MSE491:

Introduction to Machine Learning in Mechatronic Systems

Project Report: Road Object Detection

Report Due: March 21st, 2021

|  |  |
| --- | --- |
| Mahmoud Kamaleldin  301324899 | - Developed models from the frameworks: Darknet, Pytorch  - Ran models on NVIDIA Jetson Nano (+ Tensorflow RT) |
| Andrei Divinagracia  301360608 | - Developed models from the frameworks: Tensorflow  - Ran models on Raspberry Pi 4 |

Table of Contents

[Introduction 4](#_Toc70078141)

[Methodology 5](#_Toc70078142)

[Detectron2 6](#_Toc70078143)

[YOLOv5 PyTorch 8](#_Toc70078144)

[Darknet 9](#_Toc70078145)

[Deployment 9](#_Toc70078146)

[Raspberry Pi 4 10](#_Toc70078147)

[Nvidia Jetson Nano 10](#_Toc70078148)

[Results and Discussions 11](#_Toc70078149)

[Faster R-CNN 13](#_Toc70078150)

[YOLOv3-Tiny 14](#_Toc70078151)

[YOLOv4-Tiny 15](#_Toc70078152)

[YOLOv4-Tiny TensorRT fp32 16](#_Toc70078153)

[YOLOv4-Tiny TensorRT fp16 17](#_Toc70078154)

[YOLOv5s 18](#_Toc70078155)

[YOLOv5-Tiny TensorRT 19](#_Toc70078156)

[Comparison 20](#_Toc70078157)

[Conclusion 21](#_Toc70078158)

[References 22](#_Toc70078159)

List of Figures

[Figure 1: Performance Comparison Between Object Detection Models [4] 5](#_Toc70071164)

[Figure 2: Example Image from BDD100K dataset 5](#_Toc70071165)

[Figure 3: JSON Label for Example Image 5](#_Toc70071166)

[Figure 4: General training pipeline 6](#_Toc70071167)

[Figure 5: Detectron2 object detection models [5] 7](#_Toc70071168)

[Figure 6: Detectron2 label conversion function 7](#_Toc70071169)

[Figure 7: Comparison of performance of YOLOv5 models on MS COCO dataset [6] 8](#_Toc70071170)

[Figure 8: Comparison of performance of object detection models on MS COCO dataset [7] 9](#_Toc70071171)

[Figure 9: Raspberry Pi 4 vs Nvidia Jetson Nano specs [8] 10](#_Toc70071172)

[Figure 10: NVIDIA® TensorRT™ Optimizer Development Flow [9] 11](#_Toc70071173)

[Figure 11: Precision-Recall curve example with AP [10] 12](#_Toc70071174)

[Figure 12: Intersection over Union [10] 12](#_Toc70071175)

[Figure 13: Faster R-CNN Loss over Epochs 13](#_Toc70071176)

[Figure 14: Faster R-CNN Example Image Inference with the Generated Bounding Boxes 14](#_Toc70071177)

[Figure 15: AP Results Per Classification of Faster R-CNN Example 14](#_Toc70071178)

[Figure 16: YOLOv3-tiny Example Image Inference with the Generated Bounding Boxes 15](#_Toc70071179)

[Figure 17: AP Results Per Classification of YOLOv3-tiny Example 15](#_Toc70071180)

[Figure 18: YOLOv4-tiny Example Image Inference with the Generated Bounding Boxes 15](#_Toc70071181)

[Figure 19: mAP Results of YOLOv4-tiny 16](#_Toc70071182)

[Figure 20: AP Results Per Classification of YOLOv4-tiny Example 16](#_Toc70071183)

[Figure 21: YOLOv4 fp32 Example Image Inference with the Generated Bounding Boxes 17](#_Toc70071184)

[Figure 22: AP Results Per Classification for YOLOv4 fp32 Example 17](#_Toc70071185)

[Figure 23: YOLOv4 fp16 Example Image Inference with the Generated Bounding Boxes 17](#_Toc70071186)

[Figure 24: Figure 22: AP Results Per Classification for YOLOv4 fp16 Example 18](#_Toc70071187)

[Figure 25: YOLOv5s Example Image Inference with the Generated Bounding Boxes 18](#_Toc70071188)

[Figure 26: Precision and Recall Performance of YOLOv5s 19](#_Toc70071189)

[Figure 27: YOLOv5 TensorRT Example Image Inference with the Generated Bounding Boxes 19](#_Toc70071190)

[Figure 28: Project Results 20](#_Toc70071191)

List of Tables

[Table 1: PyTorch Inference Box Data Format 8](#_Toc70071192)

[Table 2: Performance Results of Models under Various Hardware 13](#_Toc70071193)

# Introduction

As members of Team Phantom: SFU’s Formula SAE Electric club, our team noticed there has been a bigger push to driverless vehicles in Formula SAE competitions. Formula Student Germany had announced that they will be merging all classes of vehicle with a focus on a driverless cup [1]. As a word-class competition pushes their teams to driverless, we saw this project as an opportunity to begin on the development of the AI branch of SFU’s electric race vehicle.

Between the different features that go into a driverless vehicle, object detection represents the building block in a driverless vehicle control system. The model has many different objects on the road to detect like cars, busses, and pedestrians. Our objective is to implement the model on an embedded device using a camera for real-time processing. There are many models and frameworks that we can use to detect objects on the road. In this project, we aim to compare different models on different devices and determine the best performing setup for an electric race vehicle.

Since we are developing for an automotive product, the model must both be fast and accurate. This introduces both hardware and software factors that contribute to the performance of the model. Embedded systems have a limited amount of processing power. As a result, we have to use the most optimized object detection model on the best piece of hardware we can use. The machine vision community actively develops new frameworks that consistently pushes the bar of speed and performance. We will be utilizing a few of those frameworks and comparing them between each other, like You Only Look Once (YOLO) and Fast Region-based Convolutional Network (R-CNN). We are running these models on two embedded systems, the Raspberry Pi 4 and the Nvidia Jetson Nano.

The dataset that we will be using to train the models is the BDD100k dataset which is available to download publicly [2]. This is the largest open driving video dataset with 100,000 images and 10 classes to evaluate the exciting progress of object detection algorithms on autonomous driving. The dataset possesses geographic, environmental, and weather diversity, which is useful for training models that are less likely to be surprised by new conditions.

Faster Region-based Convolutional Neural Network (Faster R-CNN), as the name implies, is a CNN that divides the image into regions to create its classifications [3]. Improvements in the speed of the model arose when the images are fed to the CNN to generate a convolutional feature map which is then fed into a similar CNN to that of lab 3. This way, the algorithm does not have to feed 2000 region proposals to the convolutional neural network every time. Faster R-CNN further improves upon that by creating an algorithm that improves upon determining the regions of an image.

The other object detection model featured in this project is You Only Look Once (YOLO). YOLO has several different versions and iterations between many developers. It is a single-stage detector architecture that is more like a Fully CNN, which extracts features through a series of layers that a class of probabilities for many overlapping regions. In this project, we are highlighting YOLOv3-Tiny, YOLOv4-Tiny, and YOLOv5s models which are trained with different frameworks. YOLO models seemingly have the best performance based off of Figure 1, but we aim to test that hypothesis on our embedded systems.

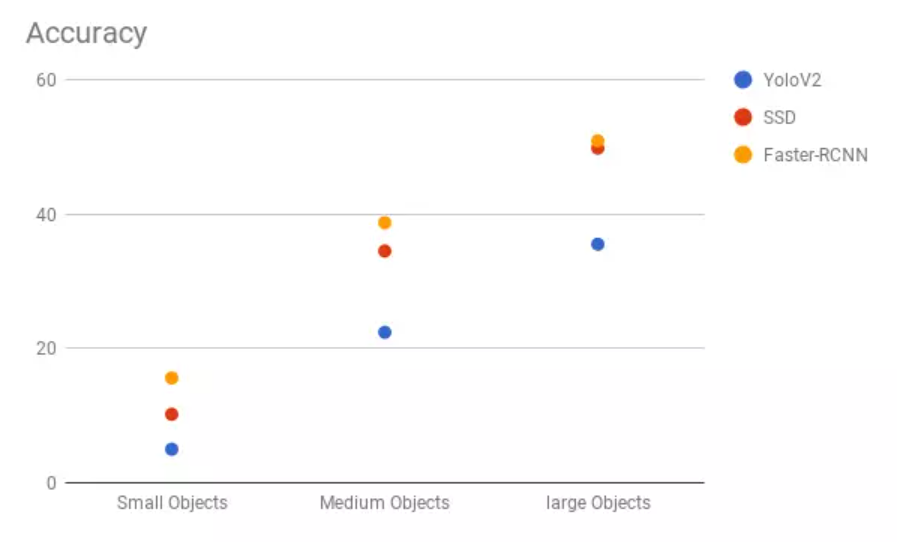


Figure : Performance Comparison Between Object Detection Models [4]

# Methodology

The purpose of this project is to train a variety of model architectures aimed at object detection. First, the dataset format is described. The BDD100k dataset used is comprised of 100,000 images split 70/20/10 for train/validation/test. The object labels for each image are stored in an accompanying JSON file shown in Figure 3.

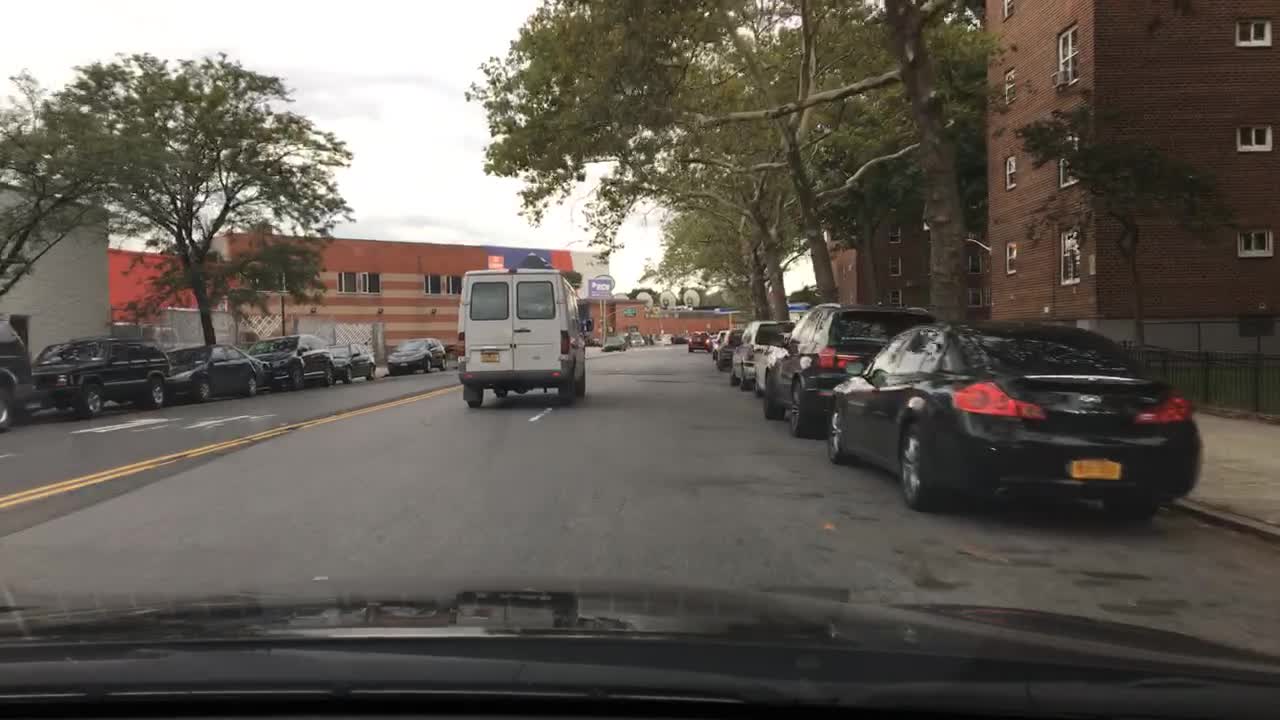
 

Figure : Example Image from BDD100K dataset

Figure : JSON Label for Example Image

Each object has certain attributes. The ones used to train the model are the category to determine object class and the box2d field which describes the (x, y) coordinates of the bounding box. The classes of interest are:

|  |  |  |
| --- | --- | --- |
| 1 – Bike | | 6 – Rider |
| 2 – Bus | | 7 – Traffic Light |
| 3 – Car | | 8 – Traffic Sign |
| 4 – Motor | | 9 – Train |
| 5 – Person | 10 – Truck | |

Now that the raw format has been explained, we move on to model creation. There are different training and deployment pipelines for each framework API and architecture. However, they all follow the same general structure outlined in Figure 4.

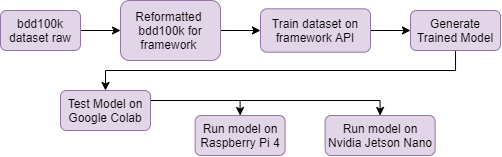


Figure : General training pipeline

The dataset is pre-processed such that the images and labels are in the format required. After that, the model architecture is defined in a configuration file that outlines how the network layers are created. The model is trained on the processed dataset which generates the weights and biases. It is initially tested on Google Colab and then deployed, when possible, on Raspberry Pi 4 and Nvidia Jetson Nano. In this project, the following frameworks are used:

* **Detectron2** is used to train the Faster R-CNN model [5]
* **YOLOv5 PyTorch** implementation is used to train the YOLOv5s model [6]
* **Darknet** is used to train the YOLOv4-tiny and YOLOv3-tiny models. [7]
* **TensorRT** is used to optimise models for Nvidia Jetson devices

## Detectron2

Detectron2 is Facebook AI Research’s state-of-the-art platform for object detection and instance segmentation. It is powered by PyTorch and features fast training and a diverse library of object detection architectures. A Faster R-CNN architecture with a 50-layer ResNet was selected from the model zoo due to having the shortest inference time. The model is trained with the Detectron2 API on Google Colab.

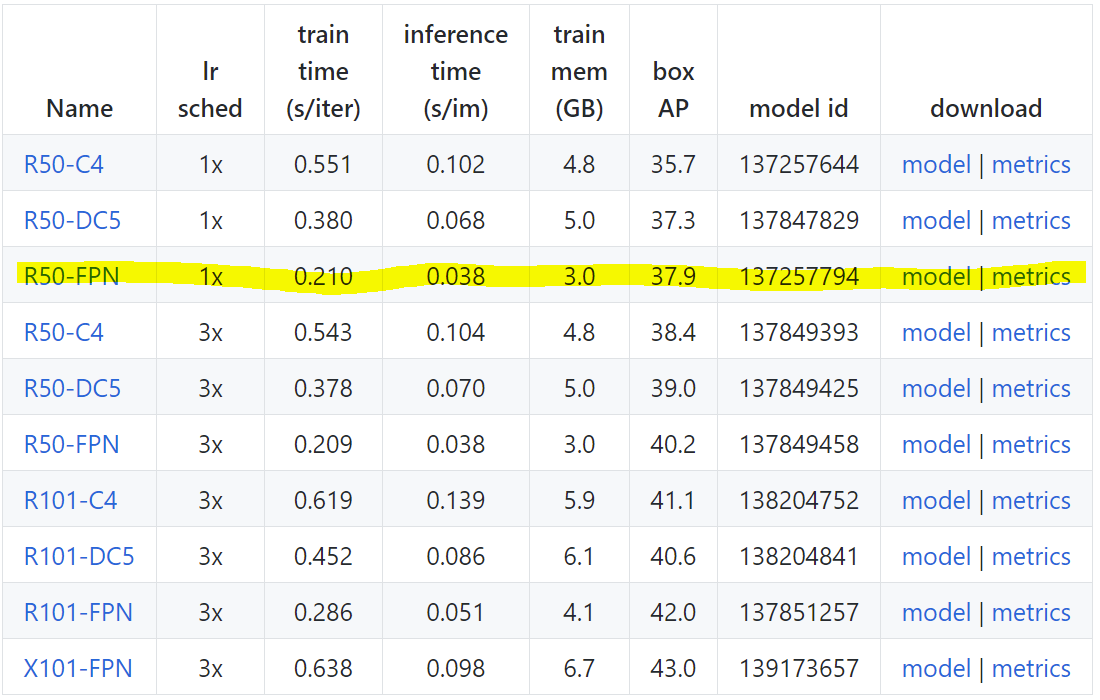


Figure : Detectron2 object detection models [5]

Detectron2 requires the data to be in a certain format in a Python3 dict. This function was used to load the JSON label file and convert it:

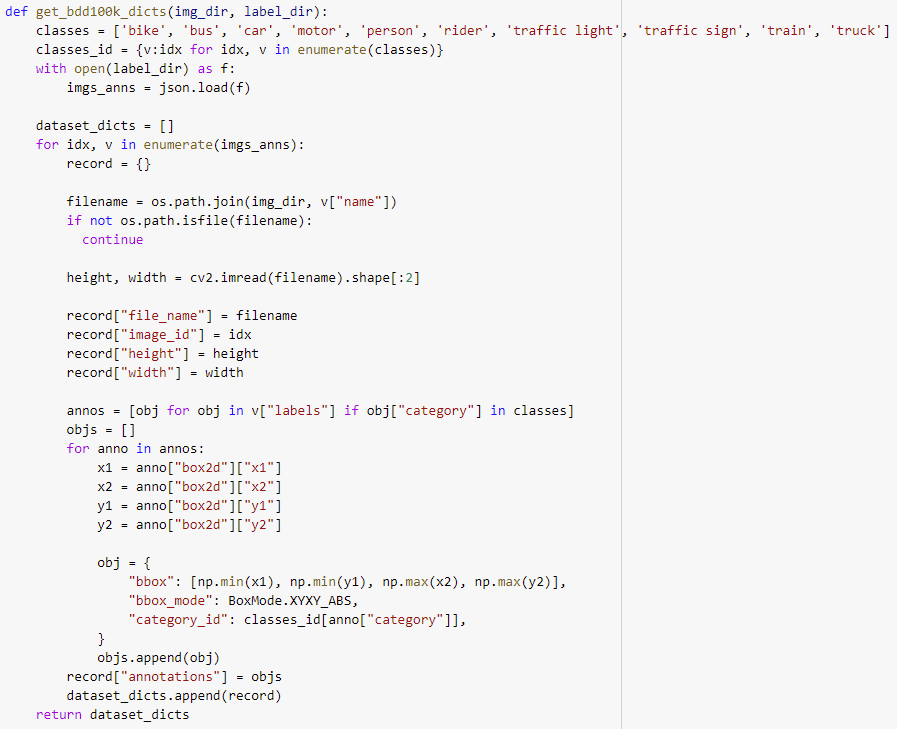


Figure : Detectron2 label conversion function

The following configurations were used for the model:

* Learning Rate: 0.00025
* Number of Classes: 10
* Batch Size: 256
* Weights: Pretrained ImageNet R-50
* ResNet Depth: 50

## YOLOv5 PyTorch

YOLOv5 is an implementation of the YOLO model in PyTorch done by Ultralytics. It does not redefine the YOLO layer architecture, but instead claims improvements in inference speed and accuracy based on improved practices and finetuning of the model as a whole.

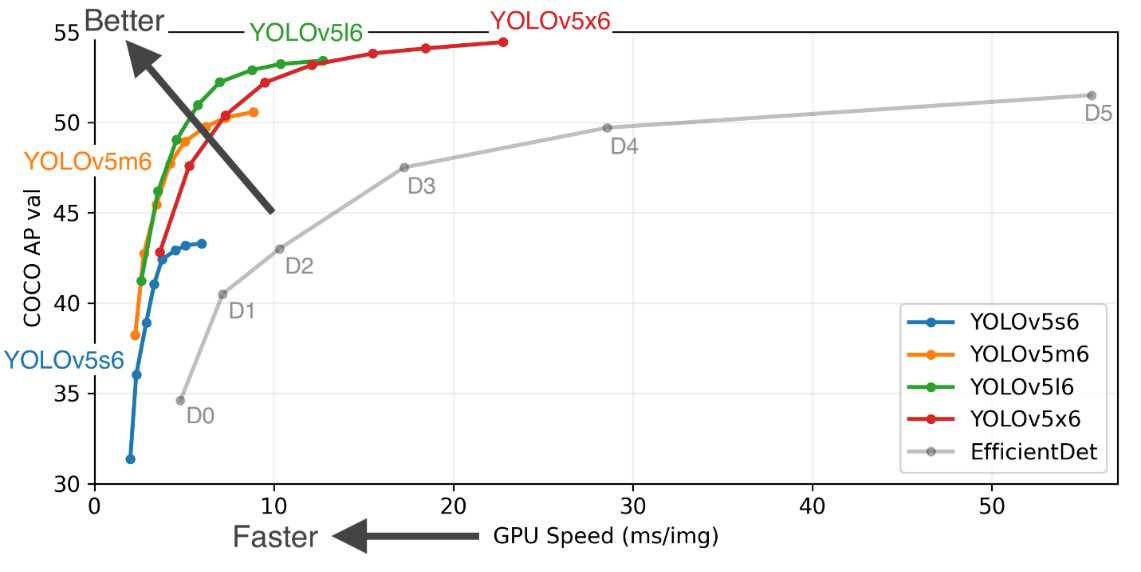


Figure : Comparison of performance of YOLOv5 models on MS COCO dataset [6]

It was a straight-forward process to select the YOLOv5 variant. There are four sizes, s, m, l, and x that are labeled based on the number of parameters in the model. YOLOv5s was selected as it had the fastest inference speed. The configuration of the model is defined in a yaml file that defines the layer shapes and the number of classes the model will predict.

The training process is straightforward, there is a train.py script provided in the YOLOv5 repo. The inputs used to train are:

* Batch Size: 32
* Image resolution: 640
* Epochs: 300

The script is run with the configuration file of the model and a file that contains the path to the images and labels used for training. The label format sample is shown in Table 1.

Table : PyTorch Inference Box Data Format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | X1 | X2 | Y1 | Y2 |
| **7** | 0.218216815234 | 0.452550609722 | 0.044950433593 | 0.086712891666 |
| **7** | 0.167049833593 | 0.431297450000 | 0.011476707812 | 0.030604550000 |
| **7** | 0.727176862500 | 0.406077041666 | 0.009563921874 | 0.015302275000 |
| **4** | 0.334896662500 | 0.482305034722 | 0.008607529687 | 0.030604550000 |
| **4** | 0.444403567578 | 0.467002756250 | 0.007651136718 | 0.034005054166 |
| **4** | 0.517089374609 | 0.459068249305 | 0.007651136718 | 0.040806065277 |

## Darknet

Darknet is an open-source neural network framework written in C and CUDA that specialized in training object detection models. It is easy to train and deploy on both CPU and GPU platforms. This framework was used for training the YOLOv3-Tiny and YOLOv4-Tiny models. The tiny versions were selected as they have the fastest inference times.

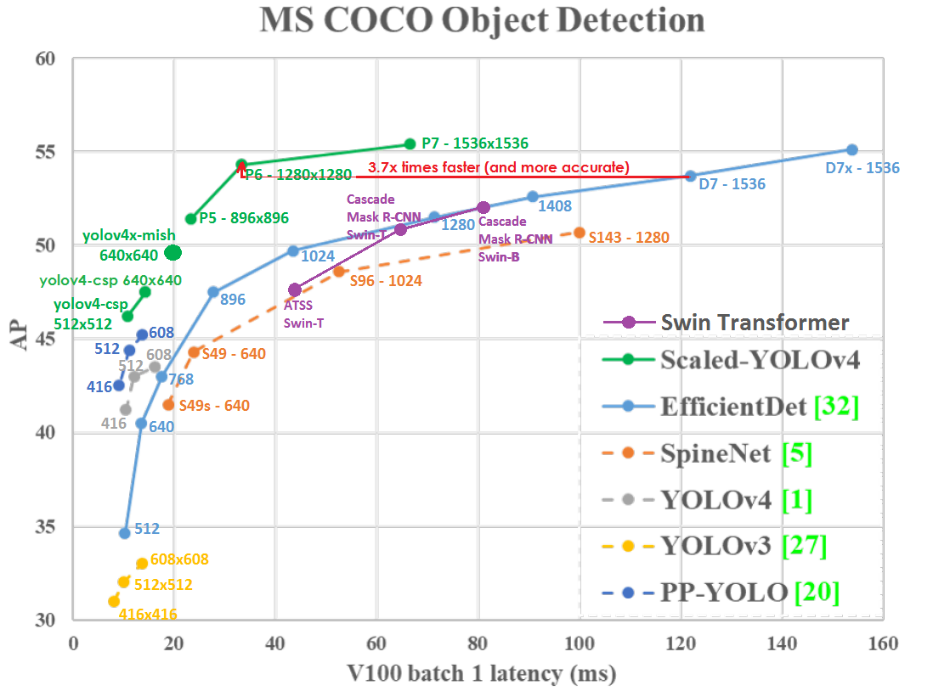


Figure : Comparison of performance of object detection models on MS COCO dataset [7]

The data format is the same as the format of YOLOv5 shown in table 1. The configuration of the model is defined in cfg file that contains the layer definitions and the following hyperparameters:

* Batch Size: 64
* Learning Rate: 0.00261

The model is trained using the darknet executable and the configuration files. It keeps training endlessly until the user is satisfied with the results and it saves the best weights to be used for inference.

## Deployment

To get a comprehensive look at how the models work on embedded systems, we set out to deploy them on two low-cost, popular platforms, the Raspberry Pi 4 and the Nvidia Jetson Nano. Both devices have similar CPU benchmarks but, the Nano has a dedicated GPU that increases its computational power when compared to the Pi.

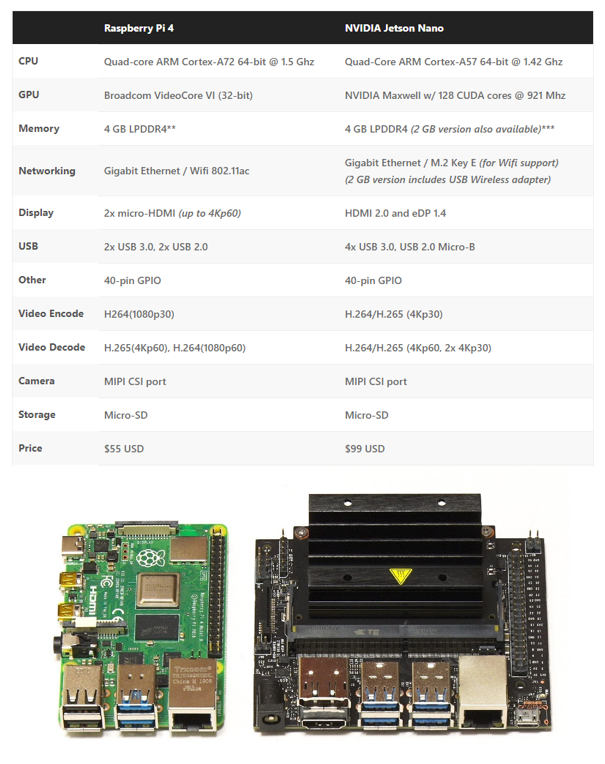


Figure : Raspberry Pi 4 vs Nvidia Jetson Nano specs [8]

### Raspberry Pi 4

For the Pi, only CPU-powered inference is available. Therefore, the models are to be deployed using the same APIs outlined above with OpenCV compiled for CPU inference.

### Nvidia Jetson Nano

Due to the presence of a GPU, there is more flexibility in how we can deploy the models on the Nano. Like the Pi, the models are deployed using the APIs outlined above except with a GPU compiled version of OpenCV. Moreover, a big advantage of the Jetson platform are the available options for optimization.

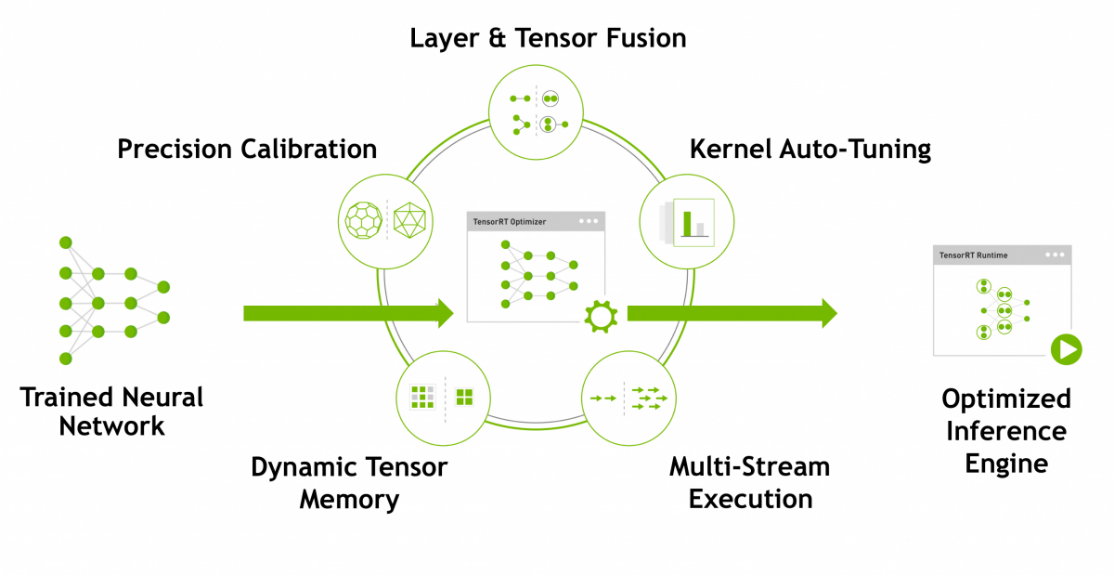


Figure : NVIDIA® TensorRT™ Optimizer Development Flow [9]

Nvidia provides an interface with the GPU called TensorRT that allows the definition of neural network layers in a more computationally efficient manner that can fully leverage the power of the Jetson platform. Furthermore, it provides a way to quantize the model to 16-bit floats which also improves inference time. Implementations of YOLOv4-tiny and YOLOv5s in TensorRT will be tested to analyze the performance improvement.

# Results and Discussions

In this section, we will discuss the performance of each model during training and inference. An important concept to define is Mean Average Precision (mAP) which is widely used in evaluating object detection models. Average Precision (AP) is the area under the precision-recall curve as shown in Figure 11.

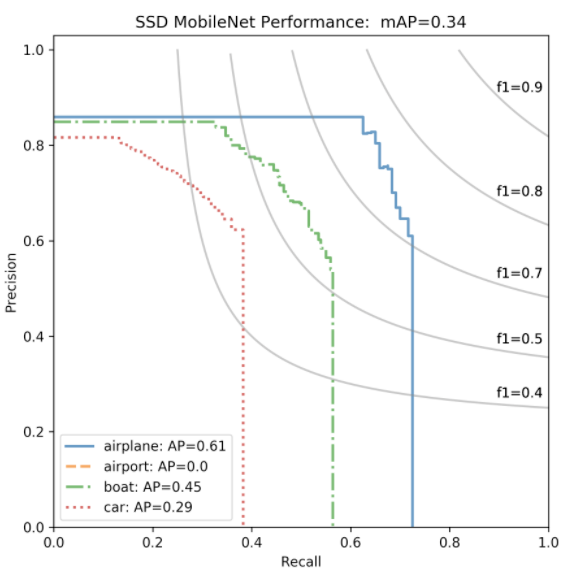


Figure : Precision-Recall curve example with AP [10]

AP is calculated for each class being detected and then mAP is calculated by taking the mean of the AP calculations. Precision and recall are defined by the following equations:

Where TP is true positive, FP is false positive, and FN is false negative. A positive in this context is defined as when the Intersection over Union (IoU) is greater than 0.5.

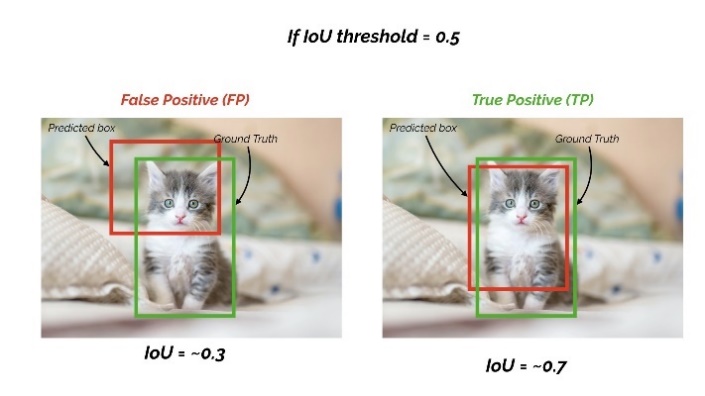
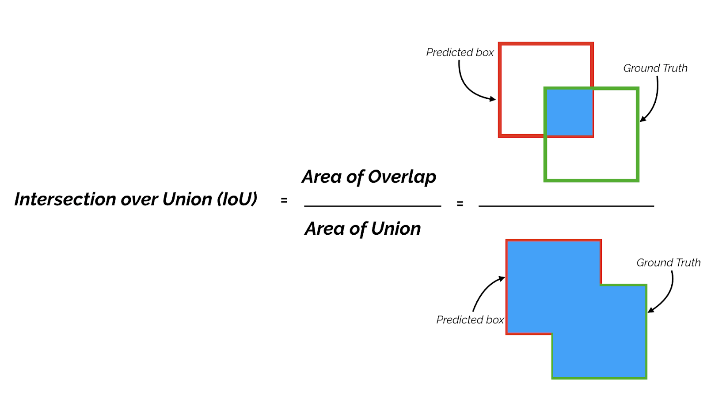


Figure : Intersection over Union [10]

A summary of the results is shown in Table 2.

Table : Performance Results of Models under Various Hardware

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Framework | Nvidia P100 [FPS] | Jetson Nano [FPS] | RPi 4 [FPS] | mAP | Precision | Recall |
| Faster R-CNN | PyTorch/ Detectron2 | 26.7 | 0.218 | DNR | 0.402 | - | - |
| YOLOv3-Tiny | darknet | 62.7 | 7.08 | 0.45 | 0.1498 | 0.44 | 0.34 |
| YOLOv4-Tiny | darknet | 64.9 | 14.1 | 0.31 | 0.3051 | 0.56 | 0.50 |
| YOLOv5s | PyTorch | 90.9 | 6.85 | 0.37 | 0.4904 | 0.7051 | 0.4515 |
| YOLOv4-Tiny TensorRT fp16 | TensorRT | - | 41.3 | - | 0.2539 | 0.453799 | 0.329336 |
| YOLOv4-Tiny TensorRT fp32 | TensorRT | - | 29.2 | - | 0.2539 | 0.453804 | 0.329479 |
| YOLOv5s TensorRT fp32 | TensorRT | - | 8.06 | - | - | - | - |
| YOLOv5s TensorRT fp16 | TensorRT | - | 11.9 | - | - | - | - |

## Faster R-CNN

As stated before, the model was trained using Facebook AI’s Detectron2. The model was trained until the total loss converged to a stable value which indicates that the training is complete. This took around 4 hours.

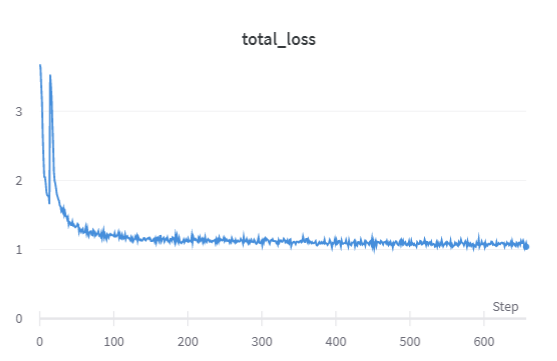


Figure : Faster R-CNN Loss over Epochs



Figure : Faster R-CNN Example Image Inference with the Generated Bounding Boxes

The AP is different for each class due to a difference in the frequency of their occurrence in the training and test datasets. They are shown in the figure below:

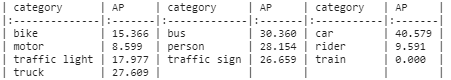


Figure : AP Results Per Classification of Faster R-CNN Example

Cars have the highest AP as they are the most represented in the training set and they are large objects in images. The mAP is 0.402 In terms of inference speed, it runs at 26.2 FPS on an Nvidia P100, 0.218 FPS on the Jetson Nano, and does not run on the Raspberry Pi 4. This speed is too low for embedded applications of machine vision therefore this model is not a good choice.

## YOLOv3-Tiny

Due to the use of an older API to train this model, there is no loss curve chart. However, the loss was monitored, and training was stopped when it converged. This took around 12 hours. This is a lightweight model that sacrifices accuracy for speed. An example of inference is shown in Figure 16.



Figure : YOLOv3-tiny Example Image Inference with the Generated Bounding Boxes

The mAP is 0.1498 and the AP for each class is shown in Figure 17.

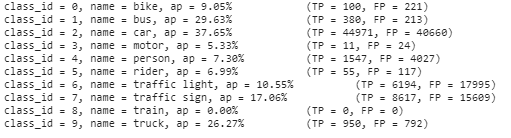


Figure : AP Results Per Classification of YOLOv3-tiny Example

In terms of inference speed, the model runs at 62.7 FPS on an Nvidia P100, 7.08 FPS on a Jetson Nano, and 0.45 FPS on a Raspberry Pi 4.

## YOLOv4-Tiny

Another lightweight model, YOLOv4-Tiny is the next iteration from YOLOv3-Tiny with improvements made to accuracy while still keeping inference time low. Training this model took around 11 hours. An example of inference is shown in Figure 18.



Figure : YOLOv4-tiny Example Image Inference with the Generated Bounding Boxes

The mAP converged to 0.3051 after 70000 iterations as shown in Figure 19.

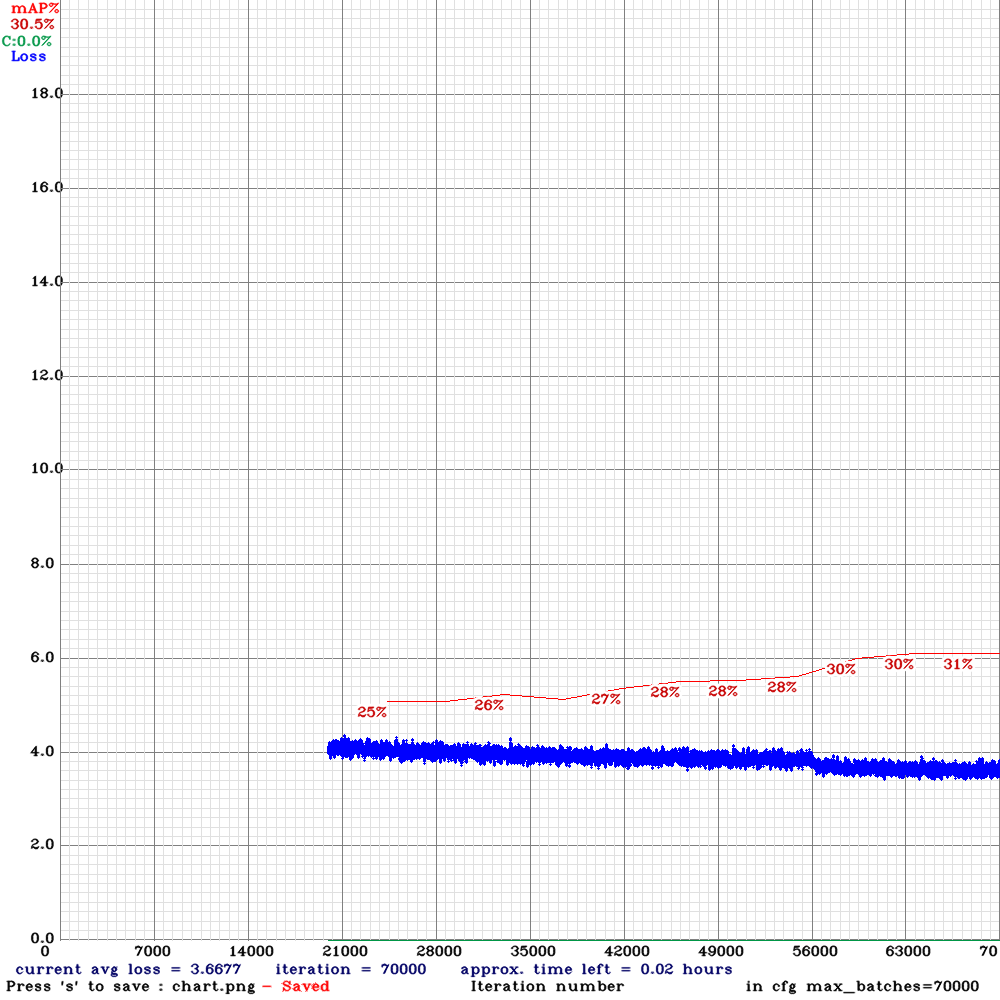


Figure : mAP Results of YOLOv4-tiny

The AP for each class is shown in Figure 20.

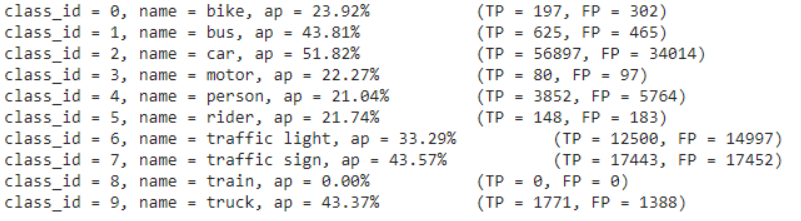


Figure : AP Results Per Classification of YOLOv4-tiny Example

The results continue the trend of car being the best identified class. In terms of inference speed, the model runs at 64.9 FPS on an Nvidia P100, 14.1 FPS on a Jetson Nano, and 0.31 FPS on a Raspberry Pi 4. It performs better than YOLOv3-Tiny on GPUs which is indicative of the improvements in the underlying structure of the model to take better advantage of GPUs during inference.

### YOLOv4-Tiny TensorRT fp32

For this model, the model weights and biases are still 32 bits but just by using the TensorRT layers, there are significant gains in inference time. The model runs at a much faster 29.2 FPS compared to the original 14.1 FPS, but there is a slight decrease in the mAP which is now 0.2539. An example of inference on an image is shown in Figure 21.

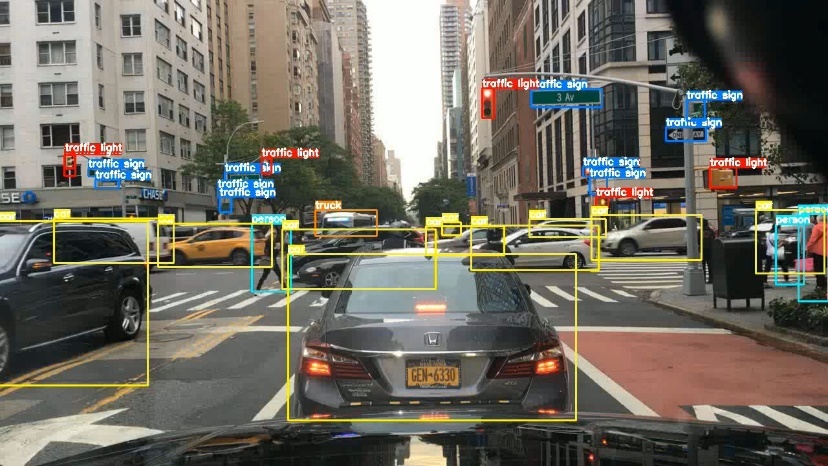


Figure : YOLOv4 fp32 Example Image Inference with the Generated Bounding Boxes

The metrics for each class are shown in Figure 22.

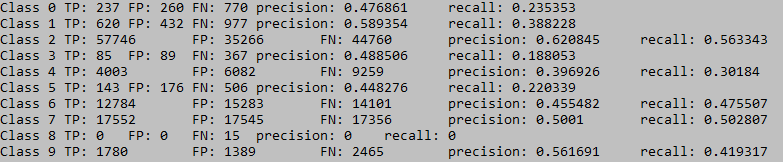


Figure : AP Results Per Classification for YOLOv4 fp32 Example

This model is very promising as 29.2 FPS can be used for real-time applications.

### YOLOv4-Tiny TensorRT fp16

To take it another step forward, the model weights and biases were quantized to 16-bit floats. This reduces the model size in half and should speed up inference. This assumption holds true as the model performs inference at an incredibly fast 41.3 FPS. There is no drop in mAP from fp32 to fp16 as it is 0.2539, but, overall, the fp16 model infers almost 300% faster than the unoptimized model. An example of inference on an image is shown in Figure 23.

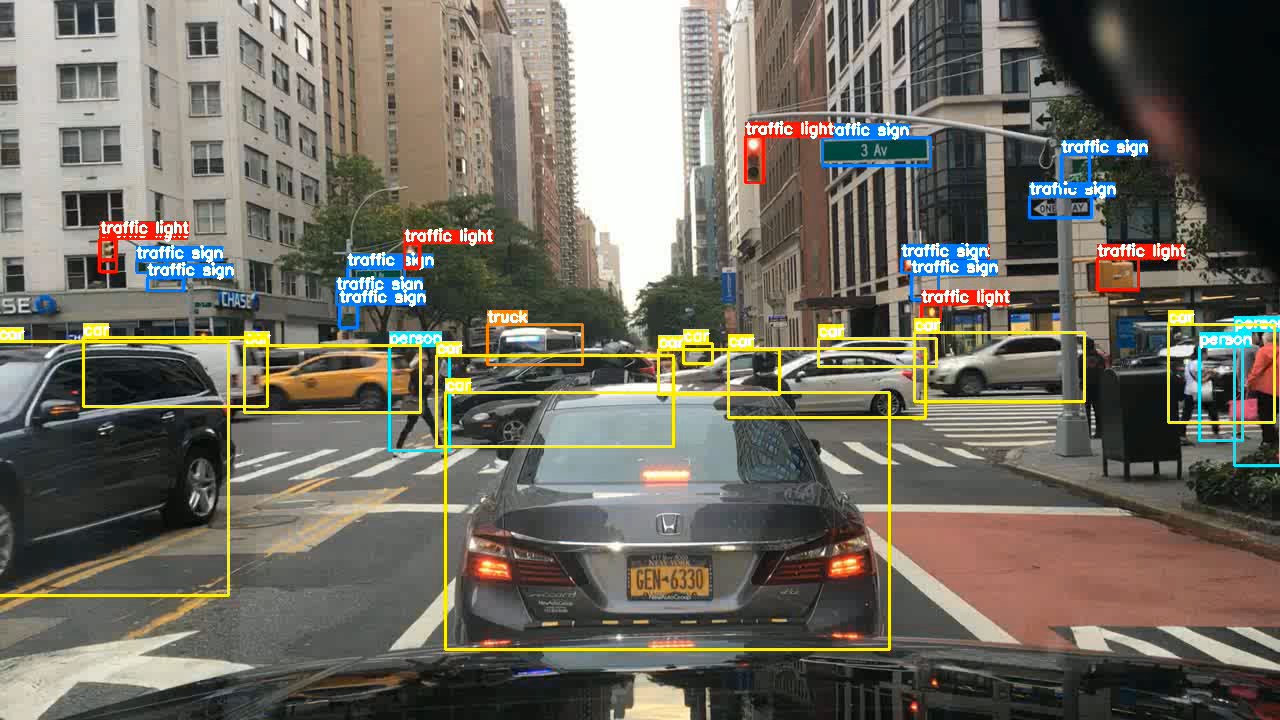


Figure : YOLOv4 fp16 Example Image Inference with the Generated Bounding Boxes

The metrics for each class are shown in Figure 22.

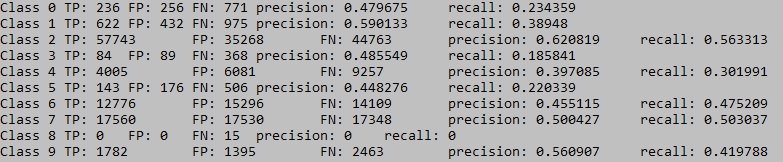


Figure : Figure 22: AP Results Per Classification for YOLOv4 fp16 Example

This model is the most promising in terms of inference speed as no other architecture or configuration comes close to this speed and this accuracy.

## YOLOv5s

This is the most up to date version of YOLO where the architecture is under active development to this day [6]. There are several models of YOLOv5 available on GitHub. Since we prioritized inference speed over mAP, we decided with the YOLOv5s model. This model took 9 hours to train. An example of the inference results is shown in Figure 25.



Figure : YOLOv5s Example Image Inference with the Generated Bounding Boxes

Evaluating the model performance, the resulting precision is 0.7051, and the recall is 0.4515. Comparatively with previous models, YOLOv5s boasts the highest mAP of 0.4904, a 60% improvement compared to other YOLO models.

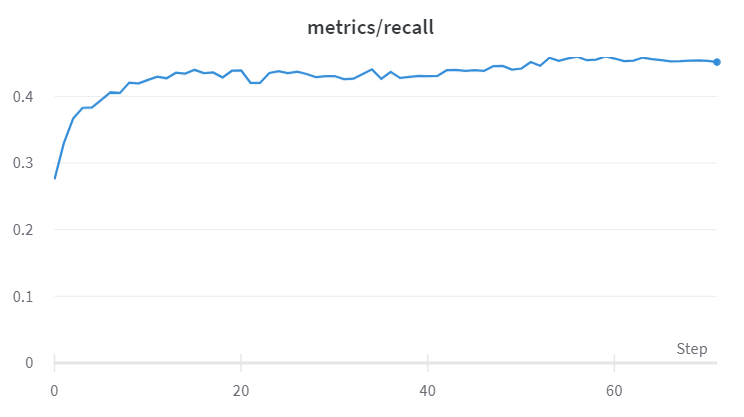
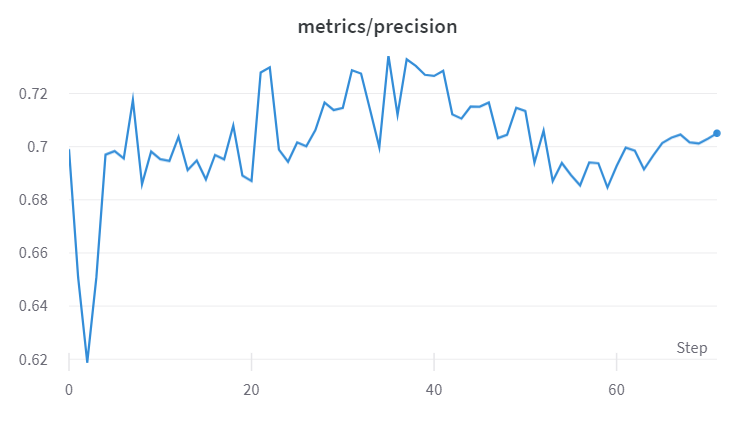


Figure : Precision and Recall Performance of YOLOv5s

In terms of speed, YOLOv5 ran the fastest on an NVIDIA P100 with 90.9 fps, and the slowest performance on the Jetson Nano of 6.85 fps amongst other YOLO versions. This model is much faster than the model with the most comparable mAP, Faster R-CNN. This further indicates the improvements of the YOLO structure every version in taking advantage of performance capabilities that advance GPUs can provide.

### YOLOv5-Tiny TensorRT

The YOLOv5s model was quantized to fp32 and fp16 with TensorRT implementation of the layers. The API does not provide a way to calculate the mAP therefore, due to time constraints, only the FPS was recorded for these models. The fp32 model performed the inference at 8.06 FPS and the fp16 model at 11.9 FPS which is a decent improvement from the original model. An example of inference is shown in Figure 27.



Figure : YOLOv5 TensorRT Example Image Inference with the Generated Bounding Boxes

It is interesting that the performance boost from using TensorRT is not as big for YOLOv5s as it was for YOLOv4-Tiny. This requires further investigation as to why that is the case.

## Comparison

Figure : Project Results

We have explored a variety of models on multiple platforms in order to generate a comprehensive view of detection of road objects. An initial point to be made is that the Raspberry Pi 4 platform is not suitable for real-time object detection. The lack of a powerful GPU limits inference time significantly with the highest being 0.45 FPS with YOLOv3-Tiny which is too slow for our objective. Thus, the comparison will be based mostly on the results from the Nvidia Jetson Nano. A chart of mAP vs FPS for the models investigated in this project is displayed on Figure 28.

There are a number of things that are surprising about these results. The Faster R-CNN was trained with the expectation that while it would be the slowest model, it would be the most accurate. However, it is second to YOLOv5s which is a testament to how much the YOLO model research has come. YOLOv5s achieves the highest mAP value by a wide margin while still having a respectable inference time. This indicates that for more powerful embedded devices such as the Jetson Xavier, it could be the optimal option in providing real-time, accurate object detection.

The YOLO models clearly outperform the R-CNN on inference time with a sacrifice to accuracy. Newer versions of YOLO performs better than its previous version in some way. For v4 and v5 of YOLO, the altered floating point significantly affects the performance. Boasting a similar accuracy, the processor has a lower

YOLOv4-Tiny is the best performer in terms of inference speed. Even greater so when the model is optimized, reaching 41.3 FPS which outperforms the other models by a very large margin. Overall, YOLOv4-Tiny TensorRT fp16 is the best model for our objective of real-time road object detection because it had the fastest inference time as well as acceptable accuracy.

# Conclusion

We set out to compare the state-of-the-art object detection models currently available on the Berkley DeepDrive dataset. Our metrics of comparison included the model accuracy but more importantly, the inference time. This is due to our objective of integrating this into an autonomous vehicle software stack which requires quick inference on resource limited embedded devices. We compared eight different model configurations spanning two devices, the Raspberry Pi 4, and the NVIDIA Jetson Nano.

Our results were fairly conclusive on a number of points. First, Raspberry Pi 4 is not well-suited for object detection applications due to its limited hardware power. On the other hand, for its price, a Jetson Nano provides incredible value for money in terms of the performance levels it achieved. At 2x the price of a Pi, it performed between 20-130 times better in inference time. YOLOv4-Tiny TensorRT fp16 is our selection for best model as it can run at 41.3 FPS with a respectable accuracy which proves that the Jetson Nano can work for real-time applications. Another interesting point we discovered is that YOLOv5 is ahead of the pack in terms of accuracy, even beating out the Faster R-CNN model. Further investigation on YOLOv5 would be valuable in order to obtain higher speeds on different versions of it like YOLOv5m or ones with TensorRT fp 16.

These results need to further be tested on our intended application, an autonomous vehicle, to determine what needs to be prioritized in the system as a whole. This the classic speed-accuracy trade-off. Our model of choice for speed is YOLOv4-Tiny TensorRT fp16, and our choice for accuracy is YOLOv5s.

In terms of future research, a more comprehensive examination of the effect of image size on inference time and mAP would provide a more complete picture. Moreover, there remains some models that were not tested due to time constraints such as the object detection model library in TensorFlow. An investigation into the performance of object detection while the device is performing other autonomous vehicle functionalities could also provide further insight on the limits of the Jetson Nano’s performance.

# References

[1] Formula Student Germany, Update Strategic Announcement, Oct 2019, https://www.formulastudent.de/pr/news/details/article/fsg-strategic-announcement/

[2] F. Yu, “BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning,” ETH Zürich, Apr 2020, https://bdd-data.berkeley.edu/

[3] R. Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms," towards data science, Jul 2018, https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[4] A. Sachan, “Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD,” 2019, https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

[5] Y. Wu, “Detectron2 Model Zoo and Baselines,” Apr 2021, https://github.com/facebookresearch/detectron2/blob/master/MODEL\_ZOO.md

[6] G. Jocher & co, “yolov5,” ultralytics, 2021, https://github.com/ultralytics/yolov5

[7] A. Bochkovskiy, “Scaled YOLO v4 is the best neural network for object detection on MS COCO dataset,” Dec 2020, https://alexeyab84.medium.com/scaled-yolo-v4-is-the-best-neural-network-for-object-detection-on-ms-coco-dataset-39dfa22fa982

[8] C. Pietschmann, “Raspberry Pi 4 vs NVIDIA Jetson Nano Developer Kit,” Jun 2019, https://build5nines.com/raspberry-pi-4-vs-nvidia-jetson-nano-developer-kit/

[9] NVIDIA Developer, “NVIDIA® TensorRT™,” 2021, https://developer.nvidia.com/tensorrt

[10] S. Yohanandan, “mAP (mean Average Precision) might confuse you!” Jun 2020, https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2