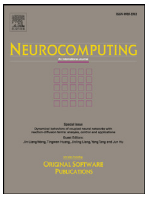
ARTICLE IN PRESS
JID: NEUCOM [m5G;May 17, 2019;15:49]
Neurocomputing xxx (xxxx) xxx
Contents lists available at ScienceDirect
Neurocomputing
journal homepage: www.elsevier.com/locate/neucom



Training binary neural networks with knowledge transfer

Sam Leroux[*], Bert Vankeirsbilck, Tim Verbelen, Pieter Simoens, Bart Dhoedt

Department of Information Technology, IDLab, Ghent University–IMEC, iGent Tower–Technologiepark-Zwijnaarde 15, Ghent B-9052, Belgium

A R T I C L E I N F O

Article history:
Received 30 March 2018
Revised 16 August 2018
Accepted 11 September 2018
Available online xxx

Keywords:
Binary neural networks
Deep learning
Knowledge transfer

A B S T R A C T

Binary Neural Networks (BNNs) use binary values for both weights and activations instead of 32 bit floating point numbers typically used in deep neural networks. This reduces the memory footprint by a factor of 32 and allows a very efficient implementation in hardware. BNNs are trained using regular gradient descent but are harder to optimise, take longer to train and generally require a more careful tuning of hyperparameters such as the learning rate decay schedule than floating point versions. We propose to use Knowledge Transfer techniques to make it easier to train BNNs. Knowledge transfer is a general technique that tries to transfer the knowledge stored in a large network (the teacher) to a smaller (student) network. In our case the teacher is a network trained with floating point weights and activations while the student is a BNN. We apply different Knowledge Transfer techniques to the task of training a BNN. We introduce a novel similarity based Knowledge Transfer algorithm and show that this technique results in a higher test accuracy on different benchmark datasets compared to training the BNN from scratch.

1. Introduction

Deep neural networks are exceptionally powerful but they also require large amounts of resources such as compute power and memory. Training a neural network is the computationally most expensive part but this is usually done offline on high performance systems in a datacenter. The basic computational operation of a neural network is a matrix-matrix multiplication. This operation is highly parallelizable and can be very efficiently performed on Graphical Processing Units (GPUs). GPUs are currently the best option to train neural networks.

Once trained the network needs to be deployed in a real-world environment. This stage (known as inference) requires less resources than training but even a moderate sized network can take billions of floating point operations (FLOPs) just to process one input. In addition the device also needs to store all parameters of the network which quickly adds up to hundreds of megabytes.

Binary neural networks are more efficient because they are constrained to binary weights and activations. This reduces the memory footprint of the weights by a factor of 32 and also allows for a very efficient implementation in hardware since the 32 bit floating point multiplications can now be replaced with bitwise logical operations [1].

Courbariaux et al. first showed that it is possible to train modern large neural networks for image classification with binary weights and activations [1]. This suggests that typical neural networks are overparameterized [2]. While we were able to replicate these results we found that BNNs typically take longer to train and are more sensitive to hyperparameters such as the architecture of the network, the initial learning rate, the learning rate decay schedule, the optimization algorithm and regularization terms.

Instead of training a BNN from scratch we propose to use the knowledge from an already trained floating point model. A floating point *teacher* model can be trained using existing state-of-the-art techniques and then be used to guide the optimisation process of the *student* network. This concept is known as Knowledge Transfer. We describe two common Knowledge Transfer techniques in Section 2. In Section 3 we present our novel similarity based Knowledge Transfer technique and we compare the three techniques applied to training binary neural networks for image classification in Section 4.

2. Related work

Deep neural networks (DNNs) have been successfully applied in various areas such as computer vision [3–5], remote sensing [6], speech recognition [7], robotics [8], metric learning [9,10] and recently even in image generation [11], style transfer [12] and caption generation [13]. For a comprehensive overview of the history of deep learning we refer to [14].

[*] Corresponding author.
E-mail address: sam.leroux@ugent.be (S. Leroux).

https://doi.org/10.1016/j.neucom.2018.09.103
0925-2312/© 2019 Elsevier B.V. All rights reserved.

2.1. Resource constrained deep learning

Various works have reduced the computational cost and/or the memory footprint of DNNs. Two of the first works to recognize the fact that neural networks typically contain redundant parameters were Optimal Brain Damage [15] and Optimal Brain Surgeon [16]. They used second order derivative information to identify the connections that can be safely pruned. More recently Han et al. proposed a three step method [17] where first the network was trained to learn which connections are important. Next the unimportant connections were pruned and finally the remaining weights were fine-tuned to compensate for the lost accuracy. This technique is able to reduce the number of parameters in state of the art networks by an order of magnitude.

Other approaches include transforming the weight matrices into low rank decompositions [18,19] or even a hashing based technique [20] where connection weights are grouped into hash buckets and all connections within the same bucket share the same value.

It is well known that full precision floating point numbers are not needed for weights and activations. 8 bit fixed point integers are usually sufficient [21] and these allow for efficient implementation in hardware. Other works further reduce the precision of the weights to 4 bits (for convolutional layers) or to two bits (for fully connected layers) [22].

2.2. Binary neural networks

In the extreme case the precision of weights and activations can even be reduced to 1 bit. This allows an extremely efficient implementation in hardware. The BinaryConnect paper by Courbariaux et al. [23] was the first to train large modern neural networks for image classification with binary weights. This was later extended in [1] to binary weights and binary activations and in a follow-up paper [24] results on the Imagenet dataset were presented. The name “binary neural network” had been used long before for networks that were capable of learning binary-to-binary mappings [25]. These networks used ternary (−1, 0, +1) or integer weights that make them more efficient in hardware implementations [26] and could be trained with different techniques such as expand-and-truncate learning (ETL) [27] or DNA-like learning [28].

The basic operation in a BNN is the binarization function that transforms the floating point weights and activations (x) to binary values (-1 and +1). This function simply thresholds the value based on the sign:

$$Binarize(x) = sign(x)\begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{1}$$

This binarization function is used to binarize both the weights of the network and the activations. The forward pass is then described by Algorithm 1. Where N is the number of layers in the network. W_k are the floating point weights of the $k-th$ layer and W_k^b are the binarized weights. Similarly, a_k are the activations of the $k-th$ layer. $Batchnorm(x)$ represents the Batch Normalization [29] operation and $BinaryDotProduct(x, w)$ calculates the binary dot product between the (binary) input vector x and the binary weights w.

Training a neural network with binary weights and activations is not straight-forward for two reasons. First, stochastic gradient descent (SGD) relies on making many small updates to weights. Every update on itself is very noisy but the noise is averaged out by accumulating many updates. Restricting the weights to binary values is a much too coarse approximation for SGD since the small updates would be lost in the quantization noise. The solution is to accumulate all updates in floating point weights and to use binarized copies for the dot product. The second problem is that the sign function that is used for binarizing the weights and activations has a zero derivative almost everywhere (hard threshold) which makes it incompatible with backpropagation since the gradient of the loss with respect to the input of the sign function would be zero [23]. The solution is to use a “straight-through estimator” [30] which approximates the outgoing gradient by the incoming gradient.

Algorithm 1 Forward pass through a BNN [23].

Input: Full-precision weights W_k for each layer k. The total number of layers N. A minibatch of data a_0.

```
1:  procedure FORWARD
2:      for k=1 to N do
3:          $W_k^b \leftarrow Binarize(W_k)$
4:          $a_k \leftarrow BinaryDotProduct(a_{k-1}, W_k)$
5:          $a_k \leftarrow BatchNorm(a_k)$
6:          if k < L then
7:              $a_k \leftarrow Binarize(a_k)$
8:          end if
9:      end for
10: end procedure
```

The XNOR-net paper [31] proposed a similar but slightly different approach where the output of the binary layers was multiplied with a floating point scale factor to recover the dynamic range. This yields better results on the Imagenet dataset but makes a hardware implementation more difficult. The first and last layers still used 32 bit floating point numbers in this implementation making it a slightly less resource efficient solution compared to fully binary networks.

It might seem surprising that it is possible to train neural networks with binary weights and activations. There has been some very recent theoretical work that gives a possible explanation. Anderson et al. [32] show that a binary approximation of a high dimensional vector still preserves the direction of the vector very well. This would suggest that the information loss caused by the binarization process is not as severe as it would seem. They also find that the batch normalized weight-activation dot products (the intermediate representations) are approximately preserved under the binarization of the weight vectors and they show that this is a sufficient condition for the binary operations to approximate the underlying floating point operations. Lastly they argue that the computations done by the first layer of neural networks trained for image classification are fundamentally different than the computations being done in the rest of the network. The impact of binarization on this layer is much more severe. This is why they suggest to use a floating point convolution for this very first layer. This layer then projects the floating point input to a high dimensional binary space.

BNNs can be evaluated much more efficiently than floating point networks but this requires custom implementations since most general purpose compute platforms like CPUs or GPUs are not optimized for binary operations. Courbariaux et al. implemented a custom GPU kernel that is able to evaluate BNNs seven times faster than a baseline kernel on GPUs [1]. Other works have designed Field Programmable Gate Array (FPGA) implementations [33] or even completely custom hardware platforms [34] to fully exploit the potential of BNNs.

2.3. Knowledge distillation

An interesting family of techniques tries to export the knowledge stored in a large model or in an ensemble of models (the teacher) to a smaller network (the student) that is more efficient to evaluate. A first version of this idea was proposed in [35] where

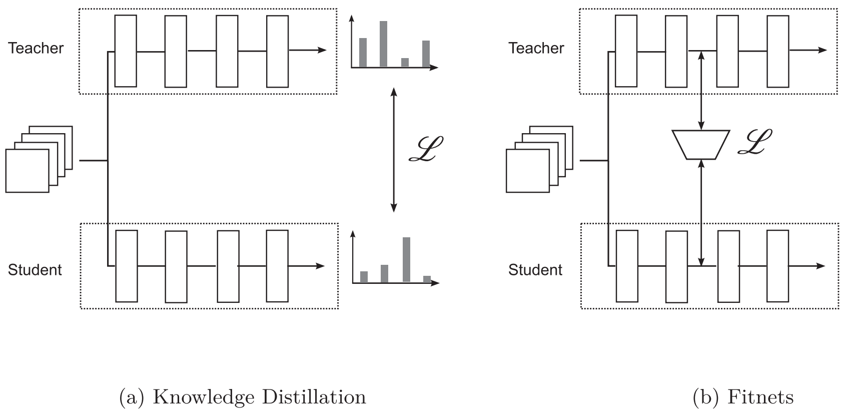


Fig. 1. Conceptual difference between Knowledge Distillation (a) and Fitnets (b). Knowledge distillation uses the soft output of the teacher as a target for the student. Fitnets use the intermediate representations to guide the learning process of the student. $\mathscr{L}$ indicates the loss function that is optimised with gradient descent.

a large trained ensemble was used to label additional data that can then be used to train a new more compact network.

More recently Hinton et al. introduced an elegant transfer technique called Knowledge Distillation [36]. It is based on the observation that the output of the trained teacher (the probability distribution of the classes) can be used as a soft target for the student. This soft target provides more information than a hard class label since it also encodes information about the similarity between classes. This makes it easier for the student to discover structure in the data. A neural network trained for classification typically uses a softmax activation (Eq. (2.3)) for the last layer. This activation transforms the logits z_i to probabilities q_i:

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \quad (2)$$

Where T is a *temperature* parameter that is typically set to 1. Smaller values of T cause the network to produce more confident results while larger values of T cause a softer probability distribution over the classes. The Knowledge Distillation technique passes a batch of training data through the teacher network and uses the obtained probability distribution as a soft target for the student. To train the student we minimize the cross-entropy loss (Eq. (3)) between the soft target $p(x)$ and the output of the student $q(x)$.

$$H(p,q) = -\sum_i p(x) log(q(x)) \quad (3)$$

If the correct labels are available for (some of) the training samples we can use a weighted sum of two cross entropy loss functions. One calculated on the soft targets and the other calculated on the hard ground truth labels.

The idea of distillation was later extended in Fitnets [37] where the intermediate representations of the teacher were used to guide the training process of the student in addition to the soft outputs. The student is encouraged to have a similar intermediate representation as the teacher. Since the dimensionalities of the intermediate representations of both networks do not necessarily correspond they added additional regressor layers that could map the intermediate representation of the student to the spatial size as the intermediate representation of the teacher. The student is trained to minimize the euclidian loss function shown in Eq. (4) where p_i calculates the intermediate representation of the teacher network up to layer i and q_j similarly calculates the activations of the student network after layer j. r is the regressor network that converts the activations of the student to the same size as the the activations of the teacher.

$$f_i(B) = \frac{1}{2}\|p_i(x) - r(q_j(x))\| \quad (4)$$

Both approaches are illustrated in Fig. 1. Knowledge distillation on the left uses the output of the teacher as a soft target to train the student while Fitnets (right) rely on layer wise pretraining.

3. Similarity based knowledge transfer

Deep neural networks use multiple layers to transform a high dimensional input into an abstract output such as a class label. Each layer transforms its input into a representation that makes it easier to distinguish the different classes for the next layer.

We propose to explicitly use this property to guide the training of the student. We pass a batch B of b images through the teacher network and record the intermediate representations $teacher_i(B)$ after layer i. We then calculate the cosine distances d_{xy} between the representations of each example pair (x, y) following Eq. (5). The resulting $b*b$ distance matrix gives us an idea of the transformation that the neural network has learned after layer i. A single element d_{xy} in the distance matrix measures how similar two input samples x and y are according to the network up to layer i.

$$d_{xy} = cos(x,y) = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\,\|\mathbf{y}\|} = \frac{\sum\limits_{i=0}^{n} x_i y_i}{\sqrt{\sum\limits_{i=0}^{n} x_i^2}\sqrt{\sum\limits_{i=0}^{n} y_i^2}} \quad (5)$$

We then pass the exact same batch through the student network to record its intermediate representations $student_j(B)$ after layer j and calculate the corresponding distance matrix again following Eq. (5).

We train the student with gradient descent by minimizing the cosine distance between the two distance matrices. This encourages the student network to learn a transformation that mimics the behaviour of the teacher. Two images that have a similar intermediate representation for the teacher should also have a similar intermediate representation in the student network. There is no constraint on the similarity of the learned representations between teacher and student, the student can learn completely different features from the teacher as long as two images that are (dis)similar to the teacher are also (dis)similar to the student. This process is illustrated in Fig. 2.

We repeat this procedure for different (i, j) layer combinations and finetune the network afterwards using supervised learning by minimizing the cross-entropy loss function from Eq. (3) between the predictions and the true labels. Our technique is also compatible with Knowledge Distillation during the finetuning stage but we found that this only has a minimal effect on the final performance. The full algorithm can be found in Algorithm 2

Algorithm 2 Pretraining and finetuning with similarity based knowledge transfer.

```
1: procedure PRETRAIN($i, j$)
2:     Input: The indices i, j of the layers of respectively the teacher and the
3:     student that knowledge should be transferred between and a set of
4:     (unlabelled) training samples.
5:     for each batch $B$ of training samples do
6:         $y_i \leftarrow teacher_i(B)$
7:         $y_j \leftarrow student_j(B)$
8:         $d_{teacher} \leftarrow cos(y_i)$
9:         $d_{student} \leftarrow cos(y_j)$
10:        $loss \leftarrow cos(d_{teacher}, d_{student})$
11:        perform gradient update to the weights of the student network
12:    end for
13: end procedure

14: procedure FINETUNE
15:     Input: A set of labelled training samples.
16:     for each batch $B$ of training samples and associated labels $L$ do
17:         $y \leftarrow student(B)$
18:         $loss \leftarrow cross\_entropy(y, L)$
19:         perform gradient update to the weights of the student network
20:     end for
21: end procedure

22: procedure MAIN
23:     Input: A list of (i,j) combinations indicating the indices of the layers of
24:     respectively the teacher and the student that knowledge should be
25:     transferred between.
26:     for each (i,j) do
27:         $pretrain(i, j)$
28:     end for
29:     $finetune()$
30: end procedure
```

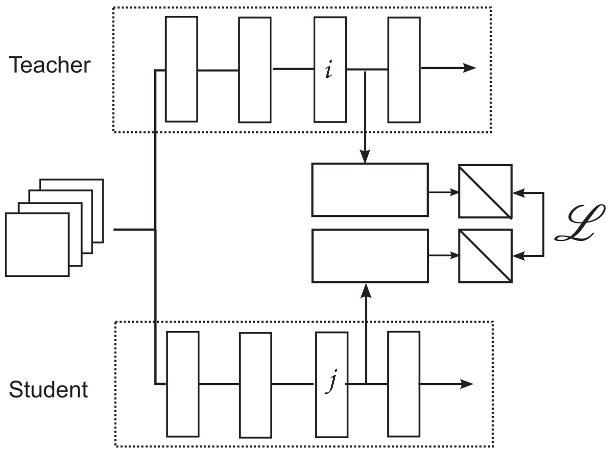


Fig. 2. Our similarity based Knowledge Transfer technique: The same batch is passed through the teacher network and the student network. Both intermediate representations are recorded and used to calculate two distance matrices. We minimise the cosine distance (indicated by $\mathscr{L}$) between the two distance matrices, forcing the student to learn a transformation where two images have a similar intermediate representation only when they have similar intermediate representations in the teacher network.

To calculate the loss term during pretraining we only need the two distance matrices. The two networks can have completely different architectures (different depth, different number of convolutional filters, different nonlinearities,...). In our case the student is constrained to binary weights and activations but this technique can also be used to train floating point student networks.

Since our pretraining step is completely unsupervised we can use large amounts of new unlabelled data and only rely on labelled data for the finetuning step.

A disadvantage of knowledge transfer methods is that we need to evaluate the teacher for every train step of the student since we need the additional training signal based on the output or the intermediate representations of the teacher. Knowledge distillation uses the outputs of the teacher as soft targets which means that we always need to evaluate the entire network. Fitnets and our similarity based approach use intermediate representations which are less expensive to obtain since we only need to evaluate part of the network. It is possible to evaluate both teacher and student networks in parallel (even on different GPUs) since they are completely independent. Another solution to reduce the overhead of evaluating the teacher every time is to cache the outputs or the intermediate representations of the teacher. The teacher is a fixed network that is not changed when training the student. It is therefore possible to pass the entire training set through the teacher once to record the intermediate representations or network output. These cached representations can then be used to calculate the different loss functions of the knowledge transfer methods. This approach reduces the training time in exchange for increased storage needs.

4. Experiments

In this section we evaluate our approach on different default image classification benchmark datasets: CIFAR10/CIFAR100 and ILSVRC2012

4.1. CIFAR10 and CIFAR100

The CIFAR-10 dataset [38] consists of 60 000 32 × 32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images. The CIFAR-100 dataset is very similar. The images have the same size but are divided into 100 classes. Each class has 500 training and 100 test images for a total of 60,000 images.

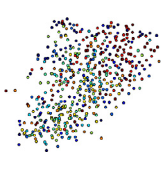
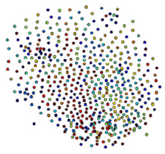
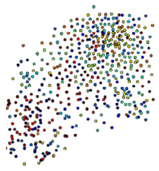
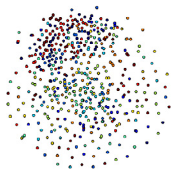
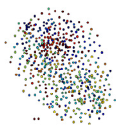
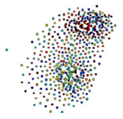
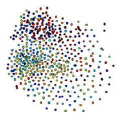
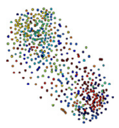
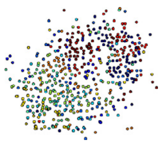
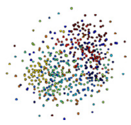
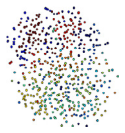
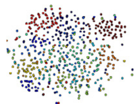
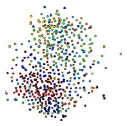
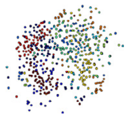
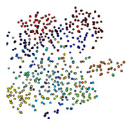
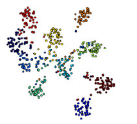
In all our CIFAR10 and CIFAR100 experiments we use the unmodified BinaryNet architecture [1] for our student. The teacher network is a Deep Residual Network [39] with 32 layers. The teacher obtains an error rate of 7% for CIFAR10 and of 30% for CIFAR100. We used Pytorch [40] for all our experiments. All networks were trained using ADAM [41] on NVIDIA GTX1080 GPUs with batchsize 64.

4.1.1. Qualitative results: Can we visualise the transferred knowledge ?

The similarity based knowledge transfer technique introduced in the previous section relies on layer wise pretraining where we iteratively train each layer to mimic the behaviour of a layer in the teacher network. The loss function forces the student to learn a mapping where images that have a similar representation in the teacher network also have similar representations in the student network.

To understand if our pretraining technique indeed learns a useful transformation we look at t-SNE [42] visualizations of the intermediate representations of the binary student network before and after pretraining each layer. These results are shown in Table 1.

Table 1
T-SNE plots of the intermediate representations of each layer before and after pretraining of that layer.

	Conv 1	Conv 2	Conv 3	Conv 4
Before				
After				
	Conv 5	**Conv 6**	**FC 1**	**FC 2**
Before				
After				

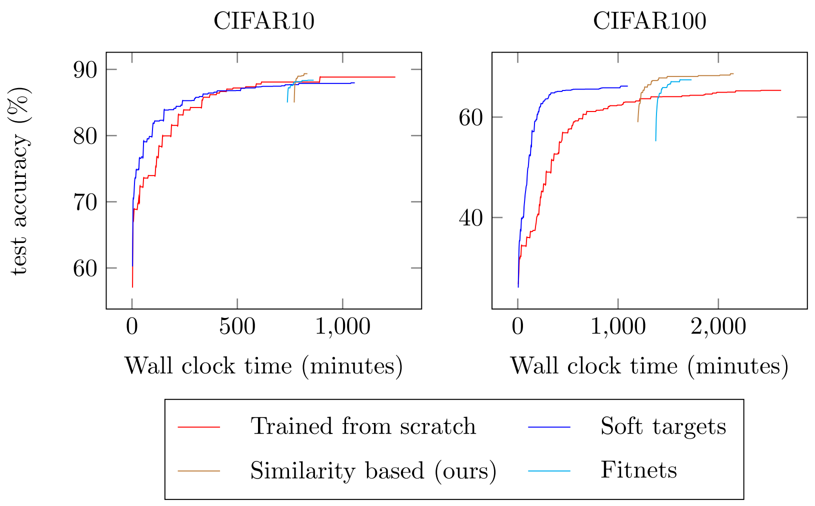


Fig. 3. Test accuracy as a function of training time for the different approaches.

t-SNE is a dimensionality reduction technique that is able to visualize high dimensional vectors in 2D scatter plots. Similar vectors (according to their euclidean distance) are shown as nearby dots in the plot while dissimilar points are further apart. Each dot corresponds to an image from the test set and the color indicates the class label. These plots were generated before the finetuning step which means that no labelled information was used while training the student.

The first scatter plot (Conv1–before transfer) shows little to no structure because all weights are initialised randomly. After pretraining this layer with our Knowledge Transfer technique we can already vaguely distinguish two clusters. Upon inspection of the samples we found that one cluster contains man-made objects such as cars, trucks and boats while the other cluster contains natural objects such as animals.

This distinction is further emphasized as we pretrain more layers. We can distinguish clear clusters of samples belonging to the same class after the last convolutional layer. The fully connected layers then further improve the decision boundary and after the last fully connected layer we can clearly discriminate the different classes, even though we have not used the class labels while training the student. This experiment shows that our Knowledge Transfer technique can train a binary neural network to distinguish between different classes based on the characteristics of the intermediate representations of the teacher network and without any labelled information.

4.1.2. Quantitative results: How does pretraining affects the test accuracy and training time ?

In our second experiment we look at the training time and the final test accuracy of the different Knowledge Transfer techniques applied to training BNNs. Fig. 3 shows the test accuracy as a function of the training time for the different approaches.We used an NVIDIA GTX1080 GPU to train all networks.

The red line corresponds to the training from scratch baseline. The dark blue line shows the accuracy when the network is trained

Table 2
Test accuracy of the BNN on CIFAR10 and CIFAR100.

Model	CIFAR10	CIFAR100
Trained from scratch	88.6%	65.3%
Distillation (soft targets)	87.9%	66.2%
Fitnet	88.4%	67.4%
Similarity based (ours)	**89.4%**	**68.7%**

with soft targets. This clearly helps the network to converge faster on both datasets but the impact is best visible on CIFAR100 where the soft targets help the network to achieve a higher test accuracy. A possible explanation is that the 100 different classes from the CIFAR100 dataset are grouped in 10 “super classes” with 10 fine grained “sub classes” each. A super class would for example be “fish” with members such as “Shark” and “Trout”. Because of this design choice there are a lot of similar classes. Distillation with soft targets can exploit this property because a single example can now for example be labelled as 60 % hamster, 30 % mouse and 10 % squirrel, providing information of all three classes to the student. Each training sample now carriers much more information compared to a single ground truth label.

Both the Fitnets approach and our similarity based technique need a pretraining stage. We start pretraining at timestamp 0 and only plot the test accuracy during the finetuning stage.

It is somewhat surprising that Fitnets work so well when training binary neural networks since Fitnets explicitly use the values of both intermediate representations and these are completely different (binary vs floating point). Yet it seems that the regressor layer that is used to change the dimensionality of the representations also takes care of the conversion of binary to floating point values. The regressor layer used floating point weights and activations. Fitnets result in a higher accuracy on both datasets compared to Distillation. On CIFAR10 this is still slightly lower than the baseline but on CIFAR100 Fitnets give us an accuracy of 67.42% compared to 65.31% for the baseline.

Our Similarity based Knowledge Transfer technique has a very similar behaviour as Fitnets. The biggest advantage of our approach compared to Fitnets is that we do not directly compare the intermediate representations. Therefore we do not require that both intermediate representations have a similar spatial size and we do not need the additional regressor layers. Instead we calculate the loss function between two similarity matrices and the dimensions of the similarity matrices only depend on the batch size. We believe that this decoupling is especially interesting when training networks with binary weights and activations since this allows us to have a completely different architectures for the student and the teacher. Our similarity based technique results in the highest test accuracy on both datasets.

The final test accuracies for all approaches are summarized in Table 2.

4.2. ImageNet

The CIFAR datasets from the previous section are small scale datasets that are easy to experiment with but the small images are not representative of real world applications. In this section we trained BNNs on the Imagenet dataset [43]. The task is to distinguish between 1000 classes. The input images are 224 by 224 pixel RGB images of real world scenes. The dataset has 1,281,167 training images. Each class has at least 732 training images.

We trained a binary version of the Alexnet architecture [3], the same network architecture that was used in the original BNN paper [24] as well as in the XNOR-net paper [31]. The drop in accuracy for a BNN compared to a floating point network is much more severe on this dataset than on the small scale datasets from the previous section. Training accurate networks with binary weights and activations remains an open problem for large and complex datasets. Table 3 shows a summary of the accuracies obtained by the different approaches.

The baseline Alexnet network with floating point operations achieves an accuracy of 56.6% (80.2% top 5). We report results for three binary neural network variants. The BNN follows the binarization approach from [24]. We reimplemented the network and training routine in Pytorch and were able to reproduce their results. We then applied our Knowledge Transfer technique and obtained a slightly higher accuracy (68.8% compared to 67.8%).

For both XNORnet and Binary Weight Networks (BWN) we were unable to exactly reproduce the results from [31] in Pytorch, probably because of different data augmentations and normalization techniques. We again find that our similarity based Knowledge Transfer technique results in slightly higher test accuracies compared to our implementations that were trained from scratch.

5. Conclusion and future work

In this work we introduced a novel Knowledge Transfer technique that uses the similarity between intermediate representations to guide the training of a student network based on a trained teacher network. We focussed on training binary neural networks for image recognition but our technique is not limited to binary neural networks nor to image classification tasks. We showed that pretraining a BNN with Knowledge transfer helps to obtain higher test accuracies compared to training from scratch. Future work will focus on improving the results on large scale datasets like Imagenet.

Table 3
Test accuracy of different binary neural network architectures on Imagenet.

Model	Top 1	Top 5
Floating point weights and activations		
Alexnet [3]	56.6%	80.2%
BNN: Binary weights and activations		
Trained from scratch [24]	41.8%	67.1%
Trained from scratch (our implementation)	41.4%	67.8%
Trained with similarity based knowledge transfer (ours)	44.2%	68.8%
XNORnet: Binary weights and activations		
Trained from scratch [31]	44.2%	69.2%
Trained from scratch (our implementation)	42.5%	68.0%
Trained with similarity based knowledge transfer (ours)	43.6%	68.7%
Binary Weight Networks (BNN): Binary weights, floating point activations		
BWN-net [31]	56.8%	79.4%
BWN-net (our implementation)	53.6%	76.8%
BWN-net trained with similarity based knowledge transfer (ours)	54.6%	77.5%

Conflict of interest

None.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of GPU hardware used for this research.

References

[1] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, in: Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or −1, 2016. arXiv: 1602.02830.

[2] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al., Predicting parameters in deep learning, Proceedings of the Advances in Neural Information Processing Systems, 2013, pp. 2148–2156.

[3] A. Krizhevsky, I. Sutskever, G.E. Hinton, in: Imagenet classification with deep convolutional neural networks, 2012, pp. 1097–1105.

[4] J. Han, D. Zhang, G. Cheng, N. Liu, D. Xu, Advanced deep-learning techniques for salient and category-specific object detection: a survey, IEEE Signal Process. Mag. 35 (1) (2018) 84–100.

[5] J. Han, R. Quan, D. Zhang, F. Nie, Robust object co-segmentation using background prior, IEEE Trans. Image Process. 27 (4) (2018) 1639–1651.

[6] G. Cheng, C. Yang, X. Yao, L. Guo, J. Han, When deep learning meets metric learning: remote sensing image scene classification via learning discriminative cnns, IEEE Trans. Geosci. Remote Sens. 56 (5) (2018) 2811–2821.

[7] A.Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, A.Y. Ng, Deep speech: scaling up end-to-end speech recognition, CoRR (2014) arXiv:1412.5567.

[8] S. Levine, P. Pastor, A. Krizhevsky, D. Quillen, Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection, CoRR (2016) arXiv:1603.02199.

[9] G. Cheng, P. Zhou, J. Han, Duplex metric learning for image set classification, IEEE Trans. Image Process. 27 (1) (2018) 281–292.

[10] J. Han, G. Cheng, Z. Li, D. Zhang, A unified metric learning-based framework for co-saliency detection, IEEE Trans. Circuits Syst. Video Technol. (2017) 2473–2483.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Proceedings of the Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.

[12] L.A. Gatys, A.S. Ecker, M. Bethge, Image style transfer using convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2414–2423.

[13] A. Karpathy, L. Fei-Fei, Deep visual-semantic alignments for generating image descriptions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3128–3137.

[14] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.

[15] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage, in: Proceedings of the Advances in Neural Information Processing Systems, 1990, pp. 598–605.

[16] B. Hassibi, D.G. Stork, et al., Second order derivatives for network pruning: optimal brain surgeon, Proceedings of the Advances in Neural Information Processing Systems, 1993. 164–164.

[17] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Proceedings of the Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.

[18] M. Jaderberg, A. Vedaldi, A. Zisserman, in: Speeding up convolutional neural networks with low rank expansions, 2014. arXiv: 1405.3866.

[19] V. Sindhwani, T. Sainath, S. Kumar, Structured transforms for small-footprint deep learning, in: Proceedings of the Advances in Neural Information Processing Systems, 2015, pp. 3088–3096.

[20] W. Chen, J.T. Wilson, S. Tyree, K.Q. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in: Proceedings of the International Conference on Machine Learning, ICML, 2015, pp. 2285–2294.

[21] V. Vanhoucke, A. Senior, M.Z. Mao, Improving the speed of neural networks on CPUs, in: Proceedings of the Deep Learning and Unsupervised Feature Learning NIPS Workshop, 1, Citeseer, 2011, p. 4.

[22] S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding, in: Proceedings of the International Conference on Learning Representations, 2015. (ICLR best paper award)

[23] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: training deep neural networks with binary weights during propagations, in: Proceedings of the Advances in Neural Information Processing Systems, 2015, pp. 3123–3131.

[24] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, in: Quantized neural networks: training neural networks with low precision weights and activations, 2016. arXiv: 1609.07061.

[25] D.L. Gray, A.N. Michel, A training algorithm for binary feedforward neural networks, IEEE Trans. Neural Netw. 3 (2) (1992) 176–194.

[26] R. Sato, T. Saito, Stabilization of desired periodic orbits in dynamic binary neural networks, Neurocomputing 248 (2017) 19–27.

[27] J.H. Kim, S.-K. Park, The geometrical learning of binary neural networks, IEEE Trans. Neural Netw. 6 (1) (1995) 237–247.

[28] F. Chen, G. Chen, Q. He, G. He, X. Xu, Universal perceptron and DNA-like learning algorithm for binary neural networks: non-LSBF implementation, IEEE Trans. Neural Netw. 20 (8) (2009) 1293–1301.

[29] S. Ioffe, C. Szegedy, in: Batch normalization: accelerating deep network training by reducing internal covariate shift, 2015. arXiv: 1502.03167.

[30] Y. Bengio, N. Léonard, A. Courville, in: Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. arXiv: 1308.3432.

[31] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: imagenet classification using binary convolutional neural networks, in: Proceedings of the European Conference on Computer Vision, Springer, 2016, pp. 525–542.

[32] A.G. Anderson, C.P. Berg, in: The high-dimensional geometry of binary neural networks, 2017. arXiv: 1705.07199.

[33] Y. Umuroglu, N.J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, K. Vissers, Finn: a framework for fast, scalable binarized neural network inference, in: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM, 2017, pp. 65–74.

[34] X. Sun, X. Peng, P.-Y. Chen, R. Liu, J.-s. Seo, S. Yu, Fully parallel RRAM synaptic array for implementing binary neural network with (+ 1,−1) weights and (+ 1, 0) neurons, in: Proceedings of the Twenty Third Asia and South Pacific Design Automation Conference, IEEE Press, 2018, pp. 574–579.

[35] C. Bucilua, R. Caruana, A. Niculescu-Mizil, Model compression, in: Proceedings of the Twelfth ACM SIGKDD international conference on Knowledge Discovery and Data Mining, ACM, 2006, pp. 535–541.

[36] G. Hinton, O. Vinyals, J. Dean, in: Distilling the knowledge in a neural network, 2015. arXiv: 1503.02531.

[37] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, in: Fitnets: hints for thin deep nets, 2014. arXiv: 1412.6550.

[38] A. Krizhevsky, G. Hinton, in: Learning multiple layers of features from tiny images, 2009.

[39] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, CoRR (2015) arXiv:1512.03385.

[40] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, in: Automatic differentiation in pytorch, 2017.

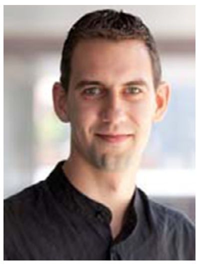
[41] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, CoRR (2014) arXiv:1412.6980.

[42] L.v.d. Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (Nov) (2008) 2579–2605.

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. (IJCV) 115 (3) (2015) 211–252, doi:10.1007/s11263-015-0816-y.

Sam Leroux received his M.Sc. degree in Information Engineering Technology from Ghent University, Belgium in July 2014. In September of that year, he joined the Department of Information Technology at Ghent University, where he is active as a Ph.D. student. His main research interests are machine learning, neural networks, deep learning and cloud computing. He is also active as a teaching assistant for various courses in both the bachelor and master of Science in Information Engineering Technology program.

Bert Vankeirsbilck received a M.Sc. Degree (2007) and a Ph.D. Degree (2013) in Computer Science Engineering from Ghent University. Since June 2013, he has been active as a postdoctoral research at the dept of Information Technology at the same university. From a Ph.D. topic on optimization of quality of experience for mobile thin client systems, the focus broadened towards resource constrained computing and distributed intelligence, mostly supported by software design based on edge cloud architectures.



Tim Verbelen received his M.Sc. degree in Computer Science from Ghent University, Belgium in June 2009. In July 2013, he received his Ph.D. degree with his dissertation “Adaptive Offloading and Configuration of Resource Intensive Mobile Applications”. Since August 2009, he has been working at the Departement of Information Technology (INTEC) of the Faculty of Engineering at Ghent University, and is now active as postdoctoral researcher. His main research interests include mobile cloud computing and adaptive software. Specifically he is researching adaptive strategies to enhance real-time applications such as Augmented Reality on mobile devices.

Pieter Simoens received his M.Sc. degree in Electronic Engineering (2005) and Ph.D. degree (2011) from the Ghent University, Belgium. During his Ph.D. research, he was funded by the Fund for Scientific Research Flanders (FWO-V). In 2012, he was a visiting researcher at the School of Computer Science of Carnegie Mellon University, USA. Currently, he is professor affiliated with the Department of Information Technology of the Ghent University and with imec. He is teaching courses on Mobile Application Development and Software Engineering. His main research interests include mobile cloud offloading, service-oriented networking, edge/fog computing paradigms, and service engineering for advanced mobile applications. In these fields, he is author and co-author of more than 70 papers published in international journals or in the proceedings of international conferences. He has also been involved in several national and European research projects (FP6 MUSE, FP7 MobiThin, H2020 FUSION).

Bart Dhoedt received a Masters degree in Electro-technical Engineering (1990) from Ghent University. His research, addressing the use of micro-optics to realize parallel free space optical interconnects, resulted in a Ph.D. degree in 1995. After a 2-year post-doc in opto-electronics, he became Professor at the Department of Information Technology. He is responsible for various courses on algorithms, advanced programming, software development and distributed systems. His research interests include software engineering, distributed systems, mobile and ubiquitous computing, smart clients, middleware, cloud computing and autonomic systems. He is author or co-author of more than 300 publications in international journals or conference proceedings.