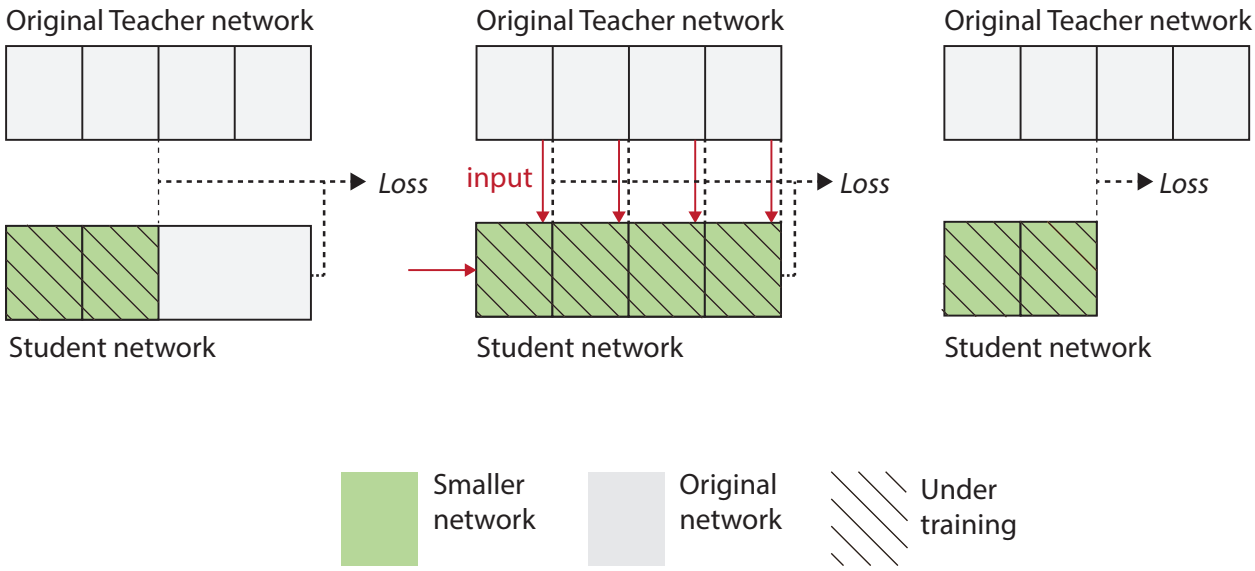

Training Binary Deep Neural Networks Using Knowledge Distillation

SOFIA LUNDBORG

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:57118

Training Binary Deep Neural Networks Using Knowledge Distillation

SOFIA LUNDBORG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Chalmers University of Technology
Gothenburg, Sweden 2020

Training Binary Deep Neural Networks Using Knowledge Distillation
SOFIA LUNDBORG

Supervisor: Giovanni Volpe, Department of Physics at Gothenburg University
Examiner: Giovanni Volpe, Department of Physics at Gothenburg University

Master's Thesis 2020:57118
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of three different knowledge distillation techniques

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Training Binary Deep Neural Networks Using Knowledge Distillation
SOFIA LUNDBORG
Chalmers University of Technology

Abstract

Binary networks can be used to speed up inference time and make image analysis possible on less powerful devices. When binarizing a network the accuracy drops. The thesis aimed to investigate how the accuracy of a binary network can be improved by using knowledge distillation. Three different knowledge distillation methods were tested for various network types. Additionally, different architectures of a residual block in ResNet were suggested and tested. Test on CIFAR10 showed an 1.5 % increase in accuracy when using knowledge distillation and an increase of 1.1 % when testing on ImageNet dataset. The results indicate that the suggested knowledge distillation method can improve the accuracy of a binary network. Further testing needs to be done to verify the results, especially longer training. However, there is great potential that knowledge distillation can be used to boost the accuracy of binary networks.

Keywords: deep neural networks, knowledge distillation, binary neural networks.

Acknowledgements

I want to thank my supervisors at Bit Addict Karl Svensson, Fredrik Ring and Niclas Wikström for guiding me through the whole process of writing this thesis. You have helped me find a relevant subject, structure the work, and contribute with helpful knowledge and insights. I also want to thank my supervisor och examiner Giovanni Volpe for making this thesis possible. Lastly, I want to express gratitude to all the staff at Bit Addict for your warm welcome and great company - making the whole experience more enjoyable.

Sofia Lundborg, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1

Introduction

In recent years, there have been great advances in the field of image analysis, including tasks such as image classification, object detection, face recognition, and segmentation. This has been possible thanks to the development of deep Convolutional Neural Networks (CNNs). However, most of these deep CNNs require powerful hardware with sufficient memory and computing resources, for example GPU. Meanwhile, there are new technologies e.g. virtual reality, augmented reality and other small devices that could benefit from image analysis, but are not equipped with a powerful GPU. To be able to run real-time image analysis tasks on these less powerful devices one would need to reduce inference time as well as memory size of the CNNs. There are several methods to reduce memory size and inference time. One way is to design compact networks [1][2]. Other methods are pruning [3], quantization [4][5], or a combination of them [6]. This project has however studied another approach - binarization.

Binarization is the extreme case of quantization where all weights are set to either 1 or -1 [7]. To take this one step further, one can also binarize the inputs of every convolutional layer in the CNN [8]. This makes it possible to replace the matrix-matrix multiplication in convolution layers by logical operations and in turn reduce inference time substantially. It has been shown that the inference time on a CPU can be accelerated by $\approx \times 50$ by binarization [9]. It can also reduce memory size up to $\times 32$ [8].

When a network is binarized, there is a substantial loss in accuracy. Several methods to close the gap between the binary and full precision networks have been proposed [10][11][12][13]. A method for improving the accuracy of a network, not specific to binary networks, is knowledge distillation, where a student network is trained with the help of a larger teacher network by mimicking the teacher network's output in certain layers. There are multiple measures to compare how similar two networks are - their softmax output [14], their distribution in intermediate layers [15][16], their flow between layers [17], or simply the mean square error between feature responses [18][19][20].

In classical knowledge distillation methods, the student network is a smaller full precision network in comparision to its teacher network. One could say that the student network has a lower representational capability (the functions which the model can learn) compared to the teacher network. The same is true for a binarized

network and its corresponding full precision network. It is therefore possible that knowledge distillation can be used for binary networks as a smart way of initialising the network.

This thesis aims to investigate how knowledge distillation can be used to improve the accuracy of a binary deep convolutional neural network. The following research question will be answered:

- Does some knowledge distillation technique works better for binary networks than others?
- Can the architecture of a network be modified to improve the effect of knowledge distillation training?
- Are some network types better suited for knowledge distillation?

1.1 Limitations

The main advantage of using networks that binarize both its weights and its inputs is that the matrix-matrix multiplication in the convolutions layers can be replaced by logical operations. To make this binary multiplication efficient on a CPU or other computational device, one needs to manage the execution scheduler and how memory is stored. Methods for this have been developed [9][21], but it will not be implemented in this project. Instead, the focus will be to investigate binary networks' ability to learn to make accurate predictions.

There are multiple types of binary networks [10][11][12][13], as well as ternary networks (inputs set to 0, 1, -1) [22]. The ideas presented in this project are not limited to a specific binary network type. However, in this work, only Xnor-nets [10] and Xnor++-nets [11] will be evaluated, incorporating ideas from Bi-Real nets [12].

The list of different network architectures is long and constantly growing - leNet, VGG, AlexNet, DarkNet, ResNet to name a few. ResNet is a widely used network that has shown being suitable for binary networks [10][11][13][12]. To make the results of this work easy to compare with others, only ResNet [23] will be studied in this thesis.

Due to time limitation and long training times, no hyper-parameter optimisation was made and the training was kept relatively short.

2

Theory

The aim of this thesis is to investigate the combination of binary network and knowledge distillation. This section will present a background of previous research in the fields of binary networks (Section 2.1) and knowledge distillation (Section 2.2).

2.1 Binary neural networks

How binary networks are trained are described in this section, both for networks using binary weights as well as networks using both binary weights and inputs. In Section 2.1.3 a few methods to improve the accuracy of binary networks are described.

2.1.1 Using binary weights

It is not obvious what a binary neural network is. What part of the network is binarized - is it the weights? the inputs? Courbariaux et al. introduced the concept with their network BinaryConnect [7] which binarizes the weights in the convolutional layers. The weights are real-valued during training but are set to binary values in the forward propagation, using the sign function:

$$w_{binary} = \begin{cases} +1 & \text{if } w_{real} \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{2.1}$$

The backward propagation is done with the binary weights, but it is the real-valued weights that are updated. The reason being that the changes during the parameter update are too small to change the sign on a binary value, and the weights would remain unchanged. The real-valued weights are binarized again in the forward propagation for the next mini-batch. Since the magnitude of the real-valued weights does not affect the sign, the weights are clipped to remain in the interval $[-1, 1]$ to avoid growing unnecessarily large (with no impact). See Algorithm 1.

Algorithm 1 Training of BinaryConnect network where w_{real} is the real valued weights, b is the biases, $t+1$ is the next time step and η is the learning rate.

for all mini batches **do**

1. **Forward propagation:**
   $w_{binary} = \text{sign}(w_{real})$ for all layers.
   Propagate forward and get loss $\mathcal{L}$ using the binary weights
2. **Backward propagation:**
   Calculate $\frac{\partial \mathcal{L}}{\partial w_{binary}}$ and $\frac{\partial \mathcal{L}}{\partial b}$ for all layers.
3. **Update parameters:**
   $w_{real}^{t+1} = \text{clip}\left(w^t - \eta \frac{\partial \mathcal{L}}{\partial w_{binary}}\right)$
   $b^{t+1} = b^t - \eta \frac{\partial \mathcal{L}}{\partial b}$

end for

When the training is finished, the real-valued weights can be removed and we are left with a network with reduced memory and inference time.

2.1.2 Using binary weights and inputs

As a further development of BinaryConnect, Courbariaux et al. proposed a network with both binary weights and inputs, BNN [8]. The input to every convolutional layer is binarized, i.e. the sign function is applied to the input. Binarization makes it possible to replace multiplication in the convolutional layers with the logical operators $XNOR$ and *bitcount* [8]. $XNOR$ is a logical gate which follows the rules:

input	input	XNOR
1	1	1
1	-1	-1
-1	1	-1
-1	-1	1

Table 2.1: *Rules for XNOR gate.*

bitcount simply sum the vector. Figure 2.1 visualises how a binary multiplication can be substituted with $XNOR$ and *bitcount*.

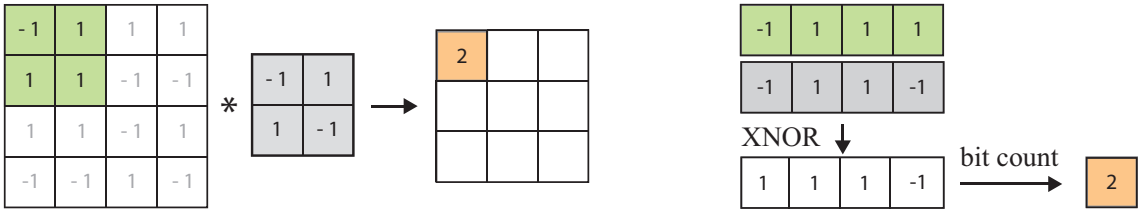

Figure 2.1: *Illustration of how matrix multiplication is replaced by applying XNOR-operation on the two vector/matrices and then summing the results with bitcount operator. To the left, we have a regular convolution where the values of the input and weights are element-wise multiplied and then summed. To the right, the corresponding XNOR-botcount operation is shown.*

2.1.2.1 Efficiency of binary CNNs

The matrix multiplications in the convolutional layers are the most computationally demanding and time-consuming task during inference [24]. Different theoretical speed-up factors has been reported. The calculated potential speed up varies depending on the network, as well as how the efficiency is calculated. Some only look at the speed up for the convolutional layer - $\times 23$ [8], $\times 58$ [10], $\times 64$ [11], while others look at the complete network - $\times 7$ [8] on an MLP network with 2048 units, $\times 11$ on ResNet18 and $\times 19$ on ResNet34 [12].

The actual efficiency improvements also depend on the computing device (CPU/GPU etc.) and the framework (TensorFlow/Pytorch etc.). Xu et al. implemented a computing kernel for Python storing the binary weights in unsigned ints [25]. However, the performance of their kernel only increased $\approx \times 1.5$ on a CPU and performed worse than Pytorch in-build matrix multiplication on a GPU. Hu et al. also stored the weights in unsigned ints, but they used locally-aware layout (not the conventional image-to-column method) and vector parallelism to achieve $\times 50$ acceleration on a CPU [9]. Zhang et al. have implemented another binary framework for ARM-devices, achieving $\approx \times 3.5$ compared to TensorFlow light and $\approx \times 20$ compared to Caffe on Bi-Real 18 network [21].

2.1.2.2 Gradient approximation in backward propagation

The training of a binary network is not trivial. The derivative of the sign function is equal to zero everywhere except at origo, where it is not defined (impulse function). This makes the conventional back-propagation training problematic since the gradient of the loss function with respect to the weights will be zero for all layers except the last. Let $\mathcal{L}$ be the loss, A_{real} the activation function for real-valued output of the previous layer, and A_{binary} the binarization function, i.e. sign-function. The derivative of the loss with respect to A_{real} can be written as

$$\frac{\partial \mathcal{L}}{\partial A_{real}} = \frac{\partial \mathcal{L}}{\partial A_{binary}} \frac{\partial A_{binary}}{\partial A_{real}} = \frac{\partial \mathcal{L}}{\partial A_{binary}} \delta(A_{real}) \tag{2.2}$$

where $\delta(A_{real})$ is the delta-function, i.e.

$$\delta(x) = \begin{cases} \infty & \text{if } x = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

In the binary network proposed by Courbariaux et al. $\delta(A_{real})$ is approximated as

$$\frac{\partial A_{binary}}{\partial A_{real}} \approx \begin{cases} 1 & \text{if } -1 \leq A_{real} \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

This is then used in the backward propagation. The approximation of the gradient is illustrated in Figure 2.2, and its corresponding function in Figure 2.3.

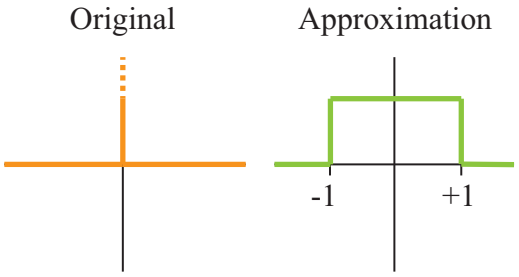

Figure 2.2: *The left figure shows the gradient of the sign function (impulse function) and the right figure its approximation which is used in backward propagation.*

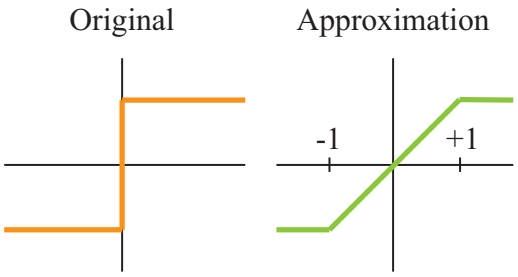

Figure 2.3: *The left figure shows the sign function and the right figure its approximation.*

Equation 2.4 is the standard way of approximating the gradient (used in BNN, XNOR, XNOR++ networks) but alternatives have been studied. In Bi-Real networks, the gradient of the sign function is approximated by a piecewise linear function according to Equation (2.5) [12].

$$\frac{\partial A_{binary}}{\partial A_{real}} \approx \begin{cases} 2 + 2A_{real} & \text{if } -1 \leq A_{real} \leq 0 \\ 2 - 2A_{real} & \text{if } 0 \leq A_{real} \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

2.1.3 Improving accuracy of binary networks

Courbariaux et al. showed that it was possible to train a binary network for image classification on the MNIST and CIFAR10 dataset. However, when trained with ImageNet there was a large drop in top 1 accuracy - from 56.6 % to 27.9 % on AlexNet [10]. To close this gap, several methods have been proposed. One way is to change the networks architecture (Section 2.1.3.1), another to add scaling factors (Section 2.1.3.2).

2.1.3.1 Network architecture

A common network architecture is ResNet [23] and it is used in almost all binary network implementation. ResNet is particularly suitable for binary network due to the shortcuts (residual connections). ResNet consists of blocks containing two convolutional layers each followed by a batch normalisation layer and a ReLU layer. For every block there is a shortcut adding the values of the input to the block with the values obtained after the second batch normalisation layer, see Figure 2.4.

For a binary network, the ReLU function before the convolutional layers are replaced by a sign function. At every sign layer, information is lost - when the real-valued output from the batch normalisation layers are binarized. But the shortcuts in ResNet allows for the information to be stored and propagated throughout the network. For the second convolutional layer, the information is lost in the binarization step. To remedy this, Liu et al. proposed an additional shortcut in their Bi-Real network [12] and achieved a top 1 accuracy of 56.4 % on ImageNet. This idea was used by

Bethge et al, but they also adjusted the network architecture additionally to minimise information loss [13] and achieved a top 1 accuracy of 56.7 %. In Figure 2.4, a ResNet block is visualised in a full precision network (left), in a binary network (middle) and in a Bi-Real network (right).

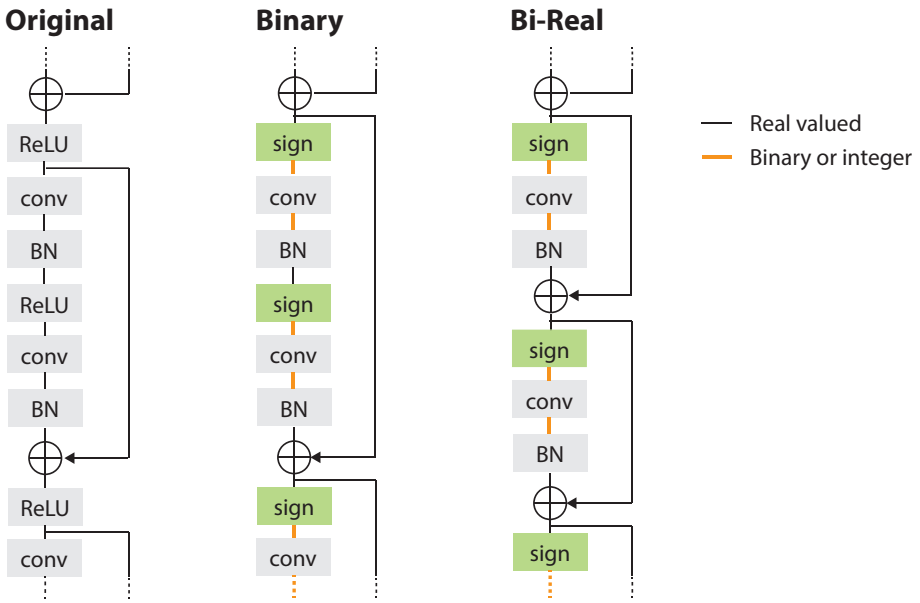

Figure 2.4: *To the left - a block of ResNet network for a conventional full precision network. In the middle - a binary version of a ResNet block. To the right - a ResNet block with double shortcuts used in Bi-Real networks. The values are binary after the Sign function and integers after the convolutional layers. This is indicated by an orange line. In the other positions, the values are real and it is represented as a black line.*

2.1.3.2 Scaling factors in convolution layers

Instead of modifying the network architecture, Rastegari et al. introduced XNOR-net which uses scaling factors in the convolutional layers to approximate a full-precision convolution better [10]. They calculated scaling factors by computing the mean of input and weights and thereby retain some of the otherwise lost information. The calculation of the scaling factors is visualised in Figure 2.5. The scaling factor connected to the input, k_{ij}, is calculated as the mean over the input channels and the receptive field:

$$k_{ij} = \frac{1}{n}\|X\|_{l1} \tag{2.6}$$

where X is a sub-tensor of the input with the same size as the weight tensor. The calculation result for each sub-tensor of the input is collected in the matrix K.

The scaling factor connected to the weights, α is computed as the total mean for each weight tensor - one for each output channel:

$$\alpha = \frac{1}{n}\|W\|_{l1} \tag{2.7}$$

where W is the weight tensor.

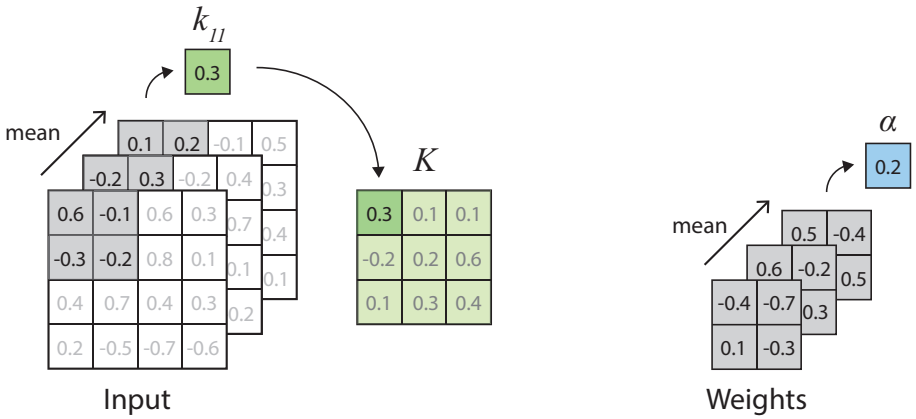

Figure 2.5: *Computation of scaling factors used in XNOR-nets.*

The full precision convolution is approximated by

$$I * W \approx K \cdot \text{sign}(I) * \alpha \text{sign}(W) \tag{2.8}$$

where $*$ is convolution, $\cdot$ is scalar product, I is the input, W is a weight tensor related to one output channel and K and α are the scaling factors calculated by Equation (2.6) and (2.7).

In XNOR-nets, the two scaling factors, α and K are computed analytically. In contrast to this Bulat et al. suggested to fuse these factors into one, and instead of calculating the scaling factors, learning them via backpropagation [11]. In XNOR++ networks the convolution is approximated as

$$I * W \approx \Gamma \cdot (\text{sign}(I) * \text{sign}(W)) \tag{2.9}$$

where $*$ is convolution, $\cdot$ is scalar product, I is the input and W is a weight tensor related to one output channel and Γ is a matrix containing learnable scaling factors. The output dimensions of the convolution is $\mathbb{R}^{o \times h_{out} \times w_{out}}$ where o is number of output channels, h_{out} is the output height, w_{out} the output width.

Γ has the same dimensions as the convolution output, $\mathbb{R}^{o \times h_{out} \times w_{out}}$. There are different ways of constructing Γ. One possibility is to let every element in Γ be a learnable parameter. Another way is to let Γ be the outer product of three vectors

$$\Gamma = \alpha \otimes \beta \otimes \gamma, \quad \alpha \in \mathbb{R}^o, \beta \in \mathbb{R}^{h_{out}}, \gamma \in \mathbb{R}^{w_{out}} \tag{2.10}$$

This method has fewer learnable parameters and are therefore less prone to overfitting.

2.2 Knowledge Distillation

Knowledge distillation is a method for boosting the accuracy of a network, by learning with the help of a pre-trained large teacher network which performs well on the task. It was first introduced by Hinton et al. where the student network learned to mimic the softmax output of the teacher network instead of the true label - it distils knowledge from the teacher network [14]. The idea of Hinton et al.'s work was to make use of the information in the relative probabilities of incorrect answers in the teacher model since it can give an idea of how the teacher model tends to generalise. The softmax is a value $o_i \in [0, 1]$ and is computed as

$$p_i = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}} \tag{2.11}$$

where T is called temperature - the larger the temperature, the softer the outputs (the output elements have more similar values). A regular loss is the cross-entropy loss:

$$\mathcal{L} = -\sum_i q_i \log(p_i) \tag{2.12}$$

where p_i is the softmax output and $\boldsymbol{q}$ is the target ($q_i \in \{0, 1\}$). Hinton et al. instead uses the softmax output from the teacher network as $\boldsymbol{q}$.

Romero et al. developed the idea of distilling knowledge further by proposing to make the student network mimic the teacher network in intermediate layers [26]. Several comparison measures between feature responses have been proposed. Zagoruyko et al. suggested to compare the absolute values of the feature response (flattened to a 2D tensor) as a measure of the importance (attention) of a single neuron w.r.t. a specific input [15]. Huang et al. proposed a similar, but more general, measure to match the distributions of student and teacher network [16].

2.2.1 Alternative training schemes

The previously mentioned knowledge distillation methods train in a conventional way but with an unusual loss function; either containing a term including soft output or feature responses from intermediate layers. As for any optimisation of a deep CNN, the problem is non-convex and the search space huge. To make the search space smaller and therefore the training more stable, alternative training schemes have been proposed.

2.2.1.1 *Method I* - progressive learning

Wang et al. suggested a progressive approach, where only a sub-part of the network is trained at every stage [20], see *method I* in Figure 2.6 for an illustration of the method. The student network and the teacher network is divided into sub-networks. Even though the student network is smaller, the features of the student and teacher network need to have the same dimensions at the end of the sub-network. This is not a problem when the student network is a binarized version of the teacher network since the architecture is identical.

Assume the network is divided into a few parts. The first sub-network consists of the first part, i.e. the first layers of the network. The second sub-network consists of the first two parts and so on. The loss is divided into two terms - a local loss $\mathcal{L}_{local}$ and a classification loss $\mathcal{L}_{cls}$ (cross entropy-loss). The two losses are balanced by a scaling factor β, see Equation (2.13).

$$\mathcal{L}_I = \beta\mathcal{L}_{local} + (1-\beta)\mathcal{L}_{cls} \tag{2.13}$$

where $\beta \in [0, 1]$. The local loss is the mean square error between the feature responses of the teacher's and student's sub-network respectively:

$$\mathcal{L}_{local} = \frac{1}{n}\sum_i (x_i^{teacher} - x_i^{student})^2 \tag{2.14}$$

where $\boldsymbol{x}^{teacher}$ and $\boldsymbol{x}^{student}$ are the outputs from the teacher and student network in an intermediate layer and n is the number of elements in the output.

The input is propagated through the complete network and the gradients are computed with respect to the total loss, but only the active sub-network is updated. The sub-networks are trained successively, and in the last step, all layers in the network are trained.

2.2.1.2 *Method II* - simultaneously learning

In the method proposed by Wang et al. (*method I*) the individual parts of the networks are trained multiple times. Instead, Koratana et al. suggested a method where all parts of the network are trained simultaneously, see *method II* in Figure 2.6 [19]. The loss consists of two terms - a local loss, $\mathcal{L}_{local}$, and a knowledge distillation loss, $\mathcal{L}_{KD}$, balanced by a scaling factor β:

$$\mathcal{L}_{II} = \beta\mathcal{L}_{local} + (1-\beta)\mathcal{L}_{KD} \tag{2.15}$$

where $\alpha \in [0, 1]$. The knowledge distillation loss consist of two parts - the cross-entropy loss of the true label, $\mathcal{L}_{hard}$ and the cross-entropy of the soft output as in equation (2.12), $\mathcal{L}_{soft}$. The two terms are balanced by scaling factor α according to

$$\mathcal{L}_{KD} = \alpha\mathcal{L}_{hard} + (1-\alpha)\mathcal{L}_{soft} \tag{2.16}$$

The local loss is calculated as the mean square error as in Equation (2.14). The main difference to *method I* is that the input to every sub-network is not the output from the previous layer in the student network but the previous layer in the teacher network. This makes it possible to train all sub-networks simultaneously while still reducing the search space.

2.2.1.3 *Method III* - progressive learning without scaling factors

In *method I* and *II* there are multiple hyper-parameters that needs to be optimised (e.g. temperature and balancing terms). Gao et al. proposed a progressive method

where there is only one term in the loss - the mean square error according to Equation (2.14) [18], i.e.

$$\mathcal{L}_{III} = \frac{1}{n}\sum_i (x_i^{teacher} - x_i^{student})^2 \tag{2.17}$$

The student network is trained one sub-network at a time, gradually reaching the final size.

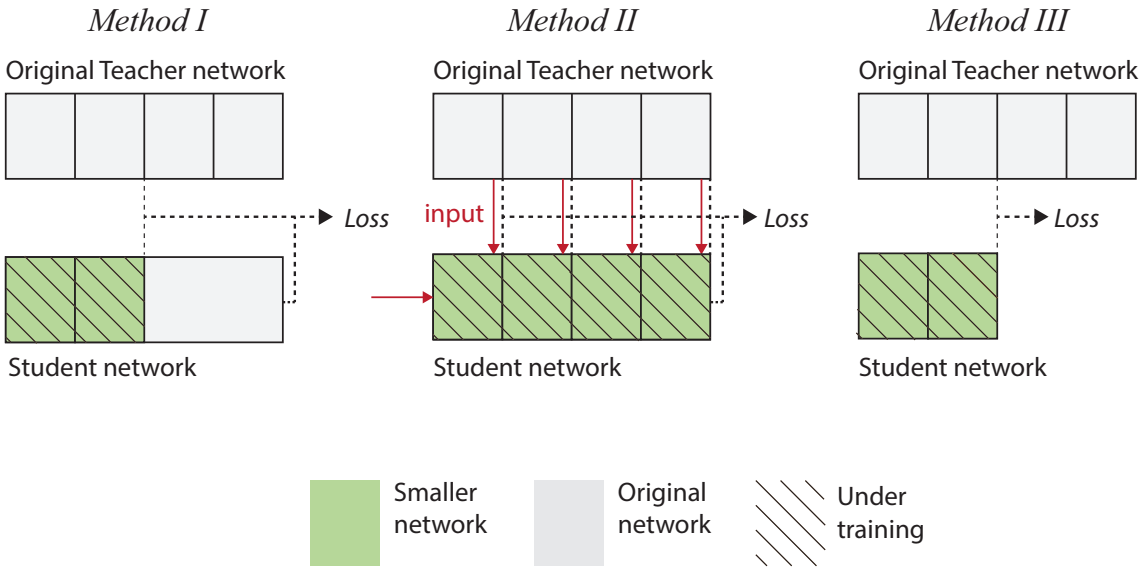

Figure 2.6: *Schematic figure of method I [20], method II [19] and method III [18]*

2.2.2 Binarization and knowledge distillation

Knowledge distillation in combination with binarization has been studied by Xu et al.[27]. In their work, they only binarized the weights (and not the inputs as done in this work, which makes their problem easier). They tested it on very large networks used for object detection such - YOLO [28] and MobileNet [29]. They used a total loss which included both a loss from the final layer and mean square error loss from intermediate layers. In contrast to the training schemes presented in Section 2.2.1, they binarized the network in reversed order with respect to inference. The logic behind the reversed order is that binarizing the weights in the last layers had a small effect on the network while binarizing the first layers had a large negative effect on the performance of the model [30]. The resulting accuracy dropped from the 78 % (full precision network) to 72 % (their binary network) on the KITTI dataset [31], which can be seen as a relatively small decrease.

3

Method

The aim of this thesis was to investigate if the accuracy of a binary network can be improved by using a knowledge distillation training method. There are several different knowledge distillation methods (Section 2.2), binary network types (Section 2.1.3.2) and binary network architectures (section 2.1.3.1). To evaluate the effects of these different cases a relatively small dataset, CIFAR10, was used. The tests using CIFAR10 is described in Section 3.1. In able to compare the result of this project with other work on binary networks, it is necessary to use the same dataset and the most commonly used is ImageNet. In Section 3.2, tests performed with ImageNet are described. The models are built in the Pytorch framework [32].

3.1 Tests using CIFAR10

CIFAR10 is a dataset consisting of 32×32 colour images in 10 classes, with 60,000 images per class [33]. There are 50,000 training images and 10,000 validation images. The images were normalised w.r.t. mean and variance for the three colour channels. When training, the input was randomly flipped horizontally and the mini-batch size was set to 256.

The network used together with CIFAR10 was ResNet20. It consists of 3 large blocks - conv2_x, conv3_x and conv4_x each consisting of 3 residual blocks which in turn consists of two convolutional layers and a residual connection. The original structure of a ResNet block can be seen to the left in Figure 2.4. Before the first large block, there is a convolutional layer and after the last large block, there is an average pooling layer followed by a fully-connected layer. The architecture of ResNet20 is presented in Table 3.1.

Layer Name	Output size	ResNet20
conv1	32 x 32 x 16	3 x 3, 16, stride 1
conv2_x	32 x 32 x 16	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix}$ x 3, stride 1
conv3_x	16 x 16 x 32	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix}$ x 3, stride 2
conv4_x	8 x 8 x 64	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix}$ x 3, stride 2
average pool	1 x 1 x 64	8 x 8 average pool
fully connected	10	64 x 10 fully connected

Table 3.1: *Architecture for ResNet20. The network consists of an initial convolutional layer, followed by 3 large blocks each containing 3 residual blocks. One residual block is made up of 2 convolutions layers with kernel size 3 and corresponding batch norm layers as well as a residual connection (shortcut). Following the residual blocks there is an average pooling layer and lastly a fully connected layer.*

The first and last layer in ResNet20 was not binarized. This is common practice for binary networks because it does affect the networks ability to predict while it does not affect the speed significantly since the first layer has a small channel size (3) and the last layer is a fully connected layer (equivalent to filter size of 1 in a convolutional layer) [10].

During training the Adam optimiser was used. Similarly to stochastic gradient descent, SGD, it uses the first-order gradient to update the weights. However, in contrast to SGD, Adam calculates an adaptive learning rate for each individual weight from estimates of first and second moments of the gradients. Using Adam reduces the need for tuning of hyperparameters and it has shown to work better than SGD for binary input networks [8].

For all tests using a knowledge distillation method, the ResNet20 was divided in the same way - into the three large blocks: conv2_x, conv3_x and conv4_x. The training was divided into two parts - knowledge distillation training followed by a conventional cross-entropy training. In the cross-entropy training, the network was initialised to the network trained by knowledge distillation instead of randomly initialised. The model was trained for 120 epochs in the cross-entropy training. Initial learning rate for cross-entropy training was set to 0.01 and the learning rate was decreased by a factor of 10 at epoch 70, 90, 100 and 110. The ResNet block used was the double shortcut with ReLU as described in Section 3.1.1.

Three tests were made using CIFAR10 - network architecture, knowledge distillation method and network type. The best result from the previous tests were used in the following test, i.e. not all combinations were tested. See Figure 3.1 for a schematic figure of the progressive testing.

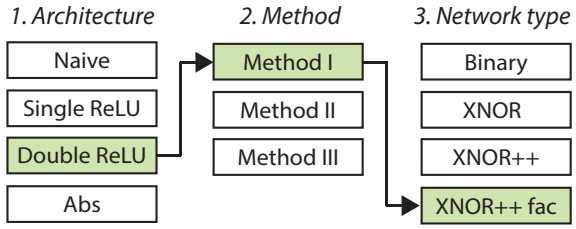

Figure 3.1: *First, different architectures of ResNet block is tested. Then knowledge distillation methods are tested using the best result from the previous tests. Lastly, different network types are tested.*

3.1.1 Architecture of ResNet block

A block in a full precision ResNet is visualised in Figure 2.4 and it consists of two convolutional layers each followed by a batch norm layer. The first batch norm layer is followed by a ReLU layer while the second is followed by the summation with the residual connection (shortcut). A ReLU layer is then applied to the summed output and the results are saved as the shortcut to the next block. The distribution after the second batch norm layer is on average Gaussian distributed, but this is not the case for the output from the previous layer (the shortcut) since all the negative values are removed by the ReLU layer. This makes the distribution after the summation shift its mean as well as extend the tail for positive values. An example of the distribution of the output before and after the summation of the previous output is shown in Figure 3.3. Figure 3.2 shows where the distributions are sampled.

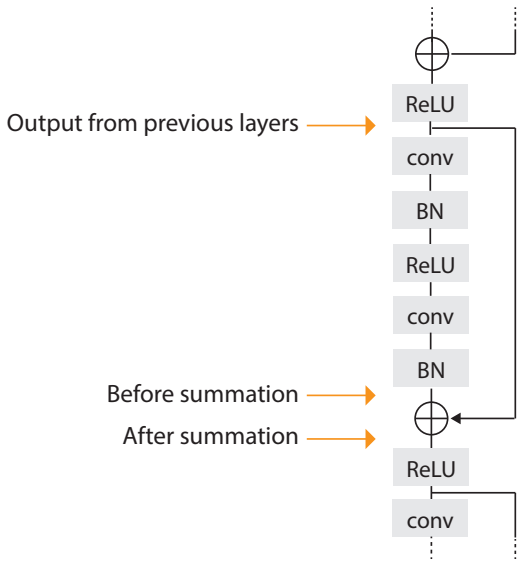

Figure 3.2: *Where the distributions are sampled in Figure 3.3.*

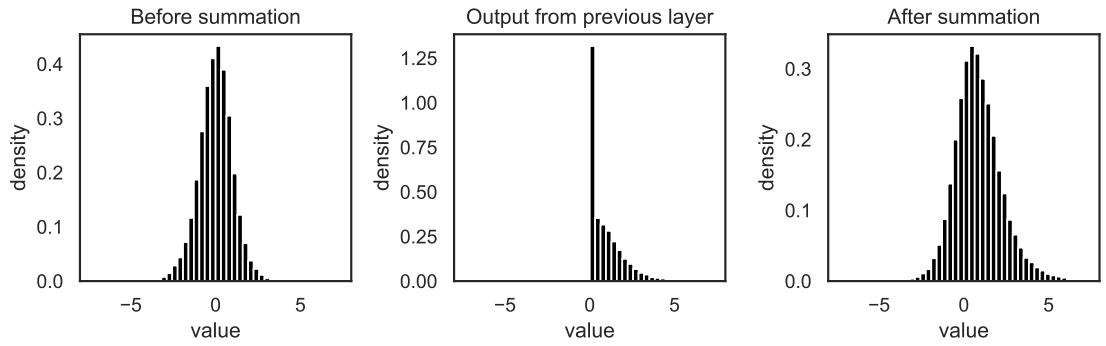

Figure 3.3: *Distributions of the output at different stages in convolutional layer 7. The left figure shows the distribution of output after the second batch normalisation layer, while the middle figure shows the shortcut and the right figure the summation of the two. In the rightmost figure, one can see that the mean has shifted and the tail is extended for positive values*

The non Gaussian distribution poses a problem. Previous works with binary networks, removes the ReLU function which makes the shortcut and the result of the summation Gaussian distributed. If one wants to use a knowledge distillation method that compares intermediate output between the teacher and the student network, it is reasonable that the intermediate output of teacher and student network should have the same distribution. One way of getting the same distributions as the teacher is to re-introduce the ReLU activation in the shortcut, see Figure 3.4 second to the left. However, when applying a ReLU function to the shortcut, information is lost.

Another approach is to take the absolute value of the shortcut and multiply it with a scaling parameter $\alpha \in [0, 1]$ to get similar distributions. What the value of α should be is not clear. The mean of the distribution is shifted towards higher positive values progressively throughout the network, and fewer and fewer values are set to zero by the ReLU function. One possibility is to let α be a learnable parameter for every convolutional layer. See Figure 3.4 for a schematic figure of the suggested architecture.

In Section 2.1.3.1, Bi-real nets and double shortcuts are mentioned. The idea of double shortcuts is that less information is lost in the binary step of the second convolution. The extra shortcut in Bi-Real nets changes the distributions in two ways. First, there is the absence of the ReLU function as previously discussed. Secondly, it increases the variance when summing two Gaussian distributions. In this project, a network with a different type of ResNet block is proposed, which has double shortcuts but results in the same distribution as an original full precision ResNet block. The ResNet block is visualised in Figure 3.4 furthest to the right. In contrast to Bi-Real nets, the two shortcuts differ. One shortcut adds the output from the two batch norm layers and scales the results with $1/\sqrt{2}$ to achieve the same distribution as if this extra shortcut did not exist. The second shortcut is identical to a single shortcut with ReLU activation.

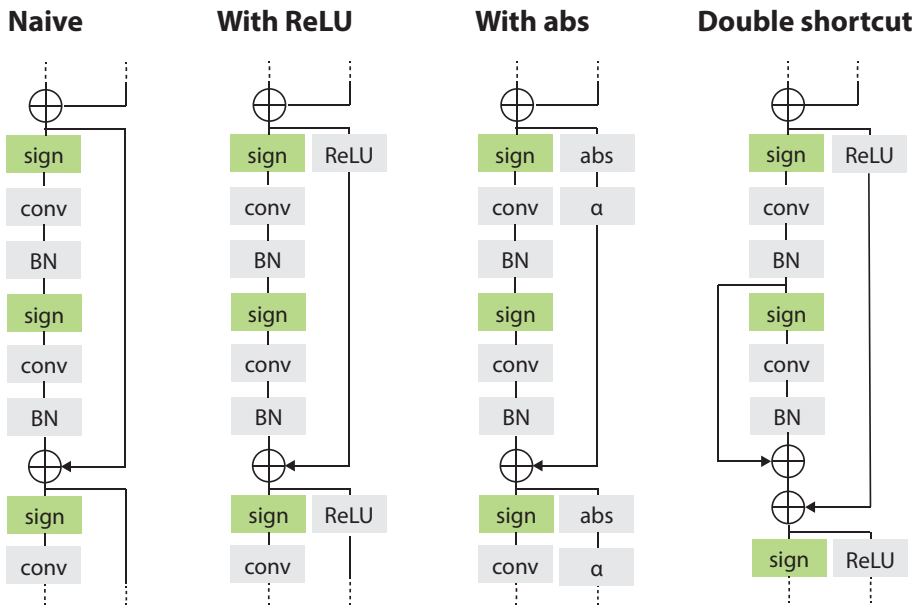

Figure 3.4: *Tested architectures for ResNet block.*

In contrast to the other methods, *method III* uses either cross-entropy loss or mean square error loss and not a combination of the two. The problem of mimicking the response of the teacher network and binary student network in the intermediate layers is more overdetermined compared to the cross-entropy loss in the final layer. To get a better understanding of how the ResNet blocks performs on the different losses *method III* is used for testing.

The four architectures of ResNet blocks shown in Figure 3.4 were tested on knowledge distillation *method III*. When training with *method III*, the network was progressively binarized and trained one large block at a time, i.e. in three parts conv2_x, conv3_x and conv4_x. The initial learning rate for every block was 0.01 and was decreased by a factor of 10 at epoch 25, 30 and 35. *Method III* was followed by a cross-entropy training for 120 epochs. Figure 3.5 shows a schematic figure of which knowledge distillation method and network type that is used in the test.

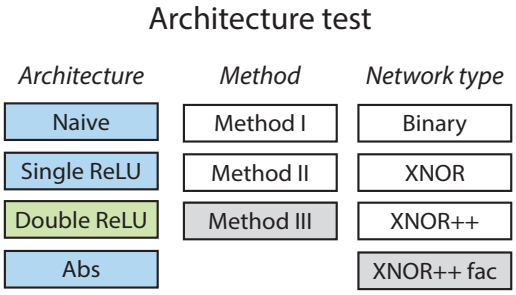

Figure 3.5: *The four different architectures of ResNet block were tested. Knowledge distillation method III was used and the factorised version of XNOR++ scaling factors was used. Double ReLU was used in the following tests.*

3.1.2 Comparison of Knowledge Distillation techniques

As described in the theory Section 2.2, there are multiple knowledge distillation techniques. In many real-life scenarios when image analysis is used, the complexity of the problem and therefore the size of the networks are large. When training a large network the search space is huge. The knowledge distillation methods using alternative training schemes, see Section 2.2.1, can reduce the search space and consequently make the training more stable. In this project, only these methods has been studied, i.e. *method I, II*, and *III* presented in Section 2.2.1.

Testing of *method I* was done where each sub-network was trained for 40 epochs with an initial learning rate of 0.01 which was decreased by a factor 10 at epoch 20, 30 and 35. Multiple values of β was tested, $\beta \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. It was followed by a cross-entropy training for 120 epochs. Figure 3.6 shows a schematic figure of which ResNet block and network type that is used in the test.

Testing of *method III* was done where each sub-network was trained for 40 epochs with an initial learning rate of 0.01 which was decreased by a factor of 10 at epoch 25, 30 and 35. It was followed by a cross-entropy training for 120 epochs.

In *method II* all layers are trained simultaneously and the model was trained for 60 epochs in the knowledge distillation phase. The initial learning rate was set to 0.01 and was decreased by a factor of 10 at epoch 30, 40, 50 and 55. The values for α and the temperature was set to the optimal values found in [19]: $\alpha = 0.95$ and $T = 6$, while multiple values of β was tested, $\beta \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. It was followed by a cross-entropy training for 120 epochs.

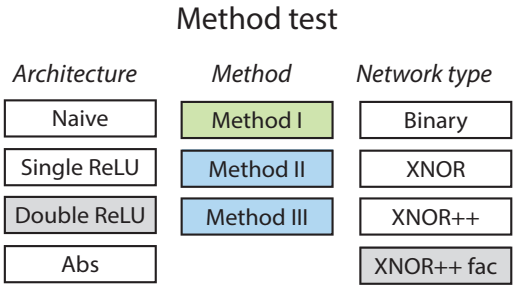

Figure 3.6: *The three different knowledge distillation methods were tested using double ReLU shortcut and factorised XNOR++ scaling factors. Method I was used in the next test.*

3.1.3 Binary network type

Multiple version of binarized networks has been proposed. To evaluate if any specific type works better with knowledge distillation, ordinary binary network [8], XNOR network [10] and XNOR++ network were tested using knowledge distillation *method I*. For XNOR++, both the ways of constructing the scalar parameter Γ was tested, i.e. Γ containing a factor for every value of the output of the convolutional layer or Γ being factorised: $\Gamma = \alpha \otimes \beta \otimes \gamma$. The same parameters, epochs, learning rate etc. were used as in the training of *method I* in Section 3.1.2 and the double ReLU shortcut architecture was used. *Method I* was followed by a cross-entropy training

for 120 epochs. Figure 3.7 shows a schematic figure of which ResNet block and knowledge distillation method that are used in the test.

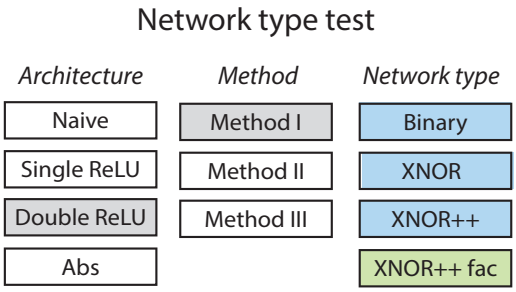

Figure 3.7: *Four different network types were tested using double ReLU shortcut and knowledge distillation method I. The factorised version of XNOR++ network was used in the following ImageNet test.*

3.2 ImageNet test - *Method I* on ResNet18

To compare with other results, *method I* is tested with the common large dataset ImageNet and network ResNet18. ImageNet is a dataset containing colour images of 1000 different classes [34]. The training set consists of a little more than 1.2 million images and the validation set consists of 50,000 images. When training, the images were randomly resized and cropped to the size 224×224 and then randomly flipped horizontally, which is the standard way of augmenting the input. During testing, the images were center-cropped and normalised. During training and testing the images were normalised w.r.t. mean and variance for the three colour channels. The models were trained with mini-batch size 64. The mini-batch size was set to fit on the GPU.

The network used together with ImageNet was ResNet18. Its architecture is shown in Table 3.2. It consists of an initial convolutional layer and a max-pooling layer followed by four main blocks each containing two residual blocks. One residual block is made up of 2 convolutional layers and a residual connection, see Table 2.4. Following the residual blocks is an average-pooling layer and lastly a fully connected layer.

Layer Name	Output size	ResNet18
conv1	112 x 112 x 64	7 x 7, 64, stride 2
conv2_x	56 x 56 x 64	3 x 3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix}$ x 2, stride 1
conv3_x	28 x 28 x 128	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix}$ x 2, stride 2
conv4_x	14 x 14 x 256	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix}$ x 2, stride 2
conv5_x	7 x 7 x 512	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix}$ x 2, stride 2
average pool	1 x 1 x 512	7 x 7 average pool
fully connected	1000	512 x 10 fully connected

Table 3.2: *Architecture of ResNet18. It is made up of the four large blocks conv2_x, conv3_x, conv4_x and conv5_x which each consists of two convolutional layers with kernel size 3. Before the residual blocks, there is an initial convolutional layer and a max pool layer. After the residual blocks, there is an average pooling layer and a fully connected layer.*

During training with *method I*, ResNet18 was divided into the four large blocks conv2_x, conv3_x, conv4_x and conv5_x. The first and last layer in ResNet18 was not binarized and the residual block used was the double shortcut with ReLU, which is described in Section 3.1.1. Adam was used as optimiser for both the knowledge distillation training and cross-entropy training. Each sub-network is trained for 5 epochs with the initial learning rate of 0.01. The learning rate is decreased by a factor of 10 at epoch 2, 4 and 5. During cross-entropy training the model is trained for 25 epochs with an initial learning rate of 0.001 which decreases by a factor of 10 at epoch 15, 20 and 23.

To compare the accuracy with and without knowledge distillation, a network without knowledge distillation pre-training was trained using cross entropy in the same way as the cross-entropy training after *method I*.

4

Results

In this section, the results of the tests described in section 3 are presented. Two datasets were used to test how binary networks in combination with knowledge distillation performs - CIFAR10 and ImageNet. CIFAR10 is a relatively small dataset and extensive tests were made to evaluate which knowledge method, network architecture and binary network type worked best. The results of the CIFAR10 dataset is presented in Section 4.1. To verify the results and to compare with others work, the method suggested in this report is evaluated on the ImageNet dataset, which is much larger than CIFAR10. The results of the ImageNet test can be seen in Section 4.2.

4.1 Results using CIFAR10 dataset

The testing with CIFAR10 can be divided into three parts. First different architectures of the residual block were tested, see Section 4.1.1. Secondly, different knowledge distillation methods were tested, see Section 4.1.2. Lastly, different types of binary network types (XNOR, XNOR++, etc.) were tested, see Section 4.1.3. Ideally, all possible combinations of residual block architecture, knowledge distillation method and network type should have been tested. However, in this report, only the most promising results from the previous test were used in the subsequent tests.

4.1.1 Architecture of residual block

To evaluate how well the different architectures of a ResNet block performs, they were tested on *method III* followed by a regular cross-entropy training. Four different blocks were tested - a naive block where ReLU is substituted by a Sign activation, single ReLU block which is the same as the naive but with a ReLU activation in the shortcut, double ReLU which contains two shortcuts in every block and an abs-block where the ReLU activation is replaced by absolute value function multiplied by a learnable scaling factor. The ResNet blocks tested are described in Section 3.1.1. The results are presented in Figure 4.1, where training of *method III* is shown in the leftmost figure and the cross-entropy training is shown in the middle and rightmost figure. See Figure 3.5 for a schematic figure showing which settings are varied and which are fixed.

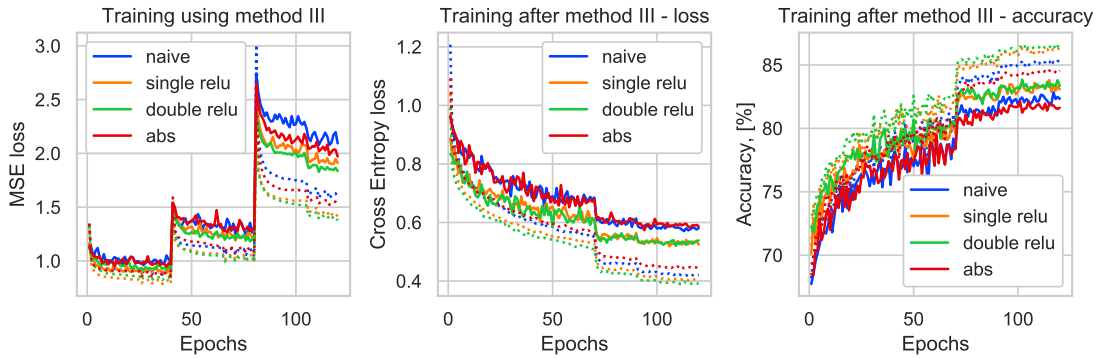

Figure 4.1: *Method III with different architectures of the residual block. The blue represents a naive block where the ReLU activation is replaced by a Sign activation. The yellow represents a block identical to an original full precision ResNet block, but with a Sign activation before every convolutional layer. The green represents a block with two shortcuts in every ResNet block. The red represents a block where absolute value and a scaling factor is used instead of ReLU activation in the shortcut. The different residual architectures tested is presented in Section 3.1.1. The solid lines represent validation loss/accuracy while the dotted lines represent train loss/accuracy.*

When observing figure 4.1 one can note a few interesting things. Looking at the training using *method III*, the lowest loss is achieved by the ReLU double shortcut (green) followed by ReLU single shortcut (orange). Next comes the abs-block and highest loss has the naive architecture were no ReLU is used in the shortcut. It is not surprising that the naive block performs the worst since the student network and the teacher network does not have the same distribution, see motivation in Section 3.1.1. It is also expected that the double shortcut is better at mimicking the teacher's response since less information is lost in the binarization step of the second convolutional layer in every residual block. To avoid losing information in the ReLU step but still maintaining similar distribution, the abs-block was suggested, where the ReLU activation was replaced by absolute value and a scaling factor which was a trainable parameter. The abs-block resulted in a lower loss compared to the naive block, but worse than the ReLU blocks even though less information got lost.

The results of the cross-entropy training differ from the *method III* training. In this "regular" training, the abs-block performs worse than the naive block. One can also observe that the single and double connections of a ReLU-block gave a very similar final accuracy.

The ReLU double shortcut architecture performed best in both the knowledge distillation training and the cross-entropy training. The following tests all use double ReLU shortcut.

4.1.2 Knowledge Distillation technique

In this section the results of the different knowledge distillation methods *I, II* and *III* described in Section 2.2.1 is presented. First, the results of one run for *method I, II, III* and their corresponding cross entropy training are presented. In Section 4.1.2.4, the accuracy achieved by *method I, II, III* is compared for various values

of the scaling factor β in *method I* and *II*. Double ReLU shortcut was used since it performed best at mimicking the intermediate feature response from the teacher network, while also achieving good results of the cross-entropy training. See Figure 3.6 for a schematic figure showing which settings are varied and which are fixed.

4.1.2.1 Training with *method I*

In the leftmost figure in Figure 4.3, the loss of *method I* training is presented. The scaling factor balancing the mean square error loss in intermediate layers and the cross-entropy loss of the target was set to $\beta = 0.4$, see Equation (2.13). The training using *method I* was followed by regular training with cross-entropy loss and these results are shown in the middle and right figures. As a reference, a training without initialising the network by training with a knowledge distillation method is included.

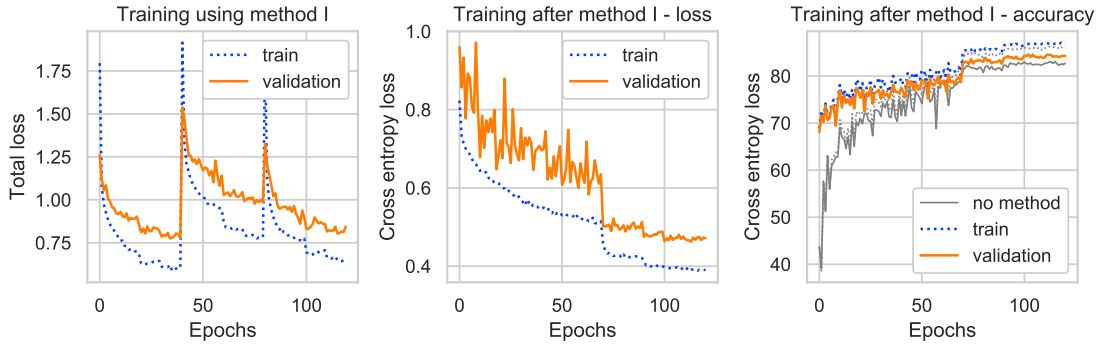

Figure 4.2: *The leftmost figure shows the training loss for method I. The middle figure shows the validation and train loss of regular training but where the network is initialised by the network trained by method I. The far right figure shows the corresponding accuracy of the regular training. The accuracy of regular training without initialisation using method I is included in the rightmost figure as a grey line. In all figures, the orange line represents the validation loss/accuracy and the blue dotted line represents the train loss/accuracy*

The final accuracy - right figure in Figure 4.2 shows that training the network with *method I* before regular training improves the final accuracy. After the first epoch, the accuracy is $\approx 70\,\%$ when using *method I* as initialisation, compared to $\approx 40\,\%$ without.

4.1.2.2 Training with *method II*

In the leftmost figure in Figure 4.3, the loss of *method II* training is shown. The scaling factor balancing the mean square error loss in intermediate layers and the loss of the final layer was set to $\beta = 0.4$, see Equation (2.15). The other hyper-parameters for the KD loss was set to $\alpha = 0.95$ and $T = 6$, see Equation (2.16) and (2.11). The training using *method II* was followed by regular training with cross-entropy loss and these results are shown in the middle and right figure. As a reference, a training without initialising the network by training with a knowledge distillation method is included.

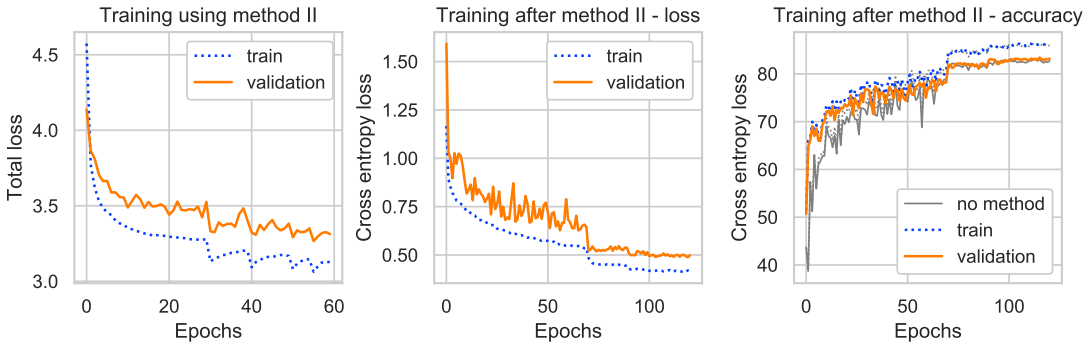

Figure 4.3: *The leftmost figure shows the training loss for method II. The middle figure shows the validation and train loss of regular training but where the network is initialised by the network trained by method II. The right figure shows the corresponding accuracy of the regular training. The accuracy of regular training without initialisation using method II is included in the right figure as a grey line. In all figures, the orange line represents the validation loss/accuracy and the blue dotted line represents the train loss/accuracy*

In Figure 4.3, the final accuracy with and without *method II* training as initialisation is presented in the rightmost figure. One can see that the final accuracy is not improved by using *method II*. After the first epoch, the accuracy is $\approx 50\,\%$ when using *method II* as initialisation, compared to $\approx 40\,\%$ without.

4.1.2.3 Training with *method III*

In the leftmost figure in Figure 4.4, the loss of *method III* training is presented. The training using *method III* is followed by regular training with cross-entropy loss and these results are shown in the middle and right figures. As a reference, a training without initialising the network by training with a knowledge distillation method is included.

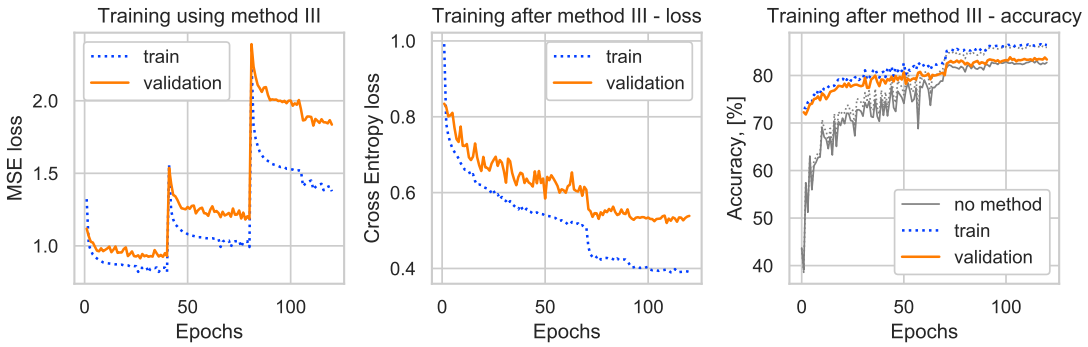

Figure 4.4: *The leftmost figure shows the training loss for method III. The middle figure shows the validation and train loss of regular training but where the network is initialised by the network trained by method III. The right figure shows the corresponding accuracy of the regular training. The accuracy of regular training without initialisation using method III is included in the right figure as a grey line. In all figures, the orange line represents the validation loss/accuracy and the blue dotted line represents the train loss/accuracy*

In Figure 4.4, one can see that the final accuracy is improved slightly when using *method III* as initialisation compared to without. After the first epoch, the accuracy is already above 70 % when using *method III* as initialisation, compared to ≈ 40 % without.

4.1.2.4 Comparison of methods

Figure 4.5 shows the final accuracy for a network that has been trained using cross-entropy loss but where the network has been initialised by training using *method I, II* or *III*. *Methods I* and *II* both contain the hyper-parameter β which balances the loss from intermediate layers and the loss from the final layer. This scaling parameter β was varied. For *method II* $\alpha = 0.95$ and $T = 6$, see Equation (2.16) and (2.11). *Method III* does not have any hyper-parameter, just like the regular training without "smart" initialisation and is therefore represented as a constant line in Figure 4.5.

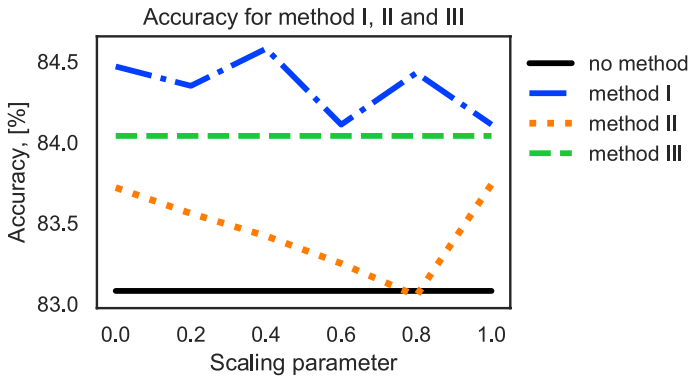

Figure 4.5: *Final accuracy for networks initialised by method I, 11, III or no method. For method I and II different values of scaling factor β is tested (x-axis). method III and no method does not contain any scaling parameter and is represented as a straight line.*

As seen in Figure 4.5, it is apparent that *method I* performs the best - increasing the accuracy by 1.5 %. It might also be possible to observe a slight decrease in accuracy for *method I* when the scaling parameter increases. Important to note is that when the scaling factor $\beta = 0$ there is no contribution from the intermediate layers loss to the total loss.

Method II and *method III* also improves the final accuracy. For *method II*, the accuracy's dependence of the scaling factor is difficult to interpret since the accuracy decreases for larger values of the scaling parameter until it is set to $\beta = 1$. Since *method I* consistently performed best, it is used in the following tests.

4.1.3 Binary network type - binary, XNOR and XNOR++

From the previous tests, the most promising block of architecture was found to be the double shortcut ReLU and the most promising knowledge distillation method was *method I*. Using these, four different binary network types were tested - binary (without scaling factors), XNOR, XNOR++ with $\Gamma \in \mathbb{R}^{o \times h_{out} \times w_{out}}$ and XNOR++ where $\Gamma = \alpha \otimes \beta \otimes \gamma$. The results are shown in Figure 4.6. Scaling factor β was set to 0.4 since it yielded the best accuracy (Figure 4.5). See Figure 3.7 for a schematic figure showing which settings are varied and which are fixed.

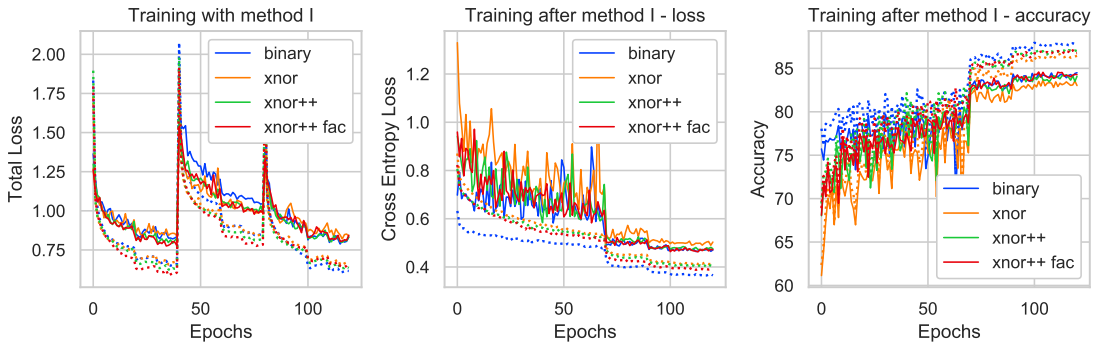

Figure 4.6: *Training of different binary network types using method I as initialisation. Training of method I is shown in the left figure followed by regular training in the middle (loss) and right figure (accuracy). Binary network, where no scaling factor were used is represented as blue. XNOR is represented as orange. XNOR++ where* $\Gamma = \alpha \otimes \beta \otimes \gamma$*, i.e.* Γ *is factorised, is represented by red. XNOR++ where* Γ *has the same dimension as output from the convolutional layer is represented by green. The solid lines represent validation loss/accuracy while the dotted lines represent train loss/accuracy.*

When observing the regular training in Figure 4.6 one can see that, surprisingly, the binary network without any scaling factors achieves the highest accuracy for the train dataset. For the validation set, there is no significant difference between the binary network and the factorised version of XNOR++. The least successful network type was XNOR which has the most unstable training as well as yields the lowest final accuracy.

4.2 Results using ImageNet dataset

Several tests were made on CIFAR10 dataset, but in order to be able to compare with others work ImageNet is used. The method that performed best in the CIFAR10 test were *method I* with double ReLU shortcut and factorised XNOR++ scaling factors. Training of *method I* on ImageNet with scaling factor $\beta = 0.4$ is presented i Figure 4.7.

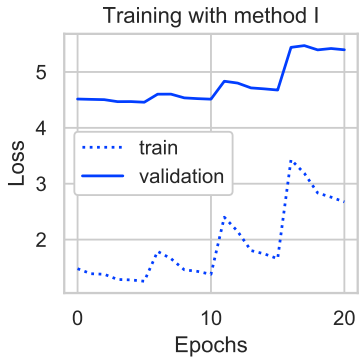

Figure 4.7: *Training using method I on ImageNet. The train loss is represented as the dotted line and the validation loss as the solid line.*

There are four large blocks in ResNet18, and the successive binarization of each block is clearly visible in Figure 4.7. As expected, the cross-entropy loss increases for every block that is binarized.

After the network was trained using *method I* it was trained using cross-entropy loss and the results can be seen in Figure 4.8. A network which is not initialised by training of *method I* is also shown.

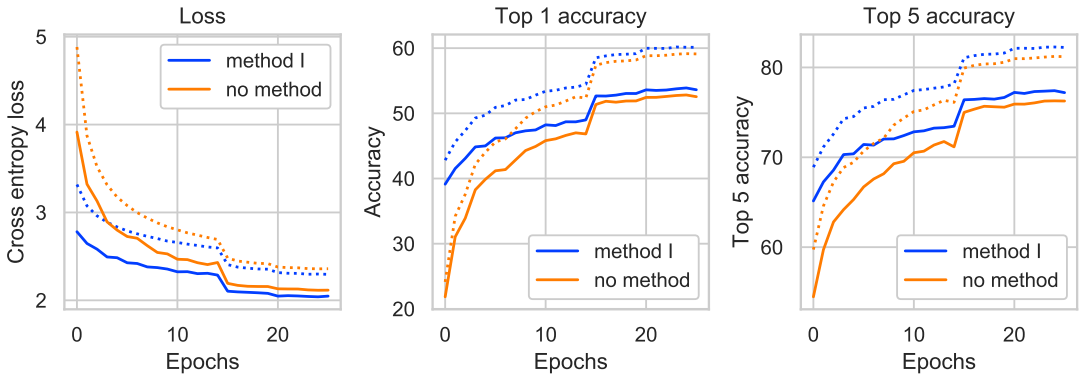

Figure 4.8: *Training using cross-entropy loss where one network has been pre-trained according to method I and one has not. The train loss/accuracy is represented as dotted lines while the validation loss/accuracy is represented by solid lines.*

The train loss in Figure 4.8 is higher than the validation loss. This might seem

unreasonable, but it can be explained by the data augmentation performed on the train data. In the accuracy calculation, non-augmented train data is used and the train accuracy is higher than validation accuracy which is expected. Observing Figure 4.8, one can see that the accuracy improves when the network is pre-trained with *method I*. The final accuracy for both networks is shown in Table 4.1 as well as the accuracy of other binary networks.

	Top 1 accuracy	**Top 5 accuracy**
Method I	53.9 %	77.4 %
No method	52.8 %	76.3 %
Binary	42.2 %	69.2 %
XNOR	51.2 %	73.2 %
XNOR++	57.1 %	79.9 %
Bi-Real	56.4 %	79.5 %
Full precision	69.3 %	89.2 %

Table 4.1: *Top 1 and top 5 accuracies for ResNet18. The first two rows are the accuracies achieved in this paper while the following four are results form others work.*

Comparing the results in Table 4.1, one can see that the results in this project are better than the XNOR-net - both with and without *method I* training. However, Bi-real and XNOR++ report better results.

4.3 Memory Savings

When binarizing ResNet18, the memory size of the model dropped from 46.8 MB to 13.3 MB ($\approx \times 3.5$ less memory is used). The weights in the convolutional layers take up 44.7 MB of the total memory, i.e. the rest of the model, e.g. the weights in the batch norm layers and the fully connected layer only constitutes 2.1 MB. The weights in the first convolutional layer, that is not binarized is only 38 KB. The added parameters α, β, γ in XNOR++ networks use only 33 KB. Almost all of the memory is consumed by the convolutional layers which are binarized. Theoretically, the memory saved by using a boolean compared to a floating-point is 32. However, the framework used (numpy in Python) saves boolean as a byte and not a bit (maximal potential memory saving is 4). If another framework was used, which represents a boolean as a bit, the memory savings will be substantially larger.

5

Discussion

In this section the results of the CIFAR10 test are discussed in Section 5.1. The discussion of the ImageNet test is found in Section 5.2. Method choices are analysed in Section 5.3 and in section 5.4 potential future work is suggested.

5.1 Analysis of CIFAR10 results

Three main things were tested using the CIFAR10 dataset - architecture of residual block, knowledge distillation techniques and if any binary network type was better suited for knowledge distillation. The results are discussed in Section 5.1.1, 5.1.2 and 5.1.3 respectively. A conclusion of the discussion is presented in the beginning of each section, then followed by the actual discussion.

5.1.1 Which architecture of residual connection is best?

The single and double ReLU architecture performed the best since the distributions of teacher and student network match. Even though the representational capability increased in the double shortcut network, it does not help the network to generalise when trained with CIFAR10 dataset.

In section 4.1.1, different architectures were trained for ResNet20 using CIFAR10. When trained using *method III*, the architecture which performed best was the double ReLU shortcut followed by ReLU single shortcut. This is not surprising since the student and teacher network has the same output distribution in the intermediate layers. It is also expected that the network with double shortcuts performed better than the single shortcut network when training towards minimising the mean square error loss. Since less information is lost in the second convolutional layer in each block the representational capability (the functions which the block/model can learn) is higher for the double shortcut block. When training with cross-entropy loss, there is no difference between the single and double shortcut. Apparently, the network cannot make use of its increased representational capability.

The student network is initialised as the teacher network. This likely makes the student network converge faster when learning to mimic the teachers feature responses. However, when the network is changed, either by removing ReLU from shortcut or by adding an extra shortcut, the information obtained from initialising from the

teacher network is largely lost. If this had a large effect, the single ReLU shortcut would have a lower loss in the first epochs compared to the double shortcut ReLU. However, this is not the case and if the effect does exist it is too small to be visible.

Furthermore, it is obvious that the block, where the ReLU activation was substituted with Abs as activation followed by a learnable scaling factor, was unsuccessful. The scaling factor was introduced to make the distributions of teacher and student network similar. There is a unique scaling factor for each layer in the network. It is possible that the individual difference between each element is too large so that the scaling factor cannot learn to mimic the distribution but instead tries to mimic the individual parameters. This does not improve the networks ability to generalise.

5.1.2 Does knowledge distillation work for binary networks?

The results from CIFAR10 tests indicate that knowledge distillation improves the network's ability to learn. Method I performed the best which in turn could have several different explanations. Three of them are:

- *When using knowledge distillation, the initial network is closer to a minimum which is equivalent to longer training time.*
- *The student network can utilise the information in the teacher network's intermediate layers.*
- *The progressive learning makes the search space gradually larger, which improves learning.*

From analysing the results, the progressive learning is likely to be the main reason why method I performs well.

The main goal of the project is to answer the question - does knowledge distillation work for binary networks? In section 4.1.2.4 the results showed that a network that has been trained according to *method I* before its regular training improved the final accuracy by 1.5 %. But why is the accuracy improved and are the results reliable?

When training a network there is a great deal of stochasticity involved. In this case, it mainly originates from the mini-batch size being much smaller than the dataset. This is not a bad thing, rather the opposite, since it helps escape local minima. However, the stochastic nature of the problem makes the results more difficult to interpret. For example, it is not safe to say that 0.4 is the best value for the scaling factor β in *method I*. It is also possible that the trend break for *method II*, where the accuracy is decreasing for large values of scaling factor β until equal to 1, could be explained by "lucky" training. To evaluate which hyper-parameter is the best, multiple runs have to be made to minimise the "lucky effect".

With this argument, all results in this report could be questioned, but there is a limit of how much the stochasticity affects the results. *Method I* constantly performs 1 % better than if no method is used, which indicate that it would improve accuracy independent of the stochastic element in the optimisation. But this leads to the

follow-up question, why does it work? The following sections will explore possible explanations.

5.1.2.1 Initially closer to minimum which is equivalent with longer training time

Compared to regular full precision networks, binary networks is not as sensitive to overfitting. Courbariaux et al. argued that binarization could be seen as a reguliser similar to dropout [35][8]. But instead of randomly setting half of the activations to zero when computing gradients of the parameters, the input and weights are binarized. Since overfitting is not a problem, the network benefits from long training times. In this thesis, knowledge distillation is used as an initialisation process. It is relevant to note that the total training time is doubled when pre-trained with *method I* or *III* compared to no pre-training. As seen in Figure 4.4, 4.3 and 4.2, the initial accuracy of the regular training is much higher if the network has been initialised with a knowledge distillation method. The training is starting closer to a minimum and it can be seen as equivalent to longer training time. A visualisation of this can be seen in Figure 5.1.

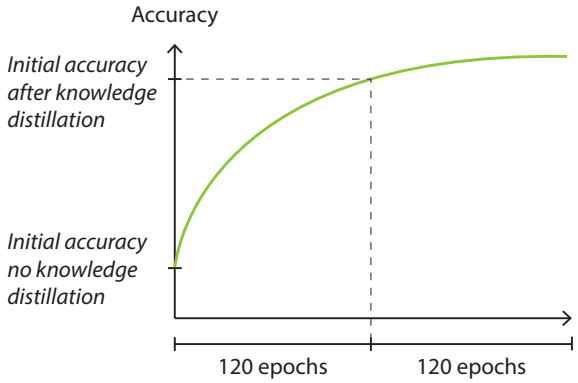

Figure 5.1: *Visualisation of how better initial accuracy means a head start and in turn is equivalent to longer training time.*

However, it is interesting to note that the initial accuracy when regular training has started, is higher for *method III* compared to *method I*, while the final accuracy is better for *method I*. This implies that it is not solely a question of longer training time but about finding a more optimal minimum. Longer training time does not explain why *method I* performs best.

5.1.2.2 Learning from intermediate layers

One hypothesis why *method I* works is that the network can make use of the information in the teacher's intermediate layers. However, the results presented in Figure 4.5 points towards the fact that low values of scaling factor β improves the accuracy the most, i.e. when no or little information in intermediate layers is used during training. As a counter-argument, *method III* boosts the accuracy even though it only

trains to mimic the feature responses of the teacher network. This suggests that information in the teacher's intermediate layers is helpful during training. However, this could be explained by a head start in training (equivalent to longer training). To conclude, it is not reasonable to believe that information from intermediate layers is the main reason why *method I* works.

5.1.2.3 Progressive learning

Another hypothesis is that the progressive training in *method I* is what makes the accuracy improve. When $\beta = 0$ for training c) the loss only consists of the cross-entropy loss of the final layers. The only difference from regular training is that the network is binarized and trained successively, starting with the first block conv2_x (see Figure 3.2 or 3.1). This procedure makes the search space gradually larger and it might minimise the risk of the network getting stuck in a sub-optimal minimum. This is the most probable reason.

5.1.3 Which network type works best?

The two network types which achieved the highest validation accuracy was the binary network without scaling factors and the factorised XNOR++ network. Both performed equally well. A reason why the scaling factors did not improve the performance could be that the batch norm layers absorb the effects of scaling factors or that the binary network converges faster.

In section 4.1.3, different binary network types were tested together with *method I* training. Unexpectedly, the binary network, i.e. the one without any scaling factors, had the lowest training loss (and highest train accuracy). This is unexpected because XNOR and XNOR++ nets were introduced to boost the accuracy of binary networks. In XNOR-nets some information about the weights and input are retained. In XNOR++ additional parameters are trained. For a regular ResNet18 both XNOR and XNOR++ improved the accuracy significantly compared to binary networks without scaling factors [10][11]. Therefore it seems likely that the accuracy would be better for networks using scaling factors.

However, there is one large difference between the XNOR and XNOR++ papers and this thesis - in this work another network structure is used. In the works of Z. Liu et al. [12] and J. Bethge et al. [13], networks with double shortcuts are used and they achieved good results without the use of scaling factors. Bethge et al. even got better results without the use of XNOR scaling factors than with when using CIFAR10 dataset. They argued that the batch norm layers following the convolutional layers might absorb the effect of scaling factors. The results in this project mirror the results of Bethge et al. which also reports lower accuracy when using XNOR scaling factors than without.

In this thesis, the XNOR++ scaling factors were tested and it gave equivalent validation accuracy as using no scaling factors. However, when observing the Figure 4.6 one can see that the train accuracy for the binary network is well above the accuracy

for XNOR++ nets. Looking at the loss, the binary network learns much faster than the other network types. This is expected since there are fewer parameters to learn compared to the XNOR++ networks and the input is not as varying as in the XNOR networks. But this also means that there is a chance that the XNOR++ networks did not train long enough, but that the potential is higher if trained long enough. This argument is strengthened by the fact that the non-factorised XNOR++ performs worse than the factorised version (more parameters to train).

5.2 Analysis of ImageNet results

An exhaustive testing was performed using the CIFAR10 dataset. However, in order to compare the results with those of others, *method I* was tested using ImageNet dataset. In section 5.2.1 the results of this thesis is compared with others work. This is followed by a discussion about if the results achieved on CIFAR10 dataset is also valid for the ImageNet problem.

5.2.1 Comparison with others work

In Table 4.1, the results of ResNet18 and ImageNet are presented. The network type used was XNOR++. It would therefor be expected that the accuracy in this work would be close to that of XNOR++ nets. Clearly, this is not the case and there are several reasons why that might be.

One cause might be that the training was too short. The XNOR++ network in the works of Bulat et al. trained for 80 epochs, while the networks in this thesis only trained for 25 epochs. Too short training might be part of the explanation, but it is not certain that longer training would increase the accuracy by more than 3 % (which is the difference between the achieved accuracy in this work and the reported accuracy by XNOR++). Another difference is the mini-batch size - 64 (this project) compared to 256 (XNOR++). A smaller mini-batch size increases the stochasticity and even though it helps to escape sub-optimal local minima, it might also make it escape an optimal minimum.

The largest difference between the XNOR++ paper and this work is that different network architectures are used. In Bulats et al.'s paper, the original architecture of the residual block is used (i.e. the Naive in figure 3.4). In this work, different architectures were tested and the best one to mimic the feature response of teacher network was found to be the architectures containing ReLU, since the student and teacher then have the same distribution. However, when the ReLU activation is applied, some information is lost. This might be the reason why the network in this thesis does not achieve higher accuracy.

If we for a moment ignore the comparison with the works of others, we can observe that the network pre-trained with *method I* performs better than the one without any "smart" initialisation. This indicates that knowledge distillation works for binary networks. In Section 5.1.2 various reasons why knowledge distillation might improve the accuracy was discussed. The discussion was based on the results of CIFAR10

dataset, and one needs to ask if the same results and reasoning holds when using the ImageNet dataset.

5.2.2 Does the analysis of CIFAR10 hold for ImageNet?

A thorough analysis was performed using the CIFAR10 dataset. With the ImageNet dataset only one test was made and it is relevant to ask if the results found by testing with CIFAR10 is applicable to the ImageNet problem. The CIFAR10 and ImageNet problem are quite different. ImageNet is a much more complex problem but has, on the other hand, more training images. In the work of Courbariaux et al., where no scaling factors were used, they achieved great results on the MNIST [36] and CIFAR10 datasets. But when tested with ImageNet and ResNet they only achieved an accuracy of 42.2 % [11] which is much lower than its full precision accuracy as well as the results reported in this, and other's work (see Table 4.1). This indicates that the results are probably not directly transferable and that the methods to increase representational capability (scaling factor and extra shortcuts) might work for ImageNet even though the CIFAR10 results did not indicate any improvements.

To discuss the difference between CIFAR10 and ImageNet, two concepts of limiting factors are used - representational capability, which is a measure of how many functions the network can learn, and the model's ability to generalise. In general, the limit of representational capability is much higher than the limit of the model's ability to generalise. The model's ability to generalise is dependent on a several factors, for example, the network architecture, regularisation techniques (dropout, L1 and L2 regularisation etc.) or the dataset itself. The more variation there is in a dataset, either naturally or by data augmentation, the easier it is for the network to generalise. The ImageNet dataset is a more complex problem and has 1.2 million data samples compared to the 60,000 images in CIFAR10. Additionally, the ImageNet input is more augmented. While the representational capability is similar for the networks used when training on CIFAR10 and ImageNet, the model's ability to generalise is higher for ImageNet. This is visualised in Figure 5.2.

Figure 5.2: *Representation of limitations of a model. In a regular full precision network, the representational capability is higher than the model's ability to generalise. The limiting factor is therefore, the ability to generalise and not the representational capability. The larger and more varied train dataset in ImageNet leads to a higher ability to generalise in comparison to CIFAR10.*

When the networks are binarized, the representational capability as well as the models ability to generalise drops. One hypothesis is that the representational capability drops below the model's ability to generalise for the ImageNet problem, but

not for the CIFAR10 problem. A visualisation of this can be seen in Figure 5.3. Adding learnable scaling parameters or an extra shortcut increases the representational capability, but not necessarily the ability to generalise. If the limiting factor is generalisation, scaling factors or extra shortcuts might not improve the learning. This might be the reason why these methods did not improve the accuracy in the CIFAR10 tests but have proven successful for ImageNet [11][13][12].

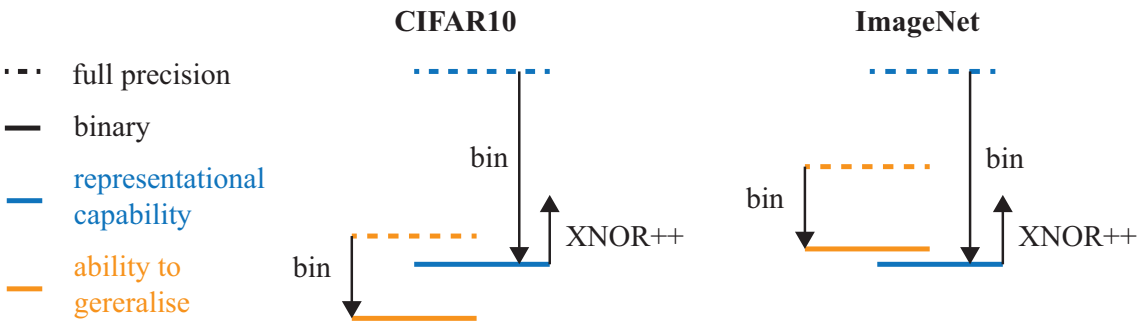

Figure 5.3: *Visualisation of the potential limiting factor for CIFAR10 and ImageNet when the network is binarized. When the network is binarized both the representational capability and the ability to generalise decreases. The networks ability to generalise is higher for ImageNet compared to CIFAR10 and it is possible that the representational capability becomes the limiting factor for ImageNet but not for CIFAR10. It is reasonable to believe that the drop in representational capability is larger than the drop in ability to generalise since binarization acts as a regulizer [8]. The representational capability can be improved by using XNOR++ scaling factors or add an extra shortcut.*

The results from previous studies have shown that binary networks using a smaller dataset such as MNIST and CIFAR10 learns better than large datasets such as ImageNet. It has also been reported that scaling factors and extra shortcuts improve the accuracy for ImageNet, but in this thesis, the CIFAR10 test did not show any indication of improved learning. This difference between CIFAR10 and ImageNet could potentially be explained by different limiting factors - CIFAR10 is limited by the ability to generalise, while ImageNet is limited by the representational capability. If this is the case, it is not possible to draw any certain conclusions of the CIFAR10 test for the ImageNet problem.

5.3 Choice of method

In this section the method choices in this thesis are discussed, more specifically the loss function used, the binarization order and that not all combinations of knowledge distillation technique, architecture and network type were tested.

5.3.1 Other loss function of intermediate layers might improve learning

In this thesis only a few knowledge distillation methods were tested and only ones using mean square loss from intermediate layers. When a network is binarized

large amounts of information is lost, and the representational capability shrinks substantially. For CIFAR10, the mean square error term in the loss in *method I* and *II* did not seem to add any valuable information to the loss. It is likely that the drop in representational capability makes the network unable to mimic the teachers feature responses sufficiently well. Another alternative to mean square error is to let the student network learn to mimic the absolute values of the teacher network in the intermediate layer as done by Zagoruyko et al. [15]. This would allow the student to learn more freely while still getting help from the teacher to focus on the important features.

5.3.2 Using knowledge distillation method that does not require same distributions

In all methods using information from the teacher's intermediate layers, the student intermediate layers need to have the same distribution to make sense. But in order for the networks to have the same distributions, the student network needs to use ReLU in its shortcut. As discussed earlier, information is lost in ReLU, and these methods might therefore not be optimal for binary networks. It might be better to instead use the original knowledge distillation training proposed by [14] which only uses the last output layer from the teacher network.

5.3.3 Order of binarization

To my knowledge, the only work using knowledge distillation in combination with binary networks is the one by Xu et al. [27]. Their work differs from this thesis in many ways. First of all, their network only binarized the weights but not the input. They also used a different dataset and a deeper network which solves an object recognition problem (compared to image classification in this work). The differences make the results difficult to compare. There were also some similarities - they used both knowledge distillation and a progressive approach when binarizing the network. In contrast to this work, they binarized the network in the opposite order (last layers first). Their reasoning being that binarizing the last layers had less effect than binarizing the first layers which were reported by Zhuang et al. [30]. A few things to note is that Xu et al. trained the whole network and not just the binary parts. They also had a much easier problem to solve since they did not binarize the input to convolutional layers as done in this work. They showed great results but the reverse order is not applicable to this problem.

Let us use a metaphor and let the training of a network be represented as making a sculpture. If binarizing and training of the first layer affects the network the most it can be represented as a hammer to carve out the initial shape. While the last layers do not change the network too much and can be represented as sand paper making the final polishing. The reverse order would imply that one starts with the sandpaper to then take out the hammer and destroy all the work done with sand paper.

Wang et al. (*method I*) tested their progressive training both forward and backwards

and got better results for the forward variant [20] which strengthens this choice of not binarizing in reverse order. Even though Wang et al. did not study binarization, they used a smaller student network and it is reasonable that their results will hold for binary networks as well.

5.3.4 Not all combinations were tested

In this project, the most promising results from the previous test were chosen as the basis for the next test. For example, was the ReLU double shortcut used when testing knowledge distillation technique and network type. To be certain that one combination of the three things tested is best all combination needs to be tested. Additionally, each test should be made multiple times to minimise the risk of misleading results from stochastic effects.

5.4 Future studies

To further investigate binarization and knowledge distillation a few possible future studies are suggested. However, the first step would be to verify the results.

5.4.1 Verify results

The results presented in Section 4 indicated that knowledge distillation techniques could improve the accuracy of a binary network. To confirm the results in this thesis, the following test could be made:

- Longer training for both CIFAR10 and ImageNet, both in knowledge distillation training and in the cross-entropy training.
- Test all combinations of knowledge distillation methods, network type and the architecture of residual shortcut.
- Make multiple runs of the same test to minimise stochastic effects.
- As discussed in Section 5.2.2 it is not possible to draw any conclusions regarding ImageNet by observing the CIFAR10 results. To evaluate which knowledge distillation method, architecture of residual shortcut and network type work best for ImageNet, all tests have to be repeated using ImageNet.

5.4.2 Adapt teacher architecture to suit optimal student architecture instead of the other way around

In this thesis the architectures of the binary ResNet block were constrained to output identical distributions as the teacher network. This limitation can be avoided by using the original knowledge distillation by Hinton et al. [14] which only uses the last output layer from the teacher network. Another approach would be to train the original full precision network using the same ResNet block as in the binary student network. So, instead of modifying the student network to match the teacher

network, the teacher network can be modified to match the student network. This might be slightly problematic since the modified teacher might not perform as well as the original teacher architecture. However, it probably still performs better than the student network and is therefore relevant to test. If the limitation of block-architecture is removed, it would be possible to evaluate how Bi-real nets [12] or DenseNet [13] can be improved by the usage of knowledge distillation.

5.4.3 Use other loss function for intermediate layers

In this project, a few versions of knowledge distillation techniques have been tested all using mean square error loss from intermediate layers. It would be interesting to investigate how other loss functions would affect accuracy. For example, the absolute value of the feature responses between student and teacher network could be compared [15] or the maximum mean discrepancy could be compared [16]. Both have shown improved results compared to regular knowledge distillation training [14].

5.4.4 Use other low-precision network types

Only completely binary networks, i.e. both input and weights are set to -1 or +1, were studied in this thesis. There are other networks proposed, for example, TBN which uses ternary inputs i.e. inputs $\in \{-1, 0, 1\}$, HORQ networks which use higher-order quantization [5] or DoReFa networks which use low bitwidth of convolutions and gradients (but $\neq 1$) [4]. Knowledge distillation methods have proven successful when a smaller, but full precision, student net have trained. It is likely to believe that knowledge distillation performs better the higher the representational capability of a network is. It would, therefore, be very interesting to see if these networks would benefit more from knowledge distillation compared to the binary networks studied in this project.

6

Conclusion

A binary network requires less memory and much less computational power, which makes it suitable for less powerful devices. The aim of this thesis was to investigate how the accuracy of a binary network can be improved by using knowledge distillation techniques, as well as answering the questions: Does some knowledge distillation technique work better for binary networks than others? Can the architecture of a network be modified to improve the effect of knowledge distillation training? Are some network types better suited for knowledge distillation? The main contributions of this thesis can be summarised as:

- A novel approach for training binary network was suggested and tested which used knowledge distillation techniques to pre-train a network.
- Different architectures of a residual block were suggested and tested on CIFAR10 dataset. An architecture with double shortcuts and ReLU activation performed best when using knowledge distillation.
- The different network types XNOR, XNOR++ and regular binary network were tested in combination with knowledge distillation. The binary and XNOR++ networks worked best.
- Three different knowledge distillation methods were tested on CIFAR10 dataset. The best performing model was trained using a progressive approach where the loss consisted of a mean square error loss from an intermediate layer in student and teacher network as well as a cross-entropy loss of final layer. It gave an improvement of 1.5 % on CIFAR10 and ResNet20.
- The same method was tested on ResNet18 and ImageNet as dataset. Knowledge distillation improved the accuracy of 1.1 % compared to regular cross entropy training.

Compared to other reported results, the accuracy in this work does not achieve as high value. This was mainly explained by two reasons - too short training time and loss of information in ReLU activation. To avoid the problem of information getting lost in the shortcut by ReLU, it was suggested that the teacher network could be trained such that the distributions in student and teacher network are the same, instead of the student network being constructed to suit the teacher network.

The results on CIFAR10 and ImageNet indicate that knowledge distillation can be

used as a method to boost the accuracy of a binary network. Further testing has do be done to verify that this statement is valid, especially longer training for the ImageNet tests. If, however, the statement holds after further testing, this could be used as a tool for improving the accuracy for binary networks and in turn, make accurate image analysis more available on less powerful devices.

Bibliography

[1] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size", *CoRR*, vol. abs/1602.07360, 2016. arXiv: 1602.07360. [Online]. Available: http://arxiv.org/abs/1602.07360.

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", *CoRR*, vol. abs/1704.04861, 2017. arXiv: 1704.04861. [Online]. Available: http://arxiv.org/abs/1704.04861.

[3] S. Han, J. Pool, J. Tran, and W. J. Dally, *Learning both weights and connections for efficient neural networks*, 2015. arXiv: 1506.02626 [cs.NE].

[4] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, *Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients*, 2016. arXiv: 1606.06160 [cs.NE].

[5] Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao, *Performance guaranteed network acceleration via high-order residual quantization*, 2017. arXiv: 1708.08687 [cs.CV].

[6] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, 2015. arXiv: 1510.00149 [cs.CV].

[7] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations", *CoRR*, vol. abs/1511.00363, 2015. arXiv: 1511.00363. [Online]. Available: http://arxiv.org/abs/1511.00363.

[8] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1", Feb. 2016.

[9] Y. Hu, J. Zhai, D. Li, Y. Gong, Y. Zhu, W. Liu, L. Su, and J. Jin, "Bitflow: Exploiting vector parallelism for binary neural networks on cpu", May 2018, pp. 244–253. DOI: 10.1109/IPDPS.2018.00034.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks*, 2016. arXiv: 1603.05279 [cs.CV].

[11] A. Bulat and G. Tzimiropoulos, *Xnor-net++: Improved binary neural networks*, 2019. arXiv: 1909.13863 [cs.CV].

[12] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, *Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm*, 2018. arXiv: 1808.00278 [cs.CV].

[13] J. Bethge, M. Bornstein, A. Loy, H. Yang, and C. Meinel, *Training competitive binary neural networks from scratch*, 2018. arXiv: 1812.01965 [cs.LG].

[14] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, 2015. arXiv: 1503.02531 [stat.ML].

[15] S. Zagoruyko and N. Komodakis, *Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer*, 2016. arXiv: 1612.03928 [cs.CV].

[16] Z. Huang and N. Wang, *Like what you like: Knowledge distill via neuron selectivity transfer*, 2017. arXiv: 1707.01219 [cs.CV].

[17] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7130–7138.

[18] M. Gao, Y. Shen, Q. Li, J. Yan, L. Wan, D. Lin, C. C. Loy, and X. Tang, *An embarrassingly simple approach for knowledge distillation*, 2018. arXiv: 1812.01819 [cs.CV].

[19] A. Koratana, D. Kang, P. Bailis, and M. Zaharia, *Lit: Block-wise intermediate representation training for model compression*, 2018. arXiv: 1810.01937 [cs.LG].

[20] H. Wang, H. Zhao, X. Li, and X. Tan, "Progressive blockwise knowledge distillation for neural network acceleration", in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 2769–2775. DOI: 10.24963/ijcai.2018/384. [Online]. Available: https://doi.org/10.24963/ijcai.2018/384.

[21] J. Zhang, Y. Pan, T. Yao, H. Zhao, and T. Mei, *Dabnn: A super fast inference framework for binary neural networks on arm devices*, 2019. arXiv: 1908.05858 [cs.CV].

[22] D. Wan, F. Shen, L. Liu, F. Zhu, J. Qin, L. Shao, and H. T. Shen, "Tbn: Convolutional neural network with ternary inputs and binary weights", in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 322–339, ISBN: 978-3-030-01216-8.

[23] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[24] P. Maji and R. Mullins, "On the reduction of computational complexity of deep convolutional neural networks", *Entropy*, vol. 20, p. 305, Apr. 2018. DOI: 10.3390/e20040305.

[25] X. Xu and M. Pedersoli, "A computing kernel for network binarization on pytorch", *ArXiv*, vol. abs/1911.04477, 2019.

[26] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, *Fitnets: Hints for thin deep nets*, 2014. arXiv: 1412.6550 [cs.LG].

[27] J. Xu, P. Wang, H. Yang, and A. M. López, *Training a binary weight object detector by knowledge transfer for autonomous driving*, 2018. arXiv: 1804.06332 [cs.CV].

[28] J. Redmon and A. Farhadi, *Yolo9000: Better, faster, stronger*, 2016. arXiv: 1612.08242 [cs.CV].

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. arXiv: 1704.04861 [cs.CV].

[30] L. Zhuang, Y. Xu, B. Ni, and H. Xu, *Flexible network binarization with layer-wise priority*, 2017. arXiv: 1709.04344 [cs.CV].

[31] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[33] A. Krizhevsky, "Learning multiple layers of features from tiny images", *University of Toronto*, May 2012.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database", in *CVPR09*, 2009.

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014, ISSN: 1532-4435.

[36] Y. LeCun and C. Cortes, "MNIST handwritten digit database", 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/.