Efficient and Robust Machine Learning for Real-World Systems

Franz Pernkopf, *Senior Member, IEEE*, Wolfgang Roth, Matthias Zöhrer, Lukas Pfeifenberger, Günther Schindler, Holger Fröning, Sebastian Tschiatschek, Robert Peharz, Matthew Mattina, Zoubin Ghahramani

Abstract—While machine learning is traditionally a resource intensive task, embedded systems, autonomous navigation and the vision of the Internet-of-Things fuel the interest in resource efficient approaches. These approaches require a carefully chosen trade-off between performance and resource consumption in terms of computation and energy. On top of this, it is crucial to treat uncertainty in a consistent manner in all but the simplest applications of machine learning systems. In particular, a desideratum for any real-world system is to be robust in the presence of outliers and corrupted data, as well as being "aware" of its limits, i.e. the system should maintain and provide an uncertainty estimate over its own predictions. These complex demands are among the major challenges in current machine learning research and key to ensure a smooth transition of machine learning technology into every day's applications. In this article, we provide an overview of the current state of the art of machine learning techniques facilitating these real-world requirements. First we provide a comprehensive review of resource-efficiency in deep neural networks with focus on techniques for model size reduction, compression and reduced precision. These techniques can be applied during training or as post-processing and are widely used to reduce both computational complexity and memory footprint. As most (practical) neural networks are limited in their ways to treat uncertainty, we contrast them with probabilistic graphical models, which readily serve these desiderata by means of probabilistic inference. In that way, we provide an extensive overview of the current state-of-the-art of robust and efficient machine learning for real-world systems.

Index Terms—Resource-efficient machine learning, inference, robustness, deep neural networks, probabilistic graphical models.

1 Introduction

Machine learning is a key technology in the 21st century and the main contributing factor for many recent performance boosts in computer vision, natural language processing, speech recognition and signal processing. Today, the main application domain and comfort zone of machine learning applications is the "virtual world", as found in recommender systems, stock market prediction, and social media services. However, we are currently witnessing a transition of machine learning moving into "the wild", where most prominent examples are autonomous navigation for personal transport and delivery services, and the Internet of Things (IoT). Evidently, this trend opens several real-world challenges for machine learning engineers.

Current machine learning approaches prove particularly effective when big amounts of data and ample computing resources are available. However, in real-world applications the computing infrastructure during the operation phase is typically limited, which effectively rules out most of the current resource-hungry machine learning approaches. There are several key challenges – illustrated in Figure 1 – which have to be jointly considered to facilitate machine learning in real-world applications:

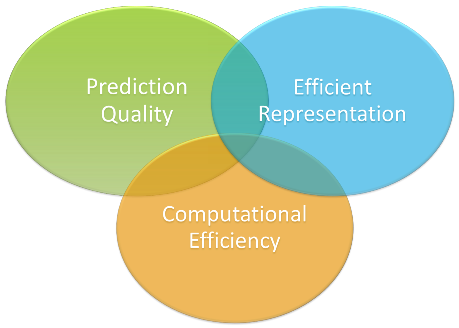


Fig. 1: Aspects of resource-efficient machine learning models.

- Efficient representation: The model complexity, i.e. the number of model parameters, should match the usually limited resources in deployed systems, in particular regarding memory footprint.
- Computational efficiency: The machine learning model should be computationally efficient during inference, exploiting the available hardware optimally with respect to time and energy. For instance, power constraints are key for autonomous and em-

- *W. Roth, M. Zöhrer, L. Pfeifenberger, and F. Pernkopf are with the Department of Electrical Engineering, Laboratory of Signal Processing and Speech Communication, Graz University of Technology, Austria.*
G. Schindler and H. Fröning are with the Institute of Computer Engineering, Ruperts Karls University, Heidelberg, Germany.
S. Tschiatschek is with Microsoft Research, Cambridge, UK.
R. Peharz is with the Machine Learning Group, Department of Engineering, University of Cambridge, UK.
Z. Ghahramani is with the Machine Learning Group, Department of Engineering, University of Cambridge, UK and Uber AI Labs, California, USA
M. Mattina is with Arm Research, Arm Ltd., Cambridge, UK.
Correspondence e-mail: pernkopf@tugraz.at

bedded systems, as the device lifetime for a given battery charge needs to be maximized, or constraints set by energy harvesters need to be met.

- Prediction quality: The focus of classical machine learning is mostly on optimizing the prediction quality of the models. For embedded devices, model complexity versus prediction quality trade-offs must be considered to achieve good prediction performance while simultaneously reducing computational complexity and memory requirements.

Furthermore, in the "non-virtual" world, we have only limited control over data quality. Corrupted data, missing inputs, and outliers are the rule rather than the exception. These real-world conditions require a high degree of robustness of machine learning systems under corrupted inputs, as well as the model's ability to deliver well-calibrated predictive uncertainty estimates. This point is especially crucial if the system shall be involved in any critical decision making processes.

In this article, we review the state of the art in machine learning with regard to these real-world requirements. We first focus on deep neural networks (DNN), the currently predominant machine learning models. While being the driving factor behind many recent success stories, DNNs are notoriously data and resource hungry, a property which has recently renewed significant research interest in resource-efficient approaches. The first part of this tutorial is dedicated to an extensive overview of these approaches, all of which exploit the following two generic strategies to (i) reduce model size in terms of number of weights and/or neurons, or (ii) reduce arithmetic precision of parameters and/or computational units. Evidently, these two basic techniques are almost "orthogonal directions" towards efficiency in DNNs, and they can be naturally combined, e.g. one can both sparsify a model and reduce arithmetic precision.

Nevertheless, most research emphasizes one of these two techniques, so that we discuss them separately. In Section 2.1, we first discuss approaches to reduce the model size in DNNs, using pruning techniques, weight sharing, factorized representations, and knowledge distillation. In Section 2.2, we focus on techniques for reduced arithmetic precision in DNNs. When driven to the extreme, this approach leads to discrete DNNs, with only a few values for weights and/or activations. Even reducing precision down to binary or ternary values works reasonably well and essentially reduces DNNs to hardware-friendly logical circuits. This extreme reduction, however, introduces challenging discrete optimization problems. Besides various optimization heuristics, such as the straight-through estimator, we also discuss an alternative approach, casting the problem as Bayesian posterior inference. The latter approach can be naturally tackled with variational approximations, leading to a continuous optimization problem.

The Bayesian approach also readily incorporates uncertainty treatment into machine learning models, as weight uncertainty represented by the (approximate) posterior directly translates into well-calibrated output uncertainties. Clearly, however, processing a full posterior during test time is computationally demanding and largely opposed to our primary goal of resource-efficiency. Thus, in practice a compromise must be made, realizable via small ensembles of discrete DNNs.

As an alternative to DNNs we discuss classical probabilistic graphical models (PGMs) in Section 3. PGMs naturally lend themselves towards resource-efficient machine learning systems, typically yielding models which are several orders of magnitude smaller than DNNs, while still obtaining decent predictive performance. Furthermore, they treat uncertainty in a natural way by virtue of statistical inference and often dramatically outperform DNNs when a considerable number of input features are missing. Moreover, generative or hybrid learning approaches yield well-calibrated uncertainties over both inputs and outputs which can naturally be exploited in outlier and abnormality detection. Similarly to DNNs, PGMs can be subjected to efficiency optimizations by employing structure learning and reduced-precision parameters. In particular, for inference scenarios like classification, they can be highly efficient, requiring only integer additions.

In Section 4 we substantiate our discussion with experimental results. First, we exemplify the trade-off between execution time, memory footprint and predictive accuracy in DNNs on a CIFAR-10 classification task. Subsequently, we provide an extensive comparison of various hardware-efficient strategies for DNNs, using the challenging task of ImageNet classification. In particular, this overview shows that sensible trade-offs can be achieved with very low numeric precision, such as only one bit per activation and DNN weight. We demonstrate that these trade-offs can be readily exploited on today's hardware, by benchmarking the core operation of binary DNNs (BNNs), i.e. binary matrix multiplication, on NVIDIA Tesla K80 and ARM Cortex-A57 architectures. Furthermore, a complete real-world signal processing example using BNNs is discussed in Section 4.3. In this example we develop a complete speech enhancement system employing an efficient BNN-based speech mask estimator, which shows negligible performance degradation while allowing memory savings of factor 32 and speed-ups of roughly a factor 10. Furthermore, exemplary results comparing PGMs and DNNs on the classical MNIST data set are provided where the focus is on prediction performance and number of bits necessary for representing the models. DNNs slightly outperform PGMs on MNIST while PGMs are able to naturally handle missing feature scenarios. An example of randomly missing features during model testing is finally provided.

2 Deep Neural Networks

DNNs are the currently dominant approach in machine learning, and have led to significant performance boosts in various application domains, such as computer vision [1], speech and natural language processing [2], [3]. In [4], key aspects of deep models have been identified explaining some of the performance gains, namely, the re-use of features in consecutive layers and the degree of abstraction of features at higher layers.

Furthermore, the performance improvements can be largely attributed to increasing hardware capabilities that enabled the training of ever-increasing network architectures and the usage of big data. Since recently there is growing interest in making DNNs available for embedded devices by developing fast and energy-efficient architectures with little memory requirements. These methods reduce either the number of connections and parameters (Section 2.1), the parameters' precision (Section 2.2), or both, as discussed in the sequel.

2.1 Model Size Reduction in DNNs

In the following, we review methods that reduce the number of weights and neurons in DNNs using techniques like pruning, sharing, but also more complex methods like knowledge distillation and special data structures.

2.1.1 Weight Pruning and Neuron Pruning

One of the earliest approaches to reduce network size is LeCun et al.'s *optimal brain damage* algorithm [5]. Their main finding is that pruning based on weight magnitude is suboptimal and they propose a pruning scheme based on the increase in loss function. Assuming a pre-trained network, a local second-order Taylor expansion with a diagonal Hessian approximation is employed that allows to estimate the change in loss function caused by weight pruning without re-evaluating the costly network function. Removing parameters is alternated with re-training the pruned network. In that way, the model can be reduced significantly without deteriorating its performance. Hassibi and Stork [6] found the diagonal Hessian approximation to be too restrictive, and their *optimal brain surgeon* algorithm uses an approximated full covariance matrix instead. While their method, similarly as [5], prunes weights that cause the least increase in loss function, the remaining weights are simultaneously adapted to compensate for the negative effect of weight pruning. This bypasses the need to alternate several times between pruning and re-training the pruned network.

However, it is not clear whether these approaches scale up to modern DNN architectures since computing the required (diagonal) Hessians is substantially more demanding (if not intractable) for millions of weights. Therefore, many of the more recently proposed techniques still resort to magnitude based pruning. Han et al. [7] alternate between pruning connections below a certain magnitude threshold and re-training the pruned network. The results of this simple strategy are impressive, as the number of parameters in pruned networks is an order of magnitude smaller ($9\times$ for AlexNet and $13\times$ for VGG-16) than in the original networks. Hence, this work shows that neural networks are in general heavily over-parametrized. In a follow-up paper, Han et al. [8] proposed *deep compression*, which extends the work in [7] by a parameter quantization and parameter sharing step, followed by Huffman coding to exploit the non-uniform weight distribution. This approach yields a 35-$49\times$ improvement in memory footprint and consequently a reduction in energy consumption of 3-$5\times$.

Guo et al. [9] discovered that irreversible pruning decisions limit the achievable sparsity and that it is useful to reincorporate weights pruned in an earlier stage. In addition to full weight matrices, they maintain a set of weight masks that determine whether a weight is currently pruned or not. Their method alternates between updating the weights based on gradient descent, and updating the weight masks by thresholding. Most importantly, weight updates are also applied to weights that are currently pruned such that pruned weights can reappear if their value exceeds a certain threshold. This yields a $17.7\times$ parameter reduction for AlexNet without deteriorating performance.

Wen et al. [10] incorporated group lasso regularizers in the objective to obtain different kinds of sparsity in the course of training. They were able to remove filters, channels, and even entire layers for architectures where shortcut connections are used.

In [11], [12], variational inference is employed to train for each connection a weight variance w_{σ^2} in addition to a single (mean) weight w_μ. After training, weights are pruned according to the "signal-to-noise ratio" $|w_\mu / w_\sigma|$. Molchanov et al. [13] proposed a method based on Kingma et al.'s variational dropout [14] which interprets dropout as performing variational inference with specific prior and approximate posterior distributions. Within this framework, the otherwise fixed dropout rates appear as free parameters that can be optimized to improve a variational lower bound. In [13], this freedom is exploited to optimize weight dropout rates such that weights can be safely pruned if their dropout rate is close to one. This idea has been extended in [15] by using sparsity enforcing priors and assigning dropout rates to groups of weights that are all connected to the same neuron which in turn allows the pruning of entire neurons. Furthermore, they show how their approach can be used to determine an appropriate bit width for each weight by exploiting the well-known connection between Bayesian inference and the minimum description length (MDL) principle [16]. We elaborate more on Bayesian approaches in Section 2.2.3.

In [17], a determinantal point process (DPP) is used to find a group of neurons that are diverse and exhibit little redundancy. Conceptionally, a DPP for a given ground set $\mathcal{S}$ defines a distribution over subsets $S \subseteq \mathcal{S}$ where subsets containing diverse elements have high probability. The DPP is used to sample a diverse set of neurons and the remaining neurons are then pruned.

To compensate for the negative effect of pruning, the outgoing weights of the kept neurons are adapted so as to minimize the activation change of the next layer.

2.1.2 Weight Sharing

A further technique to reduce the model size is weight-sharing. In [18], a hashing function is used to randomly group network connections into "buckets", where the connections in each bucket shares a single weight value. This has the advantage that weight assignments need not be stored explicitly but are given implicitly by the hashing function. This allows to train $10\times$ smaller networks while the predictive performance is essentially unaffected. Ullrich et al. [19] extended the soft weight-sharing approach proposed in [20] to achieve both weight sharing and sparsity. The idea is to select a Gaussian mixture model prior over the weights and to train both the weights as well as the parameters of the mixture components. During training, the mixture components collapse to point measures and each weight gets attracted by a certain weight component. After training, weight sharing is obtained by assigning each weight to the mean of the component that best explains it, and weight pruning is obtained by fixing the mean of one component to zero and assigning it a relatively high mixture mass.

2.1.3 Knowledge Distillation

Knowledge distillation [21] is an indirect approach where first a large model (or an ensemble of models) is trained, and subsequently soft-labels obtained from the large model are used as training data for a smaller model. The smaller models achieve performances almost identical to that of the larger models which is attributed to the valuable information contained in the soft-labels. Inspired by knowledge distillation, Korattikara et al. [22] reduced a large ensemble of DNNs, used for obtaining Monte-Carlo estimates of a posterior predictive distribution, to a single DNN.

2.1.4 Special Weight Matrix Structures

There also exist approaches that aim at reducing the model size on a more global scale by (i) reducing the parameters required to represent the large matrices involved in DNN computations, or by (ii) employing certain matrix structures that facilitate low-resource computation in the first place. Denil et al. [23] propose to represent weight matrices $\mathbf{W} \in \mathbb{R}^{m \times n}$ using a low-rank approximation $\mathbf{UV}$ with $\mathbf{U} \in \mathbb{R}^{m \times k}$, $\mathbf{V} \in \mathbb{R}^{k \times n}$, and $k < \min\{m, n\}$ to reduce the number of parameters. Instead of learning both factors $\mathbf{U}$ and $\mathbf{V}$, prior knowledge, such as smoothness of pixel intensities in an image, is incorporated to compute a fixed $\mathbf{U}$ using kernel-techniques or auto-encoders, and only the factor $\mathbf{V}$ is learned. This approach is motivated by training only a subset of the weights and predicting the values of the other weights from this subset. In [24], the Tensor Train matrix format is employed to substantially reduce the number of parameters required to represent large weight matrices of fully-connected layers. Their approach enables the training of very large fully-connected layers with relatively few parameters and they show better performance than simple low-rank approximations. Denton et al. [25] propose specific low-rank approximations and clustering techniques for individual layers of pre-trained convolutional DNNs (CNN) to both reduce memory-footprint and computational overhead. Their approach yields substantial improvements for both the computational bottleneck in the convolutional layers and the memory bottleneck in the fully-connected layers. By fine-tuning after applying their approximations, the performance degradation is kept at a decent level. Jaderberg et al. [26] propose two different methods to approximate pre-trained CNN filters as combinations of rank-1 basis filters to speed-up computation. The rank-1 basis filters are obtained either by minimizing a reconstruction error of the original filters or by minimizing a reconstruction error of the outputs of the convolutional layers. Lebedev et al. [27] approximate the convolution tensor by a low-rank approximation using non-linear least squares. Subsequently, the convolution using this low-rank approximation is performed by four consecutive convolutions, each with a smaller filter, to reduce the computation time substantially. In [28], the weight matrices of fully-connected layers are restricted to circulant matrices $\mathbf{W} \in \mathbb{R}^{n \times n}$, which are fully specified by only n parameters. While this dramatically reduces the memory footprint of fully-connected layers, circulant matrices also facilitate faster computation as matrix-vector multiplication can be efficiently computed using the fast Fourier transform. In a similar vein, Yang et al. [29] reparameterize matrices $\mathbf{W} \in \mathbb{R}^{n \times n}$ of fully-connected layers using the Fastfood transform as $\mathbf{W}{=}\mathbf{SHG\Pi HB}$, where $\mathbf{S}$, $\mathbf{G}$, and $\mathbf{B}$ are diagonal matrices, $\mathbf{\Pi}$ is a random permutation matrix, and $\mathbf{H}$ is the Walsh-Hadamard matrix. This reparameterization requires only a total of $4n$ parameters, and similar as in [28], the fast Hadamard transform enables the efficient computation of matrix-vector products. Iandola et al. [30] introduced *SqueezeNet*, a special CNN structure that requires far less parameters while maintaining similar performance as AlexNet on the ImageNet data set. Their structure incorporates both 1×1 and 3×3 convolutions, and they use, similar as in [31], global average pooling of per-class feature maps that are directly fed into the softmax in order to avoid fully-connected layers that typically consume the most memory. Furthermore, they show that their approach is compatible with deep compression [8] to reduce the memory footprint to less than 0.5MB.

2.2 Reduced Precision in DNNs

As already mentioned before, the two main approaches to reduce the model size of DNNs are structure sparsification and reducing parameter precision. These approaches are to some extent orthogonal techniques to each other. Both strategies reduce the memory footprint accordingly and are vital for the deployment of DNNs in many real-world applications. Importantly, as pointed out in [8], [32], [33], reduced memory requirements are the main contributing factor to reduce the energy consumption as well. Furthermore, model sparsification also impacts the computational demand measured in terms of number of arithmetic operations. Unfortunately, this reduction in the mere number of arithmetic operations usually does not directly translate into savings of wall-clock time, as current hardware and software are not well-designed to exploit model sparseness [34]. Reducing parameter precision, on the other hand, proves very effective for improving execution time [35]. When the latter point is driven to the extreme, i.e. assuming binary weights $w \in \{-1, 1\}$ or ternary weights $w \in \{-1, 0, 1\}$ in conjunction with binary inputs and/or hidden units $x \in \{-1, 1\}$, floating or fixed point multiplications are replaced by hardware-friendly logical XNOR and bitcount operations. In that way, a sophisticated DNN is essentially reduced to a logical circuit. However, training such discrete-valued DNNs[1] is delicate as they cannot be directly optimized using gradient based methods. In the sequel, we provide a literature overview of approaches that use reduced-precision computation to facilitate low-ressource training and/or testing.

2.2.1 Stochastic Rounding

Approaches for reduced-precision computations date back at least to the early 1990s. Höhfeld and Fahlman [36], [37] rounded the weights during training to fixed-point format with different numbers of bits. They observed that training eventually stalls, as small gradient updates are always rounded to zero. As a remedy, they proposed stochastic rounding, i.e. rounding values to the nearest value with a probability proportional to the distance to the nearest value. These quantized gradient updates are correct in expectation, do not cause training to stall, and yield substantially less bits than when using deterministic rounding.

More recently, Gupta et al. [38] have shown that stochastic rounding can also be applied for modern deep architectures, as demonstrated on a hardware prototype. Courbariaux et al. [39] empirically studied the effect of different numeric formats (floating point, fixed point, and dynamic fixed point) with varying bit widths on the performance of DNNs. Lin et al. [40] proposes a method to dramatically reduce the number of multiplications required during training. At forward propagation, the weights are stochastically quantized to either binary weights $w \in \{-1, 1\}$ or ternary weights $w \in \{-1, 0, 1\}$ to remove the need for multiplications at all. During backpropagation, inputs and hidden neurons are quantized to powers of two, reducing multiplications to cheaper bit-shift operations, leaving only a negligible number of floating-point multiplications to be computed. However, the speed-up is limited to training since for testing the full-precision weights are required. Lin et al. [41] consider fixed-point quantization of pre-trained full-precision DNNs. They formulate an optimization problem that minimizes the total number of bits required to store the weights and the activations under the constraint that the total output signal-to-quantization noise ratio is larger than a certain pre-specified value. A closed-form solution of the convex objective yields layer-specific bit widths.

2.2.2 Straight-Through Estimator (STE)

In recent years, the straight-through estimator (STE) [42] became the method of choice for training DNNs with weights that are represented using a very small number of bits. Quantization operations, being piecewise constant functions with either undefined or zero gradients, are not applicable to gradient-based learning using backpropagation. The idea of the STE is to simply replace piecewise constant functions with a non-zero artificial derivative during backpropagation, as illustrated in Figure 2. This allows gradient information to flow backwards through piecewise constant functions to subsequently update parameters based on the approximated gradients. Note that this also allows for the training of DNNs with quantized activation functions such as the sign function. Approaches based on the STE typically maintain a set of full-precision weights that are quantized during forward propagation. After backpropagation, gradient updates are applied to the full-precision weights. At test time, the full-precision weights are abandoned and only the quantized reduced-precision weights are kept.

In [43], binary-weight DNNs are trained using STE to get rid of expensive floating-point multiplications. They consider deterministic and stochastic rounding during forward propagation and update a set of full-precision weights based on the gradients of the quantized weights. In [44], the STE is used to quantize both the weights and the activations to a single bit with sign functions. This reduces the computational burden dramatically as floating-point multiplications and additions are reduced to hardware-friendly logical XNOR and bitcount operations, respectively. Li et al. [45] trained ternary weights $w \in \{-a, 0, a\}$ by setting weights lower than a certain threshold Δ to zero, and setting weights to either $-a$ or a otherwise. Their approach determines $a > 0$ and Δ by approximately minimizing a ℓ^2-norm between a full-precision matrix and the quantized ternary weight matrix.

Zhu et al. [46] extended their work to ternary weights $w \in \{-a, 0, b\}$ by learning the factors $a > 0$ and $b > 0$ using gradient updates and a different threshold Δ based on the maximum full-precision weight magnitude for each layer. These asymmetric weights considerably

1. Due to finite precision, in fact any DNN is discrete valued. However, we use this term here to highlight the extremely low number of values.

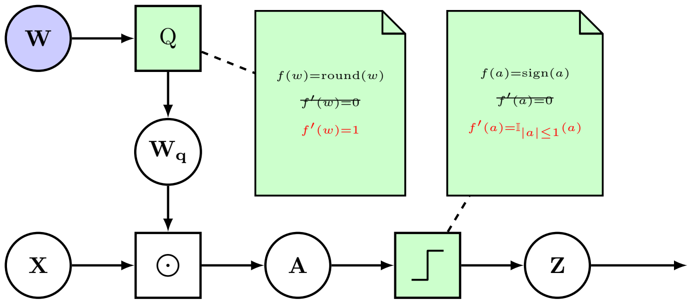


Fig. 2: A simplified building block of a DNN using the straight-through estimator (STE). Real-valued weights $\mathbf{W}$ are quantized with a quantization function Q to obtain quantized weights $\mathbf{W_q}$. These quantized weights are combined with the inputs $\mathbf{X}$ via $\odot$ (e.g. convolution) to obtain activations $\mathbf{A}$ which are subsequently fed into a non-linear activation function to obtain $\mathbf{Z}$. The blue circle indicates the real-valued parameters $\mathbf{W}$ that should be updated with gradient descent, but due to the green boxes the gradient $\nabla_{\mathbf{W}}$ is zero. During backpropagation when the chain-rule is invoked, the STE replaces the zero-derivative by a non-zero surrogate derivative to allow gradient information to flow backwards.

improve performance compared to symmetric weights as used in [45].

Rastegari et al. [47] approximate full-precision weight filters in CNNs as $\mathbf{W} = \alpha\mathbf{B}$ where α is a scalar and $\mathbf{B}$ is a binary weight matrix. This reduces the bulk of floating-point multiplications inside the convolutions to additions or subtractions, and only requires a single multiplication per output neuron with the scalar α. In a further step, the layer inputs are quantized in a similar way to perform the convolution with only efficient XNOR operations and bitcount operations, followed by two floating-point multiplications per output neuron. For backpropagation, the STE is used. Lin et al. [48] generalized the ideas of [47] and approximate the full-precision weights with linear combinations of multiple binary weight filters for improved classification accuracy. Motivated by the fact that weights and activations typically exhibit a non-uniform distribution, Miyashita et al. [49] proposed to quantize values to powers of two. Their representation allows to get rid of expensive multiplications and they report higher robustness to quantization than linear rounding schemes using the same number of bits. While activation binarization methods using the sign function can be seen as approximating commonly used sigmoid functions such as tanh, Cai et al. [50] proposed a half-wave Gaussian quantization that more closely resembles the predominant ReLU activation function. Benoit et al. [51] proposed a quantization scheme that accurately approximates floating point operations using integer arithmetic only to speed-up computation. During training, their forward pass simulates the quantization step to keep the performance of the quantized DNN close to the performance when using single-precision. At test time, weights are represented as 8-bit integer values, reducing the memory footprint by a factor of four.

Zhou et al. [52] presented several quantization schemes that allow for flexible bit widths, both for weights and activations. Furthermore, they also propose a quantization scheme for backpropagation to facilitate low-resource training, and, in agreement with earlier work mentioned above, they note that stochastic quantization is essential for their approach. In [53], weights, activations, weight gradients, and activation gradients are subject to customized quantization schemes that allow for variable bit widths, and that facilitate integer arithmetic during training and testing. In contrast to [52], the work in [53] accumulates weight changes to low-precision weights instead of full-precision weights. While most work on quantization based approaches is empirical, some recent work gained more theoretical insights [54], [55].

2.2.3 Bayesian Approaches

Alternatively to approaches reviewed so far, there exist approaches to train reduced-precision DNNs without any quantization at all. A particularly attractive option for learning discrete-valued DNNs are Bayesian approaches where a distribution over the weights is maintained instead of a fixed weight assignment. This is illustrated in Figure 3. Given a prior $p(\mathbf{W})$ on the weights, a data set $\mathcal{D}$, and a likelihood $p(\mathcal{D}|\mathbf{W})$ that is defined by a DNN, we can use Bayes' rule to infer a posterior distribution over the weights, i.e.

$$p(\mathbf{W}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{W})\ p(\mathbf{W})}{p(\mathcal{D})} \propto p(\mathcal{D}|\mathbf{W})\ p(\mathbf{W}). \qquad (1)$$

From a Bayesian viewpoint, training DNNs can be seen as seeking a point of maximum probability within such a posterior distribution. However, this approach is problematic for discrete-valued DNNs since gradient-based optimization cannot be applied. A solution is to approximate the full posterior $p(\mathbf{W}|\mathcal{D})$ using a tractable variational distribution $q(\mathbf{W})$, and to subsequently use either a maximum of $q(\mathbf{W})$ or to sample an ensemble from it. A common approximation to $q(\mathbf{W})$ assumes independence among the weights – known as mean-field assumption – which renders variational inference tractable. To compute a maximum of $q(\mathbf{W})$ or to sample an ensemble of discrete weight sets is straightforward under this assumption.

Soudry et al. [56] approximate the true posterior $p(\mathbf{W}|\mathcal{D})$ using expectation propagation [57] in an online fashion with closed-form updates. Starting with an uninformative approximation $q(\mathbf{W})$, their approach combines the current approximation $q(\mathbf{W})$ (serving as the prior in (1)) with the likelihood for a data set $\mathcal{D} = \{(\mathbf{x}, c)\}$ comprising only a single sample to obtain a refined posterior. Since this refinement step is not available in closed-form, they propose several approximations in order to yield a more amenable objective. A different Bayesian approach for discrete-valued weights has been

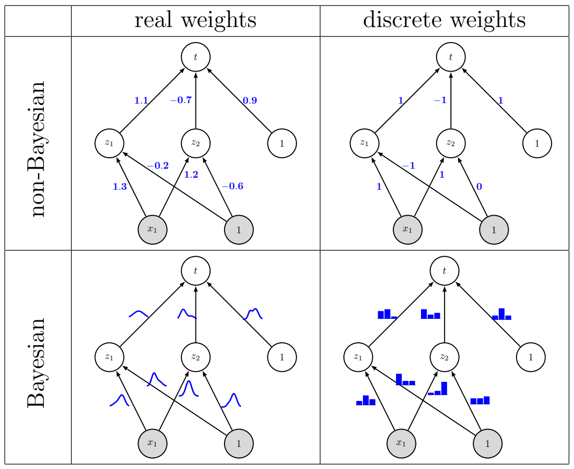


Fig. 3: Overview of real-valued vs. discrete-valued NNs and Bayesian vs. non-Bayesian NNs. The aim is to obtain a single discrete-valued NN (top right) with a good performance. In the Bayesian approach this is achieved by training a distribution over discrete-valued NNs (bottom right) and subsequently deriving a single discrete-valued NN from that distribution.

presented by Roth and Pernkopf [58]. They approximate the posterior $p(\mathbf{W}|\mathcal{D})$ by minimizing the Kullback-Leibler (KL)-divergence $KL(q(\mathbf{W})||p(\mathbf{W}|\mathcal{D}))$ with respect to the parameters $\boldsymbol{\nu}$ of the approximation $q(\mathbf{W})$. The KL-divergence is commonly decomposed as

$$\begin{aligned} KL(q(\mathbf{W})||p(\mathbf{W}|\mathcal{D})) &= KL(q(\mathbf{W})||p(\mathbf{W})) \\ &- \mathbb{E}_{q(\mathbf{W})}[\log p(\mathcal{D}|\mathbf{W})] + \log p(\mathcal{D}). \end{aligned} \tag{2}$$

This expression does not involve the intractable posterior $p(\mathbf{W}|\mathcal{D})$ and the evidence $\log p(\mathcal{D})$ is constant with respect to $\boldsymbol{\nu}$. The KL term can be seen as a regularizer that pulls the approximate posterior $q(\mathbf{W})$ towards the prior $p(\mathbf{W})$ whereas the expected log-likelihood captures the data. The expected log-likelihood in (2) is intractable, but it can be optimized using the so-called reparameterization trick and stochastic optimization [59], [60], [12]. In [58], it has been proposed to optimize an approximation to (2) using similar techniques as in [56]. Compared to directly optimizing in the discrete weight space, this approach has the advantage that the real-valued parameters $\boldsymbol{\nu}$ of the approximation $q(\mathbf{W})$ can be optimized with gradient-based techniques. In particular, they trained feed-forward DNNs using sign activations with 3-bit weights in the first layer and ternary weights in the remaining layers and achieved results that are on par with results obtained using the STE [44].

3 Probabilistic Models

Efficiency, as discussed so far, is clearly one of the main challenges for machine learning systems in real-world applications. However, in the evolution of machine learning techniques we are facing another challenge which must not be underestimated: uncertainty. The further we move machine learning towards "the wild", the more circumstances get out of our control. Consequently, any machine learning system which shall realistically be applied in real-world applications, must to some degree facilitate a mechanism to treat uncertainty in all aspects, i.e. inputs, internal states, the environment and outputs.

To this end, probabilistic methods [61], [62] are arguably the method of choice when it comes to reasoning under uncertainty. The classical probabilistic models are PGMs, which represent variables in data sets as nodes in a graph and direct variable dependencies as edges between the nodes. PGMs are a naturally resource-efficient representation of a probability distribution, especially when sparse model structures are used [63]. Additionally, their memory footprint can be reduced when considering reduced-precision parameters [64], [65].

Compared to DNNs the prediction quality of PGMs is usually inferior. Due to this fact, discriminative and hybrid learning paradigms have been proposed to substantially improve the prediction performance. On the other hand, probabilistic methods are often significantly more data efficient than deep learning techniques, in particular when rich domain structure can be exploited [66], [67]. In this article, we focus on learning *directed* PGMs also called *Bayesian networks* (BNs) [68], [62], while efficient implementations of undirected graphical models have been considered in [69].

3.1 Learning Bayesian Networks

A Bayesian network (BN) $\mathcal{B} = (\mathcal{G}, \mathcal{P}_{\mathcal{G}})$ describes the dependencies of a set of random variables $\mathbf{X} = [X_0, \dots, X_L]$ by a directed graph $\mathcal{G}$, i.e. the *structure* of the BN, and a set of conditional probability distributions $\mathcal{P}_{\mathcal{G}}$ associated with the nodes of $\mathcal{G}$, i.e. the *parameters*. The i^{th} node of $\mathcal{G}$ corresponds to the random variable X_i and the edges in $\mathcal{G}$ encode conditional independence properties between the random variables. For each X_i there is a conditional probability distribution $p(X_i \mid Pa(X_i))$, where $Pa(X_i)$ denotes the parents of variable X_i according to $\mathcal{G}$. The BN defines a probability distribution over $\mathbf{X}$ as $p^{\mathcal{B}}(\mathbf{X}) = \prod_{i=0}^{L} p(X_i \mid Pa(X_i))$. When using BNs for classification, one of the random variables $\mathbf{X}$ takes the special role of the class variable, yielding a joint distribution $p^{\mathcal{B}}(C, \mathbf{X})$ for $p^{\mathcal{B}}(C, X_1, \dots, X_L)$.

Both parameters and structure can be learned using either generative or discriminative objectives [63]. The inference performance (i.e. the prediction error) of BNs can in general be boosted when discriminative learning paradigms are used. In the generative approach, we exploit the parameter posterior, yielding the maximum-a-posterior (MAP), maximum likelihood (ML) or the Bayesian approach. In discriminative learning, alternative objective functions are considered, such as conditional log-likelihood (CLL) [70], [71], [72], classification rate (CR), or margin [73], [74], which can be applied for both structure learning and for parameter learning [63], [75]. Furthermore, hybrid parameter learning has been proposed unifying the generative and discriminative learning paradigms [76], [77], [78], [79], and combine their respective advantages (allowing, e.g. to consistently treat missing data).

3.1.1 Parameter Learning

The conditional probability densities (CPDs) of BNs are usually of some parametric form, which can be optimized either generatively or discriminatively. Several approaches to optimize BN parameters are discussed in the following.

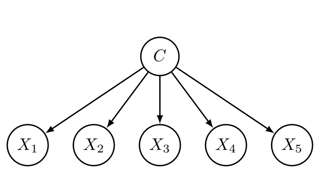
- **Generative Parameters.** In generative parameter learning the goal is to capture the generative process generating the available data, i.e. generative parameters are based on the idea of *approximating* the true underlying data distribution with a distribution $p^{\mathcal{B}}(C, \mathbf{X})$. An example of this paradigm is *maximum-likelihood* learning, i.e. optimizing the likelihood
$$\mathcal{P}_{\mathcal{G}}^{\mathrm{ML}} = \arg\max_{\mathcal{P}_{\mathcal{G}}} \prod_{n=1}^{N} p^{\mathcal{B}}(c^{(n)}, \mathbf{x}^{(n)}), \tag{3}$$
where $c^{(n)}$ and $\mathbf{x}^{(n)}$ are the instantiations of C and $X_1, \ldots, X_L$ for the n^{th} training data point respectively. Maximum likelihood parameters minimize the KL-divergence between $p^{\mathcal{B}}(C, \mathbf{X})$ and empirical data distribution, and thus, under mild assumptions, the KL-divergence between $p^{\mathcal{B}}(C, \mathbf{X})$ and the true distribution [68].
- **Discriminative Parameters.** In discriminative learning one is interested in parameters yielding good classification performance on new samples from the data distribution. Discriminative learning is especially advantageous when the assumed model distribution $p^{\mathcal{B}}(C, \mathbf{X})$ cannot capture the underlying data distribution well, as for example when rather limited BN structures are used [80]. Several objectives for discriminative parameter learning have been proposed. Here, we consider the *maximum-conditional-likelihood* (MCL) objective [80] and the *maximum-margin* (MM) objective [73], [74]. MCL parameters are obtained by maximizing
$$\mathcal{P}_{\mathcal{G}}^{\mathrm{MCL}} = \arg\max_{\mathcal{P}_{\mathcal{G}}} \prod_{n=1}^{N} p^{\mathcal{B}}(c^{(n)} | \mathbf{x}^{(n)}), \tag{4}$$
where $p^{\mathcal{B}}(C|\mathbf{X})$ denotes the conditional distribution of C given as
$$p^{\mathcal{B}}(C|\mathbf{X}) = \frac{p^{\mathcal{B}}(C, \mathbf{X})}{p^{\mathcal{B}}(\mathbf{X})}. \tag{5}$$
MM parameters $\mathcal{P}_{\mathcal{G}}^{\mathrm{MM}}$ are obtained by maximizing
$$\mathcal{P}_{\mathcal{G}}^{\mathrm{MM}} = \arg\max_{\mathcal{P}_{\mathcal{G}}} \prod_{n=1}^{N} \min\left(\gamma, d^{\mathcal{B}}(c^{(n)}, \mathbf{x}^{(n)})\right), \tag{6}$$
where $d^{\mathcal{B}}(c^{(n)}, \mathbf{x}^{(n)})$ is the *probabilistic margin* of the n^{th} sample, given as
$$d^{\mathcal{B}}(c^{(n)}, \mathbf{x}^{(n)}) = \frac{p^{\mathcal{B}}(c^{(n)} | \mathbf{x}^{(n)})}{\max_{c \neq c^{(n)}} p^{\mathcal{B}}(c | \mathbf{x}^{(n)})}. \tag{7}$$
The margin can be interpreted as the model's confidence that the n^{th} sample corresponds to the groundtruth class $c^{(n)}$. In particular, the sample is correctly classified when $d^{\mathcal{B}}(c^{(n)}, \mathbf{x}^{(n)}) > 1$. Thus, the MM objective stimulates low classification error, while well calibrated class posteriors are deliberately not enforced. In order to avoid that the model just optimizes the margins of a few samples, the sample-wise margin terms are capped by a hyper parameter γ, which is typically cross-tuned.
An alternative and simple method for learning discriminative parameters are *discriminative frequency estimates* [81]. According to this method, parameters are estimated using a perceptron-like algorithm, where parameters are updated by the prediction loss, i.e. the difference of the class posterior of the correct class (which is assumed to be 1 for the data in the training set) and the class posterior according to the model using the current parameters. This type of parameter learning yields classification results comparable to those obtained by MCL [81].
- **Hybrid Parameters.** Furthermore hybrid parameter learning which combines generative and discriminative objectives has been considered [78]. Hybrid parameters often achieve good prediction performance (due to the discriminative objective) while at the same time maintaining a generative character of the model, which is e.g. beneficial under missing input features.

3.1.2 Structure Learning

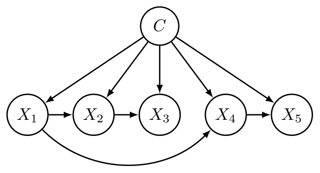
The structure of a BN classifier, i.e. its graph $\mathcal{G}$, encodes conditional independence assumptions. Structure learning is naturally cast as a combinatorial optimization problem and is in general difficult — even in the case where scores of structures decompose according to the network structure [82]. For the generative case, several formal hardness results are available, e.g. learning polytrees [83] or learning general BNs [84] are NP-hard optimization problems. Algorithms for learning generative structures often optimize some kind of penalized likelihood of the training data and try to determine the structure for example by performing independence tests [85]. Discriminative methods often employ local search heuristics [86], [87], [88].

A good overview over different BN structures is provided in [68]. Here, we focus on relatively simple structures, i.e. the naive Bayes (NB) structure and tree augmented network (TAN) structures.

- *Naive Bayes (NB).* This structure implies conditional independence of the features, given the class, cf. Figure 4a. Obviously, this conditional independence assumption is often violated in practice. Still, NB often yields impressively good performance in many applications [89].
- *Tree Augmented Networks (TAN).* This structure was introduced in [85] to relax the strong independence



(a) Naive Bayes structure



(b) TAN structure

Fig. 4: Exemplary BN structures used for classification.

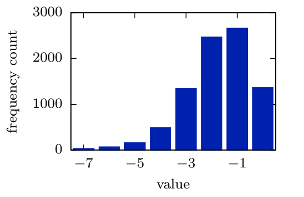
assumptions imposed by the NB structure and to enable better classification performance. In particular, each attribute may have at most one other attribute as an additional parent. An example of a TAN structure is shown in Figure 4b. TAN structures can be learned using generative and discriminative objectives [86], [87].

The reason for not using more expressive structures is mainly that more complex structures do not necessarily result in significantly better classification performance [86] while leading to models with many more parameters.

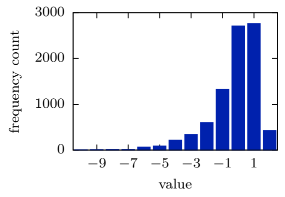
3.2 Reduced Precision in Bayesian Networks

Results from sensitivity analysis indicate that PGMs are well suited for low bit-widths implementations because they are not sensitive to parameter deviations under the following two conditions [90]. Firstly, if the conditional probabilities are not too extreme, i.e. close to zero or one, and, secondly, if the posterior probabilities for different classes are significantly different. Additionally, this is supported by empirical classification results for PGMs with reduced-precision parameters [64], [65], [91] which we present in more detail in the following.

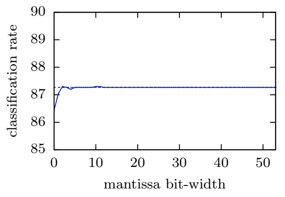
An important observation for the development of reduced-precision BNs is that for prediction it is sufficient to compute the product of conditional probabilities of the variables in the Markov blanket of the class variable C [68]. Equivalently, this corresponds to the computation of a sum of log-probabilities which can often be very efficiently implemented. Furthermore, storing log-probabilities makes it easy to satisfy the condition from sensitivity analysis that it is important to be relatively accurate about small probabilities. The required precision for storing reduced-precision floating point numbers depends on the range of values which the parameters of the BN assume, hence it is instructive to look at the histograms of log-probabilities for a BN trained for classifying handwritten digits. Such histograms are shown in Figures 5a and 5b. The log-probabilities assume only a small range of values, and considering the exponent of the corresponding (normalized) floating-point representation, we observe that there is only a small tail of large (negative) exponents, i.e. small probabilities. This indicates, following the results of sensitivity analysis outlined above, that quantizing the



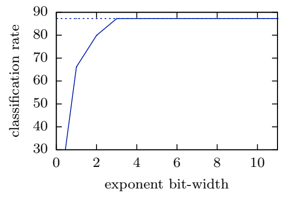
(a) logarithmic probabilities



(b) exponent of logarithmic conditional probabilities in double-precision



(c) varying mantissa bit width, using full bit-width for exponent



(d) varying exponent bit width, using full bit width for mantissa

Fig. 5: Top row: Histograms of (a) the log-parameters, and (b) the exponents of the log-parameters of a BN classifier for handwritten digit data with ML parameters assuming NB structure. Bottom row: Classification rates for varying bit widths of (a) the mantissa, and (b) the exponent, for handwritten digit data, NB structure, and log ML parameters. The classification rates using full double-precision logarithmic parameters are indicated by the horizontal dotted lines.

log-probabilities should not reduce classification performance significantly. Indeed, as illustrated in Figures 5c and 5d the performance of BNs with reduced-precision floating point numbers quickly reaches the performance of BNs with full-precision parameters with increasing parameter precision.

These illustrative results are for BNs with generatively optimized parameters, i.e. maximum-likelihood parameters. This does not necessarily imply that BNs with discriminatively optimized parameters are also well-suited for reduced-precision parameters as discriminative parameters are in general more extreme, i.e. closer to zero or one. However, Tschiatschek et al. [92] conducted an exhaustive evaluation of BNs with reduced-precision floating point parameters comparing BNs with generatively and discriminatively optimized parameters for the case in which these parameters are first estimated using full-precision floating point numbers and subsequently quantized to some desired (reduced) precision. Their results indicate that BNs with discriminatively optimized parameters are almost as robust to precision reduction as BN classifiers with generatively optimized parameters. Furthermore, even large precision reduction does not decrease classification performance significantly. In general a mantissa with only 4 bits and a 5 bit exponent are sufficient to achieve close-to-optimal performance. These findings are consistent among a large set of diverse data sets and BN structures.

3.2.1 Learning Optimal Reduced-Precision Parameters

Reduced-precision BNs for classification achieve remarkable performance when these parameters are obtained by rounding full-precision parameters. Nevertheless, a natural question that arises is whether improved performance can be achieved by learning parameters that are tailored for reduced precision. This question was affirmatively studied in [93], [65]. The authors proposed a branch-and-bound algorithm for finding globally optimal discriminative fixed-precision parameters. The resulting parameters have superior classification performance compared to parameters obtained by simple rounding of double-precision parameters, particularly for very low number of bits, cf. Section 4.4. Again, these findings are consistent among a large set of diverse data sets and BN structures [93], [65].

3.2.2 Online Learning in Reduced Precision

While in many applications suitable reduced-precision parameters for BNs can be precomputed using the techniques outlined in the previous section, there are applications requiring to learn parameters *within* the application, i.e. on a system supporting only reduced-precision computations. Examples include applications requiring fine-tuning of parameters for domain adaptation or adaptation of parameters to user preferences. Thus it is important to enable learning reduced-precision parameters for BNs using reduced-precision computations only. In [93], this setting was investigated. The authors propose algorithms for learning ML parameters and for learning MM parameters.

The algorithms are developed for the *online setting*, i.e. when parameters are updated on a per-sample basis. In this setting, learning using reduced-precision computations requires specialized algorithms because gradient-descent (or gradient-ascent) procedures using reduced-precision arithmetic typically do not perform well. The problem is resolved by using precomputed lookup tables of small sizes for log-parameters which can be efficiently indexed by keeping and (on overflows) scaling feature counts. The resulting algorithms have very low computational demands, mainly requiring counters and a little memory for storing the lookup tables. At the same time the proposed algorithms yield parameters with close-to-optimal performance while only having slightly slower convergence than comparable algorithms using full-precision arithmetic.

4 Experimental Results

In this section, we first exemplify the trade-off between model performance, memory footprint and computation time on the CIFAR-10 classification task in Section 4.1. This example highlights that finding a suitable balance between these requirements remains challenging due to diverse hardware and implementation issues. Furthermore, we provide an extensive comparison between a rich collection of hardware-efficient approaches discussed in this paper, on the challenging task of ImageNet classification in Section 4.2. In Section 4.3 we present a real-world speech enhancement example, where hardware-efficient BNNs have led to dramatic memory and computation time reductions. Section 4.4 shows exemplary results comparing PGMs and DNNs on the classical MNIST data set. The focus here is on prediction performance and the number of bits necessary to represent the models. We conclude the experimental section with an example of randomly missing features during model testing (see Section 4.5). Such scenarios can be easily treated with probabilistic models.

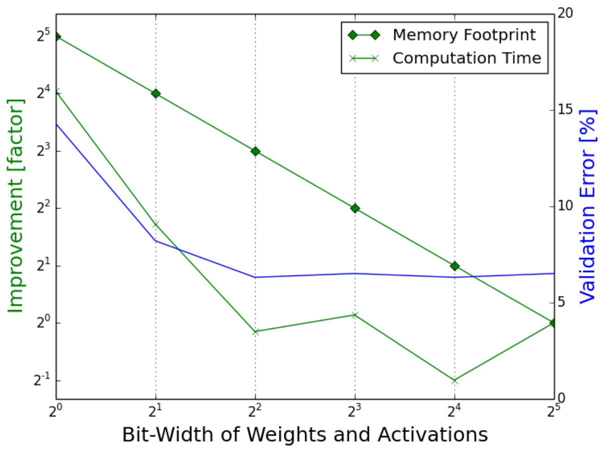


Fig. 6: Improvement of reduced precision over single-precision floating point on memory footprint and computation time (green) and the respective validation error of ResNet-32 on CIFAR-10 (blue).

4.1 Prediction Accuracy, Memory Footprint and Computation Time Trade-Off

To exemplify the trade-off to be made between memory footprint, computation time and prediction accuracy, we implemented general matrix multiply (GEMM) with variable-length fixed-point representation on a mobile CPU (ARM Cortex A15), exploiting its NEON SIMD instructions. Using this implementation, we ran a 32-layer ResNet NN with custom quantization on weights and activations representation [52] and compare these results with single-precision floating point. We use the CIFAR-10 data set, containing color images of 10 object classes (airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks).

Figure 6 reports the impact of reduced precision on runtime, memory requirements and classification accuracy, averaged over the test set. As can be seen, reducing the bit width to 16, 8 or 4 bits does not improve runtimes and even is harmful in the case of 4 bits. The reason for this behavior is that our implementation uses bit-width doubling for these precisions in order to ensure correct GEMM computations. Since bit widths of 2 and 1 do not require bit-width doubling, we obtain runtimes close to the theoretical linear speed-ups. In terms of memory footprint, our implementation evidently reaches the theoretical linear improvement. While reducing the bit width of weights and activations to only 1 or 2 bits improves memory footprint and computation time significantly, these settings also show decreased performance. In this example, the sweet spot appears to be 2 bit precision, but also the predictive performance for 1 bit precision might be acceptable for some applications. This extreme setting is evidently beneficial for highly constrained scenarios and is easily exploited on today's hardware, as shown in the following section.

4.1.1 Computation Savings for BNNs

In order to show that the advantages of binary computation translate to other general-purpose processors, we implemented matrix-multiplication operators for NVIDIA GPUs and ARM CPUs. Classification in BNNs can be implemented very efficiently as 1-bit scalar products, i.e. multiplications of two vectors $\mathbf{x}$ and $\mathbf{y}$ of length N reduce to bit-wise *xnor()* operation, followed by counting the number of set bits with *popc()*:

$$\mathbf{x} \cdot \mathbf{y} = N - 2 * popc(xnor(\mathbf{x}, \mathbf{y})), x_i, y_i \in [-1, +1] \quad \forall i. \tag{8}$$

We use the matrix-multiplication algorithms of the MAGMA and Eigen libraries and replace float multiplications by *xnor()* operations, as depicted in Equation (8). Our CPU implementation uses NEON vectorization in order to fully exploit SIMD instructions on ARM processors. We report execution time of GPUs and ARM CPUs in Table 1. As can be seen, binary arithmetic offers considerable speed-ups over single-precision with manageable implementation effort. This also affects energy consumption since binary values require less off-chip accesses and operations. Performance results of x86 architectures are not reported because neither SSE nor AVX ISA extensions support vectorized *popc()*.

TABLE 1: Performance metrics for matrix · matrix multiplications on a NVIDIA Tesla K80 and ARM Cortex-A57.

arch	matrix size	time (float32)	time (binary)	**speed-up**
GPU	256	0.14ms	0.05ms	**2.8**
GPU	513	0.34ms	0.06ms	**5.7**
GPU	1024	1.71ms	0.16ms	**10.7**
GPU	2048	12.87ms	1.01ms	**12.7**
ARM	256	3.65ms	0.42ms	**8.7**
ARM	513	16.73ms	1.43ms	**11.7**
ARM	1024	108.94ms	8.13ms	**13.4**
ARM	2048	771.33ms	58.81ms	**13.1**

While improvements of memory footprint and computation time are independent of the underlying tasks, the prediction accuracy highly depends on the complexity of the data set and the used neural network. Simple data sets such as MNIST, allow for aggressive quantization without affecting prediction performance significantly, while binary/ternary quantization results in severe prediction degradation on more complex data sets, such as ImageNet.

4.2 Resource-Efficient DNNs on ImageNet

We compare the performance of different quantization strategies on the example of AlexNet [1] on the ImageNet ILSVRC-2012 data set [94]. Since 2010, ImageNet is the data set for the annual competition called the Large-Scale Visual Recognition Challenge (ILSVRC). ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories, comprising roughly 1.2M training and 50k validation images with high resolution. The ILSVRC is considered to be one of the most challenging data sets for DNNs and, consequently, for quantization. It is common practice to report two prediction-accuracy rates: Top-1 and Top-5 accuracy, where Top-5 is the fraction of test images for which the correct label is among the five labes. Table 2 reports the accuracy gap (Top-1 and Top-5) between single-precision floating point and the respective quantization approach.

First, we compare several strategies that quantize weights of a DNN on the basis of the Top-1 accuracy gap. Deep compression (DC) [8] effectively reduces weights (using weight sharing) to 8 bit (convolutional layers) and 5 bit (fully-connected layers) in order to obtain full-precision prediction accuracy. Binarization of weights was first introduced by binary connect (BC) [43] with about a -21.2% Top-1 accuracy gap. The introduction of scaling coefficients by XNOR-Net [47] outperformed BC by a large margin with prediction performance close to full-precision weights. Quantizing weights to a ternary representation is superior to a binary representation on large-scale data sets. TWN [45] reduced the gap to full precision AlexNet to only -2.7% and TTQ [46] even outperformed full-precision by 0.3%. SYQ [95] further improves ternary quantization by using pixel-wise instead of layer-wise scaling coefficients. At the cost at a higher memory footprint, they are able to outperform Top-1 and Top-5 prediction performance of single-precision floating point by 1.5% and 0.6% respectively.

Whereas binarization of weights works well on AlexNet, binarization of activations shows severe performance degradation (-28.7% and -12.4% Top-1 accuracy gap for BNN [44] and XNOR-Net [47], respectively). QNNs [96] and HWGQ [50] tackle this problem by using more bits for activations while binarizing weights: for instance, using 2-bit activations decreases the Top-1 gap to -5.6% for QNN and -5.8% for HWGQ. TSQ [97] further improves the approach of HWGQ and achieves -0.5% Top-1 gap (with ternary weights and 2-bit activations). SYQ and DeepChip [98] require 8-bit fixed point in order to maintain full-precision accuracy.

4.3 A Real-World Example: Speech Mask Estimation using Reduced-Precision DNNs

We provide a complete example employing hardware-efficient BNNs applied to acoustic beamforming, an important component for various speech enhancement systems. A particularly successful approach employs DNNs to estimate a speech mask, i.e. a speech presence

TABLE 2: Accuracy gap between single-precision floating point and different state-of-the-art quantization approaches of AlexNet on ImageNet for different bit-width combinations of Activations (A) and Weights (W).

A-W	Gap	DC	BC	BNN	XNOR	DoReFa	TWN	TTQ	QNN	HWGQ	SYQ	TSQ	DeepChip
32-32	Top-1	57.2	56.6	56.6	56.6	57.2	57.2	57.2	56.6	58.5	56.6	58.5	56.2
	Top-5	80.3	80.3	80.2	80.2	80.3	80.3	80.3	80.2	81.5	80.2	81.5	78.3
32-8/5	Top-1	0.0	–	–	–	–	–	–	–	–	–	–	–
	Top-5	0.0	–	–	–	–	–	–	–	–	–	–	–
32-2	Top-1	–	–	–	–	–	-2.7	+0.3	–	–	–	–	–
	Top-5	–	–	–	–	–	-3.5	-0.6	–	–	–	–	–
32-1	Top-1	–	-21.2	–	+0.2	–	–	–	–	–	–	–	–
	Top-5	–	-19.3	–	-0.8	–	–	–	–	–	–	–	–
8-2	Top-1	–	–	–	–	–	–	–	–	–	+1.5	–	+0.2
	Top-5	–	–	–	–	–	–	–	–	–	+0.6	–	+0.7
2-2	Top-1	–	–	–	–	–	–	–	–	–	-0.8	-0.5	–
	Top-5	–	–	–	–	–	–	–	–	–	-1.0	-1.0	–
2-1	Top-1	–	–	–	–	–	–	–	-5.6	-5.8	-1.2	–	–
	Top-5	–	–	–	–	–	–	–	-6.5	-5.2	-1.6	–	–
1-1	Top-1	–	–	-28.7	-12.4	-11.8	–	–	–	–	–	–	–
	Top-5	–	–	-29.8	-11.0	-11.0	–	–	–	–	–	–	–

probability of each time-frequency cell. This speech mask is used to determine the power spectral density (PSD) matrices of the multi-channel speech and noise signals, which are subsequently used to obtain a beamforming filter such as the minimum variance distortionless response (MVDR) beamformer or generalized Eigenvector (GEV) beamformer [99], [100], [101], [102], [103], [104]. An overview of a multi-channel speech enhancement setup is shown in Figure 7.

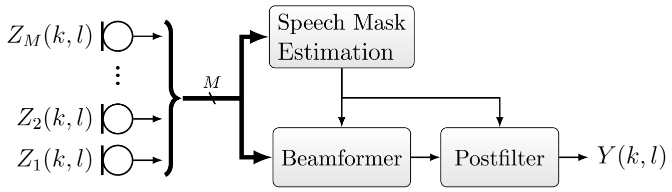


Fig. 7: System overview, showing the microphone signals $Z_m(k,l)$ and the beamformer+postfilter output $Y(k,l)$ in frequency domain.

In this experiment, we compare single-precision DNNs and BNNs trained with STE [44] for the estimation of the speech mask. For both architectures, the dominant Eigenvector of the noisy speech PSD matrix [104] is used as feature vector, where it is quantized to 8 bit integer values for the BNN. As output layer, a linear activation function is used, which reduces to counting the binary neuron outputs, followed by normalization to yield the speech presence probability mask $\mathrm{p}_{SPP} \in [0,1]$. Further details of the experimental setting can be found in [105].

4.3.1 Data and Experimental Setup

For evaluation we used the CHiME corpus [106] which provides 2 and 6-channel recordings of a close-talking speaker corrupted by four different types of ambient noise. Ground truth utterances (i.e. the separated speech and noise signals) are available for all recordings, such that the ground truth speech masks $\mathrm{p}_{SPP,opt}(k,l)$ at time l and frequency bin k can be computed. In the test phase, the DNN is used to predict $\hat{\mathrm{p}}_{SPP}(k,l)$ for each utterance, used to estimate the corresponding beam-former. A single-precision 3-layer DNN with 513 neurons per layer and BNNs with 513 and 1024 neurons per layer are used. The DNNs were trained using ADAM [107] with default parameters and a dropout probability of 0.25.

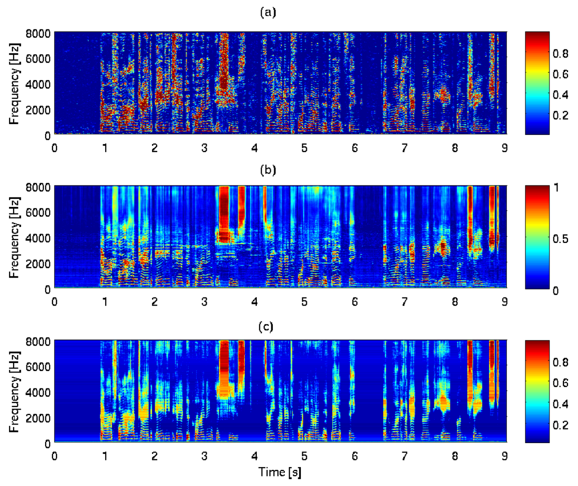


Fig. 8: Speech presence probability mask: (a) Optimal speech mask $\mathrm{p}_{SPP,opt}(k,l)$; (b) Prediction of $\hat{\mathrm{p}}_{SPP}(k,l)$ using DNNs with 513 neurons/layer; (c) Prediction of $\hat{\mathrm{p}}_{SPP}(k,l)$ using BNNs with 1024 neurons/layer.

4.3.2 Speech Mask Accuracy

Figure 8 shows the optimal and predicted speech masks of the DNN and BNN for an example utterance (F01_22HC010W_BUS). We see that both methods yield very similar results and are in good agreement with the ground truth. Table 3 reports the prediction error $\mathcal{L} = \frac{100}{KL}\sum_{k=1}^{K}\sum_{l=1}^{L}\left|\hat{\mathrm{p}}_{SPP}(k,l) - \mathrm{p}_{SPP,opt}(k,l)\right|$ in [%]. Although single-precision DNNs achieved the best prediction error on the test set, they do so only by a small margin. Doubling the networks size of BNNs slightly improved the error on the test set for the case of 6 channels.

TABLE 3: Mask prediction error $\mathcal{L}$ in [%] for DNNs with 513 neurons/layer and BNNs with 513 or 1024 neurons/layer.

model	neurons / layer	channels	**train**	**valid**	**test**
DNN	513	2ch	5.8	6.2	7.7
BNN	513	2ch	6.2	6.2	7.9
BNN	1024	2ch	6.2	6.6	7.9
DNN	513	6ch	4.5	3.9	4.0
BNN	513	6ch	4.7	4.1	4.4
BNN	1024	6ch	4.9	4.2	4.1

4.3.3 *Perceptual Audio Quality*

Given the predicted speech mask $\hat{\mathrm{p}}_{SPP}(k,l)$, we construct the GEV-PAN beamformer [108] for both the 2 and 6-channel data. The overall perceptual score (OPS) [109] is used to evaluate the performance of the resulting speech signal $Y(k,l)$ in terms of perceptual speech quality. Ground truth estimates required for these scores are obtained using the $\mathrm{p}_{SPP,opt}(k,l)$ and the GEV-PAN.

Table 4 reports the OPS given the enhanced utterances of the GEV-PAN beamformer. GEV-PAN outperforms the CHiME4-baseline enhancement system, i.e. the BeamformIt!-toolkit [106], and the front-end of the best CHiME3 system [110], i.e. CGMM-EM. Doubling the network size of BNNs mostly improves the OPS scores. In general, BNNs achieve on average only a slightly lower OPS score than the single-precision DNN baseline.

TABLE 4: Overall perceptual score (OPS) for various beamformers (BeamformIt, GEV-PAN, MVDR) using DNNs and BNNs for speech mask estimation.

method	set	**train**	**valid**	**test**
CHiME4 baseline	simu	33.11	34.73	31.46
(BeamformIt), 5ch [106]	real	29.97	36.45	36.74
CGMM-EM with MVDR	simu	52.15	43.02	40.59
and postfilter, 6ch [110]	real	44.95	41.89	36.87
DNN (513 neurons / layer)	simu	**64.21**	**61.74**	**56.32**
with GEV-PAN, 2ch	real	**64.21**	**62.72**	**56.32**
BNN (513 neurons / layer)	simu	58.11	57.58	57.58
with GEV-PAN, 2ch	real	56.79	57.52	41.24
BNN (1024 neurons / layer)	simu	**61.64**	**60.78**	**54.20**
with GEV-PAN, 2ch	real	**61.64**	**60.78**	**45.22**
DNN (513 neurons / layer) ,	simu	**67.98**	**66.76**	**68.71**
with GEV-PAN 6ch	real	**69.98**	**70.33**	**63.28**
BNN (513 neurons / layer) with	simu	61.44	55.87	62.39
with GEV-PAN, 6ch	real	63.03	64.77	64.52
BNN (1024 neurons / layer)	simu	**65.59**	**64.98**	**68.41**
with GEV-PAN, 6ch	real	**67.91**	**68.41**	**59.94**

4.4 Resource-efficient DNNs and PGMs on MNIST

While PGMs with sparse structures such as NB or TAN are usually computationally efficient and have a small memory footprint, they often do not achieve the same prediction performance as DNNs. However, PGMs have advantages in important settings for applying machine learning in "the wild", e.g. when a considerable number of input features is missing. Both modeling approaches have rich capabilities for machine learning on embedded devices. Here, the classification performance of both reduced-precision PGMs and DNNs is compared on the MNIST data.

4.4.1 *Data*

The MNIST data set for handwritten digit recognition [111] contains 60000 training images and 10000 test images of size 28×28 with gray-scale values. Some samples from the data set are shown in Figure 9. For DNNs the training set is further split into 50000 training samples and 10000 validation samples. Each pixel is treated as feature, i.e. $\mathbf{x} \in \mathbb{R}^{784}$. For PGMs and *small-size* DNNs the data is down-sampled by a factor of two, resulting in a resolution of 14×14 pixels, i.e. $\mathbf{x} \in \mathbb{R}^{196}$.

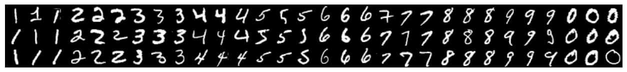


Fig. 9: Samples of MNIST.

4.4.2 *Results*

We report the performance of reduced-precision DNNs with sign activations using variational inference (NN VI) [58] and using the STE (NN STE) [44]. As a baseline, we compare with real-valued (32 bit) DNNs (NN real) trained with batch normalization [112], dropout [113], and ReLU activations. A three layer structure with $1200 - 1200$ hidden units is selected. Several hyperparameters for all methods were tuned using 50 iterations of Bayesian optimization [114] on a separate held-out validation set. For NN VI, we used 3-bit weights for the input layer and ternary weights $w \in \{-1, 0, 1\}$ for the remaining layers. During training, dropout was used to regularize the model. Results for the most probable model from the approximate posterior $\mathbf{W} = \arg\max_{\mathbf{W}} q(\mathbf{W})$ are reported. For NN STE, the weights in the input layer were quantized to 3-bit weights as above, and the weights of the remaining layers were always quantized to binary values. In addition to dropout, NN STE also uses batch normalization which appears to be a crucial component here. Although batch normalization requires real-valued parameters, it merely results in a shift of the sign activation function that introduces only a marginal computational overhead at test time [115].

We contrast the results for the reduced-precision DNNs with those of BNs. In particular, the NB structure and MM parameter learning has been used (see Section 3).

The classification errors (CE) [%], the model size (#Param) [kbits] and the model configuration for the DNNs and BN are shown in Table 5. NN STE (3-bit) performs on par with NN VI (3-bit) while NN (real) with a three layer structure (i.e. 1200-1200 neurons) slightly outperforms both. The classification performance of the BN is worse compared to DNNs. However, the achieved performance is impressive considering that only $6{,}720$

parameters (each represented by 6-bits) are used[2], i.e. the BN is a factor of 60, 120, 1900 smaller than NN STE, NN VI and NN real using a 3 layer structure with $1200-1200$ hidden units, respectively.

Moreover, we scaled the NN STE down to about the same model size of 40 kbits as the BN with NB structure and MM parameters using the down-sampled MNIST data (BN NB MM). Results show that NN STE slightly outperforms the BN when using batch normalization. Furthermore, NN STEs have one hidden layer, while the BN is *shallow* which might explain some of the performance gain. Better performance with BNs can be achieved with more expressive structures, e.g. a BN classifier with TAN-MM structure has $31{,}399$ parameters and achieves a classification error of about $4{,}75\%$ [87]. Furthermore, classification in BNs is computationally extremely simple – just the joint probability $p^{\mathcal{B}}(C, \mathbf{X})$ has to be computed. This amounts to summing up the log conditional probabilities for each feature X_i. These results suggest that BNs enable a good trade-off between computational requirements for inference, memory demands and prediction performance. Additionally, they are advantageous in case of missing input features. This is shown in Section 4.5.

TABLE 5: Classification errors (CE) [%], model size (#Param) [kbits] and model configuration for different NN models and BNs. NN real: Real-valued DNNs; NN STE: DNNs with 1 or 3 bit weights in the first layer, binary weights in the remaining layers. NN VI: DNNs with 1 or 3 bit weights in the first layer, ternary weights in the remaining layers. BN: Bayesian network with naive Bayes (NB) structure and MM parameters using 6-bit parameters.

Classifier	CE [%]	(#Param) [kbits]	input size	input layer	batch norm	layers/ neurons
NN real	0.87	76 800	28×28	32-bit	yes	1200-1200
NN STE	1.24	2 550	28×28	3-bit	yes	1200-1200
NN VI	1.28	4 790	28×28	3-bit	no	1200-1200
BN NB MM	6.72	40	14×14	-	-	-
NN STE	4.25	40	14×14	3-bit	yes	65
NN STE	7.82	40	14×14	3-bit	no	65
NN STE	3.72	40	14×14	1-bit	yes	193
NN STE	6.99	40	14×14	1-bit	no	193

The classification results for BN NB MM over various bit width is shown in Figure 10. In particular, we show results for full-precision floating point parameters, reduced-precision fixed-point parameters obtained by rounding and optimal reduced-precision fixed-point parameters obtained by the algorithm outlined in Section 3.2.1. The performance of the reduced-precision parameters quickly approaches that of full-precision parameters with an increasing number of bits. The optimal reduced-precision parameters achieve improved performance over the reduced-precision parameters obtained by rounding, in particular for low numbers of bits.

2. After discretizing input features and the removal of features with constant values across the data set.

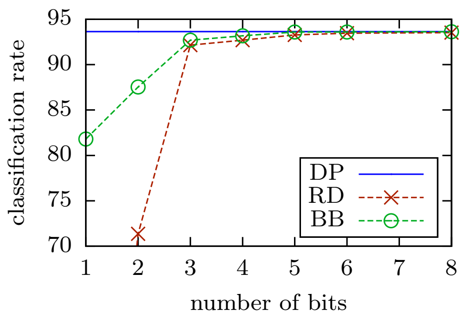


Fig. 10: Classification rates of BN classifiers using NB structure for MNIST and discriminative MM parameters. Parameters computed using the branch-and-bound approach (BB) [93] outperform parameters obtained by first computing full-precision parameters and subsequently rounding them to the desired precision (RD). The classification rates using double precision parameters (DP) upper bounds the performance of the classifiers with reduced-precision parameters.

4.5 Uncertainty Treatment

A key advantage of probabilistic models is that they allow to treat uncertainty in a consistent manner. While there are many types of uncertainty [61], e.g. data uncertainty stemming from noise, predictive uncertainty stemming from ambiguities, or model uncertainty, all of these can be treated in a uniform manner by virtue of probabilistic inference.

As an example, consider that a classifier has been trained on a fully observed data set (i.e. there are no input features missing), but it shall be applied in a setting where inputs drop out at random. This missing-at-random (MAR) scenario [116], although being an arguably simple and common one, is still a major cause of trouble for purely discriminative approaches like DNNs. The problem here is that DNNs at best represent a conditional distribution $p(C \,|\, \mathbf{X})$, which does not capture any correlations within $\mathbf{X}$. In a full joint distribution $p(C, \mathbf{X})$, as represented by PGMs, the MAR scenario is naturally handled by marginalizing missing features. In particular, given values $\mathbf{x}_o$ for a subset of input features $\mathbf{X}_o \subset \mathbf{X}$, we use $p(C \,|\, \mathbf{x}_o) = \frac{\int p(C,\mathbf{x}_h,\mathbf{x}_o) d\mathbf{x}_h}{\sum_c \int p(c,\mathbf{x}_h,\mathbf{x}_o) d\mathbf{x}_h}$ for classification, where $\mathbf{X}_h = \mathbf{X} \setminus \mathbf{X}_o$.

There is, however, a hinge for PGMs trained in a discriminative way: while discriminative training generally improves classification results on completely observed data, we cannot expect that these models also are robust under missing inputs. This can be easily seen by factorizing the joint $p(C, \mathbf{X})$ into $p(C \,|\, \mathbf{X}) p(\mathbf{X})$. Discriminative learning deliberately ignores $p(\mathbf{X})$ and focuses on tuning $p(C \,|\, \mathbf{X})$. In order to treat missing inputs in a consistent manner, however, we need to faithfully capture $p(\mathbf{X})$ as well. To this end, we might use hybrid generative-discriminative methods [78], [79], which aim at a sensible trade-off between predictive accuracy and "generativeness" of the employed model.

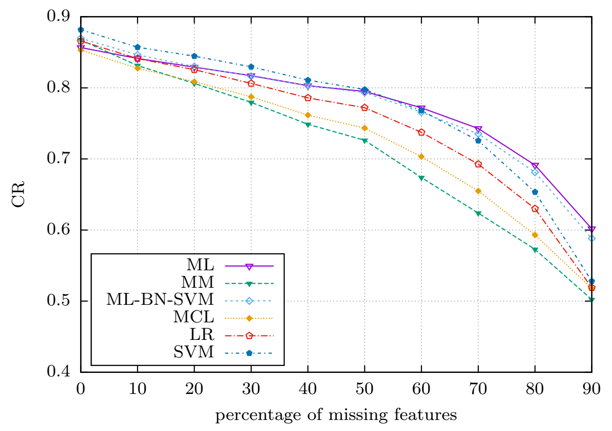


Fig. 11: Classification rate (CR) of various models under missing input features. ML, MM, ML-BN-SVM, and MCL are BNs trained with maximum-likelihood, maximum margin [74], a hybrid objective [78] and maximum-conditional likelihood, respectively. LR is logistic regression and SVM is a kernelized support vector machine. For LR and SVM, missing features were treated using 5-nearest neighbor imputation; for BNs missing features were treated by marginalization.

We demonstrate this effect using BNs trained with ML, MM, a hybrid ML-MM objective (ML-BN-SVM) [78], and MCL, and compare with classical logistic regression (LR) and kernelized SVMs. Since LR and SVMs cannot deal with missing inputs, we used 5-nearest neighbor imputation before applying the model. In Figure 11 we show the classification rate of all models as a function of percentage of missing features, averaged over 25 UCI data sets. We see that the model trained with ML is most robust under missing input features, and for more than 60% of missing features it outperforms all other models. The hybrid solution ML-BN-SVM has the second highest accuracy under no missing features and is almost as robust as the purely generative solution. The purely discriminative models MM, MCL, LR, and SVM are clearly more sensitive to missing features. Furthermore, note that K-NN imputation for LR and SVM requires that the training set is available also during test time; thus, this approach to treat missing features also comes with an significant additional memory requirement.

5 Conclusion

We compared deep neural networks (DNNs) and probabilistic graphical models (PGMs) regarding their efficiency and robustness for real-world systems, focusing on the possible trade offs of computation/memory demands and prediction performance. In general, DNNs require large amounts of computational and memory resources while PGMs with sparse structure are usually computationally efficient and have a small memory footprint. Unfortunately, PGMs often do not accomplish the same prediction performance as DNNs do, but they are able to treat uncertainty in a natural way and show benefits in case of missing features. Both modeling approaches have rich capabilities for machine learning on embedded devices.

For DNNs we discussed approaches for model size reduction. Furthermore, a comprehensive overview of DNNs with reduced-precision parameters was provided, with a focus on binary and ternary weights.

For PGMs we summarized discriminative and hybrid parameter and structure learning techniques to improve the prediction performance. Furthermore we devoted a section on PGMs using reduced-precision parameters. In experiments, we demonstrated the trade-off between prediction performance and computational- and memory requirements for several challenging machine learning benchmark data sets. Furthermore, we presented exemplary results comparing reduced-precision PGMs and DNNs.

Acknowledgments

This work was supported by the Austrian Science Fund (FWF) under the project number I2706-N31 and the German Research Foundation (DFG). Furthermore, we acknowledge the LEAD Project Dependable Internet of Things funded by Graz University of Technology and the SiliconAlps project Archimedes funded by the Austrian Research Promotion Agency (FFG). This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 797223 — HYBSPN.

We acknowledge NVIDIA for providing GPU computing resources.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1106–1114.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 3104–3112.

[4] Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[5] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems (NIPS)*, 1989, pp. 598–605.

[6] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems (NIPS)*, 1992, pp. 164–171.

[7] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 1135–1143.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *International Conference on Learning Representations (ICLR)*, 2016.

[9] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 1379–1387.

[10] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 2074–2082.

[11] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2011, pp. 2348–2356.

[12] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1613–1622.

[13] D. Molchanov, A. Ashukha, and D. P. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning (ICML)*, 2017, pp. 2498–2507.

[14] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 2575–2583.

[15] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 3288–3298.

[16] P. D. Grünwald, *The minimum description length principle*. MIT press, 2007.

[17] Z. Mariet and S. Sra, "Diversity networks: Neural network compression using determinantal point processes," in *International Conference of Learning Represenation (ICLR)*, 2016.

[18] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning (ICML)*, 2015, pp. 2285–2294.

[19] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *International Conference on Learning Representations (ICLR)*, 2017.

[20] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural Computation*, vol. 4, no. 4, pp. 473–493, 1992.

[21] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Deep Learning and Representation Learning Workshop @ NIPS*, 2015.

[22] A. Korattikara, V. Rathod, K. P. Murphy, and M. Welling, "Bayesian dark knowledge," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 3438–3446.

[23] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 2148–2156.

[24] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 442–450.

[25] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 1269–1277.

[26] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *British Machine Vision Conference (BMVC)*, 2014.

[27] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," in *International Conference on Learning Representations (ICLR)*, 2015.

[28] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. N. Choudhary, and S. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *International Conference on Computer Vision (ICCV)*, 2015, pp. 2857–2865.

[29] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. J. Smola, L. Song, and Z. Wang, "Deep fried convnets," in *International Conference on Computer Vision (ICCV)*, 2015, pp. 1476–1483.

[30] 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.

[31] M. Lin, Q. Chen, and S. Yan, "Network in network," in *International Conference of Learning Represenation (ICLR)*, 2014.

[32] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 367–379. [Online]. Available: https://doi.org/10.1109/ISCA.2016.40

[33] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 10–14.

[34] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *International Symposium on Microarchitecture (MICRO)*, 2016, pp. 20:1–20:12.

[35] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop @ NIPS*, 2011.

[36] M. Höhfeld and S. E. Fahlman, "Learning with limited numerical precision using the cascade-correlation algorithm," *IEEE Trans. Neural Networks*, vol. 3, no. 4, pp. 602–611, 1992.

[37] ——, "Probabilistic rounding in neural network learning with limited precision," *Neurocomputing*, vol. 4, no. 4, pp. 291–299, 1992.

[38] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1737–1746.

[39] M. Courbariaux, Y. Bengio, and J. David, "Training deep neural networks with low precision multiplications," in *International Conference on Learning Representations (ICLR) Workshop*, vol. abs/1412.7024, 2015.

[40] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *CoRR*, vol. abs/1510.03009, 2015.

[41] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning (ICML)*, 2016, pp. 2849–2858.

[42] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, vol. abs/1308.3432, 2013.

[43] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 3123–3131.

[44] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 4107–4115.

[45] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *CoRR*, vol. abs/1605.04711, 2016.

[46] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *International Conference on Learning Representations (ICLR)*, 2017.

[47] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 525–542.

[48] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Neural Information Processing Systems (NIPS)*, 2017, pp. 344–352.

[49] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *CoRR*, vol. abs/1603.01025, 2016.

[50] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave Gaussian quantization," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5406–5414.

[51] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713.

[52] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *CoRR*, vol. abs/1606.06160, 2016.

[53] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[54] H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein, "Training quantized nets: A deeper understanding," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 5813–5823.

[55] A. G. Anderson and C. P. Berg, “The high-dimensional geometry of binary neural networks,” in *International Conference on Learning Representations (ICLR)*, 2018.

[56] D. Soudry, I. Hubara, and R. Meir, “Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights,” in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 963–971.

[57] T. P. Minka, “Expectation propagation for approximate Bayesian inference,” in *Uncertainty in Artificial Intelligence (UAI)*, 2001, pp. 362–369.

[58] W. Roth and F. Pernkopf, “Discrete-valued neural networks using variational inference,” in *International Conference of Learning Represenation (ICLR), submitted*, 2018.

[59] D. J. Rezende, S. Mohamed, and D. Wierstra, “Stochastic backpropagation and approximate inference in deep generative models,” in *International Conference on Machine Learning (ICML)*, 2014, pp. 1278–1286.

[60] D. P. Kingma and M. Welling, “Auto-encoding variational Bayes,” in *International Conference on Learning Representations (ICLR)*, 2014, arXiv: 1312.6114.

[61] Z. Ghahramani, “Probabilistic machine learning and artificial intelligence,” *Nature*, vol. 521, pp. 452–459, 2015.

[62] F. Pernkopf, R. Peharz, and S. Tschiatschek, *Introduction to Probabilistic Graphical Models*. Elsevier, 2014, vol. 1, ch. 18, pp. 989–1064.

[63] F. Pernkopf and J. Bilmes, “Efficient heuristics for discriminative structure learning of Bayesian network classifiers,” *Journal of Machine Learning Research*, vol. 11, pp. 2323–2360, 2010.

[64] S. Tschiatschek and F. Pernkopf, “On Bayesian network classifiers with reduced precision parameters,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 4, pp. 774–785, 2015.

[65] ——, “Parameter learning of Bayesian network classifiers under computational constraints,” in *European Conference on Machine Learning (ECML)*, 2015, pp. 86–101.

[66] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science*, vol. 350, pp. 1332–1338, 2015.

[67] D. George, W. Lehrach, K. Kansky, M. Lázaro-Gredilla, C. Laan, B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang, A. Lavin, and D. S. Phoenix, “A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs,” *Science*, 2017.

[68] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[69] N. Piatkowski, S. Lee, and K. Morik, “Integer undirected graphical models for resource-constrained systems,” *Neurocomputing*, vol. 173, pp. 9–23, 2016.

[70] H. Wettig, P. Grünwald, T. Roos, P. Myllymäki, and H. Tirri, “When discriminative learning of Bayesian network parameters is easy,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, pp. 491–496.

[71] T. Roos, H. Wettig, P. Grünwald, P. Myllymäki, and H. Tirri, “On discriminative Bayesian network classifiers and logistic regression,” *Machine Learning*, vol. 59, pp. 267–296, 2005.

[72] R. Greiner, X. Su, S. Shen, and W. Zhou, “Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers,” *Machine Learning*, vol. 59, pp. 297–322, 2005.

[73] Y. Guo, D. Wilkinson, and D. Schuurmans, “Maximum margin Bayesian networks,” in *Uncertainty in Artificial Intelligence (UAI)*, 2005, pp. 233–242.

[74] F. Pernkopf, M. Wohlmayr, and S. Tschiatschek, “Maximum margin Bayesian network classifiers,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 521–532, 2012.

[75] F. Pernkopf and M. Wohlmayr, “Stochastic margin-based structure learning of Bayesian network classifiers,” *Pattern Recognition*, vol. 46, no. 2, pp. 464–471, 2013.

[76] G. Bouchard and B. Triggs, “The tradeoff between generative and discriminative classifiers,” in *IASC International Symposium on Computational Statistics (COMPSTAT)*, 2004, pp. 721–728.

[77] C. Bishop and J. Lasserre, “Generative or discriminative? Getting the best of both worlds,” *Bayesian Statistics*, vol. 8, pp. 3–23, 2007.

[78] R. Peharz, S. Tschiatschek, and F. Pernkopf, “The most generative maximum margin Bayesian networks,” in *International Conference on Machine Learning (ICML)*, 2013, pp. 235 – 243.

[79] W. Roth, R. Peharz, S. Tschiatschek, and F. Pernkopf, “Hybrid generative-discriminative training of gaussian mixture models,” *Pattern Recognition Letters*, vol. 112, pp. 131 – 137, 2018.

[80] T. Roos, H. Wettig, P. Grünwald, P. Myllymäki, and H. Tirri, “On discriminative Bayesian network classifiers and logistic regression,” *Machine Learning*, vol. 59, no. 3, pp. 267–296, 2005.

[81] J. Su, H. Zhang, C. X. Ling, and S. Matwin, “Discriminative parameter learning for Bayesian networks,” in *International Conference on Machine Learning (ICML)*, 2008, pp. 1016–1023.

[82] D. M. Chickering, D. Heckerman, and C. Meek, “Large-sample learning of Bayesian networks is NP-hard,” *Journal of Machine Learning Research*, vol. 5, no. Oct, pp. 1287–1330, 2004.

[83] S. Dasgupta, “Learning polytrees,” in *Uncertainty in Artificial Intelligence (UAI)*, 1999, pp. 134–141.

[84] D. Heckerman, D. Geiger, and D. M. Chickering, “Learning Bayesian networks: The combination of knowledge and statistical data,” *Machine learning*, vol. 20, no. 3, pp. 197–243, 1995.

[85] N. Friedman, D. Geiger, and M. Goldszmidt, “Bayesian network classifiers,” *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[86] F. Pernkopf and J. A. Bilmes, “Efficient heuristics for discriminative structure learning of Bayesian network classifiers,” *Journal of Machine Learning Research*, vol. 11, no. Aug, pp. 2323–2360, 2010.

[87] F. Pernkopf and M. Wohlmayr, “Stochastic margin-based structure learning of Bayesian network classifiers,” *Pattern Recognition*, vol. 46, pp. 464–471, 2013.

[88] E. J. Keogh and M. J. Pazzani, “Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches.” in *Artificial Intelligence and Statistics (AISTATS)*, 1999.

[89] H. Zhang, “The optimality of naive Bayes,” in *Florida Artificial Intelligence Research Society Conference (FLAIRS)*, 2004, pp. 562–567.

[90] H. Chan and A. Darwiche, “When do numbers really matter?” *Artificial Intelligence Research*, vol. 17, no. 1, pp. 265–287, 2002.

[91] S. Tschiatschek, K. Paul, M., and F. Pernkopf, “Integer Bayesian network classifiers,” in *European Conference on Machine Learning (ECML)*, 2014, pp. 209–224.

[92] S. Tschiatschek, P. Reinprecht, M. Mücke, and F. Pernkopf, “Bayesian network classifiers with reduced precision parameters,” in *Machine Learning and Knowledge Discovery in Databases*. Springer Berlin Heidelberg, 2012, pp. 74–89.

[93] S. Tschiatschek, “Maximum margin Bayesian networks (asymptotic consistency, hybrid learning, and reduced-precision analysis),” dissertation, Technische Universität Graz, 2014.

[94] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, “ImageNet: A large-scale hierarchical image database,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[95] J. Faraone, N. Fraser, M. Blott, and P. H. Leong, “SYQ: Learning symmetric quantization for efficient deep neural networks,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[96] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *Journal of Machine Learning Research*, vol. 18, no. 187, pp. 1–30, 2017.

[97] P. Wang, Q. Hu, Y. Zhang, C. Zhang, Y. Liu, and J. Cheng, “Two-step quantization for low-bit neural networks,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[98] G. Schindler, M. Zöhrer, F. Pernkopf, and H. Fröning, “Towards efficient forward propagation on resource-constrained systems,” in *European Conference on Machine Learning (ECML)*, 2018.

[99] E. Warsitz and R. Haeb-Umbach, “Blind acoustic beamforming based on generalized eigenvalue decomposition,” in *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, 2007, pp. 1529–1539.

[100] E. Warsitz, A. Krueger, and R. Haeb-Umbach, “Speech enhancement with a new generalized eigenvector blocking matrix for application in a generalized sidelobe canceller,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2008, pp. 73–76.

[101] J. Heymann, L. Drude, and R. Haeb-Umbach, “Neural network based spectral mask estimation for acoustic beamforming,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 196–200.

[102] J. Heymann, L. Drude, A. Chinaev, and R. Haeb-Umbach, "BLSTM supported GEV beamformer front-end for the 3RD CHiME challenge," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2015, pp. 444–451.

[103] H. Erdogan, J. Hershey, S. Watanabe, M. Mandel, and J. L. Roux, "Improved MVDR beamforming using single-channel mask prediction networks," in *Interspeech*, 2016.

[104] L. Pfeifenberger, M. Zöhrer, and F. Pernkopf, "DNN-based speech mask estimation for eigenvector beamforming," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 66–70.

[105] M. Zöhrer, L. Pfeifenberger, G. Schindler, H. Fröning, and F. Pernkopf, "Resource efficient deep eigenvector beamforming," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[106] J. Barker, R. Marxer, E. Vincent, and S. Watanabe, "The third 'CHiME' speech separation and recognition challenge: Dataset, task and baselines," in *IEEE 2015 Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2015.

[107] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[108] L. Pfeifenberger, M. Zöhrer, and F. Pernkopf, "Eigenvector-based speech mask estimation using logistic regression," in *Interspeech*, 2017, pp. 2660–2664.

[109] V. Emiya, E. Vincent, N. Harlander, and V. Hohmann, "Subjective and objective quality assessment of audio source separation," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 19, no. 7, 2011.

[110] T. Higuchi, N. Ito, T. Yoshioka, and T. Nakatani, "Robust MVDR beamforming using time-frequency masks for online/offline ASR in noise," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 4, 2016, pp. 5210–5214.

[111] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[112] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[113] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[114] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 2960–2968.

[115] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. H. W. Leong, M. Jahre, and K. A. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (ISFPGA)*, 2017, pp. 65–74.

[116] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*. John Wiley & Sons, 2014, vol. 333.

Franz Pernkopf received his MSc (Dipl. Ing.) degree in Electrical Engineering at Graz University of Technology, Austria, in summer 1999. He earned a PhD degree from the University of Leoben, Austria, in 2002. In 2002 he was awarded the Erwin Schrödinger Fellowship. He was a Research Associate in the Department of Electrical Engineering at the University of Washington, Seattle, from 2004 to 2006. Currently, he is Associate Professor at the Laboratory of Signal Processing and Speech Communication, Graz University of Technology, Austria. His research interests include machine learning, resource-efficient neural networks, probabilistic graphical models, and speech processing applications.

Wolfgang Roth received the BSc and MSc degrees (with distinction) in computer science from Graz University of Technology in 2015. Currently, he is with the Signal Processing and Speech Communication Laboratory at Graz University of Technology where he is working toward the PhD degree. His research interests include resource-efficient deep neural network, Bayesian learning and statistical pattern recognition.

Matthias Zöhrer received his MSc (Dipl. Ing.) degree in Telematik at Graz University of Technology, Austria, in summer 2013. Since 2013 he is a Research Associate at the Laboratory of Signal Processing and Speech Communication, Graz University of Technology, Austria. His research interests include deep learning, GPU optimized processing, robust speech recognition, speech enhancement and beamforming.

Lukas Pfeifenberger received the M.Sc. (Dipl. Ing. FH) degree in computer science from the University of Applied Sciences, Salzburg, Austria, in 2004. Since 2005 he has been working in the electronics industry on projects pertaining to FPGA design, DSP programming and communication acoustics. In 2013, he received the M.Sc. (Dipl. Ing.) degree in Telematik at Graz University of Technology, Austria. Since 2015 he has been a Research Associate at the Laboratory of Signal Processing and Speech Communication, Graz University of Technology, Austria. His research interests include machine learning, computer vision and speech enhancement. He currently pursues research projects in blind source separation and acoustic echo control.

Günther Schindler is a doctoral candidate at the Institute of Computer Engineering at the Ruprecht-Karls University of Heidelberg (Germany). He has received his MSc in computer engineering and BSc in electrical engineering in 2016 from the Ruprecht-Karls University of Heidelberg respectively in 2014 from University of Applied Sciences Munich. His research interests include architectures and algorithms for application specific computing on accelerators and methods for resource-efficient deep neural networks.

Holger Fröning is since 2011 an associate professor at the Institute of Computer Engineering at the Ruprecht-Karls University of Heidelberg (Germany). During this time, he was as visiting scientist with NVIDIA Research (Santa Clara, CA, US), and visiting professor at Graz University of Technology, Austria. From 2008 to 2011 he reported as postdoctoral fellow to Jose Duato from Technical University of Valencia (Spain). He obtained his PhD and MSc degrees 2007 respectively 2001 from the University of Mannheim, Germany. He has received a Google Faculty Research Award in 2014, and best paper awards at ICPP and IPDPS. His research interests include hardware and software architectures, related co-design, data movement optimizations, and power and energy consumption of computing systems.

Sebastian Tschiatschek received the BSc and MSc degrees (with distinction) in electrical engineering from the Graz University of Technology (TUG) in 2007 and 2010, respectively. He conducted his master's thesis during a one-year stay at ETH Zurich, Switzerland. In 2014 he earned a PhD degree at the Signal Processing and Speech Communication Laboratory at TUG. From 2015-2017 he was a postdoc in the Learning and Adaptive Systems Group at ETH Zurich. Currently he is a research associate at Microsoft Research Cambridge, UK. His research interests include submodular functions, their application in the machine learning related applications, Bayesian networks and inference under computational constraints.

Robert Peharz received the MSc degree in computer engineering and the PhD degree in electrical engineering from Graz University of Technology. His main research interest lies in machine learning, in particular probabilistic modeling, with applications to signal processing, speech and audio processing, and computer vision. Currently, he is a Postdoc in the Machine Learning Group at the University of Cambridge. He was recently awarded with a Marie-Curie Individual Fellowship.

Matthew Mattina is Senior Director of Machine Learning & AI Research at Arm, where he leads a team of researchers developing advanced hardware, software, and algorithms for machine learning. Prior to joining Arm in 2015, Matt was Chief Technology Officer at Tilera, responsible for overall company technology, architecture and strategy. Prior to Tilera, Matt was a CPU architect at Intel and invented and designed the Intel Ring Uncore Architecture. Matt has been granted over 30 patents relating to CPU design, multicore processors, on-chip interconnects, and cache coherence protocols. Matt holds a BS in Computer and Systems Engineering from Rensselaer Polytechnic Institute and an MS in Electrical Engineering from Princeton University.

Zoubin Ghahramani Zoubin Ghahramani is Professor of Information Engineering at the University of Cambridge, where he leads the Machine Learning Group. He studied computer science and cognitive science at the University of Pennsylvania, obtained his PhD from MIT in 1995, and was a postdoctoral fellow at the University of Toronto. His academic career includes concurrent appointments as one of the founding members of the Gatsby Computational Neuroscience Unit in London, and as a faculty member of CMU's Machine Learning Department for over 10 years. His research interests include statistical machine learning, Bayesian nonparametrics, scalable inference, probabilistic programming, and building an automatic statistician. He has held a number of leadership roles as programme and general chair of the leading international conferences in machine learning: AISTATS (2005), ICML (2007, 2011), and NIPS (2013, 2014). In 2015 he was elected a Fellow of the Royal Society.