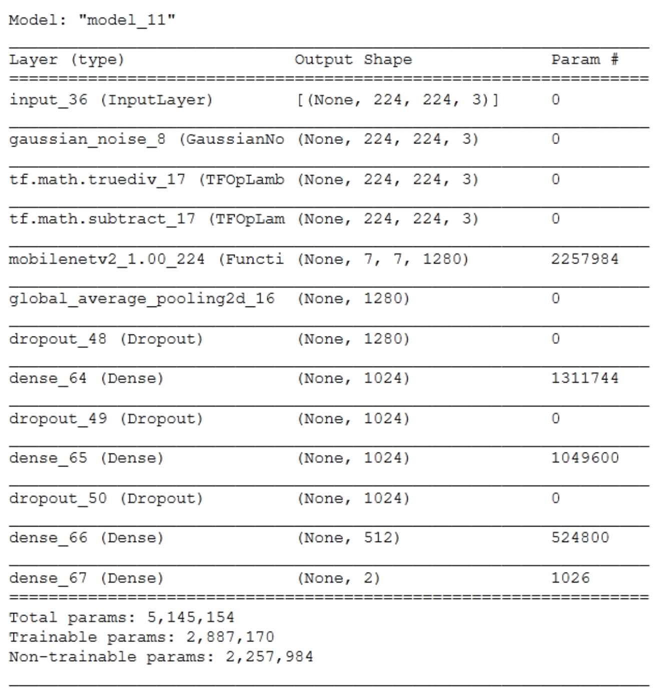
For my pipeline design, I knew I would be unable to run a model that could run in real-time. While I would have been able to run a basic object detector such as the very lightweight Tiny Yolo V3 (Cepni, Atik and Duran, 2020), I would not have the computational power to extract the region of interest and pass it to my mobilenet based model in real-time.

As a result, the best solution I came to was using a cv2.VideoCapture object. This gave me a flexible way to iterate and do operations on frames of a video while maintaining the original order of the video frames. For storing results I created a python list to store each count i.e. one for the Count of Cars, one for sedan count etc. I appended the result of the current count at the end of every frame. In the end, I was easily able to convert these into a data frame which was then exported into an excel file.

When it came to implementing TinyYolo I first followed the tutorial on the website where I downloaded the weights[1] but I soon found this was useless to me as while I could download results as a JSON file I needed to extract the boundary boxes using created by Yolo and pass them down the pipeline. I ended up using a solution online[2] that involved reading the Yolo cfg files and weights into OpenCv2 using the function readNetFromDarknet(). This gave me a lot more options as I could parse the output layer and collect the Region of Interest (referred to as ROI from here) and then push them on through my classifier. I collected the ROI by only cropping the area of the boundary box that Yolo detected a car in.

In part 2 I take the ROI and pass it through my classifier, the mobilenetV2 based Keras model. I chose mobileNetv2 due to its accuracy and its speed(El-Dairi and House, 2019). I decided to do this frame by frame instead of passing all the region of interests to the second stage. This way when I came to optimise the pipeline flow there would not be a time where one part of the pipeline was not being used.

[1] https://pjreddie.com/darknet/yolo/
[2] https://www.learnpythonwithrune.org/how-to-get-started-with-yolo-in-python/



```
Model: "model_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_36 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
gaussian_noise_8 (GaussianNo (None, 224, 224, 3)       0
_________________________________________________________________
tf.math.truediv_17 (TFOpLamb (None, 224, 224, 3)       0
_________________________________________________________________
tf.math.subtract_17 (TFOpLam (None, 224, 224, 3)       0
_________________________________________________________________
mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
_________________________________________________________________
global_average_pooling2d_16  (None, 1280)              0
_________________________________________________________________
dropout_48 (Dropout)         (None, 1280)              0
_________________________________________________________________
dense_64 (Dense)             (None, 1024)              1311744
_________________________________________________________________
dropout_49 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_65 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dropout_50 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_66 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_67 (Dense)             (None, 2)                 1026
=================================================================
Total params: 5,145,154
Trainable params: 2,887,170
Non-trainable params: 2,257,984
_________________________________________________________________
```

1 My Keras Model Layers

The Model I used in part 2 is a Keras model based on MobileNetv2. I followed an online guide that I sued to get the basics right on transfer learning right[3]. I decided to use a size of 224 by 224 as the input image as the video we are making predictions from does not have very high resolution. I added 2 layers before the mobilenet layers: 1 that was a layer that normalise the input to prepare it for mobilenet and another to add some gaussian noise to the input.

After mobileNet I used a max-pooling layer to reduce the output from the mobilenet layers. I then added some dense layers with ReLu activation and some dropout layers to avoid overfitting.

Calculating F1 Score:

Before we evaluate the results of our model, I need to discuss the way I calculated the F1 score. F1 score is the which is a harmonic mean of precision and recall. I referenced a medium article[4] on the topic when it came to this. I realised it would not be possible to truly calculate the F1 frame by frame as this would be too time-consuming. Thus, my F1 score is not a true F1 score for the detection of cars but rather the count of cars in the frame.

First, I calculated the Recall and Precision for each frame. I regarded the ground truth count as a reference to compare to. If my count was equal to the ground truth count, my Recall and Precision would be 1 and so my F1 score would be 1. If my count was higher than the ground truth count, I regarded the difference as False Positives and the true count as the True Positives. If my count were

[3] https://github.com/ferhat00/Deep-Learning/blob/master/Transfer%20Learning%20CNN/Transfer%20Learning%20in%20Keras%20using%20MobileNet.ipynb

[4] https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd

lower than the ground truth count, I would regard my count as the True Positive and the difference between my count and the truth as False Negatives.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

My accuracy Metric F1 Score

Once I had precision and recall for every frame, I could calculate the F1 score for every frame. I then took the average to evaluate the overall performance. While this solution works, it has some major downsides. The biggest being that it is easily fooled. For example, if I had a frame with 4 cars, 2 SUVs and 2 sedans, and classified the sedans as SUVs and vice versa I would get an F1 score of 1 when it should be 0. As a result, we should use the F1 score tentatively and regularly check the results manually to see how we are doing.

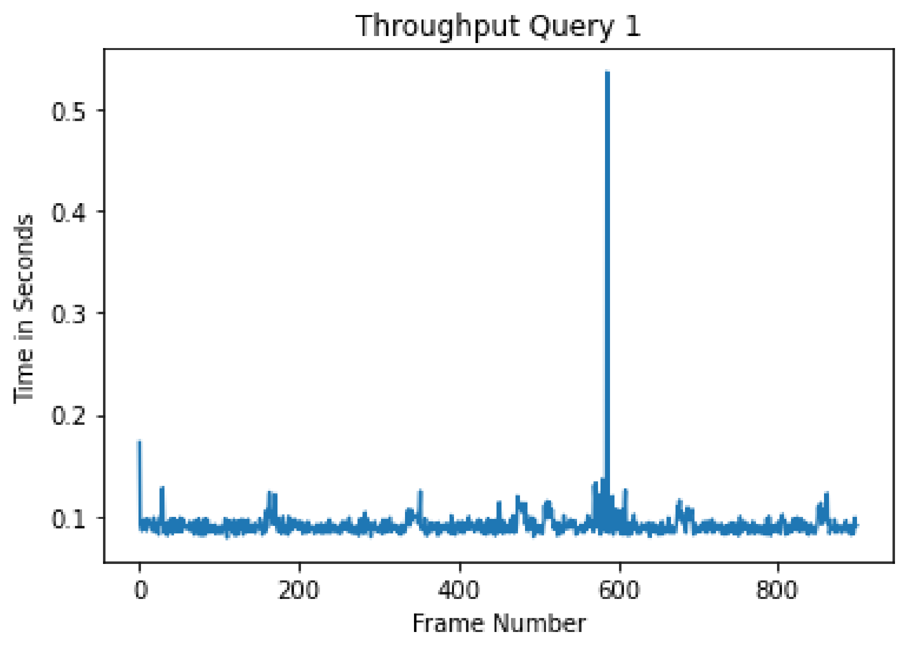
Pipeline Output

Query 1 Results:

For my Object Detector, I got an F1 result of 0.8449. I was quite happy with it as it is a very fast model but at the cost of accuracy. I improved this result a bit by increasing boundary boxes when there were too small and experimenting with the thresholds for the non-maximal suppression function.

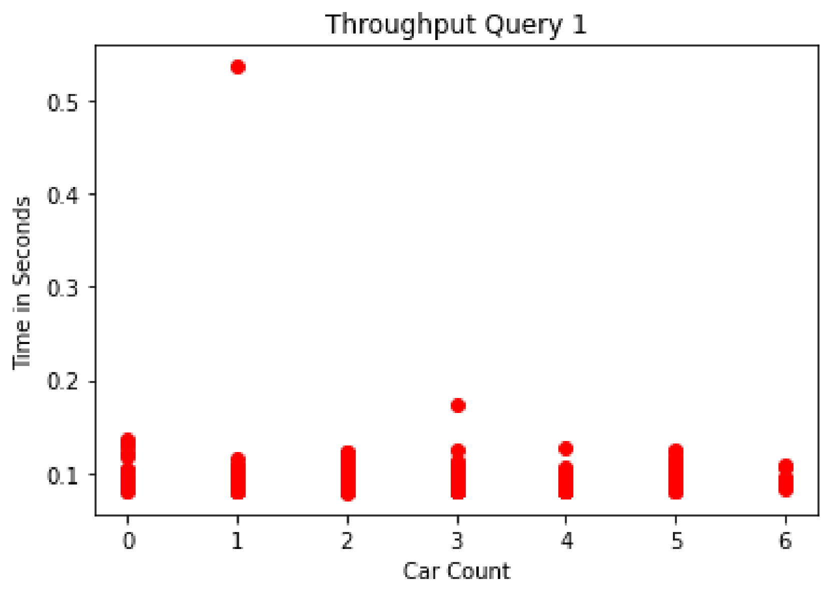
Query 1 had an Execution time of 82.496 seconds for 900 frames which works out roughly at 0.092 per second for throughput. I am happy with these times as it could work in real-time on a very slow framerate video. Please note I recorded all execution and throughput times before adding an option to display a random frame

As you can see below, I have plotted the throughput times for each frame and it seems to be a very consistent bar some 2 times and these outlying times could be explained by my CPU dealing with something else.



Throughput Query 1

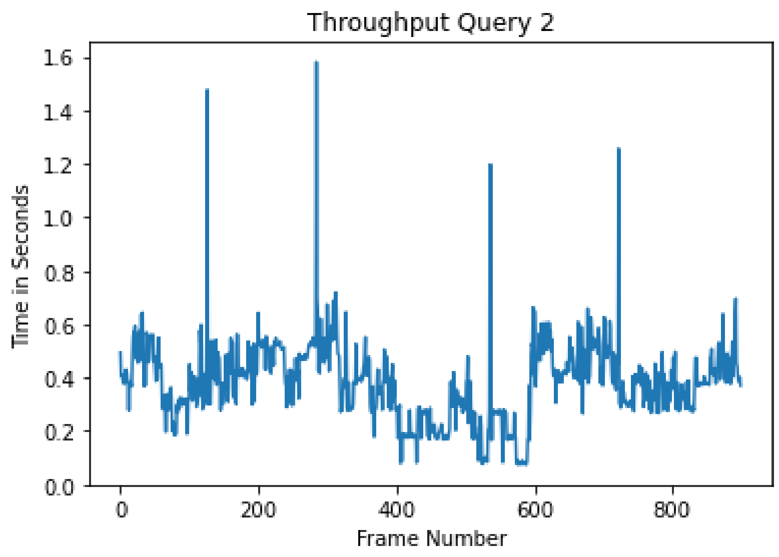
I was also interested in the number of cars in a frame that affected its throughput time but fortunately, it seems to not affect it. This is good as it suggests that our model would be able to handle a busier street without harming its performance too much.



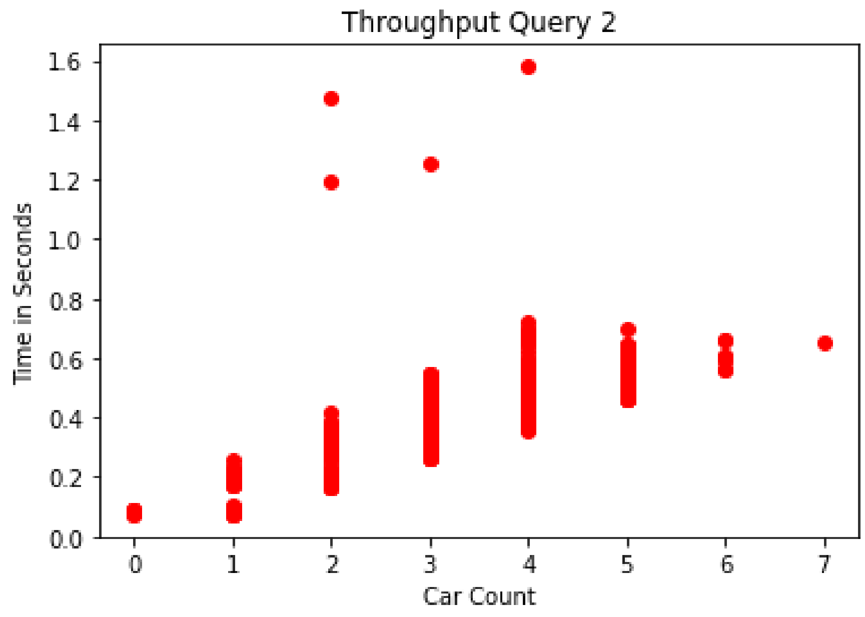
Throughput Time Per Car Count

Query 2 Results:

For Query 2 the best results I could get were an F1 score of 0.6581 for SUV count and 0.6613 for Sedan count with an average throughput time of 0.348. This is far from being real-time as it took 313.43 seconds to process the whole 30-second video. As you can see below there are more outlying throughput times and throughput times are higher overall. There seem to be 4 frames where the throughput time is over 4 seconds. I am thankful this was the worst because scenario as if every frame was like this execution time would have 15 minutes.



I think it's interesting to note that there seems to be a trend in the graph below. As the number of cars in the frame rises there seems to a rise in the throughput time. This is an effect of my pipeline where the classifier is run for every detected object from Yolo.



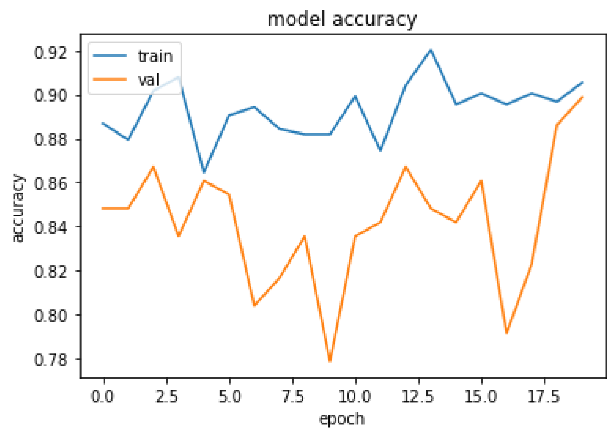
Model Training Evaluation

For my training dataset, I downloaded 550 images of both car types using the chrome extension Image Downloader. I manually went through and deleted any that I thought would contaminate the training of my classifier. I then split them into 2 different subfolders with 20 per cent of them being my validation data. Within each of these folders, I had a subfolder corresponding to the class names and stored each image to their respective class name. I have included this folder in my download link.

I used an ImageDataGenerator to generate my training data from here using the flow_from_directory function. This was a very useful method as it generated augmented Image data from my images for every epoch of training. I told it to add some random augmentation such as flips, shears, rotations and zooms. I also passed it a function to prepare all my training images. This resized them and expanded the dimensions. This was just to ensure that input shape would not be an issue when passing to the Keras model. I repeated the process for my validation but did not augment the data this time.

I loaded the mobileNetV2 model with the imagenet weights and I didn’t include the top layer. I froze the weights in the base model and then added my layers around it. I spent a lot of time trying different combinations of layers and different data augmentations. If my training accuracy jumped to 1 very quickly, I knew that I was going to overfit my data and stopped and added some dropout layers or some more augmentation in my data generator.

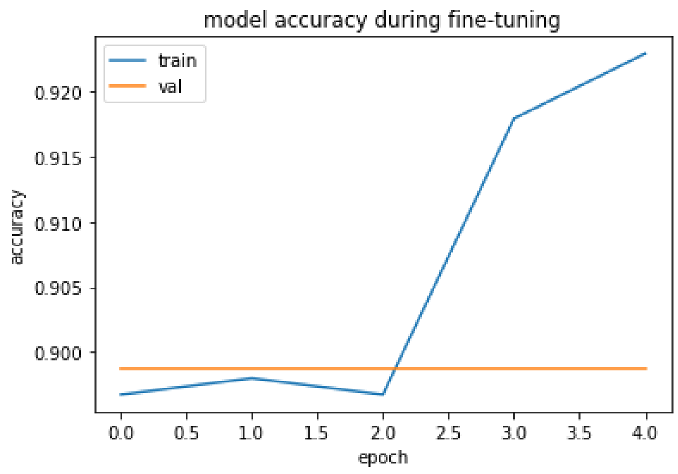
For my training, I used Adam as the optimiser, categorical_crossentropy as my loss function and Accuracy as my metric. I used 20 epochs as I didn’t see any improvement if I increased this and training time got too long past this.



You can see my training progress on the left. I was a bit surprised that it didn’t increase more smoothly. The validation data, in particular, spent more time decreasing than increasing I think.

My final training accuracy was 0.9060 and my final validation accuracy was 0.8987.

After I trained the model I unfroze the weights in the base mobilenet Model and done some fine-tuning training using all the weights in the model. I used a very small learning rate and only 5 epochs for this Fine Tuning as I did not want to overfit the model.

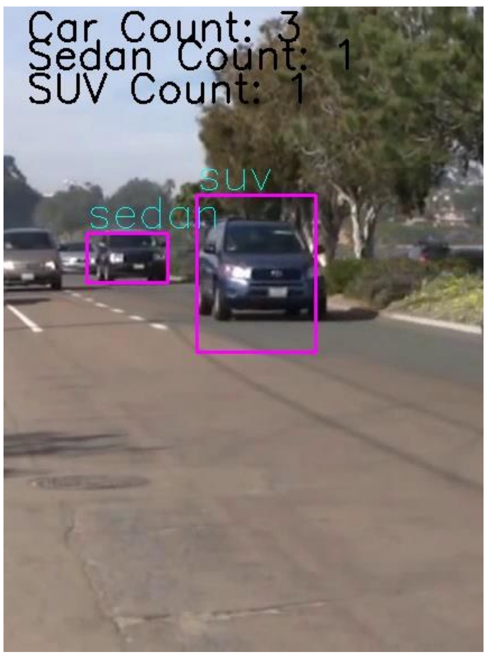


Sadly, my validation accuracy did not increase during this fine tuning and stayed at 0.8987 but I did see a small increase in my training accuracy which stopped at 0.9268.

Design Strengths and Weaknesses

Overall, I think there are some big weaknesses in my model mostly on the accuracy side. The detector part works as well as could be expected for Tiny Yolo and I think I improved a bit by dealing with overlapping boundary boxes. At the beginning I had a lot of trouble with smaller boundary boxes of cars in the distance but unfortunately, I still had some problem with boundary boxes not overlapping but one being contained completely within another. I was not able to solve this problem and openCVs Non Maximal Suppression could not seem to stop this case and thus it hampered my accuracy slightly.

My classifier was a source of problems for me. My model found it quite difficult to deal with the low-quality video. Most attempts to correct this led to severe over fitting. In the end I found the best F1 score I could get was approximately 0.67 for classifying F1 scores. Due to the problems with the F1 score I discussed earlier this should definitely not been taken as the true accuracy of the model. Looking back on my video shows a lot of instances where cars are misclassified.



My method of cropping out the ROI also led me to some problems. I had to avoid getting crops that overlapped the edge of an image as this would cause my program to crash. As result you can see the car of the left is counted in the car count but does not get classified as either an SUV or Sedan.

On to some good news about my model I found it performed fast. For reference I have a Intel i5

CPU with 2.50GHz. My object detection with yolo had a throughput of 0.092 which I was happy with as It could run on a very low frame rate, 10 FPS, video in real time. While my full cascade was slower, but I was still happy with it with a throughput speed of 0.348, as it carried a lot more overhead with the classifier being run for every cropped image. The throughput reflects this as there was a trend of it increasing with the number of cars in the frame. Unfortunately, I was unable to optimise the pipeline.

Another downside of my model was that I would get vastly different results from training using the same hyperparameters. I suspect this is due to my data being generated/augmented using ImageDataGenerator. While this is not a bad thing it itself, but you would expect similar weights from same training data which my method robs us off.

Download Link for Submission Files

https://nuigalwayie-my.sharepoint.com/:u:/g/personal/j_orourke11_nuigalway_ie/ET2HYeVVSmJOhPluTuZlSYABtMLR8tz3XEGdipSqlXmiDg?e=B7eWE4

References

Cepni, S., Atik, M. E. and Duran, Z. (2020) ‘Vehicle detection using different deep learning algorithms from image sequence’, *Baltic Journal of Modern Computing*, 8(2), pp. 347–358. doi: 10.22364/BJMC.2020.8.2.10.

Howard, A. G. *et al.* (2017) ‘MobileNets: Efficient convolutional neural networks for mobile vision applications’, *arXiv*.