Add a SideNet to your MainNet

Adrien Morisot

Independent Researcher

Abstract

As the performance and popularity of deep neural networks has increased, so too has their computational cost. There are many effective techniques for reducing a network's computational footprint–quantisation, pruning, knowledge distillation–, but these lead to models whose computational cost is the same regardless of their input. Our human reaction times vary with the complexity of the tasks we perform: easier tasks–e.g. telling apart dogs from boats–are executed much faster than harder ones–e.g. telling apart two similar-looking breeds of dogs. Driven by this observation, we develop a method for adaptive network complexity by attaching a small classification layer, which we call SideNet, to a large pretrained network, which we call MainNet. Given an input, the SideNet returns a classification if its confidence level, obtained via softmax, surpasses a user-determined threshold, and only passes it along to the large MainNet for further processing if its confidence is too low. This allows us to flexibly trade off the network's performance with its computational cost. Experimental results show that simple single hidden layer perceptron SideNets added onto pretrained ResNet and BERT MainNets allow for substantial decreases in compute with minimal drops in performance on image and text classification tasks.

1 Introduction

In recent years, neural networks have increased dramatically in size: [1] estimate a 300,000x growth in compute since 2012, with a doubling period of 3.4 months. Since the "bitter lesson" [28] of machine learning seems for now to be true, and performance on machine learning tasks appears to scale with model size and amount of training data [15], this trend is unlikely to decelerate anytime soon. Neural networks are increasingly used in industry to power various large-scale applications–from voice recognition (Google's Assistant and Apple's Siri are powered by neural networks [30, 14]) to image processing and natural language understanding–, so lowering the computational cost of these models at inference time is a pressing problem. There are ways of reducing the compute footprint of neural networks: pruning removes less important connections between neurons [5]. Quantisation reduces the number of bytes used by each of the network's parameters [12]. Knowledge distillation uses a larger network to train a smaller network on the large one's outputs [16].

These methods, although powerful, still lead to networks spending the same amount of compute on each input, regardless of the complexity of the input. Yet humans take different amounts of time to solve different tasks based on the complexity of the tasks. It is easier to quickly distinguish between a bear and a boat than it is to quickly distinguish between an Alaskan Malamute and a Siberian Husky. This observation gave rise to the field of conditional computation, in which neural network compute costs are diminished by not passing the input through the entire graph, but instead only a subset of it. This lets the network spend less compute time on easier inputs/tasks, and has the added benefit of allowing the network to be sensitive to computational budgets (if the budget is high, the network can afford to use more compute).

Existing implementations of conditional computation are generally complicated to engineer, and consequently are not used much in industry [2]. To remedy this, we propose the simplest model of conditional computation: attaching a single hidden layer perceptron, which we call SideNet, to an intermediate representation of a pretrained network, which we call MainNet. Unlike most existing

https://doi.org/10.7557/18.6286

conditional computation methods, the SideNet is straightforward to train, and attaching a SideNet to a MainNet is easy to engineer.

We also make three noteworthy observations: (i) When attached to the early intermediate representations of ResNets, the classification confidences of SideNets are calibrated, whereas the classification confidences of their ResNets are not. (ii) SideNet-based compute reduction can be complementary to knowledge-distillation and pruning: applying SideNets to DistilBERT [25], a heavily compressed transformer model, still yields noticeable performance savings ($\approx 30\%$) for a small drop in test accuracy ($\approx 0.5\%$). (iii) SideNets make it easy to explore compute-accuracy space, by making it continuous rather than discrete.

2 Related Work

Architectures similar to SideNets: [24] first run an image through a small convolutional neural network to ascertain whether or not it can be classified with high confidence. If it cannot, they send the image to a larger network, and use that classification as the final one. [4] build on this. They first run an image through a small AlexNet classifier [20], and a regression model determines the confidence level of the classification. If it is high, the classification is returned; otherwise, the image is sent through a GoogLeNet classifier [29], where the same regression is applied. If the confidence is still too low, it is sent through a ResNet [13], where a final classification is returned. Our method differs from these because in ours less computation is wasted: if the SideNet's confidence in its prediction is not high enough to return a classification, then the intermediary representation it used will continue flowing along the MainNet, and will not have to be recomputed from scratch.

[21] is the paper most resembling ours: they run an image through a main backbone network, along with multiple small classification networks along the backbone's side that interrupt the flow of the image through the main model if their confidence is high enough. They demonstrated that their method provided significant energy savings on a Raspberry Pi computer. [31] build on this, by using attention mechanisms with their side classification networks, and training them with knowledge-distillation and a genetic algorithm. Our method differs from these because it only uses one SideNet, which makes training substantially easier (training a network with multiple heads requires properly weighting the losses of each head, which is challenging).

There are a variety of other architectures involving conditional computation: [3] use reinforcement learning to learn a policy that directs an input only through discrete parts of a network, rather than the whole network. However, backpropagating through discrete random variables is inefficient and slow. [2] introduce a method to turn these discrete random variables continuous, to increase the rate of learning, and use it to train control networks, networks that control the amount of compute used at inference.

Studying intermediate representations: [20] find that early layers of convolutional neural networks mostly pick out simple textures and lines. This suggests that if an image is texturally simple or distinctive, it should be able to be classified in early parts of the network, rather than at the very end. [21] argue that this holds: their model was confident in its predictions when the input was fairly straightforward, and passed it off to the deeper model when it was more visually complex (e.g. the digit 1 is less complex than the italicised digit *1*, and was classified earlier in the network).

Similarly, in natural language processing, [6] find that early layers of BERT (a large transformer architecture by [8]) attend to broad features of an input, as opposed to later layers that tend to focus on a certain particular aspect of an input, and [22] find that [CLS] tokens are heavily overparameterised, and can be shrunk substantially without affecting performance.

3 Method

Mathematical framework. A neural network M, at a high level, is a function approximator. It maps inputs x to outputs y: $M(x) = y$. Supervised learning involves training the parameters of M to best fit the training data $(x, \tilde{y})$. We can decompose this mapping M into sub-components, and view it as a composition of transformations M_1, M_2, ..., M_n of the input x into intermediate representations x_1, x_2, ..., x_m.

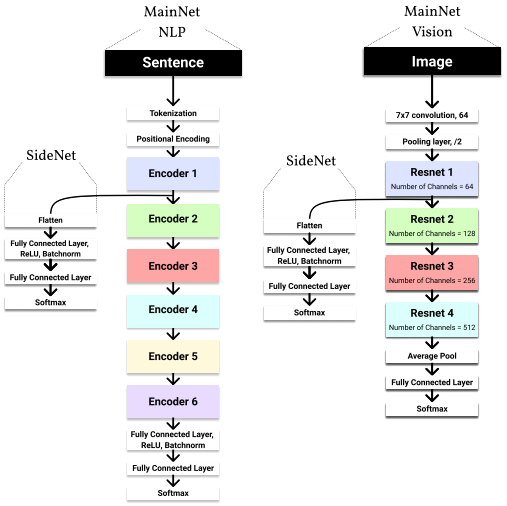

Figure 1: SideNets can be attached to a wide variety of networks. Here we visualise two networks equipped with SideNets. *Left:* A DistilBERT transformer. Its MainNet is untouched, but a SideNet is added between encoders 1 and 2. *Right:* A ResNet. Its MainNet is untouched, but a SideNet identical to the one for DistilBERT is added after its first block.

For simple architectures, like VGG [26], the compositions can be written simply, as below:

$x_1 = M_1(x)$,

$x_2 = M_2(x_1) = M_2 \circ M_1(x)$,

...

$y = x_n = M_n \circ M_{n-1} \ldots \circ M_2 \circ M_1(x)$,

where the M_i are convolutional layers, max-pooling layers, fully connected layers, and non-linear layers.

More complicated architectures are more involved to formalise, but can nonetheless still be made to fit this framework of intermediate representations. For example, if a layer of the network involves a skip connection from layer i to layer k, then we can write x_k and M_k as:

$x_k = M_k(x_{k-1}, x_i) = x_{k-1} + x_i$.

Architecture. We call the net M the MainNet. On top of this MainNet backbone, we propose adding a SideNet, a simple task-specific network S which takes as input one of the MainNet's intermediary representations x_c, and returns a probability distribution over the classes $y_c = S(x_c)$. In our experiments, we choose S to be extremely simple: a fully connected layer, a non-linear ReLU layer [23], a batch normalisation layer [17], a final fully connected layer, and a softmax layer (or a sigmoid layer in the case of binary classification). Although the softmax operation is not a true reflection of the model's confidence [9], we find that using it as a proxy for model confidence works well empirically.

SideNets can be attached to any intermediate representation x_i; in Figure 1 we illustrate two possible locations for SideNets on two different architectures: the DistilBERT transformer for natural language processing and the ResNet for computer vision.

Training SideNets. To train the SideNet quickly, we can freeze the weights of the MainNet, and update the SideNet's weights on the normal training data. The SideNet, by construction, has very few parameters, and the input data only needs to flow through a small fraction of the MainNet to get to the SideNet, so the optimisation is fast and converges quickly. Multiple SideNets S_1, ..., S_p with parameters W_1, ..., W_p can be trained in parallel at different points along the MainNet, as long as they return separate losses L_{S1}, ..., L_{Sp}, since by construction $\frac{\partial W_i}{\partial L_j} = 0, \forall i \neq j$, provided that the SideNets remain independent of each other. While training this way is significantly faster, it does come with a significant performance cost (on the order of 3% in our experiments), so in performance-critical models, fine-tuning the whole model with the SideNet is preferable.

To fine-tune the weights of the MainNet alongside those of the SideNet, we can backpropagate over the weighted sum of their losses. If the MainNet's loss is L_M and the SideNet's loss is L_S, then we can backpropagate over a loss $L = L_M + \alpha L_S$. In our experiments we always pick $\alpha = 1$. To fine-tune the weights of the MainNet alongside those of multiple SideNets, each with losses L_{S1}, ..., L_{Sp}, the same principle applies.

SideNets at inference time. To classify an input image x, we run x through the MainNet until we obtain the intermediary representation x_c, then pass x_c through S to obtain a classification $\hat{y}$ and confidence level $\hat{p}$. If the confidence level exceeds a threshold θ, then the classification is returned immediately, without having x_c pass through the rest of the MainNet. If the confidence level is below θ, then x_c is passed back to the MainNet, where it

returns a final classification y.

4 Classification Experiments

We perform all experiments on a single NVIDIA RTX 2070 GPU. All experiments use the Adam [18] optimiser, with default parameters. We use an initial learning rate of .0003 and train for 50 epochs in the ResNet experiments; we use an initial learning rate of .000003 and train for 20 epochs in the BERT and DistilBERT experiments. In both cases, we use a learning rate decay of 3 after 5 epochs in which the validation loss doesn't go down.

For all experiments, our SideNet is a single hidden layer perceptron, with an input size equal to the number of elements in x_c (a flattened version of x_c for images), a hidden layer with 32 units,[1] a batchnorm layer, a ReLU layer, and a classification layer (softmax for multi-class classification, sigmoid for binary).

CIFAR10. We assess our method's performance on the CIFAR10 dataset [19], a dataset of 60,000 colour images, 32x32 pixels, with 10 classes of 6,000 elements each. We use ResNet18, 34, 50, and 152 (with weights pretrained on ImageNet) as the core architecture of the MainNet. Since they were pretrained on ImageNet, which has 1,000 classes, we replace their final fully connected layer with a fully connected layer with the same architecture as a SideNet, described above. We attach the SideNet to the output of the Resnet 1 block illustrated in Figure 1. The SideNet is fine-tuned with the last layer of the MainNet.

We evaluate our method on the test set with different thresholds θ by plotting the model's accuracy with respect to the amount of compute used. We use the average number of parameters used for a single input as a proxy for the amount of compute used (since this number stays fixed, whereas the average number of floating point operations would vary based on the size of the input). The results are plotted in Figure 2. We find that architectures using SideNets can use significantly less compute than architectures without SideNets, and still maintain the same accuracy. We also note that adding a SideNet makes it easy and cheap to explore the space of models with different compute and accuracy levels: simply adjust the threshold θ. In order to explore this same compute-accuracy space with knowledge-distillation or pruning, we would have to repeatedly do so from scratch.

[1] We found that increasing this number did not have much of an effect.

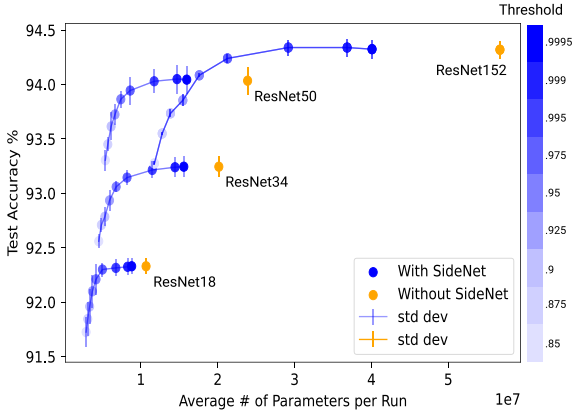

Figure 2: Plot of test accuracy with respect to average number of parameters per run, for different thresholds θ, for different depths of ResNets, with and without SideNets. The results are averaged over 5 runs, with error bars indicating standard deviations.

We also test the SideNet's calibration. A classification model is calibrated when the probability p it assigns to an input x belonging to a certain class is equal to the actual probability of the model classifying it correctly. For example, if a weather model predicts every day for 100 days that it will be sunny with 75% certainty, and at the end of the 100 days there were indeed 75 sunny days, then that model is calibrated. More formally, given an input x whose true classification label is y, if a model M assigns to x a classification of $\hat{y}$ with confidence $\hat{p}$, then M is calibrated iff $\mathbb{P}(\hat{y} = y|\hat{p} = p) = p, \forall p \in [0, 1]$. Calibration is a useful property for a model to have, since it "knows what it doesn't know". We quantify calibration using the expected calibration error (ECE). We first bin our predictions into 8 equally spaced classification confidence bins, consisting of n_i predictions each: confidences between 0.2 and 0.3 go into bin 1, ..., confidences between 0.9 and 1 go into bin 8 (there are no bins between 0.1 and 0.2 because in our experiments both SideNets and MainNets always have confidence above 0.2). The

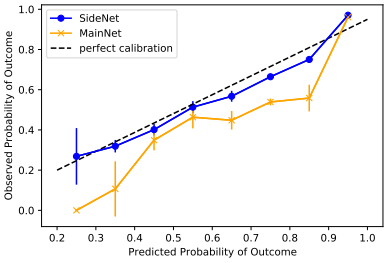

Figure 3: Calibration plot of a ResNet50's SideNet and MainNet, averaged over 5 runs, with standard deviations. The SideNet's classifications are significantly closer to perfect calibration than those of the MainNet. The results were obtained on the test set.

Table 1: ECE scores for different SideNets and MainNets, evaluated on the test set. The lower the ECE, the more calibrated the model. The values are averaged over 5 runs, and include standard deviations.

	SideNet ECE	MainNet ECE
ResNet152	**.30** $\pm$ .06	1.1 $\pm$ .18
ResNet50	**.41** $\pm$ .10	.91 $\pm$ .17
ResNet34	**.38** $\pm$ .08	1.0 $\pm$ .14
ResNet18	**.31** $\pm$ .08	1.0 $\pm$ .05

ECE is computed by calculating the average distance between confidence and accuracy for each bin: ECE=$\sum_i \frac{n_i}{n}|acc(i) - conf(i)|$, $i = 1, \ldots, 8$.

[11] find that deep convolutional neural networks are not calibrated. We reproduce their results, and find that our MainNet classifications are not calibrated, with high ECE scores. However, the classifications of our SideNets are well calibrated. Table 1 details the ECE scores for SideNets and MainNets, and Figure 3 gives a specific example of how the MainNet is uncalibrated relative to the SideNet. This is helpful for setting the confidence threshold θ: it means that setting $\theta = 0.85$ will lead to a SideNet with minimum accuracy 85%.

SST2. To assess our method's performance on natural language processing tasks, we apply it to the SST-2 dataset [27], a dataset of 9613 movie reviews, labelled as positive or negative. Our train/validation/test split is 5000/1613/3000.

We use pretrained DistilBERT [25], BERT-base, and BERT-large [8] models as the core architectures of the MainNet. For the MainNet's final classification layer, we add a fully connected layer with the same architecture as a SideNet. We attach DistilBERT's SideNet after its first transformer block (out of 6), as in Figure 1; we attach BERT-base's SideNet after its fourth transformer block (out of 12), and BERT-large's after its eighth encoder block (out of 24). The SideNet is fine-tuned along with the last layer of the MainNet.

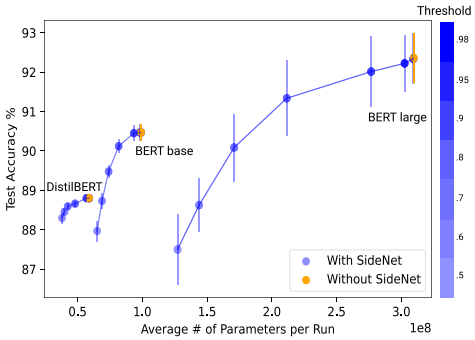

Figure 4: Plot of test accuracy with respect to average number of parameters per run, for different thresholds θ, for different transformer models, with and without SideNets. The results are averaged over 5 runs, with error bars indicating standard deviations.

BERT-base and DistilBERT use 768 dimensional tensors to represent each token, and so the total parameter count overhead of the SideNet is $768 \times 32 + 32 \times 1 \approx 25000$, which is $\approx 0.03\%$ of BERT-base's $\approx$100M parameter count, and $\approx 0.04\%$ of DistilBERT's $\approx$60M parameter count. BERT-large uses 1024 dimensional tensors to represent each token, so its overhead is $1024 \times 32 + 32 \times 1 \approx 35000$, which is $\approx 0.01\%$ of BERT-large's $\approx$300M parameter count.

We evaluate on the test set with different thresholds θ, and plot the results in Figure 4, with the same methodology as with Figure 2. We find that adding SideNets allows for substantial decreases in compute, albeit with a greater loss in accuracy than the CIFAR10 example above. However, as above, we note that the addition of SideNets allows for a much easier exploration of compute-accuracy space. If we wanted a model with 200M parameters, rather than the 300M of BERT-large or the

100M of BERT-base, then rather than train that 200M parameter model from scratch, we could easily attach a SideNet to a pretrained BERT-large, and get a model that on average uses 200M parameters per run, with an accuracy above BERT-base, but below BERT-large.

Furthermore, it is worth highlighting that adding a SideNet to DistilBERT manages to reduce its average parameter use by 30%, at a cost of 0.5% test accuracy, despite it already being a version of BERT-base that was compressed using extensive model pruning and knowledge distillation. In comparison, DistilBERT lost 1.4% test accuracy on the SST-2 task after losing 40% of its parameters. This suggests that adding SideNets is a compute reduction method that can effectively complement knowledge distillation and model pruning.

As in the CIFAR10 case, we found that the SideNets were calibrated. However, we found that the pretrained transformers were also calibrated, duplicating the findings of [7].

Does the SideNet lower the MainNet's accuracy? It could be argued that the addition of the SideNet to the training task would lead to a decrease in the final accuracy of the MainNet, since the training procedure splits its attention between minimising the SideNet and the MainNet's loss. We find that this is not the case, and that adding the SideNet does not seem to have a negative effect on the MainNet's accuracy. Our findings are summarised in Table 2. Anecdotally, we find that ensembling the SideNet and MainNet predictions provided a slight boost to final accuracy over just the MainNet's predictions.

Table 2: Test accuracies for final MainNet classifications when trained with and without a SideNet, on computer vision (CIFAR10) and natural language processing (SST-2) classification tasks. The values are averaged over 5 runs, including standard deviations.

	ResNet18	ResNet34
SideNet	$92.3 \pm .1$	$93.2 \pm .1$
No SideNet	$92.1 \pm .1$	$92.9 \pm .2$

	ResNet50	ResNet152
SideNet	$94.0 \pm .2$	$94.3 \pm .1$
No SideNet	$93.6 \pm .3$	$93.9 \pm .2$

	DistilBERT	BERT-base	BERT-large
SideNet	$88.8 \pm .1$	$90.6 \pm .2$	92.2 ± 1.2
No SideNet	$89.0 \pm .1$	$90.6 \pm .3$	92.8 ± 0.5

5 Conclusion & Future Work

In this work we propose attaching a SideNet, a small single hidden layer perceptron, onto the intermediate representations of a MainNet, a large pretrained network, and using the SideNet's confidence level to determine whether an input should be classified by the SideNet or passed back to the MainNet. SideNets are easy to implement, fine-tune, and deploy, and provide substantial compute savings at little cost to model accuracy, for both natural language processing and computer vision tasks.

We also find that SideNets in the early layers of ResNets are calibrated, while the ResNets themselves are not, and that SideNets can significantly reduce the amount of compute used by DistilBERT at minimal cost to accuracy, despite DistilBERT already being a highly compressed model. Finally, increasing or decreasing the threshold θ for the model's confidence allows us to painlessly explore compute-accuracy space, by making continuous what was once discrete.

SideNets open several avenues for further study:

1. SideNets perform well on classification tasks. Do they perform equally well on more complicated, higher dimensional tasks, such as image segmentation or machine translation?

2. SideNets help reduce DistilBERT's total compute, with minimal loss in accuracy, even though DistilBERT is already a highly compressed model. What is the interplay between different forms of model compression, and to what extent can they be combined?

3. SideNets are small and shallow networks. Does this make them more susceptible to being fooled by adversarial attacks [10]?

We hope to investigate these questions further in future work.

References

[1] D. Amodei and D. Hernandez. AI and compute. https://blog.openai.com/ai-and-compute, 2018.

[2] A. Bapna, N. Arivazhagan, and O. Firat. Controlling computation versus quality for neural sequence models. *arXiv preprint arXiv:2002.07106*, 2020.

[3] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[4] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 527–536. JMLR. org, 2017.

[5] G. Castellano, A. M. Fanelli, and M. Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531, 1997.

[6] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[7] S. Desai and G. Durrett. Calibration of pre-trained transformers. *arXiv preprint arXiv:2003.07892*, 2020.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Y. Gal. Uncertainty in deep learning. *University of Cambridge*, 1:3, 2016.

[10] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[11] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017.

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

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, and et al. Streaming end-to-end speech recognition for mobile devices. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019. doi: 10.1109/icassp.2019.8682336. URL http://dx.doi.org/10.1109/ICASSP.2019.8682336.

[15] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. Patwary, M. Ali, Y. Yang, and Y. Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

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

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] S. Leroux, S. Bohez, E. De Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, and

B. Dhoedt. The cascading neural network: building the internet of smart things. *Knowledge and Information Systems*, 52(3):791–814, 2017.

[22] A. Matton and L. de Oliveira. Emergent properties of finetuned language representation models, 2019.

[23] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[24] E. Park, D. Kim, S. Kim, Y. Kim, G. Kim, S. Yoon, and S. Yoo. Big/little deep neural network for ultra low power inference. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 124–132, 2015.

[25] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[27] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[28] R. S. Sutton. The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`, 2019.

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL `http://arxiv.org/abs/1409.4842`.

[30] S. Team. Hey siri: An on-device dnn-powered voice trigger for apple's personal assistant. *Apple Machine Learning Journal*, 1(6), 2017.

[31] L. Zhang, Z. Tan, J. Song, J. Chen, C. Bao, and K. Ma. Scan: A scalable neural networks framework towards compact and efficient models. In *Advances in Neural Information Processing Systems*, pages 4029–4038, 2019.