Proceedings of the 4th International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/46/20241043

Latency reduction with compression-aware training for efficient distributed computing of Convolution Neural Networks

Xiangzhou Sun

The Webb Schools, Claremont, United States

jonas.sun2020@gmail.com

Abstract. To decrease workload on lightweight devices, this project accelerates the computation of Convolution Neural Networks (CNNs) and preserves accuracy through modifying the CNNs' training process. First, this research implements distributed computing to optimally divide the network workload onto both devices and the cloud. To reduce communication latency between devices and the cloud, this research introduces feature pruning by setting elements in the communicated feature to 0. However, naively pruning the feature causes a significant accuracy drop. To compensate for this limitation, this research applies pruning-aware training to preserve the CNNs' task performance. This research evaluates the proposed methods on multiple datasets and CNN models, like VGG-11 and ResNet-18 with PyTorch. Empirical results demonstrate that the methods can reduce the computational latency by 50-75% with a negligible 1% accuracy loss. Specifically, this research first identifies the system bottleneck by comparing on-device, on-cloud, and communication latencies (on-device: 14.8%, on-cloud: 1.7%, communication: 83.5%). Then, this research compares multiple pruning strategies and observe the superiority of magnitude-based pruning. At 0.992 sparsity, magnitude-based pruning outperforms other strategies by 45% in accuracy. Finally, this research verifies the effectiveness of the proposed pruning-aware training method by comparing it with the baseline at various splitting points and networks. Pruning-aware training decreases the accuracy loss by up to 26% at 0.998 sparsity. In conclusion, even though distributed computing accelerates applications on lightweight devices, compressing the communication cost is crucial and challenging. This research proposed methods effectively reduce communication latency without sacrificing accuracy, conserving the effectiveness of CNN.

Keywords: Convolution Neural Network (CNN), Distributed Computing, Pruning, Compression-aware Training.

1. Introduction

As technology continues to evolve in contemporary society, more portable devices like phones, laptops, Virtual Reality (VR) headsets, and intelligent glasses support computer vision AI applications. As described in *Unity virtual reality projects*, VR devices provide an immersive experience for users through applications such as object detection, gaze tracking, and movement recognition, which are all essential functionalities for realistic displays or entertainment [1]. Other researches also investigate CNN, the status quo of computer vision problems, in mobile devices for object of face recognition [2].

DOI: 10.54254/2755-2721/46/20241043

However, these applications usually require machine learning to improve performance and adapt to personalized changes. For example, to simulate driving, lightweight devices use CNN to detect hand gestures, allowing users to practice steering vehicles without injuries or consequences [3].

According to *Artificial neural networks*, most modern computer vision machine learning techniques are based on CNN such as VGG or ResNet [4]. Despite the impressive performance of CNNs, they require expensive and time-consuming computations and considerable memory consumption. However, most mobile devices have constrained computation and memory resources to reduce the product weight, size, and price [5]. While keeping the device lightweight decreases the performance of neural networks, incorporating more hardware to improve latency will result in a hefty device, which worsens the user experience. This problem is significant and relevant, because on-device latency will inevitably obstruct the advancement of realism and quality in portable devices.

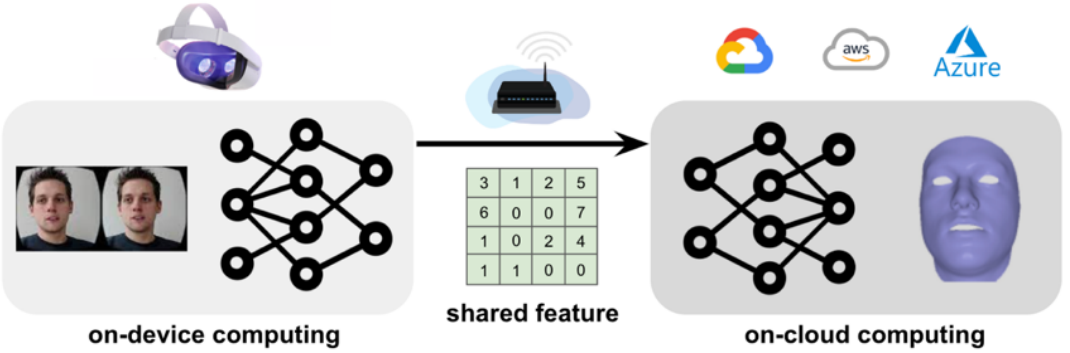

Figure 1. Illustration of the distributed computing paradigm.

In distributed computing, a neural network is split into a head model (on-device part) and a tail model (on-cloud part). The intermediate shared feature from the head model is transmitted via wire/wireless connection from devices to the cloud.

Under this context, this research proposes using a compression-aware distributed computing model to reduce latency on CNNs of portable devices, and this research question is: in terms of latency and accuracy, how effective is this compression-aware distributed computing model for Convolution Neural Networks?

This framework splits the neural network workload on both mobile devices and the centralized cloud with distributed computing [6]. Neural networks consist of layers of neurons that take the input of the previous layer to produce output for the following layer. Thus, neural network computations are dividable into intermediate features as long as the layers are connected through inputs and outputs. Instead of completing the calculations solely on the device or the cloud, this research optimizes the overall system efficiency by leveraging local and cloud computing with the ideal split point in the neural network:

$$[total_l = cloud_l + communication_l + device_l] \quad (1)$$

While computing more layers on the head-mounted device cause heating and speed issues, more layer computation on the cloud increases the cost of feature communication. Existing work employs exhaustive searching across layers in a deep neural network to locate the optimal splitting point in terms of accuracy and overall efficiency [7]. One limitation of this method is the high data communication cost, which this research can reduce via compression.

In this work, this research demonstrates that it can compress the communicated data by pruning elements in the feature. Pruning reduces the data communication cost by converting elements in the intermediate feature to zero. However, directly pruning the data causes the task accuracy to drop significantly because, during the pruning process, pruning removed weight connections in the network to decrease the model size and introduced information loss.

DOI: 10.54254/2755-2721/46/20241043

This research resolves this issue by implementing pruning-aware training, which includes pruning in the network training process. Gradually, the neural network becomes accustomed to the compression and reacts accordingly, improving the final network accuracy. Therefore, this research implements the pruning in both the testing and training phases.

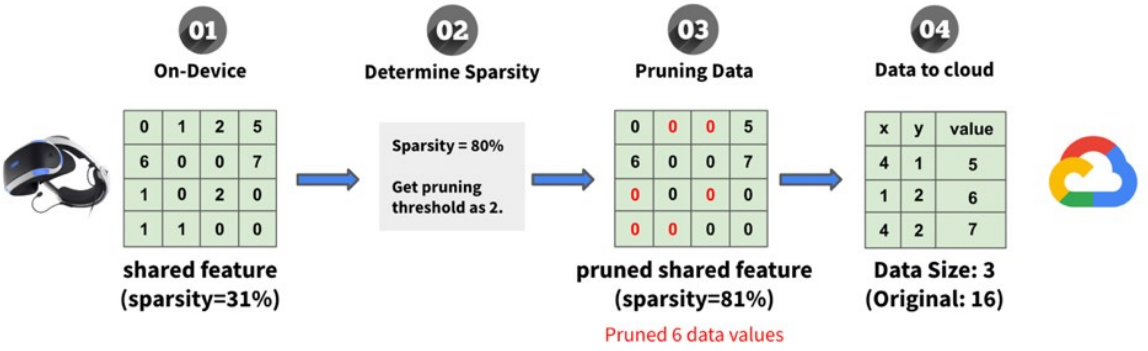

Figure 2. Pipeline of magnitude-based feature pruning for reduced communication volume.

Given a share feature, this research first calculates the pruning threshold for a specific target sparsity. Then, this research prunes the feature by comparing magnitude of elements in the feature with the threshold. If one element's magnitude is smaller than the threshold, this research sets it to zero. Otherwise, this research preserves its value. After the above process, this research only transmits the pruned feature to the cloud. As a result, this research decreases communicated volume and cost significantly by pruning, as only a small percentage of non-zero elements and their coordinates will be transmitted.

Also, instead of randomly removing weights, this method rounds smaller numbers in the intermediate feature to zero, efficiently decreasing the amount of data communication while maintaining sufficient levels of accuracy. Even though this research observes this method to reduce the neural networks precision slightly, its latency improvement and data communication reduction are significant.

2. Related Work

In order to optimize system level efficiency, others proposed distributed computing to split the machine learning workload onto both edge devices and the cloud.

For instance, Kang proposed the Neurosurgeon method [7]. The Neurosurgeon system exhaustively searches through all the possible splitting points in a DNN and locates the optimal layer by profiling. This system adapts to various DNN architectures and hardware platforms, reducing latency by 3.1 × on average and energy consumption by 59.4%. However, they did not compress the communicated data to reduce communication latency, a bottleneck in the overall latency.

Another study conducted by Matsubara utilizes edge computing and network distillation to minimize the computational complexity of neural networks [8]. Their method first divides complicated DNNs into a head and tail model to run on both the device and the cloud. The head model is then "shrunk" with network distillation, a state-of-the-art technique that trains simpler "student" models to approximate the output of more complicated "teacher" models. The distillation is beneficial in model complexity reduction while maintaining stable performance and preventing overfitting. Compared to splitting without network distillation, their method reduces bandwidth by 98% and computation load by 85%. However, neural distillation introduces accuracy loss after replacing complicated networks with smaller, simpler ones. The "student" models are also computationally costly and time-consuming to train and test.

In addition, a survey explores research challenges and applications of recent studies concerning the distributed computing of deep neural networks [9]. Matsubara compares it against model compression and neural distillation on various Computer Vision (CV) and Natural Language Processing (NLP) tasks.

DOI: 10.54254/2755-2721/46/20241043

3. Methodology

3.1. Method 1: Distributed Computation with optimal layer split point

Given a neural network of N layers, this research wishes to locate the optimal splitting point that minimizes overall latency. The overall latency consists of three parts: on-device computational latency, communication latency, and on-cloud computational latency. If all neural network layers compute on-cloud, the data communication size will be the same as the size of the raw input. However, if all neural network layers compute on-device, the communication latency will be zero while the latency of computation on-device would increase. Therefore, the position of the split point will impact the three latencies simultaneously.

In this work, this research aims to balance the three latencies by pinpointing the split location. This method optimally distributes the computation to the device and cloud, resulting in latency reduction and communication savings.

First, for an arbitrary split point, this research profiles the overall latency, which consists of on-device, communication, and on-cloud latency. Then, this research selects the optimal split point among all possible layers according to their profiled overall latency. After the optimal split point selection, this research computes the first part of the network workload on the device and transfer the shared feature to the cloud via the internet.

3.2. Method 2: Pruning shared feature according to magnitude for communication saving

After offloading part of the computation from the device to the cloud, this research aims to find the bottleneck of this framework by calculating and comparing the latency of each:

$$[bottleneck_l = max\{cloud_l, communcation_l, device_l] \tag{2}$$

The bottleneck to distributed computing is data communication. Even though the size of the shared feature is smaller than the raw input, feature communication still contributes to a considerable portion of the overall latency. In this work, this research aims to prune the shared feature from the device according to the magnitude of each element in the shared feature. This method minimizes accuracy degradation while significantly reducing communication size between the device and the cloud.

Since the elements with zero values are skipped while transferring data, pruning reduces communication size by setting shared feature values to zero. First, this research obtains the shared feature by computing the first part of the neural network on the device. Then, this research applies the method to prune the feature for communication savings. During the pruning, this research arbitrarily determines a sparsity *s*, which specifies the percentage of zero elements in the feature tensor. For example, in a shared feature with 10000 elements, a 90% sparsity decreases data size to 1000, which is 10% percent of its original size:

$$[overall_l = cloud_l + communication_l \times s + device_l, 0 < s < 1] \tag{3}$$

After determining a desired sparsity, this research calculates the pruning threshold *t* for the feature by sorting them according to their magnitude. Any element value less than the pruning threshold will be set to zero, while any element greater than the threshold remains unchanged. For a given element *x* in feature n, the pruning $f(x_n)$ can be expressed as:

$$f(x_n) = \begin{cases} 0, x_n < t, \\ x_n, x_n >= t \end{cases} \tag{4}$$

DOI: 10.54254/2755-2721/46/20241043

Python code for Pruning by random:

```
def set_zero_randomly(inp, target_sparsity):
    current_sparsity = get_sparsity(inp)
    if current_sparsity >= target_sparsity:
        return inp
    else:
        # if element is 0 mask is 0
        zero_mask = (inp != 0).float()
        # generate uniform distribution
        random_mask = torch.rand_like(inp)
        real_mask = zero_mask * random_mask
        inp_flatten = real_mask.flatten(start_dim=0)
        kth_value = torch.topk(inp_flatten.abs(), int((1 -
        target_sparsity) *inp.numel()), largest=True).values[-1]
        mask = real_mask.abs() >= kth_value
        return mask.float() * inp
```

Python code for Magnitude-based Pruning:

```
def set_zero_magnitude(inp, target_sparsity):
    current_sparsity = get_sparsity(inp)
    if current_sparsity >= target_sparsity:
        return inp
    else:
        inp_flatten = inp.flatten(start_dim=0)
        kth_value = torch.topk(inp_flatten.abs(), int((1 -
        target_sparsity) *inp.numel()), largest=True).values[-1]
        mask = inp.abs() >= kth_value
        return mask.float() * inp
```

3.3. Comparison between Pruning by random and Magnitude-based Pruning

Multiple pruning algorithms decrease communication sizes, such as random or magnitude-based pruning. To determine the optimal method for pruning, this research compares the two algorithms based on accuracy and efficiency.

Theoretically, this research concludes that pruning the shared feature based on magnitude is more effective than other methods, such as pruning randomly. By pruning according to the elements' values, the neurons with the smallest values will be converted to 0. As they are the values closest to 0, their changes will be minimal compared to converting greater values. If this research randomly prunes the data in the shared feature, it may accidentally convert substantial elements to 0, while smaller values remain unchanged. Therefore, pruning through magnitude generally causes a less significant change in the shared feature than pruning randomly, which improves accuracy effectively.

To demonstrate magnitude-based pruning's effectiveness, this research conducts empirical experiments to compare the two methods. This research implements the two pruning methods on identical models and shared features. The method with higher accuracy is the more optimal one.

Even though pruning reduces latency significantly, removing elements in the shared feature causes the cloud-side neural network to receive less accurate data. If the sparsity is overly high, the testing accuracy of the network will drop notably due to an overwhelming information loss introduced by pruning.

DOI: 10.54254/2755-2721/46/20241043

3.4. Method 3: Pruning-aware training to improve accuracy

A prominent issue in Method 2 is that, during inference, the unexpected information loss introduced by pruning causes a notable accuracy drop. Since the shared feature of the neural network is not compressed during training, the network cannot anticipate this information loss and, therefore, cannot adapt accordingly. To minimize the effects of pruning, this research implements pruning-aware training to improve the accuracy of this distributed computing framework.

The pruning-aware training method applies pruning to both the training and inference phases. Consequently, the neural network can adapt to information loss introduced by pruning and adjust its weights accordingly during training via stochastic gradient descent.

For instance, some elements in the shared feature are essential in an accurate prediction for the on-cloud network. If pruning accidentally prunes these notable elements, an accuracy drop is inevitable due to unexpected information loss. However, with pruning-aware training, the network learns to reduce dependency on elements that will be pruned and emphasize other un-pruned elements to produce correct predictions. Therefore, pruning-aware training better prepares the neural network for computing accurate outputs amid the influence of compression.

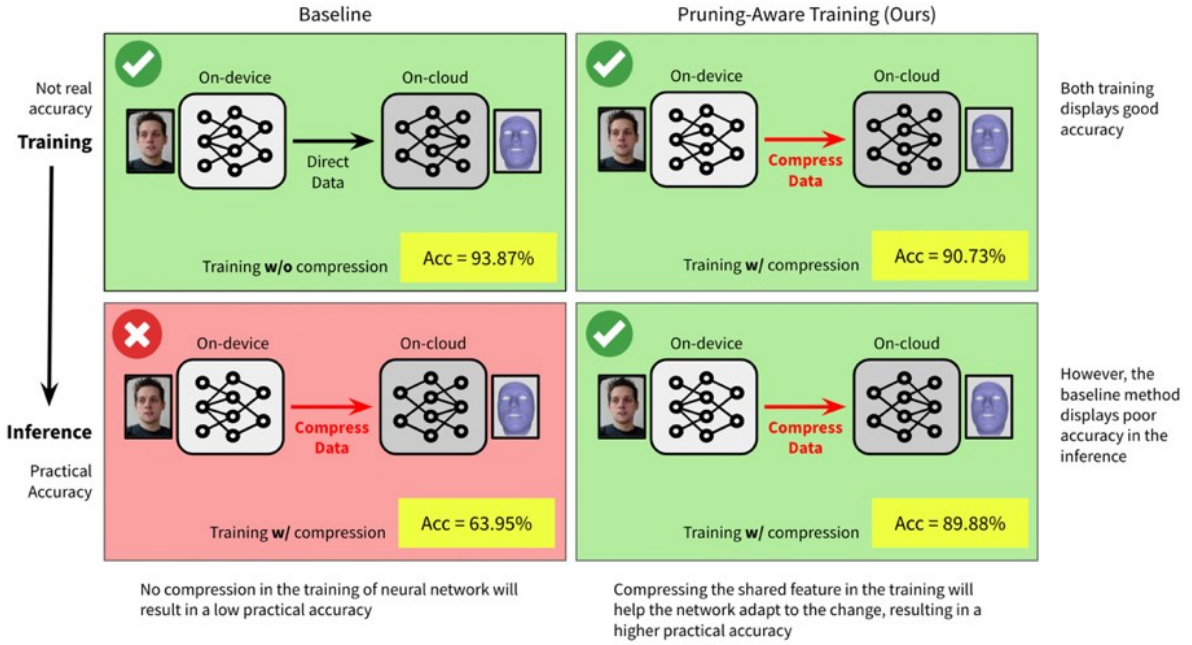

Figure 3. Comparison between pruning-aware training (right) and baseline method (left).

During model training, the baseline method (top left) demonstrates a higher accuracy than this method (top right). However, during inference (actual usage), the baseline method (bottom left) presents significantly lower accuracy than this method (bottom right).

Figure3 explains why pruning-aware training results in higher accuracy than the baseline method during inference. Even though the baseline method may obtain a higher accuracy during training (93.87%), the accuracy plummets when evaluating the test dataset (63.95%) due to unexpected information loss. However, the training and testing accuracy for pruning-aware training are less varied and more consistent (90.73% and 89.88%), as the network senses and adjusts to the compression of the shared feature. Therefore, pruning-aware training notably increases accuracy and produces a more practical neural network.

4. Hypothesis

First, according to the neurosurgeon method proposed by Kang in related works, distributed computing effectively reduces latency in multiple DNN architectures [8]. Therefore, this research hypothesizes that

DOI: 10.54254/2755-2721/46/20241043

the compression-aware distributed computing model, which applies a similar framework to other convolutional neural networks on AR/VR systems, will yield a similar result in reducing latency.

Also, according to previous work on structural pruning and particle filtering, a pruned network can preserve some of its original accuracies through retraining [10]. Even though this work emphasized the pruning of the network weights on multiple layers, it may also imply that retraining a network with pruning at one shared feature in a distributed computing model can also preserve a certain degree of accuracy. Thus, this research predicts the pruning-aware training method to minimize accuracy degradation.

5. Experiments

5.1. Setup

This research uses the VGG-11 [11] and ResNet-18 models [12]. This research conducts the experiments with PyTorch version 1.13.1 on the NVIDIA A100-SXM4-40GB GPU provided by Google Colaboratory [13]. This research conducts each experiment with 3 trials and took the average of the values to complete the graphs.

5.2. Experiment 1: Identifying the bottleneck of overall latency.

For distributed computing of neural networks on VR systems, this research aims to minimize overall latency, which consists of on-device latency, communication latency, and on-cloud latency. To effectively reduce the overall latency, this research first identifies the bottleneck of the system, the most time-consuming part of overall latency, by calculating on-device, communication, and on-cloud latency individually.

In this experiment, this research assumes the computation power of the on-device system to be 125 GOPs (Giga Operations Per Second). This research also determines that the cloud equips a more capable GPU and thus has a five times higher computation power (625 GOPs) than the devices'. In addition, this research measures the total computation amount for both devices and the cloud by counting the multiply-accumulate (MAC) unit. A MAC unit contains one multiplication and accumulation.

This research sets the data communication bandwidth as 300Mb per second according to the 5th generation mobile network (5G) [14].

With the previous setup, this research calculates communication latency, on-device latency, and on-cloud latency. As shown in **Figure 4**, the results demonstrate that communication latency is the bottleneck of distributed computing. Without compressing the data, the communication latency contributes to 54.0% of the overall latency. However, if this research compresses the data with a sparsity of 0.9, it decreases the communication latency to 10.5% of the overall latency. This result demonstrates that data compression is necessary and beneficial to distributed computing.

Splitting at the 17th instead of the 13th manifests similar results. Without compressing the shared feature, the communication latency is 83.5% of the overall latency. After compression with 0.9 sparsity, the communication latency reduces to only 33.5% of the overall latency.

Therefore, this research concludes that the bottleneck of distributed computing is communication latency, and compression effectively mitigates this problem.

DOI: 10.54254/2755-2721/46/20241043

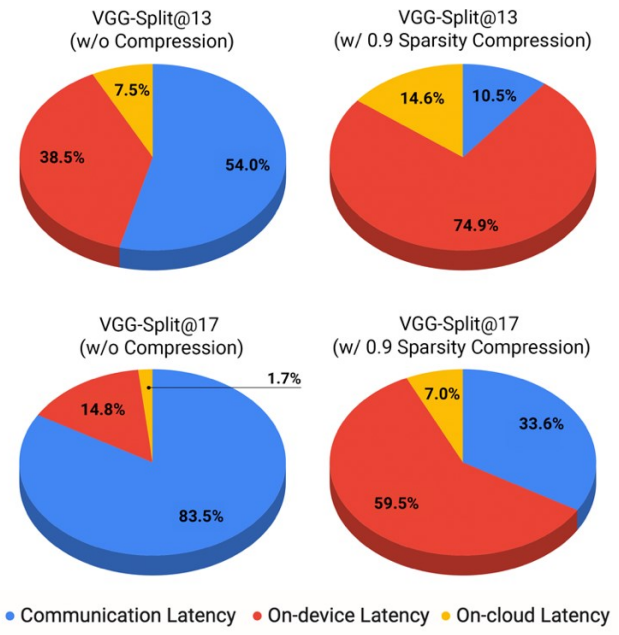

Figure 4. The three latencies of the distributed computing model.

The pie charts display data communication as the bottleneck of the overall latency through comparing the three latencies of the distributed computing model. This research also demonstrates how compressing the data significantly reduces this latency. VGG-11 Network as example: For the 13th (top 2 charts) and 17th (bottom 2 charts) layer, this research calculates the percentage of communication latency (blue), on-device latency (red), and on-cloud (yellow) in the overall latency. Pie charts on the left: Without compression. Pie charts on the right: With 0.9 (90%).

5.3. Experiment 2: Comparing pruning algorithms in terms of accuracy.

After identifying data communication as the bottleneck of the distributed computing framework, this research decided to prune the shared feature at the split layer to decrease the shared feature size. There are several algorithms for pruning, including random pruning and magnitude-based pruning. The research compares the two distinct algorithms under various sparsity levels ranging from 0.9 to 0.9996. This research tests two different splitting points (13th and 17th layers) in the VGG-11 network and present the result in **Figure 5**.

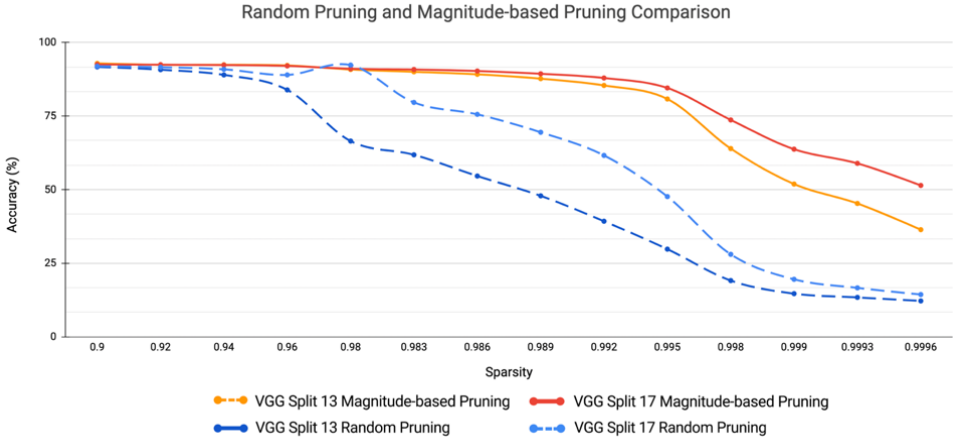

Figure 5. Random pruning and magnitude-based pruning.

DOI: 10.54254/2755-2721/46/20241043

Accuracy comparison between the two pruning algorithms: Random pruning and magnitude-based pruning. The orange and red line represent magnitude-based pruning for the 13th and 17th layer. The dark and light blue line represent random pruning for the 13th and 17th layer.

At 0.9 sparsity, both pruning methods demonstrate a similar accuracy, at approximately 92.5%, comparable to the accuracy without compression.

As the sparsity increases, the accuracy of the random pruning method begins to reduce, dropping to around 40%. On the other hand, the magnitude-based pruning method's accuracy decreases slightly by 10%. When this research continues to increase the sparsity from 99.2%, the accuracy of random pruning declines further to around 10-15% at 0.9996 sparsity. However, magnitude-based pruning maintains the accuracy at 69%, almost 50% higher than random pruning at the same layer. As this research continues to increase the sparsity, random pruning's accuracy decreases more rapidly from 0.96 sparsity, dropping to around 40% at 0.992 sparsity. On the other hand, magnitude-based pruning declines slowly by less than 10% at the same sparsity level.

Results of random pruning and magnitude-based pruning at the 17th layer produce a similar pattern. At sparsity 0.9, both pruning algorithms yield identical accuracy at around 92%. After sparsity increase, the accuracy for random pruning drops to 70% while magnitude-based pruning remains at around 90%. As the sparsity continues to increase, the accuracy of random pruning drops to approximately 28% accuracy at 0.998, while magnitude-based pruning maintains an accuracy of 73%. Therefore, this research concludes that magnitude-based pruning achieves a higher accuracy than random pruning.

5.4. Experiment 3: Comparing baseline method with pruning-aware training in terms of accuracy

After deciding the optimal pruning algorithm in Experiment 2, this research improves the accuracy of the network further through pruning-aware training, as discussed in Sec.3.4.

In this experiment, this research compares the accuracy of the baseline method without pruning-aware training with the framework. The setup implemented in this experiment is identical to previous experiments.

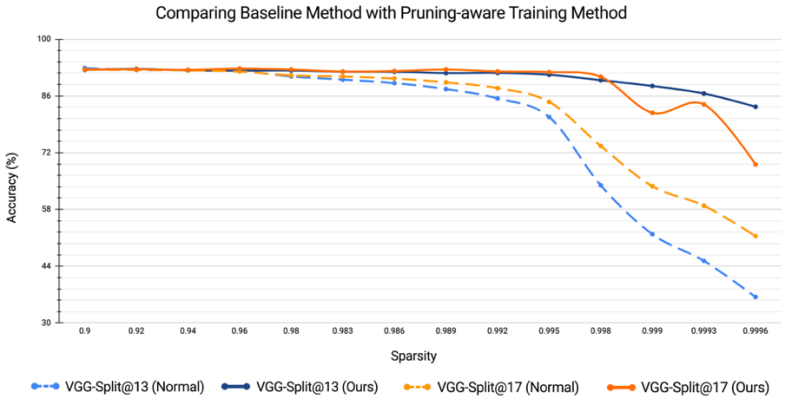

Figure 6. Comparison between the baseline method without pruning-aware training and the method with pruning- aware training.

The baseline method is represented with dotted lines, while the pruning-aware training is represented with normal lines.

As shown in **Figure 6**, baseline and pruning-aware training methods have a similar testing accuracy at low sparsity (0.9-0.96). As the sparsity increases from 0.96 to 0.995, this research observes the baseline method of both layers to reduce in accuracy, dropping from 92% to around 89%. However, the pruning-aware training maintains a high accuracy at around 92%.

When sparsity increases to 0.9996, the accuracy of the baseline method plummets to around 36%. At the same sparsity, the accuracy for the pruning-aware training method remains at 83%, almost a 50%

DOI: 10.54254/2755-2721/46/20241043

increase from the baseline method. This research observes a similar trend with the model split at the 17th layer.

In conclusion, the baseline method and pruning-aware training both yield high accuracy at low sparsity. However, when sparsity increases, pruning-aware training sustains a high accuracy, while the accuracy of the baseline method reduces significantly. Therefore, the pruning-aware training method is superior to the baseline method.

5.5. Experiment 4: Comparing overall latency reduction in terms of accuracy and latency

In the previous experiments, this research concluded that magnitude-based pruning and pruning-aware training are effective methods for compressing the shared feature and minimizing accuracy loss. In this experiment, this research compares the uncompressed network with the compressed network by calculating the overall latency and accuracy.

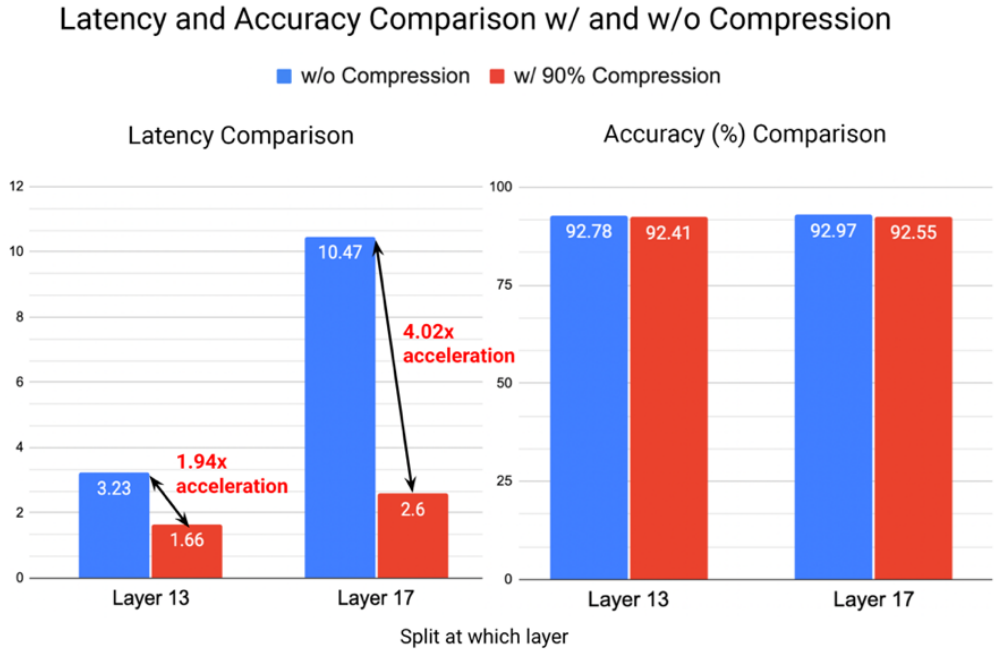

Figure 7. Comparing baseline distributed computing with this method that compressed the shared feature by 90%.

In Figure 7, the arrows signify the decrease in overall latency (blue) and accuracy (red).

This research takes the results from Experiment 1 for the overall latency for the uncompressed method. For the compressed model, this research sets the sparsity to 90%. As shown in **Figure 7**, the overall latency of the uncompressed model is 3.23ms, and the accuracy is 92.89%. When this research applies compression to the model, the overall latency is reduced significantly by 1.57ms (48% decrease), while the accuracy only declined slightly by 0.37%, an almost negligible amount. The results demonstrate that this method drastically reduces overall latency with a notably less trade-off of accuracy.

This research observes the benefit of this proposed compression to be more pronounced when applying compression at the 17th layer's output feature. The overall latency reduces by 7.87ms of the original 10.47ms, almost a 75% decrease. Meanwhile, the accuracy only drops slightly by 0.42%. With a negligible decline in accuracy, this research considerably lowers the overall latency.

5.6. Experiment 5: Generalization to other CNN architectures

To validate the generalizability of the proposed compression method on different networks, this research further applies the pruning-aware training method to other neural architectures, such as ResNet-18.

DOI: 10.54254/2755-2721/46/20241043

In this experiment, this research reproduces the settings of Experiment 3 and compare the baseline method with this pruning-aware training method on ResNet-18 instead of VGG-11.

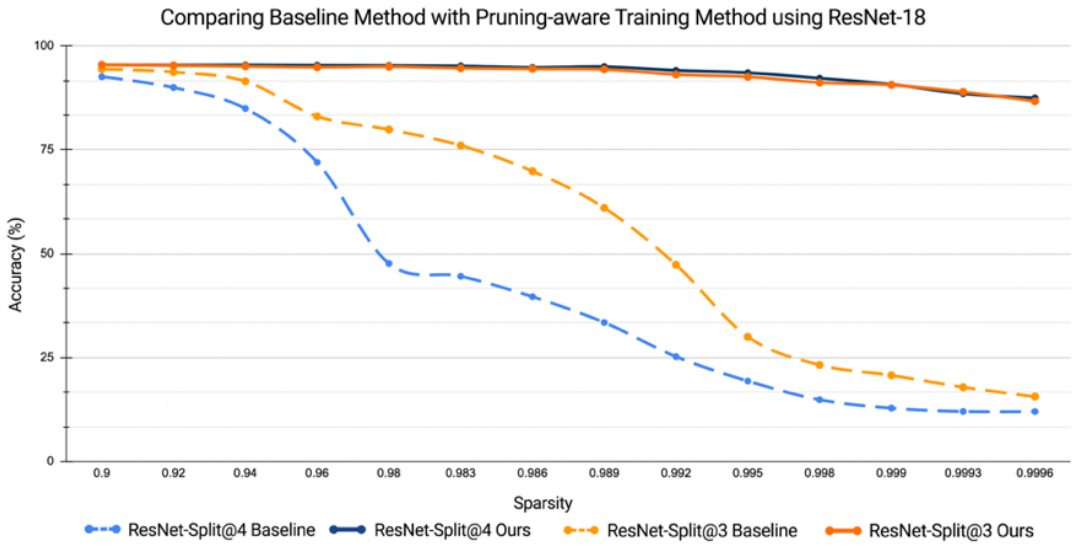

Figure 8. Comparison between the baseline method without pruning-aware training and the method with pruning-aware training using ResNet-18.

The baseline method is represented with dotted lines while the pruning-aware training is represented with normal lines.

As demonstrated in **Figure 8**, this research observes a similar trend as Experiment 3, validating the generalizability of this methods. As the sparsity increases, the pruning-aware training method preserves the accuracy, while the accuracy of the baseline method drops significantly. For example, at a sparsity of 0.992, the baseline method's accuracy reduces to 25.25%. In comparison, pruning-aware training yields an accuracy of 94.12%, considerably higher than the baseline method's accuracy. At 0.9996 sparsity, the baseline method's accuracy is merely 12.03% in contrast to the 87.48% of the pruning-aware training method. At all sparsity levels, the pruning-aware training method is superior to the baseline method in terms of accuracy.

This research notices a similar pattern when it distributed the network at different blocks of ResNet-18. For example, when this research splits the network at the 3rd block instead of the 4th, the pruning-aware training method is also more effective than the baseline method.

5.7. Results Analysis

This research's results are similar with a study that investigated compression-aware training with the Frank-Wolfe algorithm in neural networks [15]. By applying low-rank decompositions on convolutional layers and pruning convolutional filters, their study decreases parameters in neural networks and preserves accuracy through compression-aware training. At a sparsity of 0.98, their compression-aware training on unstructured weight pruning led to only a slight accuracy decrease. Similarly, the pruning-aware training also only lost about 0.28% of accuracy at a 0.98 sparsity, which is a similar outcome.

Since this research conducted the experiment through a well-established platform and with a fixed dataset for training and testing, the results are mostly consistent and accurate. However, possible sources of variability include the randomization of data during training and the weights pruned during random pruning.

When training the neural network, this research enabled the "shuffle" option for each batch of data. This can effectively prevent over-fitting. The random factor in "shuffle" causes the final accuracy of the

DOI: 10.54254/2755-2721/46/20241043

network to change slightly with every trial. However, this source of variability is insignificant compared to the overall trend.

Also, when pruning weights randomly in one of the experiments that tested different pruning algorithms, the randomness also caused accuracy to vary. For example, if the method accidentally pruned more significant elements in the shared feature, the final accuracy obviously decreases.

An example of bias in the experiments may be the non-linear increment of sparsity to display the effectiveness of pruning-aware training and magnitude-based pruning. For example, instead of increasing the sparsity in a linear fashion, this research augmented it with smaller values as the sparsity approaches 1, similar to a power or exponential function. Even though this method of displaying data demonstrates the effects of this methods optimally, the fixed intervals between each value may perplex audiences and misguide them of the trend in a linear sparsity representation.

6. Conclusions

Conventionally, neural networks in portable devices compute either on-cloud or on-device. In this work, it introduces a distributed computing method for optimal neural network workload distribution in lightweight devices. In the results, this research demonstrates that the bottleneck of this model's overall latency lies in data communication. Motivated by the above observation, this research investigates the trade-off between the neural network's accuracy degradation and the pruning rate of the shared feature. After comparing multiple pruning methods, this research concludes that magnitude-based pruning compresses the data with minimal loss of accuracy. For the neural network to adapt to the pruning, this research also applies the pruning-aware training method, which incorporates compression during the training for higher testing accuracy. This research also tests this method on different layers and with multiple neural networks, such as VGG-11 and ResNet-18, to prove the effectiveness of pruning-aware training on all neural architectures.

As a result, this research verified the hypothesis and developed the compression-aware distributed computing model, a system that utilizes the pruning and the distributed computing model to significantly decrease the size of transferred data while sustaining sufficient accuracy. The system accommodates all VR devices and effectively reduces their workload, allowing the device to become lightweight and energy-efficient. At a sparsity of 0.995, this research reduces the communication size by 99.5% yet only sacrifice less than 1% of accuracy on average. At the same sparsity, the baseline method without pruning-aware training suffers from more than 45% accuracy degradation on average. Overall, with an accuracy drop of less than 0.5%, this method can decrease the overall latency by 48% and up to 75%, depending on the location of the split point.

Acknowledgements

I express my wholehearted gratefulness to all individuals who supported and assisted me in this research or experimental process of this project.

First, I immensely appreciate the counsel of my supervising and computer science teacher, Ms. Suarez, who taught me numerous important concepts with the internet and always believed in this project. Her selfless education inspired many of this research innovations, allowing me to achieve this results today.

Next, I acknowledge science department chair Ms. Nacionales and science teacher Dr. Martin for their constant guidance and motivation to this research. Many thanks to my school, the Webb Schools, for providing me the opportunity to study and pursuit my interests in the Science Research Lab afternoon activity, where I obtained invaluable experiences with classmates and teachers. Finally, my greatest gratitude for all the personal support along the way, including my relatives, friends, and classmates, who all drove me to finish this difficult research.

References

[1] Linowes, J. (2015). *Unity virtual reality projects*. Packt Publishing Ltd.

DOI: 10.54254/2755-2721/46/20241043

[2] Yanai, K., Tanno, R., and Okamoto, K. (2016). Efficient mobile implementation of a cnn-based object recognition system. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 362– 366.

[3] Xu, D. (2006).A neural network approach for hand gesturere cognition in virtual reality driving training system of spg. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume **3**, pages 519– 522. IEEE.

[4] Yegnanarayana, B. (2009). *Artificial neural networks*. PHI Learning Pvt. Ltd.

[5] Ponomarev, E., Matveev, S., Oseledets, I., and Glukhov, V. (2021). Latency estimation tool and inves- tigation of neural networks inference on mobile gpu. *Computers*, **10**(8):104.

[6] Bertsekas, D. and Tsitsiklis, J. (2015). Parallel and distributed computation: numerical methods. Athena Scientific.

[7] Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. (2017). Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, **45**(1):615–629.

[8] Matsubara, Y., Baidya, S., Callegaro, D., Levorato, M., and Singh, S. (2019). Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 21–26.

[9] Matsubara, Y., Levorato, M., and Restuccia, F. (2021). Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys (CSUR)*.

[10] Anwar, S., Hwang, K., and Sung, W. (2017). Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, **13**(3):1–18.

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

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

[13] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, **32**.

[14] Wisely, D., Wang, N., and Tafazolli, R. (2018). Capacity and costs for 5g networks in dense urban areas. *IET Communications*, **12**(19):2502–2510.

[15] Zimmer, M., Spiegel, C., and Pokutta, S. (2022). Compression-aware training of neural networks using frank-wolfe. *arXiv preprint arXiv:2205.11921*.