

Final Project Report

# Traffic Sign Detection

August 6<sup>th</sup>, 2023

## **GROUP 3**

Thien An Trinh  
Roman Burekhin  
Athira Devan  
Lester Azing

## Table of Contents

<b>1. INTRODUCTION .....</b>	<b>4</b>
<b>2. METHODOLOGY.....</b>	<b>5</b>
2.1. <i>Data</i> .....	5
2.1.1. Data Source.....	5
2.1.2. Data Preprocessing .....	6
2.1.3. Data Augmentation.....	6
2.2. <i>Models and Frameworks</i> .....	7
2.2.1. Single Shot MultiBox Detector (SSD) .....	7
2.2.2. Faster R-CNN.....	8
2.2.3. You Only Look Once (YOLO).....	8
2.2.4. Frameworks .....	9
2.3. <i>Training and Evaluation</i> .....	11
<b>3. RESULTS AND DISCUSSION .....</b>	<b>12</b>
3.1. <i>Mean Average Precision (mAP)</i> .....	12
3.2. <i>Speed</i> .....	14
3.3. <i>Size</i> .....	14
3.4. <i>Training Time</i> .....	15
3.5. <i>Model Selection Summary</i> .....	16
<b>4. INFERENCE AND DEMONSTRATION .....</b>	<b>16</b>
4.1. <i>Images</i> .....	16
4.1.1. Images From Validation Set.....	16
4.1.2. Images From the Internet.....	17
4.2. <i>Videos, Streamlit and Webcam</i> .....	19
<b>5. CONCLUSIONS .....</b>	<b>19</b>
<b>6. REFERENCES .....</b>	<b>19</b>

## List of Figures

Figure 1. Sample Images from the Dataset (obtained via (Road Sign Detection)) .....	5
Figure 2. Distribution of Object Classes in the Dataset .....	5
Figure 3. Data Augmentation Configuration for the TensorFlow Framework.....	6
Figure 4. Data Augmentation Configuration for the YOLOv5 Framework.....	7
Figure 5. Single Shot MultiBox Detection (image obtained from the origin paper (Liu et al., 2015)) .....	7
Figure 6. Faster R-CNN Mechanism (image obtained from the original paper (Ren et al., 2015)) .....	8
Figure 7. YOLO Mechanism (image obtained from the original paper (Redmon et al., 2015)).....	9
Figure 8. TensorFlow Model Zoo. The chosen models are highlighted in red boxes. ....	10
Figure 9. YOLOv5 GitHub Page .....	10
Figure 10. Snapshots of (a) Number of Training Steps and Batch Size, (b) Model Training at the last training step of TensorFlow Framework .....	11
Figure 11. Snapshots of (a) Number of Epochs and Batch Size, (b) Model Training at the last epoch of the YOLOv5 Framework .....	12
Figure 12. Snapshot of an Evaluation Results in the TensorFlow Framework.....	12
Figure 13. Snapshot of an Evaluation Results in the YOLOv5 Framework.....	12
Figure 14. mAP of CNN Models at IoU=0.5:0.95, evaluated on the Validation Set .....	13
Figure 15. mAP of CNN Models at IoU=0.5, evaluated on the Validation Set .....	13
Figure 16. Inference Time of CNN Models, evaluated on Validation Set .....	14
Figure 17. Size of CNN Models .....	15
Figure 18. Time per Training Step of CNN Models.....	15
Figure 19. Traffic Sign Detection by YOLOv5 on Images from the Validation Set .....	17
Figure 20. (a) Traffic Light, (b) Speed Limit, and (c) Stop Sign Detection by Faster-RCNN ResNet50 on Images from the Internet .....	18

## List of Tables

Table 1. Best Model Selections in each Criterion, based on Evaluation Results .....	16
---	----

## 1. INTRODUCTION

We are living in a world that is moving towards automation. From robot arms assembling individual components into complete cars to smart household appliances that have been transforming our homes, the benefits of autonomous applications are undeniable. The automobile industry is following the same trend. Not only autopilot systems assist drivers by bringing them better driving experience, but they can also help reduce the number of accidents. For example, a vehicle with a smart traffic sign detection system can “see” all the signs ahead including those the driver could miss, and thereby perform proper actions timely in time-sensitive situation.

In that context, our project focused two paramount objectives: first, to meticulously benchmark a range of model architectures to identify the most optimal and efficient solution for this challenging task, and second, to deploy and demonstrate precise and reliable detection of critical traffic signs.

Before proceeding into further detail, it is crucial to address certain concepts related to how the task was framed. First of all, it was determined that the project is a computer vision task – a domain in which images are processed and analyzed in order to extract useful information that can drive decision-making (Arabnia et al., 2018; Yoshida, 2011). Second, within the computer vision domain, this project is specifically an *object detection* task where an image was analyzed not for obtaining the semantic meaning of the whole image (i.e., *image classification*), or for segmenting the image into meaningful regions (i.e., *image segmentation*), but rather to identify targeted objects that are present in the images and determine where on the images they are located. In this project, the objects of interest were *traffic lights*, *stop signs*, *speed limit signs* and *crosswalk signs*, which are the fundamental elements that guide drivers and traffic flow. Third, the algorithms required for this task were defined to be deep learning (DL) convolutional neural networks (CNNs) which has always been the state-of-the-art in the domain for a decade. Furthermore, the project also leveraged an advanced DL technique called *transfer learning*, in which neural networks that were pretrained on a large dataset are fine-tuned on the dataset of interest instead of being trained from scratch, and therefore are capable of attaining high evaluation scores in the new domain. The details of the data, pretrained-models, training and evaluation frameworks are now discussed in the following section.

## 2. METHODOLOGY

### 2.1. Data

#### 2.1.1. Data Source

The data for this project was from Kaggle and is available at this reference (*Road Sign Detection*). The dataset comprises of two folders, each of which consists of 877 files. The first folder is named “*images*” which contains 877 road sign images in PNG, whereas the other is named “*annotations*” and has 877 corresponding XML files that store the image annotations in the PASCAL VOC format (Everingham et al., 2015). The images belong to 4 distinct classes, namely *traffic light*, *stop*, *speed limit*, and *crosswalk* which are the target objects to detect (Figure 1, Figure 2).



Figure 1. Sample Images from the Dataset (obtained via (Road Sign Detection))

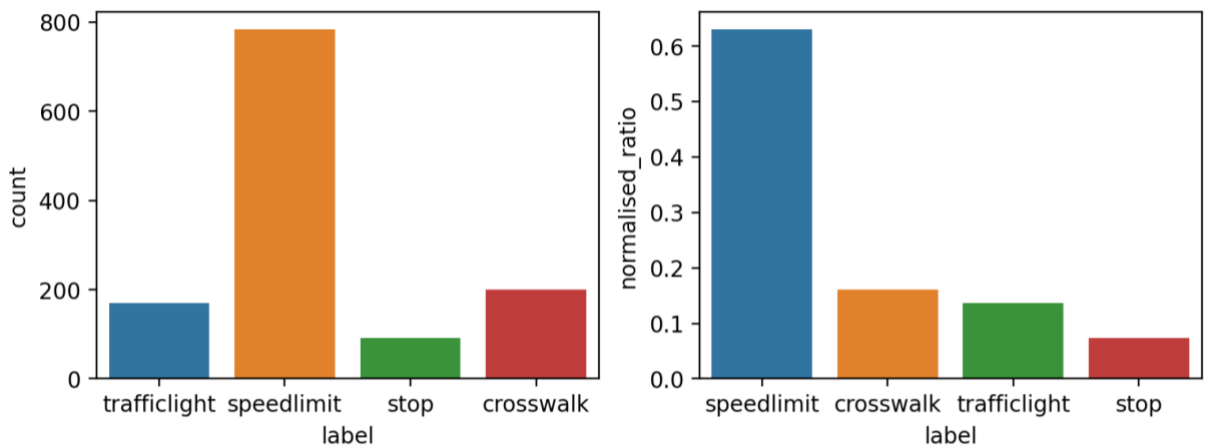


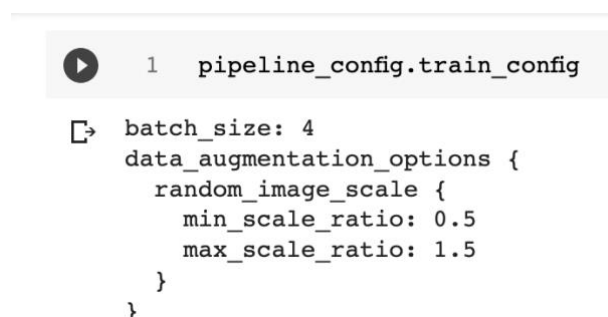
Figure 2. Distribution of Object Classes in the Dataset

### 2.1.2. Data Preprocessing

The data was split into a training and a validation set with the ratio of 4:1. As a result, the training set contained 701 images and the validation set had 176. Additionally, since the original format of the annotations were not ready to be used in the training pipeline, label format conversion was performed. Since the models used in the experiments required two different training frameworks, there were two types of conversion. The first framework was the TensorFlow object detection API which required the annotations in TFRecord format, while the Ultralytics framework for YOLOv5 expected the annotations in text format with each line describing the bounding boxes of the objects. More details of the frameworks are discussed in Section 2.2. It is notable that the training and validation sets had separate annotation conversion, so that resulting files were distinctly addressed in the training configuration to be the training and validation annotations respectively. Further details of the data preprocessing steps can be found in the Jupyter notebooks.

### 2.1.3. Data Augmentation

Data augmentation is a technique that helps diversify the training data by adding variations to the images. For fair benchmarking and evaluation, similar augmentations had to be used in both frameworks (e.g., if the TensorFlow models used shearing at 30°, YOLOv5 had to use the same transformation), meaning that we could only choose the augmentation options that were available in both frameworks. Additionally, in the context of traffic sign detection, some transformations would not be applicable such as vertical or horizontal flips, or 90° rotation. Due to these requirements, there were a limited number of options available that could be utilized. In this project, we could select only one augmentation option that fitted the situation, and it was image scaling (Figure 3, Figure 4). Nevertheless, this single augmentation proved to be effective as described in detail in a later section of this report.



```
1 pipeline_config.train_config
  batch_size: 4
  data_augmentation_options {
    random_image_scale {
      min_scale_ratio: 0.5
      max_scale_ratio: 1.5
    }
  }
```

Figure 3. Data Augmentation Configuration for the TensorFlow Framework

```

hyp.scale-augmentation.yaml ×
21 perspective: 0.0
22 scale: 0.5
23 shear: 0
24 translate: 0

```

Figure 4. Data Augmentation Configuration for the YOLOv5 Framework

## 2.2. Models and Frameworks

As the objective of this project was to develop a working app, it was important to diversify the models in the experiment phase in order to discover the optimal one for deployment. Considering the CNNs in the field of object detection, these following architectures stood out as the state-of-the-art.

### 2.2.1. Single Shot MultiBox Detector (SSD)

The Single Shot MultiBox Detector was firstly introduced in 2015 (Liu et al., 2015). This CNN architecture employs a set of default boxes of various aspect ratios and scales which, upon inference, can be adjusted to match the object shapes for better detection. Moreover, to cope with objects of different sizes, it leverages the fact that different feature maps have different resolutions, and therefore uses multiple predictions from these layers to compute the final detections. Figure 5 describes the working mechanism of an SSD network and is taken from the original paper. Note that in Figure 5(a) there are two objects of different shapes and sizes, of which smaller object can be better detected with the blue default box on a higher resolution feature map (Figure 5(b)), while the location of the larger object would be more efficiently detected by the red default box on a lower resolution one (Figure 5(c)).

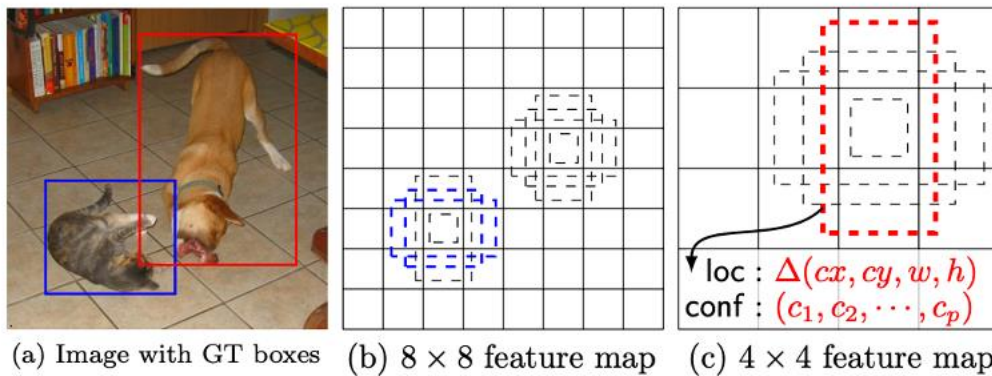


Figure 5. Single Shot MultiBox Detection (image obtained from the origin paper (Liu et al., 2015))

### 2.2.2. Faster R-CNN

A competitor of the SSD family is the Faster Region-based Convolutional Neural Networks (Faster R-CNN), which was introduced also in 2015 (Ren et al., 2015). Unlike SSD models, R-CNNs are two-stage object detection algorithms that (1) produce region proposals that suggest the location of the objects based on which (2) boundary boxes and objectness scores are predicted. In Faster R-CNN architecture, the author introduced a Region Proposal Network (RPN) that address the model's slow runtime issue which had been the bottleneck of any two-stage object detection networks.

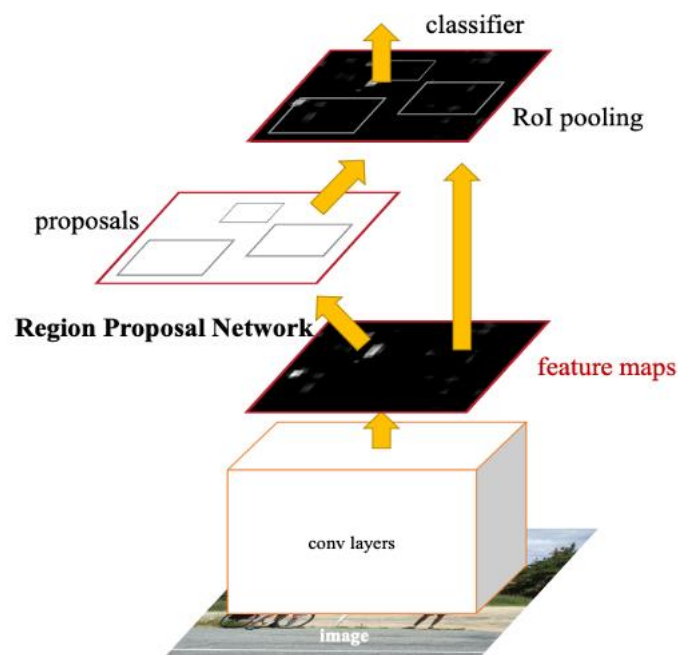


Figure 6. Faster R-CNN Mechanism (image obtained from the original paper (Ren et al., 2015))

### 2.2.3. You Only Look Once (YOLO)

You Only Look Once (YOLO) was another framework that was introduced in 2015 (Redmon et al., 2015) whose one of the prominent features is that it is extremely fast. In YOLO architecture, an image is divided into an  $S \times S$  grid, and on each grid cell the model predicts  $B$  bounding boxes (where each bounding box consists of 5 values,  $x$ ,  $y$ ,  $w$ ,  $h$  and confidence) and the object class. With this mechanism the model sees the whole image once and therefore it is a single-stage object detection algorithm. Since the introduction of the original architecture, there has been much evolution of the YOLO family, resulting in YOLOv2 (Redmon & Farhadi, 2016), YOLOv3 (Redmon & Farhadi, 2018), YOLOv4 (Bochkovskiy et al., 2020), YOLOv5 (Ultralytics), YOLOv6 (Li et al., 2022), YOLOv7 (Wang et al., 2022), and most recently YOLOv8 (Ultralytics). This study used YOLOv5 for benchmarking and deployment.



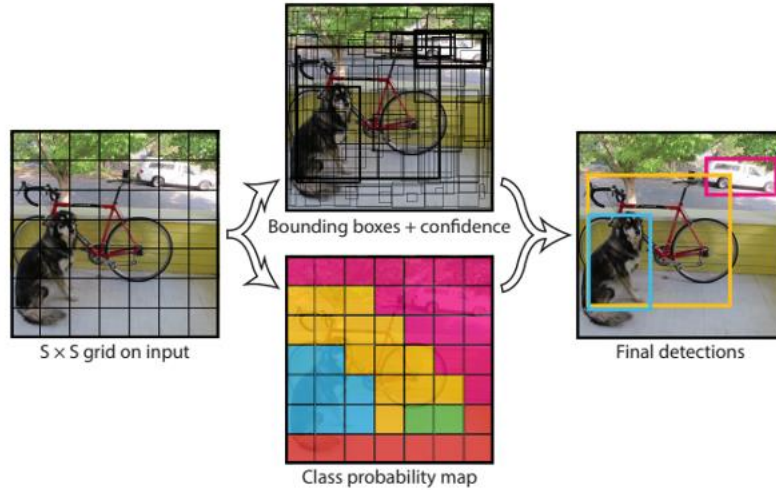


Figure 7. YOLO Mechanism (image obtained from the original paper (Redmon et al., 2015))

#### 2.2.4. Frameworks

To achieve high precision, the models were not built and trained from scratch but instead via transfer learning. Specifically, the models were downloaded with pretrained weights and thereby served as a starting point for fine-tuning. The pretrained models were obtained from two sources: the SSD and Faster R-CNN models were acquired from the TensorFlow2 Detection Model Zoo (TensorFlow) (Figure 8), while the pretrained YOLOv5 was obtained by cloning the model's repository on GitHub (Ultralytics) (Figure 9).

On Figure 8, highlighted in red boxes are the chosen models from Model Zoo. They were, namely, *EfficientDet D1*, *SSD MobileNet FPNLite*, *SSD ResNet50 FPN (RetinaNet50)*, and *Faster R-CNN ResNet50*. There are some points worth highlighting. First, all of the chosen models had similar training resolution of  $640 \times 640$ , and so does YOLOv5, to ensure fair benchmarking. Second, there were SSD and Faster R-CNN models of similar ResNet50 backbone so SSD can be compared against Faster R-CNN in this task. Third, since the project focused on not only experimenting but also deployment, the model list was extended to included SSD MobileNet and EfficientDet. On the one hand, MobileNet is a CNN developed by Google which were claimed to be suitable on mobile and embedded device (Howard et al., 2017). On the other hand, EfficientDet is an object detection network family which utilizes compound scaling method and weighted bi-directional feature pyramid network (BiFPN) to optimize model efficiency (Tan et al., 2019). Since these two models were claimed to be efficient in terms of speed and size, they were considered suitable candidates for deployment and hence were involved in this project.

models / research / object\_detection / g3doc / tf2\_detection\_zoo.md

Preview Code Blame 70 Lines (62 loc) · 10.3 KB

EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes
EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes
EfficientDet D7 1536x1536	325	51.2	Boxes
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3	Boxes
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38.3	Boxes
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29.3	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31.0	Boxes

Figure 8. TensorFlow Model Zoo. The chosen models are highlighted in red boxes.

ultralytics / yolov5

Code Issues 173 Pull requests 69 Discussions Actions Projects 1 Wiki Security Insights

yolov5 Public Sponsor Watch 353 Fork 14.3k Star 40.4k

master 15 branches 10 tags

Go to file Add file Code

glenn-jocher Update Discord invite to <https://ultralytics.com/discord> (#11894) ✓ acdf73b 3 days ago 2,680 commits

.github	Update Discord invite to <a href="https://ultralytics.com/discord">https://ultralytics.com/discord</a> (#11894)	3 days ago
classify	Update <code>check_requirements()</code> ROOT (#11557)	2 months ago
data	remove objects with iscrowd=True in Objects365 (#11788)	3 weeks ago
models	Update setup.cfg (#11814)	3 weeks ago
segment	Update setup.cfg (#11814)	3 weeks ago
utils	Comet updates (#11818)	3 weeks ago
.dockerignore	Add .git to .dockerignore (#8815)	last year
.gitattributes	git attrib	3 years ago
.gitignore	Ignore *_paddle_model/ dir (#10745)	6 months ago
.pre-commit-config.yaml	Update setup.cfg (#11814)	3 weeks ago
CITATION.cff	Update LICENSE to AGPL-3.0 (#11359)	3 months ago
CONTRIBUTING.md	Add links to <a href="https://docs.ultralytics.com/help/">https://docs.ultralytics.com/help/</a> (#11462)	3 months ago

About

YOLOv5 in PyTorch > ONNX > CoreML > TFLite

[docs.ultralytics.com](https://docs.ultralytics.com)

ios machine-learning deep-learning ml pytorch yolo object-detection coreml onnx tflite yolov3 yolov5 ultralytics

Readme AGPL-3.0 license Code of conduct Security policy Cite this repository Activity 40.4k stars 353 watching 14.3k forks Report repository

Figure 9. YOLOv5 GitHub Page

## 2.3. Training and Evaluation

After the dataset and the models had been defined, model training was performed. Since there was no GPU available in any of the team members' computers, the models were trained on Google Colab notebooks to utilize the free-of-charge GPUs that were provided by the service. To fit the training data to the GPUs, the training batch size was set to be 4, and in the TensorFlow framework the models were trained for 5000 training steps (Figure 10). The YOLOv5 framework required number of epochs instead of training steps to be specified, therefore, with the batch size of 4 and the number of training instances equal to 701, a calculation was done to convert 5000 training steps to the equivalent value of 29 epochs (Figure 11).

(a)

```
1 # Set number of training steps
2
3 TRAINING_STEPS=5000
```

1 pipeline\_config.train\_config

batch\_size: 4

(b)

```
INFO:tensorflow:Step 5000 per-step time 0.530s
I0724 04:28:59.949985 139038888473216 model_lib_v2.py:705] Step 5000 per-step time 0.530s
INFO:tensorflow:{'Loss/classification_loss': 0.06399711,
'Loss/localization_loss': 0.0009799192,
'Loss/regularization_loss': 0.03543603,
'Loss/total_loss': 0.10041306,
'learning_rate': 0.07998606}
```

Figure 10. Snapshots of (a) Number of Training Steps and Batch Size, (b) Model Training at the last training step of TensorFlow Framework

(a)

```
1 # 3. Train params
2
3 epochs = 29
4 !python train.py --img 640 --batch 4 --epochs {epochs}
```

(b)

Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
28/28	1.09G	0.01414	0.00317	0.001363	2	640: 100% 176/176 [00:18<00:00, 9.28it/s]
	Class	Images	Instances	P	R	mAP50 mAP50-95: 100% 22/22 [00:02<00:00, 8.62it/s]
	all	176	250	0.944	0.896	0.92 0.741

Figure 11. Snapshots of (a) Number of Epochs and Batch Size, (b) Model Training at the last epoch of the YOLOv5 Framework

Following model training was the model evaluation step which generated the precision scores as shown on Figure 12 and Figure 13. These precision values, together with other evaluation metrics of interest such as inference speed, model size and training time were collected and comprehensively presented in Section 3 of this report.

Accumulating evaluation results...  
DONE (t=0.27s).

Average Precision	(AP) @[ IoU=0.50:0.95   area= all   maxDets=100 ]	= 0.600
Average Precision	(AP) @[ IoU=0.50   area= all   maxDets=100 ]	= 0.878
Average Precision	(AP) @[ IoU=0.75   area= all   maxDets=100 ]	= 0.735
Average Precision	(AP) @[ IoU=0.50:0.95   area= small   maxDets=100 ]	= 0.433
Average Precision	(AP) @[ IoU=0.50:0.95   area=medium   maxDets=100 ]	= 0.669
Average Precision	(AP) @[ IoU=0.50:0.95   area= large   maxDets=100 ]	= 0.729
Average Recall	(AR) @[ IoU=0.50:0.95   area= all   maxDets= 1 ]	= 0.584
Average Recall	(AR) @[ IoU=0.50:0.95   area= all   maxDets= 10 ]	= 0.642
Average Recall	(AR) @[ IoU=0.50:0.95   area= all   maxDets=100 ]	= 0.651
Average Recall	(AR) @[ IoU=0.50:0.95   area= small   maxDets=100 ]	= 0.472
Average Recall	(AR) @[ IoU=0.50:0.95   area=medium   maxDets=100 ]	= 0.713
Average Recall	(AR) @[ IoU=0.50:0.95   area= large   maxDets=100 ]	= 0.746

Figure 12. Snapshot of an Evaluation Results in the TensorFlow Framework

Validating runs/train/exp/weights/best.pt...  
Fusing layers...  
Model summary: 157 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs

Class	Images	Instances	P	R	mAP50	mAP50-95
all	176	250	0.956	0.871	0.916	0.737
stop	176	15	0.918	1	0.995	0.895
crosswalk	176	45	0.925	0.827	0.875	0.669
speedlimit	176	163	0.98	1	0.995	0.878
trafficlight	176	27	1	0.658	0.8	0.505

Results saved to runs/train/exp

Figure 13. Snapshot of an Evaluation Results in the YOLOv5 Framework

### 3. RESULTS AND DISCUSSION

#### 3.1. Mean Average Precision (mAP)

Figure 14 and Figure 15 show the mAPs of the models evaluated on the validation set at IoU=0.5:0.95 and IoU=0.5, respectively. There are three points worth highlighting. First, YOLOv5 achieved the highest scores among all the models, with mAP $\approx$ 0.75 at IoU=0.5:0.95 and 0.92 at IoU=0.5, while between the TensorFlow models, Faster R-CNN ResNet50 attained the highest precisions with SSD MobileNet FPNLite closely followed. In contrast, SSD ResNet50 FPN achieved the lowest precisions scores on the validation set with mAP values lower than 0.5 at IoU=0.5:0.95. Second, except for EfficientDet D1, augmentation improved the models' precisions thanks to the enhanced diversification of the training images. Lastly, our mAP results do not follow the trend reported on the Model Zoo where EfficientDet D1 scored the best, followed by the SSD ResNet50 FPN. This proves that different models performed differently on

different datasets and therefore, selecting an optimal model for a specific use case requires thorough experimenting and benchmarking procedures.

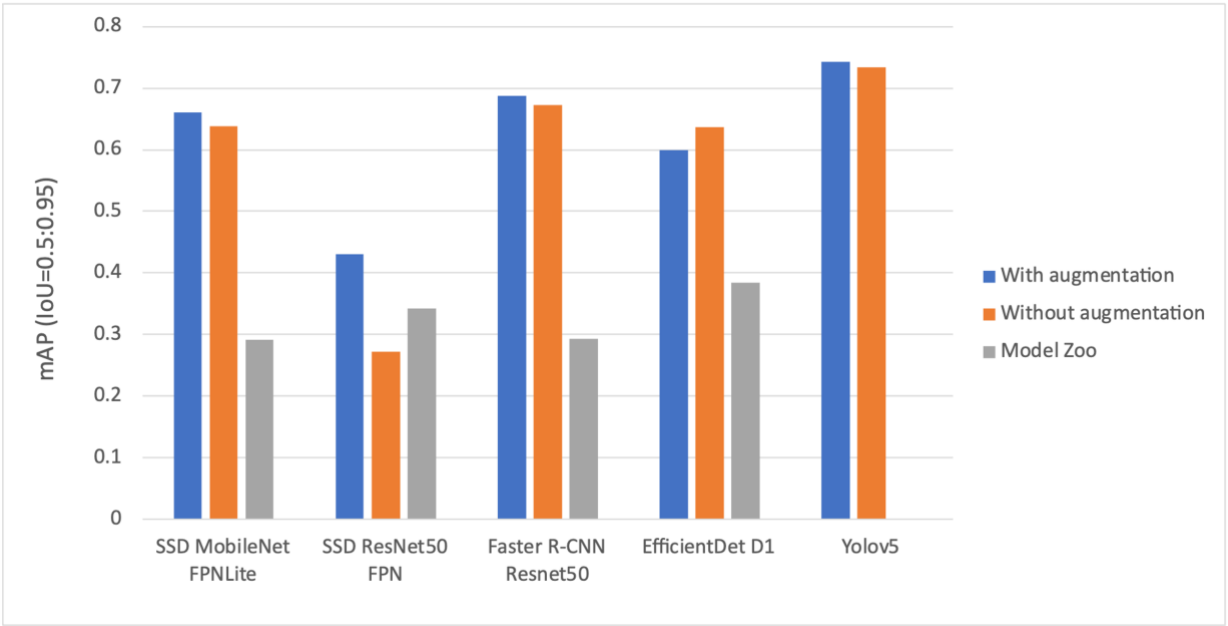


Figure 14. mAP of CNN Models at IoU=0.5:0.95, evaluated on the Validation Set

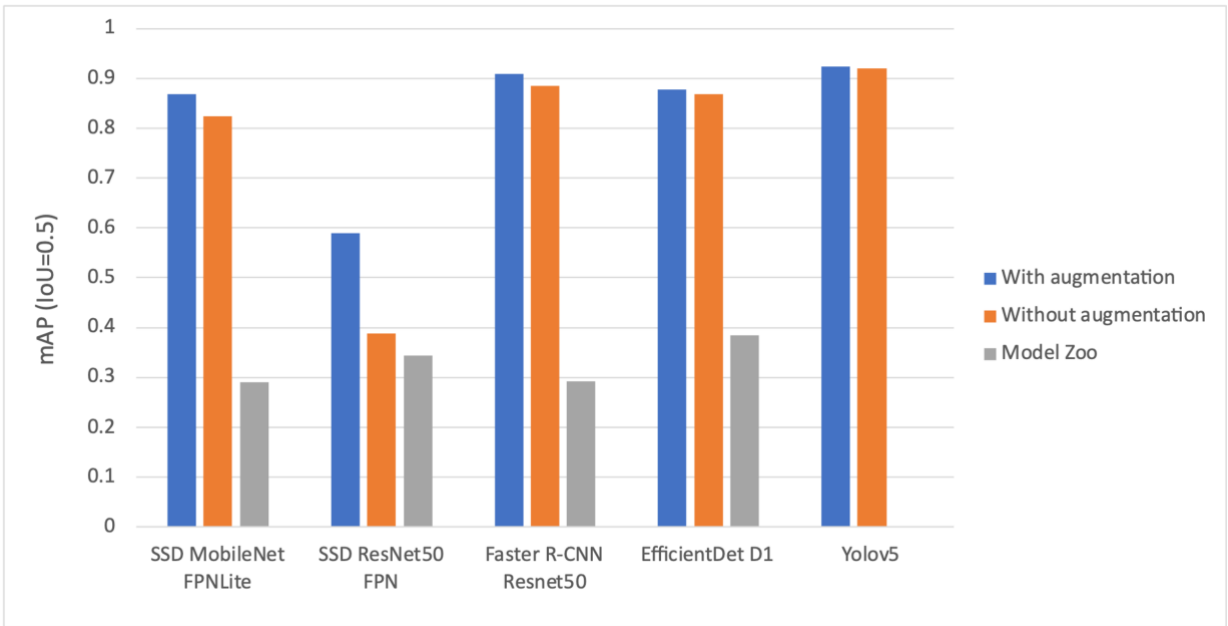


Figure 15. mAP of CNN Models at IoU=0.5, evaluated on the Validation Set

### 3.2. Speed

Besides precision, another crucial factor used to evaluate the applicability of a model is speed. Figure 16 displays the models' average inference time when testing on several test images. Faster R-CNN ResNet50, although was the model achieved the highest precision scores, took the longest time when detecting traffic signs on the test images, making it less preferable for real-time applications. On the other hand, SSD MobileNet FPNLite and EfficientDet D1, while being slightly lower in terms of precisions compared to Faster R-CNN ResNet50, respectively took only half and one-third of the time of Faster R-CNN ResNet50 to run. These results could be attributed to the fact that SSD models perform detection in a single stage, while Faster R-CNNs are two-stage object detectors. However, SSD models were not the fastest in our study: among all the models, YOLOv5 again stands out as the model with the fastest inference speed.

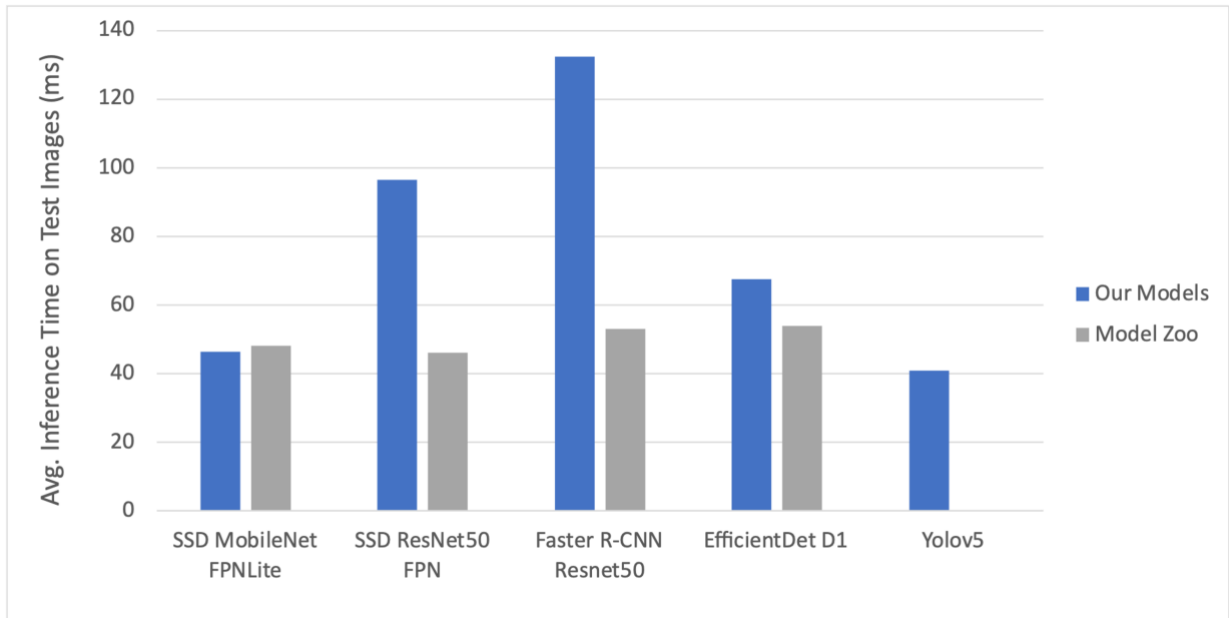


Figure 16. Inference Time of CNN Models, evaluated on Validation Set

### 3.3. Size

On edge devices, a suitable model can be slightly less precise so long as the sacrifice of precision allows for a much lighter weight. In terms of model package size, Figure 17 indicates that YOLOv5 and SSD MobileNet FPNLite are the smallest models, followed by EfficientDet D1, while SSD ResNet50 FPN and Faster R-CNN ResNet50 occupy the largest storage amounts. Here one would see that YOLOv5, SSD MobileNet FPNLite and EfficientDet D1 were not only optimized for speed (Figure 16), but also for size (Figure 17), which renders them suitable options for deployment on edge devices. It is worth highlighting that YOLOv5 was exported and stored in a format different from the TensorFlow models. Among all the models, YOLOv5 continued being the best in terms of size, whereas between the TensorFlow CNNs, SSD MobileNet FPNLite is the lightest model as its name suggests.

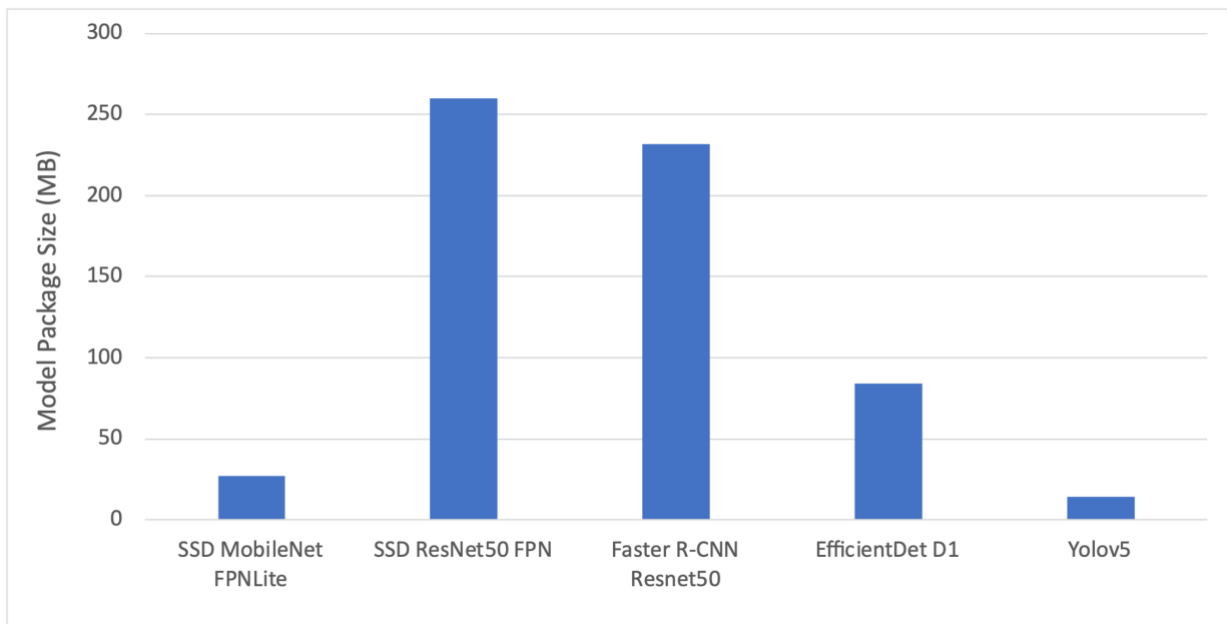


Figure 17. Size of CNN Models

### 3.4. Training Time

A process of selecting a model includes not only inference but also training process, and the time a model would take for training is a metric indicating the training cost (Figure 18). One more time YOLOv5 beat the TensorFlow models by being the fastest model to train ( $\approx 0.1$ - $0.13$ s per training step). Among the TensorFlow models, in terms of model training SSD MobileNet FPNLite again appeared to be a good candidate, while EfficientDet D1 seemed to be no longer a preferable option. It is also notable that while data augmentation enhanced precision (Figure 14 and Figure 15), it also increased the training time (Figure 18) due to the introduction of an augmentation step in the training pipeline.

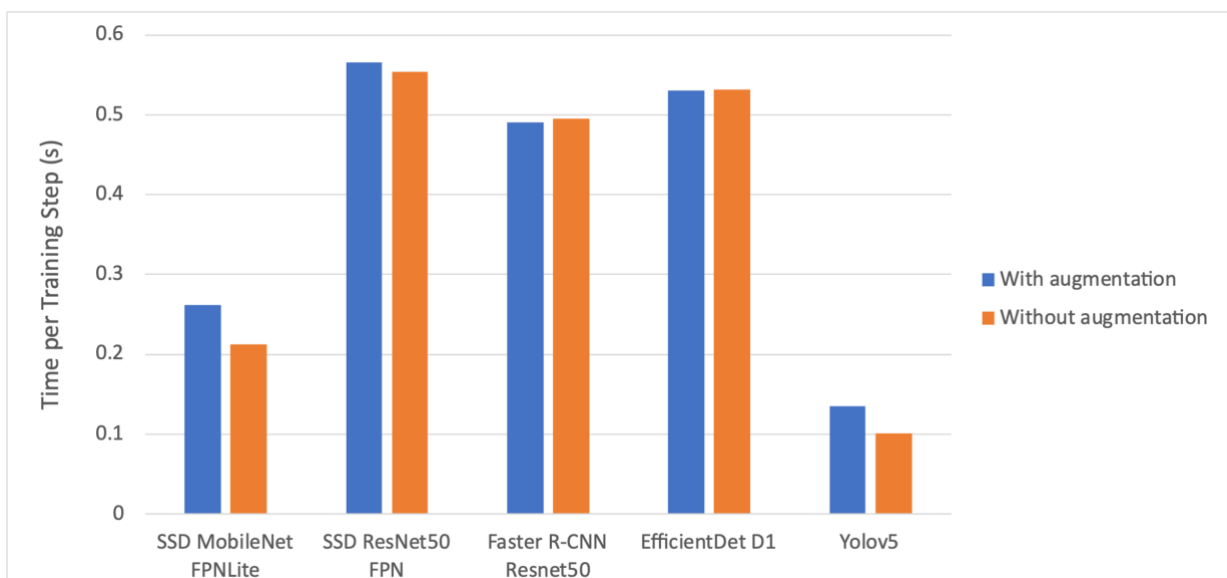


Figure 18. Time per Training Step of CNN Models

### 3.5. Model Selection Summary

Based on the evaluation results, Table 1 summarizes the best model for each criterion, with a double tick mark indicating the best model and a single tick mark indicating the second best, where applicable.

*Table 1. Best Model Selections in each Criterion, based on Evaluation Results*

	SSD MobileNet FPNLite	SSD ResNet50 FPN	Faster R- CNN ResNet50	EfficientDet D1	YOLOv5
Precision			✓		✓✓
Speed	✓				✓✓
Size	✓				✓✓
Training Time	✓				✓✓

YOLOv5 won in all criteria, while SSD MobileNet FPNLite led in three out of four metrics among the TensorFlow family. In terms of precision, the previous section has pointed out that the difference between SSD MobileNet FPNLite and Faster R-CNN ResNet50 was indeed insignificant. If selecting only from TensorFlow models, the SSD MobileNet is a most suitable candidate.

## 4. INFERENCE AND DEMONSTRATION

### 4.1. Images

#### 4.1.1. Images From Validation Set





Figure 19. Traffic Sign Detection by YOLOv5 on Images from the Validation Set

#### 4.1.2. Images From the Internet

(a)



(b)



(c)



Figure 20. (a) Traffic Light, (b) Speed Limit, and (c) Stop Sign Detection by Faster-RCNN ResNet50 on Images from the Internet

## 4.2. Videos, Streamlit and Webcam

The model inference on a video and webcam demo recording are provided in the project GitHub repository.

The Streamlit app is available here: <https://trafficsigns.streamlit.app/>

## 5. CONCLUSIONS

This report describes in detail our process of developing traffic sign detection application, from conceptualization to model training, and to deployment. Our results showed that YOLOv5 stood out to be the best model in all criteria, while SSD MobileNet FPNLite was the winner among the TensorFlow options. Our demonstrations also visualized how the trained object detection models worked on images, videos, and in real-time via webcam, and how it worked on Streamlit.

We hope that our work has provided some useful insights into the field, and paved the way for a safer, more efficient and autonomous driving experience. As we proceed to extend this project in future work, we wish to continue to help reshape the landscape of mobility for the better, and impact countless lives on the road.

## 6. REFERENCES

- Arabnia, H. R., Deligiannidis, L., & Tinetti, F. G. (2018). *Image Processing, Computer Vision, and Pattern Recognition*. C.S.R.E.A.
- Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934. Retrieved April 01, 2020, from <https://ui.adsabs.harvard.edu/abs/2020arXiv200410934B>
- Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2015). The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1), 98-136. <https://doi.org/10.1007/s11263-014-0733-5>
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861. Retrieved April 01, 2017, from <https://ui.adsabs.harvard.edu/abs/2017arXiv170404861H>
- Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., Li, Y., Zhang, B., Liang, Y., Zhou, L., Xu, X., Chu, X., Wei, X., & Wei, X. (2022). YOLOv6: A Single-Stage

Object Detection Framework for Industrial Applications. arXiv:2209.02976. Retrieved September 01, 2022, from <https://ui.adsabs.harvard.edu/abs/2022arXiv220902976L>

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2015). SSD: Single Shot MultiBox Detector. arXiv:1512.02325. Retrieved December 01, 2015, from <https://ui.adsabs.harvard.edu/abs/2015arXiv151202325L>

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640. Retrieved June 01, 2015, from <https://ui.adsabs.harvard.edu/abs/2015arXiv150602640R>

Redmon, J., & Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger. arXiv:1612.08242. Retrieved December 01, 2016, from <https://ui.adsabs.harvard.edu/abs/2016arXiv161208242R>

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv:1804.02767. Retrieved April 01, 2018, from <https://ui.adsabs.harvard.edu/abs/2018arXiv180402767R>

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497. Retrieved June 01, 2015, from <https://ui.adsabs.harvard.edu/abs/2015arXiv150601497R>

*Road Sign Detection*. <https://www.kaggle.com/datasets/andrewmvd/road-sign-detection>

Tan, M., Pang, R., & Le, Q. V. (2019). EfficientDet: Scalable and Efficient Object Detection. arXiv:1911.09070. Retrieved November 01, 2019, from <https://ui.adsabs.harvard.edu/abs/2019arXiv191109070T>

TensorFlow. [https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/tf2\\_detection\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md)

Ultralytics. YOLOv5. <https://ultralytics.com/yolov5>

Ultralytics. YOLOv8. <https://ultralytics.com/yolov8>

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv:2207.02696. Retrieved July 01, 2022, from <https://ui.adsabs.harvard.edu/abs/2022arXiv220702696W>

Yoshida, S. R. (2011). *Computer vision*. Nova Science Publishers, Inc.