Deep Learning Object Detection Report

1. Introduction

When humans look at an image, they're able to analyze the image, process the contents and understand what they're seeing and are instantly able to connect the objects in the image to objects they know. But how does a computer do this? To a computer, an image is a collection of RGB pixels; so, how exactly does a computer look at an image and decipher the objects within the image? This is one of the fundamental problems of computer vision known as object detection where an image is input, and the output would be a precise estimation of the classification of the objects as well as the location of the object classified in the image which are usually represented as bounding boxes. There has been considerable amount of work done for object detection using deep learning techniques. Neural networks adept to object detection have been trained with high accuracy rates on large datasets.

In this project, the main objective was to learn to partially re-train a convolutional neural network adept at object detection to only detect two kinds of objects: cars and trucks. The input to our trained model would be an image and the model would predict the classification as well as locations of the cars and/ or trucks within the image. The rest of the report is organized as follows: Section 1 introduces the project, section 2 explains the architecture used. The details of the dataset and training as well as graphical results and analysis are showcased in section 4 and section 5 concludes the paper.

2. RetinaNet

For this project, transfer learning was used to retrain an object detection model to only detect two classes: cars and trucks. The model I used was RetinaNet which is a one-stage detector and makes use of focal loss. The backbone for RetinaNet that was used was ResNet50 which does feature extraction. In addition to this, there are two subnetworks for classification and bounding box regression; this forms the RetinaNet. The focal loss, as shown in equation 1, focuses on training on hard negatives i.e. the rare class where p_t is the model's estimated probability, γ is used to focus on the hard examples and α is to offset the class imbalance. The RetinaNet model architecture is shown in Figure 1. In addition to this loss, classification and regression bounding box loss are also calculated during training.

$$FL(p_t) = -\alpha(1 - p_t)^{\gamma} log(p_t) \tag{1}$$

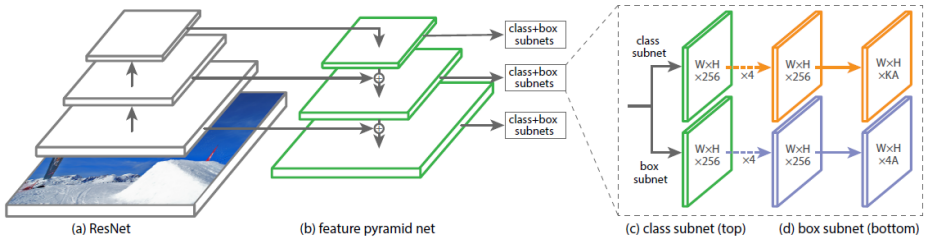

Figure 1: RetinaNet Architecture

When analysing the data before beginning the training, I noticed a major difference in the number of instances of cars and trucks in both the training and validation datasets. Since the RetinaNet loss function deals with class imbalance, I felt it was the best model to use for this project.

3. Experimental Results & Analysis

In this section, I explain the training and implementation details, dataset used, explain the hyperparameter tuning done before and during training and showcase the results of the experiments.

3.1 Dataset and Implementation Details

The dataset for this project were images of a junction which showcased cars and trucks. Since, the coordinates given were normalised and in a different format to what I required for training, I converted them to create training and validation annotation text files. The original coordinates were converted to the format <x1>, <y1>, <x2>, <y2> where (x1, y1) was the top-left corner of the bounding box and (x2, y2) was the bottom right corner of the bounding box. The first step was to download the model implementation as well as the pre-trained weights which were the result of training the RetinaNet on the COCO dataset for the purpose of multi-class object detection. Training was done using the training script on a GPU.

3.2 Hyperparameter Tuning

To be able to decide the batch size I want to use, I conducted experiments with different batch sizes, keeping all other variables constant in all experiments. I tried four different batch sizes which were: 1, 4, 8 and 16. I considered the mAP score for the validation set after running the model for 10 epochs for each experiment. I found that batch sizes 8 and 16 did well and the difference in their results was negligible, as can be seen in Figure 2. I also took into account the time it took to train; batch size 8 was significantly faster as showcased in Figure 3 and this was the batch size, I chose to continue training with batch size 8.

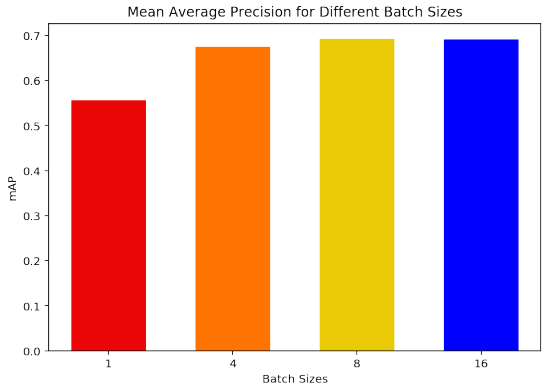

Figure 2: mAP for Batch Sizes 1, 4, 8, 16

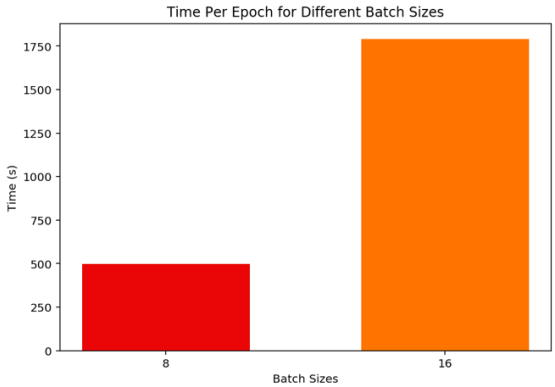

Figure 3: Training time for 1 Epoch for Batch Sizes 8, 16

3.3 Results

The model uses three loss functions to evaluate performance during training: the focal loss function, classification loss function and regression loss function for the bounding boxes. The change in the loss functions through training for 17 epochs can be seen in Figures 4, 5 and 6. The loss decreases until epoch 15 and then steadies. The model stopped training at epoch 17 as early stopping was used. To evaluate the performance of the model during and post training, Mean Average Precision (mAP) is used on the validation dataset and test set respectively. The improvement in mAP as training progresses can be seen in Figures 7, 8 and 9 which ensures that the chosen parameters work well.

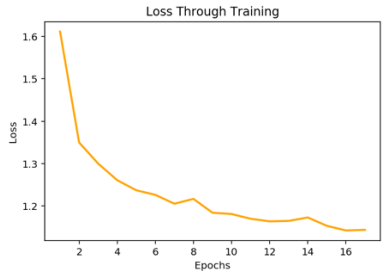

Figure 4: Focal Loss for 17 Epochs

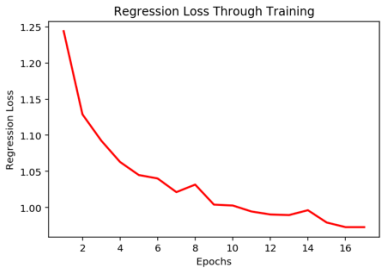

Figure 5: Regression Loss for 17 Epochs

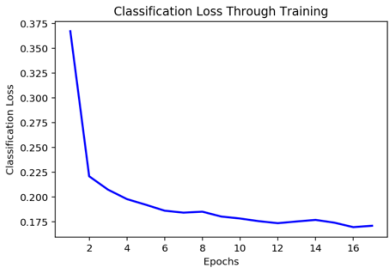

Figure 6: Classification Loss for 17 Epochs

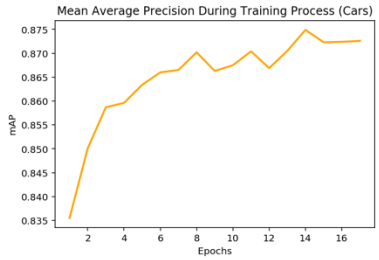

Figure 7: mAP during training for Cars

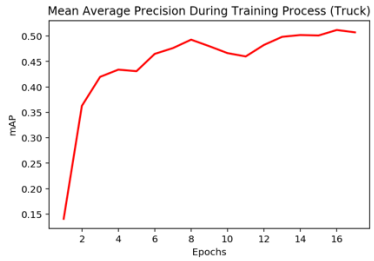

Figure 8: mAP during training for Trucks

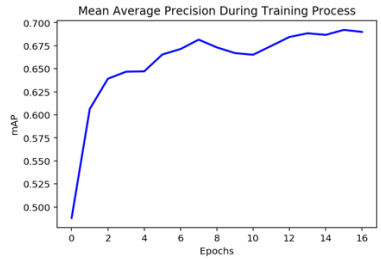

Figure 9: mAP during training

The evaluation of the model on the test set is showcased in Figure 10 which represents the Mean Average Precision (mAP) on 50% of the test set after each epoch until 17 epochs. As can be seen in the figure, the performance steadies towards the end with no further improvement in mAP.

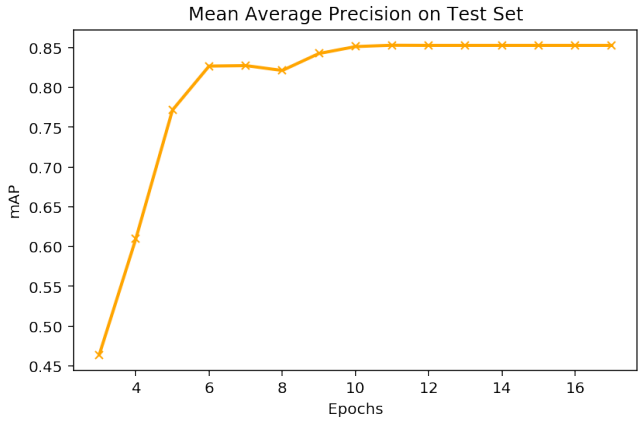

Figure 10: Evaluation using Test Set for 17 Epochs

The output of the model can also be visually represented on the test images. Figure 11 shows two randomly chosen images from the test set for which the model performed predictions. The model predicted the class (Car or Truck), the confidence score and the coordinates for the bounding boxes for each predicted object. The image on the left shows two cars being predicted (dark blue box). The image on the right showcases multiple cars and a truck being predicted (light blue box). The sample predictions showcase a positive result.

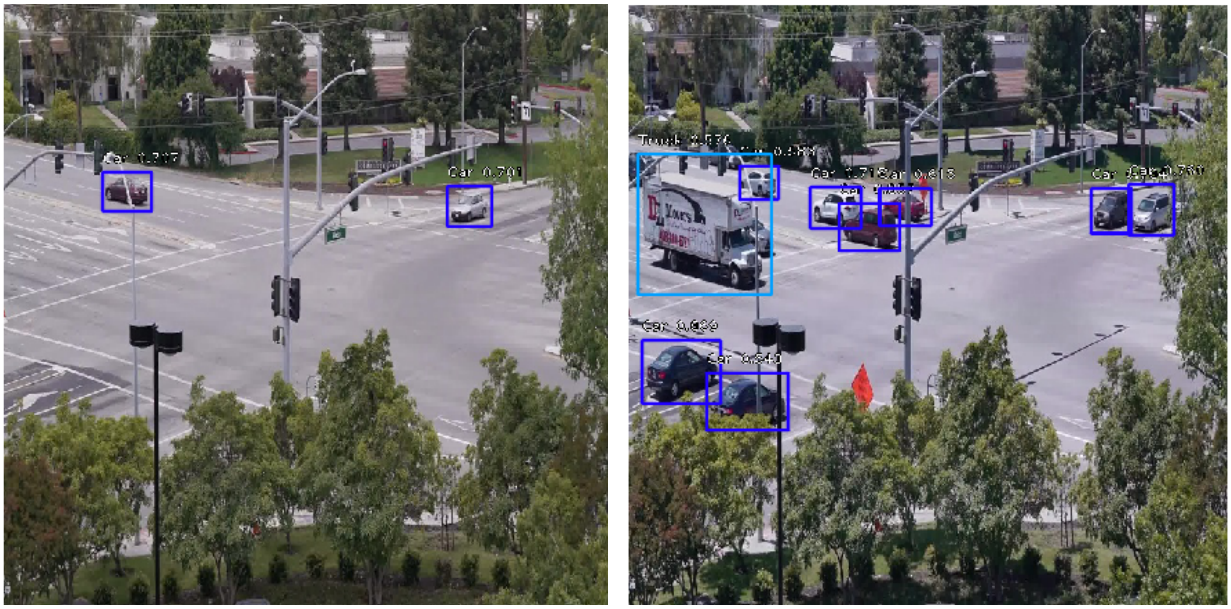

Figure 11: Sample Predictions for Randomly Chosen Test Set Images

3.4 Running the Code

The RetinaNet implementation called "keras-retinanet" would have to be installed as well as pre-trained weights for this network. Using the framework and weights, you would re-train the model using custom training and validation annotation text files using the training script. The evaluation script can be used to test the model and predictions. Instructions to run the file can be found in "ReadMe.txt" uploaded with the python scripts.

3.5 Bias Variance Analysis

Ideally, a model should be balanced with bias and variance to ensure that it neither overfits nor underfits on the training data i.e. the predictions should be as close to the ground truth as possible. Figure 12 showcases a scatter plot of ground truth and predictions for the validation set where the left image is for CX, CY coordinates and the right image are W, H. We can see that the model has trained well as the ground truth and predictions aren't sparse.

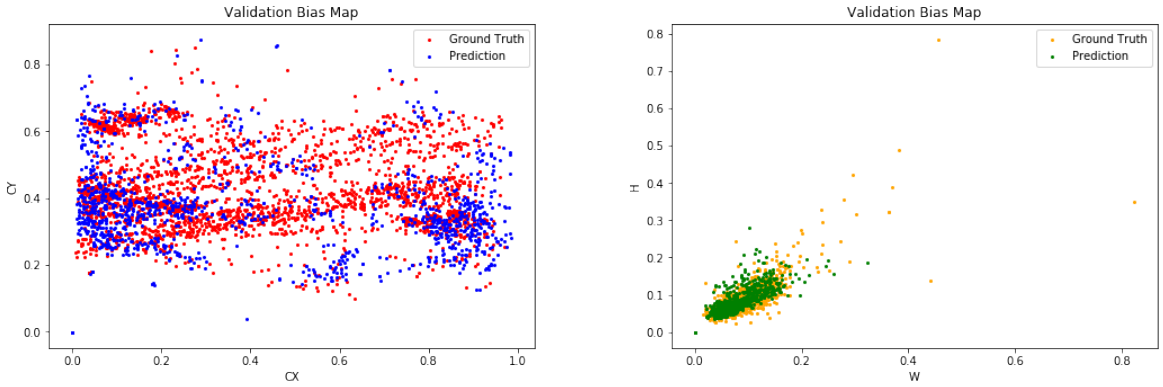

Figure 12: Validation Map to Showcase Bias and Variance

4. Conclusion

Object Detection is a fundamental problem of computer vision and being able to leverage the power of a deep neural network to perform this task is important. In this project, I was able to explore deep learning models that have been trained for object detection, I was able to learn and understand object detection better as well as use transfer learning to retrain RetinaNet to detect cars and trucks. My model did well with a mAP score of 0.8531; however, I hope to use what I've learned to train a better model with higher accuracy in the near future.

References

[1] Anastasiu, David. COEN 342 Class Slides. Santa Clara University, 2021
[2] Sik-Ho Tsang, "Review: RetinaNet — Focal Loss (Object Detection)" 2019
[3] Keras RetinaNet Tutorial: https://keras.io/examples/vision/retinanet/
[4] Matplotlib tutorial: https://matplotlib.org/