Activation Sparsity and Dynamic Pruning for Split Computing in Edge AI

Janek Haberer
jha@informatik.uni-kiel.de
Kiel University, Germany

Olaf Landsiedel
ol@informatik.uni-kiel.de
Kiel University, Germany &
Chalmers University of Technology, Sweden

ABSTRACT

Deep neural networks are getting larger and, therefore, harder to deploy on constrained IoT devices. Split computing provides a solution by splitting a network and placing the first few layers on the IoT device. The output of these layers is transmitted to the cloud where inference continues. Earlier works indicate a degree of high sparsity in intermediate activation outputs, this paper analyzes and exploits activation sparsity to reduce the network communication overhead when transmitting intermediate data to the cloud. Specifically, we analyze the intermediate activations of two early layers in ResNet-50 on CIFAR-10 and ImageNet, focusing on sparsity to guide the process of choosing a splitting point. We employ dynamic pruning of activations and feature maps and find that sparsity is very dependent on the size of a layer, and weights do not correlate with activation sparsity in convolutional layers. Additionally, we show that sparse intermediate outputs can be compressed by a factor of 3.3× at an accuracy loss of 1.1% without any fine-tuning. When adding fine-tuning, the compression factor increases up to 14× at a total accuracy loss of 1%.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Human-centered computing** → *Ubiquitous and mobile computing*.

KEYWORDS

Deep Learning, Activation Sparsity, Feature Map Pruning, Edge Computing, Offloading

ACM Reference Format:
Janek Haberer and Olaf Landsiedel. 2022. Activation Sparsity and Dynamic Pruning for Split Computing in Edge AI. In *DistributedML (DistributedML '22), December 9, 2022, Roma, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3565010.3569066

1 INTRODUCTION

Split computing executes the first layers of inference on a constrained IoT or edge device and then sends intermediate results, i.e., activations, to the cloud to complete the remaining layers on a more powerful device [6]. Sending the intermediate output instead of the initial input increases privacy [11] and can reduce network communication [21]. However, if done naively, it results in a significant communication overhead in terms of energy consumption and transmission time: In most convolutional architectures, intermediate data increases with each layer, see Figure 1. With each layer, the number of feature maps increases, which in turn increases the amount of data sent to the cloud when splitting networks. Often, this intermediate output is significantly larger than the original input. Commonly, the amount of intermediate data only reduces in the final layers.

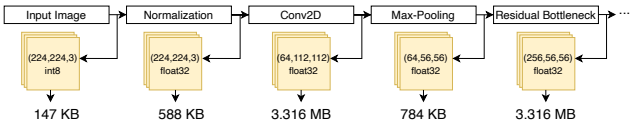


Figure 1: ResNet-50 architecture showing data sizes between the first few layers. While the input image amounts to 147 KB, data size between layers increases up to 3.136 MB after the first Residual Bottleneck. Afterward, halving with every first Residual Bottleneck of 3 remaining ResNet layers.

While activations change for each input, literature suggests that many activations in intermediate layers are close to zero [1, 8, 13]. Thus, selecting a layer with a significant degree of activation sparsity as a splitting point and removing values close to zero reduces the communication overhead. This paper analyzes the sparsity of activations and feature maps in different layers to guide the search for optimal splitting points in networks. Due to space limitations, we focus on two selected layers in ResNet-50 [4], the first max-pooling layer and the first Residual Bottleneck. On the one hand, we choose these two representative layers as they have significant differences in size. On the other hand, they are early layers, and in split computing, we commonly want to hand off the computation within the first layers.

Our results guide the identification of optimal splitting points in terms of sparsity. We introduce dynamic pruning of activations and entire feature maps, see Section 3 for details, and reveal that they reduce the amount of data transmitted with minimal impact on the model accuracy. Further, we show that fine-tuning, i.e., re-training for a couple of epochs after adding the split point, significantly limits the impact of dropping near-zero activations at the splitting point. We show that by exploiting activation sparsity, we reduce the amount of transmitted data by up to 3.3× with an accuracy drop of 1.1% without fine-tuning and up to 14× with an accuracy drop of

DistributedML '22, December 9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9922-7/22/12…$15.00
https://doi.org/10.1145/3565010.3569066

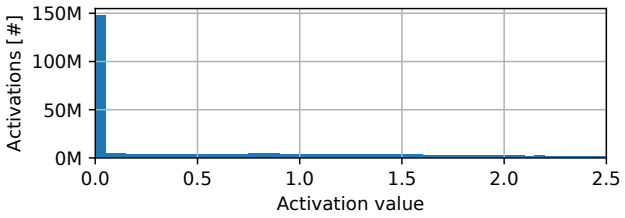


Figure 2: Histogram of every activation value in a convolutional layer generated from CIFAR-10 test set using ResNet-50. The first bin contains a large spike, containing 46.19% near zero values.

1% with fine-tuning. We also evaluate dropping entire feature maps in a convolutional network for comparison. Dynamic feature map pruning provides compression by a factor of 3.3× with an accuracy loss of 1.5% with fine-tuning.

This paper makes the following contributions:

- We analyze the sparsity of activations and guide the identification of optimal splitting points in terms of sparsity.
- We use dynamic activation pruning with and without fine-tuning to compress intermediate data by a factor of up to 14× at an accuracy loss of 1%.
- We conduct an analysis of feature maps and show that there is no strong correlation to the corresponding weights.
- We use dynamic feature map pruning with and without fine-tuning to compress intermediate data by a factor of up to 3.3× at an accuracy loss of 1.5%.

2 BACKGROUND AND RELATED WORK

This section briefly introduces weight sparsity, activation sparsity, and split computing and also discusses related work.

Weight Sparsity. Sparsity describes the proportion of the values equal to or close to zero in the output for a given input. Previous work shows that fully connected layers have high sparsity in weights, resulting in a great pruning potential [5, 22, 23]. As each connection has an associated weight, memory usage can be decreased by a significant amount. Using specialized regularization techniques can induce even more sparsity in neural networks [8, 24]. In convolutional layers, kernels provide a tool to share weights among multiple inputs and outputs. This presents some challenges in using sparsity effectively. Many approaches use kernel sparsity to reduce the number of weights in a kernel [2, 9, 16, 18].

Activation Sparsity. In contrast, others use sparsity on intermediate outputs, also called activation sparsity [12, 13, 20]. In activation sparsity, intermediate outputs are represented as sparse matrices to reduce memory usage and multiply-and-accumulate operations, thereby providing speed-up.

Figure 2 shows a histogram of activation values for a ResNet architecture trained with CIFAR-10 [7]. 46.19% of all activation values are between 0 and 0.05. While the works mentioned earlier apply sparsity to each activation value, Yang et al. [19] apply the concept of pruning using sparsity to feature maps as a whole. Using the maximum of a feature map as a characteristic value, they create a ranking of importance for all feature maps. Feature maps with low importance, i.e., low values, are removed. However, they apply their technique to entire networks while we focus on identifying single layers with a high fraction of sparse feature maps.

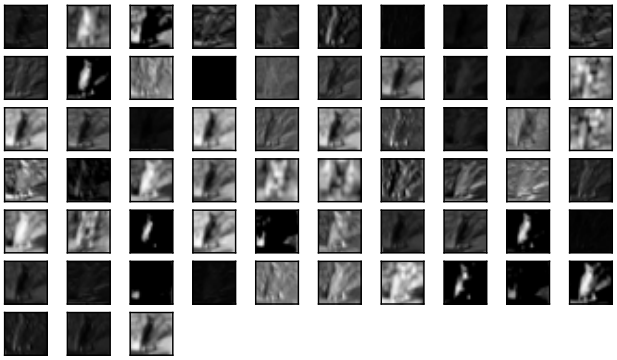


Figure 3: Visualized feature maps from an early intermediate layer. Some feature maps are completely black, indicating that we can completely remove them from any further computation. Others contain large black areas with only small white details, where we can use a sparse representation to compress feature maps.

Split Computing. For this technique, a constrained node, e.g., a sensor node, works together with a powerful edge device or the cloud. The sensor gets some sensor inputs and conducts parts of the inference, i.e., a few of the first layers, as it cannot deploy the whole network. It then sends the intermediate result, instead of the input, to the edge or cloud device, as intermediate results preserve some privacy [11] and can reduce network communication.

Figure 3 shows the intermediate feature maps of an early layer in a network. Near-black images indicate the amount of data with no impact on further computation, creating opportunities to reduce the amount of necessary data. Some works use bottleneck layers to reduce the amount of data to be sent [3, 10, 11]. For example, Yao et al. [21] used an asymmetric autoencoder to reduce the amount of computation necessary on the sensor node with a large decoder structure on the edge to reconstruct. However, they need an additional training phase, requiring some time and finding the correct architecture. While autoencoders are a powerful tool for compression, we argue that these works omit a critical step: evaluating activation sparsity in today's network architectures as an enabler for their compression. In this work, we close this gap.

3 DESIGN

This section details our design and presents how we use dynamic activation pruning and dynamic feature map pruning in split computing scenarios. In addition, we introduce fine-tuning to improve performance.

3.1 Dynamic Activation Pruning

Many pruning techniques rely on sparsity. However, they usually rely on weight sparsity instead of activation sparsity, as weights are static and do not change during inference, so we can permanently remove connections. While this reduces model size and memory

usage while speeding up inference, it does not help as much when we want to split a network and reduce network usage. This is because pruning permanently removes connections. It cannot adapt to changing inputs and instead applies the same calculation for each image.

Therefore, we employ dynamic activation pruning based on sparsity which depends on the particular input, to reduce network communication and memory usage. Instead of keeping all the output after a convolutional layer, we identify values close to zero and remove them. To keep track of where the values were, we use a simple bitmap where a bit indicates for each position whether it is a removed value or not. On the server side, we can reconstruct the positions of all transmitted values using the bitmap and set the remaining values to zero. Because activations rely only on weights and input, and weights are constant during inference, inputs directly influence activations. This means that dynamic activation pruning is dynamic and tailored to the input, contrary to weight pruning. Although activation sparsity does not help much when pruning, as accelerators are not designed for dynamic missing values in a feature map, it does help to reduce network communication, which is our goal here.

3.2 Dynamic Feature Map Pruning

Figure 3 shows feature maps for one input image in one of the first layers of ResNet-50. The values are colored according to a grey scale, where black represents the value zero. We can see a notable amount of entirely black feature maps and, in general, a significant amount of black areas in most feature maps, which have no impact on the network's inference. In dense layers, for example, one can decide whether to prune for each connection. In contrast, with convolutional layers, one can remove either single values from the kernels or whole kernels. The former requires sparse matrix calculations to gain anything for pruning. Although the result is sparse and helps to reduce network communication, it suffers the same issues as pruning individual connections in a dense layer since it only considers weights and not activations. For removing kernels, it is a similar story. We can remove kernels and, thereby, feature maps. However, this uses only weights and is, therefore, static.

Instead, this paper uses a metric to estimate the importance of feature maps and then decide which feature maps to keep based on that criteria. We focus on a feature map's mean and maximum as metrics. The idea is that the maximum of a feature map determines the maximum impact of the feature map on the next layer. Therefore, a small maximum means a negligible impact on the next layer, as all activations within the map are small. On the other hand, a big maximum value indicates at least some value with a high impact on the next layer. Then, the process is similar to dynamic activation pruning. We first calculate the metric's values for each feature map and then use a threshold to remove feature maps with a value below the threshold. On the server side, we merely need to know the positions of the dropped feature map, which we encode as a bitmap. As a result, we now remove irrelevant channels in each intermediate data output and thereby reduce network communication.

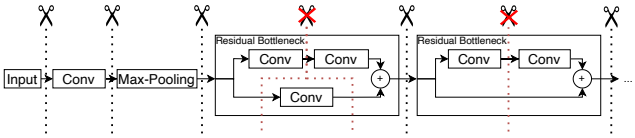


Figure 4: First few layers of ResNet-50 with potential splitting points. While the first three cuts and those before or after a Residual Bottleneck are good options, it is possible to cut through a block. This, however, means we have to keep track of two sets of feature maps, which cannot help us reduce the total size.

3.3 Challenges

Splitting Points. One challenge with finding a splitting point within a network is that many modern networks use parallel layers. For instance, ResNet-50 introduced skip connections. Inception [15] uses five types of convolutions in parallel and then combines them. And with attention mechanisms [**?**] and transformers [17] even more parallelity is introduced to networks. However, parallel computations pose a problem splitting a network as they significantly increase the amount of data. Therefore, when deciding where to split a network, we need to consider this and split at points where these parallel branches are combined. Figure 4 shows the first few layers of ResNet-50 and potential splitting points. The cuts inside Residual Bottlenecks are the major issue here as they have parallel computations, thus creating additional data we would have to transmit. Therefore, we only cut between Residual Bottlenecks, i.e., where the parallel layers have been combined. While other popular networks introduce even more parallel branches, they still combine these branches every few layers resulting in enough choices for a splitting point instead of trying to split through parallel branches.

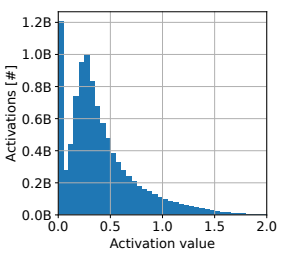
Fine-Tuning. Another challenge is that, as we show in our evaluation (see Section 4), removing values close to but not exactly zero impacts the network's accuracy. Networks often over-provision and have more parameters than necessary for the problem. In those cases, we can remove activations without any impact on the accuracy up to a certain threshold. Thus, the more aggressively we remove, the more the accuracy gets negatively affected. Other networks are tailored very well to a problem, i.e., just enough parameters to solve it. In this case, the performance directly suffers when we remove parameters. To enable our approach of activation pruning even when a network is tuned for a problem, we introduce a fine-tuning phase, during which the splitting technique is employed so that the network can adapt and performance can be restored.

4 EVALUATING DYNAMIC PRUNING

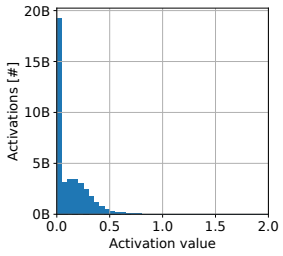
First, this section analyzes activation sparsity followed by dynamic activation pruning, then it analyzes feature maps followed by dynamic feature map pruning.

4.1 Experiment Setup

For the analysis, we focus on ResNet-50 and inspect the intermediate outputs for the dataset ImageNet [14]. We use pre-trained



(a) Pixel activations after first max-pooling layer



(b) Pixel activations after first Residual Bottleneck

Figure 5: Histogram of every activation value after the first max-pooling layer and after the first Residual Bottleneck in ResNet-50 generated from ImageNet validation set. In both cases, the leftmost bin containing values close to zero is the largest. However, this bin contains only 12% of all activations in the max-pooling layer.

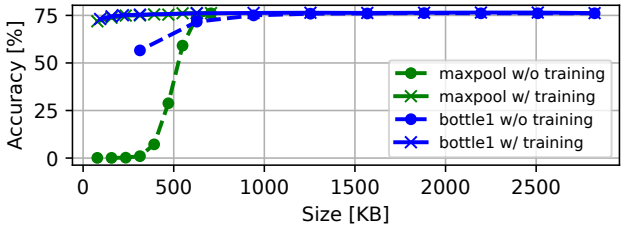


Figure 6: Effect of dynamic activation pruning on compressed size and corresponding accuracy with and without fine-tuning. When compression reaches a size of 750 KB or less, accuracy drops significantly without fine-tuning. Fine-tuning manages to prevent that accuracy drop until trying to compress to less than 300 KB.

networks for the initial analysis and then compare them to networks after fine-tuning. We use the validation set of 50,000 images to gather data and the whole training set for fine-tuning.

We use PyTorch, and first, we evaluate and analyze the model on the given datasets for sparsity. Then we prune according to activation sparsity in ResNet and apply fine-tuning. We extended the ResNet implementation to be able to set specific activations to zero or sort feature maps using a given metric and then set a given fraction of feature maps to zero. Our implementation acts similarly to a dropout layer: we remove a fraction of feature maps according to their sparsity and, during fine-tuning, allow the model to learn to cope with the loss of feature maps.

4.2 Dynamic Activation Pruning

This section discusses our evaluation results on analyzing activation sparsity and the impact of removing individual activation values on a network's accuracy.

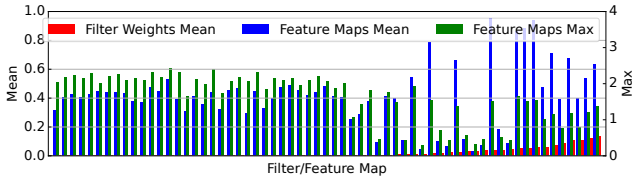
4.2.1 Activation Sparsity. Figure 5 shows the activation values at two selected cut-off points in ResNet-50 for the ImageNet validation set, namely after the first max-pooling and after the first residual layer, respectively. The chosen bin size for values is 0.05, i.e., the first bar shows the number of values between 0 and 0.05. One main difference between the two cut-off points is the number of pixel activations as the shape of the data after the first max-pooling layer is (64, 56, 56), totaling 784 KB, whereas one Residual Bottleneck later, the shape is (256, 56, 56), which totals 3.136 MB. The second major difference is the relative distribution of activation values. Figure 5a shows a high number of activations near zero after the max-pooling layer. However, compared to the total number of activations, only 12% of all activations are near zero. Figure 5b shows that after the first Residual Bottleneck 48% of all activation values are near zero. This means there is a large relative gain in sparsity when computing a few more layers. However, we must remember that the number of values also increases by 4x. So even though there is more relative sparsity after the max-pool layer, the absolute amount of non-zero values after the first Residual Bottleneck is still more than all values combined after the max-pooling layer. Therefore, when we consider, for example, 0.05 as the threshold for non-zero values, we would choose the max-pooling layer as a splitting point, as much fewer non-zero values remain.

Next, we consider increasing the threshold that determines what is considered a near-zero value. If we increase it from 0.05 to 0.1 and 0.15, the relative amount of near-zero values increases to 14.8% and 19.2%, respectively, after the max-pooling layer. After the Residual Bottleneck, we see an increase to 56% and 64.5%, respectively. Overall, there are more values close to zero after the Residual Bottleneck. However, for compression, the max-pooling layer poses a better splitting point, as the amount of non-zero values is smaller.

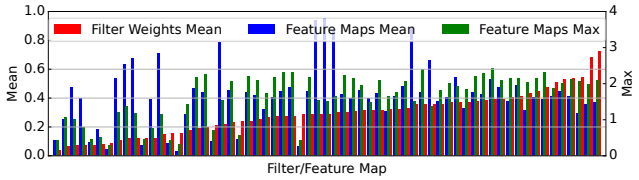
4.2.2 Dynamic Activation Pruning. Figure 6 shows the differences in accuracy for the max-pooling layer, called *maxpool*, and the first Residual Bottleneck, called *bottle1*, when we employ dynamic activation pruning. With 90% data removed from the *bottle1* intermediate outputs and without fine-tuning, there is still 313 KB of data left to transmit, which – in combination with the experienced accuracy drop – is not a suitable option. With the removal of 60% or 70%, we only lose up to around 1% in accuracy compared to the baseline accuracy of 76.1%, leaving us with 1,254 KB and 940 KB, respectively. The max-pooling layer achieves 75.7% accuracy with 705 KB of data left, which drops to 71% with 627 KB of data and then to 24.5% with 548 KB.

Next, we apply fine-tuning to evaluate how an additional training phase after adding dynamic activation pruning to the network can recover losses. We use a pre-trained ResNet-50 and, during training, apply dynamic activation pruning for different thresholds, determining what values get set to zero. As a result, the accuracy increases for each threshold in both layers, see Figure 6. For smaller thresholds, the accuracy increases only slightly, but starting at a compressed size of 548 KB, the difference in accuracy is more than 15%. And – most importantly – while trying to compress below 500 KB without any additional training destroys the inference accuracy completely. However, training manages to recover most of it, achieving at least 70% accuracy until a compressed size of 78 KB.

Overall, our results show that the chosen layer can make a huge difference in the possible compression using sparsity in a splitting



(a) Filter weights of the layer directly after the shown feature maps.



(b) Filter weights of the layer directly before the shown feature maps.

Figure 7: Each red, blue, and green bar are corresponding filter and feature map. We compare the feature map metrics with incoming and outgoing weights and find no strong correlation between feature map metrics and weights.

scenario without training. In scenarios where we use an additional training phase, however, accuracies for both layers are very comparable at similar compression sizes. Despite their huge size difference, this suggests a large degree of unnecessary information in bigger layers.

4.3 Dynamic Feature Map Pruning

Next, we evaluate the impact of removing whole feature maps instead of single values. We set feature maps to zero, i.e., all values within a feature map, based on their magnitude, determined by their maximum value, as introduced in Section 3.

4.3.1 Feature Map Analysis. Figure 7 shows for the *maxpool* layer each feature map maximum and mean with the mean of the corresponding absolute weights, sorted by the weights means. Figure 7a shows the incoming weights with the respective feature map they create, while Figure 7b shows the outgoing weights of the next layer. Since each kernel of the following layer has a channel for each feature map, we average over the weights of the respective channel of all kernels. For both cases, we can see that rising weights do not mean that activations are increasing, i.e., there is no direct correlation. Looking at the weights of the previous layer in Figure 7a, we first notice many near-zero kernels. Despite this, the means and maxima of the feature maps of these kernels are greater than some means and maxima of feature maps with higher average kernel weights. Similarly, when looking at the weights of the next layer in Figure 7b, we can see the occurrence of higher feature map means and maxima increasing with greater kernel weights. However, some feature maps with high mean and maximum with low corresponding kernel weights still exist. As the feature map mean and maximum indicate the impact of the feature map on the following layers, it is insufficient to only take kernel weights into account when deciding what feature maps or kernels to keep. In the case of looking at the incoming kernel weights, there is only a very low correlation. In the case of looking at the outgoing weights, where there is a correlation, we would still miss some higher activation feature maps when only relying on kernel weights. Overall, weights are not a good option to determine which feature maps have a low impact on classification for different inputs.

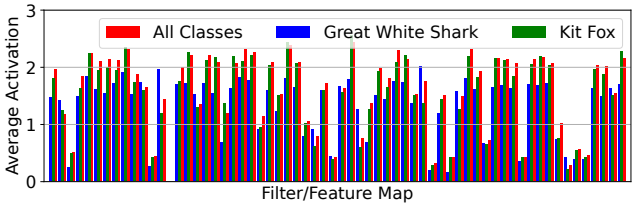


Figure 8: Average maximum values of feature maps for the classes *great white shark* and *kit fox* compared to the average of all classes. While *kit fox*'s average maxima are similar to the average of all classes, the maxima of *great white shark* show a noticeable difference between classes.

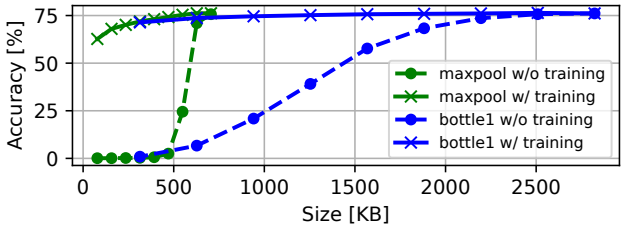


Figure 9: Effect of dynamic feature map pruning on compressed size and corresponding accuracy with and without fine-tuning. When removing more than 20% of feature maps, accuracy drops significantly without fine-tuning.

Figure 8 shows the average maximum over the full ImageNet validation set for each feature map compared to the average maximum over the validation images of two classes. Specifically, we show the average maxima for two selected classes of ImageNet, *kit fox* and *great white shark*. We can see that most of the kit fox's feature map maxima are very similar to the average maxima of all classes. However, the maxima of the great white shark's feature maps differ significantly from the all classes average. This indicates that activations are very dependent on their class. Whereas most images contained within ImageNet are pictures on land, only a fraction of the images are underwater images. However, in the case of the class *great white shark*, there are underwater images mixed with images taken above water. Overall, feature maps differ greatly across different classes, warranting a solution based on activations.

4.3.2 Dynamic Feature Map Pruning. Figure 9 shows the change in accuracy when we set feature maps to zero without fine-tuning in the same two layers, *maxpool* and *bottle1* as before. Compared to Figure 6, the accuracy drops considerably more in both layers without training. For any given amount of removed values or feature maps, the accuracy is lower when we remove feature maps. This is because a feature map can contain sparsity while having a high

maximum value. We cannot retain these specific non-sparse values when removing entire feature maps.

While accuracy drops significantly in the max-pooling layer at 548 KB when removing individual activations, removing feature maps leads to a similar drop in accuracy between 548 KB and 627 KB. However, the difference is much more significant in the *bottle1* layer. Dynamic activation pruning can retain more than 75% of accuracy with only 940 KB of data, but dynamic feature map removal requires 2.2 MB to achieve the same accuracy. Overall, with half the data, i.e., 1.6 MB, dynamic feature map pruning can only achieve an accuracy of 57.7%. We get a comparable accuracy with 1.6 MB of feature maps using only 313 KB of individual activations.

Next, we evaluate the benefits of fine-tuning to retain accuracy in dynamic feature map pruning, see Figure 9. Dynamic feature map pruning with fine-tuning achieves better accuracies in the max-pooling layer than dynamic activation pruning without fine-tuning. However, dynamic activation pruning with fine-tuning outperforms dynamic feature map pruning in both layers at every compression level, regardless of whether we fine-tune it. This is due to the inability to retain individual value with high impact when removing entire feature maps. Therefore, dynamic activation pruning without fine-tuning can even outperform a network, trained for dynamic feature map pruning, as is the case in layer *bottle1*.

Overall, dynamic feature map pruning is an easy way to prune feature maps in layers where pruning individual values requires sparse matrix computation. Still, it can only remove up to around 20% in a layer without fine-tuning and without taking a significant hit to the accuracy. With fine-tuning, dynamic feature map pruning manages to compress intermediate data by a factor of up to 2.5× depending on the layer's size, while at most losing 0.9% accuracy.

5 CONCLUSION

In this paper, we analyze the sparsity of activations in a state-of-the-art network and exploit this sparsity with dynamic activation pruning and dynamic feature map pruning for efficient split computing. We show that activations contain up to 48% near-zero values. For feature maps, we show that there is almost no correlation between weights and activations, indicating a huge potential for compression compared to weight-based methods. However, we find that compression using dynamic feature map pruning only reduces intermediate data by up to 20% until losing a high degree of accuracy regardless of the layer's size. On the other hand, using dynamic activation pruning, we can reduce intermediate data by a factor of up to 3.3× in large layers with a drop of 1.1% in accuracy. Fine-tuning helps with accuracy after compression, allowing accuracy in case of dynamic feature map pruning to degrade gracefully and in case of dynamic activation pruning to improve compression by a factor of 2.5× with a loss of 1.1% accuracy in the smaller layer and up to 14× in the bigger layer with an accuracy loss of 1%. We argue that our results apply to many of today's established architectures, as they were inspired by ResNet's architecture. Nonetheless, we leave a detailed evaluation as future work.

ACKNOWLEDGEMENTS

This project has received funding from the Federal Ministry for Economic Affairs and Climate Action under the Marispace-X project grant no. 68GX21002E.

REFERENCES

[1] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. *SIGARCH Comput. Archit. News* 44, 3 (jun 2016), 1–13. https://doi.org/10.1145/3007787.3001138

[2] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. 2020. Fast Sparse ConvNets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[3] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. 2019. BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6. https://doi.org/10.1109/ISLPED.2019.8824955

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[5] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.* 22, 241 (2021), 1–124.

[6] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *SIGARCH Comput. Archit. News* 45, 1 (apr 2017), 615–629. https://doi.org/10.1145/3093337.3037698

[7] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[8] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. 2020. Inducing and Exploiting Activation Sparsity for Fast Inference on Deep Neural Networks. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 5533–5543.

[9] Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. 2019. Exploiting Kernel Sparsity and Entropy for Interpretable CNN Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[10] Yoshitomo Matsubara, Davide Callegaro, Sabur Baidya, Marco Levorato, and Sameer Singh. 2020. Head Network Distillation: Splitting Distilled Deep Neural Networks for Resource-Constrained Edge Computing Systems. *IEEE Access* 8 (2020), 212177–212193. https://doi.org/10.1109/ACCESS.2020.3039714

[11] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. 2022. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges. *ACM Comput. Surv.* (mar 2022). https://doi.org/10.1145/3527155 Just Accepted.

[12] Chanyoung Oh, Junhyuk So, Sumin Kim, and Youngmin Yi. 2021. Exploiting Activation Sparsity for Fast CNN Inference on Mobile GPUs. *ACM Trans. Embed. Comput. Syst.* 20, 5s, Article 77 (sep 2021), 25 pages. https://doi.org/10.1145/3477008

[13] Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeff Pool, Youngeun Kwon, and Stephen W. Keckler. 2018. Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 78–91. https://doi.org/10.1109/HPCA.2018.00017

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[15] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 31, 1 (Feb. 2017). https://doi.org/10.1609/aaai.v31i1.11231

[16] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. 2017. Sparsity Invariant CNNs. In *2017 International Conference on 3D Vision (3DV)*. 11–20. https://doi.org/10.1109/3DV.2017.00012

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc.

[18] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning Structured Sparsity in Deep Neural Networks. In *Advances in Neural Information*

Processing Systems, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc.

[19] Qing Yang, Jiachen Mao, Zuoguan Wang, and Hai Li. 2019. DASNet: Dynamic Activation Sparsity for Neural Network Efficiency Improvement. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. 1401–1405. https://doi.org/10.1109/ICTAI.2019.00197

[20] Tzu-Hsien Yang, Hsiang-Yun Cheng, Chia-Lin Yang, I-Ching Tseng, Han-Wen Hu, Hung-Sheng Chang, and Hsiang-Pang Li. 2019. Sparse ReRAM Engine: Joint Exploration of Activation and Weight Sparsity in Compressed Neural Networks. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 236–249. https://doi.org/10.1145/3307650.3322271

[21] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep Compressive Offloading: Speeding up Neural Network Inference by Trading Edge Computation for Network Latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (Virtual Event, Japan) *(SenSys '20)*. Association for Computing Machinery, New York, NY, USA, 476–488. https://doi.org/10.1145/3384419.3430898

[22] Xuda Zhou, Zidong Du, Shijin Zhang, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2019. Addressing Sparsity in Deep Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 10 (2019), 1858–1871. https://doi.org/10.1109/TCAD.2018.2864289

[23] Jingyang Zhu, Jingbo Jiang, Xizi Chen, and Chi-Ying Tsui. 2018. SparseNN: An energy-efficient neural network accelerator exploiting input and output sparsity. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 241–244. https://doi.org/10.23919/DATE.2018.8342010

[24] Xiaotian Zhu, Wengang Zhou, and Houqiang Li. 2018. Improving Deep Neural Network Sparsity through Decorrelation Regularization. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 3264–3270. https://doi.org/10.24963/ijcai.2018/453