TentacleNet: A Pseudo-Ensemble Template for Accurate Binary Convolutional Neural Networks

TentacleNet: A Pseudo-Ensemble Template for Accurate Binary Convolutional Neural Networks

Luca Mocerino, Andrea Calimera
Politecnico di Torino, 10129 Torino, Italy

Abstract

Binarization is an attractive strategy for implementing lightweight Deep Convolutional Neural Networks (CNNs). Despite the unquestionable savings offered, memory footprint above all, it may induce an excessive accuracy loss that prevents a widespread use. This work elaborates on this aspect introducing TentacleNet, a new template designed to improve the predictive performance of binarized CNNs via parallelization. Inspired by the *ensemble learning* theory, it consists of a compact topology that is end-to-end trainable and organized to minimize memory utilization. Experimental results collected over three realistic benchmarks show TentacleNet fills the gap left by classical binary models, ensuring substantial memory savings w.r.t. state-of-the-art binary ensemble methods.

***K*eywords** Deep Learning · Machine Learning · Binary Neural Network · Optimization

1 Introduction

Convolutional Neural Networks (CNNs) are known to be highly redundant, a positive characteristic for the training because it helps to achieve higher accuracy, but highly undesired during inference, when extra-functional metrics, like latency, energy and memory footprint, are just as important. No matter if the target is a cloud application hosted on a server queried by millions of users, or a mobile application run on low-power cores with limited resources, an efficient use of CNNs calls for effective optimization strategies.

There is plenty of compression or approximation techniques that serve this purpose operating at different levels of abstraction which leverage different knobs [1–3]. At the bit-level, *binarization* is a very attractive option. The pioneering idea, firstly introduced in [4] and then elaborated in [5] and [6], is to project weights and/or activations into a binary space. Moving from multi-bit representations (either floating-point or fixed-point) to single-bit has clear advantages, such as the lowering of the memory footprint and a better use of the available bandwidth since operands can be packed in a single line and accessed in parallel. Moreover, it allows the replacement of real and integer arithmetic with bit-wise operators, e.g. parallel Boolean XNOR and pop-counting [6], which are faster and less resource demanding. This latter aspect is however influenced by the type of hardware available. General purpose architectures grew to support mainstream applications operating on single-precision floating-point, therefore, less frequent instructions and unusual data representations, like those deployed in binary CNNs, have been dropped for the sake of area efficiency. To fill this gap, software macros can be used to unpack data and properly feed the execution units. This may introduce substantial performance overhead [7]. Dedicated hardware accelerators, like those introduced in [8–11], may be a better option as they can push binary CNNs toward impressive speed-up.

In spite of the potential savings brought, the use of binary CNNs is still quite limited, sometimes prohibitive, because of the poor predictive quality. For instance, compared to full-precision (32-bit floating-point), the accuracy drop may range from 2% to 10%, but even more depending on the complexity of the task [6]. The objective of this work is to address this limitation introducing a new model template named *TentacleNet*. The basic working principle is inspired by the *ensemble learning* theory, well known in machine and deep learning [12], that is, the assembly of many *weak* classifiers enables a *strong* predictor with higher accuracy. However, TentacleNet shows distinctive features that have been specifically designed to leverage the power of binary BNNs and to optimize resource usage. Moreover, it is end-to-end trainable and can be applied to any generic CNN model using the training procedures available in common deep learning frameworks. Due to its parallel topology, TentacleNet is ready for the forthcoming generation of parallel architectures with heterogeneous accelerators [13].

Experimental results collected over three computer vision tasks, i.e. image classification on CIFAR-10 and CIFAR-100 [14] and facial expression recognition on the FER13 data-set [15], reveal TentacleNet can reach the accuracy

of full-precision models, yet ensuring much lower memory footprint compared to state-of-the-art binary ensemble methods [16].

2 Background and Previous Works

2.1 Binarized Neural Networks

The recent literature shows several binarization methods for CNNs. In [4], Courbariaux et al. introduced the concept of CNNs with binary weights in the range {-1, 1}, leaving the activations in full-precision (floating-point 32-bit). In [5], the same authors presented a full-binary CNN where also the activations get projected in a binary space using a *sign* activation function. That is the first example of a full-binary CNNs processed with bit-wise XNOR and pop-counting, with no floating-point arithmetic. As side effect, the prediction accuracy suffered substantial degradation ($\geq 10\%$). Later, M. Rastegari et. al. presented XNOR-Net [6], an alternative architecture to mitigate the accuracy drop by re-scaling the binary output of each convolutional layer through a full-precision normalization layer. The improvement over [5] was remarkable: up to 16.3% on ImageNet [17]. The XNOR-Net represents still today the state-of-the-art for binary CNNs and therefore we borrowed the same architecture in this work (simply referred as BNN hereafter). However, our strategy can be extended to any type of binarized network.

2.2 Feature Extraction in a BNN

Given $\mathbf{x} \in \mathbb{R}^{ch \times w_{in} \times h_{in}}$ as the input feature and $\mathbf{w} \in \mathbb{R}^{ch \times kw \times kh}$ as the weight tensor, their convolution is approximated as follows:

$$\mathbf{x} * \mathbf{w} \approx \text{popcount}\ (\mathbf{X}\ \text{xnor}\ \mathbf{W}) \cdot K \cdot \alpha \qquad (1)$$

where K and α are the scaling factors. While weights ($\mathbf{W}$) are binarized off-line, the binary activation function is fused with the batch normalization and hence run online. Given the parameters of the batch normalization, i.e. variance σ^2, mean μ, scale γ, shift factor β, ϵ a coefficient for numerical stability, a generic feature map $\mathbf{x}$ is binarized as follows:

$$\mathbf{X} = BinACT_{0,1}(x) = \begin{cases} 1\ x \geq c \\ 0\ x < c \end{cases} \qquad (2)$$

with $c = \mu - \beta/\gamma\sqrt{\sigma^2 + \epsilon}$. As additional details, we do not use K, that is the activations scaling factor, in order to speed-up the training stage; c and α are represented as 32-bit floating point numbers.

It is worth to notice that within a BNN, the first and the last layers are kept and processed to full-precision (floating-point 32-bit) in order to mitigate the accuracy drop induced by the binarization of the inner layers. This is a relevant characteristic exploited by TentacleNet.

The efficient processing of a BNN requires an effective implementation of parallel bit-wise XNOR, pop-counting, and bit-2-word packing/unpacking (e.g. from 1 to 32-bit and vice-versa). While new specialized cores have an extended instruction-set coupled with dedicated hardware units, e.g. [8], for many general-purpose cores the only viable option is to make custom software macros. The performance gap between hardware acceleration and software implementation is large, with the latter being much slower [7]. For instance, in [18] Moss et al. showed that a custom FPGA-based inference engine gets 8.5× faster and 20× more energy efficient. TentacleNet is orthogonal to the kind of hardware, but it would benefit most from custom accelerators.

2.3 Ensembles Learning

Ensemble methods are well-known tools in statistics and machine learning; they are commonly used to improve resilience against under-/over-fitting [12]. The basic principle is simple, yet effective: use multiple *weak* estimators to build up with a single *strong* classifier. Random forests are practical examples, where the weak classifiers consist of decision trees [19].

Existing ensemble strategies mainly differ on (i) the training procedure adopted and (ii) how the outcome of the weak estimators are grouped and evaluated. The taxonomy is as follows:

Bagging [20]. The training dataset D is randomly partitioned into N sub-sets d_i ($i \in [1, N]$), with N the number of weak estimators. Each weak estimator is trained using d_i as the training set. During inference, the outputs of the N estimators are averaged or evaluated with a voting mechanism.

Boosting [21, 22]. Each weak estimator is trained (separately) over the full dataset D. The outputs of the N estimators are then fused using a linear transformation whose coefficients are learned at training time, for instance using AdaBoost algorithm [23].

Stacking [24, 25]. The N weak estimators are trained on the original data D. Then, their outputs are used as training-set for an additional meta-estimator, which is run in sequence during inference. The key feature is that the stack is built upon heterogeneous estimators.

There exist different works that proposed the use of ensemble methods for deep neural networks. Remarkable results are reported in [26], where the authors adopted a boosting strategy on image classification, but also in CoopNet [27] which combines multiple precision models to improve accuracy and inference latency. Even more interesting, the concept of ensemble learning can be found in the internal architecture of the most recent CNN models. For instance ResNet [28], DenseNet [29] and Inception series [30] have layers which combine branches produced by the previous layers to improve performance. This resembles an ensemble learning structure indeed.

All the above methods were thought to improve accuracy, with no particular attention to the complexity of the weak classifiers. The result is a dramatic increase in the model size. When extra-functional metrics (e.g. memory and la-

tency) enter the cost function, the selection of the weak estimators should be resource-oriented and not just accuracy-driven. In this regard, binary CNNs are good candidates: they are weak, fast and small. Some recent works explored this option. For instance, in [16] the authors adapted the classical ensemble methods to binary CNNs, *bagging* and *boosting* in particular. The collected results revealed that a large number of BNNs is needed to get close to the accuracy of the full-precision model, thus resulting in large memory space. TentacleNet takes a step forward, showing that binary ensembles can reach high accuracy with fewer resources.

3 TentacleNet

3.1 Architecture

TentacleNet is a parallel template embedding lightweights BNNs in a pseudo-ensemble structure. It serves any kind of feed-forward CNN, namely, any full-precision CNN can be translated following the TentacleNet template and exploit the binary computation.

A high-level view is depicted in Fig. 1. The inner core consists of n parallel branches, the *tentacles*, which play as the weak estimators. Each tentacle (labeled as BNN_i) is a replica of the binarized floating-point model, except for the first and last layer which are shared among all the tentacles, these are the *Convolutional Block* and the *Fully-Connected Block* in Fig. 1. The former (grey box) is in charge of producing a common activation map fed as input to all the tentacles. It contains three sub-layers, convolution (CONV), normalization (Norm) and activation (ACT). The latter (blue box) implements the actual classification of the binary features extracted along the tentacles. It is worth emphasizing that all the tentacles operate full binary operations [-1,1], while the *Convolutional Block* and the *Fully-Connected Block* are taken to full-precision [FP], i.e. floating-point 32-bit. This design choice is inherited from state-of-the-art BNN models [4–6], which suggest leaving the first and last layer to full-precision gets higher accuracy. Another important aspect is that the sharing of the two blocks, the most memory demanding due to high arithmetic precision, contributes to save memory space.

The shape of the shared *Fully-Connected Block* differs depending on the topology of the original full-precision model. If the original model does produce the C logits through global pooling (where C is the number of classes), namely, without any fully-connected layer (Fig. 2-a), the logits are simply concatenated as a 1-D vector of cardinality $N \times C$ and then fed as input to the *Fully-Connected Block*, which is a dense layer of shape $N \times C$ inputs and $N \times C^2$ weights. Otherwise, if the original model has its own fully-connected layer to produce the logits (Fig. 2-b), we drop it out and concatenate the feature maps as a 1-D vector of cardinality $N \times K$, with K the number of features of each weak estimator; in such case the *Fully-Connected Block* is a dense layer of shape $N \times K$ inputs and $N \times K \times C$ weights.

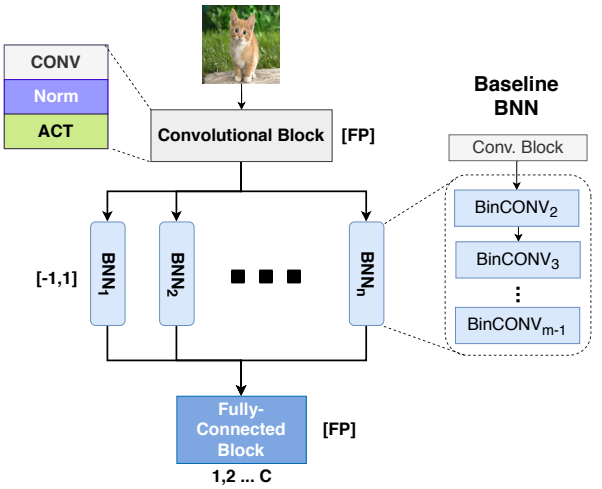

Figure 1: TentacleNet architecture. CONV refers to convolutional layers, Norm to normalization layer, ACT to activation layers. The prefix Bin stands for binary.

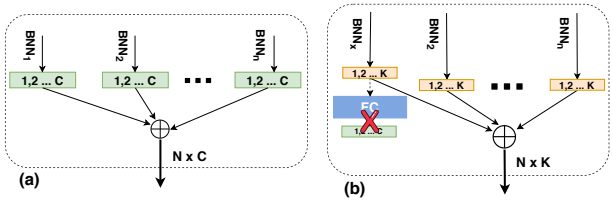

Figure 2: Composition of the Fully-Connected Block

3.2 Building and Training Methodology

TentacleNet can be seen as a pseudo-ensemble that implements some mixed features belonging to the stacking and boosting methods, in particular: the stack is composed of heterogeneous learners with different data-representation; the outputs of the weak estimators are evaluated through a linear transformation; all the layers, including the first and last block, are trained within a single procedure using the same data-set. The result is an end-to-end trainable model whose parameters can be learned through classical back-propagation.

The assembling of TentacleNet encompasses few stages. The entry level is a pre-trained floating-point CNN model binarized following the topology described in [6], namely, first and last layers as floating-point and the inner $m-2$ layers as binary. The sequence of such $m-2$ binary layers (from BinCONV_2 to BinCONV_{m-1} as reported in the left diagram of Fig.1) builds a tentacle. n replicas of the same tentacle are placed in parallel (from BNN_1 to BNN_n in Fig. 1) and then tied to the top and the bottom with the first convolutional block and the last fully connected block as described in the section 3.1. Once the TentacleNet is assembled, the training procedure described in [6] is deployed to learn the weights of the binary tentacles and the weights of the shared layers.

In order to guarantee enough expressive power and reduce the risk of under-/over-fitting, the *tentacles* are initialized with different seeds. Saxe et al. [31] demonstrated that weights initialized with orthonormal or orthogonal bases achieve better performance. In the binary domain, the matrices that satisfy these conditions are called *Hadamard matrices*. They are square matrices of order 1, 2, or $4n$, with $n \in \mathbb{N}$; the entries are -1 and 1 and can be generated using Sylvester's method. In order to adapt the Hadamard matrices to the dimensions of the binary kernels within the tentacles, the rows are randomly removed (to reduce the rank) or replicated (to increase the rank). The resulting *pseudo*-Hadamard matrices are sub-optimal but still a favorable initialization [5].

4 Experimental results

4.1 Benchmarks and Data-sets

TentacleNet has been evaluated on the following tasks.
Image Classification (CIFAR-10/100) - the standard image classification problem; the data-set contains 60k 32x32 RGB labeled images and can be configured for 10- or 100-class recognition [14].
Facial Expression Recognition (FER13) - emotion recognition from facial expression; the data-set collects 36k 48x48 gray-scale facial images labeled with seven different facial expressions [15].

Each of the above tasks is implemented through a specialized CNN model as reported in Table 1; the same models work as baseline to build the TentacleNet. The table collects the classification accuracy (%) and the model size (kB) of the three networks trained in full-precision 32-bit (row FP32) and after binarization (row BNN); here the BNN models refer to XNOR-Net [6]. As expected, BNNs reach remarkable memory reduction (e.g. 24.2× for ResNet9 on CIFAR-100) at the cost of significant accuracy loss (8.05% as the worst-case).

4.2 Training and Inference Set-Up

For each task a dedicated TentacleNet is built starting from the BNN model. The training of TentacleNet iterates for 300 epochs using an adaptive learning rate (lr) schedule: lr updated with step 0.1 every 15 consecutive epochs in which the validation loss does not change. Both training and inference stages are implemented using PyTorch (version 1.1.0) and made run on a server powered with 40-core Intel Xeon CPUs and accelerated with the NVIDIA Titan Xp GPU (CUDA v10.0).

Since the focus of this paper is on the accuracy-vs-memory tradeoff of binary ensembles, the assessment of hardware-dependent extra-functional metrics, like latency and energy, is left aside as part of future works.

	Dataset	CIFAR-10	CIFAR-100	FER13
	Baseline Model	NiN [32]	ResNet9 [28]	FerNet
FP32	Accuracy (%)	88.11	68.25	65.16
	Model Size (kB)	3778	19984	1880
BNN	Accuracy (%)	85.20	60.20	62.86
	Model Size (kB)	181	826	64

Table 1: Benchmarks: Datasets and CNNs

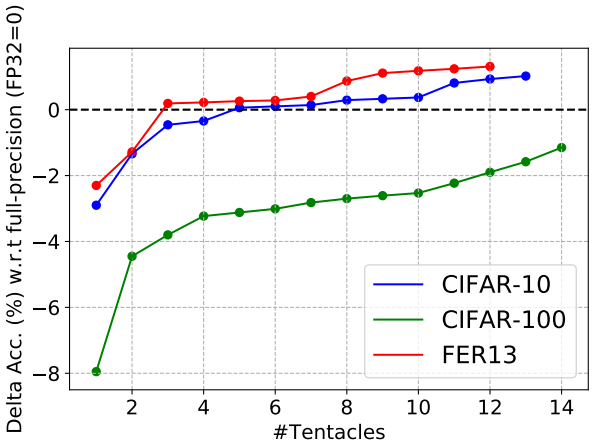

Figure 3: Delta Accuracy (%) w.r.t. the FP32 model as function of the number of Tentacles.

4.3 Performance assessment

The objective of this section is twofold: (i) prove that TentacleNet can push binary CNNs towards full-precision accuracy; (ii) show that TentacleNet outperforms existing binary-ensemble methods, both in terms of accuracy and memory.

Concerning the first issue, Fig. 3 reports a parametric analysis of the classification accuracy achieved by TentacleNet on the three benchmarks. The line plot shows the delta accuracy, which is the difference between TentacleNet and the FP32 model; the break-even point is centered on zero (horizontal dotted line). Deploying just one tentacle, TentacleNet collapses to the original BNN model, namely, the accuracy drop is the same reported in Table 1. As a general trend, the distance to FP32 gets smaller with the number of tentacles. For NiN over CIFAR-10 and FerNet over FER13 TentacleNet reaches the break-even with 3 and 5 tentacles respectively, and it goes even above towards positive values, +1.00% with 13 tentacles for CIFAR-10 and +1.31% with 12 tentacles for FER13, meaning that it outperforms FP32 models with much less weight memory: 645kB vs 3778kB for CIFAR-10 (83% savings), 188kB vs 1880kB for FER13 (90% savings). The behavior for ResNet over the more complex CIFAR-100 data-set is less performing compared to the other two benchmarks, yet remarkable. With 14 tentacles, the delta accuracy improves from -8.05% to -1.15%, very close to FP32, still ensuring low memory footprint, 11465 kB vs 19984 (42.6% less). For all the three benchmarks, additional experiments revealed the accuracy of TentacleNet saturates, namely,

there is no further improvement by increasing the number of tentacles; the top right points of the three lines in the plot of Fig. 3 show the highest accuracy that can be reached. For what concerns the second issue, we provide

Benchmark	Template		Δ (%)	#Ensemble/ Tentacle	M. Size (kB)
CIFAR-10 (NiN)	[16]	Bagging	0	12	2167
		Boosting	0	8	1445
		TentacleNet	0	5	645 (**55.3%**)
CIFAR-100 (ResNet9)	[16]	Bagging	-4.82	30	24755
		Boosting	-4.77	25	20629
		TentacleNet	-1.15	14	11465 (**44.4%**)
FER13 (FerNet)	[16]	Bagging	-0.35	11	697
		Boosting	-0.67	26	1648
		TentacleNet	0	3	188 (**73.0%**)

Table 2: TentacleNet vs BENN [16]

a quantitative comparison against BENN [16], which is state-of-the-art for binary ensembles. The BENN strategy is to apply standard ensemble methods to BNNs, bagging and boosting in particular. To ensure a fair comparison we implemented and applied the two BENN methods on the benchmarks under analysis. The main results are collected in Table 2, which reports the delta accuracy w.r.t. the FP32 model (as in Fig.3), the number of ensembles (for BENN) or tentacles (for TentacleNet), and the memory footprint (the percentage reported in brackets refers to the memory savings of TentacleNet w.r.t. the smallest BENN model). When possible, the comparison is done at the break-even point with the FP32 model (i.e. Δ=0), otherwise at the highest achievable accuracy. TentacleNet is more accurate and more compact than BENN over the three benchmarks. For instance, considering the NiN model over CIFAR-10, both BENN and TentacleNet achieve the accuracy of the FP32 model, but TentacleNet needs less memory to store the weights (55.3% less w.r.t. boosting). Larger savings have been observed for FerNet over the FER13 benchmark, where BENN and TentacleNet are almost equivalent in terms of accuracy (TentacleNet +0.35% more accurate than BENN bagging), but with a large memory spread (TentacleNet is 73% smaller than BENN bagging). Also for the most complex network, namely ResNet over CIFAR-100, TentacleNet reaches higher accuracy (+3.62% w.r.t. BENN boosting) with large memory savings (44.4%). To be noted that the memory footprint of the smallest BENN (20629 kB) gets bigger than the original FP32 model (19984 kB). Overall, TentacleNet is more accurate and much smaller than other binary ensemble methods.

5 Conclusions

References

[1] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] S. Han *et al.*, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 243–254.

[3] L. Mocerino *et al.*, "Energy-efficient convolutional neural networks via recurrent data reuse," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 848–853.

[4] M. Courbariaux *et al.*, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[5] ——, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[6] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[7] Y. Hu *et al.*, "Bitflow: Exploiting vector parallelism for binary neural networks on cpu," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2018, pp. 244–253.

[8] Y. Umuroglu *et al.*, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[9] A. Al Bahou *et al.*, "Xnorbin: A 95 top/s/w hardware accelerator for binary convolutional neural networks," in *2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE, 2018, pp. 1–3.

[10] Y. Li *et al.*, "A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks," in *FPGA*, 2017, pp. 290–291.

[11] R. Andri *et al.*, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 236–241.

[12] T. G. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, ser. MCS '00. London, UK, UK: Springer-Verlag, 2000, pp. 1–15.

[13] A. Garofalo *et al.*, "Pulp-nn: Accelerating quantized neural networks on parallel ultra-low-power risc-v processors," *arXiv preprint arXiv:1908.11263*, 2019.

[14] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[15] I. J. Goodfellow *et al.*, "Challenges in representation learning: A report on three machine learning contests," *Neural Networks*, vol. 64, pp. 59–63, 2015.

[16] S. Zhu *et al.*, "Binary ensemble neural network: More bits per network or more networks per bit?" in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4923–4932.

[17] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[18] D. J. M. Moss *et al.*, "High performance binary neural networks on the xeon+fpga™ platform," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–4.

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] ——, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug 1996.

[21] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, Jun 1990.

[22] ——, "The boosting approach to machine learning: An overview," in *Nonlinear estimation and classification*. Springer, 2003, pp. 149–171.

[23] Y. Freund *et al.*, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[24] D. Wolpert *et al.*, "Combining stacking with bagging to improve a learning algorithm," *Santa Fe Institute, Technical Report*, 1996.

[25] L. Breiman, "Stacked regressions," *Machine Learning*, vol. 24, pp. 49–64, 1996.

[26] M. Moghimi *et al.*, "Boosted convolutional neural networks." in *BMVC*, 2016, pp. 24–1.

[27] L. Mocerino *et al.*, "Coopnet: Cooperative convolutional neural network for low-power mcus," *arXiv preprint arXiv:1911.08606*, 2019.

[28] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[29] G. Huang *et al.*, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[30] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[31] A. M. Saxe *et al.*, "Dynamics of learning in deep linear neural networks," in *NIPS Workshop on Deep Learning*, 2013.

[32] M. Lin *et al.*, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.