

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.

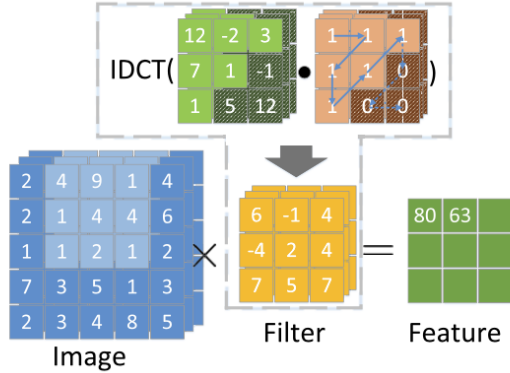


Figure 1: Illustration of the proposed frequency regularization. The tail elements of tensors in the frequency domain are truncated first, then input into the inverse of the discrete cosine transform to reconstruct the spatial tensor for learning features [1].

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.

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 sim-

3 Brief Summary of Existing Work

We cover the methods for implementing neural networks on the mobile over four sections. In Section 3.1, upgrading hardware on mobile devices and which parts are introduced. At the algorithm level, 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 3.2. 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 3.3. 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 3.4.

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 NestDNN

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

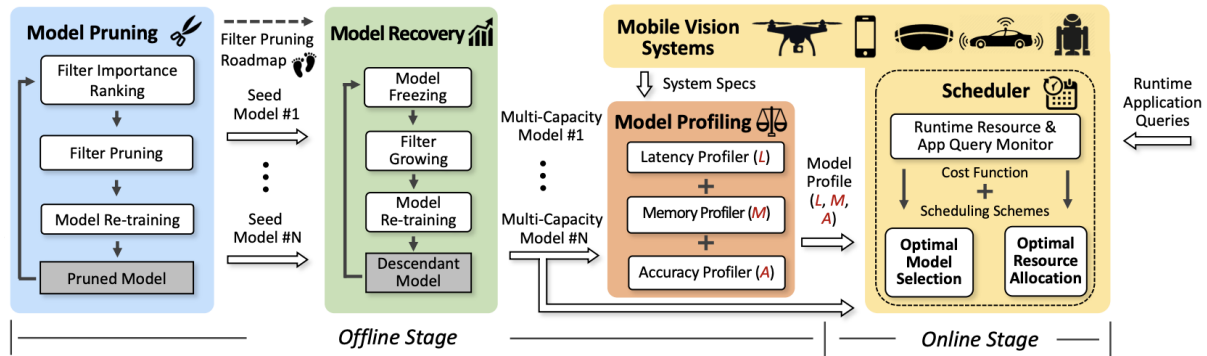


Figure 2: NestDNN architecture [3].

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]. There are five models involved in NestDNN as shown below:

- **Vanilla Model:** Off-the-shelf deep learning model (e.g., ResNet) trained on a given dataset (e.g., ImageNet).
- **Pruned Model:** Intermediate result obtained in model pruning stage.
- **Seed Model:** The smallest pruned model generated in model pruning which meets the minimum accuracy goal set by the user. It is also the starting point of model recovery stage.
- **Descendant Model:** A model grown upon the seed model in model recovery stage. It has a unique resource-accuracy trade-off.
- **Multi-Capacity Model:** The final descendant model that has the capacities of all the previously generated descendant models.

3.3 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].

3.4 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 depict 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 neural network. Make sure that the accuracy and velocity is acceptable for the neural 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 library 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

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].

Below figure shows the timeline that we are planning for the implementation.

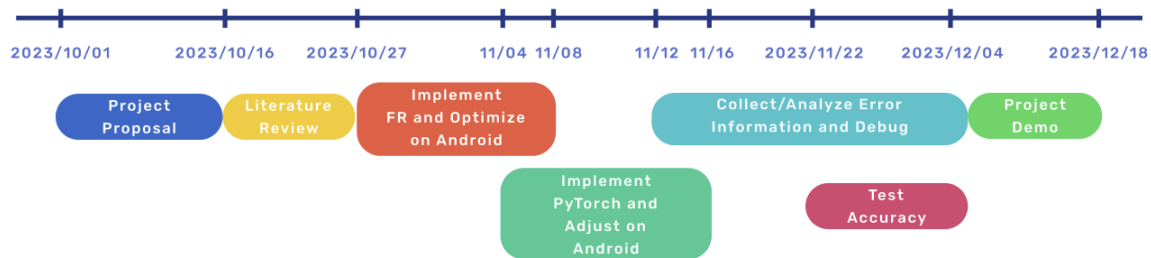


Figure 3: Timeline for FR implementation on Mobile Devices

6 Short Description of 5 LABS

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 High-Performance ELM

7.2 Review of AI Benchmark

7.3 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. 3.2 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 3.2. In the existing paper [3], here is the mathematical equation of TRR below. 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.

$$TRR_i = \|F_i(\text{anc}) - F_i(\text{neg})\|_2^2 - \|F_i(\text{anc}) - F_i(\text{pos})\|_2^2 \quad (1)$$

where:

F denotes the generated feature map;

Anc, pos, neg denotes a triplet includes an anchor image, a positive image, and a negative image respectively.

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.

7.4 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. 4 shows the scheme of these three steps of this algorithm.

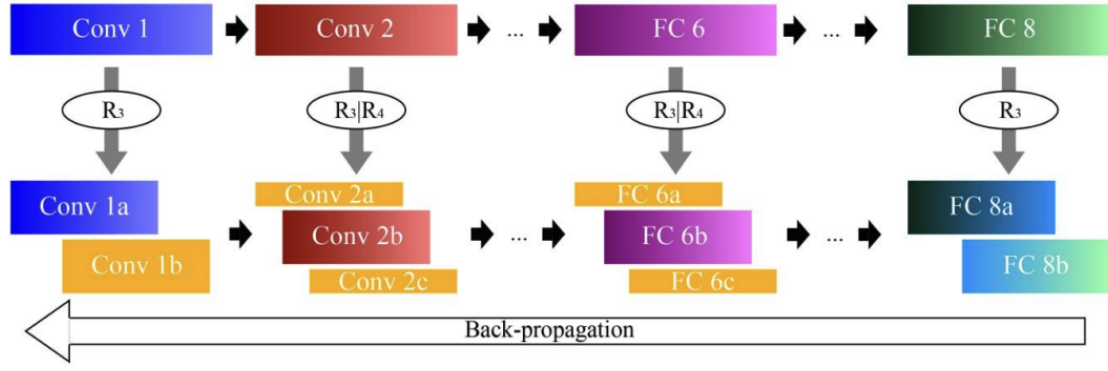


Figure 4: The scheme of one-show whole network compression, which includes three steps [4].

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.

7.5 Review of Frequency Regularization

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. Nstdnn: 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, 2023.