Benchmarking the Accuracy of Algorithms for Memory-Constrained Image Classification

Benchmarking the Accuracy of Algorithms for Memory-Constrained Image Classification

Sebastian Müksch*
sebastian.mueksch@gmail.com

Theo Olausson*
tolausso@inf.ed.ac.uk

John Wilhelm*
jwilhelm@inf.ed.ac.uk

Pavlos Andreadis
pavlos.andreadis@ed.ac.uk

School of Informatics, University of Edinburgh, UK

Abstract—**Convolutional Neural Networks, or CNNs, are the state of the art for image classification, but typically come at the cost of a large memory footprint. This limits their usefulness in edge computing applications, where memory is often a scarce resource. Recently, there has been significant progress in the field of image classification on such memory-constrained devices, with novel contributions like the ProtoNN, Bonsai and FastGRNN algorithms. These have been shown to reach up to 98.2% accuracy on optical character recognition using MNIST-10, with a memory footprint as little as 6KB. However, their potential on more complex multi-class and multi-channel image classification has yet to be determined. In this paper, we compare CNNs with ProtoNN, Bonsai and FastGRNN when applied to 3-channel image classification using CIFAR-10. For our analysis, we use the existing Direct Convolution algorithm to implement the CNNs memory-optimally and propose new methods of adjusting the FastGRNN model to work with multi-channel images. We extend the evaluation of each algorithm to a memory size budget of 8KB, 16KB, 32KB, 64KB and 128KB to show quantitatively that Direct Convolution CNNs perform best for all chosen budgets, with a top performance of 65.7% accuracy at a memory footprint of 58.23KB.**

I. Introduction

Applying image classification in the real world is a task which comes with several innate challenges: occlusion, intra-class variability, varying lighting conditions and many more. Significant progress has been made towards solving this open problem via deep learning, in particular in the form of Convolutional Neural Networks, or CNNs [12]. However, prominent models to have achieved state-of-the-art performance tend to be very large (e.g. [16]). This limits the feasibility of applying CNNs to carry out image classification on memory-constrained devices. One alternative is to offload the inference to a data centre but this brings several challenges of its own, in particular with respect to reliability, its affect on the overall system cost [18] and a potential worsening of battery life and latency [15]. The rise of 5G connectivity could mitigate some of these concerns by offering greater reliability and stability of communication. However, several interesting areas of application may still simply be incompatible with offloading due to matters of privacy [17] (affecting e.g. surveillance cameras) or the previously mentioned impact on latency and battery life.

To circumvent the issues associated with offloading, one may instead carry out inference directly on the edge by using models with a small enough memory footprint. The quest to construct powerful models with tiny memory footprints has sparked diverse streams of research targeting equally diverse applications. As far as image classification goes, experimental results have thus far been centred around the MNIST data set [14]. However, in the present day the usefulness of this data set is limited: recent works are starting to reach saturating levels of performance (i.e. above 99% test set accuracy of models fitting within 2KB of memory [8]), and the single-channel nature of the data set may bias the state of the art towards methods which may not generalise well when the input images consist of several channels, e.g. coloured images in RGB or HSV encoding. Motivated by these insights, this paper presents new research by comparing the state-of-the-art methods for memory-constrained image classification on a significantly more challenging data set, CIFAR-10 [11], thus establishing new benchmarks.

Section II briefly discusses the specifics of the CIFAR-10 [11] data set. Section III introduces four methods from the literature along with their perceived strengths and weaknesses in simpler applications. Following this, Section IV outlines our methodology for configuring these models optimally for the CIFAR-10 task, given the memory constraints, including a new method of adjusting FastGRNN [3] for multi-channel images. Finally, Section V describes our experiments and their results while Section VI adds concluding remarks and outlines future work.

II. Problem Definition

The CIFAR-10 data set consists of 60,000 32×32 3-channel colour images, divided into 10 classes such as airplane, automobile and dog [11]. It comes split in two balanced sets of 50,000 images for training and 10,000 for testing. For our analysis, we extract a validation set consisting of exactly 1000 images of each class from the training set, leaving 40,000 images for training. The algorithms will then be compared using the test set accuracy.

III. Algorithms under Consideration

In this section we briefly introduce *Direct Convolutions* [8], *FastGRNN* [3], *Bonsai* [13] and *ProtoNN* [4] as the state of the art for memory-constrained image classification. We highlight and discuss each method's image classification performance as has previously been reported in the literature.

* Co-First Author.

A. *Direct Convolutions*

The *Direct Convolution* [8] method significantly reduces the memory overhead of using CNNs through clever re-use of memory. Memory used to store the pixels of an input feature map is progressively replaced with the activations of the layer as the inputs become stale, i.e. all activations depending on the input pixel have been computed [8]. Though deceptively simple, this method is made significantly more complicated when a layer increases the channel depth: i.e. when the number of channels in the input is strictly less than that in the output. In such scenarios, naively processing the pixels in row-major order would cause the memory to be freed in a fragmented manner, making it difficult to re-use the freed memory. To avoid this issue, a *herringbone* strategy is proposed where the pixels are traversed in alternating row- and column-major order, which provably uses optimal space [8] in addition to that taken up by the input image.

Currently, this method holds the record performance (99.15% test accuracy) on 10-class MNIST classification for models with memory footprints on the order of kilobytes and it does so with a model of only 2KB of memory [8]. However, no results for a data set other than MNIST are presented in [8], nor were any such results identified elsewhere.

B. *FastGRNN*

Gating mechanisms have a rich history and have long been used to stabilize training and improve performance of RNNs, such as in Long Short-Term Memory units, or LSTMs, [9]. However, gates add extra parameters to the model and thus increase its memory footprint; as a result the gating mechanisms used in LSTMs have since been refined by GRUs [6] and UGRNNs [2]. *FastGRNN* [3] is a gated RNN which continues along this line of producing smaller gating mechanisms.

FastGRNN uses only two prediction parameters to compute both the gate activation and the hidden state, and the model can then further be compressed through quantization or keeping the two prediction parameters low-rank, sparse or both. To allow for this compression, FastGRNN uses a three-stage training process split evenly over the training epochs. First, the sparsity constraints are ignored while a low-rank version of the parameters are learned. Secondly, the sparsity structure is learned along with the support for the prediction parameters. Finally, the support set of the parameters is frozen and the model is then fine-tuned [3].

Impressively, FastGRNN is able to accurately capture the wakeword "Hey Cortana" with a model size of only 1KB [3]. However, somewhat surprisingly, FastGRNN is also apt at image classification, achieving 98.2% accuracy with a 6KB model on a pixel-by-pixel version of 10-class MNIST [3]. As such, it makes for an interesting contender in our analysis despite recurrent models typically not being used for image classification in the wider literature.

C. *Bonsai*

Bonsai [13] is a decision-tree based algorithm for resource-constrained machine learning, which along with learning a non-linear tree also learns a low-dimensional projection. The model size is kept small by training a single tree rather than an entire forest on the low-dimensional projected data and by ensuring that the learned projection matrix is sparse.

Though the authors carry out experiments targeting several data sets, the results are mainly compared to those of pruned version of large networks, rather than architectures which directly target the resource-constrained systems. While it may have been state-of-the-art at the time of its publication, Bonsai's 97.01% 10-class MNIST test set accuracy at 84KB [13] is now outperformed by both Direct Convolutions and FastGRNN. On a 2-class version of CIFAR, this method is shown to achieve 73.02% accuracy while only using up 2KB of memory; at 16KB it reaches 76.64% [13].

D. *ProtoNN*

The *ProtoNN* algorithm takes inspiration from the familiar *k-Nearest-Neighbours* method. ProtoNN performs inference analogously to k-NN, however, it distinguishes itself by requiring several orders of magnitude less space and time. This is achieved by learning a small set of informative *prototype* data points to compare against at time of inference, along with a sparse projection onto low-dimensional space [4].

Compared to the other methods, ProtoNN achieves a less impressive 95.88% accuracy on 10-class MNIST classification and requires 64KB of memory to do so [4]. However, similarly to Bonsai, ProtoNN is tested on a 2-class version of CIFAR on which it achieves 76.35% accuracy with 16KB [4].

IV. METHODOLOGY

This section details how each of the methods outlined in Section III can be applied to solve the 3-channel image classification, along with the way in which we optimise each method for the given memory budgets.

A. *Direct Convolutions*

In order to obtain 99.15% accuracy on 10-class MNIST introduced in Section III-A, a sampling-based neural architecture search was performed in [8]. Given the strong performance of this method on MNIST, we adopt this search and its space for the CIFAR-10 experiments as well. Specifically, this involves first enumerating a large set of model architectures, then calculating the memory requirements for each. From this data a given number of models satisfying the memory size budget are sampled. Each of the selected models is partially trained (e.g. for 5 epochs) and the best model based on validation accuracy after this limited training is identified. Finally, the identified model is trained fully using early stopping, and this is considered the optimal model. We deem this sampling approach reasonable given recent empirical and theoretical evidence that randomly searching for hyper-parameters can be more efficient than a guided or grid search [1].

B. *FastGRNN*

To our knowledge, we are the first to apply FastGRNN to a domain with multi-channel images. This raises the question of how to model the input data as a time series in order to benefit from the recurrent nature of the network. In simple single-channel images such as those found in MNIST [14], the input data can be turned into a time series either by feeding the network with a single pixel at a time (e.g. [3]) or by feeding the network with a group of pixels, e.g. a full row or column, at a time (e.g. the examples given in [5]). In either case, multi-channel images complicate this process by introducing an implicit trade-off between proximity in the time series between pixels which lie close together within a single channel (e.g. adjacent pixels in the red channel) and pixels which lie close together across the channels (e.g. the first pixels in each of the three channels).

We propose three different methods for classifying multi-channel images with FastGRNN. These share the basic assumption that each data point in the time series is one row of one channel in the input, but differ in how the data is fed into the network. **Row-major**: Feed the data into a single FastGRNN unit, followed by a fully-connected layer, starting first with all red rows, then all green rows and finally all blue rows. **Channel-major**: Feed the data into a single FastGRNN unit, followed by a fully-connected layer, starting with the first red row, the first green row, the first blue row, then the second red row, the second green row and the second blue row, etc. until the last blue row. **Multi-FastGRNN**: Feed the data into three separate FastGRNN units, one for each channel, followed by a fully connected layer. Feed each unit with the rows of the channel corresponding to the unit, in order.

For an RNN, learning features from several elements in its input sequence is tied to their *temporal latency*, i.e. distance between them in the sequence. With the row-major method, we focus on features that relate the pixels in the first red row to those in the second red row etc., where it takes time to see the next channel. With the channel-major method, we focus on features that relate pixels in the first rows of each channel, where it takes time to see the next row. As such there is a trade-off between setting up for intra- and inter-channel features. With the Multi-FastGRNN architecture we seek to circumvent this trade-off by explicitly separating the channels and training one FastGRNN unit per channel. To retain the ability to learn cross-channel features, we then concatenate the outputs of each unit and feed the combined output into a fully-connected layer.

The size of a FastGRNN unit is predominantly determined by its input dimensionality and its hidden dimensionality. As briefly mentioned in Section III, the model can then be further compressed if the predictor parameters U and W [3] are kept low-rank and sparse, or if the parameters are quantized. In this analysis, we carry out a grid search on the hidden dimensionality and the sparsity of U and W to tune the size of the models, to accommodate for the large amount of size budgets and input sequencing methods to investigate. We thus keep the prediction parameters full-rank and do not carry out quantization, a trade-off which we do not believe to be a cause for concern as our analysis involves comparatively large memory budgets for the domain (up to 128KB).

C. Bonsai

Applying Bonsai to multi-channel inputs is straightforward as it does not assume any spacial relationship between features.

As a Bonsai model is parameterised by the depth of the decision tree and the dimensionality of the projection matrix [13], we exhaustively search over these two parameters. We begin by fixing the tree depth to 1 and train models with varying dimensionality of the projection matrix, starting with a dimensionality of 1 and then incrementing it until we reach a model that no longer fits into the largest memory budget. After that, we increment the tree depth and repeat the process, until the tree depth is too large to to yield any models fitting within the maximum memory budget. In order to make the search tractable, all other remaining parameters will be set based on recommended values sourced from the litrature.

D. ProtoNN

Like Bonsai, ProtoNN does not assume any spacial relationship between elements of the feature vectors. Therefore, multi-channel images can simply be flattened. More interestingly, ProtoNN exposes three parameters which have an effect on the model size: the dimensionality of the projected space, the number of prototypes to learn, and a sparsity constraint on the learned matrices (to allow for compression). The model's performance is also affected by the parameter γ from the Gaussian kernel similarity function. Thus the best performing ProtoNN model for each memory budget can be obtained by carrying out a grid search over these parameters along with the learning rate of the optimization.

V. Experiments

In this section we set up and perform experiments following the methodology outlined above. For the ProtoNN, Bonsai, and the FastGRNN methods we use the versions included in the EdgeML library [5], while for Direct Convolutions we base our experiments on the software package provided by [7].

A. Direct Convolutions

As detailed in Section IV-A, we will employ a sampling based neural architecture search to obtain the best performing models for this method. In particular, we will sample 750 models for each memory budget from the viable model set[1] and partially train each selected model for 5 epochs. We use the Adam optimiser [10] for training, with an initial learning rate set to 0.005 which is then decayed by 0.95 each epoch. From this partial training the best model for each memory budget, based on validation set accuracy, is identified. These models are then trained for 100 epochs, using early stopping with a patience of 5 epochs. Finally, the test set accuracies of these models are computed, which are given in Table I.

[1]Detailed fully in https://arxiv.org/pdf/2005.04968.pdf, Table 1 & 2. [Includes fixed parameters of the models.]

Budget	Model	Test Accuracy [Size]
$\leq$ 8KB	$A, C_2(16,(3,3)), C_1(8,(3,3)), C_1(32,(3,3)), M, Dr, D^*$	0.604 [5.39KB]
$\leq$ 16KB	$A, C_1(6,(3,3)), C_1(32,(1,1)), M, C_2(64,(3,3)), Dr, D^*$	0.629 [8.65KB]
$\leq$ 32KB	$A, C_1(8,(1,1)), C_2(16,(3,3)), C_1(64,(5,5)), M, Dr, D^*$	0.643 [19.91KB]
$\leq$ 64KB $\leq$ 128KB	$A, C_1(64,(3,3)), M, C_1(64,(1,1)), C_1(64,(5,5)), Dr, D^*$	0.657 [58.23KB]

TABLE I: Best models for the Direct Convolution method. Symbols A, C_1, C_2, M, Dr, D^* denote Average Pooling, Convolution, Depth-wise Convolution, Max. Pooling, Dropout and Dense (with ReLU activation) Layers, respectively. C_1 and C_2 are followed by their variable parameters $output\_dim$ then $kernel\_size$.

B. FastGRNN

We begin by constructing several candidate models for each size budget. For the row- and channel-major models we vary the density of U and W independently between 0.1, 0.2 and 0.3 based on recommendations by the EdgeML library [5]. However, we also consider fully dense models where the densities of U and W are both 1.0. We then construct models by varying the hidden dimensionality in steps of 15 between 0 and 225. To make training tractable, we discard all but three models per size budget, keeping the models which approach the size budget most closely. For Multi-FastGRNN we fix the density of U to 1.0 as we find the compression of U to have a negligible effect on the model size due to the small hidden dimensionality per unit, which we vary in steps of 5 between 0 and 100. We again discard all but three candidate models per budget. However, as the search does not yield three models for the 8KB size budget, we manually add a dense model with hidden dimensionality 12 and a sparse model with hidden dimensionality 14 and density 0.1 for W.

We are left with 45 models to train. Due to the large number of models to compare, we fix the update and gate non-linearities to the hyperbolic tangent and sigmoid function, respectively. These are identified as good defaults by [3].

We use an Adam optimiser [10] with learning rate 0.01, which is decayed by a factor of 0.1 after 100 epochs for the 32, 64, and 128KB model candidates. Starting at the 64KB budget, we also introduce a weight decay of 5×10^{-4} to reduce overfitting. We fix the batch size to 100 and the number of epochs to 150. Due to FastGRNN's three-stage training process, we do not carry out early stopping. Instead we roll the model back to the post-compression stage epoch at which it obtained the best validation accuracy before computing the test accuracy. Final results are given in Table II.

C. Bonsai

We search over possible Bonsai models by fixing the depth of the Bonsai tree and varying the dimensionality of the projection matrix, as outlined in Section IV-C. We use the Adam optimiser [10] with an initial learning rate of 0.01, along with a batch size of 224 (the square root of the original 50,000 training samples [11]).

For the Bonsai-specific hyper-parameters, we set the sigmoid sharpness to 1 and set a regulariser of 10^{-3} for the prediction parameters W and V and the branching parameter θ. For W and V we set a sparsity of 0.3 and for θ a sparsity of 0.62. Finally, we set a regulariser of 10^{-4} with sparsity 0.2 for the projection parameter Z, all based on recommendations in [13].

We train all models for 200 epochs. We do not perform early stopping due to Bonsai's sequence of distinct training phases [13], but for each memory size budget choose the model with the highest validation accuracy after the 200 training epochs.

Tree depths 1 to 8 are explored, however, as Figure 1 demonstrates the validation accuracies of trees with depth 7 and 8 are, on average, significantly lower than for shallower Bonsai trees. We attribute this to a necessarily smaller dimensionality of the projection matrix required to fit deeper trees into the memory budgets. For example, a Bonsai tree of depth 5 can utilise a projection matrix with dimensionality up to 16 and achieving up to 37.6% validation accuracy with a memory size of 94.52KB. Compare this to a tree of depth 8 which can utilise a projection matrix up to only dimensionality 4 and only achieves a maximum accuracy of 16.3% while requiring 89.25KB of memory. We therefore stop exploring deeper Bonsai trees and summarise the best models in Table III.

	Budget	$\leq$ 8KB	$\leq$ 16KB	$\leq$ 32KB	$\leq$ 64KB	$\leq$ 128KB
Row	Dim. (Input, Hidden)	32, 45	32, 75	32, 120	32, 150	32, 210
	Density (W, U)	0.2, 0.2	0.1, 0.2	0.1, 0.2	0.1, 0.3	0.1, 0.3
	Test Accuracy [Size]	0.471 [7.57KB]	0.515 [14.23KB]	0.541 [31.17KB]	0.572 [63.56KB]	0.587 [118.50KB]
Channel	Dim. (Input, Hidden)	32, 45	32, 60	32, 105	32, 150	32, 150
	Density (W, U)	0.2, 0.2	0.3, 0.3	0.3, 0.2	0.1, 0.3	0.1, 0.3
	Test Accuracy [Size]	0.482 [7.57KB]	0.533 [15.80KB]	0.553 [30.07KB]	0.589 [63.56KB]	0.589 [63.56KB]
Multi	Dim. (Input, Hidden)	32, 12	32, 20	32, 35	32, 55	32, 90
	Density (W, U)	1.0, 1.0	1.0, 1.0	0.3, 1.0	1.0, 1.0	0.3, 1.0
	Test Accuracy [Size]	0.447 [7.94KB]	0.447 [15.06KB]	0.527 [28.75KB]	0.558 [63.87KB]	0.558 [124.09KB]

TABLE II: Best models for FastGRNN. Dimensionalities for the Multi-FastGRNN are as for each FastGRNN unit.

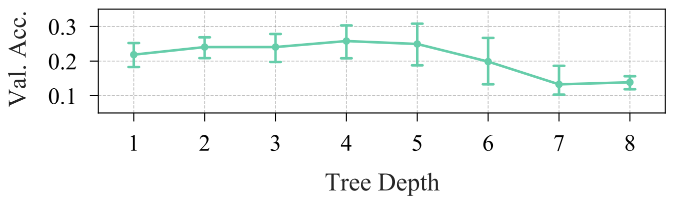

Fig. 1: Average validation accuracy of 129 Bonsai models for given tree depth, all within 128KB of memory.

D. ProtoNN

We carry out a grid search (as outlined in Section IV-D) by varying both the dimensionality of the projected space and the number of prototypes in $\{2, 4, 8, 16, 32, 64\}$ in order to obtain exponentially larger models. For the parameter γ [4], we use

Budget	$\leq$ 8KB	$\leq$ 16KB	$\leq$ 32KB	$\leq$ 64KB	$\leq$ 128KB
Depth, Dim.	5, 1	2, 3	2, 6	3, 11	5, 12
Test Accuracy	0.149	0.153	0.221	0.325	0.377
[Size]	[7.88KB]	[15.43KB]	[30.85KB]	[60.86KB]	[94.52KB]

TABLE III: Best Bonsai models for each memory budget.

1.5×10^n, where we range n from -4 to 4 in integer steps. We vary the parameter γ this extensively due the method's sensitivity to it [4]. For the sparsity constraint, we initially consider dense models, i.e. sparsity value 1.0. Finally, the learning rate is ranged in 0.1, 0.01 and 0.001 and we train for 100 epochs with early stopping with a patience of 10 epochs.

The models in this experiment ranged in size from 24.67KB to 805.38KB. The best model for the memory budget of 32KB had the learning rate, projection dimension, number of prototypes and γ parameters set to 0.01, 2, 16, and 1.5×10^{-4}, respectively. The size of this model was **25.34KB** and its test set accuracy was **0.127**. For the 64KB and 128KB budgets the best model had parameters 0.001, 4, 16, and 1.5×10^{-4} (order as above), its size was **50.05KB** and its test set accuracy was **0.142**. We find that out all of the models trained in this experiment, the best model with respect to validation accuracy was of size 787.02KB but still only obtained the final test accuracy of 0.145. That is, the performance of the algorithm only marginally increased even when the strict memory budget was disregarded. Given this result, intensive searching of models for the lower memory bounds is omitted, as well as tuning of the sparsity parameter, as compared to Direct Convolutions, this algorithm is non-competitive.

VI. Conclusion

In conclusion, we have seen that the algorithms presented in Section III vary wildly in how they adapt to the more complex task of classifying CIFAR-10 images. ProtoNN failed to achieve even 15% accuracy, which suggests that it struggles to keep up as the complexity of the task increases. Bonsai did fit the data but peaked at a modest 37.7% accuracy. Surprisingly, FastGRNN proved apt at multi-channel image classification, obtaining 58.9% test set accuracy with a footprint of 63.56KB. At the same time, FastGRNN also proved sensitive to the way that in which input image was turned into a time series, something which we devoted significant attention to in this analysis. Multi-FastGRNN consistently performs the worst for every size budget, and the channel-major models perform consistently the best, though only narrowly so for the 128KB budget. The fact that the channel-major models consistently beat out the row-major models suggests that FastGRNN finds inter-channel features to be more representative than intra-channel features for the CIFAR-10 data set. Multi-FastGRNN may then be lagging behind due to the final fully-connected layer not being able to combine the intra-channel features from each unit into descriptive inter-channel features.

Ultimately, CNNs using Direct Convolutions [8] dominate our analysis in this paper, obtaining a 65.7% test set accuracy with only 58.23KB of model memory usage.

Future work would extend the analysis presented in this paper to a wider set of image classification data sets to strengthen or disprove our conclusion that CNNs dominate image classification, even in the memory-constrained domain. Additionally, experiments investigating the latency and energy efficiency of each model would be of interest, given these are also important factors to consider in this domain.

References

[1] James Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *The Journal of Machine Learning Research*, 13:281–305, 03 2012.

[2] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks, 2016.

[3] Aditya Kusupati et al. FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network. In S. Bengio et al., editor, *Advances in Neural Information Processing Systems 31*, pages 9017–9028. Curran Associates, Inc., 2018.

[4] Chirag Gupta et al. ProtoNN: Compressed and Accurate kNN for Resource-Scarce Devices. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1331–1340, 2017.

[5] Don Kurian Dennis et al. EdgeML: Machine Learning for resource-constrained edge devices. Retrieved January 2020.

[6] Kyunghyun Cho et al. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[7] Albert Gural. Code for reproducing work of ICML 2019 paper: Memory-Optimal Direct Convolutions for Maximizing Classification Accuracy in Embedded Applications. Retrieved January 2020.

[8] Albert Gural and Boris Murmann. Memory-Optimal Direct Convolutions for Maximizing Classification Accuracy in Embedded Applications. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2515–2524, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[11] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, 04 2009.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[13] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1935–1944, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[14] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[15] Hellen Norman. Living on the Edge: Why On-Device ML is Here to Stay, Apr 2019.

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[17] J. Viega and H. Thompson. The State of Embedded-Device Security (Spoiler Alert: It's Bad). *IEEE Security Privacy*, 10(5):68–70, Sep. 2012.

[18] S. Yu, X. Wang, and R. Langar. Computation Offloading for Mobile Edge Computing: A Deep Learning Approach. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6, 2017.