U.S. Traffic Sign Object Detection using TensorFlow 2 with Transfer Learning

James Edward Starks III

***Abstract*—This project aims to use TensorFlow 2 and Transfer Learning to localize and classify traffic signs from an image stream directed forward from the front of a vehicle. The goal is to achieve a high recall and precision score by leveraging existing deep learning object detection models. The traffic signs targeted for this project are: stop, speed limit, do not enter, and yellow railroad crossing signs.**

I. Introduction

Vision-based object detection systems have become a major part of modern cars' Advanced Driver-Assistance Systems (ADAS). They are tools used to help the driver reach their destination safely. ADAS features include adaptive cruise control, lane-keep assistance, front-end collision avoidance, blind-spot alert, object detection, etc. The current estimate is about 10% of cars worldwide utilize ADAS features, and it is projected to increase to 50% by the year 2030 [1]. In addition, deep learning networks can be trained to detect vehicles, pedestrians, signs, hazards, etc., and be deployed in ADAS.

Many distractions may cause a driver to overlook a traffic sign. This project aims to leverage deep learning techniques and transfer learning to detect and classify specific traffic signs within an image. One use case for this project is to display graphics of the traffic signs detected by the system. The driver can then use it as a reference, or the ADAS system can use it for lane detection, object warning, etc.

Developing a model that can detect all existing traffic signs is not practical for a one-semester project, which is why only four traffic sign types are targeted for the project. The four signs that the model will be trained to localize and detect are speed-limit, stop, railroad crossing warning (yellow), and do not enter signs. According to drivers of cars with an ADAS', 57% of drivers claimed that the car's ADAS had prevented them from getting in a crash [2]. Therefore, providing a second chance to the driver to reference a traffic sign helps avoid road violations and lessen their chance of causing an accident.

Developing Deep Convolution Neural Networks (DCNNs) from scratch that can quickly and efficiently detect images takes enormous resources and time: to speed up the processes, transfer learning can be trained to adapt to a new dataset. Transfer learning is the project's approach to detect and localize traffic signs.

II. Dataset

A. Acquisition

The dataset used in this project was manually collected from Google maps and labeled. Labeling was done using the LabelImg python graphical image annotation tool [3]. For the dataset, LabelImg was used to set bounding boxes for all 80 images manually. These boundaries were saved as an Extensible Markup Language (XML) file, which had to be converted to a TensorFlow Record (TFRecord) for the TensorFlow 2 object detection training scripts. To convert XMLs to TFRecods, a slightly modified version of a python script from GitHub was used to speed up the development process.[4].

There are four classes of images used in this dataset. Each class contains 20 images, with a 20% data split. Table 1 is a breakdown of the number of training and testing samples, and Fig. 1 is a visualization of two random samples from each class.

TABLE I. Dataset Split

Traffic Sign Dataset Class Information			
Classes	***Training***	***Test***	***Total***
Speed limit Sign	16	4	20
Stop Sign	16	4	20
Yellow Railroad Crossing Sign	16	4	20
Do Not Enter Sign	16	4	20

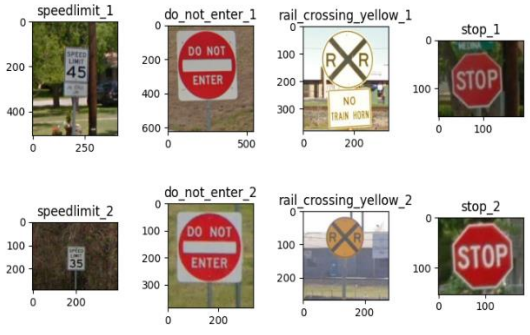

Fig. 1. Two random images from each class are shown in the figure.

B. Data Augmentation

Due to the small dataset size and the large training data usually required for DCNNs, data augmentation was used to help improve the model's performance. The traffic signs' intended viewing angle and orientation must be considered when deciding how to augment the data. Typical traffic signs face the front of the vehicle and are oriented so the driver can read it comfortably. Then, as the driver approaches the sign, the perspective causes it to enlarge before it starts to shear and then moves out of view. Also, it is essential to consider the brightness of the environment due to sunny or cloudy days. With brightness and scaling augmentation layers, the perspective of training data can emulate actual conditions. Table 2 shows the options and the parameters for each augmentation layer used in all four models, and in fig. 2, a few of the augmented training data samples are shown.

TABLE II. DATA AUGMENTATION PIPELINE

Data Augmentation Options		
Option	***1***	***2***
Random Adjust Brightness	Max Δ: 0.05	N/A
Random Image Scale	Min Δ: 0.50	Max Δ:1.50

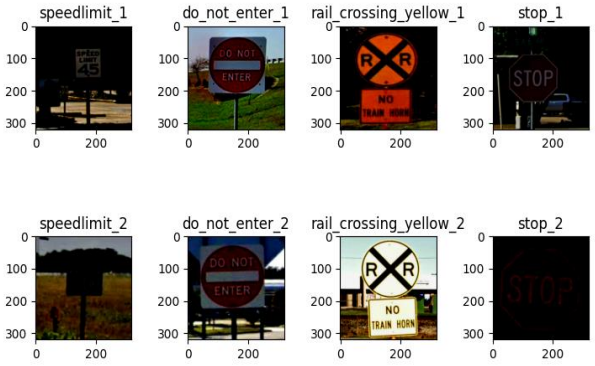

Fig. 2. This figure shows two images from each of the classes from the dataset with data augmentation applied.

C. TensorFlow Model Garden

The TensorFlow Model Garden "provides users [with] a centralized place to find code examples of state-of-the-art (SOTA) and reusable modeling libraries for TensorFlow 2" [5]. Specifically, the object detection libraries and models in the research collection were used to develop the models in this project. These are a collection of pre-trained research models in TensorFlow 1 and 2. Fig. 3 illustrates the development process for building, training, evaluating, and an inference model using the TensorFlow Model Garden. Furthermore, the last stage of fig. 3 mentions that a TensorFlow Lite (TFLite) inference model is generated. TFLite inference models can be used for deployment on a microcontroller or mobile device.

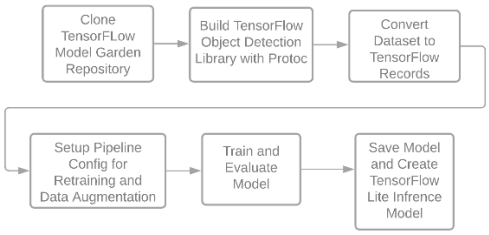

Fig. 3. The development process of the object detection model is shown in the figure.

1) TensorFlow 2 Detection Model Zoo

The TensorFlow 2 Detection Model Zoo is located in the object detection research directory of the TF Model Garden. The Model Zoo is an extremely powerful directory of pre-trained models that can be used for transfer learning. Table 3 contains a list of pre-trained models used in this project for transfer learning and performance metrics. The speed and mean average precision (mAP) scores are based on the model's performance on the COCO (Common Objects in Context) 2017 dataset. These generalized performance metrics give a rough idea of what performance and inference speed can be expected. Additionally, each model is designed to output bounding boxes. The resolution at the end of each model name refers to the resolution the input image is scaled to before processing.

a) SSD MobileNet V2 FPN Lite

The SSD Mobilenet V2 FPN Lite (SSD MobileNet) model is a Single-Shot multi-box Detection (SSD) network intended for object detection applications on mobile devices [7]. The backbone of the model is based on a Feature Pyramid Network (FPN), a ResNet network with taps at different feature scaling layers [7]. These taps are then fed to the rest of the network to generate bounding box locations and classifications [7]. The SSD MobileNet models were chosen due to the efficiency of their inference models.

b) EfficientDet D0 and D1

The EfficientDet D0 and D1 (EfficientDet) are SOTA object detectors balancing precision and training computational requirements [8]. This model boasts short computation time and high levels of accuracy with few training steps required [8]. Since the EfficientDet claims high levels of accuracy with few training steps, only 7k training steps were chosen versus the 25k steps for the SSD MobileNet.

TABLE III. TRANSFER LEARNING MODELS

Transfer Learning Models		
Model	***Speed(ms)***	***COCO mAP***
SSD MobileNet V2 FPN Lite 320x320	22	22.2
SSD MobileNet V2 FPN Lite 320x320	39	28.2
EfficientDet D0 512x512	39	33.6
EfficientDet D1 640x640	54	38.4

III. Training & Validation Results

Below in each model results section are two subsections, one is the training analysis, and two is the validation analysis. In section one, the loss functions are of the classification and localization accuracy. The classification loss function plots display the loss in the classification accuracy, and the localization loss is the loss of the bounding box regressor. The bounding box regressor is the kernel that sweeps through the image, trying to locate one of the trained objects. Next, the learning rate graphs are included as another indicator of the models converging. Lastly, a graph of the time spent per training step of the model is displayed, providing insight into training efficiency.

The second section contains the analysis of the validation process. When analyzing the performance of an object detection model, the average recall and mean average precision (mPA) are the go-to metrics. The recall score measures the model's ability to classify detected samples correctly. A true positive is counted when the intersection of the union (IoU) area is between 0.50 and 0.95 (the area in the overlapping region of the bounding box and the grounded truth box) and is correctly classified. When the grounded truth is present, but the model fails to detect it, it is classified as a false negative, while a true negative is every part of the image outside the grounded truth box that the model does pick up on. The last figure in each validation section looks at the testing results with bounding boxes and confidence levels superimposed on the image. The specific "do not enter" sign displayed in each test image was chosen because it was the most incorrectly classified sample.

Analysis of the training and validation data will be discussed in the following section, "*Training & Validation Analysis*."

A. SSD MobileNet V2 FPN Lite 320x320 (MobileNet320)

1) Training

a) Loss

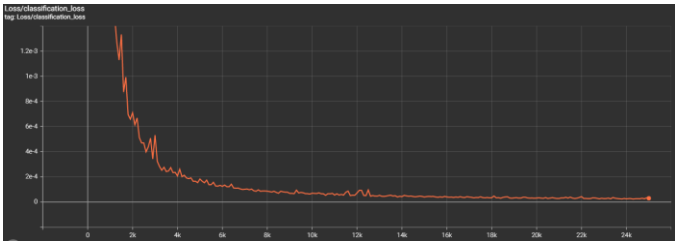

Fig. 4. Classification loss by steps plot.

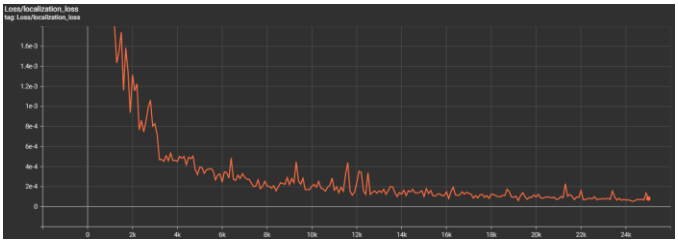

Fig. 5. Localization loss by steps plot.

b) Learning Rate

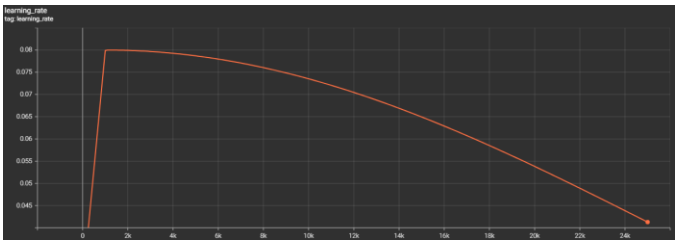

Fig. 6. Learning rate by steps plot.

c) Steps/Second

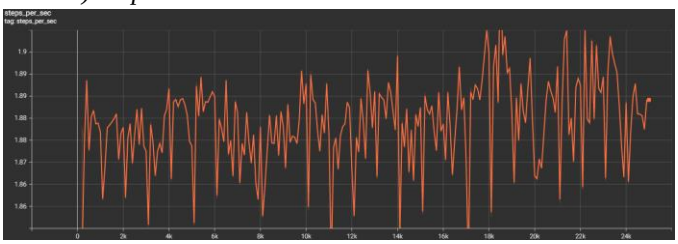

Fig. 7. Seconds per step plot.

2) Validation

TABLE IV. Mobile Net 320x320 Metrics

Metrics		
Metric	***Score***	***Step***
Average Recall 1 Detection/Image	0.8625	25k
Average Precision Over Classes with IOU range (0.5 to 0.95)	0.8552	25k

Fig. 8. The predicted bounded box generated by the Mobilenet320 model is on the left, and the grounded truth box is on the right.

B. SSD MobileNet V2 FPN Lite 640x640 (Mobilenet640)

1) Training

a) Loss

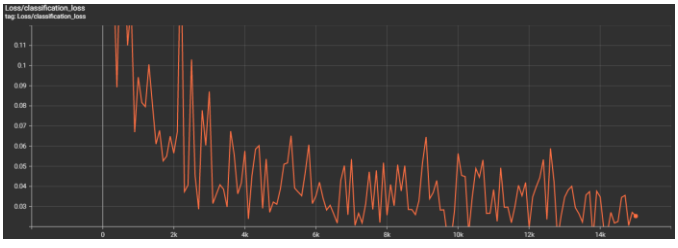

Fig. 9. Classification loss by steps plot.

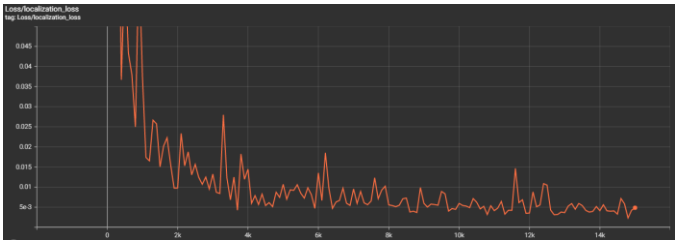

Fig. 10. Localization loss by steps plot.

b) Learning Rate

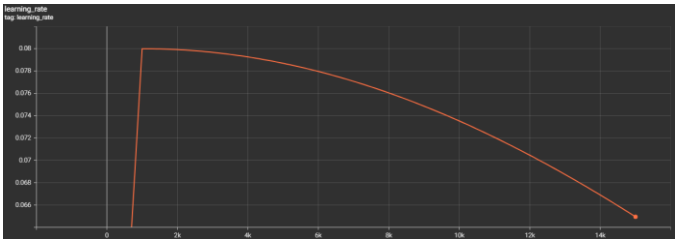

Fig. 11. Learning rate by steps plot.

c) Steps/Second

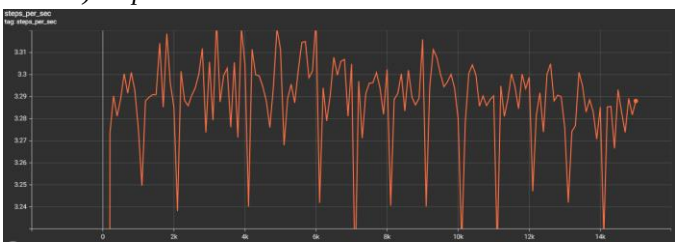

Fig. 12. Seconds per step plot.

2) Validation

TABLE V. MOBILE NET 640X640 METRICS

Metrics		
Metric	***Score***	***Step***
Average Recall 1 Detection/Image	0.8625	25k
Average Precision Over Classes with IOU range (0.5 to 0.95)	0.9163	25k

Fig. 13. The predicted bounded box generated by the Mobilenet640 model is on the left, and the grounded truth box is on the right.

C. EfficientDet D0 512x512 (EfficientDet D0)

1) Training

a) Loss

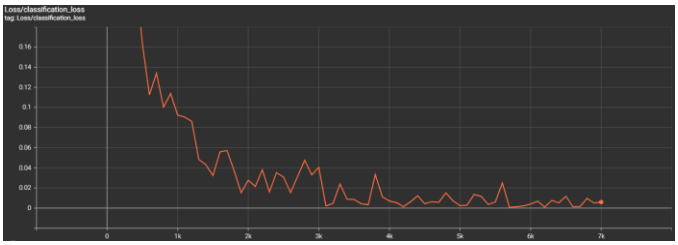

Fig. 14. Classification loss by steps plot.

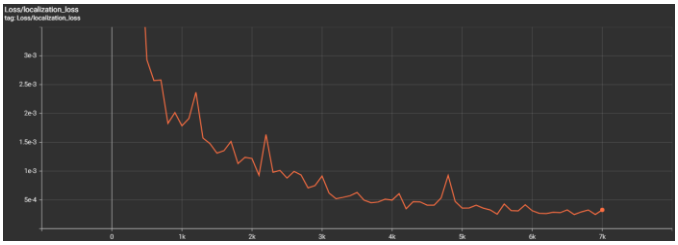

Fig. 15. Localization loss by steps plot.

b) Learning Rate

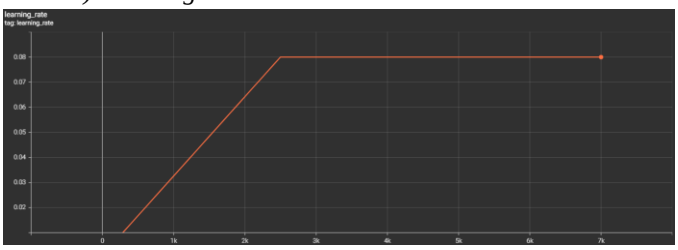

Fig. 16. Learning rate by steps plot.

c) Steps/Second

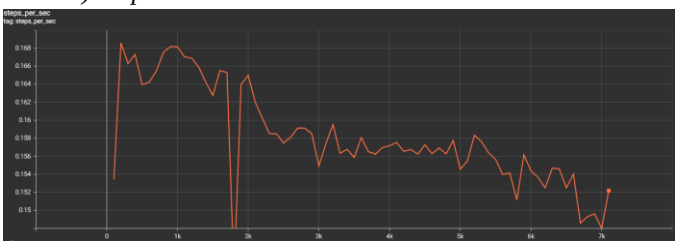

Fig. 17. Seconds per step plot.

2) Validation

TABLE VI. MOBILE NET 320X320 METRICS

Metrics		
Metric	***Score***	***Step***
Average Recall 1 Detection/Image	0.8687	25k
Average Precision Over Classes with IOU range (0.5 to 0.95)	0.8552	25k

Fig. 18. The predicted bounded box generated by the EfficientDet D0 model is on the left, and the grounded truth box is on the right.

D. EfficientDet D1 640x640 (EfficientDet D1)

1) Training

a) Loss

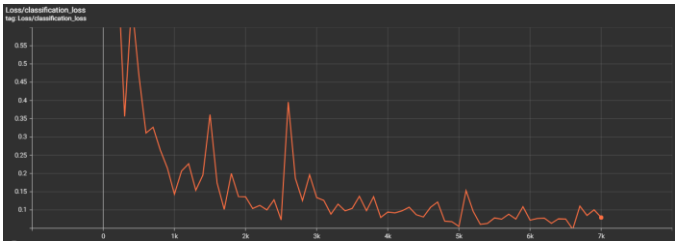

Fig. 19. Classification loss by steps plot.

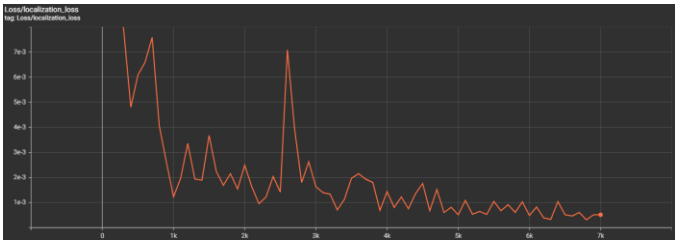

Fig. 20. Localization loss by steps plot.

b) Learning Rate

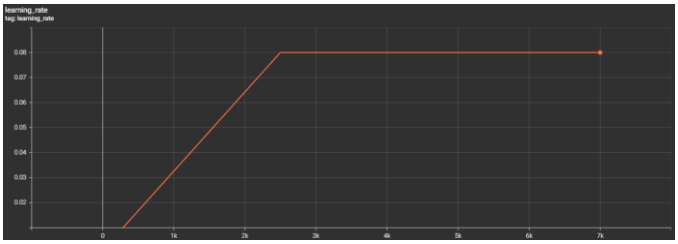

Fig. 21. Learning rate by steps plot.

c) Steps/Second

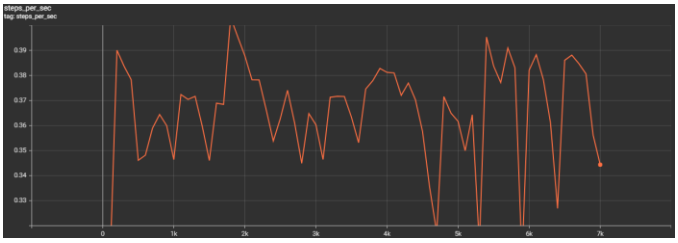

Fig. 22. Seconds per step plot.

2) Validation

TABLE VII. MOBILE NET 320X320 METRICS

Metrics		
Metric	***Score***	***Step***
Average Recall 1 Detection/Image	0.9187	25k
Average Precision Over Classes with IOU range (0.5 to 0.95)	0.9050	25k

Fig. 23. The predicted bounded box generated by the EfficientDet D1 model is on the left, and the grounded truth box is on the right.

IV. TRAINING & VALIDATION ANALYSIS

All training loss functions appear to converge for all the models trained, as shown in figs. 4, 5, 10, 11, 14, 15, 19, and 20, respectively. One thing to note is that the EfficientDet model's loss decreases faster than the MobileNet's loss function plots. Overall, the loss functions in all the models decrease and stabilize as more steps are processed. A decreasing loss function indicates the model is improving its performance on the training dataset, and when it starts converging, the model reaches its maximum potential on that dataset.

The model performance metrics were calculated based on the test data earlier defined in table 1. In table 8, the recall scores stay relatively stable between the four models, with EfficientDet512 coming slightly on top. The mPA, on the other hand, varies from model to model. It is important to understand that the EfficientDet model only trained for 7k steps and has mPA and recall scores within 1% of the MobileNet models. Even the MobileNet640 and EfficientDet640 precision and recall scores are relatively similar despite EfficientDet640 training for fewer steps.

TABLE VIII. VALIDATION MODEL PERFORMANCE

	Metrics	
Model	***Recall***	***Precision***
MobileNet 320x320	0.8625	0.8552
MobileNet 640x640	0.8625	0.9163
EfficientDet 512x512	0.8687	0.8552
EfficientDet 640x640	0.8552	0.9050

V. CONCLUSION

Tensorflow is a powerful tool that can quickly deploy and retrain SOTA object detection models. While training these models takes a long time, many free and cheap online compute

clusters can be used to offload computation. Additionally, tons of resources and tutorials exist on the internet that can teach you how to train your own custom model. The TensorFlow 2 Model Zoo is a great place to start transfer learning and dive deeper into the deep learning process.

You may have noticed a few discrepancies compared to the original project proposal. The first approach to this project was ultimately off-target, focusing on image classification techniques without considering the object localization process. Here is a brief overview of my work before switching over to TensorFlow 2 object detection. A custom DCNN was developed with integrated data augmentation layers, and a data augmentation python script was made that used Keras layers to generate additional training and testing samples. The model was extremely accurate (>99%) at determining the classification of the images. Still, quite obviously, it could not localize objects on an idea: this is when the project shifted to TensorFlow 2.

A. *Improvements?*

The most reliable way to improve the model is to get better data and more of it! Unfortunately, having such a small dataset means if the model is to get good at detecting specific objects, it must train on images with objects roughly the same size and proportions. Ideally, a dataset of annotated U.S. traffic signs images taken from a dashcam would be available. Unfortunately, for the four models trained in this project, object detection only begins to work when the sign fills up roughly the same screen area that the training samples did. As stated before, this problem could be addressed by increasing the data set.

Another region of improvement is with model pipeline and data augmentation layer selection. Referring to fig. 2, the speed-limit sign that has been augmented is complexly blacked out. This bug would effectively bias the data incorrectly, decreasing its performance.

Lastly, the most obvious way to increase the mPA of the models is to increase the number of steps the model trains for simply. Based on the loss function plots trend, these models could have continued to train, possibly improving a little more. These are just a few improvements that came up when working on the project. In conclusion, the EfficientDet 640x640 model performed the best, considering it only needs a fraction of the steps necessary in the MobileNets to achieve somewhat similar results.

VI. References

[1] "Huge opportunity as only 10% of the 1 billion cars in use have ADAS features," *www.canalys.com*. https://canalys.com/newsroom/huge-opportunity-as-only-10-of-the-1-billion-cars-in-use-have-adas-features.

[2] J. Huetter, "Consumer Reports: 57% of members say ADAS prevented a crash," *Repairer Driven News*, Jun. 25, 2019. https://www.repairerdrivennews.com/2019/06/25/consumer-reports-57-of-members-say-adas-prevented-a-crash/ (accessed Oct. 21, 2021).

[3] T. Lin, "labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images," *PyPI*. https://pypi.org/project/labelImg/ (accessed Nov. 08, 2021).

[4] N. Renotte, "nicknochnack/GenerateTFRecord," *GitHub*, Oct. 09, 2021. https://github.com/nicknochnack/GenerateTFRecord (accessed Nov. 15, 2021).

[5] "Introducing the Model Garden for TensorFlow 2," *TensorFlowBlog*. https://blog.tensorflow.org/2020/03/introducing-model-garden-for-tensorflow-2.html (accessed Nov. 15, 2021).

[6] M. Maithani, "EfficientDet: Guide to State of The Art Object Detection Model," *Analytics India Magazine*, Dec. 17, 2020. https://analyticsindiamag.com/efficientdet/ (accessed Dec. 06, 2021).

[7] P. Jin, V. Rathod, and X. Zhu, "Pooling Pyramid Network for Object Detection," Jul. 2018.

[8] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," Jul. 2020.