Real-time object detection on Mobile networks

Project for E599: High performance Big Data Systems

Mariem Loukil
School of Informatics, Computing and Engineering
Indiana University
Bloomington, Indiana, USA
mloukil@iu.edu

ABSTRACT

The aim of this project is to perform real-time object detection on IndyCar images and videos using object detection models that could be deployed on mobile and edge devices. In this work, we compare different models and architectures (SSD-Mobilenet-v1, SSD-Mobilenet-v2 and Tiny-Yolov3) to see which gives the best accuracy and performance in the context of our given problem (on CPU and GPU)

KEYWORDS

Real-time, Object detection, Deep Learning, Tensorflow, Tensorflow model zoo, Tiny-yolov3

1 INTRODUCTION

Nowadays object detection is one of the most important research fields when it comes to Machine Learning, Deep learning and Artificial Intelligence in general. Many existent models are competing to achieve the best object recognition and detection. Soon, the task of accomplishing object detection was no longer enough because there are other criteria that the model should take into consideration and satisfy.

First, it's the real-time aspect of the detection. Not only should the models detect and recognize the object but it should also realize it in real-time. In some cases, the detection time can be very critical. For example, for self-driving cars, detecting a pedestrian crossing should be done in real-time and initiating a reaction accordingly should be done very fast to avoid any problems that can happen. The second aspect is for the object detection model to be able to run on constrained environments like Mobile phones, IoT object etc. This is an important aspect because the models are becoming more and more complicated and resource demanding by the day. It has been shown that compute systems requirements are doubling each 3 months [1]. Such Big models cannot be deployed on edge devices because of their size. Therefore, it is necessary to have models that are not just able to detect the object but also to have a size small enough to be deployed on an edge device.

The main goal of this project is to experiment with different object detection models that are able to run on mobile devices and compare them in order to choose the best one for our use case. The data we are using is IndyCar data which consists of race car images. Our objective is to find the optimal model that will detect the cars without trading off the performance and time because they are critical for our case.

, ,
.

2 RELATED WORK

MobileNets[5] is one of the most famous object detection models. It was introduced by Google in 2017 as a solution for mobile and embedded vision applications. They describe it as " light weight deep neural network"[5] that "efficiently trade off between latency and accuracy"[5]. Although its principal implementation is for image classification it can and has been used for object detection objectives. This model exploits SSD (SSD: Single Shot MultiBox Detector) [7] that was also previously introduced by Google which is a new method of detecting objects within an image

Pelee (or PeleeNet) [12] is another object detection model that performs real-time on mobile devices. Their approach is to conventional convolutional layers as opposed to other models that rely on depth-wise separable convolutions (like MobileNets). PeleeNet is a version of DenseNet and an optimization of SSD [7]. Their focus is to create a model that improves speed compared to existent with acceptable accuracy.

YOLO (You Look Only once) [9] is a real-time object detection approach that was first introduced in 2016. The novelty about the approach is that the detection was considered as a regression problem instead of a classification problem. Over the years different versions of YOLO have been introduced as an improvement of the original model (YOLOv2 [10], YOLO9000[10], Tiny-YOLOv2[8], YOLOv3[11], Tiny-YOLOv3[11],YOLO Lite Tiny-YOLOv2[8], Nano YOLO [13], etc.). Then newer versions of YOLO take the object detection on mobile and small devices into consideration. For example, YOLO Lite is destined for non-GPU devices and the Tiny versions are supposed to perform as good as the other versions in constrained environments. [6]

3 ARCHITECTURE AND IMPLEMENTATION

3.1 Architecture

There many networks that give great results in terms of real-time object detection. However, these networks cannot be deployed in constrained environments like mobile phones or edge devices. Thus, our choice of architectures was based on the fact that they can be deployed on Edge.

3.1.1 SSD-Mobilenet.

Mobilenet networks (mobilenetv1 and mobilenetv2) are originally classification networks and on their own cannot perform object detection and can run efficiently on mobile devices . That is why their the output of the different convolution layers is fed into an SSD model that is in charge of using those output to perform the actual object detection. The figure below shows the architecture of mobilenetv1. The difference between mobilenetv1 and mobilenetv2

is that the second version of the network introduced the idea of a residual connection that in theory is supposed to make the new network faster [4]. We will be implementing both networks to see what kind of results they produce and we will compare those results in the next sections.

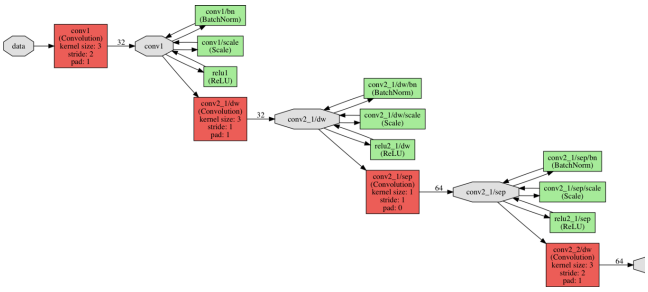

Figure 1: Mobilenetv1 Architecture [3]

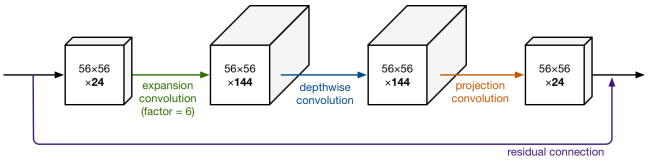

Figure 2: Mobilenetv2 Architecture [4]

3.1.2 *Tiny-yolov3.*

Yolov3 is a very known and used real-time object detection algorithm. Although its results are very good, the model is very heavy and it cannot be deployed on mobile devices. That is why a "tiny" version of it was introducted to enable it to be deployed and run in constrained environment. In our implementation, we saw that Tiny-yolov3 runs inference around 20 times faster on CPU.

Tiny-yolov3 has only 19 convolutional layers, whereas Yolov3 has 53 convolutional layers which is why the network is considered simpler to deploy on deifferent devices

3.2 Implementation

One of the hardest part of this project is to find implementations of the chosen network that can actually be ran. On github there are many implemenations. However, it usually involves many packages and dependencies that is very hard to make them work and interact together.

Besides, most of the present implementation, are very rigid when it comes to data, as in, they only work on famous datasets like COCO dataset [1] or Pascal VOC dataset [2]. And, in those cases it is impossible due to the complexity of the network and the code involved to intervene and adapt the network to our code. Because the main goal in there is to train the networks from scratch on the IndyCar data.

[1] cocodataset.com
[2] host.robots.ox.ac.uk/pascal/VOC/

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Figure 3: Tiny-yolov3 Architecture [2]

3.2.1 *SSD-Mobilenet.*

For SSD-Mobilenetv1 and SSD-Mobilenetv2, we chose to work with Tensrflow model zoo models [3]. It provides pretrained models on various datasets. The goal here is to train from scratch and not use a pretrained model because we are only detecting on class (cars) and race cars are different from ordinary cars.

The implementation is in Tensorflow and it uses Tensrflow object detection API [4]

The models that we chose to work with are:

- ssd_mobilenet_v1_quantized_coco
- ssd_mobilenet_v2_quantized_coco

For the ssd_mobilenet_v1_quantized_coco we used the following configurations

- Batch size: 128
- Learning rate: 0.05
- Optimizer: Momentum
- Number of steps: 25,000

For the ssd_mobilenet_v2_quantized_coco we used the following configurations

- Batch size: 24
- Learning rate: 0.004
- Optimizer: Momentum
- Number of steps: 25,000

After training both models, we froze the best checkpoint and used it to run inference on both CPU and GPU on images, video and streaming video. Both models use the cocoAPI [5] for the metrics calculations such mAP. This API offers very detailed metrics for the detection

3.2.2 *Tiny-yolov3.*

For Tiny-yolov3, finding an implemntation that is flexible enough to adapt and train on our dataset wasn't obvious nor easy. There

[3] SSD-Mobilenetv1
[4] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.
[5] https://github.com/cocodataset/cocoapi

are many implementation of the original model YOLOv3 but not for the Tiny-yolov3 version.

Besides unlike the mobilenet networks, YOLO networks and its descendants aren't implemented in python, which makes the task of understanding them, modifying and updating the code according to our specific needs very tough.

We eventually found a Tensorflow implementation of Tiny-yolov3 [6]. However, this implementation is not documented at all. But, the directory and folders structure looked very similar to the Tensorflow-yolov3 [7] implementation. So, we tried to adapt that implementation to what we found. The model is originally trained on the Pascal VOC dataset. So, even though we changed the actual data that we trained with we kept the same structure and requirement as the Pascal VOC dataset. For the Tiny-Yolov3 we used the following configurations

- Batch size:30
- Learning rate: 1e-3
- First stage epochs: 40
- Second stage epochs: 60
- Optimizer: Adam

After training the model, we froze the best checkpoint and used it to run inference on both CPU and GPU on images, video and streaming video.

4 EXPERIMENTS

4.1 Environement

4.1.1 *CPU Environment.*

- Ubuntu 18.04.3 LTS
- 48 Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz CPU(s)

4.1.2 *GPU Environment.*

- Fedora 7.7
- 8 Tesla K 80
- CUDA Version: 10.1

4.2 Dataset

The dataset consists of the IndyCar images data that were annotated. The annotations indicate the coordinates of the box surrounding the object. There can be multiple objects in a file, and each image has one annotation file.

The dataset is composed of

- 2873 training images
- 184 Test images

the figure below shows and example image:

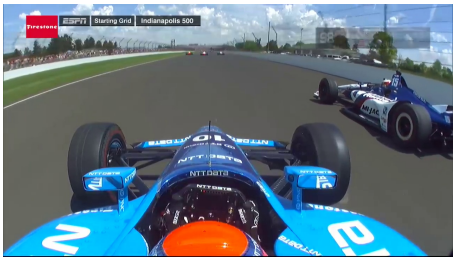

Figure 4: Data image example

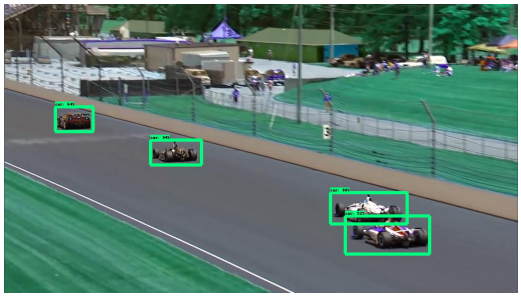

Figure 5: SSD-Mobilenetv1 image result

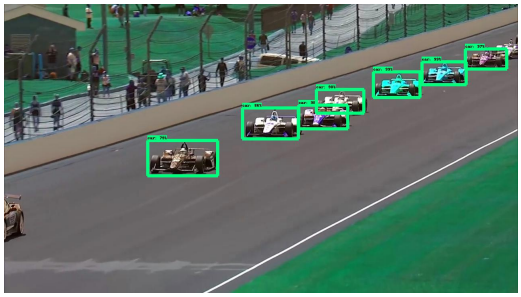

Figure 6: SSD-Mobilenetv2 image result

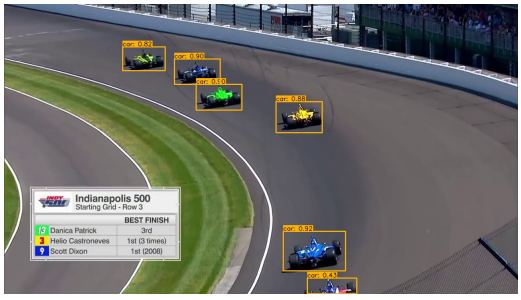

Figure 7: Tiny-yolov3 image result

4.3 Results

4.3.1 *Images results.*

The 3 images below shows an example of the output of each model.

4.3.2 *Mean average precision (mAP).*

Like mentioned previously the mobilnet models were trained on COCO and they use coco API metrics to measure mAP which is different from the way VOC calculates mAP. No only the cocoAPI provides more detailed results but the main mAP score is considered very low compared to the mAP obtained by VOC mAP (around 30% vs. 78%)

[6] https://github.com/ootzfoo/TensorFlow-yolov3-tiny
[7] https://github.com/YunYang1994/tensorflow-yolov3

But, the mAP that cocoAPI gives is no other than the average of variants of mAP are over 10 IoU thresholds (i.e., 0.50, 0.55, 0.60, …, 0.95) [14] and mAPIoU=.50 of cocoAPI is identical to the Pascal VOC metric. Therefore, this will enable us to compare the different models like in the table below.

Table 1: mAP results for the 3 models

	SSD-Mobilenetv1	SSD-Mobilenetv2	Tiny-yolov3
mAP	72.44%	64.27%	78.04%

Table 2: mAP detailed results for SSD-Mobilenetv1 and SSD-Mobilenetv2

	SSD-Mobilenetv1	SSD-Mobilenetv2
mAP Large	65.45%	63.94%
mAP Medium	32.03%	24.31%
mAP Small	7.2%	3.73%

Tiny-yolov3 outperforms both SSD-Mobilenetv1 and SSD-Mobilenet2 when it comes to mAP (for mAPIoU=.50) with 78.04%. In second place, is SSD-Mobilenetv1 with 72.44% and SSD-Mobilenetv2 has the worst mAP with 64.27%. As Figure 8 shows, Tiny-yolov3 made 654 correct predictions out 951 objectes dettectted. The Table2 shows

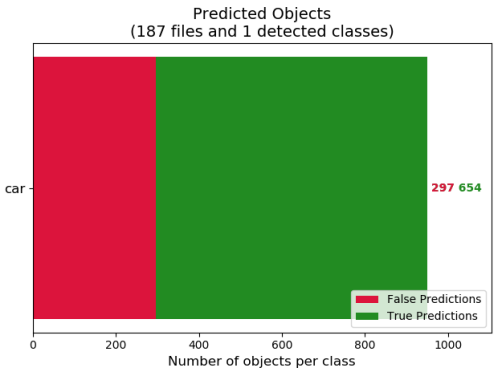

Figure 8: Tiny-yolov3 prediction results

the detailed mAP results for SSD-Mobilenetv1 and SSD-Mobilenet2. Both models perform the best on detecting large objects (96x96 bigger) and their performance is very close. However, it decreases for medium objects (between 32x32 and 96x96) to around half for SSD-Mobilenetv1 (32.03%) and to 24.31% for SSD-Mobilenetv2. For the small objects (less than 32x32) both models have bad results with 7.2% for SSD-Mobilenetv1 and only 3.73% for SSD-Mobilenetv2.

In conclusion Tiny-yolov2 in our case is the best model to choose when it comes to mAP and the actual detection. And although SSD-Mobilenetv2 is promoted as an improved version from the first it gives lower results onb all mAP levels.

For our project how good the detection is not the only criteria that will enable us to choose the best model because the real-time aspect and the latency of the detection are determinant of the best model.

4.3.3 Models performance and latency.

To test the performance of each model, we tried not only to inspect how long it takes to perform object detection on a images and how many frames per second can each model work on but also to have a more detailed preview of how long each task of the detection is taking. Therefore, we measured the capture time (the time to load the image) , the preprocessing time, the actual inference time and the postprocessing time. We performed this on both CPU and GPU. The results of the experiments are shown in Figures 9 and 10. Obviously the object detection rubns a lot faster on GPU than on CPU. So the detailed results on GPU aren't very visible on FIgure 9. That's why we included the Figure 10 that shows the results only on GPU.

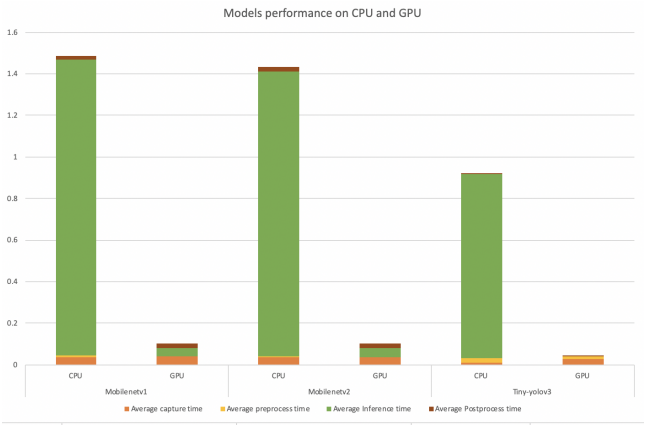

Figure 9: Models performance on CPU and GPU

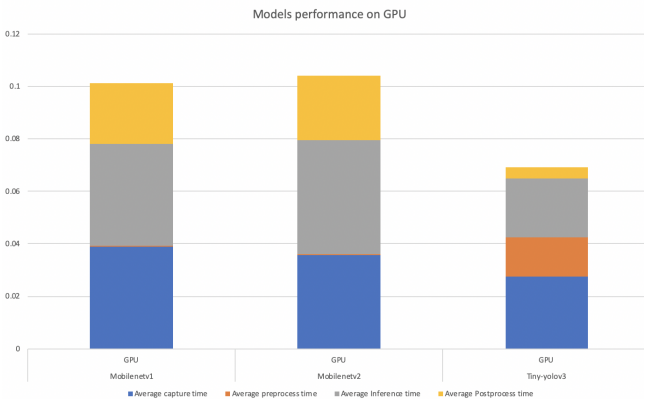

Figure 10: Models performance on GPU

The first finding is that on CPU the inference time is the one taking around 90% of the total time.

Tiny-yolov3 outperforms the other two models on both CPU and GPU. On CPU, it is around 1.55 times faster than the other models (0.934 seconds). And SSD-Mobilenetv2 is slightly faster

Table 3: Performance on CPU

	SSD-Mobilenetv1	SSD-Mobilenetv2	Tiny-yolov3
Average detection time (s)	1.488	1.431	**0.934**
Frames per second (FPS)	0.762	0.698	**1.07**

Table 4: Performance on GPU

	SSD-Mobilenetv1	SSD-Mobilenetv2	Tiny-yolov3
Average detection time (s)	0.101	0.104	**0.0739**
Frames per second (FPS)	9.870	9.626	**13.538**

than SSD-Mobilenetv1 on CPU (1.431 seconds compared to 1.488 SSD-Mobilenetv1)

On GPU however, SSD-Mobilenetv1 performs slightly better than SSD-Mobilenetv2 (0.101 seconds compared to 0.104 for SSD-Mobilenetv2). But Tiny-yolov3 is again winning the race by being around 1.4 times faster than the two other models with 0.0739 average total detection time and 13.538 frames per second. However, on GPU, for Tiny-yolov3, the part that is taking most of the time (even more than the inference time) is the loading part. This may be due to the fact that the loading is on done on the machine's CPU instead of GPU.

To conclude, performance wise, Tiny-yolov3 is by far the best model to choose from among the 3 models. Also, SSD-Mobilenetv2 is in theory faster than SSD-Mobilenetv1 however it only performs slightly better on CPU but is slightly slower on GPU.

5 FUTURE WORK

As for future work, these models are designed to work on mobile and edge devices and more likely in no-GPU environments. Therefore it would be interesting to deploy the models on mobile devices or Edge TPU to see how their performance get impacted and if the best performing model that we have so far will be as good in those constrained environments as it was on CPU and GPU.

Also a possible improvement . is to focus more on the Tiny-yolov3 models, since it's the best performing on both the accuracy and detection level and on the performance level to see if there are ways to improve the obtained results (for example try to reduce the images loading time on GPU

6 CONCLUSION

In this project our goal was to choose object detection models that were designed to work on mobile and edge devices that have less storage and less computational power than computers.

We focused mainly on 3 models SSD-Mobilenetv1, SSD-Mobilenetv2 and Tiny-yolov3 and we trained them from scratch on our IndyCar images data. The results of the 3 models were good, but Tiny-yolov3 outperformed the other two models on all levels (mAP and detection, perfromance and detection time and even training time).

Compared to YOLOv3, Tiny-yolov3 has a lower mAP (because the network is less complex) but it can run inference on images around 20 times faster than the original version.

One of the biggest challenges of this project was to find implementations with dependencies and packages that can run together without any issues and flexible enough to train on our specific data. In fact there are many implementation but that will only run on pre-set well known datasets like COCO dataset of Pascal VOC Dataset.

ACKNOWLEDGMENTS

To professor Judy Qiu and the Teaching assistant Selahattin Akkas who were always present and always available to help during the implemntation of this project. They provided ressources, guidance and advice and were there to help me solve the different issues I confronted. To all the github and stackoverflow community, without whom I don't think many projects would have seen the light

REFERENCES

[1] Dario Amodei. 2019. AI and Compute. https://openai.com/blog/ai-and-compute/
[2] He, Chih-Yuan Huang, Wei, Li Yunfang, and Guo Anfu. 2019. TF-YOLO: An Improved Incremental Network for Real-Time Object Detection. *Applied Sciences* 9 (08 2019), 3225. https://doi.org/10.3390/app9163225
[3] Matthijs Hollemans. 2017. Google's MobileNets on the iPhone. https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/
[4] Matthijs Hollemans. 2018. MobileNet version 2. https://machinethink.net/blog/mobilenet-v2/
[5] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[6] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).
[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.
[8] Jonathan Pedoeem and Rachel Huang. 2018. YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers. *arXiv preprint arXiv:1811.05588* (2018).
[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.
[10] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.
[11] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).
[12] Robert J Wang, Xiang Li, and Charles X Ling. 2018. Pelee: A real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems*. 1963–1972.
[13] Alexander Wong, Mahmoud Famuori, Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, and Jonathan Chung. 2019. YOLO Nano: a Highly Compact You Only Look Once Convolutional Neural Network for Object Detection. *arXiv preprint arXiv:1910.01271* (2019).
[14] Nick Zeng. 2018. An Introduction to Evaluation Metrics for Object Detection. https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/