Binary Neural Networks for Object Detection

by

Yizhou Wang

to obtain the degree of Master of Science in Embedded Systems
at the Delft University of Technology,
to be defended publicly on Thursday August 29, 2019 at 4:00 PM.

Student number:	4740629	
Project duration:	October 1, 2018 – August 29, 2019	
Thesis committee:	Dr. ir. Z. Al-Ars,	TU Delft, Chair
	Dr. A. van Genderen,	TU Delft
	Dr. W. Pan,	TU Delft
	B. Zhu, MSc,	TU Delft

This thesis is confidential and cannot be made public until August 31, 2020.

TU Delft

Abstract

In the past few years, convolutional neural networks (CNNs) have been widely utilized and shown state-of-the-art performances on computer vision tasks. However, CNN based approaches usually require a large amount of storage, run-time memory, as well as computation power in both training and inference time, which are usually used on GPU based machines to ensure the speed for inferences. But they are usually insufficient to be deployed on low-power applications. Although many approaches were proposed to compress and accelerate the CNN models, most of them were only evaluated on relatively simple problems (e.g. image classification), which only support limited real-world applications. Especially, among those methods, binary quantization can achieve very high model compression, but only a few works have been observed to utilize it on more complex tasks. Therefore, the exploration and evaluations of applying binary quantization on more complex tasks like object detection are worthwhile, which can be used in much more applications like autonomous driving and face detection. In this project, we apply and evaluate two different binary quantization approaches, named ABC-Net [32] and PA-Net on object detection tasks. Also, we specify the exact implementation details for the binary convolutional operations in this project. As a result, we can achieve maximally 6.1× (around 16% of the full-precision model) compression, and minimal 2.5% accuracy reduction for weight quantization. The weight quantized models were able to outperform some existing real-time detectors in terms of both accuracy and storage size. Although large accuracy reduction was observed for input quantization, the quantized model could still maintain an acceptable accuracy compared to existing real-time object detectors.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Dr. Zaid Al-Ars for the help and guidance during the project. Thanks to Dr. Wei Pan for sharing inspiring ideas about this project. Furthermore, thanks for Baozhou Zhu for all his great help and discussions during the whole project.

I would also like to thank the Quantum & Computer Engineering department, TU Delft for providing the compute servers used to run the experiments of this project. And thanks for all of my colleagues for the inspiring discussions.

Last, but certainly not least, many thanks to my family for their support during not only the project, but also the study in TU Delft, and throughout my life.

Yizhou Wang
Delft, August 2019

1 Introductions

1.1. Context

Deep neural networks (DNNs) have achieved great successes in various research topics such as computer vision, natural language processing, speech recognition, etc. Among those topics, computer vision tasks such as image classification [40], object detection [13, 31], and semantic segmentation [13, 31] have attracted increasing research interests due to the potential in a wide range of real-world applications like autonomous driving, human-machine interaction, etc.

Although, in the past few years, convolutional neural networks (CNNs) have been widely utilized and they have shown state-of-the-art performances on computer vision (CV) tasks. Those CNN based approaches require a large amount of storage, run-time memory, as well as computation power in both training and inference time. For instance, popular CNN models in image classification like the single-precision floating-point ResNet-101 [22] consume 158 MB parameters storage and over 300 MB memory bandwidth to feed-forward only one image, and the consumption is even bigger for VGG-16 [41], which requires 490 MB for parameters storage and over 800 MB memory bandwidth during inferences. For more complex tasks like object detection, the size of the models [33, 39], and the run-time memory utilization will be even larger. Moreover, as those large CNN models do not fit in on-chip storage, they will require a large amount of costly DRAM access as well as a large amount of computation resource for dot products, which will make the power consumption easily exceeding the budget for normal embedded devices [19]. Thus, in practice, most of the CNN based systems are running on machines equipped with powerful GPUs. Those CNN based systems cannot usually be deployed on cheaper machines with strict memory, power, and execution time constrains (e.g. smartphones, wearable devices, drones, etc), as they could easily run out of the computation resources.

To increase the scalability and to reduce the cost, research to compress and accelerate the CNN models have drawn growing attention in both the industry and in academia. Many approaches were proposed to compress and accelerate the deep CNN models, such as connection pruning and weight sharing [2, 21], low-precision/binary quantization [32, 35, 45], code-book based compression [5, 19], and the combination of multiple compression technique [19]. Almost all of those approaches were only proved and evaluated for image classification problems, where the system only needs to determine one label for each input image. However, the majority of vision systems are required to provide more information than image classes: multiple labels for all the objects appeared in the single input image respectively, and even the locations of those objects. This kind of problem, known as object detection and instance segmentation, have been the primary focus in most of CV applications (e.g. autonomous driving, drones, pose estimations). The models [33, 39] are made more complex to ensure satisfactory performance in those problems. Despite the fact that there is significant demand for model compression of those complex CV tasks, only a few results have yet been reported in the literature, especially for binary quantization, see Chapter 2. Thus, further evaluations and improvements of those compression approaches on relatively complex CV tasks are needed.

1.2. Problem description and research questions

Most of the existing CNN quantization methods [8, 10, 19, 32, 35] can largely reduce the network size in image classification tasks, but they will also result in quantization errors and will further lead to performance reduction in terms of accuracy. Especially, binary quantization of neural networks usually gives the largest compression by quantizing the floating-point parameters to only 1 bit (up to 32x compression for single-precision and up to 64x compression for double-precision models), while it could result in a relatively high quantization errors. Although, researchers have shown that it is possible to compress the models with the classification accuracy being preserved [8, 10, 35] in major image classification data set. And attempts have been made to make the classification performance tend to be lossless after binary quantization with more accurate quantization methods [32]. However, only a few published results have been reported about the model compression on object detection, especially for binary quantization. Recently, a large accuracy reduction was observed for binary quantized CNN in object detection tasks even with very limited compression [42], see Chapter 2. Meanwhile, the newly proposed binary quantization scheme PA-Net by Zhu et al. from our groups was also only designed and tested on image classification tasks, further evaluations on complex tasks are also valuable.

In short, the goal of this project can be described using the research questions listed below:

- Is it possible to utilize binary convolutional neural networks for object detection tasks (i.e. will the training phase converge towards a usable model)?
- How much accuracy and compression can we achieve on object detection tasks using binary quantization? And how can we make some trade-off between compression rate and accuracy?
- Evaluate the newly proposed PA scheme on object detection tasks and compare it with other state-of-the-art binary quantization approaches.

To reach its goal, this project can be divided into the following parts:

- **Literature Survey:** Study the existing neural network architectures for object detection. Meanwhile, the existing approaches for network compression, especially, the existing attempts for binary quantization in object detection should be reviewed.
- **Design and implementation:** Design the binary neural networks for object detection based on the existing approaches.
- **Experiments and Evaluation:** Evaluate and compare different approaches on the major date set for object detection.
- **Analysis and Improvements:** Analyze the results of the experiments, and make further improvements based on it.

1.3. Thesis outline

The main objective of this project is to verify the possibility of utilizing binary quantization on CNNs for object detection tasks. To be more specific, binary CNNs with multiple existing quantization approaches as well as a newly proposed CNN quantization approaches, namely piece-wise approximation networks (PA-Net), will be implemented and evaluated for object detection tasks. Furthermore, strategies to construct and train binary neural networks with preserved performance in terms of accuracy for object detection tasks will also be discussed.

This thesis includes seven chapters. In Chapter 1, the context of the project, the project outline and the motivation behind this project will be introduced. In Chapter 2, we will discuss the background for object detection and neural network compression. Different approaches for object detection and binary compression will be discussed and compared. We will also talk about the related works on networks compression for object detection in this chapter. Then in Chapter 3, the implementation details of the baseline networks and the strategies used to train the networks will be discussed. Introductions to the experiment setup, data set, as well as the evaluation methods will also be given in Chapter 3. Then in Chapter 4 and 5 we will discuss the detailed methodologies for ABC [32] and PA quantization, respectively. Furthermore, experiments and evaluations will also be presented in Chapter 4 and Chapter 5. Based on the results from the previous chapters, in Chapter 6 further discussion and the comparison of different approaches will be provided. And further compression of the binary models obtained from the previous chapters as well as the trade-off between compression and accuracy will be discussed. Finally, in Chapter 7, we will summarize the project and give suggestions for future work.

2
Background and related work

2.1. Neural networks for object detection

In this section, we will first introduce the underlying knowledge of convolutional neural networks, and then the background information of object detection problems as well as the state-of-the-art approaches to handle it will also be discussed.

2.1.1. Convolutional neural networks (CNNs)

a) CNN architectures

CNN is a kind of neural networks in deep learning, which are widely used in visual recognition problems. Comparing to other visual recognition methods, such as SVM classifiers, CNN can handle the input image to the desired result end-to-end without any human-designed features. Typical CNN consists of the combination of the following components:

Convolutional layer: The main components of a typical CNN, which aims to collect the local information from the previous layer and map them to higher-level features in the later layer. Different from the convolutions in signal and image processing, the convolution operations in CNN are sliding dot products between the inputs and the filter kernels, where each filter kernel will slide over the input image to calculate the dot products and the result of each sliding window position will be mapped as a single element (or feature) on the output feature map.

Non-linearity layer: An non-linearity layer, or activation layers is normally added after a convolutional layer, which aims to add non-linearity to the model. In activation layers, typical non-linearity function will be used to map the element in the input feature map to the activation map element-wisely.

Pooling layer: Pooling layers are responsible to downsample the activation maps, which are normally added after multiple convolutional layers. Similar to the convolutional layers, pooling layers also collect the information from the inputs using sliding windows, and all the elements at a single window-position will be downsampled to one element on the output feature map. The goal of pooling is to gradually reduce the computations as well as to reduce the probabilities of over-fitting when the networks go deeper.

Fully connected layer: Typically, fully connected layers are added at the end of a CNN to connect the hidden layers and the output layer. The parameters in fully connected layers are trained to summarise the features and map them to a vector with the element representing the score of each class.

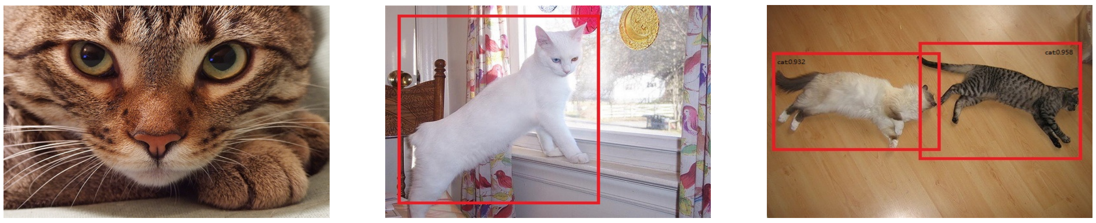
In CNN, the convolutional layers usually takes most of the computation resources, due to a large number of floating-point multiplications between the high-dimension convolutional kernels and the inputs, which are the bottleneck of execution speed for most of the typical CNN models.

b) Overview of CNNs for large scale image recognition

Researches to improving the performances of CNN in computer vision tasks have attracted great concerns since AlexNet achieved significant successes on ImageNet large scale classification challenge in [28]. Simonyan et al. utilized smaller 3×3 convolution kernels and 2×2 maximum pooling in VGG networks [41], which improved the classification performances with over 3 times deeper architectures than AlexNet. Although VGG showed convergences with the depth up to 19 layers, He et al. found an unexpected performance reduction when increasing depth of the network, even without overfitting [22]. Addressing this issue, He et al. proposed residual neural networks, which utilized shortcut connections to take previous layers' information into accounts. As a result, ResNet was able to converge with over 1000 layers, and ResNet also showed further increases in classification accuracy on ImageNet. In the past few years, there were some other attempts, to improve both the accuracy and the efficiency for image recognition problems, such as DenseNet [25] and MobileNet [24]. Those CNNs are not only used in image classification tasks, but also in much more complex tasks like object detection and segmentation, which will be introduced in the next section.

2.1.2. Object detection

In computer vision and recognition problems, object detection aims to determine the semanteme and the location of some specific objects in digital images or videos. To be more specific, in object detection tasks, the systems are required to not only give the label, but also the location of the target instances. As shown in Figure 2.1, in practice, such location information is normally presented by giving the coordinates of the bounding boxes, and the regions which contain the target objects is called foreground, while the regions without targets are named background. Furthermore, different from typical classification problems, which treat the whole image as one single instance, object detection systems need to handle multiple instances with uncertain quantities and locations at the same time. Thus, it is easy to see that guessing the correct results for object detection is way more difficult than the classification problem [13]. But it is also true that detection can fit the needs of more applications than classification.



(a) **Image classification** (b) **Classification+Localization** (c) **Object detection**

Figure 2.1: Example for different tasks: **(a) Image classifications:** whole Image as one instance, only class information needed. **(b) Classification+localization:** Only one object, but both class and location information are needed. **(c) Object detection:** Multiple objects with unknown quantities, both classes and information are required.

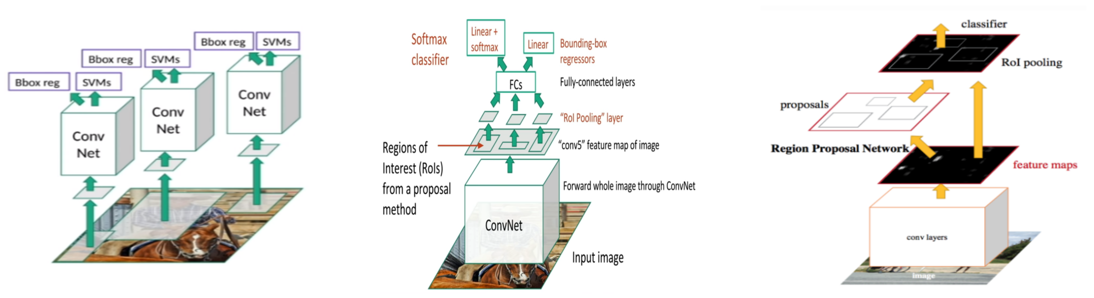
The methods to solve the object detection problem can be divided into hand-engineered features based machine learning approaches, or CNN based deep learning approaches. For the machine learning-based approaches, the idea is to carefully design and extract the features from the images [12, 34, 44], and then design the classifiers (e.g. support vector machine) based on them. However, in recent years, deep learning-based approaches [33, 38, 39] have achieved a significant performances' improvement comparing to those machine learning methods, and have been used in a wide range of real-world applications, such as face detection, human pose estimation, etc. Another advantage of deep learning-based methods is that the feature extraction can be handled by typical convolution and fully connected layers, which makes the whole process end-to-end during both training and inference time.

2.1.3. Object detection networks

There are mainly two types of neural networks for object detection: the region proposal based two stages networks [11, 39], and the one-stage networks [33, 38]. Both of these two kinds of networks consist of the backbone networks and head networks. The backbone networks act as feature extractors, where the feature maps will be extracted from input images or video frames in the backbone networks. And the head networks are normally added after the backbone networks to summary the features and generate the corresponding labels as well as the bonding box coordinates of the target objects. Normally, the backbone networks are classic CNN from classification tasks, such as Alex Net [28], VGG Net [41] and ResNet [22]. In practice, to ensure good performance, the networks should be deep and wide enough, and thus, they need to be trained on richer data sets. However, it is much more costly to label the images for object detection than classification, so the training set in object detection is typically smaller than that in classifications. Hence, the backbone networks are usually trained on large scale data set for classification tasks. (e.g. ImageNet classification [40]), and fine-tuned together with the head networks on detection data sets.

a) Region based two-stages networks

As introduced earlier in this chapter (see Figure 2.1), the difference between object detection and classification is that detection tasks are required to not only classify the type of the targets but also determine the location of them in a single image. The main difficulty is that normal classification systems cannot figure out the regions where they should perform the classifications, and thus, it is inevitable to first 'manually' locate all the possible targets before applying a typical classifier. To do that, the most direct way is to use the sliding window approach, where we need to define a group of sliding windows with different sizes and aspect ratios, and feed all the sub-regions cropped by the sliding windows to the classifiers to determine the classes. However, in object detection tasks, the number, location, size, aspect ratio of the targets are unknown, and thus, to figure out the target objects, a huge number of sub-regions sampled by sliding windows need to be fed into the CNN, which makes it extremely computational expensive! Addressing this issue, instead of naively collect the regions using sliding windows, different region proposal methods were presented to efficiently estimate the regions where the targets might exist [43], which lead to a considerable reduction on the number of sub-regions. Then the proposals (or region of interests) generated by region proposals can be fed into the classifiers for further classification.



(a) **R-CNN** (b) **Fast R-CNN** (c) **Faster R-CNN**

Figure 2.2: Architecture of R-CNN series: **(a) R-CNN [17]:** whole Image as one instance, only class information needed. **(b) Fast R-CNN [16]:** Only one object, but both class and location information are needed. **(c)Faster R-CNN [39]:** Multiple objects, both classes and information are required.

R-CNN Series: With the region proposal approach, R-CNN (R stands for regions) was proposed by Girshick et al. [17] in 2014, known as the first CNN based approach which successfully leaded to large performance improvement on object detection tasks. The idea of R-CNN is first to generate the region proposals using selective search [43], then all the proposals are fed into CNN separately for feature extractions, and finally the SVM classifier is applied on the top of the feature maps extracted by CNNs for classifications. Meanwhile, shown in Fig.2.2, as the region generated by the region proposal algorithm, is only acted as a rough estimation of the target location, to ensure more accurate localization, budding boxes regression is performed together with the network parameters updating during training time. However, R-CNN is still very slow to run (47 seconds per image with VGG-16 as backbone network [16]). Because, the expensive convolution operations will be performed separately for thousands of region proposals in each image to obtain the feature map for each proposal, and those feature maps will cost hundreds of gigabytes during training [16], which make R-CNN completely expensive to train in both time and storage issues.

To accelerate the detection processes, Fast R-CNN was proposed by Girshick et al. in [16]. To reduce the computation cost, instead of obtaining the proposals directly on the input image in R-CNN, the input image will be first fed into the backbone CNN for feature extractions and the region proposals are collected from the feature maps generated by the backbone CNN. As shown in Figure 2.2, Fast R-CNN significantly reduced the computation cost by simplifying the over thousand times of forwarding propagation of the region proposals to only one forward propagation of the original input image. As a result, Fast R-CNN achieved over 200x speed up and higher detection accuracy comparing to R-CNN [16].

Although the computation cost of the convolution operations was largely reduced by sharing the convolution layers, computing the region proposals from the feature map was still time-consuming, which dominated the computation time of Fast R-CNN [39] (over 85% of the computation time for one image). Addressing this issue, Ren et al. proposed the Faster R-CNN in [39], which aims to further improve the speed and accuracy. Similar to Fast R-CNN, Faster R-CNN also utilized the shared backbone convolution layers to extract the feature map from the input image, but instead of using selective search, a novel region proposal network (RPN) was used to generate the region proposals with pure convolutional layers. The idea was to add a sub-network consists of several convolution layers on the top of the backbone CNN, which took pieces of the feature map as input and generated the proposals as output based on foreground/background classifications and bounding boxes regressions [39]. The generated proposals were then reshaped to the same size through RoI (region of interests) pooling, and the final labels and the locations for each proposal were determined by performing classification and bounding box regression separately through the head networks. As the sub-network is fully convolutional networks, it makes the whole Faster R-CNN an end-to-end process and can be largely benefited by the utilization of GPU paralleling. As a result, the RPN approach enabled a quick region proposal in around 10ms per image using GPU [39]. Meanwhile, Faster R-CNN also shows state-of-the-art detection accuracy even in nowadays.

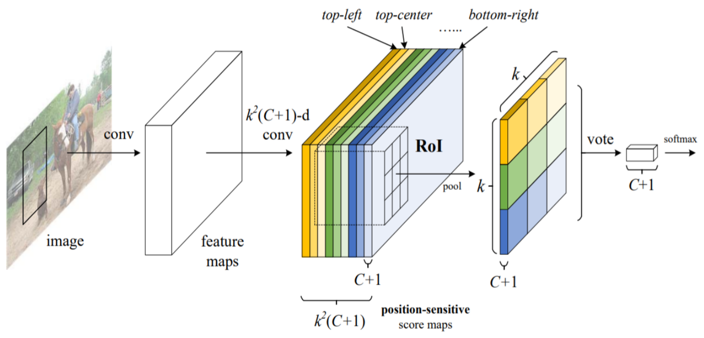


Figure 2.3: R-FCN Architecture [11]

R-FCN: Another region-based two-stages model, named region-based fully convolutional networks (R-FCN), was proposed by Dai et al. in [11] to further accelerate Faster R-CNN by sharing more computation. Similar to Faster R-CNN, R-FCN also used RPN to generate proposals. However, in

Faster R-CNN all the proposals were fed independently into the following head networks after the RoI pooling, where costly convolution operations were performed multiple times for each input image. Thus, in R-FCN, Dai et al. removed the head networks after RoI pooling and used novel position-sensitive score map and average voting layer instead to calculate the final detection results. This extra position-sensitive score map and voting layers only took negligible overheads, so almost all of the computations were taken in convolutional layers. Consequentially, each input image can pass through the whole R-FCN with almost all computations being shared. As a result, R-FCN run 2 to 2.5x faster than Faster R-CNN during inference time, with a small reduction in detection accuracy.

b) One Stage Networks

Although attempts were made to accelerate the region proposal process (e.g. Faster R-CNN) for the two-stages networks, the region-based two-stages networks were still computationally expensive. Thus, one stage models without the region proposal were proposed for more efficient object detection.

YOLO: Redmond et al. proposed a unified object detector YOLO (You only look once) [38], which aimed to achieve real-time object detection with GPUs (e.g. 45 frames per seconds). YOLO detector also regards the object detection task as regression plus classification problem, which takes the input image and output the regressed bounding boxes as well as the targets' labels. But different from the two-stage approaches, the input image will be only looked once, which means the images can be directly feed through a single forward pass without the overhead to calculate region proposals. As a result, YOLO can run in 45 fps (frame per seconds) during inference time, and the fast version YOLO with less convolutional layers can even run in 155 fps. However, as the cost of the speed, YOLO showed a large accuracy reduction compare to Faster R-CNN. The trade-off for speed and accuracy had been made by the authors in the following version of YOLO in [36, 37].

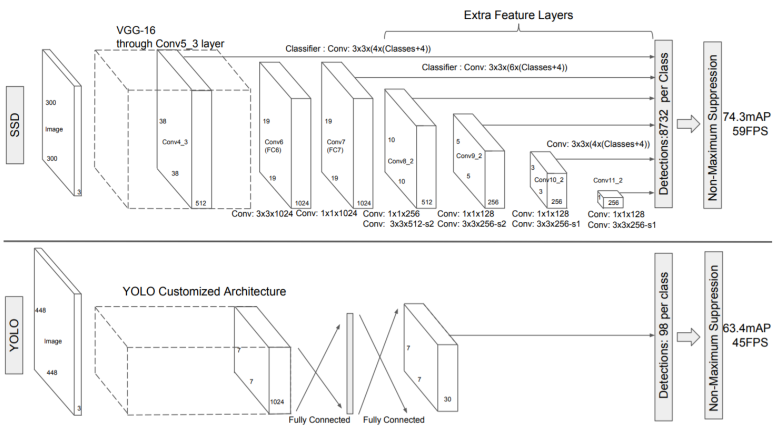


Figure 2.4: YOLO and SSD Architectures [33, 38]

SSD: Single-shot detector (SSD) was published by Liu et al. in [33], aiming to solve the accuracy reduction in real-time single-stage detectors. The main resource of the accuracy reduction is that detectors like YOLO were suffering from localizing objects in different scales, especially for small objects [33, 38], which can be mitigated by using region proposals like the two-stages model. Addressing this issue, SSD uses a multi-boxes scheme to take into account not only the feature map from the last convolution layer but also features from the shallower layers. The authors figured out that the feature maps in shallower layers will contain more information for smaller objects, and in deeper layers, the larger object will be represented better. Thus, by combing the features from different layers, the detector can handle objects in different scales and sizes even without region proposals. As a result, SSD was able to achieve 59 fps with preserved detection accuracy.

c) Comparison

In conclusion, Table 2.1 shows a comparison of the modern object detectors introduced in this section. Among those models, R-CNN was the first successful attempt to utilize CNNs in object detection tasks, where the efficiency of R-CNN was limited by the unnecessary repetitions of convolution operations as well as the region proposal process. Fast R-CNN largely boosted the detection speed by sharing the computation of most of the convolution layers for the whole input image, but the slow region proposal process then became the bottleneck. Addressing this issue, in Faster R-CNN, RPN was proposed to accelerate the region proposal step. And as a result, Faster R-CNN showed a significant speed-up as well as state-of-the-arts accuracy comparing to the previous R-CNNs. Inspired by Faster R-CNN, R-FCN showed higher speedups with an accuracy reduction by sharing the computations of not only the backbone network but also the head network.

On the other hand, one-stage models showed much higher speeds, but relatively lower accuracy. YOLO and Fast YOLO were able to achieve real-time detection by removing the region proposal processes and combining the object localization with classification in one single forward pass. While, YOLO showed unsatisfactory performances to detect the object in different scales, especially for small objects. Comparing to YOLO, SSD [33] introduced a novel multi-layer anchor scheme to take multi-level feature maps into account, which boosted the performance to detect objects in different scales. Consequentially, SSD was able to run in real-time with competitive accuracy on GPUs during inference time.

Table 2.1: Comparison for modern object detectors

Two-stages Models	**Localization**	**Computation Sharing**	**Accuracy**	**Speed**
R-CNN	Selective Search	No sharing	+	+
Fast R-CNN	Selective Search	Shared backbone	++	++
Faster R-CNN	RPN+anchor	Shared backbone	++++	+++
R-FCN	RPN+anchor	Share all	+++	++++
One-stage Models				
YOLO	Anchor	Share all	++	+++++
Fast YOLO	Anchor	Share all	+	++++++
SSD	Multi-layer anchors	Share all	++++	+++++

2.2. Overview of deep neural network compression

Although those CNN based systems are very powerful in computer version tasks, a large amount of storage, memory, and computational power consumption are required to run those models. Thus, in practice, most of the CNN models are trained deployed on machines with powerful GPUs. To increase the scalability of those CNN models on cheaper machines, several approaches have been presented in recent years, which will be briefly introduced in this section.

2.2.1. Low-precision and binary quantization

In practice, the parameters of CNNs models are usually represented by 32-bits single-precision floating-point numbers or 64-bits double-precision floating-point numbers. Those CNN models are largely accelerated by using GPUs, which are very efficient to handle floating-point operations. However, on normal CPUs or embedded devices, floating-point operations are usually very costly. Modern CNN models, such as ResNet-50 (3.6 billion FLOPs per image [22]) and VGG-19 (19.6 billion FLOPs per image [22]) could easily run out of the computational resource on those machines. In low-power embedded applications, one straightforward but effective way to reduce the cost is cutting down the number of FLOPs with fix-point representations. In [7, 18], CNN models with 32-bits and 16-bits fix-point representation were proved to be able to work on image classification tasks with negligible accuracy decreases. Courbariaux et al. showed that was also possible to train the networks with low-precision fix-point representation [9]. Further approach with 8-bits representation for both parameters and inputs was proposed and evaluated on ImageNet [45]. To the extreme case of low-precision representation, another trend was to quantize the network parameters and even the inputs to only 1-bit binary number [8, 10]. However, those methods resulted in large reductions in classification accuracy. Further explorations were made to improve the accuracy, such as ternary quantization [1], adding scaling factors [32, 35], using the linear combination of multiple binary bits [35, 45]. The detailed comparison of different binary quantization approaches will be discussed in the next section.

2.2.2. Code-book based quantization

Code-book based quantization aims to quantize each specific network parameter to an element from the code-book, where the code-book is a finite set consisting of real numbers. As a result, all the original parameters in the models will be replaced by the elements from the code-book, and therefore the model size can be largely reduced. There are several different ways to encode the parameters. For instance, Chen et al. extend hashing method to DNNs in [5] by randomly separating the network parameters into different hash groups and encoding the values in the same group to one specific number. Chen et al. further improved the performance of the compressed models by grouping the weights from the frequency domain's point of view in [6]. Han et al. applied Huffman encoding in CNN [19], and achieved a large reduction in network size.

2.2.3. Connection pruning

Connection pruning reduces the computational complexity directly by figuring out and removing certain redundant connections from CNNs. In [3, 29], attempts were made by adding sparsity constraints to the neural network. Lebedev et al. further introduced group-sparsity regularization during training time with group-wise brain damage method, which simplified the convolutional operations to the multiplications of spare matrices. To reduce the efforts to represent the pruned networks with irregular connections, as well as to make the pruned network better fiting the parallel computing system, Anwar et al. proposed structured pruning method in [2]. Structured pruning added sparsity at different scales, such as channel-wise and kernel-wise sparsity. Han et al. proposed a three-stages pruning approach in [20]. They first trained the network to figure out the essential connections, then they removed the redundant connection from the network, and finally, they performed training on the compressed network again. Han et al. also proposed a deep compression method in [19], which significantly reduced the size of the networks by performing connection pruning, low-precision quantization, and Huffman encoding simultaneously to the networks.

2.3. Related works in binary neural networks

2.3.1. Binary neural networks on image classification

Table 2.2 illustrates the comparison among different binary quantization methods. In theory, for convolutional layers, the maximum compression achieved with binary quantization is 32× and 64 × for single-precision and double-precision floating-point numbers respectively. For instance, BC (BinaryConnect) [10] simply quantized the weights of the convolutional parameter to 1-bit $\{-1, 1\}$ representation, which has the highest compression among all methods. And BNN (binarized neural network) [8] further quantized the input images to simplify the costly floating-point arithmetic (e.g. multiplications, additions) to bit operations like XNOR and bit count. BNN was able to achieve significant compression as well as dramatic speedups on image classification tasks. However, these early quantization methods like BC and BNN showed large reductions in classification accuracy (comparing to the full precision AlexNet on ImageNet classification, approximately 20% and 30% decreases with BC and BNN respectively were reported in [35]).

To solve the accuracy issues, Rastegari et al. proposed a more accuracy approach named BWN (binary weight networks) [35]. BWN used an extra scaling factor α for each convolutional kernel aiming to better represent the full precision weights. To be more specific, during inference time, each parameter in the convolutional kernel was represented by $\{-\alpha, \alpha\}$ instead of $\{-1, 1\}$ in BC. The shifting parameters were determined by solving regression problems during training time, and they were saved together with the binary weights for further inferences. The shifting parameter only took an overhead of 32-bits for each convolutional kernel, which was negligible for CNN models with over a hundred megabytes. Similarly, in XNOR-Net [35], the scaling factors were utilized to represent the binary inputs with $\{-\alpha, \alpha\}$ with small overheads, while the convolutional operations in the proposed XNOR-Net were still maintained in XNOR and bit count form. Therefore, BWN was able to improve $\sim 17\%$ accuracy comparing to BC on ImageNet classification, and XNOR-Net was able to improve $\sim 25\%$ accuracy comparing to BNN [35].

Another trend to improve the accuracy was using ternary representations with $\{-1, 0, 1\}$ to quantize the parameters. Li et al. proposed TWN (ternary weight networks) [30] by using this $\{-1, 0, 1\}$ representation for the weight. For ResNet-18, TWN increased around 2% accuracy on ImnageNet classification comparing to BWN. Also, the ternary neural network (TNN) with both ternary inputs and ternary weights was proposed and evaluated on FPGA in [1]. However, to represent the ternary weights or inputs with $\{-1, 0, 1\}$, 2-bits numbers are required, which results in relatively large overheads comparing to binary quantization approaches.

Moreover, further methods were proposed to enable trade-offs between compression and accuracy. For instance, Zhou et al. proposed DoReFa-Nets, which used combinations of multiple-bits binary numbers to represent both the weights and inputs. The number of bits used to quantize the model with DoRaFa-Net scheme can be flexibly defined, and thus the compression of DoReFa-Net can be defined as $k \times \{-1, 1\}$ with $32/k\times$ memory and storage-saving, where k means k-bits quantization. Another important contribution of DoRaFa-Net is that it can be also trained with lower bit-width gradients, which not only accelerate the inferences but also the training processes.

Aiming to further reduce the quantization errors, Lin et al. proposed an accuracy-toward scheme, named ABC-Net (Towards accurate binary convolutional network) to train binary networks in [32]. Similar to XNOR-Net, ABC-Net uses scaling factors defined by regressions. However, instead of only 1-bit in BNN and BWN, ABC-Net uses the linear combination of flexible multiple-bits to represent the convolutional kernel and the inputs, and ABC-Net uses one specific scaling factors for each bit. In other words, the weights of ABC-Net can be represented as $\{-\alpha_i, \alpha_i \| i \in [1, k], i \in \mathbb{Z}\}$, where k means k-bits quantization and α_i is the scaling factor for each bit respectively. Similarly, the inputs in ABC-Net scheme was represented by $\{-\beta_i, \beta_i \| i \in [1, m], i \in \mathbb{Z}\}$, where β_i is the scaling factor for each quantized bit of the inputs. With such approach, ABC-Net can achieve almost lossless accuracy: within 2% accuracy loss on ImageNet classification with 5-bits quantization when use quantized weights and full precision inputs, and within 6% accuracy loss on ImageNet classification with 5-bits quantization when use quantized weight and inputs. As the cost of high performances, the compression of ABC-Net is relatively low (e.g. ~6.4×for 5-bits ABC-Nets). The detailed approach of ABC-Net will be discussed in Chapter 4.

With the similar motivation to ABC-Net an unpublished approach proposed by our group, named PA-Net.[1] (piecewise approximation network). Similar to ABC-Net, PA-Net also uses a flexible multiple-bits quantization scheme. However, instead of using the factor αs to scale the convolutional kernel as whole [32], PA-Net scheme approximate both the convolutional kernels and inputs by looking on the elements' level. To be more specific, PA-Net will divide each element into the corresponding pieces by the numerical value, and one specific scaling factor will be tuned to scale the elements in each piece. Thus by using such element-wise quantization, PA-Net scheme can better represent the convolutional kernels and the input images, even with the same compression($\{-\alpha_i, \alpha_i \| i \in [1, k], i \in \mathbb{Z}\}$, where k means k-bits quantization) as ABC-Net scheme. Comparing to the full precision models, PA-Net

[1]PA-Net was first proposed by Zhu et al. from computer engineering group from EEMCS faculty, TU Delft.

achieved approximately 1% accuracy dedication for ResNet in different depths, which can outperform ABC-Net on image classification tasks. The detailed methodologies of PA-Net will be discussed in Chapter 5.

Methods	Weights	Inputs	Operations	Memory
Full precision	F	F	+, - , ×	1
BC [10]	$\{-1,1\}$	F	+, -	~32×
BNN [8]	$\{-1,1\}$	$\{-1,1\}$	xnor,bitcount	~32×
BWN [35]	$\{-\alpha,\alpha\}$	F	+,-	~32×
XNOR [35]	$\{-\alpha,\alpha\}$	$\{-\beta,\beta\}$	⊙,bitcount	~32×
TNN [1]	$\{-1,0,1\}$	$\{-1,0,1\}$	and,bitcount	~16×
TWN [30]	$\{-\alpha,0,\alpha\}$	F	+,-	~16×
DoReFa [45]	$k\times\{0,1\}$	$k\times\{0,1\}$	+,-,x,and	$32/k$
ABC [32]	$\{-\alpha_i,\alpha_i \|\| i\in[1,k], i\in\mathbb{z}\}$	$\{-\beta_i,\beta_i \|\| i\in[1,m], i\in\mathbb{z}\}$	+,-,⊙,bitcont	$32/k$
PA^2	$\{-\alpha_i,\alpha_i \|\| i\in[1,k], i\in\mathbb{z}\}$	$\{-\beta_i,\beta_i \|\| i\in[1,m], i\in\mathbb{z}\}$	+,-,and, bitcont	$32/k$

Table 2.2: Comparison for different quantization approaches. Where **F** means 32-bits floating numbers, and **k** means k-bits quantization. **Memory** represents the memory or storage saving for convolution operations. *and*, ⊙, and *bitcount* are bit operations representing AND, XNOR, and bit counting, respectively.

2.3.2. Related work of binary neural networks for object detection

As also introduced in Chapter 1, although many approaches were proposed for binary quantization of DNNs and CNNs, almost all of them were designed and evaluated for image classification problems. However, only a few results were reported about the binary quantization based model compression for object detection tasks, which are much more complicated comparing to image classification.

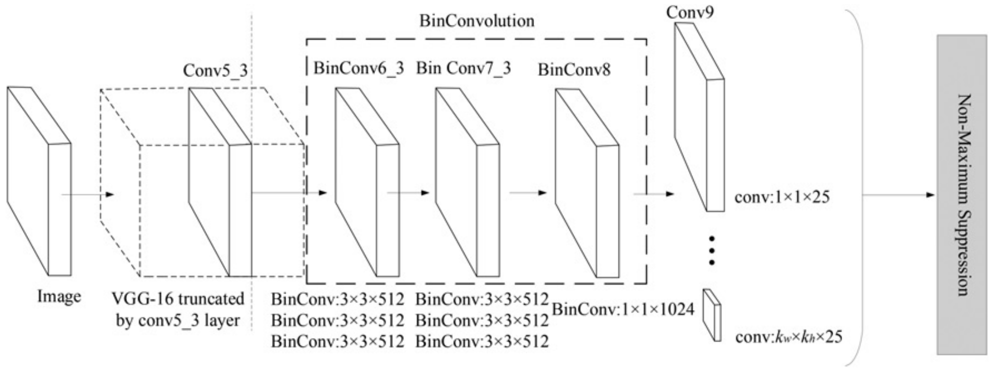


Figure 2.5: The unified network proposed in [42]

In [42] Sun et al. proposed a unified network for object detection and quantized their network based on binary quantization approach. The proposed one stage unified object detector can be shown in Figure 2.5 below. Their proposed network also utilized the backbone and head-network based architecture, and similar to [33, 39], VGG-16 pre-trained on ImageNet classification tasks was chosen as the backbone network. Four additional convolutional layers are added on the top of the backbone network as the head network. Similar to other one-stage object detectors, region proposal stages are not used, and all the convolutional operations except $Conv9$ are shared by the whole image in one single forward pass. In the last convolution layer ($Conv9$), convolutional kernels with different sizes from 1×1 to $kw\times kh$ are used to generate the features for object classification and bounding box regression[42]. During training time, the mean square error is used for box regression, and the class scores are obtained by calculating the softmax losses [42]. As multiple different convolutional kernels are utilized to generate the final features, the total losses are calculated by accumulating the losses from all the kernels [42].

Based on the proposed unified network, Sun et al. proposed a partially quantized network for further memory saving and speedups, where the head network and the last convolutional layer are kept in full precision and the convolutional layers, as well as the inputs in the head network from $Conv6\_3$ to $Conv8$, are binarized with approaches introduced in [35]. However, with such an approach, dramatic accuracy reductions were observed from their experiments on PASCAL VOC data set, where the mean average precision(mAP) dropped from 68.9% to 44.3% [42].

The result of [42] illustrates that, binarizing both the weights and inputs of CNN resulted in significant performance reductions on object detection tasks, even with the majority of the parameters in the network kept in full precision. This also showed that training binary CNNs for object detection tasks is much more difficult than for classification problems. Thus, further exploration and improvement for binary networks on object detection are consequential for practical applications.

3 Solution strategy and experiment setup

In this chapter, the implementations of the baseline full-precision object detectors will be discussed. And the experimental setups, as well as the data set used to evaluate the models will be described. The metrics to measure the model performances will also be included. Finally, we will talk about the strategies used to train the models during the experiments.

3.1. Experiments details

The main experiments of the project were performed on the GPU server of computer engineering group, EEMCS faculty, TU Delft with one NVIDIA K40 GPU. The models are implemented with Python. And in this section, we will introduce the data set used to train and evaluated the models, as well as the evaluation methods used in the experiments.

3.1.1. Data set

In this project, all the models are trained and evaluated on the data set of PASCAL VOC (visual object classes) challenge [13], which is one of the most popular data set used to evaluate the models for object detection problems. The contents of the data set for PASCAL VOC challenges were changed each year from 2005 to 2012. Typically, most of researches [11, 33, 38, 39] for object detection problems only adopted the data sets from 2007 (PASCAL VOC2007), 2012 (PASCAL VOC2012), or the combination of PASCAL VOC2007 and PASCAL VOC2012.

The PASCAL VOC2007 challenges contained totally 9,963 images for both image classification and object detection problem. And 24,640 objects within 20 different categories were annotated for the object detection problem, such as a person, car, airplane, etc. The whole data set can be divided into the train and validation set (VOC07 $trainval$) and the test set (VOC07 $test$). To be more specific, the $trainval$ set include 5,011 images with 15,662 labeled objects, and the $test$ set include 4,952 images with 8,978 labeled objects. Similarly, PASCAL VOC2012 consists of the $trainval$ set with 11,540 images for 27,450 objects, and non-public $test$ set with approximately equal targets comparing to VOC2012 $trainval$, where the target objects are within the same 20 categories as PASCAL VOC2007. The $test$ sets must keep strictly independent to the training processes.

In the researches for object detection during recent years, there were two most widely used ways to evaluate the models on PASCAL VOC data set, which will also be adopted in this project: a) train the models on PASCAL VOC2007 $trainval$, and evaluate on PASCAL VOC2007 $test$, b) train the models on PASCAL VOC2007 + PASCAL VOC2012 $trainval$ and evaluate them on PASCAL VOC2007 $test$. The former option was the standard way used in 2007 challenges, and the latter was chose to evaluate the models with much more training samples.

3.1.2. Evaluation metrics

Evaluating an object detector is much more complex than evaluating an image classifier because both accuracies for classification and localization should be considered. In researches and practical applications, mean average precision (mAP) is the most widely used metrics to evaluate object detectors, in the sense of both localization and classification accuracy. But before the mean average precision, we will first introduce precision, recall, and intersection over union, which will be used in mAP calculations.

a) Intersection over union (IoU)

Intersection over Union (IoU) is the metric to measure the overlapping between two areas, and in object detection, it is usually used to measure the overlapping between the areas bounded by the predicted box and the ground truth box. The calculation for IoU is illustrated in Equation 3.1, where A_p and A_{gt} represent the predicted and ground truth areas respectively. The value of IoU is between 0 to 1, and in object detection tasks, high IoU means high overlapping between predicted boxes with ground truth boxes and represents high localization accuracy.

$$IoU = \frac{Area(A_p \cap A_{gt})}{Area(A_p \cup A_{gt})} \tag{3.1}$$

b) Precision and recall

Before talking about the precision and recall, we should first introduce four notions, named true positive, false positive, true negative, and false negative, which are usually mentioned in information retrieval and searching systems:

- **True positive (TP):** In object detection, TP typically represents the objects being detected which should indeed be detected (the prediction matches the ground truth). Normally, the prediction is said to match the ground truth if the IoU between them is larger than one threshold (e.g. threshold=0.5 is used for PASCAL VOC).
- **False positive (FP):** The objects being detected by the detector, but should not be detected (the prediction does not match the ground truth). Normally, the prediction is considered to be FP if the IoU between them is below the threshold.
- **True Negative (TN):** The instances not being detected, which should not be detected.
- **False Negative (FN):** The instances not being detected, but should be detected. For example, the undetected ground truth.

It is important to note that, typically, TP, FP, TN, and FN are independent for each object class. Based on the four notions, precision is defined in Equation 3.2, which measures the ability of models to identify relevant targets. The precision is defined as the fraction between the correct detection and all detection in object detection problems.

$$precision = \frac{correct\,detection}{all\,detection} = \frac{TP}{TP + FP} \tag{3.2}$$

However, in object detection, only the precision cannot clearly describe the performances of the models. Because precision only measures the correctness of the detection, while the undetected contents will not be concerned. Thus, the recall is defined and utilized simultaneously with precision, as shown in Equation 3.3. Recall measures the ability of models to figure out the relevant objects, which is defined as the fraction between the correct detection and all the ground truth.

$$recall = \frac{correct\,detection}{all\,ground\,truth} = \frac{TP}{TP + FN} \tag{3.3}$$

c) Mean average precision

One good way to evaluate the performance of an object detector is to combine the precision with recall for each specific object class. The object detector is considered to have a good performance if for all object classes, the precision maintains in a high level with the increase of the recall, which means that the object detector has good abilities to both retrieve the relevant target (high recall) as well as identify the target (high precision). However, in practice, object detectors usually suffer from the precision deduction, when trying to figure out all desired targets (or ground truth) to achieve high recall. Because to retrieve all ground-truth objects, the object detectors should increase the number of detected objects (positive), which will cause the increase of false-positive objects, and consequentially the decrease in precision. This suggests that considering the precision on single recall value is usually not enough.

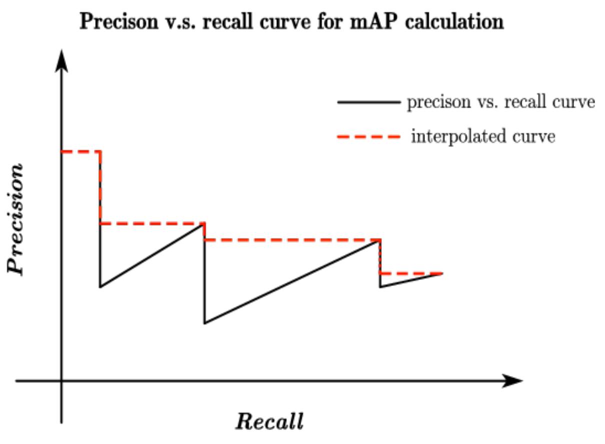


Figure 3.1: One example for precision v.s. recall curve and the AP calculation. The mAP will calculated using the areas under the interpreted precision v.s. recall curve (red).

Thus, in PASCAL VOC challenge [13], instead of one precision and recall, the average precision (AP) for all recall values was used to measure the overall performance of an object detector on all the different recall levels. In PASCAL VOC standard [13], the average precision can be obtained by calculating the areas under the precision v.s. recall curves. One example of the precision v.s. recall curves and the AP (area under curve) calculation used in PASCAL VOC data set [13] can be found in Figure 3.1. Note that all the precision and recall here are for each particular object class separately. To evaluate the performance for all the object classes, the mean average precision can be obtained by calculating the average of APs over all the classes, shown in Equation 3.4. N_{cls} is the total number of classes. r is the unique recall value for each data point. And $\rho_{interp} = max_{\tilde{r}>r_{n+1}}\rho(\tilde{r})$, where $\rho(\tilde{r})$ is the corresponding precision for recall $\tilde{r}$.

$$mAP = \frac{1}{N_{cls}} \sum_{N_{cls}} \sum (r_{n+1} - r_n)\rho_{interp}(r_{n+1}) \tag{3.4}$$

It is also important to note that in PASCAL VOC standard, the IoU threshold used to determine the true positive is set to 0.5. This threshold is different in other data set. For example in COCO [31] set, the AP for multiple IoU thresholds (from 0.5 to 0.95 with a step size of 0.05) will be considered, and the overall mAP will be obtained by calculate the average for all those thresholds.

3.2. Implementation of full precision baseline networks

As introduced in Chapter 2, different object detectors were proposed to improve the performance in terms of both detection speed and detection accuracy. In general, among those object detectors, region-based two-stage models tend to achieve higher accuracy and batter ability to be generalized with different backbone architectures [26]. However, those region-based methods like Faster R-CNN and R-FCN were observed to require more computations (e.g. FLOPs, memory usages) comparing to the single-stage model like SSD [26, 33]. Thus, in practice, most of the visual recognition systems are based on single-stage approaches. Those two-stages models usually are very difficult to be deployed on low-power machines due to the power and memory constraints. Therefore, the exploration to compress those accurate but heavy models are worthwhile for real-world applications. And for single-stage like YOLO and SSD, although they can run in real-time with the support of GPUs, further compression to reduce the computational cost for low-power devices is still worthwhile.

Due to the reasons discussed above, in this project, we first chose the Faster R-CNN as one of our baseline model, which is one of the most accurate but slowest approaches among all the state-of-the-art object detectors. And the SSD model was also chosen as another baseline to explore further run-time resource reductions, as it showed equal or even higher performance compared to Faster R-CNN in some experiments [26].

3.2.1. Implementation for Faster R-CNN

a) Backbone network

Figure 3.2 illustrates the architectures of the full-precision Faster R-CNN baseline network. Instead of using VGG network as shown in the implementation in [39], we choose to use ResNet as the backbone networks for our Faster R-CNN implementation. There are several reasons to use ResNet for the backbone. Firstly, existing researches in [22, 26] showed that ResNet can better extract the features from the input images, and as a result, ResNet successfully achieved higher accuracy for both the classification and the object detection tasks comparing to VGG based networks. Secondly, ResNet was proved to be relatively robust to the error resulted by binary quantization in previous works of binary networks for image classification in [32, 35]. Lastly, ResNet is computationally more efficient than VGG networks: VGG-16/19 nets require 15.3 and 19.6 billion FLOPs respectively to feed one single image while even the deepest 152 layers ResNet-152 only takes 11.3 billion FLOPs for single image [22]. Thus, the ResNet based Faster R-CNN tend to be more accurate and computational cheaper comparing to the VGG based implementation in [39].

In this project ResNet in different depths will be used as the backbone for the baseline full-precision Faster R-CNN networks. To be more specific, similar to [23], we adopt ResNet [22] from $conv1$ to $conv4\_x$ to extract the feature maps as the inputs of the RPN head for region proposals generating, which is shown in Figure 3.2. And we adopt $conv5\_x$ on the top of the ROI pooling layer, which will be fine-tuned to calculate the feature map of each generated region proposal (or region of interests) for final object classifications. All the backbone ResNet were pre-trained on ImageNet classification and will be fine-tuned together with the head network end-to-end during training time.

b) Region proposal network

Shown in Figure 3.3, the implementation of the region proposal network(RPN) is kept the same as that proposed in original paper [39]. Firstly, a 3×3 convolutional layer is attached to the top of the feature maps extracted from $conv4\_x$. Then two branches are used for the foreground/background classification and the preliminary bounding box regression respectively. 1×1 convolution is used in both the of the branches. For the foreground/background classification branch, each pixel in the outputs from the last 3×3 convolutinal layer will be mapped to $2k$ two-classes scores (representing the foreground and the background respectively) with the 1×1 convolution, where k is the number of anchors defined before training. And the final class of each anchor will be determined by the softmax classifier. Similarly, this 1×1 convolution layer is also used in the bounding boxes branch, which maps each pixel from the outputs of 3×3 convolutinal layer to k bounding boxes coordinates in 4 dimensions, and the preliminary bounding box regression will be performed to optimize the coordinates for better

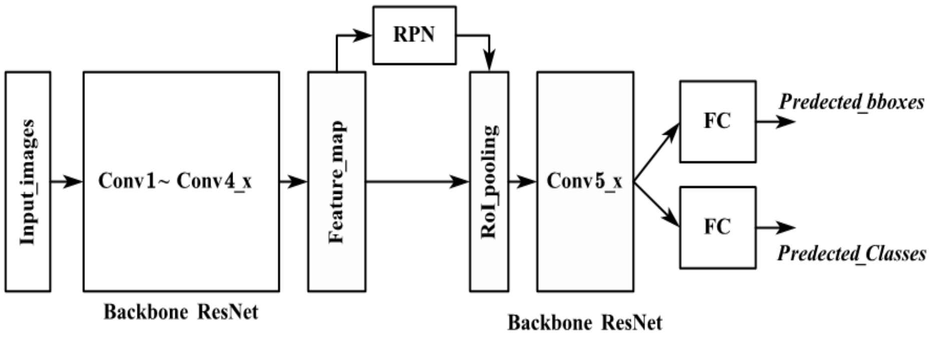


Figure 3.2: The implementation of the baseline full-precision Faster R-CNN: $Conv1$ to $Conv5\_x$ are pre-trained ResNet blocks [22]. **FC** stand for the fully connected layers. RPN is the region proposal network with the same implementation in [39]. And **RoI_pooling** is the region of interests pooling layer used to collect the proposals and map them to the same size for further convolutional operations. *Predicted_bboxes* and *Predicted_Classes* are the outputs of the network for bounding boxes predictions and the target class predictions respectively.

localization. Finally, before generating the proposals, non-maximum suppression will be utilized to filter the overlapping boxes [39].

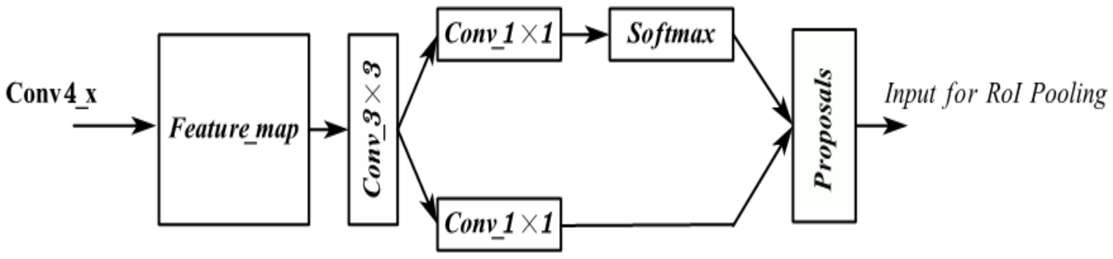


Figure 3.3: The Implementation of the region proposal networks in Faster R-CNN[39]: The RPN takes the feature map extracted from $conv4\_x$ and generate the region proposals by applying foreground/background classification and bounding boxes regressions with a fully convolutional form.

3.2.2. Implementation for SSD

Similarly, ResNet in different depths are chosen as the backbone networks for the SSD networks instead of the original implementation with VGG nets in [33] due to the same reasons discussed above. It is important to note that the original SSD added several extra plain convolutional layers (e.g.VGG like convolutional layers) on the top of the backbone networks to extract the feature maps in different levels. However, as discussed in the ResNet paper [22], such plain-style CNNs showed a large accuracy degeneration when the networks become deep. Thus, we cannot just simply apply them for feature maps extractions when using a deep backbone network. For instance, a ResNet-50 based SSD contains over sixty convolutional layers, and ResNet-101 based SSD contains approximately 120 convolutional layers, where it is very likely to suffer from the accuracy degeneration issues discussed in [22]. Thus, instead of using the plain convolutional layers, extra residual blocks are used to extract the feature maps for SSD on the top of the ResNet backbone networks. This implementation was also used in [15].

Figure 3.4 showed the implementation of SSD network for the experiments in this project. Similarly to the Faster R-CNN implementation above, $Conv1$ to $Conv5\_x$ from the pre-trained ResNet models are used as the backbone network for SSD. And we apply residual blocks as the extra feature layers on the top of the backbone network and we keep the number of feature maps the same as the original implementation [33]. To be more specific, 3×3 convolutional kennels are applied as the classifiers for

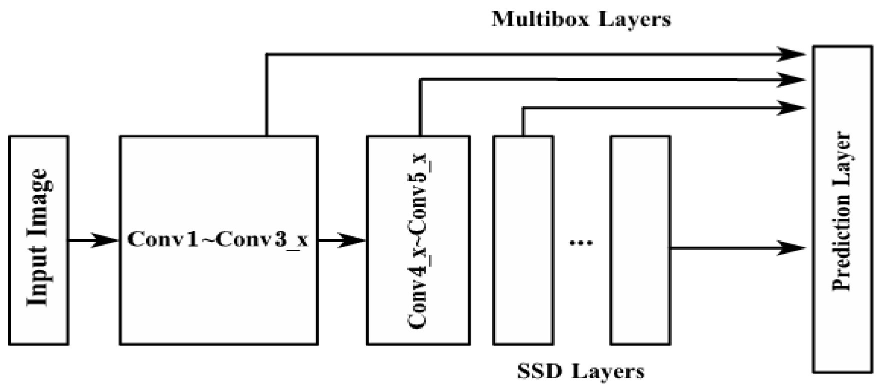


Figure 3.4: The Implementation of the baseline full-precision SSD network: $Conv1$ to $Conv5\_x$ are pre-trained ResNet blocks [22]. Feature maps extracted from $conv3\_x$, $conv5\_x$ as well as all extra SSD layers will be used as multiple box features for the final prediction.

feature maps collected from the different layers, and the size of each feature map is also kept totally the same as the original implementation in [33] and [14], which is extracted from $Conv3\_x$, $Conv5\_x$ and all the extra SSD layers respectively.

3.2.3. Training the baseline full precision networks

a) Faster R-CNN

For both Faster R-CNN and SSD, the backbone ResNet networks were pre-trained on ILSVRC data set [40]. And for Faster R-CNN, instead of the multiple-stages training process [39], we fine-tuned the RPN together with pre-trained backbone network end-to-end with the joint multi-task loss function based approach introduced in [39] and their official code. The loss function for the end-to-end training is showed below:

$$L = L_{head} + L_{rpn} + L_{reg} \tag{3.5}$$

The L_{head} and L_{rpn} represent the final prediction losses and the region proposal losses respectively, and L_{reg} is the regression losses to prevent over-fitting. L_{head} and L_{rpn} are the same as the loss function used in the original implementation [16, 39], which includes the multi-task loss for object classification and localization. For the classification losses, the softmax losses over the foreground/background classes are used for RPN loss L_{rpn}, and the softmax losses over all the object classes are used for the head loss L_{head}. The softmax loss (or cross-entropy loss) function is shown in Equation 3.6 below. N_{cls} is the number of classes (2 for L_{rpn}, and total number of classes for L_{head}). $y_{x,l}$ is the indicator in $\{0, 1\}$, $y_{x,l} = 0$ when the prediction x is correct for ground truth label l, or $y_{x,l} = 1$ otherwise. $p_{x,l}$ is the probability for prediction x on class l.

$$L_{class} = -\sum_{l}^{N_{cls}} y_{x,l} \log(p_{x,l}) \tag{3.6}$$

While, for bounding box regression loss, smoothed L1 loss (or equally Huber loss) is applied for both the head and RPN loss functions. The smooth L1 loss can be shown in Equation 3.7, where l and g represent the predicted and ground truth bounding boxes coordinate respectively. The final box loss for each proposal is the sum of the smoothed L1 losses for the four coordinates calculated by Equation 3.7 [16].

$$smooth\_L_1(l-g) = \begin{cases} 0.5(l-g)^2 & \text{if} |l-g| < 1 \\ |l-g| - 0.5 & \text{otherwise,} \end{cases} \tag{3.7}$$

During training, Faster R-CNN models are trained with the momentum optimizer with the momentum of 0.9, weight decay of 0.0005, learning rate of 0.01 with a 0.1 decay factor, and mini-batch size of 1, which is the same as [39].

b) SSD

On the other hand, as SSD networks do not contain the region proposal network, the SSD models can be trained with one Multi-boxes loss function, which is the same as loss function in the original paper [33]. The specific Multi-box loss function [33] is shown in Equation 3.8 below.

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)) \tag{3.8}$$

N is the number of anchor boxes. L_{conf} and L_{loc} are the classification confidence (or score) and localization loss respectively. The confidence loss function L_{conf} takes the ground truth labels of the targets (c) and the predicted classes (x) as inputs and calculate the multi-class softmax losses to obtain the confidence for the predictions of input images. To be more specific, to calculate the losses during training, SSD will first figure out the positive (objects) and negative (backgrounds) anchors by matching every anchor box to the ground truth bounding box. The anchor is set to positive when the IoU between the anchor box and gourd truth box is larger than 0.5. The confidence function L_{conf} takes both positive and negative anchors into account during training. As the same as in [33] the confidence loss function for SSD is shown in Equation 3.9. N is the number of anchor boxes, and $y_{i,j}$ is the binary indicator, which is set to 0 when the prediction for ith anchor equals to the jth ground truth label. p and 0 represent the object label and background label respectively. As a result, Equation 3.9 calculates the softmax loss for the class predictions in all positive anchors and also penalizes the loss according to the confidence in the background class 0.

$$L_{conf} = -\sum_{l \in Pos}^{N} y_{i,j} \log(c_i^p) - \sum_{i \in Neg} \log(c_i^0) \text{ where } c_i^p = \frac{\exp(x_i^p)}{\sum_p \exp x_i^p} \tag{3.9}$$

And similar to Equation 3.7, the localization loss function takes the ground truth bounding boxes(g), the predicted bounding boxes (l) as well as the class predictions x to calculate the smooth L1 cost between ground truth and predicted coordinates. The hyper parameter α is used to balance the localization and the confidence losses, which is set to 1 determined by cross-validation in [33]. Same as the implementation in SSD paper [33], we trained SSD using momentum optimizer with the momentum of 0.9, weight decay of 0.0005, learning rate of 0.01 with 0.1 decay factor, and a batch size of 32.

c) Performances

Both of the Faster R-CNN and SSD are fine-tuned on PASCAL VOC *train_val* data set and evaluated on PASCAL VOC 2007 *text* set [13] as introduced in last section. The performances of the baseline full precision models can be found in Table 3.1 below.

3.3. Strategies to train binary object detectors

Before we discuss the detailed methods and implementations for binary quantization of the object detectors in next chapter, in this section we will first introduce the strategies to train the binary object detectors.

Detector	Backbone network	Training set	mAP@IoU=0.5
Faster R-CNN	ResNet-18	VOC07	60.057
	ResNet-50	VOC07	65.381
	ResNet-50	VOC07+12	73.432
	ResNet-101	VOC07+12	75.41
SSD-300	ResNet-50	VOC07+12	74.35

Table 3.1: Performances for baseline full precision object dictators. VOC07 means PASCAL VOC2007 train_val set, and VOC07+12 means the combination of PASCAL VOC2007 train_val and PASCAL VOC2012 train_val set. All the results were evaluated on PASCAL VOC2007 test set.

3.3.1. Binary object detector outlines

The previous works on binary quantized convolutional neural networks for both the classification [32, 35], and the object detection [42] have observed non-negligible accuracy reduction after quantization even with some layers kept in 32-bits full-precision numbers. This suggests that the binary networks should be carefully designed. The general strategies used to design binary object detectors are as follows:

- **First convolutional layer and the last layer:** Large accuracy deductions were witnessed in existing works [32, 35] when the networks being fully binarized. However, researchers figured out that higher accuracy can be preserved when keeping the first convolutional layer and the last layer (mostly fully connected layers) in full precision. This design choice was widely applied and proved helpful in binary quantization models in past few years [32, 35].

- **Layers in the head networks:** The layers in the head network of object detectors are usually used to generate the feature maps, where the classification stages are always performed directly on the output of those layers. For example, in the RPN of Faster R-CNN shown in Figure 3.3, the outputs of the $conv\_1 \times 1$ will be used for the foreground/background classification. And in SSD, the outputs of multi-box convolutional layers will be used as the final feature maps for object classification. Thus, similar to the last layer of the whole network, those layers will be kept in full precision.

- **Further trade-off[1]:** Except the layers discussed above, further trade-off between the accuracy and compression can be made by carefully choosing a part of the intermediate layers to maintain in full precision. Or more layers can be quantized for further compression. The trade-off will be discussed in Chapter 7.

Based on the strategies above, for Faster R-CNN, we will first binarize the backbone convolutional layers from $Conv2\_x$ to $Conv5\_x$, but keeping the first $Conv1$ layer, the layers in RPN, and the last fully connected layers for both objects classification and bounding boxes regression in full-precision. On the other hand, for SSD, all the backbone layers except $Conv1$ are binarized, while all the convolutional layers in the head network will be first kept in full-precision.

3.3.2. Training binary object detectors

There are basically two options can be applied to train the binary object detectors, which is shown as follows:

- **Option 1:** The training process can be separated into two stages. For the first stage, the full-precision backbone network for image classification task will be trained on ImageNet. Then for

[1]We will first adopt the first two strategies for the experiments from Chapter4 to 6, and make further trade-off in Chapter7

object detector, the full-precision backbone network, as well as the head network will be fine-toned simultaneously with the binary quantization scheme on the data set for object detection. With this option, the binary object detectors are trained from scratch, and this method was frequently used in existing works to train binary networks for classification [10, 32, 35].

- **Option 2:** This project suggests a different three-stages training process for binary object detectors. Similarly, on stage one, the full-precision backbone network will be trained on ImageNet classification problem. But, then on the second stage, full-precision object detectors will be trained with the strategies discussed in section 3.2.3 and the original papers [33, 39]. Finally, on stage three, we fine-tuned the pre-trained object detectors from stage two with the binary quantization scheme on the object detection data set.

Although option 1 was frequently used in the binary networks for image classification problems, we observed large accuracy degeneration when training the object detectors with this option. Thus, in this project, we adopted the three-stages strategies during training time. As a result, much higher accuracy was achieved by adopting option 2 instead of option 1. Table 3.2 illiterates the experiments performed to compare the two methods. By applying option 2, we achieved approximately 1.5 times higher mAP than using option 1, when training binary Faster R-CNN, based on ResNet-18 and ABC-Net with 5 bits binary representation.

Detector	Backbone network	Method	mAP@IoU=0.5
Faster R-CNN	ResNet-18	option 1	37.17
	ResNet-18	option 2	55.13

Table 3.2: Performances of the Faster R-CNN trained with the two different options. Both of them are based on ResNet-18, and quantized with ABC-Net with 5-bits binary representation. The models are fine-tuned on PASCAL VOC2007 trainval. All the results were evaluated on PASCAL VOC2007 test set.

The reasonable explanation is that by fine-tuning on the pre-trained detectors, the binary object detectors can better approximate the parameters from full-precision models, and therefore it tends to converge easier than option 1 during training time.

4

Accurate binary convolutional networks (ABC-Net)

As introduced in Chapter 2, accurate binary convolutional networks (ABC-Net) is an accuracy-oriented scheme to binarize the convolutional neural networks proposed in [32]. The key idea of ABC-Net is to represent the network parameters and input images with a linear combination of multiple binary numbers. Although ABC-Net requires higher bit-length, it can sufficiently save the accuracy for binarizing image classification models, and therefore ABC-Net is a more suitable approach to be used to check the possibility of unitizing binary quantization on more complex object detection problems.

In this chapter, the detailed ABC-Net based binarization scheme will be discussed. And this chapter further specifies a practical way to perform the binary convolutional operation with ABC-Net scheme. Then, the experiments will be performed to evaluate the binary object detectors quantized with ABC-Net.

4.1. Methodology

The ABC-Net scheme includes both the weights and inputs quantization. And the methodologies for both binary convolution in forward propagation, and the gradients based weight updates in backward propagation were involved. It is important to note that, similar to BWN [35], in ABC-Net scheme, the 32-bits full-precision parameters are used during training time, and the binary weights and inputs are applied only for inference time [32].

4.1.1. Weight quantization

a) Forward propagation

For weights quantization, two options were proposed in [32], which is approximating the weight as a whole, and approximating the weight channel-wisely. As the channel-wise approximation requires a large amount of memory during training, we approximate the parameters as a whole to save the memory during training time.

To be more specific, the full-precision weight for each convolutional layer W can be represented by the linear combination of M binary filters from B_1 to B_M as shown in Equation 4.1. Note that, each binary kernel B_m has the same size with the full precision kernel W, which are 4-D tensors with the channel of layer input c_in, the channel of the layer output c_{out}, the convolutional kernel width w, and the convolutional kernel height h. And each full-precision parameter from W is then represented by the weighted sum of the binary weight from B_m, where the binary element is scaled by the corresponding scaling factor α_m.

$$W \approx \sum_{m=1}^{M} \alpha_m B_m, \text{ where } W \in \mathbb{R}^{c_{out} \times c_{in} \times w \times h} \text{ and } B \in \{-1, 1\}^{c_{out} \times c_{in} \times w \times h} \tag{4.1}$$

The binary kernels B_is are directly mapped from the full-precision kernel W, with Equation 4.2, where u_i is the parameters to shift the weights in range $[-std(W), std(W)]$.

$$B_m = F_{um}(W) = sign(W - mean(W) + u_m std(W)),\ i = 1, 2, \cdots M \tag{4.2}$$

$$u_m = -1 + \frac{2(m-1)}{M-1} \tag{4.3}$$

The shifting parameter u_ms are hyper parameters pre-defined by using Equation 4.3. Or equally, u_ms can be also maintained as a trainable parameters to be defined together with the whole network during weight updates. Then the scaling factor α_ms can be determined by solving the regression problem as shown in Equation 4.4. $B = [v_{B_1}, v_{B_2}, \cdots, v_{B_M}]$ is the matrix consists of the vectorized binary weights v_{B_m}s, where v_{B_m}s have fixed values calculated from Equation 4.2 above. And v_W is the vectorized full-precision weight.

$$\min_{\alpha} J(\alpha) = ||v_W - B\alpha||^2 \tag{4.4}$$

Recap the full-precision convolutional operations in CNNs, each element in the output feature map for the current layer is obtained by the sum of element-wise products (dot products) between the weight kernel and input image shown in Equation 4.5, where $x_{c,w,h}$, $w_{c,w,h}$ and $y_{c',w',h'}$ represent each single element from the input, weight kennel and output feature map respectively. The above ABC-Net binarization can not only reduce the bit length of the parameters but also simply this full-precision convolutional operations.

$$y_{c',w',h'} = \sum_{c,w,h} x_{c,w,h} w_{c,w,h} \tag{4.5}$$

With the ABC-Net weight quantization method, this thesis further specify a practical way to perform the binary convolutional operation to obtain each output element in Equation 4.6, where $signset(.)$ is the bit operation to set the sign of each input element $x_{c,w,h}$ according to the binary weight element $b^m_{c,w,h}$. As the result, the quantized convolutional operation only require M multiplications for one output feature, which largely reduces the computational cost during forward propagation. Meanwhile, the weight kernels can be compressed to $M/32$ of the full-precision kernels, which also saved the storage and the run-time memory.

$$y_{c',w',h'} = \sum_{m=1}^{M} \alpha_m \sum_{c,w,h} signset(x_{c,w,h}, b^m_{c,w,h}) \tag{4.6}$$

For more general overview, the forward propagation for the ABC-Net binary convolutional layers can conclude as in Equation 4.7 below, where $BinWConv$ represent the binary convolution operation with sign-setting. B_ms and α_ms are solved by Equation 4.2 and Equation 4.4 respectively.

$$Y = \sum_{m=1}^{M} \alpha_m BinWConv(B_m, X) \text{ where } B_m = F_{u_m}(W), \tag{4.7}$$

b) Backward propagation

As for the backward propagation, the gradients can be calculated and fed backward with the standard optimiser like SGD (stochastic gradient descent), expect the non-differentiable binary mapping function $B_m = F_{u_m}(W)$ in Equation 4.2. Addressing to this issue, ABC-Net adopts straight-through estimator (STE) proposed in [4] to estimate the gradients flowing through B_ms. Here, the concepts of STE is to copy the gradient with respect to the output directly as an estimator for the gradients of the non-differentiable binary mapping function [4]. Then the specific backward propagation function for the ABC-Net binary convolution layer can be defined in Equation 4.8 below, where C represents the cost during training.

$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial Y}\left(\sum_{m=1}^{M} \alpha_m \frac{\partial Y}{\partial B_m}\frac{\partial B_m}{\partial W}\right) \overset{STE}{\approx} \frac{\partial C}{\partial Y}\left(\sum_{m=1}^{M} \alpha_m \frac{\partial Y}{\partial B_m}\right) = \sum_{m=1}^{M} \alpha_m \frac{\partial C}{\partial B_m} \tag{4.8}$$

It is important to note that, during training, the weights W, and all the gradients are kept in full-precision. But only the binary weights B_ms and scaling factors α_ms will be saved for inferences.

4.1.2. Input/activation quantization

a) Forward propagation

Based on the approach discussed above, with the binary weights, ABC-Net is able to simply the convolutional operation with only a few multiplications, and with significantly reduced model size. However, further speedups can be achieved by also binarizing the inputs of the convolutional layers.

ABC-Net applies the binary activation function to create a binary feature map as the input to next convolutional layers. To be more specific, the first step of inputs quantization is to bound the inputs in the range $[0, 1]$ with Equation 4.9 below, which is similar to [45]. v_ns are additional shifting parameters for the further steps to generate multiple binary bases, and N is the number of binary bases used to represent the full-precision inputs. r_n represents one single bounded element of the nth binary base R_n, generated with the corresponding shifting parameter v_n. And x denotes one single element from the full-precision inputs X of the activation layer.

$$R_n = h(X, v_n) = clip(X + v_n), \text{ for } n = 1, 2, \cdots, N \tag{4.9}$$

where,

$$clip(x) = \begin{cases} x & \text{if } 0 \le x \le 1 \\ 1 & \text{if } x > 1 \\ 0 & \text{if } x < 0 \end{cases} \tag{4.10}$$

The binary activation function then is defined in Equation 4.11, where the activated binary feature maps A_ns are obtained from the activation function $H(R_n)$. And R_n denotes the inputs with all elements rectified with the mapping function in Equation 4.9 above.

$$A_n = H(R_n) = 2I(R_n) - 1, \tag{4.11}$$

$I(.)$ is the indicator function performing element-wise operation for each element of R_n shown in Equation 4.12.

$$I(r) = \begin{cases} 1 & \text{if } r \ge 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

As a result, similar to the weights quantization discussed above, the approximated activation X_{appro} can be represented by the linear combination of multiple binary activations with Equation 4.13, where X_{appro} and A_n represent the approximated activation and binary activation bases respectively with channel c_x, height h_x, and width w_x. β_ns are the scaling factors similar to α_ms for the weight quantization above. Note that, the scaling factors β_ns and the shifting factor v_ns are both trainable parameters, which will be updated during training time. It is important to note that X_{appro} here is just for understanding and discussion. In forward propagation, X_{appro} will not be actually calculated, while the binary activation bases A_ns will be directly used to compute the output of binary convolutional operations, which will be introduced in section 4.1.3 latter.

$$X_{appro} = \sum_{n=1}^{N} \beta_n A_n, \text{ where } X_{appro} \in \mathbb{R}^{c_x \times w_x \times h_x} \text{ and } A \in \{-1, 1\}^{c_x \times w_x \times h_x} \tag{4.13}$$

b) Backward propagation

As also discussed above, during forward propagation, binary activation bases A_ns will be directly used instead of X_{appro}. Thus, during backward propagation, the gradients of the binary activation function will be calculated separately regarding to each base, and they will be accumulated to back-propagate to the shallower layers. The backward propagation function for each binary base is defined in Equation 4.14, where C is the cost calculated through the forward pass, and $\circ$ denotes element-wise product (a.k.a Hadamard product). To be specific, the gradients with respect to the full-precision inputs can be calculated by applying chain role for activation function $A_n = H(R_n)$ though all the N binary bases independently. And similar to Equation 4.8, STE [4] is utilized to approximate the gradient for the non-differentiable binary mapping function $\frac{\partial A_n}{\partial R_n}$ (gradients flowing through Equation 4.11).

$$\frac{\partial C}{\partial X} = \frac{\partial C}{\partial A_n}\frac{\partial A_n}{\partial R_n}\frac{\partial R_n}{\partial X} \overset{STE}{\approx} \frac{\partial C}{\partial A_n} \circ I'(R_n - v_n), \tag{4.14}$$

where $\frac{\partial R_n}{\partial X} = I'(Rn - v_n)$ shown in Equation 4.15 is the gradient for the clipping function $clip(.)$ in Equation 4.10, which is differentiable in $r \in [0, 1]$ but non-differentiable for $r \notin [0, 1]$. STE is only applied for the range where $r \notin [0, 1]$.

$$I'(r) = \begin{cases} 1 & \text{if } 0 \leq r \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

4.1.3. Multiplication-free binary convolutional operation

With the inputs also being binarized, the operations to generate each output with binary weights showed in Equation 4.6 can be further reduced to Equation 4.16, where $a^n_{c,w,h}$ represent the binary inputs of nth base A_n. $signset(.)$ is the bit operation to set the sign which is the same as in Equation 4.6. $xnor$ is the bit XNOR operation used to represent the multiplications between the elements in $\{-1, 1\}$. Furthermore, as the shifting parameters α_ms and β_ns are already defined during training time, the term $\alpha_m\beta_n$ can be pre-calculated and can be directly used with look-up tables. Although no further compression will be achieved, the inputs binarization enables much faster binary operation without any multiplications during inference time. Furthermore, the number of additions can also be reduced, because the summation of binary numbers can be simplified to bit-count operations.

$$y_{c',w',h'} = \sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{c,w,h} signset(\alpha_m\beta_n, b^m_{c,w,h}\ xnor\ a^n_{c,w,h}) \tag{4.16}$$

In conclusion, the overview of forward propagation for ABC-Net based convolutional layers with binary weights and inputs are defined in Equation 4.17, where $BinConv$ denotes the binary convolutional operations without multiplication. Each full-precision element from the inputs and weights is quantized to N and M bits. Binary weight and input bases B_m and A_n are obtained from Equation 4.2 and Equation 4.11 respectively.

$$Y = \sum_{m=1}^{M} \sum_{n=1}^{N} BinConv(\alpha_m \beta_n, A_n\, xnor\, B_m), \text{ where } B_m = F_{u_m}(W) \text{ and } A_n = H(R_n) \tag{4.17}$$

4.1.4. Training

To train the object detector, this project used the same training algorithm discussed in the original paper [32]. During training time, the full-precision gradients are used to update the network parameters. At each training step, only the full-precision weights will be updated, and updated full-precision weights will be converted to binary to calculate the forward propagation in the next step. The ABC-Net scheme based binary object detectors can also be trained with optimizers such as ADAM and Momentum. As the summery, the detailed training algorithm can be found in Algorithm 1 below.

Illustrated in Algorithm 1, $BatchNorm(.)$ and $BackBatchNorm(.)$ represent the forward and backward function for the batch normalization layer [27]. And $BackConv$ is the backward propagation of convolutional layers. $Updata(.)$ is the weight updating methods such as ADAM or Momentum. And $UpdateLR(.)$ denotes the learning rate updating method such as learning rate decay. The algorithm here is based on the training algorithm introduced in [32], but extra branch is added to train the network with only weight quantization.

4.2. Experiments

The previous researches on the performances of the binary CNNs for both classification tasks [8, 32, 35], and object detection tasks [42], showed that using binary inputs/activation will result in a large reduction in accuracies. Because binarizing the inputs will easily destroy the information in original feature maps, especially for the deep convolutional layers whose receptive field could involve the information from over one hundred pixels in the original images. Although the binary models could still be convergent with such crude activation quantization in image classification tasks, over 35% accuracy lost have already been observed on object detection task even with over half of the layers kept in full-precision [42]. It is highly possible that the object detector would even be not convergent with the increase of the quantized layers. Thus, as one main objective of the project, in this section, we start with only quantizing the weights of the object detectors to check the possibilities to utilize binary quantization on object detection tasks. Furthermore, we will also explore the possibility to binarize both weights and inputs with ABC-Net scheme discussed above.

4.2.1. Weight quantization

As introduces in Section 3.3, the three-stages training strategy is used to train the binary weight object detectors. All the backbone ResNet models are trained on ILSVRC12 ImageNet classification data set. And for the second stage, the full-precision models are trained with the setting introduced in Section 3.2.3, which are exactly the same baseline full-precision models.

The third stage for Faster R-CNN models, the full-precision baseline models are fine-tuned with ABC-Net quantization scheme directly on PASCAL VOC data sets. To be more specific, as discussed in Section 3.3, all the layers are binarized, except the first convolutional layer, the last fully connected layer as well as the RPN layers. The binary Faster R-CNN models are also trained with the momentum optimizer with the momentum of 0.9, weight decay of 0.0005, learning rate of 0.001 with decay factor of 0.1, and mini-batch size of 1. As a result, the trained models are also evaluated on PASCAL VOC07 *test* set. The performances of the quantized Faster R-CNN models are shown in Table 4.1 below.

Generally speaking, the binary Faster R-CNNs were able to be convergent based on ResNet backbone networks in different depths, with binary weights and full-precision input/activation. With the in-

Algorithm 1 ABC-Net training algorithm [32]

```
Input: one mini-batch of input images and their ground truth, number of bits $M$ used to represent the weights, number of bits $N$ used to represent the activation/input, the full-precision weight $W$, initialized parameters $u_m$s, $v_n$s, $\beta_n$s, and the learning rate $\eta$.
Output: updated full-precision weights $W$, and learning rate $\eta$ for next mini-batch.
  forward propagation:
  for $l = 1$ to $L$ do
      Compute $\alpha_m^l$ and $B_m^l$, where $m = 1, 2, \cdots, M$ for $l$th layer using Eqn.4.4 and Eqn.4.2;
      if Using binary activation/inputs then
          Compute $lth$ convolutional layer's output $Y^l$ using Eqn.4.16;
      else
          Compute $lth$ convolutional layer's output $Y^l$ using Eqn.4.7;
      end if
      Optionally apply pooling operation;
      Compute batch normalization $X^l \leftarrow BatchNorm(Y^l)$;
      if $l < L$ then
          if Using binary activation/input then
              Compute $A_n^l$ for $n = 1, 2, \cdots, N$ using Eqn.4.11;
          else
              Compute full-precision activation $A^l$ (e.g. ReLU);
          end if
      end if
  end for
  backward propagation:
  Compute $g_{A^L} = \frac{\partial C}{\partial A^L}$;
  for $L = L$ to 1 do
      if $l < L$ then
          if Using binary activation/input then
              Compute $g_{A^l} = \sum_{n=1}^{N} \beta_n \frac{\partial C}{\partial A_n} \circ I'(R_n - v_n)$ based on Eqn.4.14;
          else
              Compute $g_{A^l}$ (e.g. $BackReLU$);
          end if
      end if
      Compute $g_Y^l \leftarrow BackBatchNorm(g_{A^l}, X^l)$
      $g_{\beta_n}, g_{B_n^l} \leftarrow BackConv(g_Y^l, B_n^{l-1}, A_m^{l-1})$;
  end for
  parameter updating:
  for $L = 1$ to $L$ do
      Compute $g_{W^l}$ using Eqn.4.8, with known $g_{B_n^l}$;
      Update $l$th layer's weight $W^l \leftarrow Update(W^l, \eta, g_{W^l})$;
      Update $\beta_n \leftarrow Update(\beta_n, \eta, g_{\beta_n})$;
      Update $v_n \leftarrow Update(v_n, \eta, g_{v_n})$
      Update learning rate $\eta \leftarrow UpdateLR(\eta)$
  end for
```

crease of network depths (or the number of parameters), the performances' gap between full-precision and binary Faster R-CNN tend to be increased correspondingly. The smallest performance gap is

Detector	Backbone	Methods	Training set	mAP(Binarized/FP)
Faster R-CNN	ResNet-18	ABC-3	VOC07	54.13 / 60.06
	ResNet-18	ABC-5	VOC07	55.16 / 60.06
	ResNet-50	ABC-5	VOC07	59.72 / 65.38
	ResNet-50	ABC-5	VOC07+12	67.69 / 73.43
	ResNet-101	ABC-5	VOC07+12	70.31 / 76.29

Table 4.1: Performances for binary Faster R-CNNs comparing to the full-precision baseline. Similarly, the mAP is reported with the PASCAL VOC standard, where IoU threshold is set to be 0.5. And the results is the average of three different initialization. **Binarized** and **FP** represent the binary and full-precision Faster R-CNN respectively. ABC-3 and ABC-5 represent the 3-bits and 5-bits ABC-Net quantization scheme respectively.

achieved by Faster R-CNN with ResNet-18 backbone and 5-bits ABC-Net binarization, which is 4.9 mAP. And the performances for both binary and full-precision models can be improved with the increasing network depths. In other words, the quantized object detector can still benefit from increasing the network depth and the number of parameters, because even the largest mAP gap between binary and full-precision network is still within 6.0 mAP with the deepest 101 layers' ResNet backbone network. Consequentially, the highest accuracy was obtained from the ResNet-101 based binary Faster R-CNN, even with the largest mAP gap between the binary and full-precision models.

On the other hand, for the single-stage SSD models, the same strategies in Section 3.3 were used for the third stage. Specifically, all the layers in the backbone networks except the $conv1$ were quantized, while the extra SSD layers were kept in full-precision for the experiments in this section, as discussed in Section 3.3. For training, we directly quantized and fine-tuned the full-precision baseline SSD models with ABC-Net scheme on PASCAL VOC data set. And similarly, we trained SSD using momentum optimizer with the momentum of 0.9, learning rate of 0.001 with decay factor 0.1□weight decay of 0.0005, and a batch size of 32. The comparisons between the full-precision baseline and the binary models for SSD are shown in Table 4.2 below.

Detector	Backbone	Methods	Training set	mAP(Binarized/FP)
SSD-300	ResNet-50	ABC-3	VOC07+12	71.19 / 74.35
	ResNet-50	ABC-5	VOC07+12	72.47 / 74.35

Table 4.2: Performances for binary SSD comparing to the full-precision baseline. Similarly, the mAP is reported with the PASCAL VOC standard, where IoU threshold is set to be 0.5. And the results is the average of three different initialization. **Binarized** and **FP** represent the binary and full-precision Faster R-CNN respectively. ABC-3 and ABC-5 represent the 3-bits and 5-bits ABC-Net quantization scheme respectively.

It is clear that, with 3-bits ABC-Net quantization and ResNet-50, the mAP gap between the full-precision baseline and the binary SSD is 3.16 mAP. This accuracy lost can be further reduced by increasing the quantization bit-length to 5-bits, where an mAP lost in only 1.88 mAP has been obtained. Similar to the trend observed in binary Faster R-CNN, the gap between full-precision and binary models tend to increase with the growing of model depths (or the number of total parameters).

The results above illustrated that, for only weight binarization, both the two-stages Faster R-CNN and single-stage SSD models were able to be convergent with ABC-Net quantization schemes in 3-bits and 5-bits. It is interesting to note that, the mAP reduction after quantization of single-stage SSD models tend to be smaller than that of two-stage Faster R-CNN. For instance, the ResNet-50 based Faster R-CNN showed approximately 5.5 mAP reduction after quantized with 5-bits ABC-Net scheme, which was much more higher than the 5-bits ResNet-50 based SSD counterpart (only 1.88 mAP reduction after quantization and the results for the models were obtained from the same training and test sets). This is because the two-stages region-based object detectors like Faster R-CNN tend to be more complex than the single-stage models, which is easier to accumulate quantization errors through the forward pass for both inference and training, and through the backward pass during training time. For instance, in Faster R-CNN two stages of classifications will be performed, where the quantization error caused by the first classification layers in RPN could result in incorrect region proposals and consequentially result in accumulated errors for the final detection layers. On the other hand, SSD models do not require the region proposal processes, which is less possibly to accumulate errors. Furthermore, SSD adopts the multi-levels' feature map scheme, where not only the output from the last layer but also the output maps from intermediate shallower layers would be used for final detection. And as the feature maps from the shallower layers contain less accumulated quantization errors, in theory, such a multi-level scheme can naturally mitigate the error accumulation. The results suggests that SSD like object detection models are more robust to accumulated quantization errors than the two-stages Faster R-CNN like networks, and therefore they are more suitable for binary quantization.

As a result, the ABC scheme based binary weight object detectors can still maintain very competitive accuracy. For instance, the 5-bits ABC-Net and ResNet-50 based SSD-300 achieved 72.47 mAP on PASCAL VOC2007 *text* set, which is only around 0.73 mAP lower than the original Faster R-CNN implementation [39], around 1.9 mAP lower than the original SSD-300 implementation [33] with much smaller model sizes.

4.2.2. Extension to input quantization

By further quantizing the input/activation discussed in Section 4.1.2, additional speedups can be achieved by simplifying the multiplications of the convolutional operations. Thus, in this section, further evaluations will be made to adopt the ABC-Net based activation/inputs quantization on the binary weights object detectors discussed in the last section.

For the Faster R-CNN models, we started with evaluating the ResNet-18 based models, as the experiments for weight quantization showed the shallower networks tend to suffer less from the quantization errors. The same strategies and training setting in the last section and Section 3.3 were adopted to tune the binary weight and binary activation Faster R-CNNs, where all the convolutional layers as well as activation layers from $Conv2\_x$ to $Conv5\_x$. The binary models are tuned on PASCAL VOC07 based on the full-precision models from stage two. However, even the shallowest ResNet-18 based Faster R-CNN failed to converge (very low mAP). The similar results were also observed on SSD models with ResNet-18.

Further attempts were made to improve the performance of binary inputs object detectors as following. But generally, the ABC-Net based input quantization failed to binarize the object detectors with accurate being preserved.

Using full-precision activation functions at the end of each ResNet block: As introduced in Chapter 2, ResNet architectures use shortcut connection to take the output from previous layers into account in each ResNet block. We consider this shortcut based architecture can mitigate the quantization error by combining outputs from shallower layers with less accumulated noise. Thus, the attempts were made to keep the last activation layer in each ResNet block in full-precision. Based on this approach, although no performance improvement can be observed for the ABC-Net based binary Faster R-CNN and SSD, higher accuracy can be achieved for PA-Net based quantization in next chapter.

Decrease the number of binary layers: This attempt was also used in existing works [42]. We firstly chose to maintain the deepest binary ResNet groups $Conv5\_x$ in full-precision, and then gradually increase the number of full-precision groups. The binary networks start to show mAP improvements when only two groups of ResNet layers in binary ($Conv3\_x$ and $Conv4\_x$), but the performance is still not sufficient (mAP less than 20). This phenomenon suggests that significant accuracy reduction is largely caused by the error accumulation of inputs quantization.

5 Piecewise approximation networks (PA-Net)

Aiming to reduce the quantization errors, and further improve the performance of binary models, piecewise approximation networks (PA-Net) was proposed and evaluated for image classification problems. As already introduced in Chapter 2, PA-Net also adopted multiple bits quantization similar to ABC-Net, but PA-Net showed better abilities to saving the accuracy of the models after quantization, even with the same quantization bit-length as ABC-Net. Thus, in this project, we will further extend and evaluate the PA-Net scheme on more complex object detection tasks, aiming to achieve more accurate detection than the ABC-Net based binary object detectors discussed in the last chapter.

In this chapter, the detailed PA-Net quantization scheme will be first discussed. And this project further specifies the inference-time binary operations for PA scheme based binary models. The performance of PA-Net on image classification tasks will first be introduced, then evaluations on object detection tasks will be performed.

5.1. Methodology

Similarly to previous works for network binarization [30, 32], PA-Net also includes the quantization approach for both the weight and activation/inputs. But differently, PA scheme was designed to approximate the full-precision weights and inputs on the element level without increasing the quantization bit-length. In this section, the detailed PA schemes for both weight and inputs quantization will be discussed.

5.1.1. Weight quantization

a) Forward propagation

Similar to ABC-Net scheme in the last chapter, PA scheme also uses multiple binary bits to approximate the full-precision weights, and the weights are approximated as a whole because the channel-wise approximation requires much more computational resources during training time.

The full-precision weights W with input channel c_{in}, output channel c_{out}, height h and width w can be approximated with combination of M binary weights by Equation 5.1.

$$W \approx \sum_{m}^{M} W_m, \tag{5.1}$$

With PA scheme, the full-precision weights are separated into M pieces and quantized with the piecewise mapping function $P(W)$ shown in Equation 5.2 below.

$$W_m = P(W) = \begin{cases} \alpha_1 \times B_{W_m} & W \in (-\infty, u_1] \\ \alpha_i \times B_{W_m} & W \in (u_{i-1}, u_i], i \in [2, \frac{M}{2}] \\ \cdots & \\ 0 \times B_{W_m} & W \in (u_{\frac{M}{2}}, u_{\frac{M}{2}} + 1] \\ \alpha_i \times B_{W_m} & W \in (u_i, u_{i+1}], i \in [\frac{M}{2} + 1, M - 1] \\ \cdots & \\ \alpha_M \times B_{W_m} & W \in (u_M, \infty] \end{cases} \tag{5.2}$$

where α_is are the scaling parameters to represent the corresponding weight pieces. u_is are the value boundaries to separate the full-precision weights into M pieces W_ms determined directly by the value of each element in the weight kernels. B_{W_m} is the binary masks with the same shape as W, where the values are set to be 1 if the elements from the full-precision kernel are located in the corresponding ranges, while set to be 0 otherwise. The weight kernels tend to have distributions close to Gaussian, which has been also observed in [32]. Therefore, the boundary points for the weights u_is are fixed using the averages $\overline{W} = mean(W)$ and standard division $\sigma_W = std(W)$ of full-precision weights. To be more specific, the M boundary points u_is are uniformly sampled from $-2\sigma_W$ to $2\sigma_W$. One example can be shown in Table 5.1 blow when using $M = 8$ to represent the weight kernels.

u_i Values	u_1	u_2	u_3	u_4
	-1.5σ	-1σ	-0.5σ	-0.25σ
u_i Values	u_5	u_6	u_7	u_8
	$+0.25\sigma$	$+0.5\sigma$	$+1\sigma$	$+1.5\sigma$

Table 5.1: Example of the boundary points u_is to separate the full-precision weights into $M = 8$ pieces, where σ is the standard division of w $\sigma = std(W)$

Then, the scaling factor α_is are determined by calculating the average value of each pieces using Equation 5.3 below.

$$\alpha_i = mean(W_m), \text{ where } i = m = 1, 2, \cdots, M \tag{5.3}$$

Again, the full-precision convolutional operation to obtain one single element on the output feature map shown in Equation 4.5 in the last chapter can be reduced to the binary weight form shown in Equation 5.4, where $x_{c,w,h}$ represent one single elements from the full-precision inputs, and $b^m_{c,w,h}$ is the binary weight of mth piece. and denotes the bit and operation. As a result, the PA-based binary weight convolutional operation to generate one output element requires only additions, and operations as well as M multiplications.

$$y_{c',w',h'} = \sum_{m=1}^{M} \alpha_m \sum_{c,w,h} x_{c,w,h} \; and \; b^m_{c,w,h} \tag{5.4}$$

More generally, the forward propagation of PA-Net based convolutional layers can be concluded in Equation 5.5, where X represents the full-precision inputs with batch size n, input channel c_{in}, height h_{in} and width w_{in}. And instead of $\{-1, 1\}$ in ABC-Net scheme, the binary bases B_{W_m}s are in $\{0, 1\}$. The output Y within $\mathbb{R}^{n \times c_{out} \times w_{out} \times h_{out}}$ are then calculated by the and operation based convolution based on Equation 5.4 above.

$$Y = \sum_{m=1}^{M} \alpha_m ConvAnd(B_{W_m}, X) \text{ where } X \in \mathbb{R}^{n \times c_{in} \times w_{in} \times h_{in}} \text{ and } B_{W_m} \in \{0,1\}^{c_{out} \times c_{in} \times w \times h} \tag{5.5}$$

b) Backward propagation

During backward propagation, the gradients with respect to the M quantized bases W_ms have to be calculated and summed up, as B_{W_m}s are used separately during forward propagation in Eqn.5.5. However, different from the straight-through estimator used in ABC-Net scheme, PA scheme uses linear functions to approximate the gradients though the piecewise function $P(W)$. The gradients approximation function is showed in Equation 5.6 below.

$$\frac{\partial W_m}{\partial W} = \begin{cases} \lambda_W(\alpha_2 - \alpha_1) & W \in (-\infty, s_1] \\ \lambda_W(\alpha_{i+1} - \alpha_i) & W \in (s_{i-1}, s_i], i \in [2, \frac{M}{2}] \\ \cdots & \\ \lambda_W(0 - \alpha_{\frac{M}{2}}) & W \in (s_{\frac{M}{2}}, s_{\frac{M}{2}} + 1] \\ \lambda_W(\alpha_{i+1} - \alpha_i) & W \in (s_i, s_{i+1}], i \in [\frac{M}{2} + 1, M - 1] \\ \cdots & \\ \lambda_W(\alpha_M - \alpha_{M-1}) & W \in (s_{M-1}, +\infty) \end{cases} \tag{5.6}$$

Note that, during backward propagation, s_is is another set of endpoints used to define the ranges where the backward approximation for each weight piece W_m should be performed. And λ_W is a predefined hyper-parameter, which is different according to the number of quantization pieces M. To be more specific, the endpoints s_i can be directly calculated using Equation 5.7.

$$s_i = \frac{(u_{i+1} + u_i)}{2}, \text{ where } i \in [1, M - 1] \tag{5.7}$$

As the result, the backward propagation for the piecewise function can be approximated with the linear slop $\lambda_W(\alpha_i - \alpha_{i-1})$. A more clear illustration of the forward and backward propagation of the piecewise function can be found in Figure 5.1 below, where the estimated slops in $[s_{i-1}, s_i]$ are used to represent the gradients flowing backwardly through the piecewise function in domain $[u_{i-1}, u_{i+1}]$.

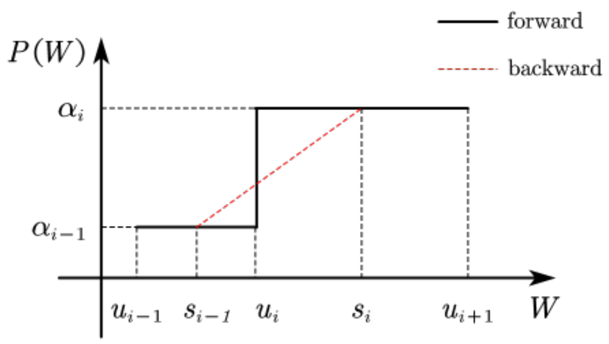


Figure 5.1: The example for the forward and backward approximation of the piecewise function $P(W)$. During forward propagation, the weights in the same piece are mapped to the corresponding α_is, while the gradients flowing through $P(W)$ are approximated with the linear slops.

5.1.2. Inputs/activation quantization

a) Foward propagation

To achieve further accelerations, PA-Net also introduces inputs quantization scheme with binary activation function. The full-precision inputs X, with batch size n, input channel c_{in}, height h_{in} and width w_{in} will also be separated into multiple pices, and represented with multiple binary masks similar to the weight quantization scheme discussed above. The full-precision activation is also represented by the linear combination of N binary bases by Equation 5.8.

$$X \approx \sum_{n}^{N} A_n, \tag{5.8}$$

And the piecewise quantization function to calculate each binary activation A_n is showed in Equation 5.9 below.

$$A_n = A(X) = \begin{cases} 0 \times B_{A_n} & X \in (-\infty, v_1] \\ \cdots & \\ \beta_i \times B_{A_n} & X \in (v_i, v_{i+1}], i \in [1, N-1] \\ \cdots & \\ \beta_N \times B_{A_n} & X \in (v_N, +\infty) \end{cases} \tag{5.9}$$

Similar to the weight quantization, scaling factor β_i is used to scale the corresponding piece bounded by the endpoints v_is. And B_{A_n}s denote the binary masks in $\{0, 1\}$, which have the same shape as the input X. But different from the u_is and α_is in weights quantization, both the endpoints v_is and the scaling factor β_is are trainable parameters, which will be fine-tuned together with the network parameters during training time. Consequentially, full-precision inputs will be mapped to the N binary masks B_{A_n}s and will be directly used to to calculate the outputs of the binary convolutional layers during inference time.

b) Backward propagation

For backward propagation, the similar piecewise approximation discussed in weight quantization above is utilized to estimate the gradients flowing backwardly through the activation quantization function $A(X)$. Then the backward propagation function for the input/activation quantization of PA scheme is defined in Equation 5.10. Note that, for more accurate approximation, the backward propagation for input quantization is divided into $N + 2$ pieces, by independently estimating the backward propagation for the lower and upper bounds of $A(X)$.

$$\frac{\partial A}{\partial X} = \begin{cases} 0 & X \in (-\infty, t_0] \\ \lambda_A(\beta_1 - 0) & X \in (t_0, t_1] \\ \cdots & \\ \lambda_A(\beta_{i+1} - \beta_i) & X \in (t_i, t_{i+1}], i = 1, \cdots, N-1 \\ 0 & X \in [t_N, +\infty) \end{cases} \tag{5.10}$$

λ_A is a pre-defined hyper-parameter, which is fixed for each CNN models. And t_is, again are the boundary points to separate the pieces. The values of t_is are determined by Equation 5.11, where Δ is a fixed parameters for each CNN model.

$$\begin{cases} t_i = (v_i + v_{i+1})/2 & \text{for } i = 1, 2, \cdots N-1 \\ t_0 = 2v_0 - t_i & \\ t_N = v_N + \Delta & \end{cases} \tag{5.11}$$

The endpoints $t_i, i = 1, 2, \cdots, N-1$ are the median points for the pieces during forward propagation. t_0 and t_N are the endpoints of the lower and upper bound ranges $X \in (-\infty, t_0]$ and $x \in [t_N, +\infty)$, where the backward propagation of the piecewise function are approximated to 0. As the results, for full-precision activation $X \in [t_{i-1}, t_i]$, the gradients following backwardly through the piecewise function $A(X)$ are estimated to $\lambda_A(\beta_i - \beta_{i-1})$ and more specifically, one example for the approximation is also illustrated in Figure 5.2 below.

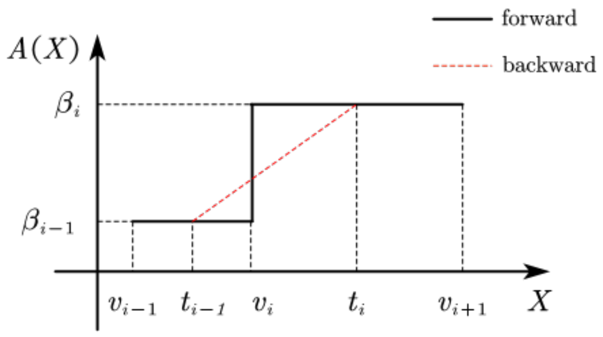


Figure 5.2: The example for the forward and backward approximation of the piecewise activation binary function $A(X)$. During forward propagation, the activation in the same piece are mapped to the corresponding β_is, while the gradients flowing through $A(X)$ are approximated with the linear slops.

Moreover, as the scaling parameter β_is, and the boundary points v_is are also trainable for input quantization, their backward propagation should also be defined for updating. Equation 5.12 shows the backward propagation for scaling parameter β_is, where C is the cost, and the backward propagation is the calculated piece-wisely. And the gradients through the elements in each piece will be summed up by $sum(.)$.

$$\frac{\partial C}{\partial \beta_i} = \frac{\partial C}{\partial A}\frac{\partial A}{\partial \beta_i} \tag{5.12}$$

$$= \begin{cases} 0 & A \in (-\infty, v_1] \\ sum(\frac{\partial C}{\partial A}) & A \in (v_i, v_{i+1}], i \in [1, N-1] \\ sum(\frac{\partial C}{\partial A}) & A \in [v_N, +\infty), i = N \end{cases} \tag{5.13}$$

Then, the piecewise backward propagation for the boundary points v_is in PA scheme is defined in Equation 5.14 below, where the gradients with respect to v_is are estimated and used for v_is' updating.

$$\frac{\partial C}{\partial v_i} \approx \begin{cases} \lambda_A(\beta_1 - 0) \times sum(\frac{\partial C}{\partial A}) & A \in (t_0, t_1] \\ \lambda_A(\beta_{i+1} - \beta_i) \times sum(\frac{\partial C}{\partial A}) & A \in (t_i, t_{i+1}], i = 1, 2, \cdots, N-1 \\ 0 & otherwise \end{cases} \tag{5.14}$$

5.1.3. PA Scheme based binary convolutional operation

Based on the activation and weight quantization approaches discussed above, the convolutional operation can be then performed without any multiplications during inference time. The specific operation to obtain one single element for the convolutional layer's output $y_{c',w',h'}$ shown in Equation 5.4 can be further reduced to Equation 5.15. $a^n_{c,w,h}$ denotes one single element represented in $\{0, 1\}$ from the nth binary inputs B_{A_n} at one sliding window position, which is on cth input channel, height coordinate h, and width coordinate w. As α_m and β_n have already been fixed in the training stage, the term $\alpha_m\beta_n$

can be pre-calculated, and therefore can be directly used in a look-up-table form during inference time. As the result, the element-wise multiplication of the convolutional operation can be reduced to only bit and operations as well as look-up-tables, and then all the elements will be summed up to obtain the outputs.

$$y_{c',w',h'} = \sum_{m=1}^{M} \sum_{n=1}^{N} \alpha_m \beta_n \; and \; (\sum_{c,w,h} a_{c,w,h}^{n} \; and \; b_{c,w,h}^{m}) \tag{5.15}$$

From a more general point of view, the forward propagation of the PA scheme based convolutional layers during inference time is summarised in Equation 5.16, where $BinConvAnd$ is the PA-based binary convolutional operation in the form of Equation 5.15. The full-precision weight W and activation/input X is quantized to only M and N bits. The piecewise function Equation 5.4 and Equation 5.15 are applied to obtain the scaled binary weight W_m and binary activation A_n respectively. The whole process is still maintained in a parallel form, which can also be benefited using hardware acceleration techniques.

$$Y = \sum_{m=1}^{M} \sum_{n=1}^{N} BinConvAnd(A_n, W_m), \text{ where } W_m = P(W) \text{ and } A_n = A(X) \tag{5.16}$$

5.1.4. Training

To train the PA-Net based binary models, similar to the approach in [32] and [35], at each training step the full-precision weight and inputs will be mapped to binary to calculate the cost for the forward propagation, while for the backward propagation of each training step, only the full-precision weights will be updated. The full-precision gradients will be calculated and fed backwardly during training, and the binary models can also be trained with typical optimizers such as ADAM and Momentum. This project basically adopts the same training algorithm the original proposed PA scheme, which is summarised in Algorithm 2 below. Algorithm 2 included an extra branch to train the models with binary weights and full-precision inputs.

Similarly, in Algorithm 2, $BatchNorm(.)$ and $BackBatchNorm(.)$ are the forward and backward propagation function for the batch normalization respectively. $BackConv()$ is the backward convolutional operation. $Update(.)$ denotes the weight updating with the optimizer, and $UpdateLR(.)$ is the learning rate updating function, such as learning rate decay. The activation/inputs can be chose to keep in full-precision, when set *using binary activation/input* flag to $false$.

5.2. Experiments

In the last chapter, the ABC-Net based quantization scheme was used to binarize the object detection networks. Attempts were made to quantize the weights as well as the inputs at the same time, and the quantized models were evaluated on PASCAL VOC data set. In this chapter, we further adopt PA-Net based quantization scheme to object detection models, as it tends to be more accurate than the ABC-Net scheme, which has already been illustrated on ImageNet image classification problems. Also, as another research question of this project, it is also interesting to evaluate the proposed PA-Net scheme on object detection tasks and to make comparison with existing binarization approaches.

Similar to Chapter 4, we first adopt weight quantization with PA scheme on object detectors and further make exploration to binarize both the inputs and weights. All the binary models will be evaluated on PASCAL VOC date set.

Algorithm 2 PA scheme based training algorithm.

```
Input: one mini-batch of input images and their ground truth, number of bits $M$ used to represent the weights, number of bits $N$ used to represent the activation/input, the full-precision weight $W$, initialized parameters $u_i$s, $v_i$s, $\beta_i$s, and the learning rate $\eta$.
Output: updated full-precision weights $W$, scaling factor $\beta_i$s, boundary points $v_i$s, and learning rate $\eta$ for next mini-batch.
 1: forward propagation:
 2: for $l = 1$ to $L$ do
 3:     Compute $\alpha_m^l$ and $W_m^l$, where $m = 1, 2, \cdots, M$ for $l$th layer using Eqn.5.3 and Eqn.5.2;
 4:     if Using binary activation/inputs then
 5:         Compute $lth$ convolutional layer's output $Y^l$ using Eqn.5.15;
 6:     else
 7:         Compute $lth$ convolutional layer's output $Y^l$ using Eqn.5.5;
 8:     end if
 9:     Optionally apply pooling operation;
10:     Compute batch normalization $X^l \leftarrow BatchNorm(Y^l)$;
11:     if $l < L$ then
12:         if Using binary activation/input then
13:             Compute $A_n^l$ for $n = 1, 2, \cdots, N$ using Eqn.5.9;
14:         else
15:             Compute full-precision activation $A^l$ (e.g. ReLU);
16:         end if
```

5.2.1. Weights quantization

The three stages training strategy is adopted also for the PA scheme based object detectors, and the backbone networks of the first stage, as well as the full-precision object detectors of the second stage, are exactly the same models used in Chapter 4. Then, the full-precision Faster R-CNNs and SSDs from the second stage are fine-tuned with the PA weight quantization scheme discussed in Section 5.1.1 on the PASCAL VOC data set.

To be more specific, on stage three for Faster R-CNNs, we binarized all the layers excepts the first convolutional layer, the last fully connected layer, and the layers in RPN sub-net. The training setting is the same as used in Chapter 4, where momentum optimizer with the momentum of 0.9, learning rate of 0.001 with decay factor $\gamma = 0.1$,weight decay of 0.0005, and mini-batch size of 1 are used to tune all PA scheme based Faster R-CNN models. The results are then evaluated on PASCAL VOC *test* set, and are illustrated in Table 5.2 below.

Detector	Backbone	Methods	Training set	mAP(Binarized/FP)
Faster R-CNN	ResNet-18	PA-5	VOC07	55.32 / 60.06
	ResNet-50	PA-5	VOC07+12	67.58 / 73.43
	ResNet-101	PA-5	VOC07+12	69.86 / 75.41

Table 5.2: Performances for binary Faster R-CNNs comparing to the full-precision baseline. Similarly, the mAP is reported with the PASCAL VOC standard, where IoU threshold is set to be 0.5. And the results is the average of three different initialization. **Binarized** and **FP** represent the binary and full-precision Faster R-CNN respectively. PA-5 represent the 5-bits PA scheme based quantization.

With PA scheme based weight quantization, Faster R-CNN with the backbone networks in different depth maintained converge, and still showed comparable detection accuracy comparing to the existing state-of-the-art ABC-Net [32] approaches with the same bit-length, and the detailed comparisons will be given in Chapter.6. As also being observed in Chapter 4, the results showed that the PA-based binary

```
17:     end if
18: end for
19: backward propagation:
20: Compute $g_{A^L} = \frac{\partial C}{\partial A^L}$;
21: for $L = L$ to 1 do
22:     if $l < L$ then
23:         if Using binary activation/input then
24:             Compute $g_{X_n^l} = \frac{\partial A_n^l}{\partial X}$ based on Eqn.5.10;
25:             Compute $g_{\beta_n^l} = \frac{\partial C}{\partial \beta_n^l}$ based on Eqn.5.12
26:             Compute $g_{v_n^l} = \frac{\partial C}{\partial n^l}$ based on Eqn.5.14
27:             $g_X^l \leftarrow \sum_n^N g_{X_n^l}$
28:         else
29:             Compute $g_X^l$ (e.g. $BackReLU$);
30:         end if
31:     end if
32:     Compute $g_Y^l \leftarrow BackBatchNorm(g_{X^l}, X^l)$
33:     $g_{W_m^l} \leftarrow BackConv(g_Y^l, W_m^l, A_n^{l-1})$ based on Eqn.5.6;
34:     $g_W^l \leftarrow \sum_m^M g_{W_m^l}$
35: end for
36: parameter updating:
37: for $L = 1$ to $L$ do
38:     Update $l$th layer's weight $W^l \leftarrow Update(W^l, \eta, g_W^l)$;
39:     Update $\beta_n \leftarrow Update(\beta_n, \eta, g_{\beta_n}^l)$;
40:     Update $v_n \leftarrow Update(v_n, \eta, g_{v_n}^l)$
41:     Update learning rate $\eta \leftarrow UpdateLR(\eta)$
42: end for
```

object detectors with greater depth (or more parameters) tend to have larger performance reductions comparing to the full-precision baseline. Although only 4.7 mAP decrease with the shallow ResNet-18 can be obtained, the deepest ResNet-101 based binary Faster R-CNN showed a significant 5.8 mAP reduction comparing to the full-precision model.

On the other hand, again we apply the training strategies in Section 3.3 to tune the single-stage SSD models, where all the layers are binarized with PA scheme based weight quantization, except the first convolutional layer, the last FC layer as well as the extra SSD layers. The SSD binary SSD models are then fine-tuned on PASCAL VOC data set with momentum optimizer, which has the momentum of 0.9, weight decay of 0.0005, and a batch size of 32. The evaluation results on PASCAL VOC2007 $test$ are shown in Table 5.3 below.

Detector	Backbone	Methods	Training set	mAP(Binarized/FP)
SSD-300	ResNet-50	PA-5	VOC07+12	72.53 / 74.35

Table 5.3: Performances for binary SSD comparing to the full-precision baseline. Similarly, the mAP is reported with the PASCAL VOC standard, where IoU threshold is set to be 0.5. And the results is the average of three different initialization. **Binarized** and **FP** represent the binary and full-precision SSD respectively. PA-5 represent the 5-bits PA quantization scheme.

It is clear that, with 5-bits PA scheme based quantization, the single-stage SSD models tend to have less accuracy reductions resulted by the quantization noise. The mAP gap between the 5-bits ResNet-50 based binary SSD and the full-precision inputs is 1.82mAP, which is only approximately 2.4% losses.

Generally, as also observed in Chapter 4, the single-stage object detectors tend to suffer fewer accuracy reductions after weight binarization, because the two-stage models are more likely to accumulate quantization errors. Although increasing the network depth (or network parameters) will result in a larger performance gap compared to the full-precision models, binary object detectors can still benefit from using stronger backbone networks.

5.2.2. Extension to input quantization

As introduced in Section 5.1.3, for further speedups, the inputs can also be quantized to further simplify the convolutional operation with only additions, subtractions as well as bit AND operations.

In this section, the attempts of quantizing the inputs with the binary activation function discussed in Section 5.1.2 are made for object detection tasks. More specifically, the 5-bits ($M = N = 5$) activation and weight quantization based on PA scheme is adopted for both the two-stage Faster R-CNN as well as the single-stage SSD models. Also, we directly fine-tune the binarized object detectors on the top of full-precision models from the second stage on PASCAL VOC data sets. And the strategies used for the ABC-Net attempts discussed in Section 4.2.2 are also adopted to the PA scheme based object detectors in this section. As for training, the same optimizer and hyperparameters in Section 5.2.1 are used.The binary weight and binary inputs models are also evaluated on the PASCAL VOC2007 *test* set. The evaluation results can be found in Table 5.4 below.

Detector	Backbone network	Training set	mAP(Binarized/FP)
Faster R-CNN	ResNet-18	VOC07	47.19 / 60.06
	ResNet-50	VOC07	49.18 / 65.38
SSD-300	ResNet-50	VOC07+12	58.6 / 74.35

Table 5.4: Evaluation results of 5-bits PA scheme based binary weights and binary inputs Faster R-CNN and SSD models. All results are obtained from PASCAL VOC2007 *test* set. All mAP are obtained using PASCAL VOC standard with IoU threshold 0.5. **Binarized** and **FP** represent the quantized models and the full-precsion models respectively.

Generally, shown in Table 5.4, both the single-stage SSD and two-stage Faster R-CNN could converge with PA quantization scheme on PASCAL VOC data set but with significant accuracy reductions. For instance, the 5-bits based binary activation Faster R-CNN showed a large mAP gap in 12.87mAP comparing to the full-precision baseline, even with the shallowest ResNet-18 backbone networks. The accuracy margin between the binary Faster R-CNN and full-precision baseline even grow to 16.18 mAP when using the deeper ResNet-50 backbone. On the other hand, the single-stage SSD model showed slightly lower accuracy reduction in 15.75mAP, with 5-bits PA-Net quantization, and the same ResNet-50 backbone network.

Despite the large mAP reduction comparing to the full-precision baseline object detectors, the PA scheme based binary models can still outperform some existing works. For example, the simplest ResNet-18 based binary weight and binary activation Faster R-CNN can outperform the existing binary object detector in [42] with smaller models size, which showed 44.3mAP@IoU=0.5 on PASCAL VOC2007. Moreover, the ResNet-50 and 5-bits PA-Net based SSD-300 still maintain acceptable accuracy in 58.6mAP, which outperforms the real-time full-precision Fast YOLO [38] (52.7mAP trained on PASCAL VOC07+12 *trainval*, and evaluated on VOC07 *test*).

6 Discussions

From Chapter 4 to 5, we introduced the methodologies to quantize the object detectors with both ABC-Net and PA-Net based scheme, and we also evaluated the trained models on the PASCAL VOC data set.

Then, in this chapter, a storage/memory analysis of the quantized model will be performed to give a better understanding of the model compression point of view. Also, we will compare the two accuracy-toward binarization approaches in the context of object detection. Furthermore, the attempts to make further compression will be made to obtain models with an even smaller size, and, based on it, one example will be showed to make trade-offs between model storage and detection precision for practical applications.

6.1. Storage/memory analysis

In Chapter 4 and Chapter 5, we evaluated the ABC-Net based and PA-Net based object detectors respectively. But in this section, we will measure the performances of the models in terms of the compression rate and storage saving.

Theoretical, the maximum compression can be achieved by the multiple-bits binary quantization schemes like ABC-Net and PA-Net is $M/32$, where M stands for the M-bits quantization. Note that the maximum compression can only be achieved only when all of the network parameters are binarized. However, in practice, binarizing all the parameters usually leads to huge accuracy reductions even for the models used in image classification problems [32, 35], and therefore only intermediate compression has been achieved by only binarizing a part of the network parameters [32, 35, 42]. Moreover, for more complex tasks like object detection, more layers would be kept in full-precision to ensure sufficient detection accuracy, which will result in lower compression.

For the results reported in Chapter 4 and Chapter 5, the first convolutional layer, the last convolutional layer as well as the RPN sub-networks of the binary Faster R-CNN models are kept in full-precision as discussed in Section 3.3. Meanwhile, similarly the first and last layers, as well as the extra SSD layers, are kept in full-precision for SSD models. More specifically, the storage of the binary object detectors can be found in Table 6.1 below.

Based on the ABC and PA schemes, the convolutional layers can be compressed to approximately 15.63% and 9.38% using 5-bits and 3-bits quantization. However, as considering parts of parameters are kept in full-precision to save the detection accuracy, the storage of the quantized models are almost dominated by the layers maintained in full-precision. As a result, PA and ABC scheme was only able to achieve the highest compression of approximately 26.31% and 31.39% with 3-bits and 5-bits quantization respectively on ResNet-101 based Faster R-CNN models, which are much lower than the theoretical maximum compression. Furthermore, for the single-stage SSD models, it is more clear that

Detector	Backbone	Methods	Quantized / MB(%)	Kept FP / MB(%)	Total (Comp.) / MB(%)
Faster R-CNN	ResNet-18	PA-3/ABC-3	3.99 (29.62%)	9.47 (70.30%)	13.47 (25.83%)
	ResNet-18	PA-5/ABC-5	6.67 (41.33%)	9.47 (58.67%)	16.14 (30.96%)
	ResNet-50	PA-3/ABC-3	8.38 (18.41%)	37.15 (81.59%)	45.53 (35.97%)
	ResNet-50	PA-5/ABC-5	13.97 (26.29%)	37.15 (72.70%)	53.13 (40.39%)
	ResNet-101	PA-3/ABC-3	15.16 (28.98%)	37.16 (71.02%)	52.32 (26.31%)
	ResNet-101	PA-5/ABC-5	25.26 (40.47%)	37.16 (59.53%)	62.42 (31.39%)
SSD-300	ResNet-50	PA-3/ABC-3	8.43 (12.09%)	61.31 (87.91%)	69.74 (46.12%)
	ResNet-50	PA-5/ABC-5	14.05 (18.64%)	61.31 (81.36%)	75.36 (49.83%)
	ResNet-101	PA-3/ABC-3	15.25 (20.15%)	60.43 (79.86%)	75.68 (38.41%)
	ResNet-101	PA-5/ABC-5	25.53 (29.70%)	60.43 (70.30%)	85.96 (38.41%)

Table 6.1: Storage saving and compression rates of the binary models discussed in Chapter 4 and Chapter 5. **Quantized** and **Kept FP** represent the storage(in MB) as well as the proportion of the binarized parameters and full-precision parameters respectively. And **Comp.** denotes the overall compression rate of the binarized models.

the overall compression is significantly limited by the parameters of the network kept in full-precision. Although the majority of the network parameters are quantized in binary representations, only maximally 38.41% overall compression was obtained.

6.2. Further compression of the object detectors

Introduced above, for the binary object doctors' implementations discussed in Chapter 4 and Chapter 5, except first and last layers, were kept in full-precision (suggested in [32, 35]), the extra layers used to handle the bounding box are also maintained full-precision to save the accuracy. This led to high storage costs, and largely limited the overall compression rate of the models, especially for the single-stage SSD models, which contained relatively high proportions of extra parameters on the top of the backbone networks. Thus, in this section, we will further explore the performances with more layers of the models quantized to binary representations.

Aiming to obtain higher compression, for the SSD models, we only maintain the first and last layers in full-precision, and binarize all the other layers including the extra SSD layers. On the other hand, for the two-stages Faster R-CNN models, we further quantize the first $conv\_3 \times 3$ convolutional layers in the sub RPN networks (see Figure 3.3) with PA scheme. But the $conv\_1 \times 1$ layers in RPN are still kept in full-precision as they are the last layers to create the feature maps for foreground/background classification to generate the region proposals. In this section, ResNet-50 based Faster R-CNN and Rsenet-50 based SSD are used to evaluate the results for further quantization. More specifically, the performance of the models in terms of both storage as well as the detection accuracy can be obtained in Table 6.2.

Detector	Methods	mAP	Quantized / MB(%)	Kept FP / MB(%)	Total (Comp.) / MB(%)
Faster R-CNN	M1+PA-5	67.58	13.97 (26.29%)	37.15 (72.70%)	53.13 (40.39%)
	M2+PA-5	60.89	19.60 (94.46%)	1.15 (5.54%)	20.75 (16.39%)
SSD-300	M1+PA-5	72.47	14.05 (18.64%)	61.31 (81.36%)	75.36 (49.83%)
	M2+PA-5	71.32	25.50 (56.48%)	19.65 (43.52%)	45.15 (29.86%)

Table 6.2: Evaluation and comparison between two methods. For **M1**, the layers are binarized except the first layer and last layers, the RPN layers, and the extra SSD layers. While, as introduced above, in **M2**, all the layers excepts the first and the last layers are binarized. Faster R-CNN models are trained and evaluated on PASCAL VOC2007, and the ResNet-50 based SSD are trained on VOC07+12 and evaluated on VOC07 *test* set. All mAP are obtained using PASCAL VOC standard with mAP threshold at 0.5. **Quantized** and **Kept FP** represent the storage(in MB) as well as the proportion of the binarized parameters and full-precision parameters respectively. And **Comp.** denotes the overall compression rate of the binarized models.

Shown in Table 6.2, significantly higher compression was achieved when quantizing the layers in head networks. The ResNet-50 based Faster R-CNN and SSD models can be compressed to only 20.75 MB and 45.15 MB, which have only approximately 16% and 30% storage size respectively comparing to the full-precision model. Not surprisingly, with higher quantization ratios, further accuracy reduction can be observed for both the models. However, the further compressed models can still maintain higher accuracy and smaller storage size comparing to other real-time full-precision models, like Fast YOLO [38] (52.7 mAP on VOC07 $test$) and Tiny YOLO v2 [36] (57.1 mAP on VOC07 $test$).

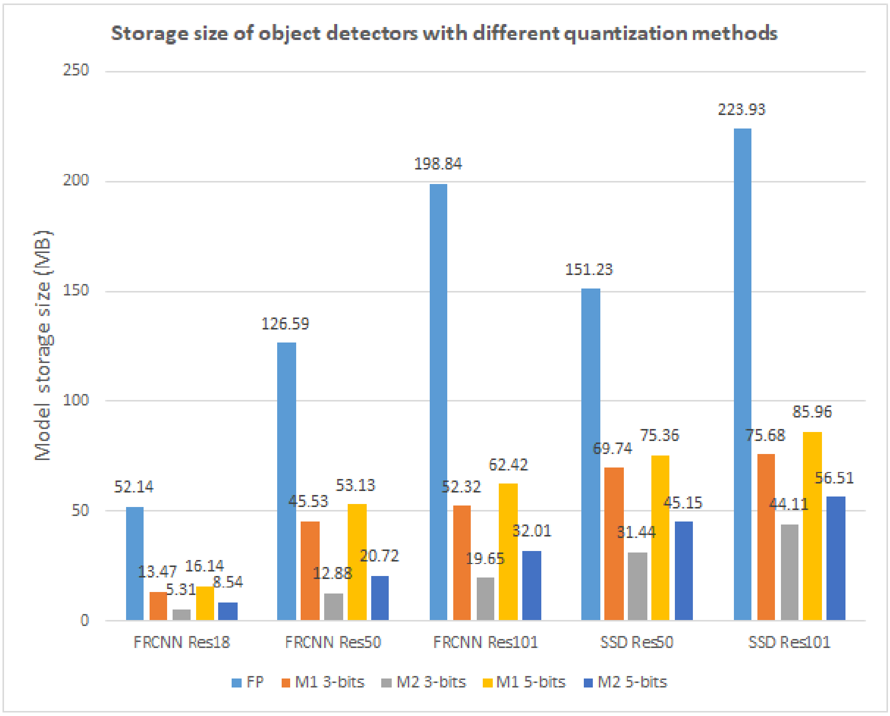


Figure 6.1: Model storage for different object detectors with 3-bits and 5-bits PA and ABC quantization scheme. **M1** is the quantization method introduces in section 3.3. And for **M2**, based on **M1**,the extra SSD layers for single stage SSD models and the $conv\_3 \times 3$ layers of the RPN for Faster R-CNN models are further quantized to obtain higher compression.

Furthermore, a better view of the model storage with both of the methods can be found in figure Figure 6.1. It is clear that the storage for the object detectors can be significantly reduced, by further convert some of the layers in the head networks to binary representations. Note that some of the results are obtained from measuring can calculating the storage of the models with random weights. This project did not train all of the models with method 2 due to the limited computational resources. But the results in Table 6.2 have already shown the trend of further accuracy reductions when binarizing more layers of the object detectors. Based on the observation in previous chapters, the deeper ResNet-50 and ResNet-101 based detectors would be very likely to suffer more accuracy reductions than the models with ResNet-18 backbone network and methods 2. In practice, one can always make such trade-offs between the detection accuracy and models storage/memory depends on the hardware platform and precision requirements.

6.3. Comparisons and trade-offs

As one of the objectives for this project. In this section, we will compare the PA binarization scheme with the existing ABC-Net scheme on object detection tasks. Moreover, we will also show an example of a brief trade-offs between storage size and detection precision for the binary object detectors obtained in the previous chapters.

6.3.1. Comparisons between ABC and PA scheme on object detection

As already discussed in Chapter 4 and Chapter 5, both of the ABC and PA scheme are multiple-bits binary quantization approaches for both network parameters and inputs quantization. ABC scheme approximates the weights and inputs by solving the optimal scaling factors which scale the binary bases as a whole (see Chapter 4). while, instead of approximate the whole weight or input tensors, PA scheme separates the weights and inputs into multiple pieces, and directly approximate each piece independently (see Chapter 5). As a result, although they compress the full-precision weights and inputs to the same bit length, the detection accuracy are much different. Table 6.3 below summaries and compares the performances of the binarized object detectors with ResNet backbone networks in different depths obtained from Chapter 4 and Chapter 5.

Detector	Backbone	mAP PA-5	mAP ABC-5
Faster R-CNN	ResNet-18	55.32	55.16
	ResNet-50	67.58	67.69
	ResNet-101	69.86	70.31
SSD-300	ResNet-50	72.53	72.47

Table 6.3: Performances comparisons between 5-bits ABC-Net and PA schemes for Faster R-CNN and SSD-300 object detectors with ResNet backbone. Except the ResNet-18 based Faster R-CNN, other models were all trained on VOC07+12. And all the models are evaluated on VOC07 *test* set, with IoU threshold at 0.5.

It is clear that, for only weights quantization, no significant differences were observed in terms of detection accuracy when using PA and ABC schemes with the same bit length for only network parameter quantization. And for the object detectors with deeper backbone networks (e.g.ResNet-50, ResNet-101) the ABC-Net based scheme tend to slightly outperform PA scheme. However, comparing to ABC-Net scheme, PA scheme has the advantages as shown below:

- **Much better performances for input/activation quantization:** Although no significant improvement can be observed on the weights quantization comparing to ABC-Net scheme, PA scheme was able to show significantly better abilities to quantize the inputs/activation than the ABC-Net scheme. As the attempts made in the previous sections (see Section 4.2.2), ABC-Net based approaches showed difficulties (very low mAP) to quantize activation. But on the contrast, experiments in Section 5.2.1 illustrated PA schemes were able to quantize the object detectors while maintaining comparable detection accuracy, which was able to outperform some real-time object detectors such as Fast YOLO [38].

- **High efficiency during training time:** Another advantage of PA scheme is that it runs much faster than ABC-Net based quantization during training time. To be more specific, ABC-Net requires considerable extra run-time memory and computation resources to solve the LSE optimization problems shown in Equation 4.4 for every weight tensors independently to obtain the scaling factor α_ms during training time. While for PA scheme, the scaling factors are obtained directly by averaging the elements from the corresponding pieces using Equation 5.3, which requires much fewer computations comparing to the ABC-Net approach. As a result, for the implementations used in this project,5-bits ABC-Net and ResNet-101 based Faster R-CNN network run at approximate 0.76 iterations per second on one Nvidia Tesla P100 GPU during training. While 5-bits PA scheme and ResNet-101 based Faster R-CNN was able to run at approximately 3.62 iterations per second on the same machine, which was around 4.76 times faster than the ABC-Net scheme.

In short, although no big performance improvements can be observed for weight quantization between using PA and ABC scheme, PA scheme can run much faster than the ABC scheme during training time. Moreover, the PA scheme outperformed ABC scheme for inputs/activation quantization, which can still lead to comparable performances on object detection tasks.

6.3.2. Trade-offs

As introduced in the previous sections, in real-world applications, one can always make trade-offs between the model storage size and the detection accuracy by carefully selecting the number of layers to apply binary quantization. Obviously, a high ratio of binary layers will result in high quantization errors but lower storage and memory consumption. To be more specific, as shown in Figure 6.2 below, brief examples were made to make trade-offs for ResNet-50 based Faster R-CNN and ResNet-50 based SSD models with PA scheme.

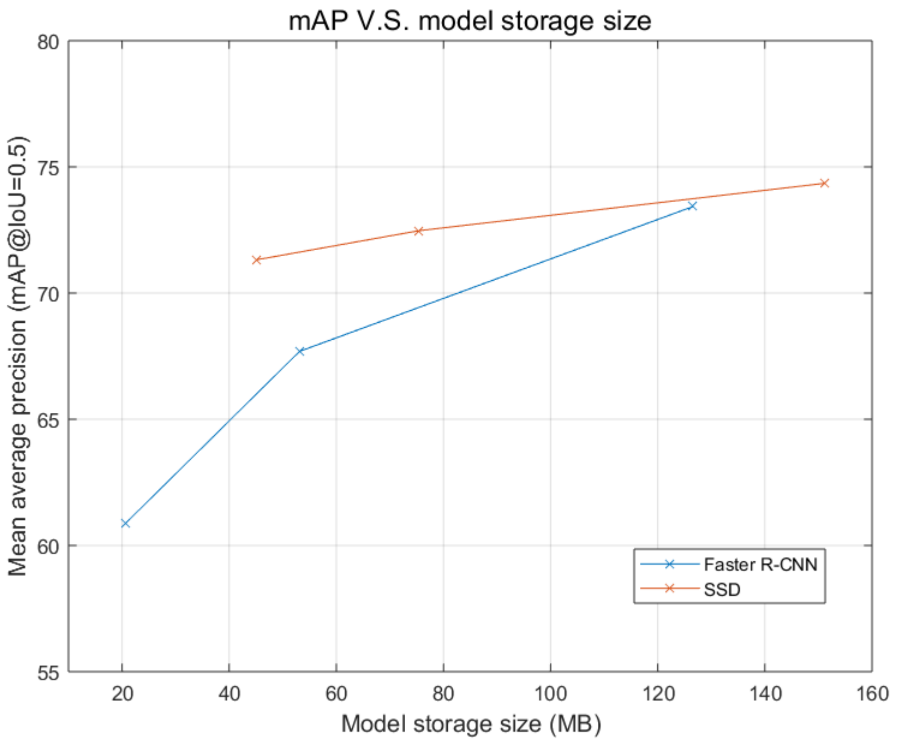


Figure 6.2: Brief trade-offs between accuray and storage size for different object detectors with 5-bits PA quantization scheme.

From Figure 6.2, we can observe that the detection accuracy for both SSD and Faster R-CNN were growing with the increasing number of full-precision layers (or model storage) as expected. It is clear that the two-stages Faster R-CNN models tend to have a sharper improvement in detection accuracy when we increase the number of parameters to be kept in full-precision. This means that Faster R-CNN can benefit more when increasing the number of high precision layers than the SSD. On the other hand, the performance of single-stage SSD models tends to be saturated when continually increasing the number of full-precision parameters, where only slight improvement on mAP can be observed when changing the model with approximately 30% storage size to full-precision model. This illustrated taht the SSD models tend to have more stable performances than Faster R-CNN models when we decrease the precision of the network parameters. In other words, the results suggested that SSD like models tend to be more robust to binary quantization errors as also discussed in previous chapters.

7 Conclusions

7.1. Summary

In this project, attempts had been made to adopt binary quantization approaches to object detection tasks. Different accuracy-preserving multiple bits quantization schemes were studied and applied to two-stages Faster R-CNN and single-stage SSD models, which are two of the most popular object detectors in real-world applications. The results were then evaluated on the PASCAL VOC data set. Known as one of the state-of-the-art binary quantization schemes, ABC-Net based approaches were first discussed and applied to compress the full-precision object detectors. And an unpublished PA approach proposed by Zhu et al. from our group was then applied and compared with the existing ABC-Net scheme on object detection problems. A three-stages training process was introduced to train the quantized object detectors, which led to higher detection accuracy than the normal two-stage training. Moreover, as for inference, we specified a practical way to calculate the ABC and PA schemes based binary convolution. The results were evaluated on PASCAL VOC data set.

For only weights binarization, ABC-Nets and PA schemes were both able to quantize the object detectors. The quantized binary models can still achieve acceptable performances on PASCAL VOC data set with the maximum accuracy reduction of approximate 6 mAP, and with the compression rate from approximately 26% to 50%. Observed from the experiment results, ABC-Nets tend to have more stable performances to quantize the models in different depths. Although PA schemes slightly outperformed ABC scheme for shallower networks (e.g. ResNet-18 based Faster R-CNN), it showed a larger mAP reduction than ABC scheme for deeper ResNet-50 and ResNet-101 based detectors. However, PA schemes can train the models with approximately 4.8 times faster than ABC scheme with the same setup. Moreover, the attempts to further reduce the models' storage sizes and increase the compression rate were also made by quantizing over 99% of the network parameters to binary representation. As a result, the ResNet-50 based Faster R-CNN models can be compressed to the storage size within 21 MB, but with a lager accuracy reduction in mAP. While the SSD models were compressed to approximately 45MB with only about 3 mAP reduction.

Furthermore, the explorations were also been made to quantize both the weights and inputs of the object detectors for further speedups and run-time resources saving. We quantized both the inputs and weights for ResNet-18 based Faster R-CNN and ResNet-50 based SSD with ABC and PA scheme. Unfortunately, ABC-Net based inputs quantization failed to achieve sufficient detection performances for both of the baseline models. However, the PA scheme is able to binarize the models with still acceptable detection accuracy on PASCAL VOC data set.

Compared to the existing works, with only weight quantization, the PA and ABC based object detection models can still preserve competitive detection accuracy. For instance, the 5-bits ABC-Net and ResNet-50 based SSD-300 achieved 72.47 mAP, which is only around 0.73 mAP lower than the original full-precision Faster R-CNN implementation [39], and around 1.9 mAP lower than the full-precision SSD-300 implementation [33] but with smaller model sizes. In terms of both accuracy and storage, the binary ResNet-50 based Faster R-CNN was able to outperform the real-time full-precision Tiny YOLO v2 [36], where Tiny YOLO v2 achieved 57.1 mAP and required 60.5 MB storage. Moreover, the PA-based binary weights and binary models achieved better performances than the existing binary weights and inputs object detectors in [42], which is 45.6 mAP on PASCAL VOC07 *test* set.

7.2. Future work

As for further work, there are several aspects that can be considered. Firstly, in terms of accuracy, although acceptable performances were achieved, the accuracy reductions from the full-precision models were still significant even with parts of the network layers kept in full-precision, especially when the inputs were binarized. Thus, further training techniques and more advanced binarization approaches are needed. For instance, attempts, like introducing an additional cost function to punish the quantization errors, or adopting a layer-by-layer training, could be reasonable ways to improve the performances. Secondly, both ABC and PA schemes are multiple-bits quantization methods, further compression can be obtained by using 1-bit compression such as BWN and XNOR scheme [35]. Thirdly, this project only evaluated the accuracy and model storage on GPU based machines to evaluate the possibilities to utilize binary quantization on object detectors, further evaluation and optimization of run-time memory, as well as computational cost (e.g. FLOPs) on corresponding hardware platforms are required.

Bibliography

[1] Hande Alemdar, Nicholas Caldwell, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. Ternary neural networks for resource-efficient AI applications. *CoRR*, abs/1609.00222, 2016. URL `http://arxiv.org/abs/1609.00222`.

[2] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *CoRR*, abs/1512.08571, 2015. URL `http://arxiv.org/abs/1512.08571`.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007. URL `http://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf`.

[4] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL `http://arxiv.org/abs/1308.3432`.

[5] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. *CoRR*, abs/1504.04788, 2015. URL `http://arxiv.org/abs/1504.04788`.

[6] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing convolutional neural networks. *CoRR*, abs/1506.04449, 2015. URL `http://arxiv.org/abs/1506.04449`.

[7] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. pages 609–622, 2014. URL `https://ieeexplore.ieee.org/document/7011421/`.

[8] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016. URL `http://arxiv.org/abs/1602.02830`.

[9] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. 2014. URL `http://arxiv.org/abs/1412.7024`. cite arxiv:1412.7024v5.pdfComment: 10 pages, 5 figures, Accepted as a workshop contribution at ICLR 2015.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015. URL `http://arxiv.org/abs/1511.00363`.

[11] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016. URL `http://arxiv.org/abs/1605.06409`.

[12] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. 1:886–893 vol. 1, June 2005. ISSN 1063-6919. doi: 10.1109/CVPR.2005.177.

[13] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[14] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017. URL `http://arxiv.org/abs/1701.06659`.

[15] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017. URL `http://arxiv.org/abs/1701.06659`.

[16] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL `http://arxiv.org/abs/1504.08083`.

[17] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL `http://arxiv.org/abs/1311.2524`.

[18] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. *CoRR*, abs/1502.02551, 2015. URL `http://arxiv.org/abs/1502.02551`.

[19] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. 2015. URL `http://arxiv.org/abs/1510.00149`. cite arxiv:1510.00149Comment: Published as a conference paper at ICLR 2016 (oral).

[20] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015. URL `http://arxiv.org/abs/1506.02626`.

[21] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[23] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL `http://arxiv.org/abs/1703.06870`.

[24] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL `http://arxiv.org/abs/1704.04861`.

[25] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL `http://arxiv.org/abs/1608.06993`.

[26] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016. URL `http://arxiv.org/abs/1611.10012`.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[29] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area v2. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 873–880. Curran Associates, Inc., 2008. URL `http://papers.nips.cc/paper/3313-sparse-deep-belief-net-model-for-visual-area-v2.pdf`.

[30] Fengfu Li and Bin Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016. URL `http://arxiv.org/abs/1605.04711`.

[31] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. 2014.

[32] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *CoRR*, abs/1711.11294, 2017. URL `http://arxiv.org/abs/1711.11294`.

[33] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. pages 21–37, 2016.

[34] David G. Lowe. Object recognition from local scale-invariant features. pages 1150–, 1999. URL `http://dl.acm.org/citation.cfm?id=850924.851523`.

[35] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016. URL `http://arxiv.org/abs/1603.05279`.

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

[37] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[38] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL `http://arxiv.org/abs/1506.02640`.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. pages 91–99, 2015. URL `http://dl.acm.org/citation.cfm?id=2969239.2969250`.

[40] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *Computer Science*, 2014.

[42] S. Sun, Y. Yin, X. Wang, D. Xu, W. Wu, and Q. Gu. Fast object detection based on binary deep convolution neural networks. *CAAI Transactions on Intelligence Technology*, 3(4):191–197, 2018. ISSN 2468-2322. doi: 10.1049/trit.2018.1026.

[43] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013. doi: 10.1007/s11263-013-0620-5. URL `http://www.huppelen.nl/publications/selectiveSearchDraft.pdf`.

[44] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. 2001.

[45] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016. URL `http://arxiv.org/abs/1606.06160`.