International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 08 | Aug 2021 www.irjet.net p-ISSN: 2395-0072

Comparison of Object Detection and Image Recognition Frameworks on Indian Currency Notes

M C Sohan[1], Akanksh A Manjunath[2], Prof. Lalitha V P[3]

[1]Undergraduate Student, R V College of Engineering, Bangalore, Karnataka, India
[2]Undergraduate Student, R V College of Engineering, Bangalore, Karnataka, India
[3]Assistant Professor, R V College of Engineering, Bangalore, Karnataka, India

---

Abstract – *The task of selecting the right image recognition and object detection framework for an application has become difficult with improvements in AI technologies as several frameworks are made available. Some frameworks perform better than others in terms of certain metrics. This article compares popular object detection and image recognition frameworks based on several metrics including accuracy metrics, efficiency metrics, training time, training ease, deployment ease and inference time. Conclusions about the behaviour and applicability of the frameworks have been made after comparing the training results on a dataset of Indian Currency Notes.*

Keywords: ***Object detection, Image classification, YOLO, Tensorflow, PyTorch***

1. Introduction

The visual system of humans is extremely fast and complex, allowing us to perform vision-based inference without any additional thought. This is possible due to the immensely complex networks of neurons in our brains which become capable of recognition, detection and localization of objects using just our visual input after being exposed to several examples of the objects. Neural networks are an attempt to replicate this behaviour of the human brain by building networks of neurons which become capable of human-like processing of inputs. Several frameworks, algorithms or techniques have been developed to aid in producing these results. With each method having its pros and cons, it becomes important to know the tradeoffs before selecting one for tackling a problem. Multiple factors have to be taken into consideration before proceeding with such a neural network method depending on the area of application, for example, to detect objects from a continuous video stream in real-time would require techniques that provide quick processing in exchange for some imperfections in localization.

With advancements in hardware, processing times have drastically reduced - this allows for more accurate algorithms to be deployed in practice since the processing time for inference would vary only slightly. There is another factor that can sometimes influence the selection of a technique - the time required to train and deploy the neural network model used for inference. Given the same data to several algorithms, the time required per epoch and the number of epochs required to train the model varies for different techniques. This article details the comparative study made using popular object detection and image recognition frameworks to compare methods based on certain metrics and to examine the effect of variations in the dataset on performance metrics.

Several popularly used frameworks like TensorFlow[1], PyTorch[2] and Darknet[3] exist for the purpose of object detection and image recognition, which can be used alongside many different algorithms to develop neural networks for the required purpose. The You Only Look Once(YOLO) approach to object detection moves away from the ideology of using classifiers to perform detection, instead, it frames the object detection problem as a regression problem to spatially separated bounding boxes and associated class probabilities[4]. Another popular library for training deep learning models (especially for image classification) is the Fast.ai library. This library sits on top of the PyTorch library and provides powerful functions that allow models to be trained and tested using just a few lines of code. It also allows lower-level access to programmers to perform minute tweaks easily [5].

The working of a neural network prompting it to detect an object or to recognize an image can be split into two stages - feature extraction and detection. During detection, initially, common features from various regions of the image are extracted using the starting layers of the neural network. These layers are usually large in comparison to the number of layers towards the end that are used to gather this feature information and perform the detection task. Extremely large datasets like ImageNet[6] are needed to train the neural networks to develop the ability to extract different features like gradient, texture, edges, etc. In the present day, almost no image recognition or object detection model is trained from scratch since several pre-trained models having excellently trained feature extraction layers are available. Only the layers closer to the output layer need to be trained to adapt these models to recognize newer objects, this method of training a model is known as transfer learning[7]. The biggest advantage of transfer learning is that despite having a smaller dataset, good results can be obtained as the number of layers that require training is significantly lesser than the total number of layers present in the network - the pre-trained layers can be retained for feature extraction.

2. Literature Review

A comparison of TensorFlow and Pytorch was previously made by Simmons and Holliday using the publicly available set of Amazon reviews for video games. The binary classification was performed using sentiment analysis to classify the reviews as being positive or negative. They compared the training time, memory usage, and ease of use of the two frameworks in the same testing environment (on the same Google Colaboratory GPU instance)[8]. There are however some peculiarities that are exhibited based on the dataset as seen in the research of Hussain et al. where performance analysis of five machine learning algorithms (Naïve Bayes, Bayes Net, K-Nearest Neighbour, Multi-Layer Perceptron, and Support Vector Machine) for recognizing daily life activities was done. It was seen that some models did well to recognize some activities and failed to recognize other activities, different models better recognized different activities[9].

Another comparison between PyTorch and TensorFlow was performed by Heghedus et al. A comparison of the two frameworks was performed on traffic data obtained from various IoT sensors. It was seen that one model outperformed the other in some scenarios whereas the result was the opposite for a few other scenarios. It was observed in the results that some network configurations performed better on one framework than the other[10]. Google Research, Brain Team studied the impact of data augmentation on object detection[11] and demonstrated that data augmentation operations borrowed from image classification may be helpful for training detection models, but the improvement is limited. Their experiments reported an improved accuracy on both COCO and PASCAL VOC datasets by an optimized data augmentation policy. This shows that augmentation to be applied is dependent on the dataset rather than the representation or the model that is being used.

An improved method for training models when a small dataset is available was described by Li et al[12]. They described a process of sample enhancement and transfer learning for the YOLO method using their small dataset containing 363 images (296 training images and 67 test images) having playing cards of 6 different types. The results show that YOLO is fast and accurate, and that sample enhancement can improve detection performance. Contrary to Li et al.[12] Abramovich et al. [13] studied the accuracy of multiclass image classification in a high dimensional setting where the number of classes were large. An extremely counter-intuitive observation was made where the results indicated that the precision of classification can improve as a number of classes grow in a higher-dimensional setting.

Redmond et al.[14] described both an improvement on the old YOLO - calling it the YOLOv2 and also a new technique named YOLO9000 in which joint training of data using both image classification and object detection datasets simultaneously allowed the training of YOLO9000 to be capable of recognizing more than 200 classes whilst having object detection annotations for only 44 classes. Padilla et al.[15] compared various available metrics for object detection and matched the metrics with the methods with which they are available. The study showed that not all metrics are available to be used with every approach readily and hence, proposed a standard implementation that can be used as a benchmark with minimum adaptation. The same authors continued their work [16] to produce an open-source toolkit that provides tools to compute metrics in a standardized manner, supporting all the popular annotation formats.

NVIDIA research[17] suggests that to capture the temporal nature of the video, Average Precision metric is not sufficient. For real-time video analysis, the inference time matters and hence an average delay metric was proposed. The research also proposes that for detection activity, the corresponding delay must be taken into consideration. Zou et al.[18] surveyed 400+ research articles to produce a summary of object detection in the past 20 years. The authors detail how the object detectors got split into two major categories (single-stage and multi-stage detectors) around 2014. The study indicates that one of the great successes of the decade was RCNN, it changed the way object detection problems were approached - improved versions of this multi-stage detector were developed (viz. Fast RCNN and faster RCNN).

3. Methodology

3.1 Dataset

We use a custom dataset of Indian currency notes (denominations 50, 100, 200, and 500) to perform the comparison. The dataset contained 490 images specifically selected and annotated for object detection purposes. It comprised of 50 images per class with just one object in the image and 90 images having multiple objects per image. Color and shape alone cannot be an indicator for the object, to ensure that this is not the case we included several objects that were not currency notes but resembled them in terms of shape and color. The notes were classified into 8 classes - a back and a front class each for the 4 denominations. This base dataset is then used to produce a dataset that is 3 times larger by using augmentation techniques like image rotation, skew and perspective warping. Care was taken that several images of notes of the same or similar color but of different currencies were added to the test set. This was to ensure that the models were able to pick up on the features of the notes and were not just relying on broad color patterns. In almost all the images, the notes are completely visible and are oriented without rotation relative to the camera frame - this was to ensure that folded/tilted notes in the background would not be recognized. The dataset used for training the models was a custom-created dataset of new Indian Currency

notes. This dataset was normalized to 2000x2000 pixel dimension and then passed through several augmentation stages of auto orientation, 90-degree rotations, slight object tilts of -7 to +7 degrees, and brightness changes. This yielded a dataset that was 3 times larger than the original raw dataset.

3.2 Metrics

We considered accuracy, training time, training ease, inference time and deployment ease as metrics of interest. Accuracy, Training time, Inference time are performance-based metrics whereas training ease and deployment ease showcase the versatility and ease of understanding and using the frameworks.

3.3 Environment

All the below-mentioned models were trained using Google Colaboratory GPU instances.

3.4 Image Classification

TensorFlow and PyTorch frameworks for image classification were used to train models. These models were then evaluated using the above-mentioned metrics. PyTorch was evaluated by using the Fast.ai library which uses PyTorch under the hood.

3.5 Object Detection

For object detection, the trending YOLOv4-Darknet, YOLOv5-PyTorch were compared. Additionally, YOLOv4 darknet weights were converted to TensorFlow weights and then converted to TensorFlow Lite format which can be directly deployed on a mobile device. The parameters and other configurations were tuned as per requirement and the models were trained. TensorFlow library was used for TensorFlow image recognition, the custom framework, Darknet[19], was built from source for YOLO v4 object detection, YOLO v5[20] repository was cloned and Fast.ai library was used for image recognition. Tensorboard was used to evaluate the various metrics like mAP, precision, recall, and box accuracy for object detection. Training and Inference times were measured using a built-in command in Google Colab.

3.6 Validation methodology

During training, the models were provided with a validation set which was a tenth of the size of the training set. Additionally, a tenth of the set was retained for testing and was never shown to the model during training. The testing of the model was done automatically by using the trained models to infer/predict the denomination and the side (front/back) of the currency. The prediction results were verified manually to ensure that the expected predictions are being made w.r.t the bounding boxes. The metrics were automatically analyzed using Tensorboard and compared for the different models. Additionally, inference time for a set of 40 images was also measured for each model.
The complete flow has been summarized as a workflow diagram in Figure 1.

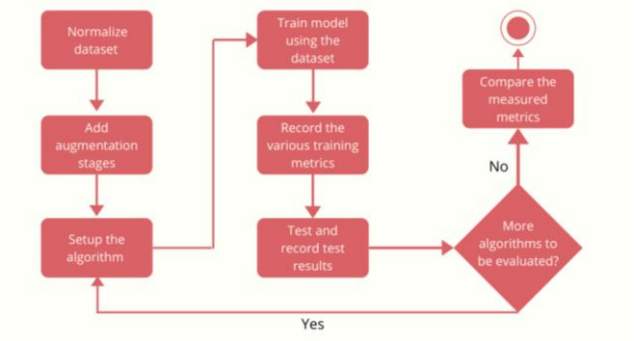


Fig-1: Workflow diagram

4. Results and Analysis

Our comparison will be applicable to all areas where objects belonging to a few classes have to be detected and in case of using the object detection algorithms and detection/recognition applications where detection time is a key factor like counting the number of eggs and larvae from a photograph of a worm's nest or counting the number of different types of cells in a medical close up image, also applications where the dataset available is not large enough to attempt a full-fledged deep learning algorithm without transfer learning. In the below paragraphs we see the results of our experiments on object detection and image recognition.

4.1 Image Recognition

For image recognition, we picked the same architecture ResNet50 for both frameworks with ReLU activation function and Softmax in the last layer. This was done so as to have a consistent architecture to compare both frameworks effectively. Also, doing so allowed us to focus on any dataset issues or discrepancies.

Reported Accuracy: TensorFlow has better validation accuracy (Figure 3) than PyTorch but performs poorer on the test dataset (Figure 4), this is because of overfitting of data which can be attributed to lack of training customization. However, with respect to PyTorch we see much better performance on the test dataset also.

TensorFlow:
Validation: 100%
Test: 10/38 = 27%
PyTorch:

Validation: 1/94 = 98.93%
Test: 72.91%

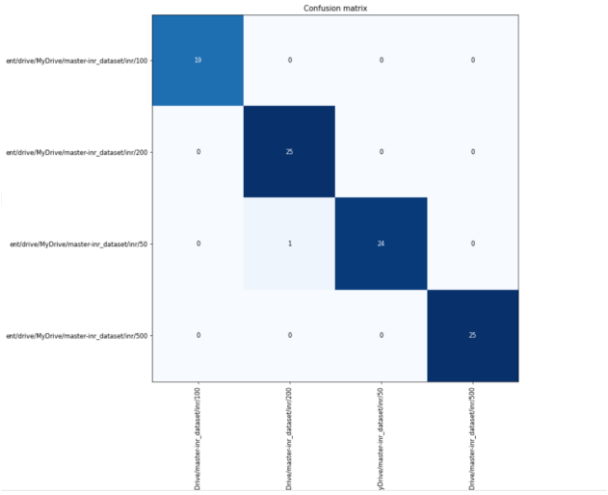
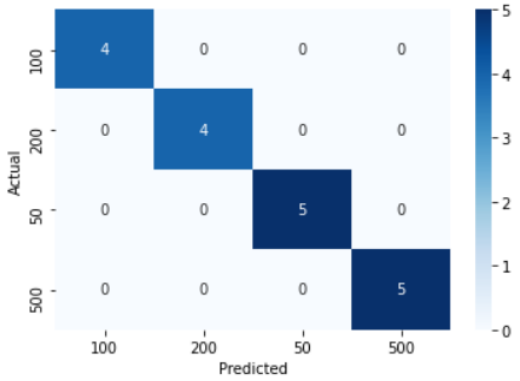


Fig-2: PyTorch confusion matrix




```
test accuracy: 100.0
```

Fig-3: TensorFlow confusion matrix

Fig-4: TensorFlow test results

Training Time: PyTorch took more time to train as it can be customized and the method of training involves selecting the range between which the learning rate is chosen. This also resulted in higher accuracy as can be seen above (Figure 2). This is known as the CLR (Cyclical Learning Rate) method (Figure 6).

TensorFlow: 2min 36s
PyTorch: 6min 36s

Training Ease:
TensorFlow: TensorFlow does not offer much customization (Figure 5) with respect to training apart from data augmentation techniques and the number of epochs, that PyTorch also offers. Hence, the training time of TensorFlow is lower compared to that of PyTorch.

```
8/8 [==============================] - 8s 595ms/step - loss: 0.6530 - accuracy: 0.8913
Epoch 2/30
8/8 [==============================] - 5s 609ms/step - loss: 0.3565 - accuracy: 0.8713
Epoch 3/30
8/8 [==============================] - 5s 623ms/step - loss: 0.1713 - accuracy: 0.8949
Epoch 4/30
8/8 [==============================] - 5s 595ms/step - loss: 0.0481 - accuracy: 0.9860
Epoch 5/30
8/8 [==============================] - 5s 618ms/step - loss: 0.0062 - accuracy: 1.0000
Epoch 6/30
8/8 [==============================] - 5s 611ms/step - loss: 0.0223 - accuracy: 1.0000
Epoch 7/30
8/8 [==============================] - 5s 600ms/step - loss: 0.0039 - accuracy: 1.0000
Epoch 8/30
8/8 [==============================] - 5s 562ms/step - loss: 0.0033 - accuracy: 1.0000
Epoch 9/30
8/8 [==============================] - 5s 588ms/step - loss: 0.0022 - accuracy: 1.0000
Epoch 10/30
8/8 [==============================] - 5s 554ms/step - loss: 0.0021 - accuracy: 1.0000
```

Fig-5: TensorFlow training

PyTorch: PyTorch can be customized as highlighted above (refer Figure 6). PyTorch also allows a range of data augmentation techniques to be applied to the data and modifications in the number of epochs. Overall, PyTorch is the most customizable framework available and this can be attributed to the history and the purpose behind developing PyTorch.

epoch	train_loss	valid_loss	error_rate	time
0	1.536565	0.362343	0.148936	01:09
1	0.768041	0.290622	0.053191	00:19
2	0.491515	0.169097	0.031915	00:19
3	0.355607	0.129965	0.021277	00:19
4	0.239797	0.123395	0.021277	00:20

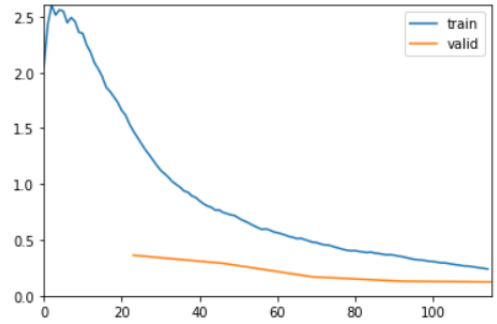


Fig-6: PyTorch training

Time for Inference (for 40 images): The time taken to load the model and run inference on a batch of 40 images is

detailed here. TensorFlow takes double the time of PyTorch to run inference as it takes a lot of time to set up the model and infer from it. This is because of the complex architecture of the framework whereas PyTorch is a relatively lighter framework.

TensorFlow: 52.5s
PyTorch: 25.3s

Deployment: TensorFlow features easy deployment and has natural libraries which port to other architectures like mobile devices. Many applications have also been developed on many platforms which showcase the ease and advantages of having a TensorFlow deployment. However, PyTorch being a relatively new framework does not support as many libraries as TensorFlow does. We have showcased a REST deployment model where the PyTorch model was exported to a pickle file and was deployed on a web service Render. PyTorch still has a long way to go with respect to deployment to reach the standards TensorFlow has set.

TensorFlow: Direct inference from webcam live video feed and as a deployed mobile application.
PyTorch: Deployed on Render web service.

4.2 Object Detection

4.2.1 YOLOv5 PyTorch

Reported Accuracy: This model featured the highest accuracy over the other models. This can be attributed to its mature backend framework and the improvements of Yolov5 over its predecessors. Accuracy is measured in terms of mAP which is the Maximum a Posteriori or MAP for short. It is a Bayesian-based approach to estimating distribution and model parameters that best explain an observed dataset. The mAP provides an alternate probability framework to maximum likelihood estimation for machine learning. We observed a mAP of 0.95 which was the highest achieved in our experiments.

Training time: Although object detection algorithms take longer to learn compared to image recognition frameworks, this was one of the few algorithms along with the framework which resulted in very low training times. The total training time of Yolov5 Pytorch was just 21min 41s for 100 epochs.

Time for inference: Inference time was 6.11 seconds for 49 images which is faster than image recognition algorithms. Also, this framework would be used in real-time scenarios and hence is expected to have low inference times. Some of the sample inference results are shown in Figure 7.

Fig-7: Yolov5 results

Training ease: Apart from data augmentation techniques, low-level customization cannot be expected in training of object detection algorithms. Different Tensorboard graphs (Figure 8) offered a bird's eye view of the training and a way to stop and restart the training whenever required since the training time was not high.

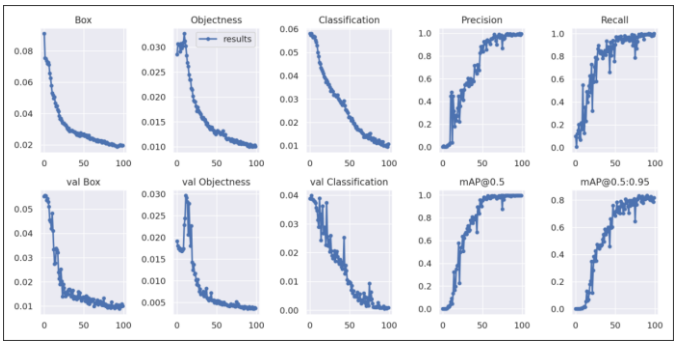


Fig-8: Yolov5 training graphs

Deployment: The only criteria where the Yolov4 performs better than Yolov5 is the deployment. Since it is a relatively new framework, there aren't as many libraries built around getting it deployed on different platforms. Hence, this is an unexplored domain as of now but can be expected to pick up quickly since the training time reduction and accuracy increase was in multiples compared to Yolov4.

4.2.2 Yolov4 Darknet

Reported Accuracy: This model featured one of the highest accuracies over all the other models. We observed a mAP of 0.993 which was the highest achieved till now in our experiments. However, since for implementations we would have to convert this to the Tflite representation, this would result in a drop of accuracy and hence justifies the high accuracy in the initial representation.

Training time: The total training time of Yolov4 Darknet was 3h 16min 46s for ~1500 epochs.

Training Ease: The lack of graphs resulted in difficulty in training. It was only possible to know how well the model is performing through just the mAP value which was calculated every 1000 epochs. Adding more dynamic graphs would help in achieving more training ease.

Time for Inference: Inference time was 7min 14s for 40 images which are and should be faster than image recognition algorithms. Sample inference results are shown in Figure 9.

Fig-9: Yolov4 results

Deployment: The only criteria where Yolov4 outperforms all the other algorithms is in a deployment where it can be converted to a Yolov4 Tflite representation which can be deployed on multiple platforms owing to its mobile nature for a small drop in the accuracy. This is acceptable since this would result in seamless deployment and hence no model would have to be trained again for mobile deployment specifically. Here we show deployment on a live webcam feed (Figure 10).

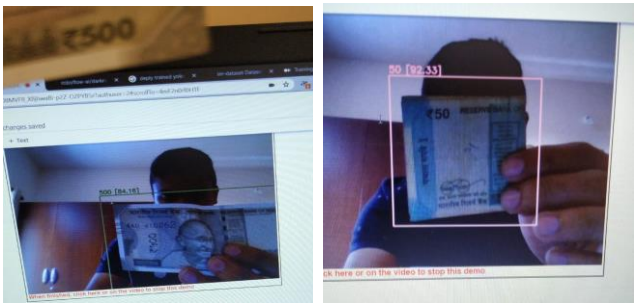


Fig-10: Yolov4 deployment

5. Conclusion and future enhancements

Our intention was to capture the difference in frameworks keeping the dataset constant to answer the question that most newcomers to the machine learning domain have, “PyTorch or TensorFlow or something else?”. Hence, the article captures our efforts, experiments, and the documentation that went behind trying to explore this ourselves. Our motivation was to capture the mind of a beginner in the field and to help them pick a framework based on a series of metrics. This might have been achieved with respect to TensorFlow and PyTorch but not including a custom framework like Darknet which has many implementations and is relatively new to the field. Also, applying this to both image recognition and object detection helped us capture the differences in the frameworks available for different use cases. It required developing a complex dataset that would suffice with minor changes for both use cases. Hence, we were able to include metrics like deployment which are usually ignored in comparison of frameworks, typically only concerning performance. This would serve as an efficient guide to cover an end-to-end machine learning project, from preparing the dataset to deploying the model on more than one platform.

Machine learning is a field known for not being kind to novices. It either requires a complex understanding of math or sometimes it is reduced to preparing the dataset and a model is picked and trained on the dataset making it an oversimplification. Capturing uncommon metrics and offering a wide array of growth was what we intended to do. A similar kind of study done on a more diverse dataset will prove to be more helpful in making a choice of the algorithm to be used. Also, introducing this technique to fields like NLP, speech processing would open up Artificial Intelligence and Machine Learning to a wide variety of audiences by making it accessible to all domainss, thus making it a global collaboration with contribution from all sectors.

REFERENCES

[1] Goldsborough, Peter. "A tour of tensorflow." arXiv preprint arXiv:1610.01178 (2016).
[2] Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen et al. "Pytorch: An imperative style, high-performance deep learning library." arXiv preprint arXiv:1912.01703 (2019).
[3] Redmon, Joseph. "Darknet: Opensource neural networks in c." (2013): 2021.
[4] Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779-788. 2016.
[5] Howard, Jeremy, and Sylvain Gugger. "Fastai: A layered API for deep learning." Information 11, no. 2 (2020): 108.
[6] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848
[7] Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." IEEE Transactions on knowledge and data engineering 22, no. 10 (2009): 1345-1359.

[8] Simmons, Chance, and Mark A. Holliday. "A comparison of two popular machine learning frameworks." Journal of Computing Sciences in Colleges 35, no. 4 (2019): 20-25.

[9] Hussain, Rida Ghafoor, Mustansar Ali Ghazanfar, Muhammad Awais Azam, Usman Naeem, and Shafiq Ur Rehman. "A performance comparison of machine learning classification approaches for robust activity of daily living recognition." Artificial Intelligence Review 52, no. 1 (2019): 357-379.

[10] C. Heghedus, A. Chakravorty and C. Rong, "Neural Network Frameworks. Comparison on Public Transportation Prediction," 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2019, pp. 842-849, doi: 10.1109/IPDPSW.2019.00138.

[11] Zoph, Barret, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. "Learning data augmentation strategies for object detection." In European Conference on Computer Vision, pp. 566-583. Springer, Cham, 2020.

[12] Li, Guanqing, Zhiyong Song, and Qiang Fu. "A new method of image detection for small datasets under the framework of YOLO network." In 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), pp. 1031-1035. IEEE, 2018.

[13] Abramovich, Felix, and Marianna Pensky. "Classification with many classes: challenges and pluses." Journal of Multivariate Analysis 174 (2019): 104536.

[14] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7263-7271. 2017.

[15] Padilla, Rafael, Sergio L. Netto, and Eduardo AB da Silva. "A survey on performance metrics for object-detection algorithms." In 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), pp. 237-242. IEEE, 2020.

[16] Padilla, Rafael, Wesley L. Passos, Thadeu LB Dias, Sergio L. Netto, and Eduardo AB da Silva. "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit." Electronics 10, no. 3 (2021): 279

[17] Mao, Huizi, Xiaodong Yang, and William J. Dally. "A delay metric for video object detection: What average precision fails to tell." In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 573-582. 2019.

[18] Zou, Zhengxia, Zhenwei Shi, Yuhong Guo, and Jieping Ye. "Object detection in 20 years: A survey." arXiv preprint arXiv:1905.05055 (2019).