Frequency Regularization: Efficient Neural Network Compression for Mobile Device Deployment

Wenhao You(wyou1@ualberta.ca)
Leo Chang(chewei@ualberta.ca)

1 Abstract

In general, neural networks require high-specification hardware because of their complexity. However, in daily life, people have the requirements for running neural network applications on their own mobile devices but have limited hardware. Even the researchers have to use the computer in their lab instead of running the pre-trained models by using mobile devices. In this proposal, we will focus on how to implement a simple and creative compression algorithm called Frequency Regularization (FR) [1], which is to compress a basic neural network on Android mobile devices. We also discuss another three existing methods to achieve the goal: upgrade hardware [2], implement NestDNN [3], and implement "One-shot Whole Network Compression" [4]. The reason we did not choose to upgrade mobile hardware is that it is difficult to achieve in a short course time. Moreover, the other two compression algorithms are more difficult to implement and not as efficient as FR. Hence, it is the reason we chose FR to do further research.

2 Introduction

Neural networks have been implemented in many applications and performed great results. However, these neural networks are usually implemented on high-specification hardware such as computers. The reason is because there are many parameters in different neural networks. The more parameters in the neural networks, the more storage and internet transmission rates are occupied. In this research, we try to look for a method that can make the neural networks run on a limited hardware device, such as a mobile device. Although mobile devices have more enhanced hardware nowadays, because of their limited size, the efficacy is still not as powerful as the computers. There are several benefits of running the neural networks on limited hardware devices, specifically, mobile devices. The most important parts are the privacy [5] and the data security. Processing data on-device ensures that personal information does not need to be uploaded or transmitted to the cloud servers, which enhances privacy [5].

Running neural networks on mobile devices can also bring other advantages. If we can implement the neural network models directly on mobile devices, it can reduce our dependence on the internet connection to some extent. Since mobile devices play a significant role in our daily lives, the procedures can be accessed easily. Moreover, for some applications that need real-time feedback, such as object recognition and image segmentation, neural networks on mobile devices can also make the process faster to improve user experience. The most important part of modern life is that we can keep the data processing and inference in the internal system of mobile devices rather than sending it to a remote server, which can increase the processing speed.

The new method, Frequency regularization (FR) [1], is based on the frequency domain to compress the neural network. It is divided into two steps: dynamic tail-truncation and inverse discrete cosine transform (IDCT). The main idea of this method is that some of the neural network parameters may be redundant in their representation. The algorithm focuses on using the frequency domain transform for network regularization to restrict information redundancy

during the training process and introduce FR as shown in Fig. 1. Therefore, the redundant part can be removed without any side effect on the performance of the network.

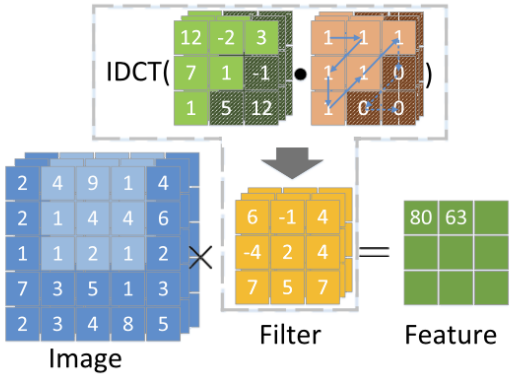

Figure 1: Illustration of the proposed frequency regularization. The tail elements of tensors in the frequency domain are truncated first, then input into the inverse of the discrete cosine transform to reconstruct the spatial tensor for learning features [1].

The known shortcoming of this algorithm is the training speed compared to other training methods, which is slower. To apply Frequency Regularization on the neural network and implement it on mobile devices, we will encounter two challenges. (i) The PyTorch library may not be capable of this new algorithm. Therefore, we need to adjust some parameters. (ii) The mobile hardware limitation that can restrict the algorithm to run.

From the above paper mentioned, Frequency Regularization performs a better compression and works on most general neural networks [1]. Combining the best of our knowledge, we believe this research contributes as the first implementation of applying Frequency Regularization to neural networks but on limited hardware devices, specifically, mobile devices. In conclusion, this paper explores a simple, efficient, and creative method to run a large neural network on mobile devices based on the Frequency regularization algorithm. Our purpose is not only to accomplish this implementation, we want to ensure the performance is similar to the result running on high-specification hardware such as computers. We hope to provide a fast and secure neural network application experience on mobile devices.

3 Brief Summary of Existing Work

We cover the methods for implementing neural networks on the mobile over five sections. In Section 3.1, upgrading hardware on mobile devices and which parts are introduced. At the algorithm level, there is method called Extreme Learning Machine (ELM) [2], which assigns hidden layer weights randomly. Section 3.2 shows the biref introduction of ELM. What's more, there is an algorithm called NestDNN [3] based on the dynamics or runtime resources to enable resource-aware multi-tenant on-device deep learning for mobile vision systems as shown in Section 3.3. Moreover, there is an algorithm called "one-shot whole network compression" [4] to compress convolutional neural networks (CNNs) for mobile deployment as shown in Section 3.4. There is also an algorithm named Frequency Regularization that helps compress the neural network by using the idea of reducing the redundancy information during the training process. The general steps are as shown and introduced in Section 3.5.

3.1 Upgrade Hardware

Hardware accelerators such as the digital signal processors offer solutions to overcome these challenges. Many manufacturers have already produced better hardware. For example, Apple presented the Metal Performance Shaders with the support of CNNs accelerated by GPU. This is a solution built on top of the Metal API and allows custom operations. Since there is an experiment [2] that confirmed the feasibility of Big Data analysis on modern mobile devices. Moreover, the hardware on Android devices is upgraded by four main companies, such as Qualcomm, HiSilicon, MediaTek, and Samsung [6], which can work with neural networks as well.

3.2 Algorithm Extreme Learning Machine (ELM)

The Extreme Learning Machine (ELM) is able to allocate the weight of the hidden layers randomly. And it can generate linear output layer as well. Compared with the traditional neural

networks, ELM provides faster training speed and high efficiency. The main idea of ELM is to generate the random hidden nodes, in order to avoid backpropagation. Its advantages are the simplicity, the efficiency, and the fast training speed. However, the biggest disadvantage of ELM is the sensitivity to the number of hidden nodes. In the experiement [2], ELM was proved to achieve the same performance with other deep learning models in many tasks. Moreover, ELM is easy to optimize since it has simple structure.

3.3 Algorithm NestDNN

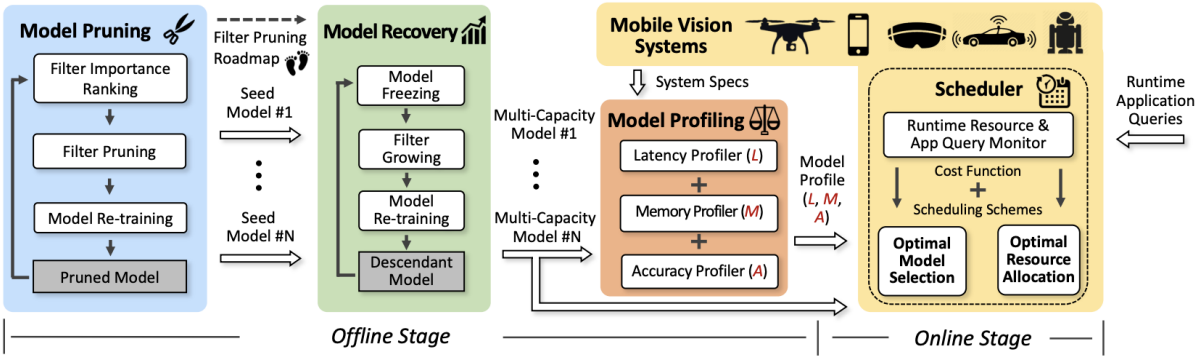

Figure 2: NestDNN architecture [3].

NestDNN is a framework that takes the dynamics of runtime resources into account to enable resource-aware multi-tenant on-device deep learning for mobile vision systems. NestDNN enables each deep learning model to offer flexible resource-accuracy trade-offs [3]. Fig. 2 illustrates the architecture of NestDNN in detail, which is split into an offline stage and an online stage. The offline stage consists of three phases: model pruning, model recovery, and model profiling [3]. Table 1 illustrates the five models involved in NestDNN.

Table 1: Five terminologies involved in NestDNN.

Terminology	**Explaination**
Vanilla Model	A basic and unmodified deep learning model (e.g. ResNet trained on ImageNet)
Pruned Model	A model optimized by removing less significant filters.
Seed Model	The smallest model after pruning, serving as a starting point for further development.
Descendant Model	Models evolved from the seed model, each offering a unique resource-accuracy trade-off.
Multi-Capacity Model	The final model that encapsulates the capabilities of all previous models, providing multiple capacities within a single model.

3.4 Algorithm One-shot Whole Network Compression

"One-shot Whole Network Compression" is a simple way to compress CNNs to make them more suitable for fast and low-power applications on the mobile. This algorithm can be divided into three steps: rank selection, tensor decomposition, and fine-tuning. Each step can be implemented easily by using public tools such as VBMF for rank selection, Tucker decomposition for tensor decomposition, and Caffe for fine-tuning [4].

3.5 Algorithm Frequency Regularization

The frequency regularization algorithm is divided into two steps, the dynamic tail-truncation and inverse discrete cosine transform (IDCT). The dynamic tail-truncation involves retaining parameters in the frequency domain when training the neural networks. The segments that illustrate high-frequency zigzag information are truncated. This procedure is executed using a dot product with a zigzag mask matrix to maintain differentiability [1]. After the procedure, the algorithm utilizes IDCT to rebuild the spatial tensors, which are subsequently employed as standard learning kernels in the networks.

4 What you Plan To Do

There are five sub-topics for implementing Frequency Regularization on Android devices. The first one "Implement Frequency Regularization" was done in Basu's lab at the University of Alberta (2023). The remaining topics are expected to be finished before the course ends.

- **Implement Frequency Regularization:** Implement the Frequency Regularization in order to compress the redundant information on nerual network. Make sure that the accuracy and velocity is acceptable for the nerual network.
- **Configure Emulator:** In this paper, we choose Android Studio as our emulator. Install it successfully on our own local machines.
- **Implement PyTorch Library:** Implement PyTorch Android libaray to the emulator. Collect and analyze the error information if we get.
- **Adjust Dependencies:** Potential errors may probably caused by Android PyTorch Library. Since the library on android lacks of improvement not as much as laptop.
- **Adjust Frequency Regularization:** We can also try to change something on the algorithm to let the new one fitting the Android system.

5 How You Plan to Implement Your Ideas (Timeline)

For the class 414, we will only focus on the Frequency Regularization (FR). The basic idea is to analyze the existed paper in Basu's lab [1]. After that, we also need to explore the basic structure of the PyTorch Mobile [7].

Fig. 3 shows the timeline that we are planning for the implementation.

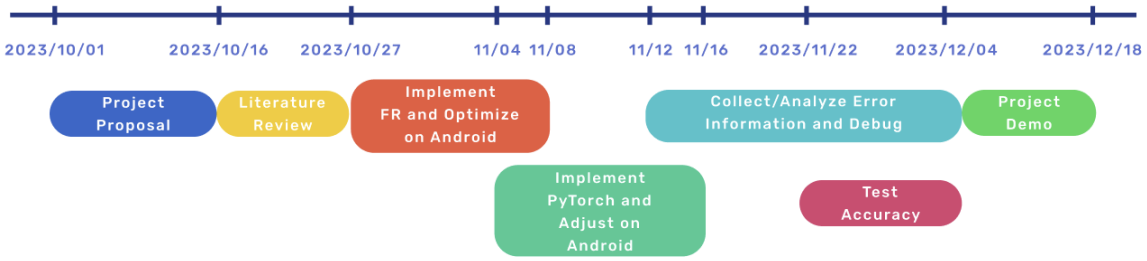

Figure 3: Timeline for FR implementation on Mobile Devices

6 Short Description of 5 LABS (Milestones)

In this paper, we will divide the project into five labs (milestones), which can help us to double check our progress and adjust it timely.

- Deploy the Frequency Regularization algorithm with the UNet (one basic Neural Network) on the computer and adjust some parameters for the Android emulator.
- Attempt to implement PyTorch library on the Android emulator and summarize the error messages in order to further analyze.
- Analyze the error messages and explore the methods to change PyTorch library on the Android.
- Attempt to change something on the FR algorithm to make it match the current Android system.
- Test all functions and neural networks to check the effectiveness and accuracy on the Android system after importing the compressed neural network. We will compare the result between the computer and mobile devices.

7 Literature Review

7.1 Review of AI Benchmark for Running Deep Neural Networks

Nowadays, with the rapid improvements in mobile devices, they have reached the levels that can only be achieved on the previous PCs. Although there are a lot of improvements, mobile devices can still not run all the artificial intelligence applications smoothly. In the existing paper [6], Ignatov explores further the current state of deep learning within the Android system.

In the early 1980s, digital signal processors (DSPs) started to appear in people's view. Fig 4 shows us the image of DSP. Some people may confused what is DSPs. Digital Signal Processors (DSPs) are designed microprocessors for efficiently implementing digital signal processing algorithms [8]. DSPs can possess a specialized hardware architecture easily. What's more, they can also handle the instruction set optimized for processing digital signals, which is considered the primary functionality. Usually, we use DSPs for processing audio, video, temperature, and the majority of types of signals. It is good for catching real-time signals and analyzing them at the same time. Nowadays, DSPs have more broad applications, such as digital TVs, radar, sonar systems, and more. To understand how DSP works, we need to figure it out the internal structure of DSP.

Figure 4: Digital signal processor (DSP) [8].

Fig. 5 illustrates the internal structure of DSP, which involved in four main components: program memory, data storage, calculation engine, and input/output. Program memory stores the program, processing the data. Data storage takes charge for storing the information, which are waited to be handled. Calculation engine, like its name, is related to the calculation, which used for perfroming some mathematical processing. And the input/output will provide a series of functions, making a bridge with the outside hardware components.

DSPs started to appear in mobile devices (i.e. mobile phones) in the 1990s, working for voice coding and compression for the first time. The power of DSPs increased significantly as time went on. For now, the powerful DSPs can easily used for some AI and deep learning applications, such as image classifiers, facial recognition, and image style transfer. But there are still some large and complex Neural Networks such as BigGAN and StyleGAN.

In this existing paper [6], a fabless semiconductor company, Qualcomm, introduced the Snapdragon Neural Processing Engine (SNPE) SDK, which is a powerful software development kit, including CPU, GPU, and DSP. Applying this SDK, allowed more Neural Networks

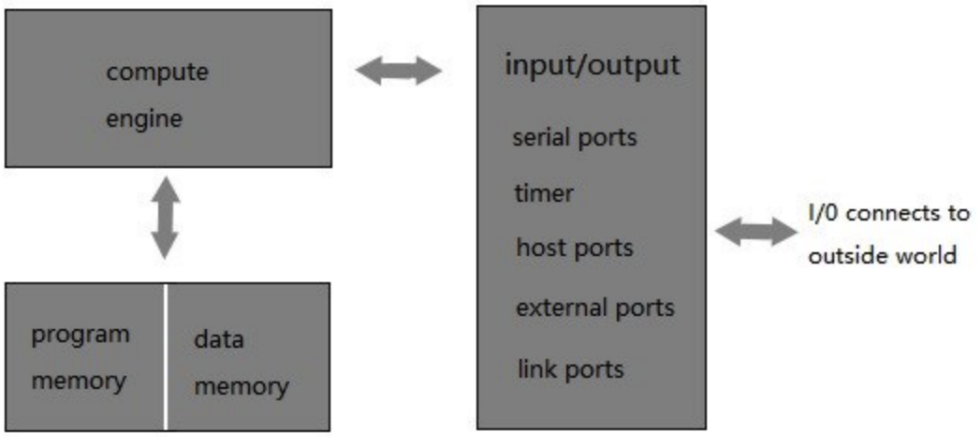

Figure 5: DSP internal structure [8].

to be deployed on mobile devices. Moreover, another company HiSilicon Kirin, introduced a specialized Neural Processing Unit (NPU) for fast vector and matrix-based computations commonly used in AI. In Ignatov's research, he also mentioned other big companies like Media Tek and Samsung, which also had technology improvements for mobile devices, being good for deployment of Neural Networks on mobile devices.

AI Benchmark is just a tool or test, which is used to measure and evaluate the performance of AL algorithms on Android devices. Most researchers use this tool to determine the efficiency of the up-to-date hardware, which is manufactured by the majority of big companies.

7.2 Review of Neural Network

In order to learn how the four algorithms, ELM [2], NestDNN [3], "One-show Whole Network Compression" [4], and Frequency Regulation [1] work, we have to figure out what is Neural Network. A Neural Network is a computational model, including many interconnected elements, which are inspired by the biological nervous system [9]. The network usually has three layers: input, hidden and output layers. In a nutshell, it is an attempt to make machines simulate the way the human brain works. For the learning process of Neural Networks, the first process is always called backpropagation, adjusting the weights of the connections based on the data the model trained on. During its training process, the network just feeds data to the input layers and computes an output. Then compare the output with the expected one, adjusting the weights of the connections to minimize the possible errors.

7.3 Review of Compression of Neural Network

With the increasing size of the neural network and the increasing amount of its complexity, it is quite difficult to deploy the neural network on devices, especially mobile devices with limited resources. In order to solve this problem, neural network compression was born at the right moment. The general goal of compression is to make the neural networks become smaller and compact but still maintain the general accuracy simultaneously. In the existing paper [10] of the compression, we can find the common ways for compressing networks below.

To understand further about compressing neural networks without losing too much accuracy and performance, we found four existing techniques. Those techniques are pruning [11], quantization [12], knowledge distillation [13], and tensor decomposition [14] respectively.

For the creative pruning step [11], it enhances its compressing networks by leveraging differentiability during the backpropagation stage. It not only allows for end-to-end learning but also shortens the training time significantly. In most of the algorithms nowadays, pruning is a

significant step. The reason is that algorithms want to prune or truncate the unimportant data to reduce the parameters of the neural networks.

The quantization in the proposed paper [12] stated that pruning can also be implemented with the quantization without interference. The main idea of quantization is to reduce the number of bits that are represented in each weight. Generally, it manages the distribution of network weights, minimizing the quantization errors. Thus, it achieves a more efficient compression model.

The idea behind the knowledge distillation is to design effective Convolutional Neural Networks (CNNs) under resource-limited environments. It is mostly focusing on multi-tasking learning. The paper proposed that knowledge distillation works well in transferring a complex model to a single distilled model. The method also nearly maintains all the improvements from the original performance [13].

For the tensor decomposition [14], it unveils the latent relations among complex structures in the neural networks, providing a pathway toward optimization. Thus, it makes sure it the holistic and effective.

In our words, we can always consider the compression as a process, which converts a large and complex book to a simple and summary abstract. Moreover, the process should also retain the main content and information.

7.4 Review of High-Performance Extreme Learning Machine (ELM)

The Extreme Learning Machine is a learning algorithm for single-hidden-layer feedforward neural networks, where the weights and biases of its hidden layer are generated randomly. The existing paper [2], reveals that because of this feature, ELM usually has a faster speed than other traditional neural network learning algorithms. Fig. 6 illustrates the main three layers in ELM.

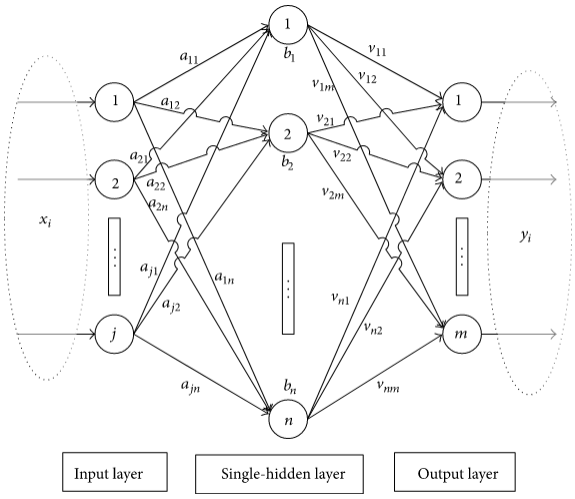

Figure 6: A simple illustration on how ELM process works [15].

Currently, ELM has become more and more important since the rise of edge computing. In Cao's existing research [16], we can find that the traditional cloud computing is not the best way anymore because of large-scale data, slow response speed and poor security. Edge computing acts as an important role, in order to solving the increasing volume of data and processing demands. Since edge computing can operate in close proximity to data sources, it enables more efficient data processing. According to the characteristics of edge computing, it allows algorithms to run on mobile devices directly, providing a more convenient computing environment for users. Despite the plenty of advantages of edge computing, it still has some limitations for the implementation of high-quality Neural Network applications.

Akusok and his teammates found a new ELM implementing method, which is tailored for mobile devices with GPU acceleration. Their method combines the speed of a direct non-iterative solver with minimal memory requirements, making it suitable for edge computing

situations. By comparing the performance between two different types of devices, iPad Pro and PC, it shows a result that the GPU on the iPad performs way better than the CPU on the laptop, which trained a 19000 neuron model using less than one gigabyte of memory successfully. Because of Akusok's experiment [2], tells us the potential for running Neural Networks on limited mobile devices and it also highlights the significance of the ELM.

ELM also has some limitations. As the existing paper [2] mentioned, first of all, ELM is sensitive to certain types of data (i.e. ill-conditioned data), making it less accurate and reliable in some special situations. It also has a tendency to overfit, which means it might perform really well on training data but not as well on unseen data. Moreover, we need to remember that there is no one-size-fits-all setting for ELM. Different situation needs to change different parameters for ELM. In our words, finding the correct settings for ELM is just like a guessing and decision game, which is not always ideal for consistent results.

7.5 Review of NestDNN

Currently, more and more mobile devices, like smartphones, drones, and augmented reality headsets, are trying to change our daily lives, leading to the increasing demand for mobile devices. Since our researchers want to run the Neural Networks on our own mobile devices, we find that an algorithm called NestDNN may probably solve this problem.

For this topic, there are two main challenges that need to be solved: Multi-tenant Characteristics and Context Changes in Mobile Settings. Mobile devices often run multiple applications at the same time, and their resource availability can fluctuate because of various factors such as launching new applications or changing application priorities. This is also the main reason our own mobile devices can not support large Neural Networks since they do not have enough memory space. Here is a new algorithm NestDNN which is designed to address the dynamic nature of runtime resources in mobile vision systems. The main purpose of this is to enable resource-aware multi-tenant on-device deep learning, allowing each deep learning model to provide flexible resource-accuracy trade-offs. Through this idea, it can always select the optimal resource-accuracy trade-off for each model dynamically, making sure to utilize all the existing memories effectively and maximizing the performance of the implementation.

Fig. 3.3 illustrates the architecture of NestDNN, which is divided into an offline stage and an online stage. The existing paper [3] about NestDNN discussed a new model pruning and recovery scheme inside the offline stage to achieve deployment of Neural Networks on mobiles.

To explain the detail of how NestDNN tackles the two challenges mentioned above, NestDNN utilizes a state-of-the-art *Triplet Response Residual* (TRR) approach to rank the filters in a basic and unmodified deep learning model, such as Vanilla Model in Table 1.

The purpose of TRR is to measure the importance of filters and rank filters using their relative importance [3]. The researchers applied the method of calculating the *L2* distances of feature maps between the positive anchor image and negative anchor image. In each epoch, the algorithm will prune the unimportant filter. After the process, it re-trained the model after pruning to compensate for the accuracy loss during the filter pruning.

In the existing paper [3], in order to solve context changes in mobile settings, NestDNN also explores an easy model freezing and filter growing, in order to generate the multi-capacity model in an iterative manner. The employment is happening during the model recovery phase where it uses the seed model as the initial point. During each iteration, model freezing is applied to freeze the parameters of all the model's filters [3]. Then, filter growing is applied to add the pruned filters. Doing these processes will create a descendant model that has a larger capacity. The accuracy of this model is re-obtained through the retraining process. Creating the descendant model is the same as training all other models, it is grown based on the precedent data. Finally, in the last term, the multi-capacity model is created. Since the algorithm repeats the iteration and creates several descendant models, after all iterations, it contains many capacities from the previous ones. Therefore, it is named the multi-capacity model.

In the last procedure of the offline stage, the architecture introduces model profiling. There is a profile to be generated by each multi-capacity model. As mentioned the multi-capacity model is created by multiple descendant models, a profile includes the inference accuracy, memory footprint, and processing latency [3].

Finally, the algorithm comes to the online stage. In this procedure, there is a scheduler that monitors the events. The event is about the change of runtime resources. The paper proposed

that when the scheduler detects any changes, the algorithm checks the profiles of all running applications, and chooses the optimized descendant model from each application. The reason to do this is to allocate the optimized resources to maximize the accuracy and minimize the processing latency of all concurrent applications [3].

In the existing experiment [3], we can find that NestDNN achieves up to a 4.2% increase in inference accuracy, doubles the video frame processing rate, and reduces energy consumption by 1.7 times. Because of this experimental result, NestDNN is indeed a useful and easy method for deploying deep learning on mobile devices.

7.6 Review of One-shot Whole Network Compression

There are many constraints of mobile devices, such as computing power, battery capacity, and memory capacity. Yong-Deok Kim [4] illustrates his method to solve this problem by presenting a new scheme for compressing convolutional neural networks (CNNs).

For this topic, there is one main challenge in deploying CNNs on mobile devices: limited resources. Running some deeper CNNs, especially for complex tasks like ImageNet classification, is still hard to implement. In the existing paper written by Kim [4], there is a new method called "one-shot whole network compression" which is a simple and effective way to compress CNNs. This existing algorithm can be divided into three steps: rank selection with variational Bayesian matrix factorization (VBMF), Tucker decomposition on the kernel tensor, and fine-tuning to recover any loss in accuracy. Fig. 7 shows the scheme of these three steps of this algorithm.

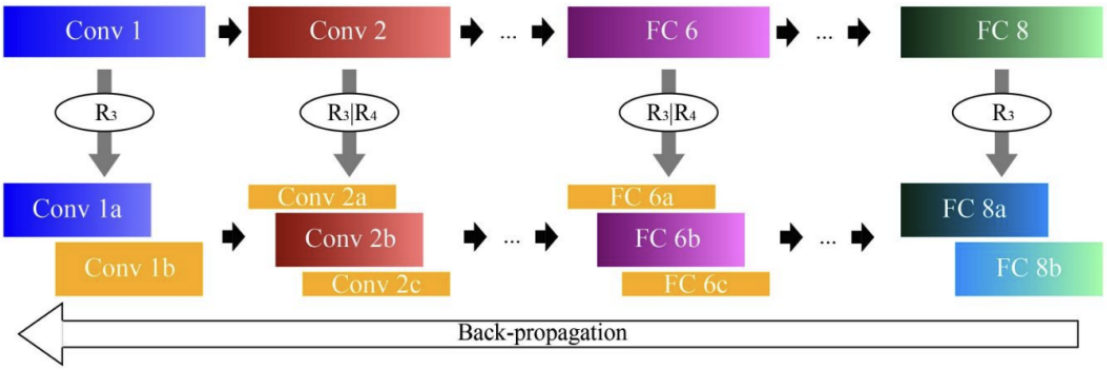

Figure 7: The scheme of one-show whole network compression, which includes three steps [4].

In Kim's paper [4], rank selection with variational Bayesian matrix factorization (VBMF) aims to determine the rank of each layer in the neural network. By applying VBMF on each kernel tensor, it can find noisy variance, rank, and rank recovery easily in an automatic way. It makes good preconditions for Tucker decomposition. Tucker decomposition on the kernel tensor is used to compress both convolutional and fully-connected layers of the neural network. It preserves the essential features, meanwhile, compressing successfully. After doing the two steps, Kim's algorithm applies find-tuning to recover the loss in accuracy. This process makes sure that the compressed model retains its performance and accuracy close to the uncompressed model.

Since the simplicity, effectiveness, and it can compress the whole network, "One-shot Whole Network Compression" can be considered as a good algorithm. However, this algorithm has some potential limitations as well. According to the experiment in Kim's paper, it is not fully investigated whether the rank selected is optimal or not. Moreover, it may affect the speed-up ratio since it lacks cache efficiency, in the 1×1 convolution.

7.7 Review of Frequency Regularization

Until now, many neural networks have been discovered or created by researchers and enhanced the features for model training. However, there exists a problem where the size of the neural networks is growing bigger. It can lead to concerns about information and storage overload because of the numerous numbers of parameters. If we want to implement the neural networks on limited hardware devices, applying some neural network compression algorithms is a method to fit the specifications.

In most neural network research, we deduce that the network performance is because of the parameters. For instance, if we implement image segmentation, high-frequency components are not as crucial for the task since they capture the details of the image which is not necessary for the segmentation. Therefore, applying the frequency domain transform is a practical method [1]. Also, since the frequency domain transform converts the signals into numerical data, we can easily manipulate the quantitative image information for the compression algorithm. In frequency regularization, it focuses on implementing the transform to restrict the information redundancy. The reason is because of the interpretation of the complicated architectures of the networks. Frequency domain transforms for compressing networks may not work as expected on the pre-trained models where the result was obtained in the previous research [1], [17].

When applying the neural network compression algorithms, we want to ensure the accuracy of the output compared with the original dataset. However, we will need to understand what shortcomings this algorithm may contain. The frequency regularization algorithm proposed in this paper contains some limitations when implemented in the neural network. During the neural network training, it is assumed that the high-frequency components are not significant. As introduced in the previous section 3.5, the segments that illustrate high-frequency zigzag information are truncated, which may eliminate the necessary details. The researchers reveal this disadvantage when applying frequency regularization in the image generation task. Parameters cannot be truncated too much compared to the task of image segmentation [1]. Another disadvantage mentioned in the paper is regarding the memory usage of the devices. Since the algorithm is handled in the frequency domain, it needs to convert the parameters into the spatial domain for convolutional operations. Therefore, the implementation of the algorithm requires more memory during the training.

Although there may still exist some shortcomings, such as the total training time, we are still researching frequency regularization to be our target to implement neural network compression. The reason is because of the main following reasons: First, we can apply this algorithm to general neural network architectures. Second, it does not reduce the accuracy percentage. We will introduce them respectively.

We can achieve frequency regularization by two processes. The first one is the dynamic tail truncation, and the other is the inverse discrete cosine transform (IDCT). Fig. 1 shows the frequency regularization process.

The dynamic tail truncation is used in this algorithm. From the research paper, the authors concluded that most parameters in convolutional networks can be eliminated. However, there exists an issue where the parameters that are not eliminated may affect the backpropagation. Therefore, the training loss may stay the same in many epochs [1]. The strategy of dynamic tail truncation addresses this issue by continuously setting a few tail elements to 0 in every epoch but not directly setting most of them at once.

The idea behind the frequency regularization is to maintain the tensors in the frequency domain instead of the spatial domain. By implementing this, it allows the tail elements to be eliminated by a dot product with the zigzag mask matrix. After the dynamic tail truncation mentioned above, the algorithm then inputs the tensors to the inverse discrete cosine transform. The purpose of the inverse discrete cosine transforms is not only the role of reconstructing the spatial tensors. An inverse discrete cosine transform process can be implemented for tensors with any form. It leads to an important functionality, for instance, applied to any tensors from neural networks [1].

7.8 Review on Implementation on Mobile Devices

Nowadays, mobile devices have more and more powerful hardware so that any neural network with fewer parameters can run. For instance, the Android library mentioned the pre-trained models are implementable, and we have successfully implemented them. However, in our project, we are trying to implement more advanced neural networks which may have more parameters involved. We also target to successfully apply the frequency regularization to the neural networks and implement the neural network on the mobile devices. In the following instances, some researchers successfully implemented the neural network on mobile devices.

In Jin's paper [18], they implement deep convolutional neural networks on a mobile coprocessor by executing an efficient routing strategy. It is to maximize the utilization of available hardware resources. Moreover, the performance is maintained to a high standard in real-world applications. The implementation is an example of a successful case to run a neural network on mobile devices by the hardware accelerator. As mentioned in previous sections, although the processor of mobile devices has dramatically increased over decades, the real-time performance is still inferior to computers. A simple explanation of the preparation for the hardware implementation in Jin's research is to apply a memory router on their hardware compromisation. The purpose of memory routers is to process all incoming and outgoing data streams. It can also distinguish works from one or more collections. When running the algorithm, the software compiler will allocate the hardware resources to the router for processing jobs. One of the main reasons for implementing a memory router is to make the implementation more efficient. The paper states that the efficient design of the memory routing can allow the maximum utilization that leads to the result in peak performance. The idea behind the implementation is to target the efficiency and scalability of mobile platforms for deep convolutional neural network [18].

In Castanyer's paper, they try to identify the most challenging part when deploying deep learning-based software in mobile applications. It includes what are the challenges that we may encounter during the implementation of the neural network on mobile devices and how we can control the complexity while maintaining a high amount of accuracy. The researchers used PyTorch and the Android framework to design some studies and set goals to diagnose the challenges. After some experiments, they discovered a few categories that needed to be considered. The first category is the challenge that is verified to be real and concerning. It includes software-data dependency, sustainability, and frameworks in the early stages. The second category is the challenge that has been identified but is not faced in the study. It includes explainability. The last category is the newly defined challenges based on their experiment studies. The researchers mainly focus on fixing the first category by applying the data augmentation technique to reduce the data-software dependency and increase the sustainability of the applications [19].

References

[1] C. Zhao, G. Dong, S. Zhang, Z. Tan, and A. Basu. Frequency regularization: Reducing information redundancy in convolutional neural networks. *IEEE Access*, September 2023. Department of Computing Science, University of Alberta, Edmonton, AB.

[2] Anton Akusok, Leonardo Espinosa Leal, Kaj-Mikael Björk, and Amaury Lendasse. High-performance elm for memory constrained edge computing devices with metal performance shaders. In Jiuwen Cao, Chi Man Vong, Yoan Miche, and Amaury Lendasse, editors, *Proceedings of ELM2019*, pages 79–88, Cham, 2021. Springer International Publishing.

[3] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. pages 115–127, 10 2018.

[4] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. 11 2015.

[5] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip Yu. Not just privacy: Improving performance of private deep learning in mobile cloud. 09 2018.

[6] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In Laura Leal-Taixé and Stefan Roth, editors, *Computer Vision – ECCV 2018 Workshops*, pages 288–314, Cham, 2019. Springer International Publishing.

[7] Pytorch mobile: End-to-end workflow from training to deployment for ios and android mobile devices, 2023. Accessed: 2023-10-15.

[8] Utmel. An overview of digital signal processor, Jul 2020.

[9] Phil Picton. *What is a Neural Network?*, pages 1–12. Macmillan Education UK, London, 1994.

[10] James O' Neill. An overview of neural network compression, 2020.

[11] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015.

[12] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[13] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Improving multi-task deep neural networks via knowledge distillation for natural language understanding, 2019.

[14] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. Learning compact recurrent neural networks with block-term tensor decomposition, 2018.

[15] Kemal Erdem. Introduction to extreme learning machines, May 2020.

[16] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE Access*, 8:85714–85728, 2020.

[17] Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong. Frequency-domain dynamic pruning for convolutional neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[18] Jonghoon Jin, Vinayak Gokhale, Aysegul Dundar, Bharadwaj Krishnamurthy, Berin Martini, and Eugenio Culurciello. An efficient implementation of deep convolutional neural networks on a mobile coprocessor. In *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 133–136, 2014.

[19] Roger Creus Castanyer, Silverio Martínez-Fernández, and Xavier Franch. Integration of convolutional neural networks in mobile applications. In *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*, pages 27–34, 2021.