PERFORMANCE ANALYSIS OF CONVOLUTIONAL NEURAL NETWORKS EXTENDED WITH PREDEFINED KERNELS IN IMAGE CLASSIFICATION

ARASH FATEHI

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITI MALAYA
KUALA LUMPUR
2022

PERFORMANCE ANALYSIS OF CONVOLUTIONAL NEURAL NETWORKS EXTENDED WITH PREDEFINED KERNELS IN IMAGE CLASSIFICATION

ARASH FATEHI

DISSERTATION SUBMITTED IN FULFILMENTOF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE (APPLIED COMPUTING)

**FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITI MALAYA
KUALA LUMPUR**

2022

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Arash Fatehi

Matric No: 17220613/1

Name of Degree: Master of Computer Science (Applied Computing)

Title of Thesis: Performance Analysis of Convolutional Neural Networks Extended with Predefined Kernels in Image Classification

Field of Study: Artificial Neural Networks

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date: 08/May/2022

Subscribed and solemnly declared before,

Witness's Signature Date: 08/May/2022

Name:

Designation:

PERFORMANCE ANALYSIS OF CONVOLUTIONAL NEURAL NETWORKS EXTENDED WITH PREDEFINED KERNELS IN IMAGE CLASSIFICATION

ABSTRACT

While Machine Learning aims to solve more challenging problems, Artificial Neural Networks (ANN) become deeper and more accurate. Convolutional Neural Network (CNN) is not an exception and state-of-art architectures consist of millions of learnable parameters. Aiming for better performance, these networks become more complex and computation intensive. Also, with the rise of IoT devices and edge computing, the importance of model acceleration and reduction of needed computing resources become more curial for training neural networks. Model acceleration and compression techniques often target reducing inference latency and memory usage, and research about reducing the training time was limited to two previous studies. Considering numerous use cases of CNNs, reducing the training time and processing cost is beneficial. CNNs are universal functions and in the case of supervised learning, they will converge to a specific desired function after training. In this research, predefined image processing kernels were merged into CNN's architecture to help the network to converge faster for the use case of image classification. This method can be applied to any classification task of multi-channel sensory data. The efficiency of the method was tested through an experiment on ImageNet, Cifar10, and Cifar100 datasets. The effects on performance were architecture dependent. In the case of CNNs with residual blocks and skip connections, the model was not able to leverage the provided information by image processing filters to converge faster, but CNNs based on VGG had a significantly (up to 125%) faster training time, which is beneficial for training models on embedded devices and edge computing.

Keywords: Deep Learning, Convolutional Neural Networks, Model Acceleration

ANALISIS PRESTASI RANGKAIAN SARAF KONVOLUSI YANG DIPERLUASKAN DENGAN KERNELS DIJELASKAN DALAM KLASIFIKASI IMEJ

ABSTRAK

Walaupun Pembelajaran Mesin bertujuan untuk menyelesaikan masalah yang lebih mencabar, Rangkaian Neural Buatan (ANN) adalah lebih mendalam dan tepat. Rangkaian Neural Konvolusi (CNN) tidak terkecuali dan seni bina terkini terdiri daripada berjuta-juta parameter yang boleh dipelajari. Menyasarkan prestasi yang lebih baik, rangkaian ini menjadi lebih kompleks dan pengiraan lebih intensif. Juga, dengan peningkatan peranti IoT dan pengkomputeran tepi, kepentingan pecutan model dan pengurangan sumber pengkomputeran yang diperlukan menjadi lebih penting untuk melatih rangkaian saraf. Teknik pecutan dan mampatan model menyasarkan pengurangan kependaman inferens dan penggunaan ingatan, dan kajian tentang mengurangkan masa latihan dihadkan kepada dua kajian terdahulu. Mengambilkira kes penggunaan CNN, mengurangkan masa latihan dan kos pemprosesan adalah berfaedah. CNN ialah fungsi universal dan dalam kes pembelajaran diselia, ia akan bertumpu kepada fungsi tertentu yang dikehendaki selepas latihan. Dalam kajian ini, kernel pemprosesan imej yang telah ditetapkan telah digabungkan ke dalam seni bina CNN untuk membantu rangkaian menumpu lebih cepat untuk kes penggunaan klasifikasi imej. Kaedah ini boleh digunakan untuk sebarang tugas pengelasan data sensor berbilang saluran. Kecekapan kaedah telah diuji melalui eksperimen pada dataset ImageNet, Cifar10, dan Cifar100. Hasil ke atas prestasi adalah bergantung kepada seni bina. Dalam kes CNN dengan baki blok dan sambungan langkau, model tidak dapat memanfaatkan maklumat yang diberikan oleh penapis pemprosesan imej untuk menumpu lebih cepat, tetapi CNN berdasarkan VGG mempunyai masa latihan yang lebih pantas (sehingga 125%), iaitu bermanfaat untuk model latihan pada peranti terbenam dan pengkomputeran tepi.

Kata kunci: Pembelajaran Dalam, Rangkaian Neural Konvolusi, Pemprocesan Model

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

ANN	:	Artificial Neural Network
CNN	:	Convolutional Neural Network
ML	:	Machine Learning
RNN	:	Recurrent Neural Network
GAN	:	Generative Adversarial Network
VAE	:	Variational Auto Encoder
GO	:	Gradient Optimization
FFD	:	Feed Forward Network
SR	:	Super Resolution
GFNN	:	Generalized Filter Neural Network
DCT	:	Discrete Cosine Transform

CHAPTER 1: INTRODUCTION

1.1. Research Background

Complex spatial pattern recognition like detecting and labeling objects in an image, for a long time considered a daunting, if not an impossible task, till a breakthrough happened and Krizhevsky et al. (2012) showed that Artificial Neural Networks (ANN) can be trained for the tasks like image classification or object detection and are capable of much more. In contrast to classic programs, an ANN will be trained instead of programmed. The idea behind ANNs based on simulating the activation of neurons, and in most ANN architectures, stacks of artificial neurons construct a layer, and the connection between these layers causes neurons in the next layer to activate. There are numerous ANN architectures with diverse applications, including Feed Forward Neural Networks (J. Wang et al., 2018), Recurrent Neural Network (RNN) (Chandra & Sharma, 2017), Convolutional Neural Networks (CNN) (Krizhevsky et al., 2017), Generative Adversarial Networks (GAN) (Creswell et al., 2018), Variational Auto-Encoders (VAE) (Hou et al., 2017), etc.

Convolutional Neural Network combines the concept of convolutions with ANN. During the training, the convolutions will learn how to extract useful features from the datasets. Performing the daunting task of feature extraction automatically, combined with their high accuracy, makes CNN a perfect choice for many use cases like Image Processing, Natural Language Processing, Data-Driven Personalized Advertising (J. A. Choi & Lim, 2020), Genome Mapping (Agarwal & Shendure, 2020), etc. The focus of this work is on using CNNs in Image Classification. Image Classification is a classification task in machine learning that maps a set of raster data to their labels. Image classification with CNNs has many real-world applications, for example: CNNs play an essential role in cancer recognition. Goyal et al. (2020) have used CNNs for detecting skin cancers, Y. Liu et al. (2019) did the same for detecting lung cancer. Adarme et al. (2020) suggests that CNN-based techniques have the best performance in autonomous deforesting detection in satellite images using image classification. Dung & Anh (2019) demonstrate autonomous crack detection in concrete using CNNs, Fujiyoshi et al. (2019) shows the role of CNNs in autonomous driving. Many use cases exist for the combination of CNNs and image classification which implies their importance in modern science and industry.

For image classification, CNNs are often used in a supervised manner and the network should be trained via samples and their known labels, which involves massive matrix calculations for feedforward and backpropagation phases. As machine learning aims to solve more challenging tasks, the complexity of neural networks increases. Also, advances in the field focus on improving accuracy by making the networks deeper. And the processing power and memory needed for training rise rapidly when the number of network layers increases which makes training of a production ready network costly. To mention a few examples, DenseNet-k=24 (Huang et al., 2017) with <u>100</u> layers contains <u>17.2</u> million learnable parameters, ResNet-50 (He et al., 2016) with <u>50</u> layers contains <u>26</u> million parameters, Xception (Chollet, 2017) with <u>71</u> layers have <u>23</u> million learnable parameters, additionally these models should be trained on massive datasets like ImageNet (Jia Deng et al., 2009) that contains more than <u>14</u> million images.

With the extensive usage of CNNs and the training cost of a production-ready model, it is crucial to optimize the networks both at the training and interference phases and this work focuses on the model acceleration of CNNs when used in a supervised manner in image classification. There exist several model acceleration methods focused on different aspects of the model's performance that target the inference latency and memory usage. The stat of art model acceleration methods is discussed extensively in Chapter 2. But, to refer to a few of them here: Zhang & Li (2020) suggest performing the convolutional operation in the furrier domain. C. Liu et al. (2019) introduces a method to use redundant kernel removal for gradient optimization (GO) in using CNNs for Super-Resolution (SR). Szegedy et al. (2016) proposed asymmetrical convolution where a d×d convolution is spatially factorized as a sequence of two layers with d×1 and 1×d convolutions which are used in the Inception model, etc.

In some cases, transfer learning solves the problem of costly training by reusing already trained convolutional layers in the target network architecture. The trained convolutional layers are specialized for the original dataset and should be trained for new content to achieve the desired performance, but the needed time and processing is relatively minimal. But using pre-trained convolutional layers is practical only if features of the destination dataset have similarities with the original dataset. Otherwise, domain adaptation would become an obstacle in using transfer learning.

Another approach to model acceleration is to focus on the network architecture, often the first layer of convolutions in trained CNNs becomes similar to edge detection filters, suggesting that adding predefined edge detection kernels to their first layer might help the network to converge faster and reduce the training time. The idea suggested by Jung et al. (2018) and the networks have been termed "Generalized Filter Neural Network (GFNN)". The architecture was implemented and tested by the author on the MNIST dataset, but at the time of writing, no analysis about its performance and side effects when used on more complex datasets has been found during the literature review.

The " Generalized Filter Neural Network" have the potential to decrease the training time with minimal degradation in accuracy and can be useful for training models on embedded or edge devices if verified. But the idea was tested only on the MNIST dataset. The MNIST dataset is not complex enough to validate the efficiency of the method, and a more comprehensive experiment is needed. It is necessary to test the method on different architecture and analyze their performance on more complex datasets like ImageNet, Cifar100, and Cifar10. This research will design and implement multiple GFNNs and will analyze and compare their performance on ImageNet_Resized (Jia Deng et al., 2009), Cifar100 (Krizhevsky & Hinton, 2009), and Cifar10 (Krizhevsky & Hinton, 2009) with their equivalent baseline versions in terms of accuracy, training time, and overfitting.

1.2. Problem Statement

With advances in the field of deep neural networks, state-of-the-art CNN architectures have become more memory and computation intensive, that is troublesome in the case of training CNNs on embedded devices and edge computing (Xu et al., 2018). With the rise of IoT devices and edge computing, it has been critical to develop model acceleration techniques compatible with low memory and processing environments (Chandakkar et al., 2017). Most model acceleration techniques focus on inference and even add complexity to the training phase, thus focusing on reducing the training time of CNNs is beneficial to address the mentioned problem. Extension of Convolutional Neural Network with General Image Processing Kernels (Jung et al., 2018) is a potential technique to reduce the training time in the use case of image classification, but the method is not compatible with multi-sensory input data and their performance has never been empirically tested on datasets with complex features and a high number of classes. Also, by applying the technique the number of learnable parameters reduces that might have a potential side effect of overfitting of the neural network that has not been tested yet.

1.3. Research Questions

i. What are model acceleration techniques and how do they apply to CNNs?

ii. Can model acceleration techniques result in a reducing the training time of CNNs?

iii. How to alter GFNNs to be applicable to multi-channel sensory data?

iv. How do GFNNs perform on multi-channel sensory data? And what is its side effect on the accuracy of the network?

1.4. Research Objectives

i. To review and compare CNN's model acceleration techniques.

ii. To review and find model acceleration techniques that target reducing the training time.

iii. To design and implement a filter architecture to combine image processing kernels with Convolutional layers when applied to multi-channel sensory data.

iv. To validate the performance of GFNNs on multi-channel datasets experimentally.

1.5. Research Significance

CNNs play an important role in the field of machine learning, they are used in various use cases and are applied to different types of input data. Current embedded and edge devices are powerful enough for the inference phase of most of the CNNs, but for the training, a much more powerful computer that is equipped with accelerators like GPUs is necessary. The needed computation and memory for the training of CNNs hinders the embedded devices to learn from the data independently and with the rise of IoT devices and edge computing, it is necessary to develop or modify CNNs architectures to be more compatible with the processing and memory constraint of embedded devices. By reducing the processing and memory footprint of the model while training, it becomes more feasible to implement standalone AI agents that leverage CNNs in their learning pipeline. This research focuses on the reduction of CNNs training resources by merging images processing filters into their architecture to help them converge faster.

1.6. Expected Outcomes

To apply the GFNNs on color images, a new method is developed for merging the image processing filters into the CNNs architectures that is compatible with multi-channel sensory data. The efficiency of the method is measured experimentally to answer below questions:

- Do the GFNNs reach a higher accuracy within a certain number of epochs?
- Do the GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks?
- Are the GFNNs more vulnerable to overfitting?

1.7. Summary

CNNs have a wide variety of use cases and while aim to solve more complex machine learning problems with high accuracy, they have become more memory and computation-intensive, and state of the art architectures consist of millions if not billions of learnable parameters. Most model acceleration techniques focus on reducing the inference latency and even add complexity to the training. Thus, to have independent AI agents capable of learning which leverage CNNs capabilities, it is needed to develop methods of making CNNs architecture more suitable for low memory and processing environment. This research provides such a method by combining image processing filters into the CNN architectures to help them converge faster.

CHAPTER 2: LITERATURE REVIEW

2.1. Introduction

After AlexNet's breakthrough (Krizhevsky et al., 2012) in 2012, deep learning has become the center of attention. Different types of ANNs developed and have been used in multi-disciplinary research. CNN is one of these architectures which received the most attention. Figure 2.1 and Figure 2.2 demonstrate the growth in the number of publications in Web of Science, Science Direct, and IEEE Xplore with "Deep Learning" and "Convolutional Neural Network" in their title, abstract, or keywords.

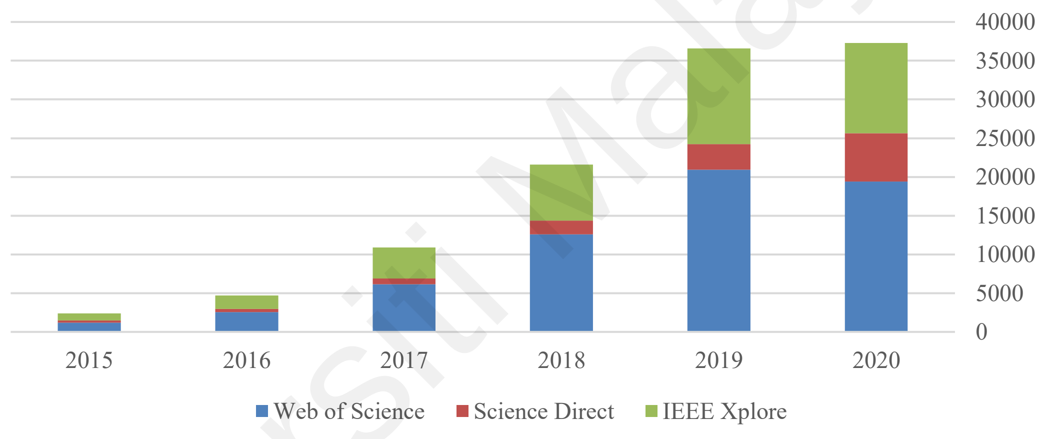

Figure 2.1: Number of publications with the keyword: Deep Learning

As showed in Figure 2.2, considerable portion of research in Deep learning is about Convolutional neural networks.

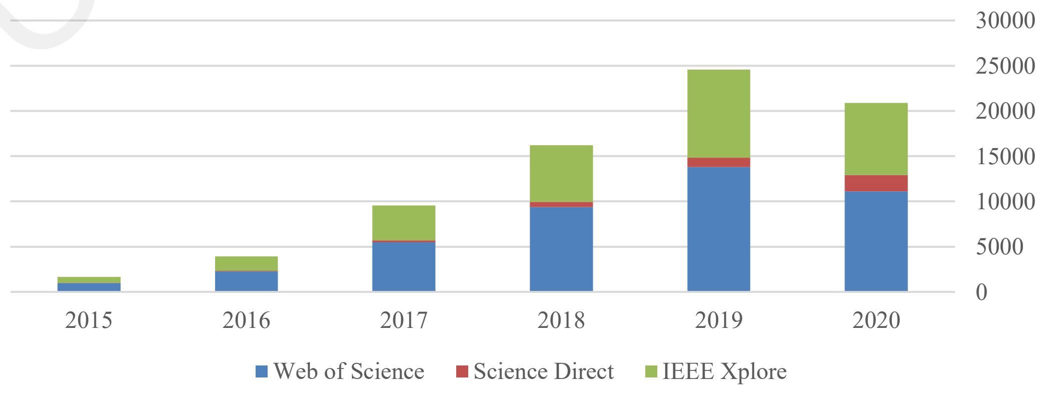

Figure 2.2: Number of publications with the keyword: CNN

CNN's good performance on difficult and sometimes ill-posed machine learning problems in addition to the increasing attention toward them has led to the development of novel CNN architectures. Table 2.1 demonstrates a list of impactful CNN architectures with their properties and contribution. In most architectures, the number of neurons along with the depth of the CNN has a direct relation with the needed processing, and it is reasonable to consider these variables as an indication of the continuous growth of needed memory and computation.

2.2. CNN Architectures

Convolutional neural networks combine the idea of convolutions with artificial neural networks, in the case of most architectures, the network consists of two separate sections. The convolutional layers learn how to extract the sample features during the training and a feed-forward layer is in charge of classification based on features delivered by convolutional layers. But not all the architectures have a simple architecture, for example, architecture of ResNet consist of residual blocks instead of standalone convolutional layers, and as explained in detail later, these residual blocks consist of a skip connection between their convolutional layers.

In case of AlexNet (Krizhevsky et al., 2012), the convolutional layers are preceded with overlapping max pooling layer that reduces the height and width of their output and produces thinner tensors for the next convolutional layer and finally the result of the last convolutional layer is fed to a feed-forward neural network, it also attaches ReLU activation layers after every convolutional and feed-forward layer.

VGG (Simonyan & Zisserman, 2015) has a uniform architecture, the convolutional part consists of 16 convolutional layers, similar to AlexNet the convolutions are 3x3 but with a greater number of kernels for every layer.

Deep neural networks often suffer from gradient vanishing in which the changes to the learnable parameters become infinitely small because of repeated multiplication. ResNet (He et al., 2016) uses the concept of residual blocks that consists of skip connections to jump over some layers to mitigate vanishing gradient when the network becomes deeper. He et al. (2016) provide evidence that network with residual blocks is easier to optimize and can gain more accuracy from their less vulnerability to gradient vanishing that leads to deeper networks.

Table 2.1: Modern CNN architectures and their properties

Architecture	Contribution	Params	Error (%)	Depth	Study
AlexNet	Uses Relu, dropout and overlap Pooling	60 M	ImageNet: 16.4	8	(Krizhevsky et al., 2012)
VGG	Homogenous topology, Uses small size kernels	138 M	ImageNet: 7.3	19	(Simonyan & Zisserman, 2015)
Inception-V4	Split transform and merge idea, Uses asymmetric filters	35 M	ImageNet: 4.01	70	(Szegedy et al., 2017)
Inception - ResNet	Uses split transform merge idea and residual links	55.8 M	ImageNet: 3.52	572	(Szegedy et al., 2017)
ResNet	Residual learning, Identity mapping-based skip connections	25.6 M 1.7 M	ImageNet: 3.6 CIFAR-10: 6.43	152 110	(He et al., 2016)
DelugeNet	Allows cross layer information flow in deep network	20.2 M	CIFAR-10: 3.76 CIFAR-100: 19.02	146	(Kuen et al., 2017)
WideResNet	Width is increased and depth is decreased	36.5 M	CIFAR-10: 3.89 CIFAR-100: 18.85	28 -	(Zagoruyko & Komodakis, 2016)
Xception	Depth wise convolution followed by point wise convolution	22.8 M	ImageNet: 5.5	126	(Chollet, 2017)
ResNeXt	Cardinality, Homogeneous topology, Grouped convolution	68.1 M	CIFAR-10: 3.58 CIFAR-100: 17.3 ImageNet: 4.4	29 - 101	(Xie et al., 2017)

Table 2.1: Continued

DenseNet	Cross-layer information flow	25.6 M 25.6 M 15.3 M 15.3 M	CIFAR-10+: 3.46 CIFAR100+:17.18 CIFAR-10: 5.19 CIFAR-100: 19.64	190 190 250 250	(Huang et al., 2017)
PyramidalNet	Increases width gradually per unit	116.4 M 27.0 M 27.0 M	ImageNet: 4.7 CIFAR-10: 3.48 CIFAR-100: 17.01	200 164 164	(Han et al., 2017)
ResNeXt101 (32x4d) + CBAM	Exploits both spatial and feature-map information	48.96 M	ImageNet: 5.59	101	(Woo et al., 2018)
CMPESE-WRN-28	Residual and identity mappings both are used for rescaling the feature-map	36.92 M 36.90 M	CIFAR-10: 3.58 CIFAR-100: 18.47	152	(Hu et al., 2018)
FixEfficientNet-L2	Using FixRes method for fixing the train-test resolution discrepancy	480 M	ImageNet: 1.3	-	(Touvron et al., 2019)

Note. Partially retrieved from (Khan et al., 2020)

ResNet suffers from the problem of "diminishing feature reuse" which happens when only a small portion of convolutional layers contribute to extracting useful features. In another word, the network can avoid learning. WideResNet (Zagoruyko & Komodakis, 2016) is a variant of ResNet where the author proves that the network performs better by increasing the width and reducing the depth of networks. This was achieved using wide residual blocks. In addition to their dimensions, the major difference between residual block and wide residual blocks is that wide residual blocks perform back normalization and ReLU before convolutions while they are after convolutions in the case of original residual blocks.

Inception network (Szegedy et al., 2017) consists of inception blocks, each block take advantage of having multiple kernel size in separate branches of calculation, and the results of all four branches concatenate at the end of the block. The branches have different kernel size for convolutional layers, first layer consists of 1x1 convolutions, second layer 3x3 convolutions, third layer 5x5 convolutions, and forth layer 3x3 max pooling. Additionally, Inception-v3 introduces the concept of separable convolutions that consists of depth-wise and point-wise convolutions which are explained later while focusing on Xception architecture.

ResNeXt (Xie et al., 2017) combines the ideas behind ResNet, VGG, and Inception. Inception layers should be highly customized, hence adapting the network for new datasets is difficult. Each ResNeXt block is a module with many uniform branches that are repeated through the architecture. ResNeXt adds a new dimension termed cardinality that refers to the number of branches in each block and the result of all branches is aggregated using summation.

Xception (Chollet, 2017) introduces the concept of modified separable convolution which is based on depth-wise and point-wise convolutions. A depth-wise convolution is a combination of per channel $n \times n$ spatial kernel that produce the same number of channels as input, in another word, each kernel in depth-wise convolution is applied on a single channel and is responsible for only one of the channels in the output. Depth-wise convolutions are lighter in comparison to conventional convolutional layers as each kernel is not applied across all the channels. On the other hand, a point-wise convolution is a 1×1 convolution to change the dimension of the output. Xception defines a modified version of separable convolutions. In the original design of separable convolutions in Inception-v3 (Szegedy et al., 2017) the point-wise convolution is performed after the depth-wise convolution, but in Xception the order of operations is reversed. Also, in

Xception separable layer there is no intermediate ReLU nonlinearity in oppose to separable layers in Inception-v3. According to the author of Xception (Chollet, 2017), the Xception without any intermediate activation function outperformed using either ELU or ReLU. It also uses skip connection through the architecture and achieved higher accuracy with the skip connection. The exit flow of the network is like other architectures and the extracted features are forwarded to a fully connected feed-forward network. Xception is claimed to have similar model size to Inception-v3 while outperforming VGG, ResNet, and Inception-v3 while using the residual architecture and skip connections are implemented. But without the skip connections its performance is lower than Inception-v3 and might be fairer to compare the performance of Xception with Residual Inception-v3 instead.

DenseNet (Huang et al., 2017) extends the idea behind ResNet, and each layer receives additional information from all of its preceding layers. In ResNet each residual block passes its stat to the next one, but DenseNet block receives a collective knowledge about the stat of all previous layers. Each DenseNet block consists of Batch-Normalization, ReLU, and a 3x3 convolutional layer that produces feature maps of $\underline{k}$ channel, which $\underline{k}$ is a configurable. To reduce the model size a combination of Batch-Normalization, ReLU, and a 1x1 convolutional layer is used first. As a result, the network can be thinner and more compact with a smaller number of kernels in each block, thus it has higher computation and memory efficiency. DenseNet has numerous advantages to ResNet, due to the skip connections the error can be directly propagated to the earlier layers that leading to strong gradient flow. As mentioned earlier, DenseNet is more efficient computationally, also, since each layer in DenseNet receives the result of all preceding layers, it tends to have a more diversified feature.

PyramidalNet (Han et al., 2017) steadily increases the width of residual blocks instead of keeping the same spatial dimension before the down-sampling. Han et al. (2017) claims that CNNs learning capabilities are limited because of the significant rise in feature maps when the networks are deep, and PyramidalNet design helps to solve the problem.

With significant improvements in CNN's architecture and performance, they consist of millions of learnable parameters that should be trained on millions of samples to achieve high accuracy. Also, with the help of model acceleration techniques, these models achieve reasonable inference latency after training, but as described in the next section, most of these techniques add extra complexity to the training phase. Thus, new methods or architectures are needed to train CNN on edge or embedded devices and create independent AI agents that leverage CNN's capabilities. In the next section state of the art model acceleration techniques are reviewed and their effects on training time, inference latency, and memory consumption are examined.

2.3. Model Acceleration

To accurately handle machine learning problems, CNNs are becoming deeper and contain millions of learnable parameters, training them on large datasets would be costly in terms of time and money. In addition to using hardware accelerators and optimized frameworks, Transfer learning and model acceleration techniques are used to tackle this problem. Transfer learning is a well-studied field, it refers to the techniques used to transfer the acquired knowledge of an already trained ANN to another network. In the case of CNNs, pre-trained convolutional layers are used in the new model by connecting a Feed Forward Network (FFN) on top of pre-trained convolutional layers. The new network will benefit from the already trained convolutions and adapt to the new dataset in a shorter time. Transfer learning is beneficial when the source and target samples are relatively similar. In Transfer learning, domain adaptation is focused on correcting marginal distribution differences or the conditional distribution differences between the source and target domains which have their problems and limitations (Weiss et al., 2016). On the other hand, most model acceleration techniques focus on the reduction of inference latency and memory consumption. Table 2.2 summarizes model acceleration methods.

Table 2.2: Different approaches to model acceleration and compression

Category	Description	Attributes
Parameter Pruning and Quantization	Reduces redundant parameters which are not affecting the performance	Robust to various settings, can achieve good performance, can support both training from scratch and pre-trained models
Low-Rank Factorization	Uses Tensor decomposition to estimate the informative parameters	Standardized pipeline, easily to be implemented, can support both training from scratch and pre-trained models
Transferred/Compact convolutional filters	Uses special structural convolutional filters to save parameters	Algorithms are dependent on applications, usually achieve good performance, only support training from scratch
Knowledge Distillation	Trains a new compact neural network with distilled knowledge of a large model	The performance is sensitive to applications and network structure, only support training from scratch

Note. Retrieved from (Cheng et al., 2017)

Parameter pruning is an acceleration method that removes insignificant connections between neurons and reduces redundancy in convolution kernels. Pruning and quantization should be configured based on the target hardware. For example, a CPU-like architecture with no parallelization may fully exploit the reduction of computations by unstructured pruning to improve speed, but a GPU-like massive parallel architecture would not.

The large number of neurons increases the inference latency and makes models challenging to implement in low-memory environments such as mobile phones and IoT devices. Low-Rank Factorization uses tensor decomposition to replace layers with an approximation of their most important weights, this way both the memory and latency of the model decrease with minimal performance degradation. Transferred/Compact convolutional filters act in a similar way and are techniques based on group theory to find the equivalent of convolutions that reduce the parameter space and save memory and computation.

The Knowledge Distillation method trains a compact and shallow network based on a deep network to mimic its output. Its performance is sensitive to applications and network structure that results to retraining the model from scratch.

Most of the mentioned methods have been more focused on inference latency and less on training time and have added an extra computation intensive step after training to the production pipeline. Table 2.3 demonstrates an overview of state of art model acceleration techniques.

Table 2.3: Overview of model acceleration techniques

Study	**Concept**	**Acceleration Method**					**Remarks**
		Parameter Pruning and Quantization	Low-Rank Factorization	Transferred/Compact convolutional filters	Knowledge Distillation	Other	
(Jung et al., 2019)	Proposed a metric that can measure the influence of a node in a layer of a neural network on a node in subsequent layers	✓	×	×	×	×	Introduced the influential scores and applied them to interpret the outcome of the models
(J. Liu et al., 2021)	Introduces dynamic kernel convolutional neural networks (DK-CNNs), an enhanced type of CNN, by performing line-by-line scanning regular convolution to generate a latent dimension of kernel weights	×	×	✓	×	✓	The proposed DK-CNNs were compared with different network structures with and without a latent dimension on the CIFAR and FashionMNIST datasets. The experimental results show that DK-CNNs can achieve better performance than regular CNNs.

Table 2.3: Continued

Study	Concept (Research motivation)	Acceleration Method					Remarks
		Parameter Pruning and Quantization	Low-Rank Factorization	Transferred/Compact convolutional filters	Knowledge Distillation	Other	
(C. Liu et al., 2019)	Proposes two computation-performance optimization methods to reduce the redundant convolution kernels of a CNN with performance and architecture constraints	✓	×	×	×	×	Optimizations achieve about 50% size reduction but only cause a minor performance drop.
(Li et al., 2020)	The method aims to exploit the discriminant and geometrical structure of data manifold by optimally preserving the local neighborhood information.	×	×	×	×	✓	The experimental results on Extended Yale B, AR, FERET, YTF datasets show that the proposed model outperforms the most recent state-of-the-art models.
(Jung et al., 2018)	Applied pre-defined kernels also known as filters or masks developed for image processing to convolution neural networks. Instead of letting neural networks find their first layer kernels, 41 different general-purpose kernels of blurring, edge detecting, sharpening, discrete cosine transformation, etc. was used	×	×	×	×	✓	The method reaches the accuracy of 90% even with only 500 training samples on the MNIST dataset that is four times faster than the traditional CNN
(Liang et al., 2020)	Aiming at channel compression, a novel convolutional construction named compact convolution is proposed to embrace the progress in spatial convolution, channel grouping, and pooling operation.	×	✓	×	×	×	Unlike traditional methods for dimensional reduction in CNN which introduce considerable learnable weights, the compact convolution can squeeze the channel dimension of feature maps with no extra parameters. Extensive experimental results demonstrated that the proposed method can not only cut down the run time on CPU and GPU but also produce promising performance.
(Xia et al., 2020)	In the proposed method the pooling layer is replaced by two continuous convolutional layers with a 3×3 convolution kernel, which a dropout layer in-between reduce overfitting, and cross-entropy kernel is used as a loss function	×	×	×	×	✓	Through experimental comparison with AlexNet, VG- GNet, and GoogLenet network models on two different data sets, it is concluded that the improved network structure in this paper has obvious advantages in terms of recognition accuracy, convergence speed, and recognition stability.

Table 2.3: Continued

Study	Concept (Research motivation)	Acceleration Method					Remarks
		Parameter Pruning and Quantization	Low-Rank Factorization	Transferred/Compact convolutional filters	Knowledge Distillation	Other	
(W. Wang et al., 2019)	Presents a novel pruning criterion based on channel-level pruning to compress CNN models. The approach utilizes layer-wise feature maps to identify redundant filters.	✓	×	×	×	×	The pruning algorithm proposed in this paper can greatly compress the original VGG-16 into a very small model (only 0.76 MB) without any loss and achieve even an extra gain in performance.
(H. Choi et al., 2020)	Proposed an approach called block change learning that performs local and global knowledge distillation by changing blocks comprised of layers. The method focuses on the knowledge transfer without losing information in a large teacher model, as the approach considers intra-relationships between layers using local knowledge distillation and inter-relationships between corresponding blocks.	×	×	×	✓	×	Tested the BCL approach in object classification and feature extraction. Specifically for feature extraction tasks, BCL showed only about 5% degradation in performance relative to approximately 17% for other methods.
(Lym et al., 2019)	Proposed PruneTrain, a CNN training acceleration mechanism that, unlike prior work, prunes the model during training from scratch with the sparsification process starting during the first training epoch.	✓	×	×	×	×	PruneTrain reduces the computations of ResNet50 for ImageNet by 40%, the memory traffic of memory-bound layers (e.g. batch normalization) by 37%, and the inter-GPU communication cost by 55% compared to the dense baseline training.
(Cho & Lee, 2019)	Proposed a strategy to automatically determine the number of parameters of a network by utilizing group sparsity and knowledge distillation (KD) in the training process and a feedback control mechanism based on the proportional control theory. The feedback control logic determines the amount of emphasis to be put on network sparsity during training and is controlled based on the comparative accuracy losses of the teacher and student models in the training	×	×	×	✓	×	Demonstrates that incorporating knowledge distillation when compressing the network with group sparsity achieves better performance, a student network trained by the proposed strategies achieved better accuracy than when trained by a model with the same sparsity.
(Chen et al., 2018)	Introduces a knowledge distillation framework to improve the performance of smaller and shallower network models.	×	×	×	✓	×	Results show that the proposed training method was effective and increased overall accuracy (3% in AID experiments, 5% in UCMerced experiments, 1% in NWPU-RESISC and EuroSAT experiments)

Table 2.3: Continued

Study	Concept (Research motivation)	Acceleration Method					Remarks
		Parameter Pruning and Quantization	Low-Rank Factorization	Transferred/Compact convolutional filters	Knowledge Distillation	Other	
(Hao-Ting et al., 2019)	A novel Layer Selectivity Learning (LSL) framework is proposed for learning deep models	×	×	×	✓	×	Compared to the baseline method at most, 12.6%, 5.1%, and 6.5% improvement of classification accuracy was achieved using the three datasets.
(Wen et al., 2016)	A Structured Sparsity Learning (SSL) method proposed to regularize the structures (i.e., filters, channels, filter shapes, and layer depth) of DNNs. Which can learn a compact structure from a bigger DNN to reduce computation cost; and obtain a hardware-friendly structured sparsity of DNN to efficiently accelerate the DNN's evaluation	✓	×	×	×	×	The method can enforce the DNN to dynamically learn more compact structures without accuracy loss. The structured compactness of the DNN achieves significant speedups for the DNN evaluation both on CPU and GPU with off-the-shelf libraries.
(Zhou et al., 2016)	This research shows that, by incorporating sparse constraints into the objective function, it is possible to decimate the number of neurons during the training stage.	✓	×	×	×	×	After applying the compression method, compact CNN contains only 30% of the original neurons without any degradation of the top-1 classification accuracy
(Alvarez & Salzmann, 2017)	Introduces a regularizer that encourages the parameter matrix of each layer to have a low rank during training.	✓	×	×	×	×	The experiments in the research have demonstrated that this approach can achieve higher compression rates than state-of-the-art methods at the time of publishing, thus evidencing the benefits of taking compression into account during training.
(Lei et al., 2019)	This paper proposed a dilated CNN model which is built by replacing the convolution kernels of traditional CNN with the dilated convolution kernels, the dilated CNN model has been tested on the MNIST handwritten digits data set	×	×	✓	×	×	Experiments showed that the dilated CNN model is less time-consuming and has higher training accuracy on the MNIST data set compared with the traditional CNN model.
(Mathieu et al., 2014)	Computing convolutions as pointwise products in the Fourier domain while reusing the same transformed feature map many times	×	×	✓	×	×	Presented a fast algorithm that outperforms known state-of-the-art implementations in terms of speed

Most research in Table 2.3 were focused on model compression to reduce memory and model acceleration to decrease inference latency. Despite being fruitful for the inference phase, they didn't emphasize reducing training time and some even add complexity to the training.

In the case of parameter pruning Wen et al. (2016), Zhou et al. (2016) and Alvarez & Salzmann (2017b) deployed pruning during the training but didn`t apply it on the source network, thus their implementation did not decrease the training time. Lym et al. (2019) work while being effective in reducing the training time is targeting multi-GPU clusters. The reduction of training time was the result of the combination of reducing computation, reducing off-chip memory access, and reducing inter-accelerator communication. The hardware setup for this research is a single GPU solution and the models are designed to fit into the graphical memory. Therefore, this work cannot benefit from the reduction of off-chip memory access and inter-accelerator commination mentioned in (Lym et al., 2019). Lym et al. (2019)'s method is applicable to single GPU setups but it is not predictable how effective it is while it cannot leverage reduction of off-chip memory access and inter-accelerator commination in a single GPU setup and testing the efficiency of the method on a single GPU setup can be a topic for future research. Thus, this work does not target Lym et al. (2019)'s method as its baseline.

Additionally, Jung et al. (2018) proposed GFNNs for the reduction of training time that is architecture and dataset independent and can be applied to any supervised use case of CNNs. CNNs are universal functions (Wiatowski & Bolcskei, 2018) that during the training, will converge to any desired function. In the case of image classification, the function maps the input tensors to the labels, and the proposed method tries to help CNNs to converge faster by adding fixed image processing filters to the first layer of CNNs. The proposed method is applicable to all hardware as the solution works at the architecture level and directly affects the time and needed number of necessary epochs in training. Also, the method is not limited to images and image processing filters, these filters can be replaced by any domain specific kernels when dealing with a spatial sensory data like voice, MRI result, etc. The original research only tested the technique on the MNIST dataset, which consists of binary 28x28 images with only 10 classes, which is not sufficient for verification of efficiency of the methods due to the lack of complexity of features in the input data. On the other hand, the side effects of the method, like its effects on overfitting should be tested.

This research aims to address the mentioned shortcomings in the original method, Figure 2.3 demonstrates the gap in the knowledge that this research is focused on. The architecture of filter layers is altered to support multi-channel sensory data and its performance is assessed through an experiment. As described in the next chapter, six GFNNs architectures are designed along with their equivalent baseline CNNs. The experiment targets three different datasets with color images and in addition to performance metrics, its effect on overfitting is also analyzed.

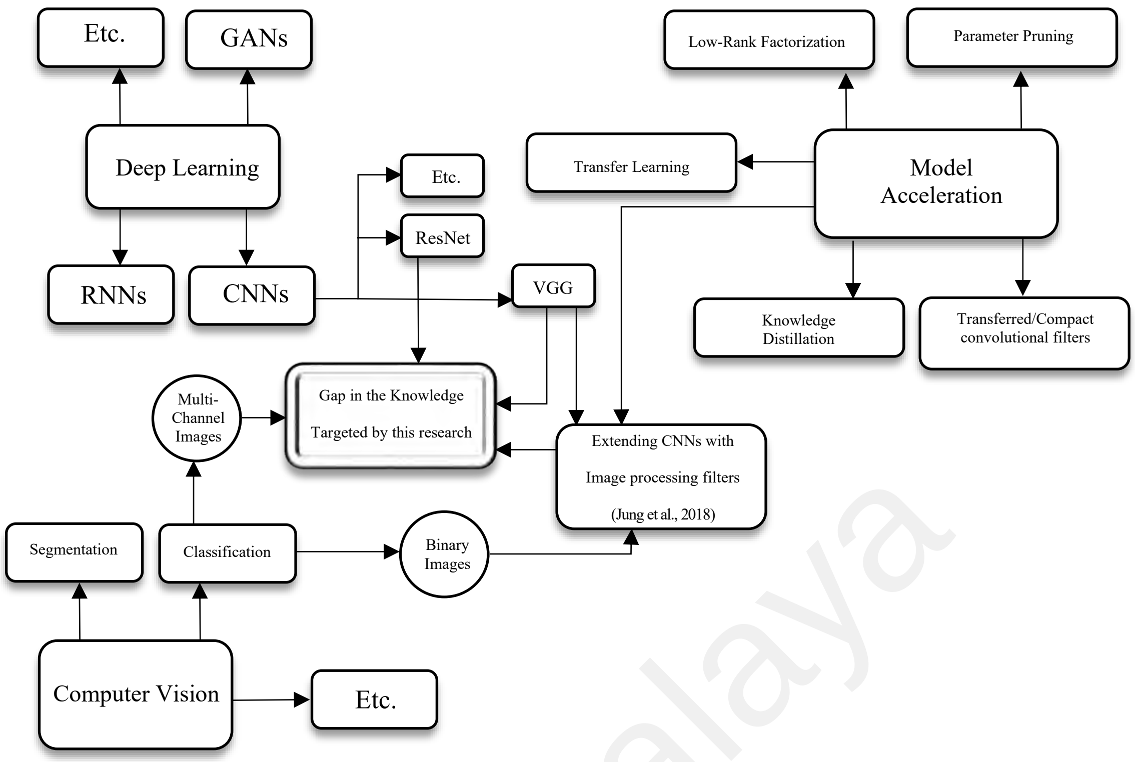

Figure 2.3: Knowledge Map

2.4. Summary

CNNs have a crucial role in deep learning, much research in the field is focused on them, and they have a good performance on many difficult and ill-posed problems in machine learning. Image classification is one of the useful use cases of CNNs. Used in lots of multidisciplinary research, they have been used in cancer detection, autonomous deforestation detection, autonomous driving, etc. The state of art CNNs aim to solve more difficult problems and most of the research has been focused on increasing the performance of the network leading to their increased processing and memory consumption. State of the art CNNs consist of millions if not billions of learnable parameters that make the training challenging and increases the inference latency. Model acceleration techniques like parameter pruning and quantization, low-rank factorization, transferred/compact convolutional filters, and knowledge distillation aim to reduce the needed processing and memory of the deployed model. After or while training, these techniques alter the architecture of the final model and make them suitable for deployment. Some model acceleration methods are hardware dependent and are effective only when using a specific type of hardware or accelerator and most of them add complexity to the training of the model and extend the training time. Jung et al. (2018) claims by merging the image processing filters inside the CNNs architecture, it is possible to help the network to converge faster, and the CNNs are capable of using the extracted feature by image processing filters that results in faster training of the CNN. But the method was only tested on MNIST handwritten digits dataset that lacks the feature complexity to prove the efficiency of the method, also, the method is only applicable to single-channel images. This research aims to address the shortcoming of the original method by modify it to apply to multi-channel images. Also, the efficiency of the method is examined empirically through an experiment that applies the method on different CNNs.

CHAPTER 3: RESEARCH DESIGN

3.1. Introduction

The main objective of this research is to first, alter the first layer of GFNNs to support multi-channel images and then verify their performance on three datasets through an experiment. Table 3.1 demonstrates the characteristics of the target datasets.

Table 3.1: Datasets and their characteristics

Dataset	Sample size	No of samples	No of classes	Reference
ImageNet_64x64	64x64x3	1,331,167	1,000	(Chrabaszcz et al., 2017)
Cifar100	32x32x3	60,000	100	(Krizhevsky & Hinton, 2009)
Cifar10	32x32x3	60,000	10	(Krizhevsky & Hinton, 2009)

The MNIST dataset consists of binary 28x28 images. In contrast, Cifar10 and Cifar100 contain color images with a resolution of 32x32. And ImageNet_64x64 consists of 64x64 color images. Having multiple channels of non-binary sensory data alone leads to much more complex features compared to a single-channel binary data. Additionally, the number of classes is much higher in Cifar100 (100 classes) and ImageNet_64x64 (1000 classes). Therefore, the neural network is forced to extract more complex features to differ between high number of classes.

For each dataset, two pairs of CNNs are designed. Each pair of networks contains a baseline CNN and a GFNN. The baseline architectures are custom implementations of ResNet and VGG. Each dataset has different characteristics, thus the architecture of CNNs (e.g., dimensionality of input layer, size of hidden layers, size of flatten fully connected layer and the size of output layer) for each dataset are different. Overall, 12 CNN architectures are designed and benchmarked.

All datasets are balanced; thus, accuracy is reported as the performance metric. Additionally, loss function and training time are collected per epoch. Table 3.2 shows the collected quantitative variables through the experiment.

The GFNNs have not been tested on datasets with color images and complex features, thus, to overcome this shortcoming, in this research three datasets with different attributes are chosen. ImageNet_64x64 (Chrabaszcz et al., 2017) is the biggest among them with 1,331,167 samples and 1,000 classes. It is the one of hardest datasets to achieve high accuracy with because data samples are low resolution (64x64) while having a large number of classes. ImageNet_64x64 is based on ImageNet dataset and has the same labels, but the samples are resized and have a lower number of samples. Cifar100 (Krizhevsky & Hinton, 2009) has samples with lower resolution (32x32), it has 100 classes and 60,000 samples. Cifar10 (Krizhevsky & Hinton, 2009) have similar attributes, except the number of classes is limited to 10, and among these three is the easiest to achieve high accuracy with.

Figure 3.1 demonstrates the flow of the experiment, first baseline versions of CNNs are designed based on their target dataset attributes, then their equivalent GFNNs that contains image processing filters are implemented. Each model is trained for 5 iterations on its target dataset.

Table 3.2 shows the list of collected quantitative variables, the average value of each variable is used for visualization and analysis. This is to average out the random effects of cache warm up, operating system scheduler overhead, TensorFlow runtime optimization on graph operations, etc. to make the comparison of baseline and GFNNs more accurate. The accuracy of the baseline and GFNNs are demonstrated and compared using graphs and tables in Chapter 4, and also the overfitting of the models is monitored by comparing the loss and accuracy of the models on training and validation samples.

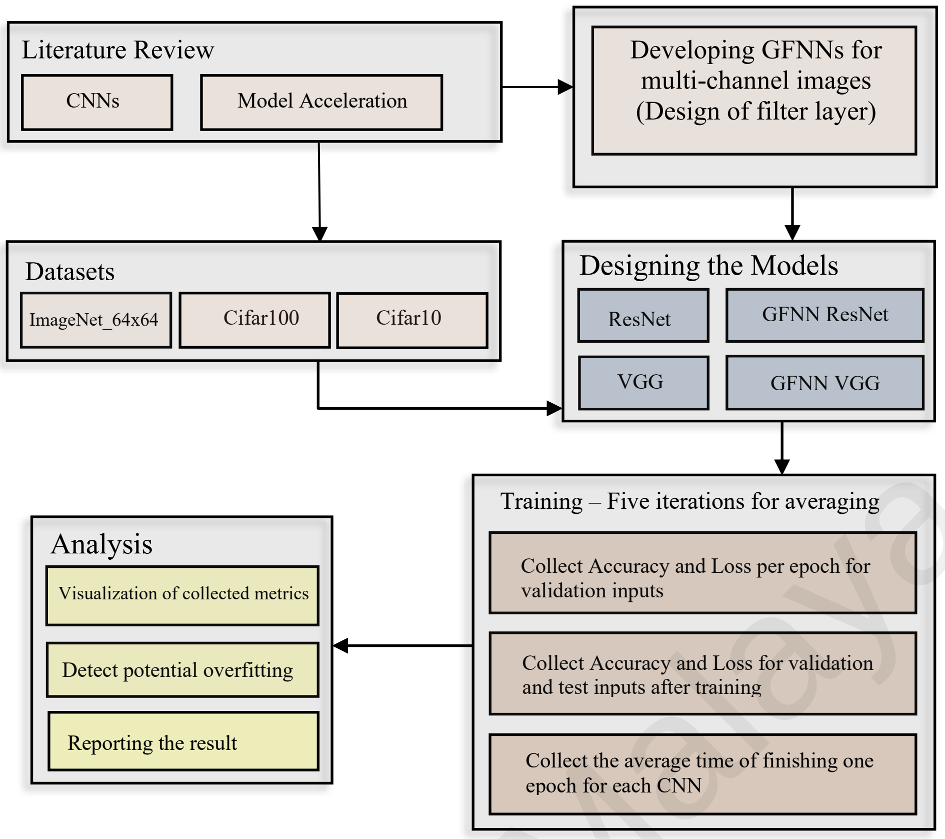
Literature Review
CNNs
Model Acceleration
Developing GFNNs for multi-channel images (Design of filter layer)
Datasets
ImageNet_64x64
Cifar100
Cifar10
Designing the Models
ResNet
GFNN ResNet
VGG
GFNN VGG
Training – Five iterations for averaging
Collect Accuracy and Loss per epoch for validation inputs
Collect Accuracy and Loss for validation and test inputs after training
Collect the average time of finishing one epoch for each CNN
Analysis
Visualization of collected metrics
Detect potential overfitting
Reporting the result

Figure 3.1: Research Design

All neural networks in this research are designed and implemented from scratch using TensorFlow's functional API. Because of limited time and processing power, only two architectures of CNNs are selected as baselines for the experiment. The first architecture that is called traditional CNN through this research is a custom implementation of VGG architecture with the same design principle. The second architecture is based on ResNet, it contains residual blocks and a feed-forward classifier similar to ResNet, but the exact configuration varies for each dataset. ResNet and VGG were selected as the base for the design of CNNs in this research because they represent two different types of CNNs architecture. In the case of VGG there is no skip connection between the layers. After the training, the first layer of VGG model extracts features that look similar to what edge detection filters produce, and it would be possible for the network to use these features if provided by image processing filters. ResNet and other architectures with skip connections use more diversified features due to the concatenation of information from previous blocks and the network might use different strategies to adapt itself to the extracted feature. The first layer of the network called the filter layer in this research, is the only difference between the baseline implementation of the neural networks and their equivalent GFNN. The image processing filters or any other potential mathematical formula that can be implemented using convolutional operations are suitable to replace some of the kernels in CNNs architecture. In this research the image processing filters merged into the CNNs by replacing the first layer's kernels with 16 filters. There exist 3 extra design choices for GFNNs, first is the choice of baseline architecture, this method is applicable on any neural network with convolutional layers, and in this research, a custom implementation of VGG and ResNet are chosen. Hardware and time limitation was the reason for this decision and testing the method on other architectures is a potential topic for the future research. The type and number of filters is the second choice and location of filters in the network is the third. The layers can partially consist of convolutional kernels and image processing filters. Also, different types of filters might have a different effect when used in first versus hidden layers. In this research first convolutional layer of baseline networks is replace by image processing filters, due to the hardware limitations, it was not possible to test multiple combinations of filter location and type, thus a specific list of filters with similar output to the first layer of normal CNNs after training were chosen. The next chapter discusses the filter layer, and the image processing filters.

3.2. Filter Layer

The first layer of GFNNs termed "Filter Layer" and instead of convolutions with learnable weights, consists of image processing filters. These filters provide the input for the second layer of normal convolutional layers. The original architecture was applied only on a binary dataset, so it is necessary to expand the concept to apply it to the multi-channel datasets targeted in this research. Because filters do not have learnable parameters, if they combine the results of channels, the network will lose its ability to extract features in different color channels. As shown in Figure 3.2, in contrast to convolutions, each filter should deliver output for each channel separately. The filter layer's output is a rank 3 tensor, with the dimensionality of 32x32x3 for Cifar datasets, and 64x64x3 for ImageNet_64x64.

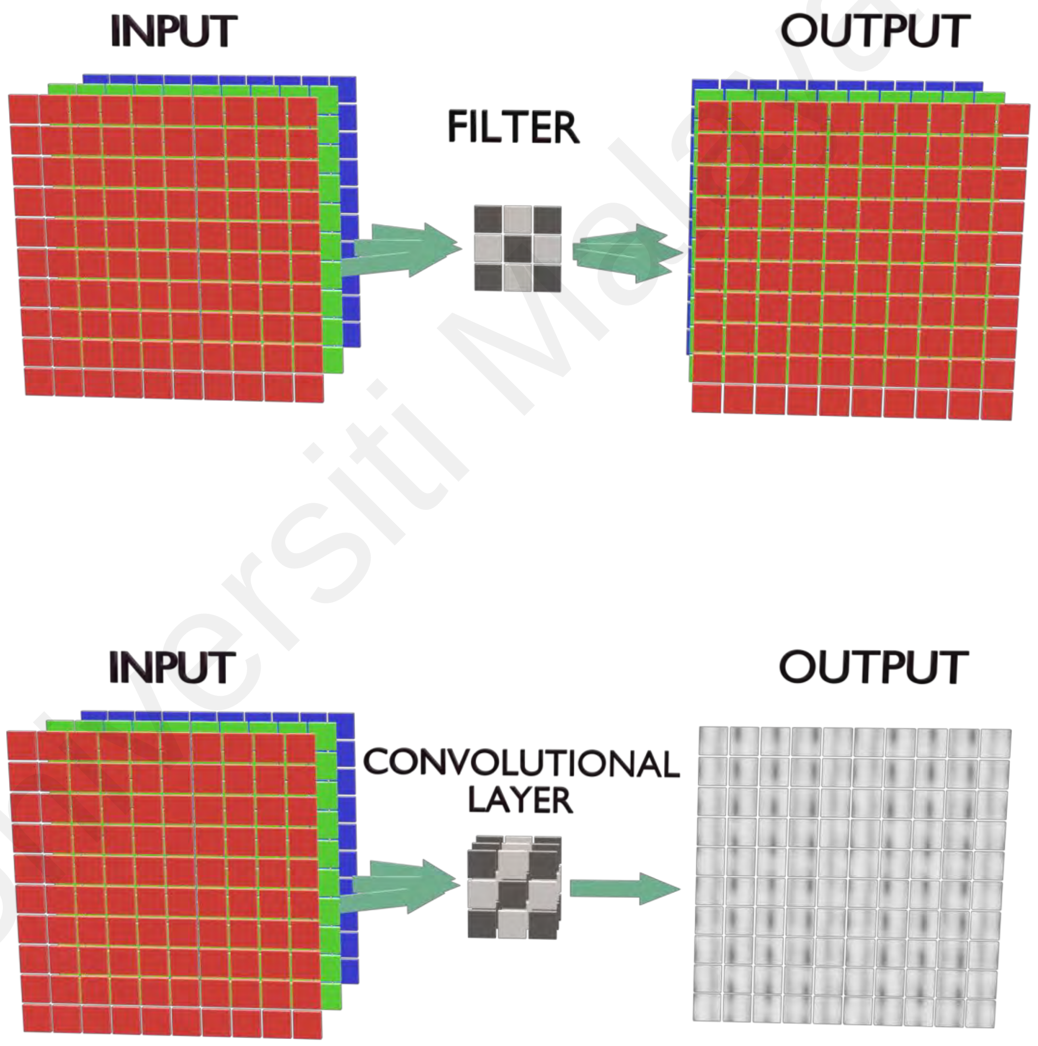

Figure 3.2: Difference Between Convolutional and Filter Layer

To keep the baseline and GFNN networks computationally comparable, the dimensionality of the Filter Layer's output should be similar to the first convolutional layer of the baseline model. Because each filter produces three channels as opposed to one channel in the case of convolutional layers, to keep the number of needed calculations relatively similar in baseline and GFNNs, the number of convolutions in the first layer of baseline networks should be three times of the number of filters in the GFNN version. The number of needed calculations for the filter layer will differ because filters don't have any learnable parameters, thus they won't cost any calculation in backpropagation algorithms. Also, each convolutional kernel will perform three times more calculations per output channel. Because the filters are applied to each channel separately, but the convolutional kernel applies to all channels. But by keeping the dimensionality output of the filter layer the same as the first convolutional layer, the networks remain computationally comparable.

3.3. Image Processing Filters

Compass gradient filters outnumber other types of filters because they have similar characteristics to the first layer of convolutions in a trained CNNs. The list of implemented filters is as below:

- two second-order filters
- one DCT filter
- one sharpening filter
- one blurring filer
- one embossing filter
- ten compass gradient filters

Totally 16 filters are applied to the input sample producing a rank 3 tensor with the dimensionality of 64x64x48 as output (in case of ImageNet_64x64), which will be equivalent of using 48 convolutions for baseline networks. Figure 3.3 demonstrates a schematic view of used filters.

In comparison with the original work by Jung et al. (2018) , the number of filters reduced from 41 to 16. This was a necessary design choice, because every filter in the new architecture produces 3 channels of data (one for each one of the RGB channels). Thus, including all 41 filters would lead to 123 channels of data as input of second layer of the network that will increase the memory consumption and also results in slower training and inference latency. Figures 3.11 and 3.12 demonstrate the schematics of the designed networks in the case of ImageNet_64x64, especially in the case of GFNN ResNet having a high dimensional tensor input propagates through the architecture of the network because in each residual block the dimensionality of the layer is the same. Even in the case of GFNN VGG, memory consumption almost tripled during the training when 123 layers has been used. The neural network framework (TensorFlow) needs to store the intermediatory result of every operation to later calculate the gradient during the backpropagation. Thus, the memory consumption of the model has a linear relation with the number of filters. An Nvidia 2060 GPU with 6 GB of graphical memory, Ryzen 4800H CPU and 16GB of DDR4 system memory were the hardware limits for this research and either the batch size or the number of filters should be reduced. During the implementation of the networks, multiple configurations of batch size and the number of filters were tested, and the result was better for a higher batch size with a reduced number of filters. The training time increased and the accuracy did not change with a higher number of filters. Therefore, it is suggestible that decreasing the number of filters in the case of multi-channel implementation performs better than decreasing the batch size.

To be faithful to the design of the GFNNs, the chosen filters are the same type of the original work and only the number of filters has decreased. Jung et al. (2018) didn't provide details about the implemented kernels, but the kernel types and number of kernels were mentioned in the original research. Because applying the method on multi-channel data triples the memory consumption of the networks, either number of filters or batch size should be reduced. Therefore, the number of sharpening filters reduced from 3 to 1, the number of blurring filters reduced from 2 to 1, and the number of compass gradient filters reduced from 32 to 10. The number of filters is the same for the rest.

The major decrease in the number of filters was for compass gradient filters which amplify information about the edges in one direction of the 2D surface. Thus, compass gradient filters were picked in a way to cover 8 major directions in the 2D space. This way the memory consumption is reduced with minimal loss of information (because of the similarity of the output of compass gradient filters). The optimal number of filters suitable for the memory limitation of this research was chosen during the network design, while a different combination of batch size and filters has been tested.

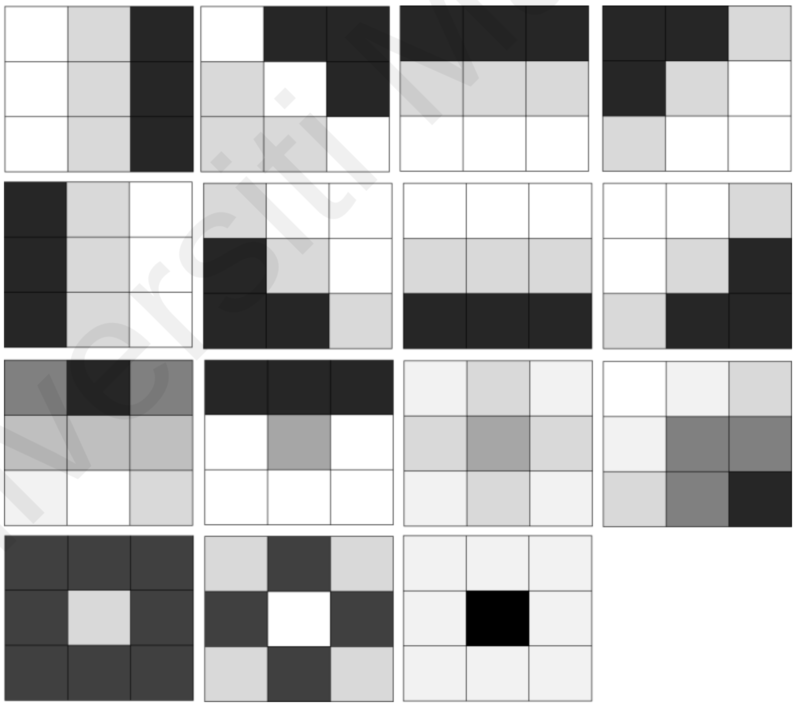

Figure 3.3: A Schematic View of The Filters (Expect DCT)

The filter layer implemented through a custom FilterLayer class in TensorFlow, below steps define the behavior of the operation:

1. The filter layer separates the input`s channels

2. Applies each of the 16 filters to each of the channels separately
3. Stores the result of each filter in the memory as a matrix
4. Stacks all the matrixes on top of each other to create a rank 3 tensor

The produced rank 3 tensor is a 48-channel data that is the input for next convolutional layer of the network.

Figure 3.4 shows a sample input from ImageNet_64x64 along with its RGB channels. For each set of the filters explained in the proceeding chapters, corresponding figures demonstrate the output of filter layer for each channel of the sample.

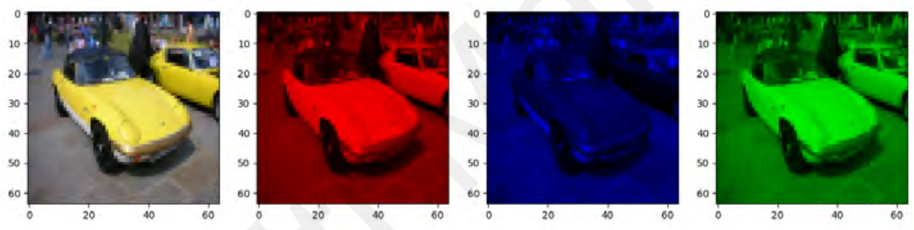

Figure 3.4: A Sample from Imagenet_64x64 Dataset

3.3.1 Second Order Filters

In addition to edge detection filters that approximate the first order of derivatives of pixel values in an image, it is possible to create the filters based on second order derivatives. While using the first order derivatives, it is possible to detect horizontal or vertical edges in the image and then combine their result, but the second order derivates can extract both at once. There are also some disadvantages to the use of second order derivatives. First, the second order derivative operators exaggerated the noise twice as first order operators, also, they don't provide any directional information about the edges.

That is why first order operators are also used in the filter layer in addition to second order filters.

Laplacian operator is a famous second order operator that is used in the filter layer, both the convolutional kernel and image processing filters in CNNs are 3x3. In the case of second order filters, the 3x3 grid of numbers should be a discrete approximation of the Laplacian operator. Two filters based on the Laplacian operator are integrated into the filter layer.

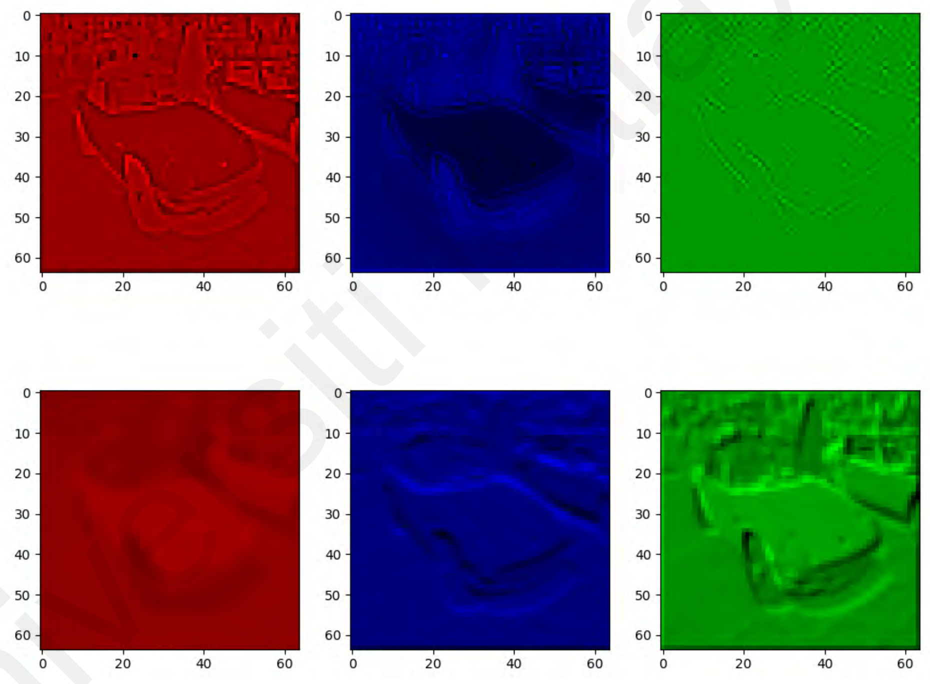

Figure 3.5: Result of Second Order Filters Applied on The Sample

3.3.2 DCT Filter

Discrete cosine transform (DCT) transfers the image from the spatial domain into the frequency domain. It represents the image as the summation of sinusoids with different magnitudes and frequencies. For a typical image, most of the visually important information is concentrated into a few of the sinusoid’s coefficients and for this reason, DCT is used in image compression algorithms. Only one DCT filter is used in the filter layer of GFNN networks, monitoring the ability of CNNs in using the information provided in the frequency domain can be a topic for future research. In contrast to other filters, the DCT filter is defined based on the dimensions of the input and it is not a kernel that goes through the image, instead, it is a mask with the same dimensions of the input image that will be applied on the image. The DCT filter needs a much lower computation compared to convolutional layers, but this difference is neglectable when compared to the whole network's needed computation, thus the GFNN version would be still computationally comparable with baseline. The DCT filter is applied to each channel separately and produces three channels and the same as other filters is replaced with three convolutional kernels in baseline networks.

The DCT filter result are not demonstrated because they are not visually relatable to the source image for human eye, as they are in frequency domain and its output channels is passed to the next layer and not merged together to form a human understandable result.

3.3.3 Sharpening and Blurring Filters

Both sharpening and blurring filers work in the spatial domain and compare the value of pixels with their neighbors. Sharpening filters enhance these differences; thus, it is a process of differentiation, but blurring filters average out the differences and is a process of integration. Two 3x3 filters with predefined values act as sharpening and blurring filters in the filter layer, each producing three channels of output with the same dimension of the input image. The sharpening filters are very sensitive to noise in opposed to blurring filters that reduces the noise in the image, thus they provide different types of features for the next convolutional layers.

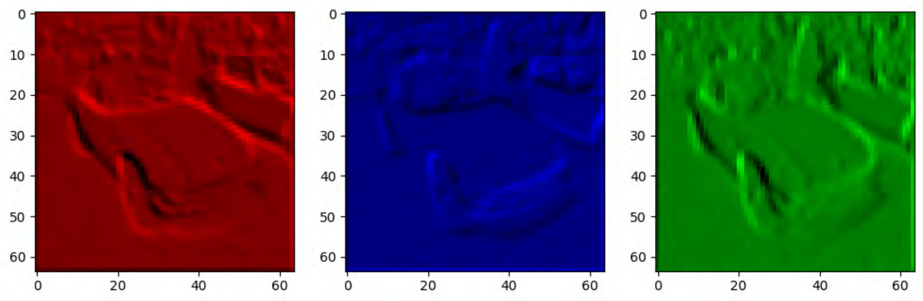
Figure 3.6: Result of Sharpening Filter Applied on The Sample

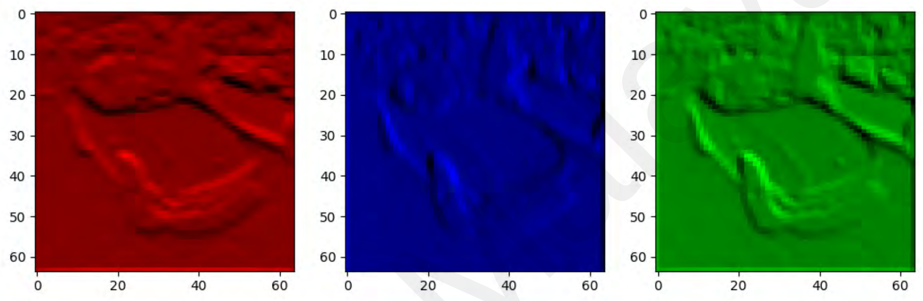
Figure 3.7: Result of Blurring Filter Applied on The Sample

3.3.4 Embossing Filter

The output of an embossing filter is an embossed image whose pixels are replaced by a shadow or a highlight. They are also termed directional difference filters and can enhance the edge in the direction of the filter. In another word they can remove the image's features except for the edges in the direction of the filter, thus the proceeding convolutions after the filter layer will receive simplified features that contain only information about the edges in the input image. Traditionally embossing filters when applied on color images followed the same rule as to when applied to gray scale images, the filter is applied to each of the color channels and the result is combined based on a mathematical formula, but the GFNNs are free to use each of these channels separately, thus the difference between the implementation of embossing filters in filter layer with their traditional definition is that their result is not combined and each output is passed to the next layer of the network in a separate channel. So, each embossing filter produces three channels in the output and should be replaced with three convolutional kernels of baseline networks.

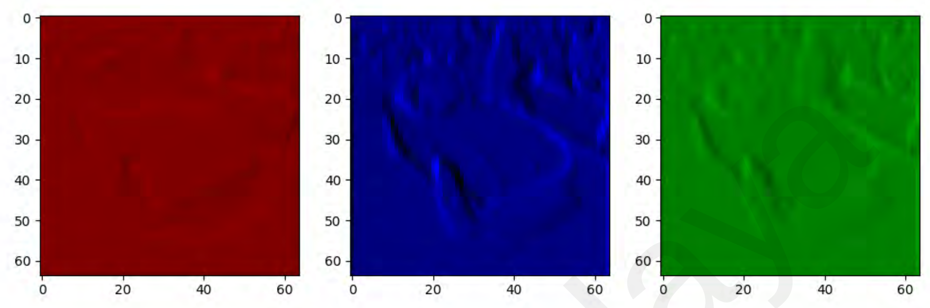

Figure 3.8: Result of Embossing Filter Applied on The Sample

3.3.5 Compass Gradient Filters

Compass gradient filters are first order filters that measure the slope of values of pixels in a specific direction. They are less sensitive to noise and provide information about the direction of the detected edges. Thus, the GFNN network can prioritize a specific direction by tuning its weight and biases of next convolutional layer, to give the GFNN network flexibility of emphasizing on a specific direction. 10 compass gradient filters are implemented in the filter layer, 8 of them refer to directions of north, north-west, west, south-west, south, south-east, east, and north-east. The other two are high pass filters that emphasize sudden changes in pixel values. The fact that GFNN networks are capable of emphasizing edges in a specific direction doesn’t mean they are not rotation invariant, as this emphasis only happens when the input data set have distinguishable features in form of edges in a specific direction, for example, this approach might be beneficial in OCR software where the general shape of the alphabet is predefined and the input is preprocessed to be in a proper orientation before being fed to the model. But the network will still be able to give equal importance to all these filters and act as an orientation invariant classifier. Gradient compass filters apply to image channels separately, thus each filter produces three channels and should be replaced by three convolutions in the baseline network. The gradient compass filters outnumber other types of image processing filters because opposed to second order filters, they can provide information about the direction of the features while being less sensitive to noise. Also, they can ensure that the network has access to all edges in every direction which is not possible with other types of filters. The high contrast filters were implemented only in direction of north and east, as increasing the number of filters would lead to a drastic increase of memory and processing in training. Also, opposed to normal gradient compass filters, high contrast filters are not dependent to the direction. And these two filters provide the GFNNs enough information about the sudden changes in pixel values.

One major difference between these filters in the filter layer to their traditional implementation is the lack of a combination for channels in the filter layer. Most of the mentioned filters have a combination formula for concatenation of extracted information while applied to color images. In the filter layer, the combination of extracted features from the different color channels is handled via the rest of the neural network's architecture, thus the raw output of the filters is passed to the convolutional layers that will decide how to use them during the training.

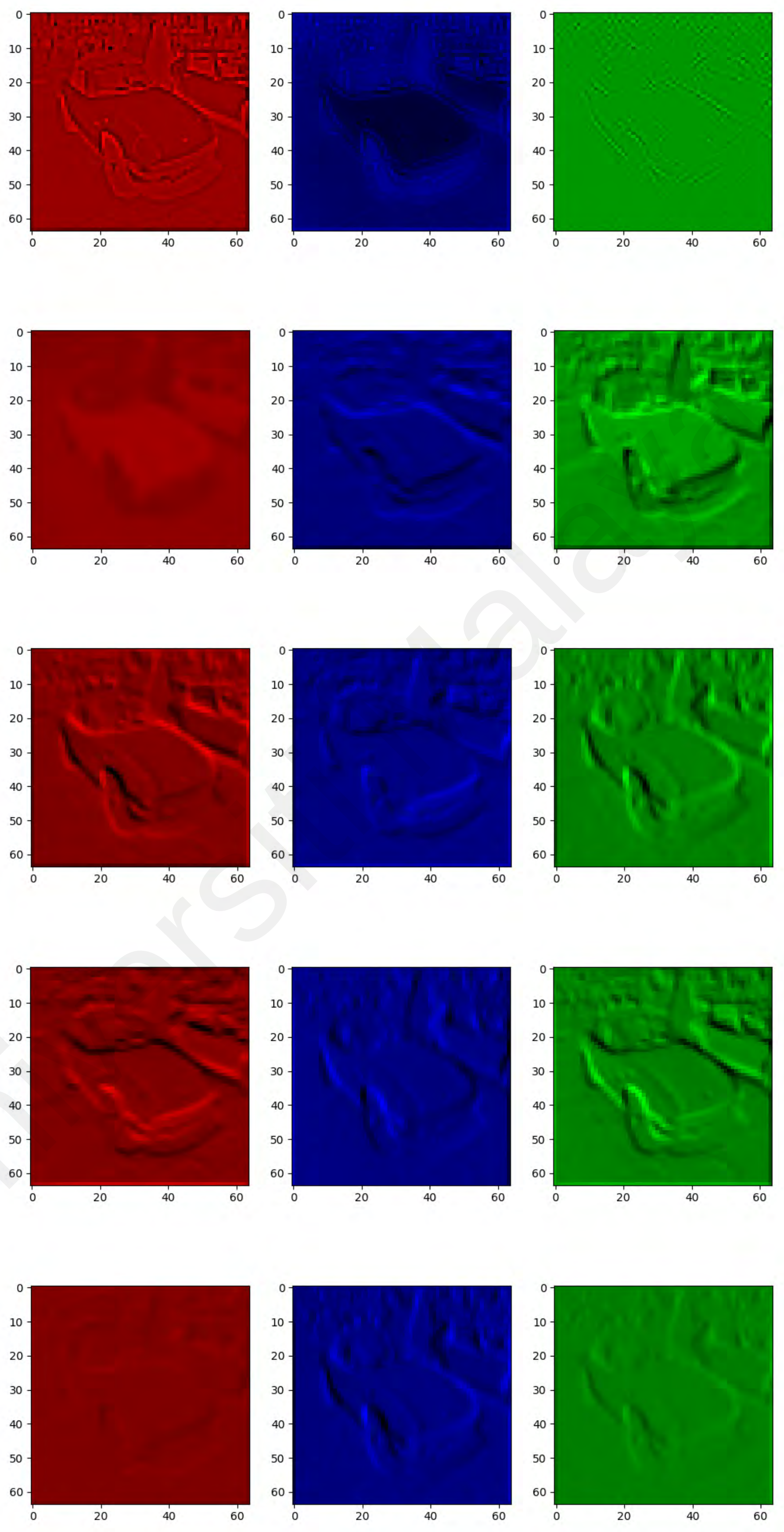
Figure 3.9: Result of Compass Gradient Filters (1-5) Applied on The Sample

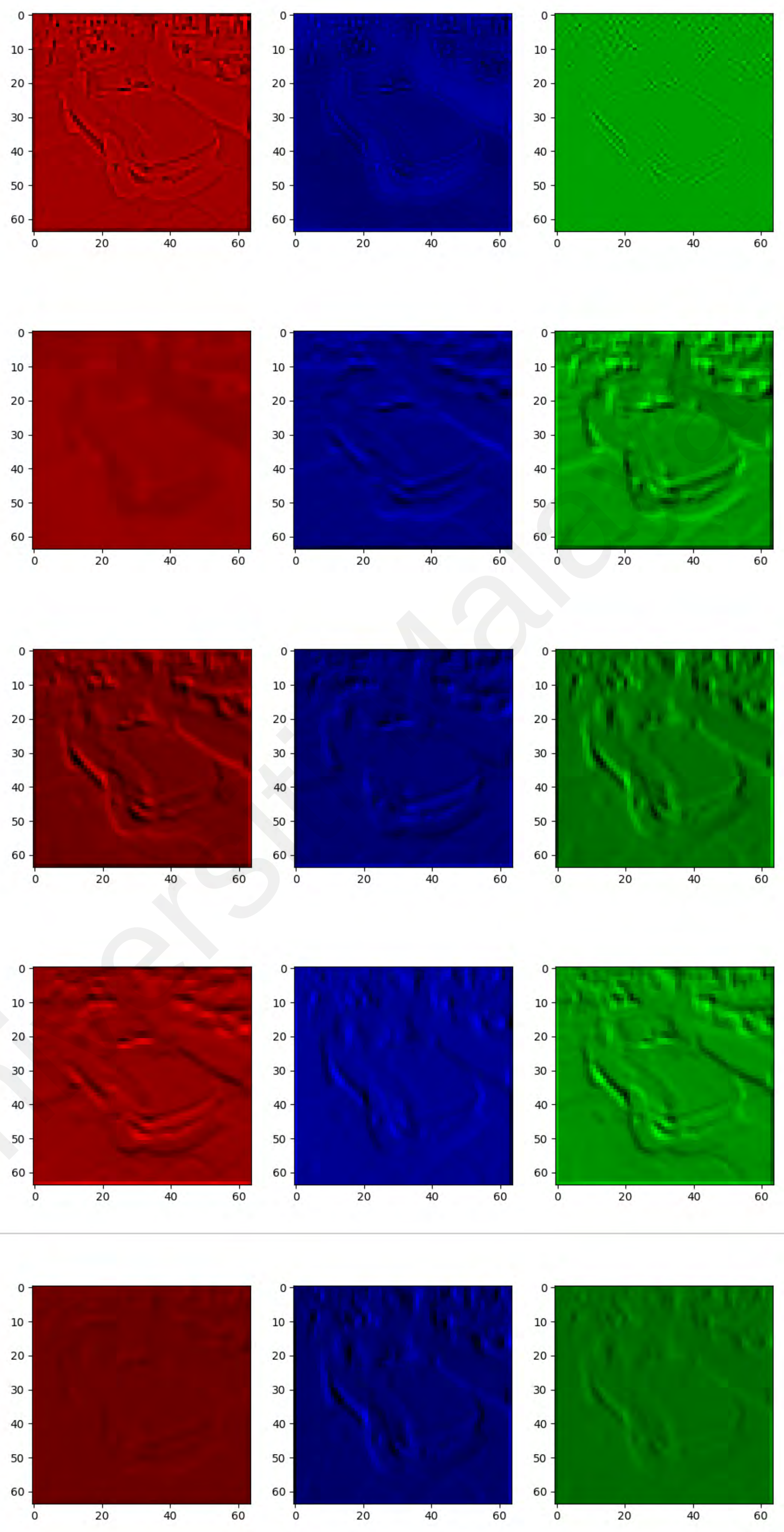
Figure 3.10: Result of Compass Gradient Filters (6-10) Applied on The Sample

3.4. Quantitative Analysis

The objective of GFNNs is to reduce the training time with minimal degradation of accuracy compared to the baseline CNNs. Thus, the network should reach the desired accuracy with a smaller number of epochs. Also, a shorter training time per epoch is expected due to the lower number of learnable parameters. Table 3.2 demonstrates the qualitative variables that are collected.

The answer to the below questions is retrieved from the analysis of collected variables.

- Do GFNNs reach a higher accuracy within a certain number of epochs?
- Do GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks?
- Are GFNNs more vulnerable to overfitting?

Table 3.2: Quantitative Variables

Name	Description	Frequency
ep_tr_ti	Training time of one epoch	Per epoch
ep_tr_ac_tr	Accuracy of network on training data within the epoch	Per epoch
ep_tr_ac_vl	Accuracy of network on validation data within the epoch	Per epoch
ep_tr_ls_ti	Value of loss function on training data	Per epoch
ep_tr_ls_vl	Value of loss function on validation data	Per epoch
tot_tr_ti	Total training time	Per training
tot_tr_ac_tr	Accuracy of network on training data after training	Per training
tot_tr_ac_vl	Accuracy of network on validation data after training	Per training

Training of each of <u>twelve</u> networks reiterates <u>five</u> times and the average is considered as the result to minimize the random impact of operating system, bus interface overhead, etc.

3.5. Network`s Architecture

Different characteristics of datasets result in a different architecture of CNNs, so the number of learnable parameters, is different for each of the designed networks. The resolution of input samples dictates the dimensionality of filter layer, and consequently affect the dimensionality of the convolutional layers. Also, number of classes affects number of learnable parameters in fully connected section of the networks. Also, because of hardware limitations, the neural networks in this research do not aim for the highest possible accuracy. Instead, the research focuses on applicability of the method on multi-channel images and monitoring the effect of adding the filter layer on the accuracy and inference latency.

For each dataset, a baseline version of either the traditional CNN or ResNet is compared with its GFNN version. The traditional CNN architecture is a simplified version of VGG (Simonyan & Zisserman, 2015) and the ResNet (He et al., 2016) consist of six residual blocks.

Table 3.3: Designed Networks Information

Dataset	Architecture	Learnable	None Learnable	Layers	Learning Rate	Epochs	Activation Function	Dropout Rate
Cifar10	Baseline VGG	692,383	672	29	0.001	40	ReLu	0.5
Cifar10	GFNN VGG	690,063	1,344	29	0.001	40	ReLu	0.5
Cifar10	Baseline ResNet	1,574,959	2,454	51	0.001	40	ReLu	N/A
Cifar10	GFNN ResNet	1,573,615	2,784	51	0.001	40	ReLu	N/A
Cifar100	Baseline VGG	1,125,426	672	29	0.002	60	ReLu	0.5
Cifar100	GFNN VGG	1,122,178	2,454	29	0.002	60	ReLu	0.5
Cifar100	Baseline ResNet	1,693,714	1,454	51	0.002	60	ReLu	N/A
Cifar100	GFNN ResNet	1,692,370	2,784	51	0.002	60	ReLu	N/A
ImageNet_64	Baseline VGG	20,580,628	672	29	0.001	30	ReLu	0.5
ImageNet_64	GFNN VGG	20,579,218	1,344	29	0.001	30	ReLu	0.5
ImageNet_64	Baseline ResNet	7,673,764	2,454	51	0.001	30	ReLu	N/A
ImageNet_64	GFNN ResNet	7,672,420	2,784	51	0.001	30	ReLu	N/A

The number of layers is the same for baseline and GFNNs, but the dimensionality of tensors that are passed between these layers is different. Table 3.3 includes the number of trainable parameters for each of the designed networks. The networks were intentionally extensively trained to determine whether the GFNNs have a different behavior regarding the overfitting or not. The next chapter discusses the result of the experiment and visualizes the comparison of baseline and GFNN versions for each dataset.

3.4.1 Fusion of Filter and Convolutional Layers

If the filter layer acts like convolutional layers and combines the information of all three channels into a single matrix, the color information would be lost as the filter layer does not poses any learnable parameter, and it produces the same output for a specific input sample. To mitigate this problem, as mentioned in Chapter 3.2, each filter of filter layer is applied to each channel of data separately. Also, each of the filters produce 3 separate matrixes as output corresponding to each channel. Therefore, practical consideration is needed to prepare the output for its next convolutional layer.

The convolution operation in neural networks receive a rank 3 tensor as input, the dimensionality of this input tensor, along with kernel size and stride determine the number of parameters in the convolutional kernel. In case of filter layers the dimensionality of input tensor is (64, 64, 48) in case of ImageNet_64x64 and (32, 32, 48) in case of Cifar datasets.

The fusion of filter layer and convolutional layer implemented using TensorFlow's functional API. The operations in TensorFlow follow an object-oriented pattern design. Therefore, the filter layer is implemented by inheriting FilterLayer class. It separates each channel of color, then applies each of the 16 filters on them and stores the results in the memory. Then stacks the 48 produced matrixes and provides the needed rank 3 tensor to the next convolutional layer.

3.4.2 Batch Normalization and Dropout

In addition to filter layer and convolutional layers, Dropout (Srivastava et al., 2014) and Batch-Normalization (Ioffe & Szegedy, 2015) are used in the architecture of networks. Thus, before demonstrating a schematic for the architecture of the networks, this chapter quickly describes them.

When using Dropout during the training, a random set of parameters get ignored. This forces the layer before Dropout to act like a layer with different weights. In effect, the updates to the network would happen with a different view of configured layers. Thus, it is a regularization method that helps prevent overfitting.

Batch-Normalization is a technique for training of deep neural networks. It standardizes the input for the next layer in each mini-batch. And dynamically reduces the number of needed epochs for training of the network. The problem that Batch-Normalization aims to solves is called “internal covariate shift”. It happens when the distribution of input of layer changes with each update of mini-batch, and this change causes the learning algorithm to chase a moving target. Batch-Normalization prevents this by standardizing the input of the layer after the update of mini-batch.

3.4.3 Network`s Schematics

In this section, the schematics of GFNN VGG and GFNN ResNet designed for ImageNet_64x64 are shown. Also, Adam used as backpropagation algorithm and sparse categorical cross entropy used as the loss function.

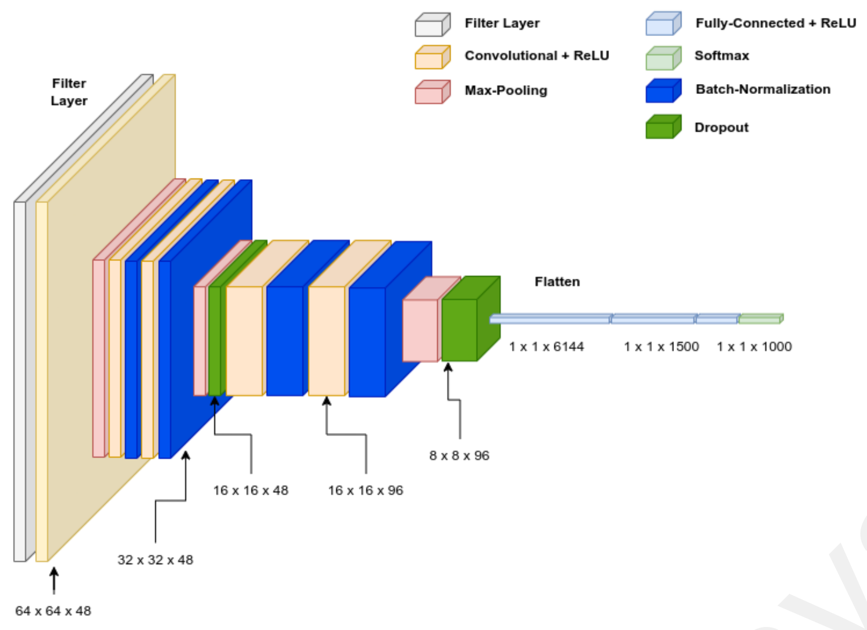

Figure 3.11: Schematic of GFNN VGG for ImageNet_64x64

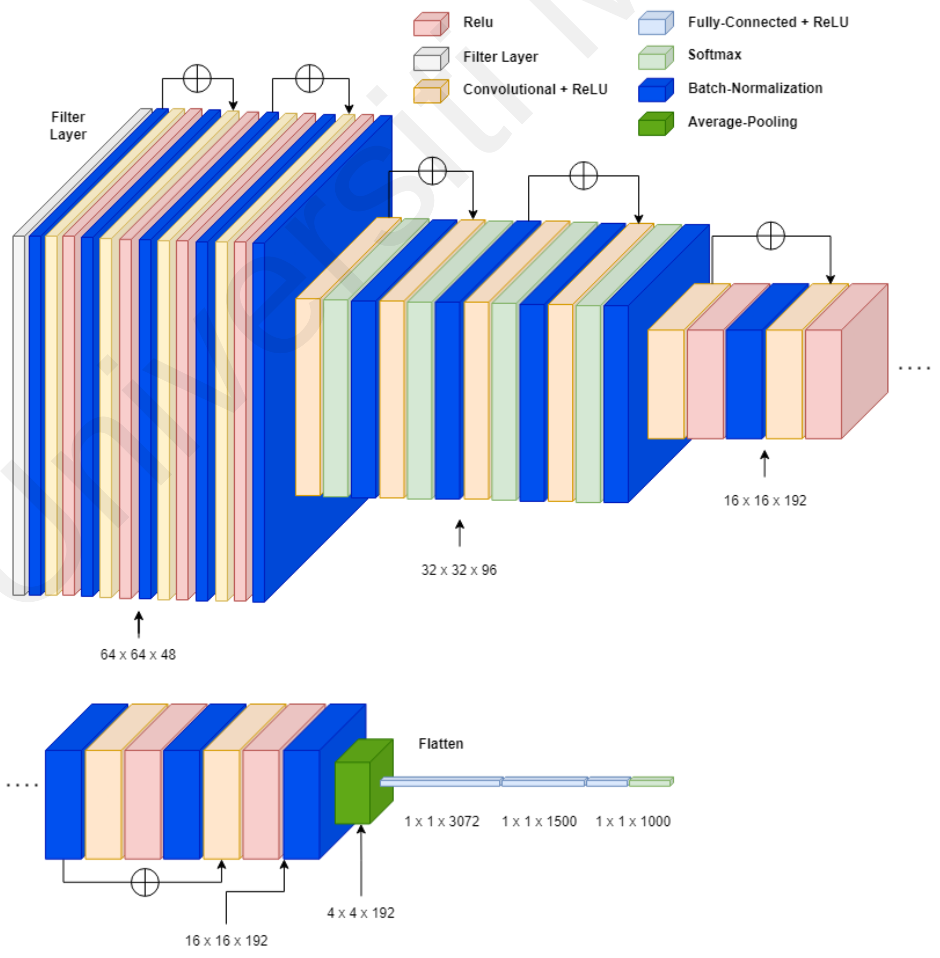

Figure 3.12: Schematic of GFNN ResNet for ImageNet_64x64

3.5. Summary

To test the GFNNs on datasets with color images, 6 pairs of CNNs are designed for ImageNet_64x64, Cifar100, and Cifar10 datasets. First baseline CNNs are designed according to the dataset's attributes, then their GFNN equivalent is implemented by replacing their first convolutional layer with 16 image processing filters. The first layer of GFNNs called the Filter Layer, consists of a customs operation defined in TensorFlow using functional API that applies the filter to each color channel separately and instead of combining the data into a single channel produces 3 different channels per filter as output. This way the GFNN network won't lose the information about different color channels of the input. Second order, DCT, sharpening, blurring, embossing, and compass gradient filters are used in the filter layer and each filter produces three channels, thus using 16 image processing filters in the filter layer is computationally equivalents to 48 convolutional layers. The compass gradient filters outnumber other types of filters because after training, the extracted features in the first layer of CNNs are similar to the output of edge detection filters. Each of the implemented models is trained on its target dataset for 5 iterations and the average of the collected quantitative variables is used for visualization and analysis of the result. A costume implementation of ResNet and VGG is chosen as baseline architectures and the networks are intentionally trained extensively to determine whether the behavior of the GFNN network is different about overfitting or not. Also, the number of layers in the baseline and their GFNN equivalent is the same, but the dimensionality of tensors that are passed between layers differs. The training time, loss and accuracy of the networks are collected per epoch both on training and validation datasets, because all three of the datasets are balanced, only accuracy is used as the performance metric, Also, Adam used as backpropagation algorithm and sparse categorical cross entropy used as the loss function for all 12 designed CNNs.

CHAPTER 4: RESULTS

4.1. Introduction

This chapter demonstrates the result of the experiment by visualizing the collected quantitative variables. Each subsection is devoted to a baseline and GFNN network that have been trained on their target dataset. Thus, proceeding six subsections are related to:

- Baseline and GFNN VGGs trained on Cifar10 dataset
- Baseline and GFNN ResNets trained on Cifar10 dataset
- Baseline and GFNN VGGs trained on Cifar100 dataset
- Baseline and GFNN ResNets trained on Cifar100 dataset
- Baseline and GFNN VGGs trained on ImageNet_64x64 dataset
- Baseline and GFNN ResNets trained on ImageNet_64x64 dataset

Below questions are answered at each subsection based on the analysis of the demonstrated variables:

1. Do the GFNNs reach a higher accuracy within a certain number of epochs?
2. Do the GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks?
3. Are the GFNNs more vulnerable to overfitting?

To average out random parameters that affects the performance like OS scheduler, cache warmup, etc., the average of five iterations of training for quantitative variables of Table 2.3 were collected. Also, to be sure that hardware performance does not vary the frequency of CPU, GPU, Memories, and bus interface was statically set and have been monitored during the experiment.

The results are demonstrated by seven graphs per network pairs as listed below:

1. Accuracy by epoch for training dataset (Line Chart)
2. Accuracy by time for training dataset (Line Chart)
3. Accuracy by epoch for validation dataset (Line Chart)
4. Accuracy by time for validation dataset (Line Chart)
5. Loss by epoch for training dataset (Line Chart)
6. Loss by epoch for validation dataset (Line Chart)
7. Cumulative training time by epoch (Line Chart)

Each graph contains the result for both baseline networks and GFNNs. The networks were trained for 40 epochs on Cifar10, 60 epochs on Cifar100, and 30 epochs on ImageNet_64x64.

In addition of line charts, the collected quantitative variables are demonstrated in form of a table, the table for each pair of baseline network and GFNN contains the average of variable:

- Training time of one epoch ($\overline{ep\_tr\_ti}$)

And also, the absolute value of variables:

- Total training time (tot_tr_ti)
- Accuracy of network on training data after training ($tot\_tr\_ac\_tr$)
- Accuracy of network on validation data after training ($tot\_tr\_ac\_vl$)

Also, the line charts are created using the collected variables, the relation between the charts and variables is described below.

In the line chart "Accuracy by epoch for training dataset", the horizontal axis is the number of epoch and the vertical axis is the value of the variable $ep\_tr\_ac\_tr$ which is the accuracy on training dataset at the end of the epoch. The value of $ep\_tr\_ac\_tr$ is drawn for the nth epoch for both of the baseline and GFNN.

In the line chart "Accuracy by time for training dataset", the horizontal axis is time and the vertical axis is the value of the variable $ep\_tr\_ac\_tr$ at a specific time. The value of $ep\_tr\_ac\_tr$ is drawn for the nth epoch of both the baseline and GFNN. But instead of no of epochs, the spanned time till the end of the nth epoch is calculated and drawn using the variable ep_tr_ti that stores the training time for each epoch. The chart demonstrates the needed training time of each network to achieve a specific accuracy on the training dataset.

In the line chart "Accuracy by epoch for validation dataset", the horizontal axis is the no of epoch and the vertical axis is the value of the variable $ep\_tr\_ac\_vl$ which is the accuracy on validation dataset at the end of the epoch. The value of $ep\_tr\_ac\_vl$ is drawn for the nth epoch for both of the baseline and GFNN.

In the line chart "Accuracy by time for validation dataset", the horizontal axis is time and the vertical axis is the value of the variable $ep\_tr\_ac\_vl$ at a specific time. The value of $ep\_tr\_ac\_vl$ is drawn for the nth epoch of both the baseline and GFNN. But instead of no of epochs, the spanned time till the end of the nth epoch is calculated and drawn using the variable ep_tr_ti that stores the training time for each epoch. The chart demonstrates the needed training time of each network to achieve a specific accuracy on the validation dataset.

In the line chart "Loss by epoch for training dataset", the horizontal axis is the no of epoch and the vertical axis is the value of the variable $ep\_tr\_ls\_ti$ which is the value of loss function while training on training dataset at the end of the epoch. The value of $ep\_tr\_ls\_ti$ is drawn for the nth epoch for both of the baseline and GFNN.

In the line chart "Loss by epoch for validation dataset", the horizontal axis is the no of epoch and the vertical axis is the value of the variable $ep\_tr\_ls\_vl$ which is the value of loss function while training on validation dataset at the end of the epoch. The value of $ep\_tr\_ls\_vl$ is drawn for the nth epoch for both of the baseline and GFNN.

In the line chart "Cumulative training time by epoch" the vertical axis is the no of epochs and horizontal axis is the time needed for nth epochs of training. The spanned time for nth epoch is the cumulative value of variable ep_tr_ti. The value has been calculated and drawn for each of the baseline and GFNN to demonstrate the training speed of the GFNNs.

The GFNNs can train faster if they can reach a specific accuracy within a lower number of epochs, or if each epoch takes less time to train. Graph no. 7 demonstrates the timing difference between epochs of the GFNNs and baseline networks and is enough to evaluate the differences of per epoch training time. Graphs no.2 and no.4 are used to evaluate and compare the needed time to reach a specific accuracy for each architecture.

In addition to accuracy and training time, the behavior of GFNN about overfitting and gradient explosion are important. Overfitting happens when the neural network maps a relation between the input data and the answer and instead of extracting useful feature from the input data, learns what is the correct answer for a specific sample. If overfitting happens the network might not perform well when receives an unseen input.

Overfitting is detected using a combination of information from graphs No.1, No.3, and No.6. If overfitting happens the value of loss function for validation dataset starts to increase and the accuracy for validation dataset starts to decrease while the accuracy of training dataset continues to improve.

Also graphs No.5 and No.6 can detect gradient explosions if it happens. Gradient explosion happens when large error gradients accumulate. In this case, large updates will be applied to the weights and biases of the network, and the network will lose its functionality. Gradient explosion can be detected by sudden changes in the value of loss function. In all experiments, the models were intentionally over-trained to analyze the behavior of GFNNs when overfitting.

4.2. Baseline and GFNN VGGs Trained on Cifar10

Table 4.1 demonstrates the value of collected quantitative variable as described in Chapter 4.1 for the baseline VGG and GFNN VGGs.

Table 4.1: Result of Quantitative Variables for VGGs on Cifar10

	tot_tr_ti	$\overline{ep\_tr\_ti}$	$tot\_tr\_ac\_tr$	$tot\_tr\_ac\_vl$
Baseline	521.00	13.54	89	83
GFNN	296.79	7.76	84	79

GFNN VGG has significantly lower training time when used on Cifar10, Training of the GFNN version finished 75% faster than the baseline while having only a 4% decrease in accuracy on validation samples.

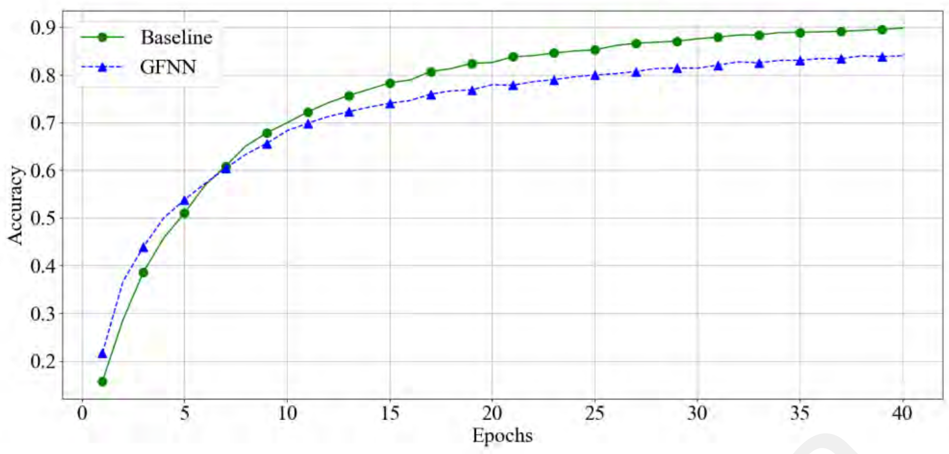

Figure 4.1: Accuracy by Epoch - VGGs - Training (Cifar10)

As demonstrated in Figure 4.1, the accuracy of GFNN VGG on training samples has a faster growth at the beginning, but its accuracy is lower than baseline at the end of the training, Figure 4.1 shows the accuracy per epoch, next Figure demonstrates the training time of the networks.

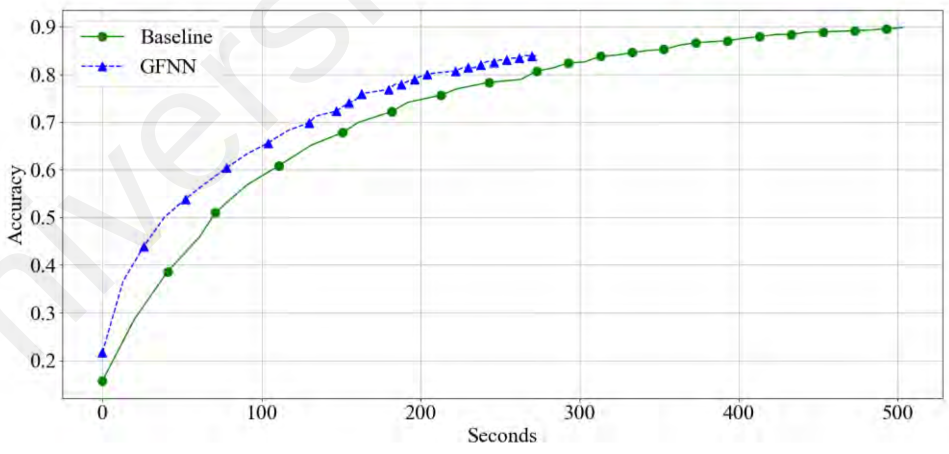

Figure 4.2: Accuracy by Time - VGGs - Training (Cifar10)

Figure 4.2 compares the needed time to achieve a certation accuracy for each of the networks, despite achieving lower final accuracy, the GFNN always is on top and has been trained faster compared to the baseline. The training time per epoch reduces significantly in the middle of the training, the reason is related to TensorFlow optimization on graph execution of models, in the case of GFNN VGG, the framework was able to optimize the execution more effectively.

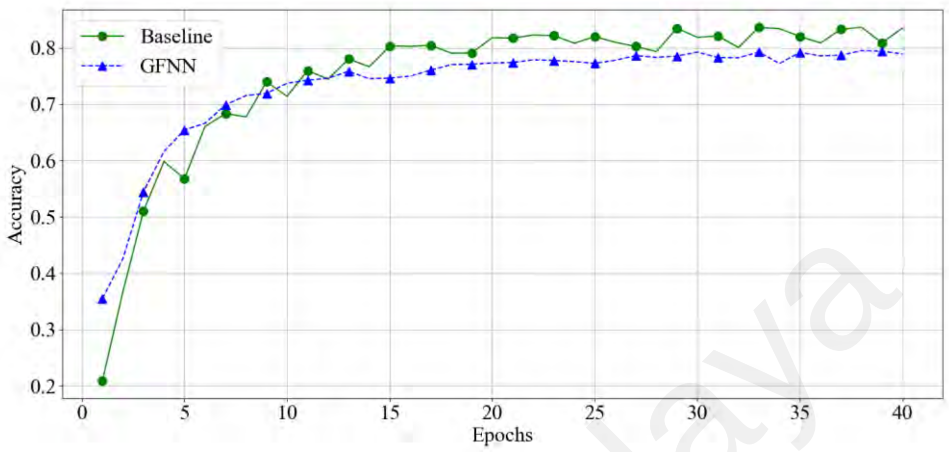

Figure 4.3: Accuracy by Epoch - VGGs - Validation (Cifar10)

While comparing the performance on validation dataset, despite having less learnable parameters GFNN VGGs can benefit from provided information by Filter Layer and have relatively identical performance to the baseline in terms of accuracy. Figure 4.3 compares the accuracy of GFNN and baseline VGG on Cifar10 validation dataset, again, the GFNN VGG's accuracy grows faster at the beginning, but the final accuracy of baseline is marginally higher. The next figure will compare the needed training time of the networks for achieving a certain accuracy.

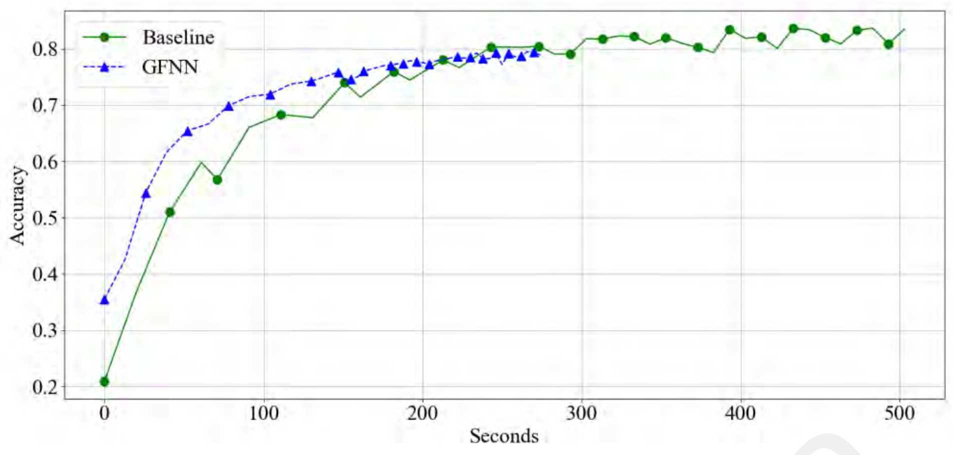

Figure 4.4: Accuracy by Time - VGGs - Validation (Cifar10)

As Figure 4.4 shows, the GFNN reaches to high accuracy significantly faster than the baseline, but the acceleration is due to faster training time per epoch rather than achieving higher accuracy with a lower number of epochs.

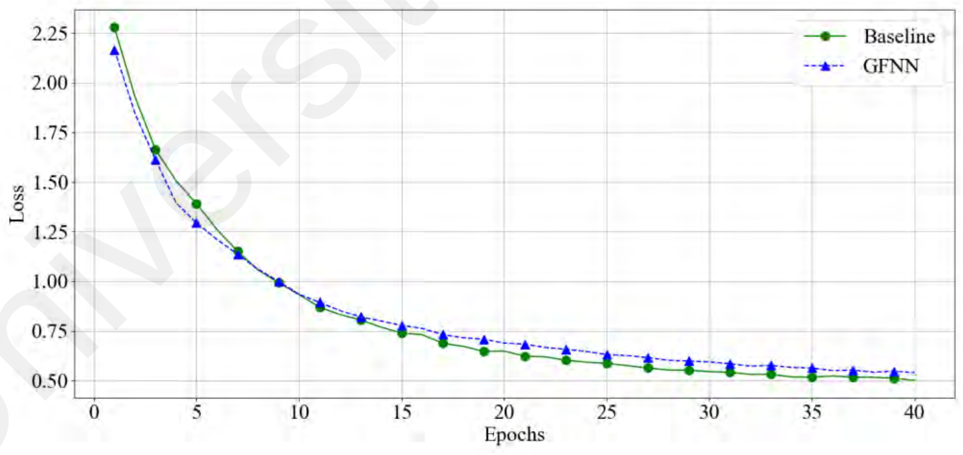

Figure 4.5: Loss Value - VGGs - Training (Cifar10)

The GFNN and baseline loss function values had an almost identical behavior on training samples, but as Figure 4.6 demonstrates the baseline's loss has more fluctuation in the case of validation dataset.

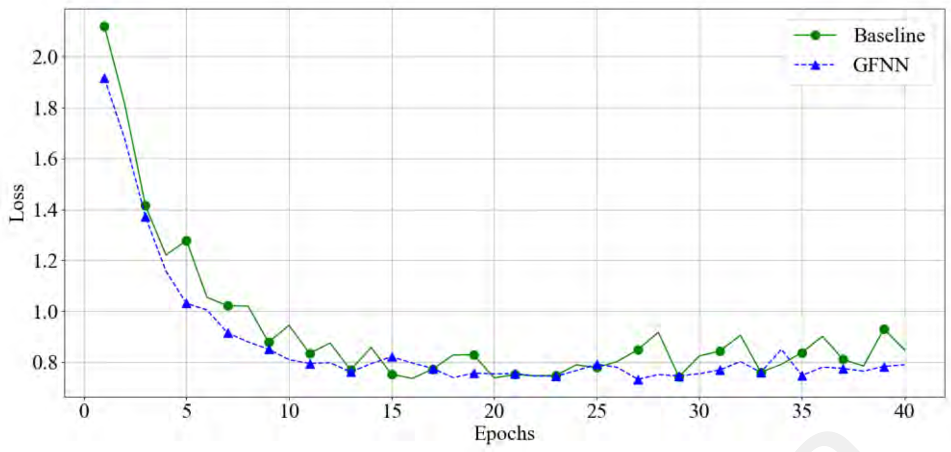

Figure 4.6: Loss Value - VGGs - Validation (Cifar10)

Both networks have a smooth decrease in loss function with no signs of overfitting, however, the GFNN had less fluctuation compared to baseline. This behavior reappears in other experiments in both cases of VGG and ResNet architectures.

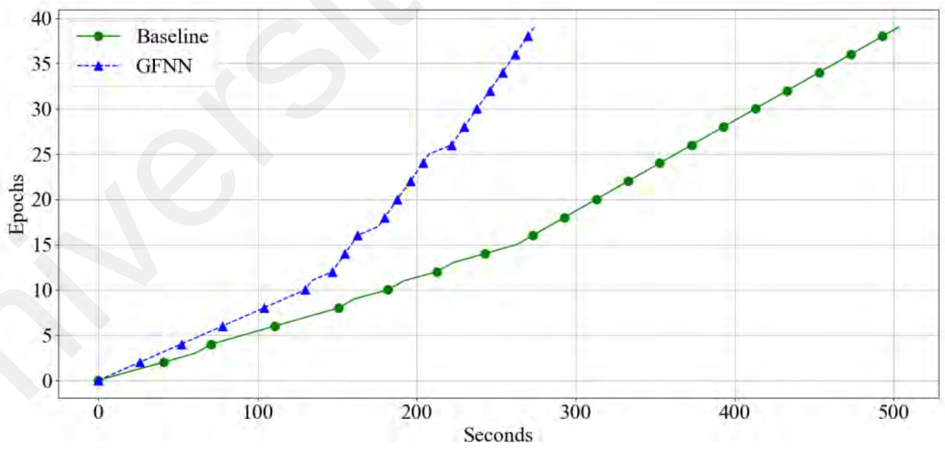

Figure 4.7: Relative Training Time -VGGs (Cifar10)

Figure 4.7 compares training time of the GFNN versus baseline and on average the GFNN's training finished 75% faster. In both cases an increase in the slope of the graphs suggests that the training time per epoch changed in the middle of the training. The reason is related to TensorFlow`s optimization on graph execution of the models, the framework was able to better optimize the execution of the GFNN.

According to the result of experiment of GFNN and baseline VGG on Cifar10:

Do the GFNNs reach a higher accuracy within a certain number of epochs? GFNN VGG has only achieved higher accuracy with lower number of epochs at the begging of the training, and generally their final performance is worse than the baseline version.

Do GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks? GFNN VGG has significantly lower training time when used on Cifar10, Training of the GFNN version finished 75% faster than the baseline while having only a 4% decrease in accuracy on validation samples.

Are GFNNs more vulnerable to overfitting? The GFNN and baseline networks had an almost identical behavior in terms of overfitting, extending the network with image processing filters and lowering the number of learnable parameters had not affected the overfitting of the networks in this specific combination of CNN architecture and dataset.

4.3. Baseline and GFNN ResNets Trained on Cifar10

Table 4.2 demonstrates the value of collected quantitative variable as described in Chapter 4.1 for the baseline and GFNN ResNets.

Table 4.2: Result of Quantitative Variables for ResNets on Cifar10

	tot_tr_ti	$\overline{ep\_tr\_ti}$	$tot\_tr\_ac\_tr$	$tot\_tr\_ac\_vl$
Baseline	1185.91	30.82	100	84
GFNN	1192.62	30.99	100	85

In contrast to traditional CNN (VGG implementation), ResNet architecture did not fully benefited from filter layer when trained on Cifar10 dataset and for most of the metrics the GFNN and baseline networks had similar behavior. This trend suggests the architectures with skip connection and residual blocks might need a different initial feature as input and edge detection image processing filters were not suitable for them to use the extracted features more efficiently. But achieved the max possible accuracy in less epochs, and consequently had a shorter training time.

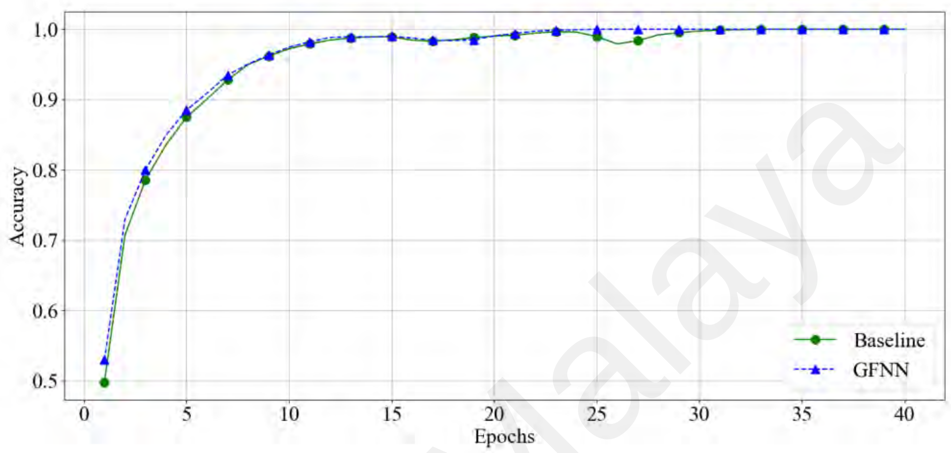

Figure 4.8: Accuracy by Epoch – ResNets - Training (Cifar10)

The Figure 4.8 demonstrates the accuracy of the networks per epochs and the baseline and GFNN version had an identical behavior in the case of training samples.

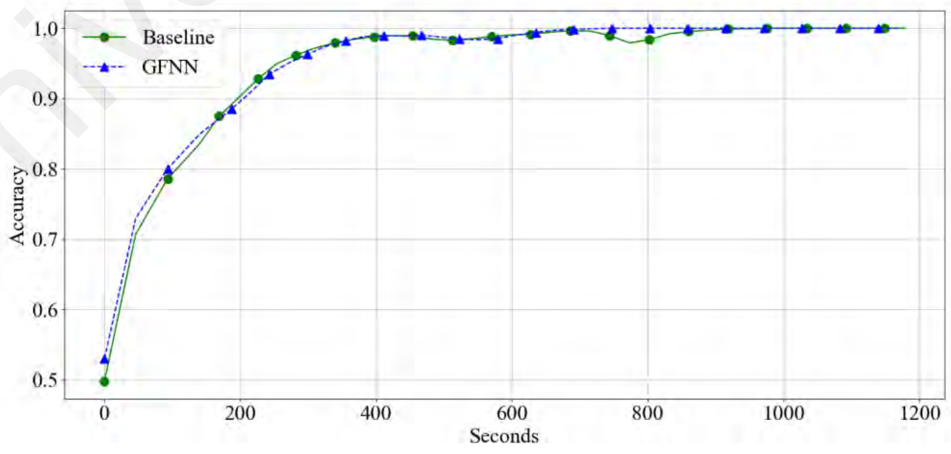

Figure 4.9: Accuracy by Time - ResNets - Training (Cifar10)

In the case of ResNet the TensorFlow framework optimization of graph execution was identical for both networks, the training time reduced per epoch after certain number of epochs, but the difference between training time per epoch for GFNN and baseline networks was in the margin of error.

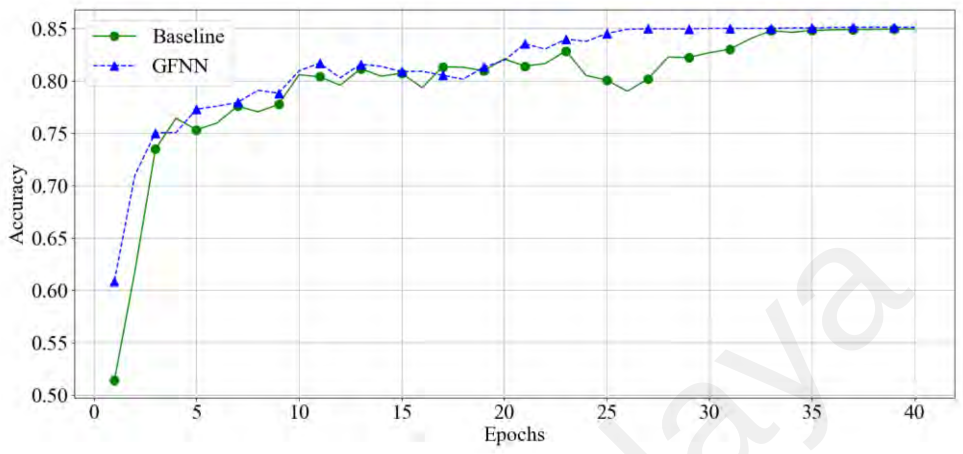

Figure 4.10: Accuracy by Epoch - ResNets - Validation (Cifar10)

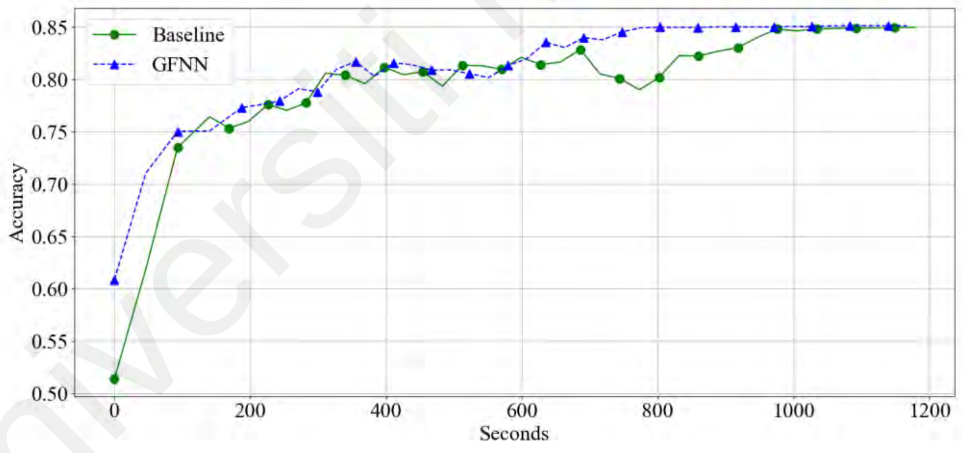

Figure 4.11: Accuracy by Time - ResNets - Validation (Cifar10)

Because training time per epoch was similar for GFNN and baseline networks, Figures 4.10 and 4.11 show an identical trend for accuracy of networks on validation dataset. As showed in Figure 4.10, the GFNN ResNet achieved the max accuracy approximately 13 epochs sooner than the baseline, and despite having an equal training time per epoch, the GFNN ResNet was faster for training the model to achieve the max possible accuracy.

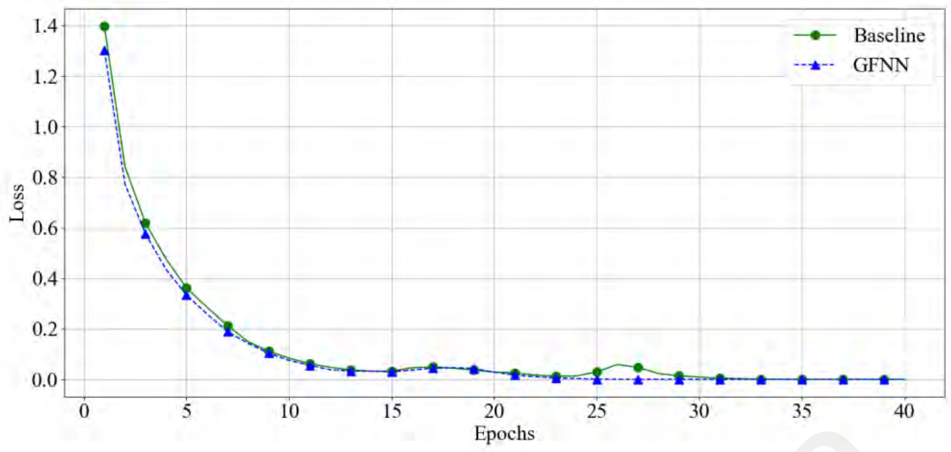

Figure 4.12: Loss Value - ResNets - Training (Cifar10)

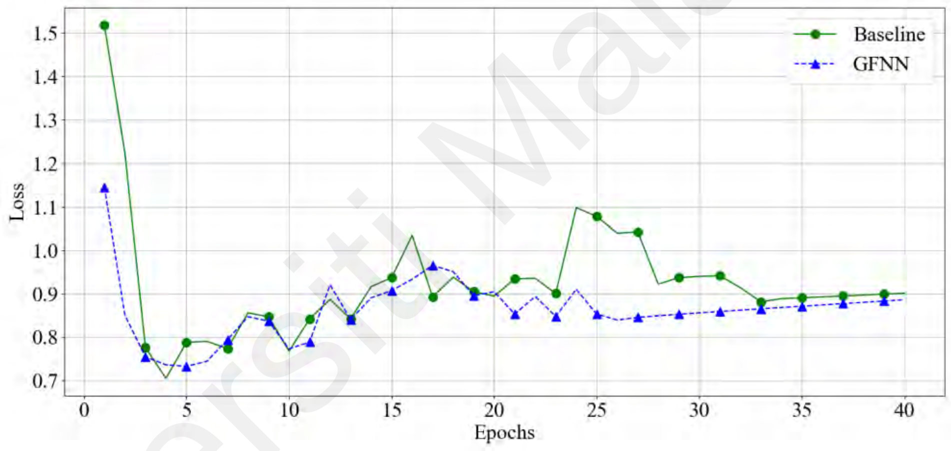

Figure 4.13: Loss Value - ResNets - Validation (Cifar10)

The loss function of GFNN and baseline ResNet had similar behavior as demonstrated in Figure 4.12, in the case of validation dataset, the GFNN ResNet converged slightly faster in early epochs, which is reflected in Figure 4.13, the accuracy of the GFNN version improves faster in the early epochs, but the improvement is minor. Despite having similar trend in the case of loss value for validation samples, the loss value of the GFNN network had less fluctuations compared to the baseline network, this effect has been seen on VGG implementations as well.

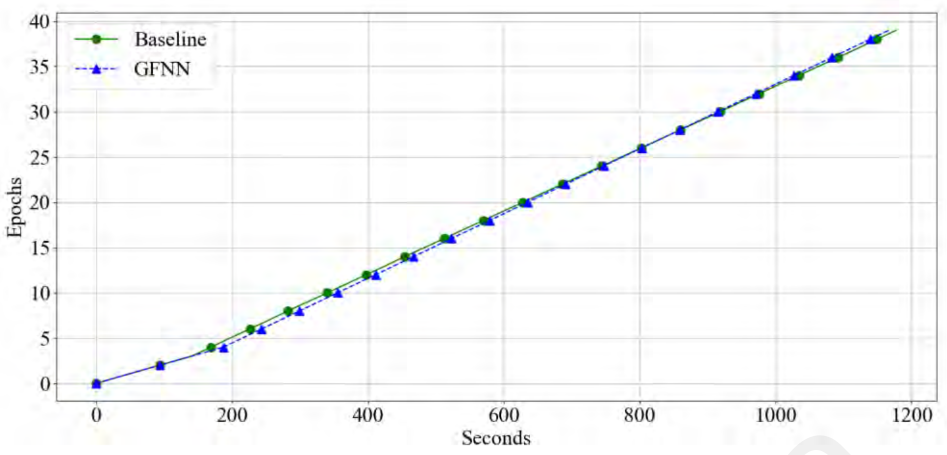

Figure 4.14: Relative Training Time - ResNets (Cifar10)

The training time per epoch for GFNN and baseline versions of ResNet is almost identical as showed in Figure 4.14, but it doesn't mean the training time of the network are the same as the GFNN ResNet achieved highest accuracy in a smaller number of epochs.

According to the result of experiment of GFNN and baseline ResNets on Cifar10:

Do the GFNN CNNs reach a higher accuracy within a certain number of epochs? GFNN ResNet achieved the highest possible accuracy on validation samples approximately 13 epochs sooner, which suggests the GFNN ResNet could reach a higher accuracy with a smaller number of epochs.

Do the GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks? The max accuracy for both networks were the same, also, training time per epoch were similar, thus training time could be faster only if the network can achieve higher accuracy in a smaller number epoch which was the case for GFNN ResNet on Cifar10.

Are the GFNNs more vulnerable to overfitting? The GFNN and baseline networks had an almost identical behavior in terms of overfitting, extending the network with image processing filters and lowering the number of learnable parameters had not affected the overfitting of the networks in this specific combination of CNN architecture and dataset.

4.4. Baseline and GFNN VGGs trained on Cifar100

Table 4.3 demonstrates the value of collected quantitative variable as described in Chapter 4.1 for the baseline and GFNN VGGs.

Table 4.3: Result of Quantitative Variables for VGGs on Cifar100

	tot_tr_ti	$\overline{ep\_tr\_ti}$	$tot\_tr\_ac\_tr$	$tot\_tr\_ac\_vl$
Baseline	743.23	12.74	78	56
GFNN	409.54	7.05	69	49

Similar to Cifar10, GFNN VGG had a good performance on Cifar100. The training for GFNN version finished 82% faster than the baseline version while having 7% decrease in accuracy on validation samples.

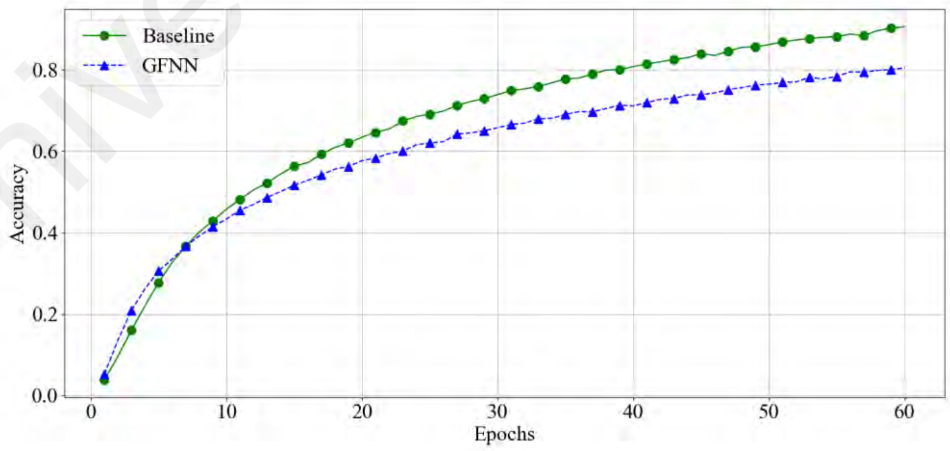

Figure 4.15: Accuracy by Epoch - VGGs - Training (Cifar100)

As demonstrated in Figure 4.15, the accuracy of GFNN VGG on training samples has a faster growth at the beginning, but its accuracy is lower than baseline at the end of the training, the difference of their accuracy is higher than Cifar10, in the case of Cifar100, GFNN VGG's accuracy had the biggest drop compared to the baseline network. Figure 4.15 shows the accuracy of the networks per epoch.

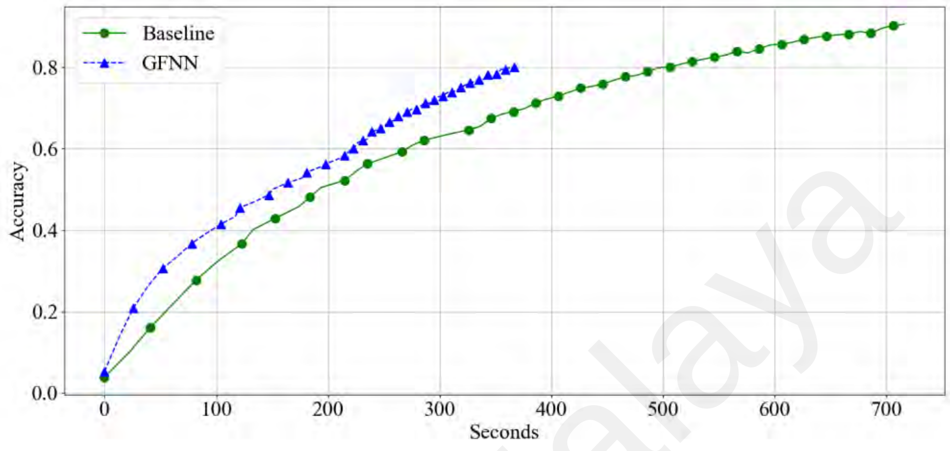

Figure 4.16: Accuracy by Time - VGGs - Training (Cifar100)

Figure 4.16 compares the needed time to achieve a certation accuracy for each of the networks, the GFNN network always is on top and has been trained faster compared to the baseline. The training time per epoch reduces significantly in the middle of the training, the reason is TensorFlow's optimization on graph execution of models, in the case of GFNN VGG, the framework was able to optimize the execution more effectively.

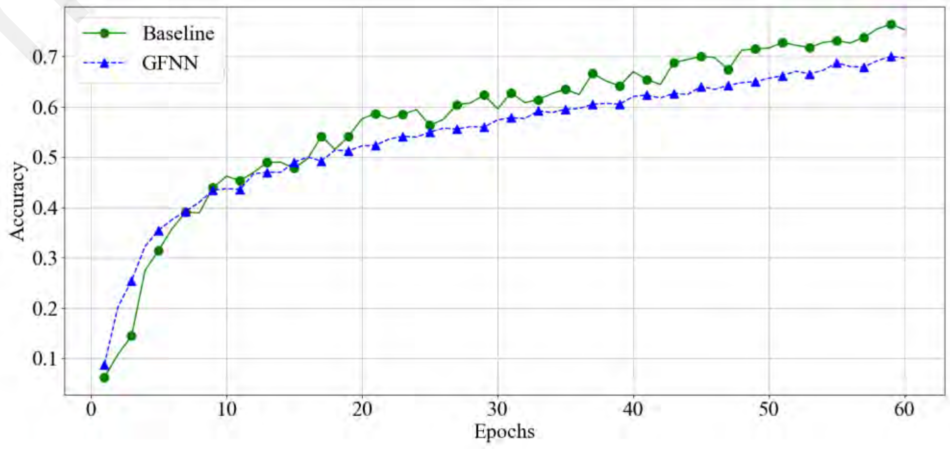

Figure 4.17: Accuracy by Epoch - VGGs - Validation (Cifar100)

As reflected in figure 4.17, despite taking much less time for per epoch training, the accuracy of GFNN is comparable with the baseline. The difference between baseline and GFNN version was bigger in case of training samples, but the final accuracy degradation was 7% on validation data. Limiting the ability of the network to extract the features of the first layer and reduction of learnable parameters had a direct impact on max achievable accuracy of the model, but as demonstrated in next figure the training time was much lower.

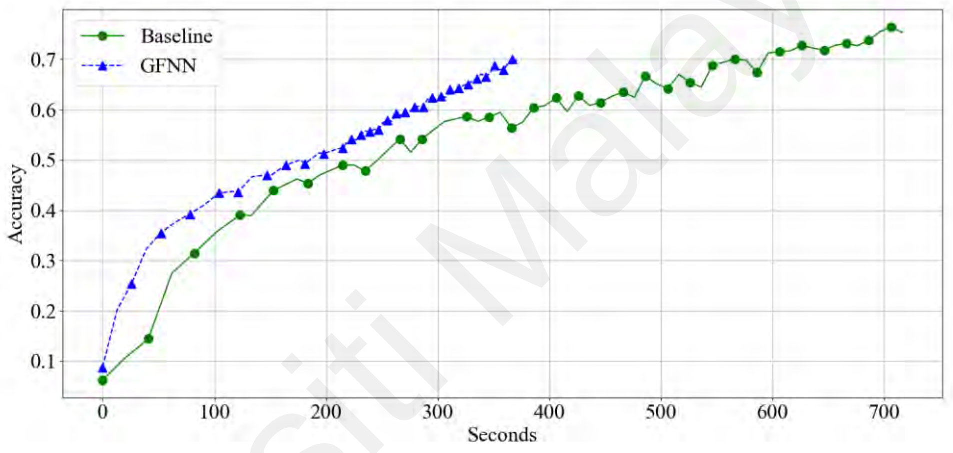

Figure 4.18: Accuracy by Time - VGGs - Validation (Cifar100)

As Figure 4.18 shows, GFNN reaches to its highest accuracy significantly faster than the baseline, but this result is due to faster training time per epoch rather than achieving higher accuracy with a lower number of epochs.

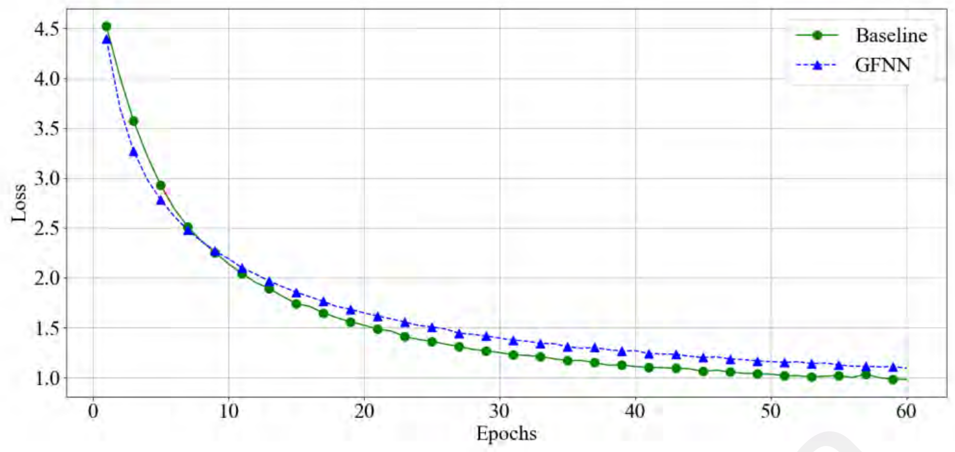

Figure 4.19: Loss Value - VGGs – Training (Cifar100)

The loss function for validation samples is demonstrated in Figure 4.20, the GFNN converges faster in the earlier epochs, but the rest is the same for both models. The networks trained for sixty epochs and according to Figure 4.19 and 4.20 there is no sign of gradient explosion.

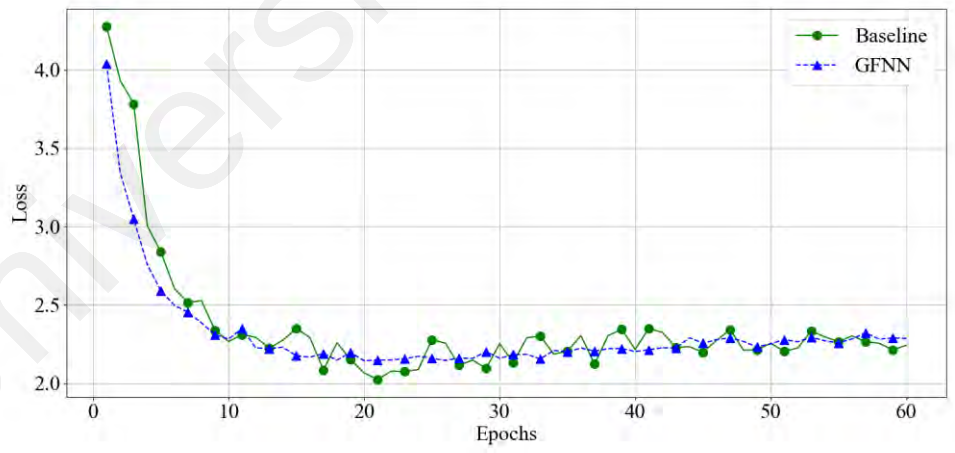

Figure 4.20: Loss Value - VGGs - Validation (Cifar100)

The GFNN and baseline loss function value had an almost identical behavior on training samples, but as Figure 4.20 demonstrates the baseline's loss has more fluctuation in the case of validation dataset.

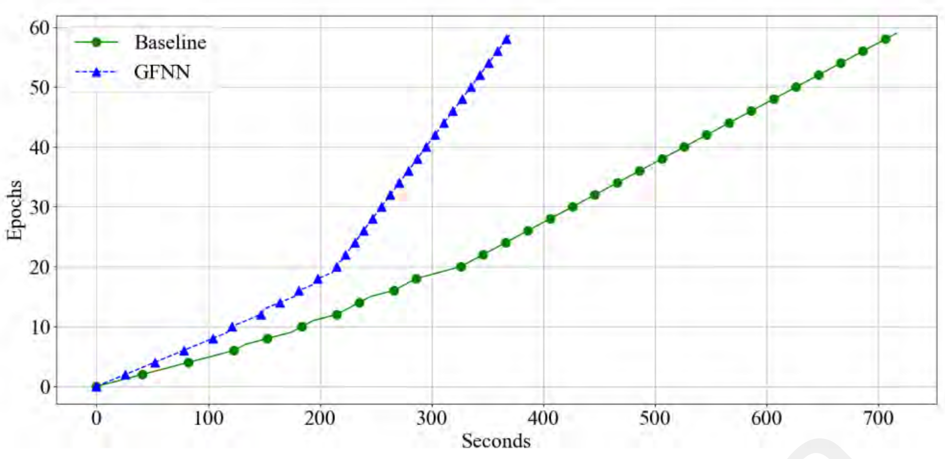

Figure 4.21: Relative Training Time - VGGs (Cifar100)

Figure 4.21 demonstrates training time of GFNN versus the baseline, the training for GFNN version finished 82% faster than the baseline version while having 7% decrease in accuracy on validation samples. In both cases an increase in the slope of the graphs suggests that the training time per epoch changed in the middle of the training. The reason is related to TensorFlow`s optimizations on graph execution of the models, the framework was able to better optimize the execution of the GFNN network.

According to the result of experiment of GFNN and baseline VGGs on Cifar100:

Do the GFNNs reach a higher accuracy within a certain number of epochs? GFNN VGG has only achieved higher accuracy with lower number of epochs at the begging of the training, and generally their final performance is worse than the baseline version.

Do the GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks? GFNN VGG has significantly lower training time when used on Cifar100, Training of the GFNN version finished 82% faster than the baseline while having a 7% decrease in accuracy on validation samples.

4.5. Baseline and GFNN ResNets trained on Cifar100 dataset

Table 4.4 demonstrates the value of collected quantitative variable as described in Chapter 4.1 for the baseline and GFNN ResNets.

Table 4.4: Result of Quantitative Variables for ResNets on Cifar100

	tot_tr_ti	$\overline{ep\_tr\_ti}$	$tot\_tr\_ac\_tr$	$tot\_tr\_ac\_vl$
Baseline	1786.40	30.56	99	53
GFNN	1776.20	30.39	99	55

In contrast to traditional CNN (VGG implementation), ResNet architecture did not fully benefited from filter layer and the GFNN and baseline networks had similar behavior. In contrast to when trained on Cifar10, the GFNN version did not achieve the max accuracy faster than the baseline, also, the ResNet networks had a gradient explosion after epoch 41 in this experiment, the excessive training was intentional and meant to cause overfitting and gradient explosion to monitor the behavior of GFNN networks.

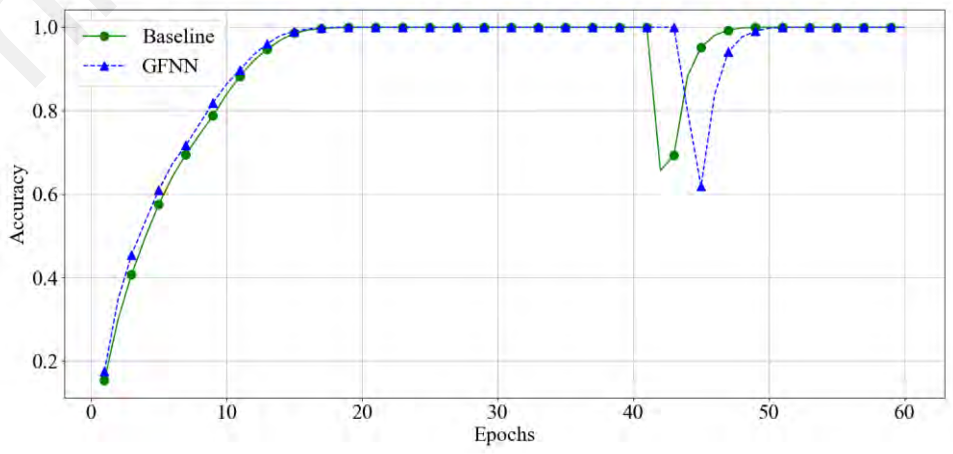

Figure 4.22: Accuracy by Epoch – ResNets - Training (Cifar100)

The Figure 4.22 demonstrates the accuracy of the networks per epochs and the base line and GFNn version had an identical behavior in the case of training samples.

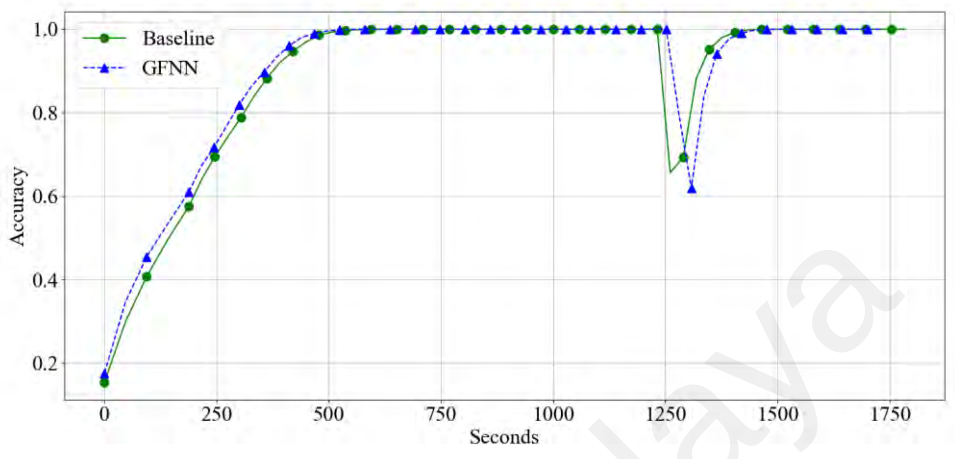

Figure 4.23: Accuracy by Time - ResNets - Training (Cifar100)

As Figure 4.22 and 4.23 show, the GFNN ResNet could not benefit from the filter layer when trained on Cifar100 and its accuracy and training time was like the baseline on training samples.

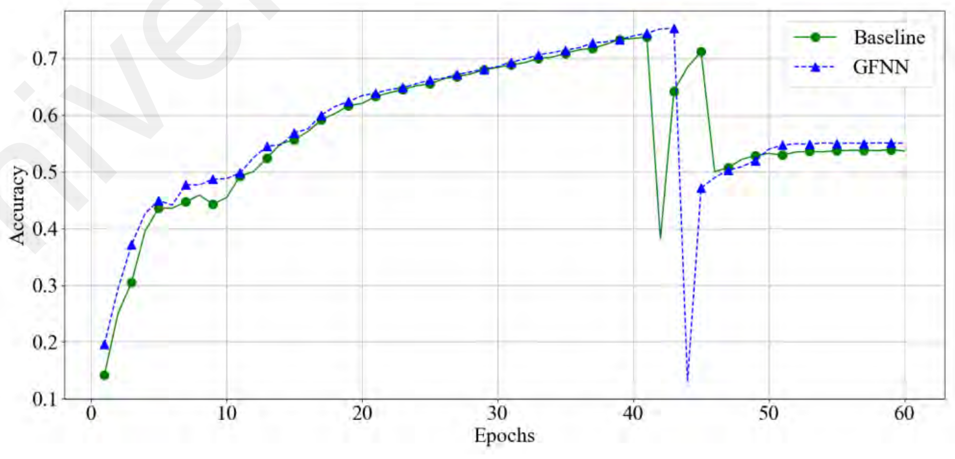

Figure 4.24: Accuracy by Epoch - ResNets - Validation (Cifar100)

As demonstrated in Figure 2.24 the accuracy of the GFNN and baseline ResNet was almost identical on training samples. Also, the graph information is only valid before gradient explosion that occurred after epoch 41 and the training was continued only to monitor the differences between two networks in the case of gradient explosion.

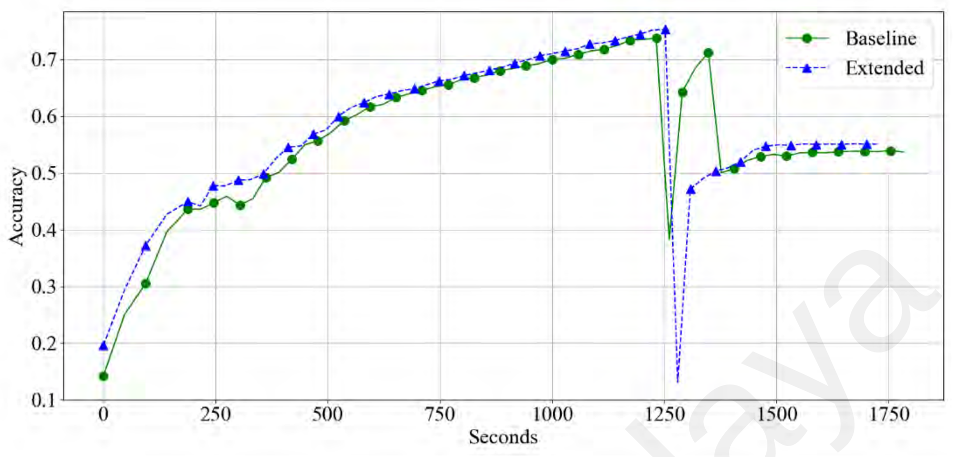

Figure 4.25: Accuracy by Time - ResNets - Validation (Cifar100)

Despite not being faster in the training, the GFNN ResNet had better final accuracy. Also, the GFNN version was slightly more resilient to gradient explosion, but this can be the result of random initialization of the model's parameters.

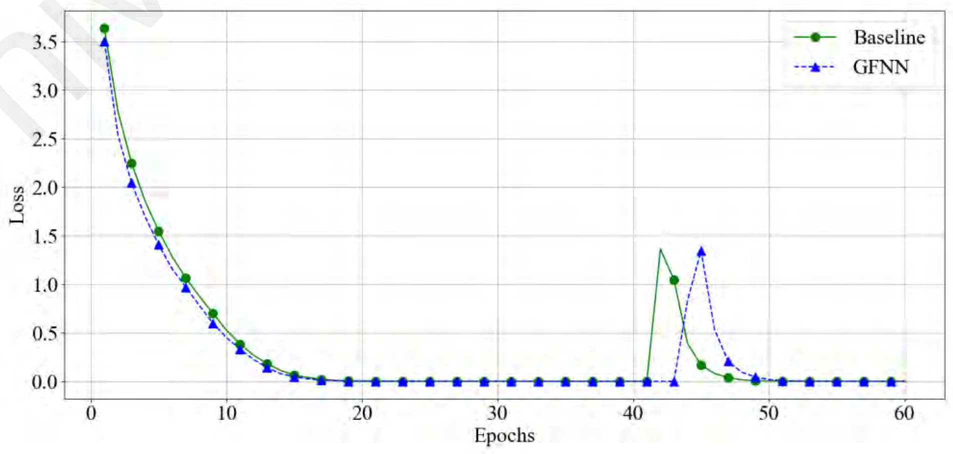

Figure 4.26: Loss Value - ResNets - Training (Cifar100)

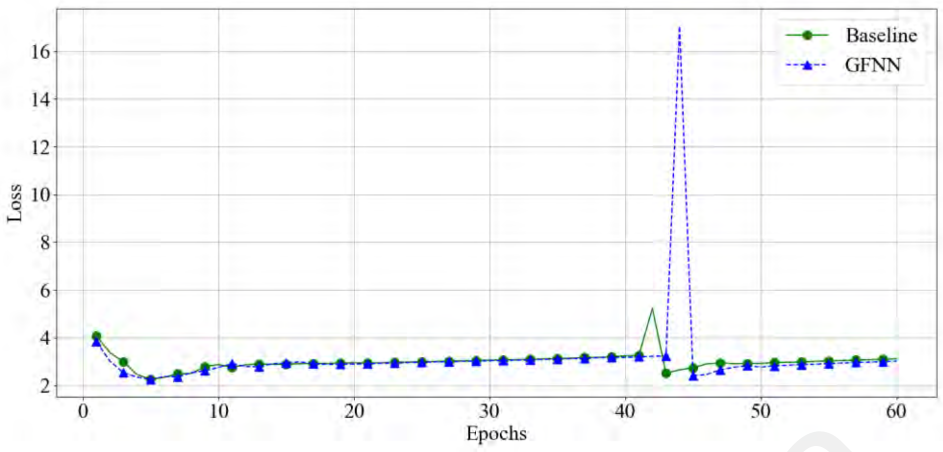

Figure 4.27: Loss Value - ResNets - Validation (Cifar100)

The loss function of GFNN and baseline ResNet had similar behavior as demonstrated in Figure 4.27, except for the delayed gradient explosion, the GFNN version did not have any different behavior regarding the overfitting and gradient explosion and the value of loss function was similar in case of both models. The identical behavior of loss value aligns with the almost identical accuracy of the GFNN and baseline models.

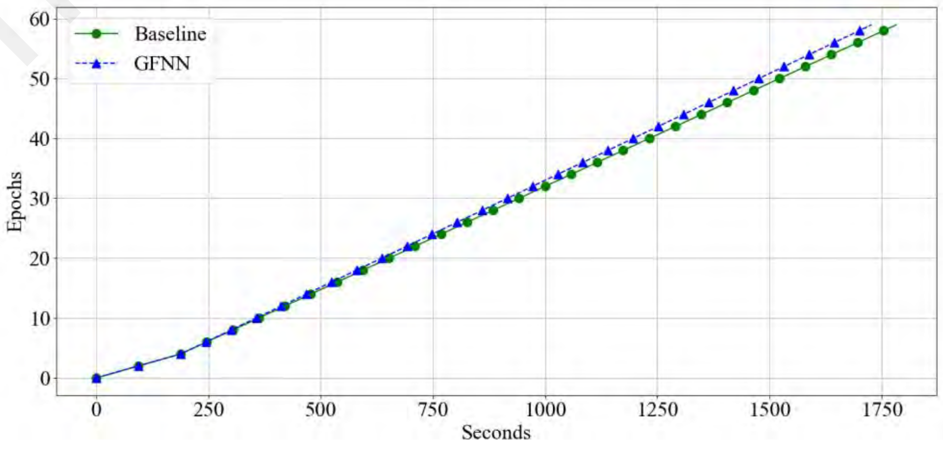

Figure 4.28: Relative Training Time - ResNets (Cifar100)

The training time for GFNN and baseline versions of ResNet is almost identical as showed in Figure 4.28 when applied on Cifar100 dataset. The GFNN ResNet was slightly faster, but the difference is in the margin of error.

According to the result of experiment of GFNN and baseline ResNets on Cifar100:

Do the GFNNs reach a higher accuracy within a certain number of epochs? Ext GFNN ended and baseline ResNet network had almost identical behavior and the GFNN ResNet could not benefit from filter layer to achieve higher accuracy in a lower number of epochs.

Do the GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks? The training time of both networks were comparable, the GFNN model achieves slightly higher accuracy and was trained slightly faster.

Are the GFNNs more vulnerable to overfitting? The gradient explosion happened in the approximately same epoch number for both networks and they had similar respond to it, thus, the GFNN and baseline networks had an almost identical behavior in terms of overfitting, GFNN the network with image processing filters and lowering the number of learnable parameters had not affected the overfitting of the networks in this specific combination of CNN architecture and dataset.

4.6. Baseline and GFNN VGGs trained on ImageNet_64x64 dataset

Table 4.5 demonstrates the value of collected quantitative variable as described in Chapter 4.1 for the baseline and GFNN VGG.

Table 4.5: Result of Quantitative Variables for VGGs on ImageNet_64x64

	tot_tr_ti	$\overline{ep\_tr\_ti}$	$tot\_tr\_ac\_tr$	$tot\_tr\_ac\_vl$
Baseline	29488.33	1017.07	45	37
GFNN	13022.00	449.22	46	33

In the case of ImageNet_64x64, the GFNN converged faster and had a significantly faster training time per epoch versus the baseline.

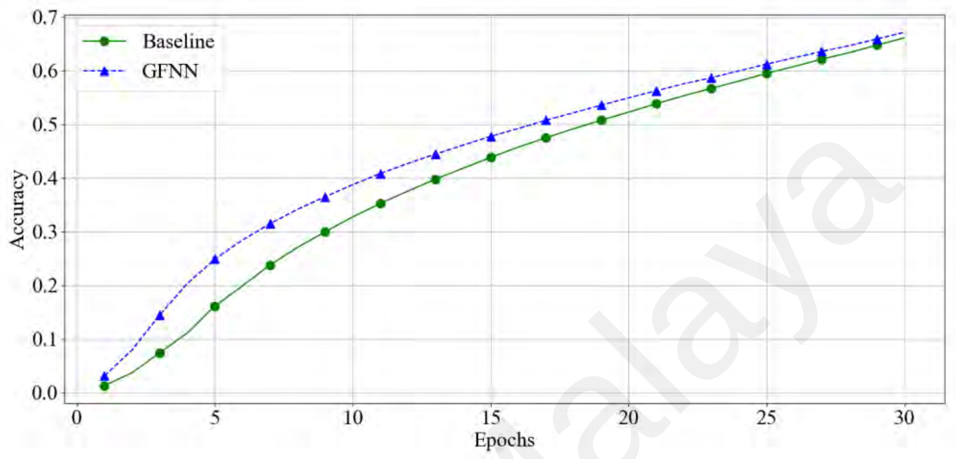

Figure 4.29: Accuracy by Epoch - VGGs - Training (ImageNet_64x64)

As demonstrated in Figure 4.29, the GFNN network achieves higher accuracy for training samples and converges faster compared to the baseline, this is not necessarily meaning the GFNN network outperform the baseline, the accuracy of the validation samples determines the model's performance.

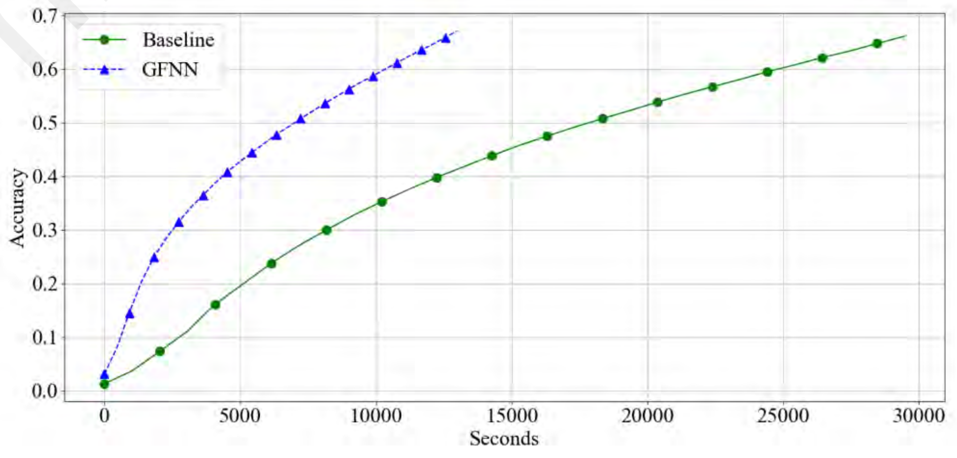

Figure 4.30: Accuracy by Time -VGGs - Training (ImageNet_64x64)

Figure 4.30 compares the needed time to achieve a certation accuracy for each of the networks, the GFNN network has the higher accuracy all the time and always is on top while has been trained significantly faster compared to the baseline.

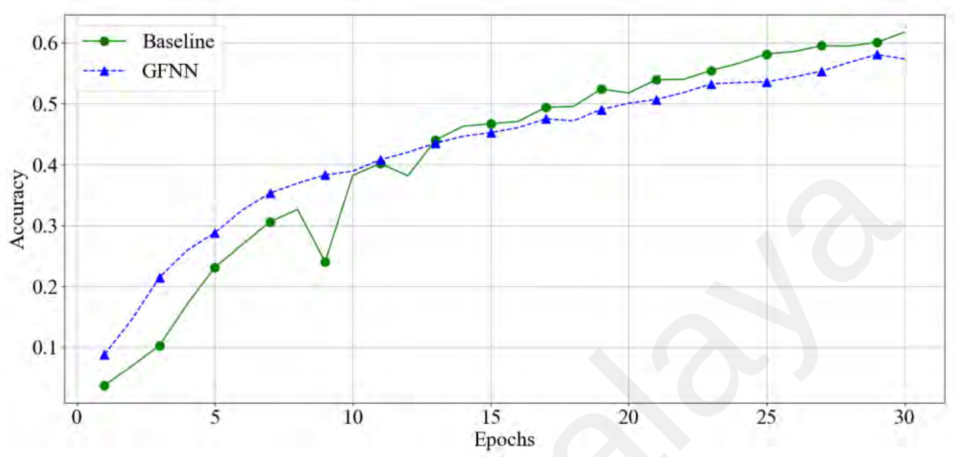

Figure 4.31: Accuracy by Epoch – VGGs - Validation (ImageNet_64x64)

While comparing the performance on the validation dataset, despite having less learnable parameters GFNN VGGs can benefit from provided information by filter layer and have relatively identical performance to the baseline in terms of accuracy. Figure 4.31 compares the accuracy of GFNN and baseline VGG on ImageNet_64x64 validation dataset, the GFNN VGG's accuracy grows faster at the beginning, but the final accuracy of the baseline is marginally higher. The next figure will compare the needed training time of the networks for achieving a certain accuracy.

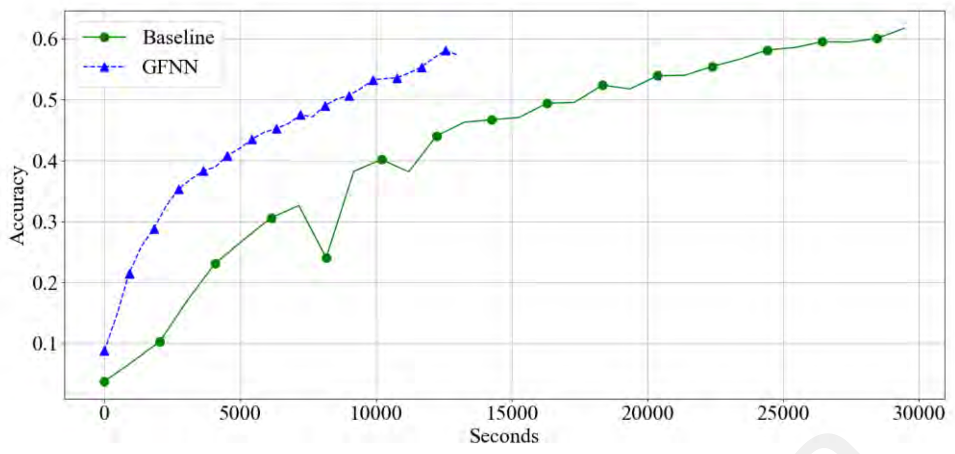

Figure 4.32: Accuracy by Time – VGGs - Validation (ImageNet_64x64)

The GFNNs had their best performance when trained on ImageNet_64x64, as Figure 4.32 shows the training finished 125% faster. The faster training is due to faster training time per epoch rather than achieving higher accuracy with a lower number of epochs.

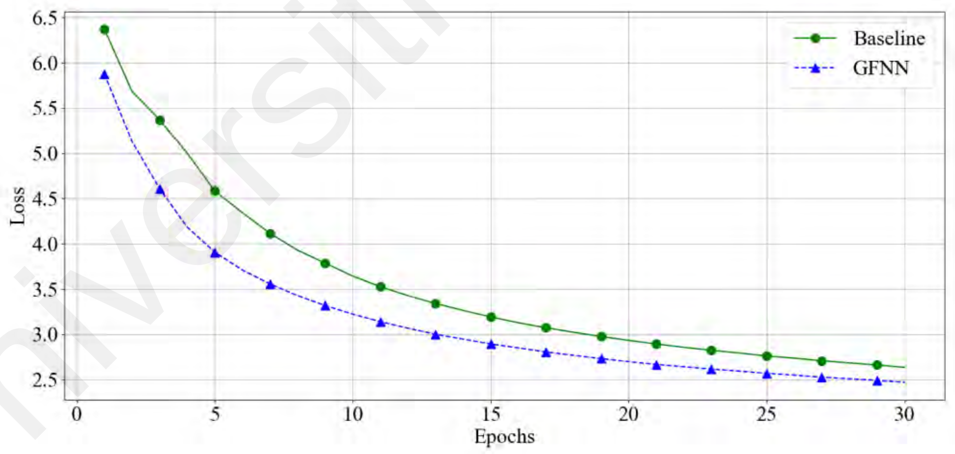

Figure 4.33: Loss Value - VGGs - Training (ImageNet_64x64)

The GFNN network converges faster and have a lower loss value on training samples, which matches with their its performance compared to the baseline.

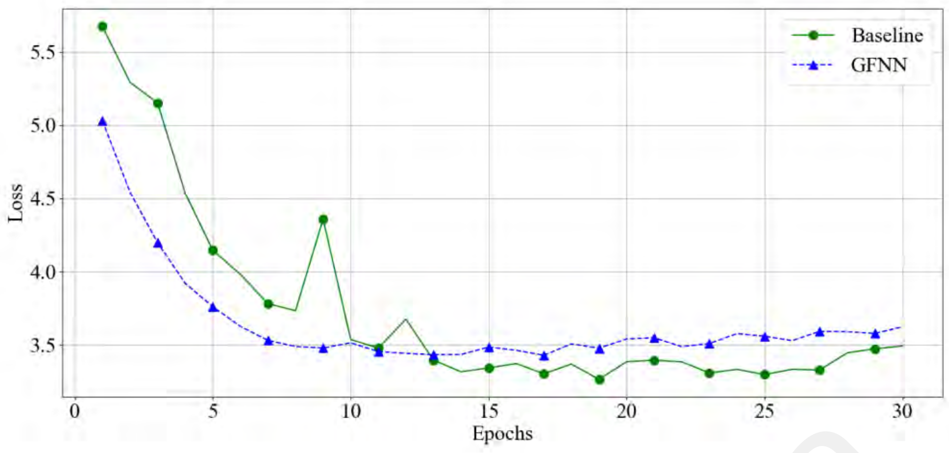

Figure 4.34: Loss Value - VGGs - Validation (ImageNet_64x64)

As demonstrated in Figure 4.34, the GFNN converged faster in the first 10 epochs, and then, the loss value plateaued afterward, the GFNN`s loss has less fluctuation and decreases smoothly.

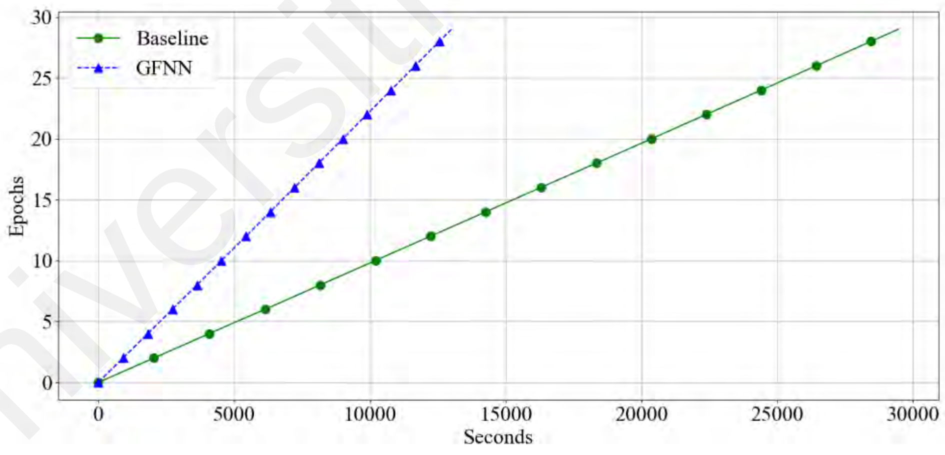

Figure 4.35: Relative Training Time - VGGs (ImageNet_64x64)

Figure 4.35 demonstrates training time of the GFNN versus the baseline per epoch. The training of GFNN version was 125% faster than the baseline, while having 5% decrease in accuracy on validation samples.

According to the result of experiment of GFNN and baseline VGGs on ImageNet_64x64:

Do the GFNNs reach a higher accuracy within a certain number of epochs? GFNN VGG has only achieved higher accuracy at the begging of the training, and generally their final performance is worse than the baseline version.

Do the GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks? GFNN VGG has significantly lower training time when used on ImageNet_64x64, training of the GFNN version finished 125% faster than the baseline while having only a 5% decrease in accuracy on validation samples.

Are the GFNNs more vulnerable to overfitting? The GFNN network had not downside regarding the overfitting compared to the baseline model.

4.7. Baseline and GFNN ResNets trained on ImageNet_64x64 dataset

Table 4.6 demonstrates the value of collected quantitative variable as described in Chapter 4.1 for the baseline and GFNN ResNets.

Table 4.6: Result of Quantitative Variables for ResNets on ImageNet_64x64

	tot_tr_ti	$\overline{ep\_tr\_ti}$	$tot\_tr\_ac\_tr$	$tot\_tr\_ac\_vl$
Baseline	78755.20	2716.85	95	40
GFNN	77139.73	2660.69	94	41

Similar to what happened in the case of Cifar100, the GFNN ResNet had no a faster training time and could not benefit from the filter layer, thus the training time and performance metrics of the GFNN network were identical to the baseline.

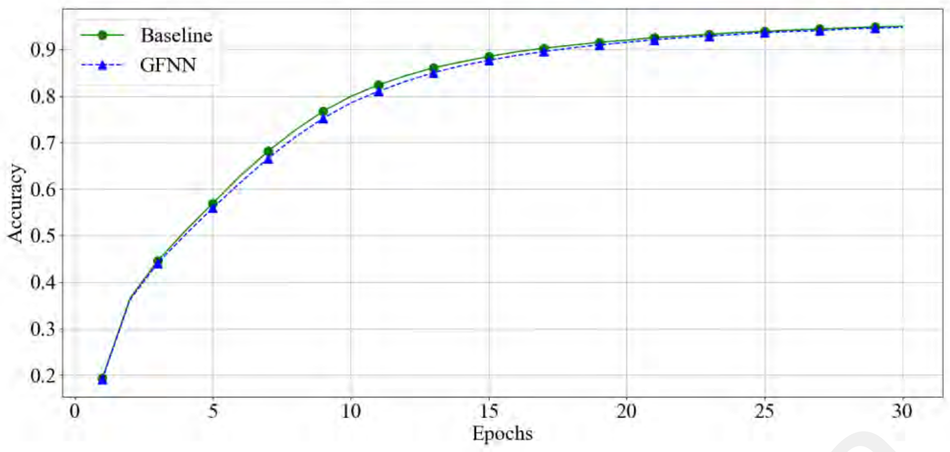

Figure 4.36: Accuracy by Epoch - ResNets - Training (ImageNet_64x64)

The Figure 4.36 demonstrates the accuracy of the GFNN and baseline networks per epochs, and they had an identical behavior in the case of training samples.

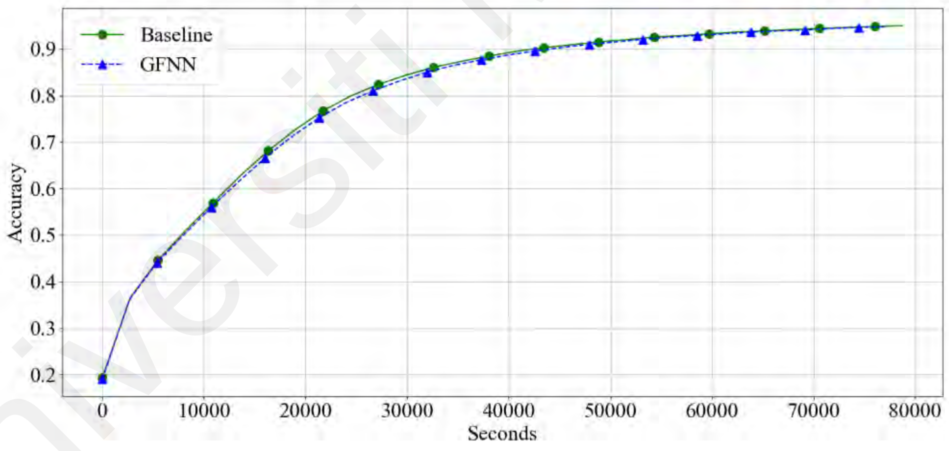

Figure 4.37: Accuracy by Time - ResNets - Training (ImageNet_64x64)

As Figure 4.37 and 4.36 show, the GFNN ResNet could not benefit from the filter layer when trained on Imgenet_64x64 and its accuracy and training time was like the baseline on training samples. The training of the GFNN network was slightly faster, but like the case of Cifar100, the difference is not considerable.

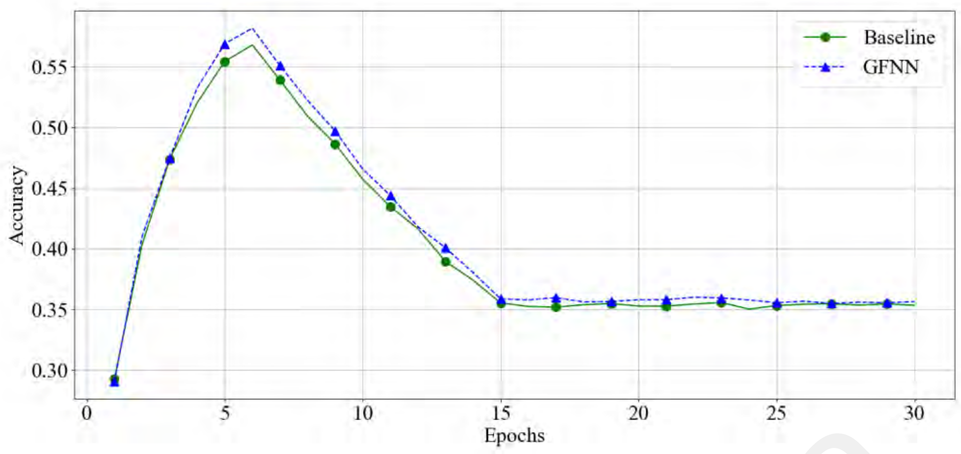

Figure 4.38: Accuracy by Epoch - ResNets - Validation (ImageNet_64x64)

As demonstrated in Figures 4.38 and 4.39, both networks start overfitting after the sixth epoch with identical behavior. But the GFNN network achieves the highest possible accuracy despite being limited by lower learnable parameters and not being free to determine the extracted features of the first layer.

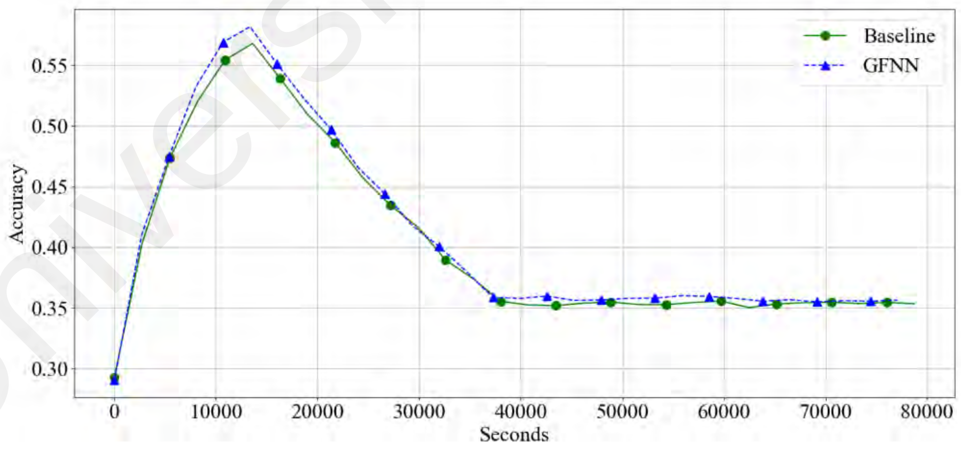

Figure 4.39: Accuracy by Time - ResNets - Validation (ImageNet_64x64)

Because training time per epoch was similar for GFNN and baseline networks, Figures 4.38 and 4.39 show an identical trend for accuracy of networks on validation dataset.

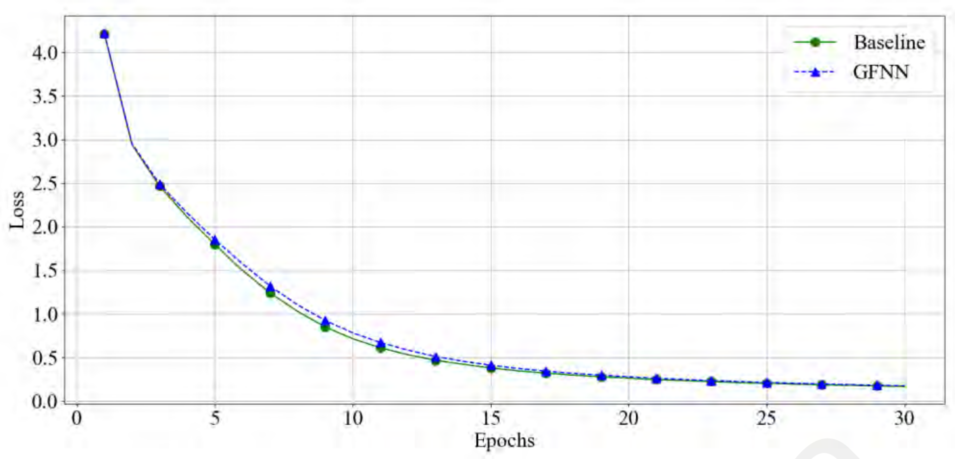

Figure 4.40: Loss Value - ResNets – Training (ImageNet_64x64)

Figure 4.41 suggests both GFNN and baseline networks have started overfitting after the sixth epoch, it is also reflected in Figures 4.38 and 4.39 where the accuracy on validation samples starts to decrease. Identical behavior in GFNN and baseline version suggests that filter layer did not have any negative impact on overfitting.

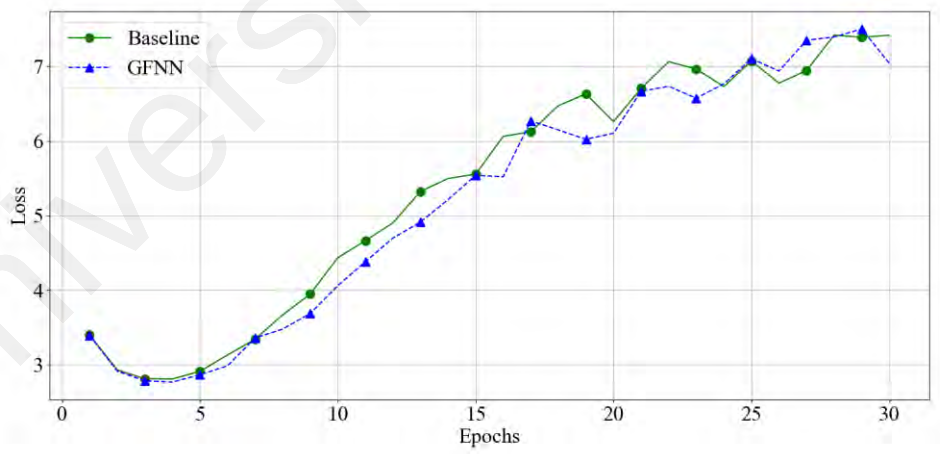

Figure 4.41: Loss Value - ResNets - Validation (ImageNet_64x64)

The loss value of both networks on validation samples starts to increase while overfitting which matches with the decrease in accuracy of validation samples. The network was intentionally trained excessively to overfit and as demonstrated in Figure

4.41, the GFNN network is not more vulnerable to overfitting and has a similar loss and accuracy to baseline while overfitting.

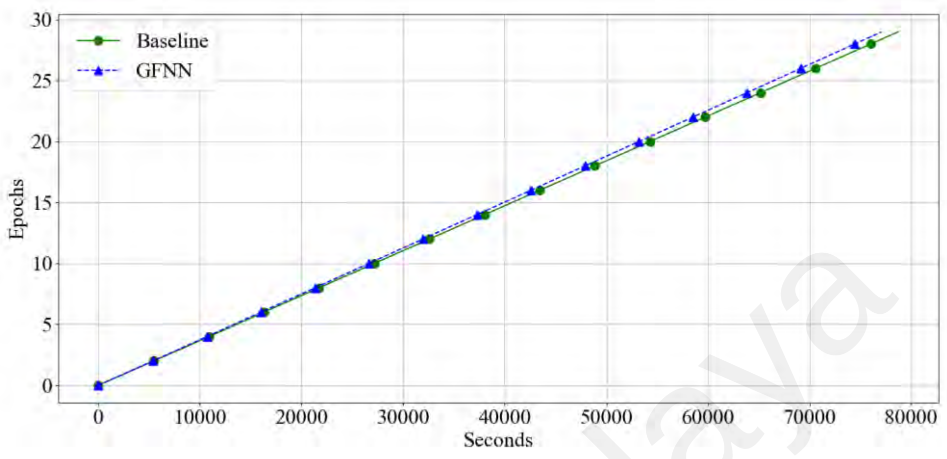

Figure 4.42: Relative Training Time - ResNets (ImageNet_64x64)

The training time for the GFNN and baseline versions of ResNet is almost identical as shown in Figure 4.42. Training of GFNN model was slightly faster, but the difference is in the margin of error. Also, the accuracy of the GFNN model was slightly higher in the case of validation samples.

According to the result of experiment of GFNN and baseline ResNets on ImageNet_64x64:

Do the GFNNs reach a higher accuracy within a certain number of epochs? The GFNN ResNet had an identical behavior to the baseline while applied to the ImageNet_64x64, generally ResNet architecture could not benefit from the provided information in the filter layer, but this effect exacerbated when applied on more complex datasets.

Do the GFNNs reach a certain accuracy sooner? What is the difference for max possible accuracy of networks? The GFNN network achieved a slightly higher accuracy, also, training time per epoch were similar, thus training time could be faster only if the network can achieve higher accuracy in a smaller number epoch. This was not the case when trained the GFNN ResNet on ImageNet_64x64, interestingly a similar architecture was achieving the max accuracy in less epochs in case of the Cifar10 dataset, suggesting that ResNet cannot benefit from filter layer when the network become more complex.

Are the GFNNs more vulnerable to overfitting? The GFNN and baseline networks had an almost identical behavior in terms of overfitting, extending the network with image processing filters lowering the number of learnable parameters had not affected the overfitting of the networks in this specific combination of CNN architecture and dataset, the overfitting happened in both network with identical trends of loss and accuracy, suggesting filter layer had not any effect on the overfitting.

4.8. Summary

The effect of extending CNNs with filter layer was different based on the architecture and complexity of the dataset. The GFNN models based on VGG had a significant boost in their training time while having slight degradation in their max accuracy. The worst accuracy degradation was 7% happened while training the VGG based networks on the Cifar100 dataset. The VGG based GFNN networks achieved 75% faster training in the case of Cifar10, 82% faster training in the case of Cifar100, and 125% faster training in the case of ImageNet_64x64.

The ResNet based GFNN networks on the other hand were minimally affected by the filter layer except for Cifar10. The GFNN ResNet managed to achieve its max accuracy 13 epoch earlier than the baseline while trained on Cifar10, but in the case of Cifar100 and ImageNet_64x64, the GFNN and baseline models had identical behavior while GFNN networks having slightly higher accuracy on validation datasets.

Regarding the overfitting and gradient explosion, both ResNet based and VGG based GFNN networks had no vulnerability compared with their baseline versions, suggesting that extending the network with the filter layer did not have any negative effects on overfitting and gradient explosion.

CHAPTER 5: CONCLUSION

In this work, the effects of extending CNNs with image processing filters were evaluated experimentally. The model acceleration technique was altered to support multi-channel sensory data. For each of Cifar10, Cifar100, and ImageNet_64 datasets, two pairs of networks based on VGG and ResNet were designed as the baseline, and their performance and training time were compared with their GFNN peers.

The results suggest that, extending CNNs with image processing kernels can improve training time of the models in specific cases. Its effectiveness depends on the complexity of the dataset and the network's architecture. Opposed to ResNet, where improvements were minimal, training time of VGG based GFNNs reduced significantly. Extending VGG based CNNs with the filter layer accelerated training time, 75% on Cifar10, 82% on Cifar100, and 125% on ImageNet_64x64, with a maximum 7% accuracy degradation. Also, the GFNN networks didn't have any disadvantages regarding overfitting or gradient explosion. The GFNNs are beneficial to accelerate the training time of CNNs on edge devices or embedded devices with limited processing power. Also, the method can be applied to color images or any other multi-dimensional sensory data.

Hardware was the major limitation of this experiment that hindered testing of the method on a variety of CNN architecture with different sets of image processing filters. Achieving a higher accuracy is also definitely possible for the used datasets, but to fulfill the objectives of this research some attributes of the hardware like CPU, GPU frequency, processes affinity, PCI-E bus power-saving state, etc. should be configured and monitored during the training. Thus, using the cloud services was not possible as they don’t provide low-level access to hardware, therefore this research was limited to the available GPU at the time. And the focus of the research was on the applicability of the method for multi-channel data rather than on achieving high accuracy.

Future work can focus on the effects of different types of image processing filters or effective merging of them with residual CNN architectures. Also, it can target other classification problems and implement the filter layer based on the mathematical properties of a target dataset's domain.

REFERENCES

Adarme, M. O., Feitosa, R. Q., Happ, P. N., Almeida, C. A. de, & Gomes, A. R. (2020). Evaluation of deep learning techniques for deforestation detection in the brazilian amazon and cerrado biomes from remote sensing imagery. *Remote Sensing*, *12*(6), 910. https://doi.org/10.3390/rs12060910

Agarwal, V., & Shendure, J. (2020). Predicting mRNA Abundance Directly from Genomic Sequence Using Deep Convolutional Neural Networks. *Cell Reports*, *31*(7), 107663. https://doi.org/10.1016/j.celrep.2020.107663

Alvarez, J. M., & Salzmann, M. (2017). Compression-Aware Training of Deep Networks. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 856–867.

Chandakkar, P. S., Li, Y., Ding, P. L. K., & Li, B. (2017). Strategies for Re-Training a Pruned Neural Network in an Edge Computing Paradigm. *2017 IEEE International Conference on Edge Computing (EDGE)*, 244–247. https://doi.org/10.1109/IEEE.EDGE.2017.45

Chandra, B., & Sharma, R. K. (2017). On improving recurrent neural network for image classification. *Proceedings of the International Joint Conference on Neural Networks*, *2017-May*, 1904–1907. https://doi.org/10.1109/IJCNN.2017.7966083

Chen, G., Zhang, X., Tan, X., Cheng, Y., Dai, F., Zhu, K., Gong, Y., & Wang, Q. (2018). Training Small Networks for Scene Classification of Remote Sensing Images via Knowledge Distillation. *Remote Sensing*, *10*(5), 719. https://doi.org/10.3390/rs10050719

Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. In *arXiv*.

Cho, J., & Lee, M. (2019). Building a compact convolutional neural network for embedded intelligent sensor systems using group sparsity and knowledge distillation. *Sensors (Switzerland)*, *19*(9), 4307. https://doi.org/10.3390/s19194307

Choi, H., Lee, Y., Yow, K. C., & Jeon, M. (2020). Block change learning for knowledge distillation. *Information Sciences*, *513*, 360–371. https://doi.org/10.1016/j.ins.2019.10.074

Choi, J. A., & Lim, K. (2020). Identifying machine learning techniques for classification of target advertising. In *ICT Express* (Vol. 6, Issue 3, pp. 175–180). https://doi.org/10.1016/j.icte.2020.04.012

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 1800–1807. https://doi.org/10.1109/CVPR.2017.195

Chrabaszcz, P., Loshchilov, I., & Hutter, F. (2017). A downsampled variant of ImageNet as an alternative to the CIFAR datasets. *ArXiv*.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine*, *35*(1), 53–65. https://doi.org/10.1109/MSP.2017.2765202

Dung, C. V., & Anh, L. D. (2019). Autonomous concrete crack detection using deep fully convolutional neural network. *Automation in Construction*, *99*, 52–58. https://doi.org/10.1016/j.autcon.2018.11.028

Fujiyoshi, H., Hirakawa, T., & Yamashita, T. (2019). Deep learning-based image recognition for autonomous driving. In *IATSS Research* (Vol. 43, Issue 4, pp. 244–252). https://doi.org/10.1016/j.iatssr.2019.11.008

Goyal, M., Knackstedt, T., Yan, S., & Hassanpour, S. (2020). Artificial intelligence-based image classification methods for diagnosis of skin cancer: Challenges and opportunities. *Computers in Biology and Medicine*, *127*, 104065. https://doi.org/https://doi.org/10.1016/j.compbiomed.2020.104065

Han, D., Kim, J., & Kim, J. (2017). Deep pyramidal residual networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 6307–6315. https://doi.org/10.1109/CVPR.2017.668

Hao-Ting, L., Shih-Chieh, L., Cheng-Yeh, C., & Chen-Kuo, C. (2019). Layer-Level Knowledge Distillation for Deep Neural Network Learning. *Applied Sciences*, *9*(10), 1966.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Hou, X., Shen, L., Sun, K., & Qiu, G. (2017). Deep Feature Consistent Variational Autoencoder. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 1133–1141. https://doi.org/10.1109/WACV.2017.131

Hu, Y., Wen, G., Luo, M., Dai, D., Ma, J., & Yu, Z. (2018). Competitive inner-imaging squeeze and excitation for residual network. In *arXiv*.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision* *and Pattern Recognition, CVPR 2017*, *2017-Janua*, 2261–2269. https://doi.org/10.1109/CVPR.2017.243

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, 448–456.

Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li, & Li Fei-Fei. (2009). ImageNet: A large-scale hierarchical image database. *CVPR09*, 248–255. https://doi.org/10.1109/cvprw.2009.5206848

Jung, J. H., Shin, Y., & Kwon, Y. (2018). Extension of Convolutional Neural Network with General Image Processing Kernels. *TENCON 2018 - 2018 IEEE Region 10 Conference*, 1436–1439. https://doi.org/10.1109/TENCON.2018.8650542

Jung, J. H., Shin, Y., & Kwon, Y. (2019). A Metric to Measure Contribution of Nodes in Neural Networks. *2019 IEEE Symposium Series on Computational Intelligence, SSCI 2019*, 1508–1515. https://doi.org/10.1109/SSCI44817.2019.9002851

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, *53*, 5455–5516. https://doi.org/10.1007/s10462-020-09825-6

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. *Cs.Toronto.Edu*, 1–58. http://www.cs.toronto.edu/~kriz/cifar.html

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*. https://doi.org/10.1061/(ASCE)GT.1943-5606.0001284

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Kuen, J., Kong, X., Wang, G., & Tan, Y. P. (2017). DelugeNets: Deep Networks with Efficient and Flexible Cross-Layer Information Inflows. *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017*. https://doi.org/10.1109/ICCVW.2017.117

Lei, X., Pan, H., & Huang, X. (2019). A Dilated CNN Model for Image Classification. *IEEE Access*, *7*, 124087–124095. https://doi.org/10.1109/ACCESS.2019.2927169

Li, Y., Cao, G., & Cao, W. (2020). LMDAPNet: A Novel Manifold-Based Deep Learning Network. *IEEE Access*, *8*, 65938–65946. https://doi.org/10.1109/ACCESS.2020.2985128

Liang, J., Zhang, T., & Feng, G. (2020). Channel Compression: Rethinking Information Redundancy Among Channels in CNN Architecture. *IEEE Access*, *8*, 147265–147274. https://doi.org/10.1109/ACCESS.2020.3015714

Liu, C., Lin, T., Wu, Y., Lin, Y., Lee, H., Tsao, Y., & Chien, S. (2019). Computation-Performance Optimization of Convolutional Neural Networks With Redundant Filter Removal. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *66*(5), 1908–1921. https://doi.org/10.1109/TCSI.2018.2885953

Liu, J., Chao, F., Lin, C. M., Zhou, C., & Shang, C. (2021). DK-CNNs: Dynamic kernel convolutional neural networks. *Neurocomputing*, *422*, 95–108. https://doi.org/10.1016/j.neucom.2020.09.005

Liu, Y., Wang, H., Gu, Y., & Lv, X. (2019). Image classification toward lung cancer recognition by learning deep quality model. *Journal of Visual Communication and Image Representation*, *63*, 102570. https://doi.org/10.1016/j.jvcir.2019.06.012

Lym, S., Choukse, E., Zangeneh, S., Wen, W., Sanghavi, S., & Erez, M. (2019). PruneTrain: Fast neural network training by dynamic sparse model reconfiguration. *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 1–13. https://doi.org/10.1145/3295500.3356156

Mathieu, M., Henaff, M., & LeCun, Y. (2014). Fast training of convolutional networks through FFTS. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 1–9.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-ResNet and the impact of residual connections on learning. *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, 4278–4284.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. *Proceedings of the IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition, *2016-Decem*, 2818–2826. https://doi.org/10.1109/CVPR.2016.308

Touvron, H., Vedaldi, A., Douze, M., & Jegou, H. (2019). Fixing the Train-Test Resolution Discrepancy. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (pp. 8252–8262). Curran Associates Inc.

Wang, J., Zhang, B., Sun, Z., Hao, W., & Sun, Q. (2018). A novel conjugate gradient method with generalized Armijo search for efficient training of feedforward neural networks. *Neurocomputing*, *275*, 308–316. https://doi.org/10.1016/j.neucom.2017.08.037

Wang, W., Zhu, L., & Guo, B. (2019). Reliable identification of redundant kernels for convolutional neural network compression. *Journal of Visual Communication and Image Representation*, *63*, 102582. https://doi.org/10.1016/j.jvcir.2019.102582

Weiss, K., Khoshgoftaar, T. M., & Wang, D. D. (2016). A survey of transfer learning. *Journal of Big Data*, *3*(1). https://doi.org/10.1186/s40537-016-0043-6

Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. *Advances in Neural Information Processing Systems*, 2082–2090.

Wiatowski, T., & Bolcskei, H. (2018). A Mathematical Theory of Deep Convolutional Neural Networks for Feature Extraction. *IEEE Transactions on Information Theory*, *64*(3), 1845–1866. https://doi.org/10.1109/TIT.2017.2776228

Woo, S., Park, J., Lee, J. Y., & Kweon, I. S. (2018). CBAM: Convolutional block attention module. *Lecture Notes in Computer Science (Including Subseries Lecture*

Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3–19. https://doi.org/10.1007/978-3-030-01234-2_1

Xia, Y., Zhou, J., Xu, T., & Gao, W. (2020). An improved deep convolutional neural network model with kernel loss function in image classification. *Mathematical Foundations of Computing*, *3*(1), 51–64. https://doi.org/10.3934/mfc.2020005

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 5987–5995. https://doi.org/10.1109/CVPR.2017.634

Xu, X., Ding, Y., Hu, S. X., Niemier, M., Cong, J., Hu, Y., & Shi, Y. (2018). Scaling for edge inference of deep neural networks. *Nature Electronics*, *1*(4), 216–222. https://doi.org/10.1038/s41928-018-0059-3

Zagoruyko, S., & Komodakis, N. (2016). Wide Residual Networks. *British Machine Vision Conference 2016, BMVC 2016*. https://doi.org/10.5244/C.30.87

Zhang, Y., & Li, X. (2020). Fast Convolutional Neural Networks with Fine-Grained FFTs. *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 255–265. https://doi.org/10.1145/3410463.3414642

Zhou, H., Alvarez, J. M., & Porikli, F. (2016). Less is more: Towards compact CNNs. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 662–677. https://doi.org/10.1007/978-3-319-46493-0_40