Lightweight and real-time object detection model on edge devices with model quantization.

Jiayi Wang
Frankfurt International School, Frankfurt, Hessen, 61440, Germany

Abstract. Environment perception is of vital importance in autonomous driving as it serves as autonomous car’s “eye” to perceive the surrounding environment. We propose a solution to achieve real-time environment perception on resource constrained edge devices like mobile phones, even though they have limited computation resources. We used float-to-int quantization based on TensorFlow Lite to quantilze the SSD model. TensorFlow Lite achieves accuracy lossless quantization with quantization-aware training by inserting fake quant operator during training, which imitates the loss introduced by quantization in feedforward training. Our experiment shows that superiority of our proposed method, which is highly speed, light weight and relatively high accuracy.

1. Introduction
Since the development of autonomous driving, many companies around the world have announced the launch of their autonomous driving project in the recent decade. As predicted, 10 million cars will be running on the road by 2020. In 10 years fully automated cars will be common and by 2050 autonomous cars will have created about 7 trillion revenue per year. The widespread autonomous vehicles can also lead to a 90% car accident reduction. As car sales and people’s passion for new technology increases day by day, the autonomous car has become a huge market. To regulate and supervise the market, the Society of Automotive Engineers (SAE) developed an autonomous driving grading system in 2014 and rated L0 to L5 with a total of six grades according to the degree autonomous driving. The degree increases from L0 to L5. The center of the autonomous driving industry is solution providers like Google, Baidu, and car manufacturers led by Tesla and BMW. These companies connected upstream with hardware and software providers, and connected downstream with consumers, play a vital part in the entire industry chain. Before the official civil use of autonomous driving, many commercial fields, like logistics and public transport, with relatively limited functions, simple road conditions, or not suitable for manual driving will give priority to full-scale autonomous driving.

The key to autonomous driving is to simulate the process of a human driver driving a car which can be divided into several parts, “environment perception”, “action planning” and “plan execution”. In these three parts, “environment perception”, as the first part, plays a major role in the overall process. It simulates the perception of a human driver and interacts with the external environment to make the autonomous driving system have a better understanding of its surrounding environment.

However, as environment perception needs a lot of computing resources to run, it might need the car manufacture to make special modifications to the car to compatible with the environment perception model. For example, Tesla chose to combine it with electronic cars to satisfy the electricity need for such a model to run. Therefore, if there is a way to implement these models without modification to the car, it would reduce the design cost and broaden the application conditions for the autonomous driving system. Implementing these models on edge devices like mobile phones could be a solution for that. On
one side they have a battery to power themselves. On the other side, they have relatively more computing resources than an on-board computer.

2. Related Work
In this paper, we will implement the environment perception system of an autonomous car on smartphones, using TensorFlow Lite [5]. TensorFlow Lite is the official framework to run inferences with TensorFlow models on edge devices. As we know, running models on edge devices, like the smartphone, has several limits that must be considered during the process of development. Edge devices usually have lower memory and lower computing ability than other kinds of devices like personal computers. In this case, TensorFlow Lite becomes a better choice.

Compared with TensorFlow [4], TensorFlow Lite has been optimized as follows:
1. Compress the model: reduce the size of the model. Therefore, it fits into an edge device.
2. Quantization: The TensorFlow model contains many matrices. The matrix value is of a 32-bit float data type. Quantization is to express these 32-bit floating-point numbers in 8-bit bytes. Therefore, less computation and memory are needed.

On the other hand, TensorFlow Lite is highly portable. It can be used in smartphone operating systems like iOS and Android and can be run and tested in other operating systems like Windows or macOS.

One of the key technologies that the environment perception system of an autonomous car uses is called object detection which can be roughly divided into two main categories: two-stage detector and one-stage detector.
1. The two-stage method, such as the R-CNN [11] and Faster R-CNN[1] algorithm. The idea is mainly to generate a bunch of candidate boxes via a selective search or Region proposal network (RPN), and then continue to regress and classify these boxes. The advantage of these two-stage methods are its high accuracy but low speed.
2. The one-stage method, such as Yolo [2] and SSD [10]. The main idea is densely sampling at different locations of the image. Various aspect ratios and scales are exploited during sampling, and then CNN extractors are used to extract features and further do classification and regression. The whole process is just a one-step, so the advantage is its speediness.

The method we used in this paper is the one-stage method and specifically is SSD. It is fast and needs relatively low computing resources, so it is the best choice for implementing environment perception on edge devices.

With deep learning, convolutional neural networks have become more and more popular. Many network designs like AlexNet [6], VGGNet [7] and GoogLeNet [8] have appeared. The general trend of current development is to obtain higher accuracy through deeper and more complex networks, but this type of network often does not have much advantage in model size and inference speed. However, edge devices like mobile phones have limited computation resources, so they need a lightweight, low-latency (and acceptable accuracy) network model. Therefore, we choose MobileNetsV2[9]. MobileNets V2 is an efficient model for embedded and visual applications. It is based on a simple neural network architecture that uses depth-wise separable convolutions to construct lightweight deep neural networks. The model introduces two hyperparameters to strike a balance between the delay and accuracy value. These two hyperparameters enable the model to choose the appropriate size for its actual application according to the problem constraints.

3. The Single Shot Detector (SSD)
The single-shot detector works as the following steps.

3.1. Features
3.1.1. Detection with multi-scale feature maps. Multi-scale feature extraction means SSD utilizes different scales feature maps. The CNN contains larger feature maps in the earlier network but gradually
reduces in size by using convolution stride 2 or pooling. The advantage is that a relatively large feature map is used to find relatively smaller objects, the small feature maps is used for detecting larger objects. As is shown in Fig. 1, the 8x8 feature maps can be divided into much more units, but the scale regarding default boxes are relatively small.

Figure 1. Feature maps: the figure above shows two example of feature maps, (a) shows a 4x4 feature map and (b) shows an 8x8 feature map. On both feature map, three different size of bounding boxes are used to predict the ground truth.

3.1.2. Convolution for detection. Unlike Yolo [2], which uses a fully connected layer at the last layer, SSD exploits convolutional neural network to extract candidate boxes from different feature map scales. For a feature map of the size m × n × p, just smaller conv kernels like 3 × 3 × p is used to get the detection score.

3.1.3. Default-box Generation. SSD draws inspiration from the idea of anchor in Faster R-CNN [1]. Each anchor unit sets default bounding boxes with various aspect ratios and scales. The predicted boxes are built on these default boxes so that it largely reduces the difficulty of training. Normally, each unit sets multiple default boxes which have different the scale and aspect ratio. For example, in Fig. 1, we can observe that an individual unit uses 3 different default boxes.

3.2. Training

3.2.1. Default box matching
During the training stage, we first decide which default box matches the ground truth boxes in an image. Two main matching principles are available between the default boxes and ground truth boxes. First,
with respect to each single ground truth boxes, we are looking for a default box which has the largest Intersection over Union (IoU) on the ground truth boxes. Then, it guarantees that each single ground truth box can match a certain default box. A default box that is to match the ground truth is called a positive instance. Otherwise, is negative instance. Please note that there are fewer ground truths in an image than the negative instances. If only match with the first principle, too many negative samples would be generated, and the positive and negative samples are extremely unbalanced. Therefore, another principle is required. The second principle is that: for the left unmatched default boxes, if the IoU is greater than a certain threshold (i.e. 0.5), a default box can also be matched with one ground truth. This indicates that a ground truth box may potentially match with multiple candidate boxes that can simplify the learning process.

Although a ground truth can be matched with multiple default boxes, the relative number of the ground truth is too few compared to the default boxes, so there will be many negative samples relative to positive samples. To guarantee the balance between the positive and negative instances, SSD adopts hard negative mining strategy, which samples negative instances and arrange them in descending order w.r.t. the confidence loss(the lower the confidence score of the predicted background, the greater of the loss). And then the top ones with relatively large error is selected as the negative instance to guarantee that the ratio of positive instances and negative instances lies near to 1:3.

3.2.2. Training objective

The final objective loss value is the weighted sum of the confidence loss value and the localization confidence loss value:

$$L(x, c, l, g) = \frac{1}{N} \left( L_{\text{conf}}(x, c) + a L_{\text{loc}}(x, l, g) \right)$$  \hspace{1cm} (1)

Where \(N\) is the positive instance number of the prior box. \(x_{ij}^p \in \{1,0\}\) is an indicator, when \(x_{ij}^p \in \{1,0\} = 1\), it means the \(i\)th default box matches the \(j\)th ground truth, ground truth is in category \(p\). \(c\) is the category confidence predictions. \(l\) indicates the value of position prediction result to the default bounding box, \(g\) is the ground truth’s position parameter. For localization loss, Smooth L1 loss [1] is adopted here as follows:

$$L_{\text{loc}}(x, l, g) = \sum_{i \in \text{Pos}} x_{ij} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(i^m - \hat{g}_j^m)$$  \hspace{1cm} (2)

$$\hat{g}_j^x = \frac{g_j^x - d_j^x}{d_j^w}, \hat{g}_j^y = \frac{g_j^y - d_j^y}{d_j^h}$$  \hspace{1cm} (3)

$$\hat{g}_j^w = \log(\frac{g_j^w}{d_j^w}), \hat{g}_j^h = \log(\frac{g_j^h}{d_j^h})$$  \hspace{1cm} (4)

Here, we are regressing the offsets to center location \((cx, cy)\) of the bounding box \((d)\) and the height and width \((w, h)\). The loss value, which is the SoftMax loss, is defined as:

$$L_{\text{conf}}(x, c) = -\sum_{i \in \text{Pos}} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in \text{Neg}} \log(\hat{c}_i^p)$$  \hspace{1cm} (5)

Where \(\hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}\) and the weighting factor \(\alpha\) is set to 1.

3.2.3. Select the aspect-ratio and scale

The scale of a default box follows a linearly increasing principle. When the feature map reduces, the scale of a default box linearly increases:

$$s_k = s_{\text{min}} + \frac{s_{\text{max}} - s_{\text{min}}}{m - 1} (k - 1), k \in [1, m]$$  \hspace{1cm} (6)
Where \( m \) is the feature map number, \( s_k \) is the ratio between the default box and the image, \( s_{\text{min}} \) and \( s_{\text{max}} \) are the max and min value of the ratio. After imposing different aspect ratio for the default box, which is usually \( a_r \in \{1,2,3, \frac{1}{2}, \frac{1}{3} \} \). The width and height can be computed by:

\[
w^a_k = s_k \sqrt{a_r}, h^a_k = \frac{s_k}{\sqrt{a_r}}
\]

For \( a_r = 1 \), we add another default box whose scale is \( s'_k = \sqrt{s_k s_k + 1} \). Therefore, each feature map has 6 default boxes. And then we center each single default box to \( (\mu, \nu) \), \( i, j \in [0, |f_k|] \), where \( |f_k| \) is the size of the default box.

### 3.2.4. Data augmentation
Data augmentation can be further involved to escalate SSD performance, the core idea include random crop, horizontal flip, color distortion, random sampling (in order to obtain the smaller training samples).

## 4. Quantization
This section explains our process of applying quantization-aware training. Our goal is to quantize the model without explicit accuracy loss. The procedure we are using for quantization is converting float32 to int8.

### 4.1. Quantization scheme
First, we define \( q \) as the quantized value and \( r \) as the real float value. We introduce a mapping relationship to represent it, presenting with formula:

\[
r = S(q - Z)
\]

In equation (8), \( S \) and \( Z \) are constant. \( S \) indicates scale, which is a positive real number and has the same type as \( r \). \( Z \) indicates Zero-point, it has the same type as quantized value \( q \) and can be regarded as a shift zero in real number to zero in quantized number. \( S \) and \( Z \) are called quantization parameters, and there is a pair of such values for each weight matrix and each activation array.

### 4.2. Matrix multiplication
For the multiplications between two floating number matrices, they can be converted into integer operations:

\[
r_a^{(i,j)} = S_a (q_a^{(i,j)} - Z_a)
\]

\[
S_3 (q_3^{(i,j)} - Z_3) = \sum_{j=1}^{N} S_1 (q_1^{(i,j)} - Z_1) s_2 (q_2^{(j,k)} - Z_2)
\]

\[
q_3^{(j,k)} = Z_3 + M \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2)
\]

From equation (8), we know that each real number \( r_i \) in the array represents a real number \( q_i \) with a pair of parameters \( S \) and \( Z \). Then for the real matrix multiplication \( R_1 \) and \( R_2 \), each real number of the result matrix can be expressed as equation (4), where

\[
M = \frac{S_1 S_2}{S_3}
\]

\( M \) is the only value in equation (11) that is not an integer. Experience shows that \( M \) is always in (0,1), so \( M \) can express as:

\[
M = 2^{-n} M_0
\]

Where \( n \) is a non-negative integer and \( M_0 \) is an integer. In this way, real number multiplications become integer multiplications.
4.3. Bias-vectors
The aforementioned section describes the matrix multiplication of weights, but in addition, a neural network also contains the mapping of bias and activation function. Since the result of the int8 type matrix after the calculation should be within int32, so we choose int32 as the type of bias. One of the reasons for this choice is that bias only occupies a small part of the entire neural network. In addition, the role of bias is particularly important. High-precision bias can reduce the deviation of the model. So, after adding bias, it becomes int32, and we need to convert the result back to int8 again, and then enter the activation. As shown in Figure 3 below:

Another way to achieve this is through post-training quantization. As shown in Figure 4, we quantize weight at the beginning before it goes into the convolution layer and quantizes again after activation. This approach is more suitable for large-scale models, and for small-scale models, it will lead to a large accuracy loss.

5. Experiment
To test our proposed method, we conduct an experiment on a publicly available dataset: COCO dataset [12].
5.1. COCO Dataset
The dataset we used to conduct our experiment is COCO-dataset. The COCO is a large-scale dataset mainly for object detection, captioning and semantic segmentation tasks. It aims at scene understanding in daily scenes that we encounter in everyday life. Totally there are 91 categories, 328k images and 2,500k labels. So far, there are 80 object classes and more than 330k images, among which 200k are assigned with a bonding box. The number of object instances is more than 1.5 million. COCO has features as following: (1)Object semantic segmentation (2)Object Recognition in Context (3)Multi-objects recognition per image (4)>300k images (5)>2 million objects (6) 80 object categories (7) five captions on average per picture (8) 250k persons with key-points. This dataset can be used to solve three main problems: object detection, recognition in context, and precise positioning of targets. Although there are fewer categories than ImageNet [12] and SUN [13], more images are here for each category, which is meaningful to explore more capacity for each category.

5.2. Evaluation Metric
We use the following indicators characterize the our method’s performance. Average precision (AP) indicates the average precision of every categories of object. Precision (P) indicates the percentage of the samples which are predicted as positive are truly positive. So, it can be expressed as:

\[ P = \frac{TP}{TP + FP} \]  \hspace{1cm} (14)

Where \( TP \) is true positive, which indicates the results number that are positive after prediction and being truly positive. \( FP \) is false positive, which represent the results number that are positive with prediction but in fact being negative.

Average recall is the average of the recall of every categories of object. Recall (R) indicates the percentage of the positive samples that have been predicted. It can be expressed as:

\[ R = \frac{TP}{TP + FN} \]  \hspace{1cm} (15)

Where \( TP \) is the same as the previous equation and \( FN \) means false negative, which represent the number of positive samples that has been predicted as negative.

Intersection over Union (IoU) is an indicator that indicates how close is the predicted result to the actual target. As its name, it can be calculated by intersection of the bounding box and ground truth boxes over the total union of the two boxes.

5.3. Result
The network architecture we are using in MobileNetV2.

| InputSize | Op    | time | channel | num | stride |
|----------|-------|------|---------|-----|--------|
| 224 \times 3 | Conv-2d | -    | 32      | 1   | 2      |
| 112 \times 32 | b-n   | 1    | 16      | 1   | 1      |
| 112 \times 16 | b-n   | 6    | 24      | 2   | 2      |
| 56 \times 24 b | b-n   | 6    | 32      | 3   | 2      |
| 28 \times 32  | b-n   | 6    | 64      | 4   | 2      |
| 14 \times 64  | b-n   | 6    | 96      | 3   | 1      |
| 14 \times 96  | b-n   | 6    | 160     | 3   | 2      |
| 7 \times 160  | b-n   | 6    | 320     | 1   | 1      |
| 7 \times 320  | Conv-2d | -    | 128     | 1   | 1      |
| 7 \times 1280 | Avg-pool | -   | -       | 1   | -      |
| 1 \times 1 \times 1280 | Conv-2d | -    | k       | -   | -      |
Table 1 shows the structure of MobileNetV2. Each line of the table describes a different type of layer. Each type of layer has an output channel $c$ and a time of repetition $n$. Stride $s$ is the stride of the first layer of each type of layers, and for the rest stride is 1. All spatial convolution uses a kernel of $3 \times 3$. For expansion factor $t$, it is 6 for every layer.

![Convolution layer structure of MobileNetV2. (a) shows the structure of convolution layer with stride of 1. (b) shows the structure of that with stride of 2.](image)

Table 2 Channels/Memory comparison between MobileNetV2 and other relevant neural networks.

| Size       | MobileNetV1 [16] | MobileNetV2 | ShuffleNet [15] (2x, g=3) |
|------------|------------------|-------------|----------------------------|
| 112 x 112  | 64/1600          | 16/400      | 32/800                     |
| 56 x 56    | 128/800          | 32/200      | 48/300                     |
| 28 x 28    | 256/400          | 64/100      | 400/600                    |
| 14 x 14    | 512/200          | 160/62      | 800/310                    |
| 7 x 7      | 1024/199         | 320/32      | 1600/156                   |
| 1 x 1      | 1024/2           | 1280/2      | 1600/3                     |
| max        | 1600K            | 400K        | 600K                       |

Table 2 shows the maximum memory/channels needed for three types of architectures including MobileNetV2 we are using (in Kb). For ShuffleNet we use $2x, g = 3$ so it matches with the performance of MobileNetV1 and MobileNetV2. As we can see MobileNetV2 has a clear advantage on this.

Table 3 MobileNet-V2 and SSD-Lite comparison and other object detection algorithms with COCO dataset.

| Network               | AP    | Param-Num  | M-Add         | Speed  | CPU       |
|-----------------------|-------|------------|---------------|--------|-----------|
| SSD-300 [10]          | 23.20 | 36.1 Million | 35.2-Billion  | -      | -         |
| SSD-512 [10]          | 26.80 | 36.1 Million | 99.5-Billion  | -      | -         |
| YOLO-v2 [17]          | 21.60 | 50.7 Million | 17.5-Billion  | -      | -         |
| MobileNetV1 + SSD-Lite| 22.20 | 5.1 Million  | 1.3-Billion   | 48ms   | 270.0 ms  |
| MobileNetV2 + SSD-Lite| 22.10 | 4.3 Million  | 0.8-Billion   | 19ms   | 200.0 ms  |
As we can see MobileNetV2 + SSDLite clearly has less parameters and multiplication (MAdd) which indicates that it is less complex and being more suitable on edge devices. It also has a relatively high accuracy and an advantage in speed.

Table 4: complexity of SSD and SSDLite Comparison.

|       | Params   | MAdd     |
|-------|----------|----------|
| SSD   | 14.8 Million | 1.25 Billion |
| SSDLite | 2.1 Million | 0.35 Billion |

From table 4 we can see that SSDLite has a significant advantage on the complexity comparing with SSD.

Figure 6 shows some results of our model. These shows that our model can detect small objects in complex real-life situation. For example, the first picture on the third row is rather complex since it contains many small objects overlapping each other. Our model can still recognize most of the targets correctly.

6. Conclusion
In this paper, we proposed a method of running object detection models on edge devices. This is achieved by quantizing the SSD model using float-to-int quantization. An experiment is also provided to show that our proposed method is effective and have practical value. We believe this method could help the implementation of object detection on edge devices and make autonomous driving more common in more cars.

References
[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: towards real-time object detection with region proposal networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15). MIT Press, Cambridge, MA, USA, 91–99.
[2] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time
Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[3] Girshick, R.: Fast R-CNN. In: ICCV. (2015)

[4] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: a system for large-scale machine learning. In Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation (OSDI’16). USENIX Association, USA, 265–283.

[5] https://www.tensorflow.org/lite

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS’12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

[7] Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. (2015). Going deeper with convolutions. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015, pages 1–9. IEEE Computer Society, 2015.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.

[10] Liu W. et al. (2016) SSD: Single Shot MultiBox Detector. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9905. Springer, Cham. https://doi.org/10.1007/978-3-319-46448-0_2

[11] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.

[12] Lin TY. et al. (2014) Microsoft COCO: Common Objects in Context. In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham. https://doi.org/10.1007/978-3-319-10602-1_48

[13] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. 2016. SUN Database: Exploring a Large Collection of Scene Categories. Int. J. Comput. Vision 119, 1 (August 2016), 3–22. DOI: https://doi.org/10.1007/s11263-014-0748-y

[14] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva and A. Torralba, "SUN database: Large-scale scene recognition from abbey to zoo," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, 2010, pp. 3485-3492, doi: 10.1109/CVPR.2010.5539970.

[15] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. Shufflenet: An extremely efficient convolutional neural network for mobile devices. arXiv:1707.01083 [cs.CV]

[16] 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. CoRR, abs/1704.04861.

[17] Joseph Redmon and Ali Farhadi. 2016. Yolo9000: Better, faster, stronger. arXiv preprint arXiv:1612.08242.