

Rochester Institute of Technology

RIT Digital Institutional Repository

Theses

6-2019

Performance Analysis of Fixed-Random Weights in Artificial Neural Networks

Humza Syed
hxs7174@rit.edu

Follow this and additional works at: <https://repository.rit.edu/theses>

Recommended Citation

Syed, Humza, "Performance Analysis of Fixed-Random Weights in Artificial Neural Networks" (2019). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

Performance Analysis of Fixed-Random Weights in Artificial Neural Networks

HUMZA SYED

Performance Analysis of Fixed-Random Weights in Artificial Neural Networks

HUMZA SYED

June 2019

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | KATE GLEASON
College of ENGINEERING

Department of Computer Engineering

Performance Analysis of Fixed-Random Weights in Artificial Neural Networks

HUMZA SYED

Committee Approval:

Dr. Dhireesha Kudithipudi <i>Advisor</i> Professor, RIT, Department of Computer Engineering	Date
--	------

Dr. Cory Merkel Assistant Professor, RIT, Department of Computer Engineering	Date
---	------

Dr. Raymond Ptucha Assistant Professor, RIT, Department of Computer Engineering	Date
--	------

Acknowledgments

I would like to acknowledge my adviser, Dr. Dhireesha Kudithipudi, and the members of the Neuromorphic AI Lab at the Rochester Institute of Technology for helping make this work possible. I would also like to acknowledge Dr. Panos Markopoulos for assistance in understanding tensor decompositions and the Air Force Research Labs for helping to fund part of this work. Lastly, I would like to acknowledge my friends and family who have supported me throughout my college career.

I would like to dedicate this work to my family who have continuously supported me throughout my life. My father would have been proud to see how far I've come and he will be forever remembered.

Abstract

Deep neural networks train millions of parameters to achieve state-of-the-art performance on a wide foray of applications. However, finding a global minimum with gradient descent approaches leads to lengthy training times coupled with high computational resource requirements. To alleviate these concerns, the idea of fixed-random weights in deep neural networks is explored. More critically the goal is to maintain performance akin to fully trained models.

Metrics such as floating point operations per second and memory size are compared and contrasted for fixed-random and fully trained models. Additional analysis on downsized models that mimic the number of trained parameters of the fixed-random models, shows that fixed-random weights enable slightly higher performance. In a fixed-random convolutional model, ResNet achieves $\sim 57\%$ image classification accuracy on CIFAR-10. In contrast, a DenseNet architecture, with only fixed-random filters in the convolutional layers, achieves $\sim 88\%$ accuracy for the same task. DenseNet’s fully trained model achieves $\sim 96\%$ accuracy, which highlights the importance of architectural choice for a high performing model.

To further understand the role of architectures, random projection networks trained using a least squares approximation learning rule are studied. In these networks, deep random projection layers and skipped connections are exploited as they are shown to boost the overall network performance. In several of the image classification experiments conducted, additional layers and skipped connectivity consistently outperform a baseline random projection network by 1% to 3%. To reduce the complexity of the models in general, a tensor decomposition technique, known as the Tensor-Train decomposition, is leveraged. The compression of the fully-connected hidden layer leads to a minimum $\sim 40x$ decrease in memory size at a slight cost in resource utilization. This research study helps to gain a better understanding of how random filters and weights can be utilized to obtain lighter models.

Contents

Signature Sheet	i
Acknowledgments	ii
Dedication	iii
Abstract	iv
Table of Contents	v
List of Figures	vii
List of Tables	xv
Acronyms	xvii
1 Introduction	1
1.1 Research Motivation	1
1.2 Research Objectives	2
2 Background & Related Work	6
2.1 The Perceptron and Multi-Layer Perceptrons	6
2.2 Convolutional Neural Networks	10
2.2.1 AlexNet	11
2.2.2 VGG	13
2.2.3 ResNet	13
2.2.4 DenseNet	14
2.3 Random Weights in ANNs	15
2.3.1 The Importance of Architecture Design	16
2.3.2 Random Weights in DNNs Achieving Comparable Performance	16
2.4 Random Projection Networks	23
2.4.1 ELM	23
2.4.2 Random Vector Functional Link Network	25
2.5 Tensor Decomposition	26
2.5.1 Tensor-Train Decomposition	30

3	Methodology	33
3.1	Hypothesis	33
3.2	Evaluation	34
3.3	Semi-Random CNNs	38
3.4	Extensions to the ELM	42
3.4.1	Convolutional ELM	43
3.4.2	Convolutional Random Vector Functional Link - Fully Connected	44
3.4.3	Tensor-Train Extreme Learning Machine	45
4	Results & Discussion	48
4.1	Semi-Random CNNs	48
4.2	Extensions to the ELM	69
5	Conclusion & Future Work	76
5.1	Future Work	77
	Bibliography	79
5.2	Appendix	86

List of Figures

2.1	Illustration of a Perceptron taking the weighted sum of its inputs and an additional bias term.	7
2.2	Illustration of an MLP. In this architecture each progressive layer takes the weighted sum of its inputs.	8
2.3	The LeNet-5 shallow CNN architecture consisting of convolutional, pooling, and FC layers [1].	12
2.4	The AlexNet CNN architecture consisting of multiple convolutional, pooling, and FC layers illustrating the impact of deep neural networks [2].	12
2.5	A ResNet residual block consisting of the use of skipped additive connectivity to alleviate the vanishing and exploding gradient problems [3].	14
2.6	A DenseNet dense block consisting of the use of skipped concatenating connectivity to increase the number of features extracting in each layer [4].	15
2.7	Reconstructions of 3 images when utilizing a fully random VGG net and a fully trained VGG net [5]. The visualization from the pool5 layer exhibits certain features learned in the trained net that inhibits reconstruction.	17
2.8	Generated textures from several images when utilizing a fully random VGG net and a fully trained VGG net [5]. In this figure each row corresponds to a convolutional layer in the random VGG net while a comparison is made between the 4th convolutional layer of both the trained and random nets.	18
2.9	Neural style transfer of images when utilizing a fully random VGG net and a fully trained VGG net [5]. The first row depicts the original image, while the second and third rows show the random VGG net and trained VGG net's content images with the applied style transfer. Note: The reference [13] in this image refers to [6]'s paper.	19

2.10	Experiments utilizing a percentage and a k amount of trained weights in deep Convolutional Neural Network (CNN) architectures. The white area indicates the percentage of weights trained, while the gray area indicates training on only k number of filters per convolutional layer. The blue line indicates the rest of the weights in the layer were left untrained, while the green line indicates the rest of the weights in the layer were zeroed out [7].	20
2.11	The WRN and DenseNet networks compared on both the CIFAR-10 (left) and CIFAR-100 (right) datasets at various training fractions per convolutional layer. Green and blue solid lines indicate networks with a fraction of trained weights and the rest left random. Green and blue dashed lines indicate the performance of the WRN and DenseNet networks with trained weights. Note: when $\text{frac} = 1.0$, this is actually when only a single filter is trained per convolutional layer [7].	22
2.12	Illustration of tensors in varying orders, or dimensions.	26
2.13	Illustration of a tensor decomposition in which an input \mathbf{T} is approximated by the summation of rank-one tensors. These rank-one tensors are calculated from the outer product of 1st order tensors a , b , and c	28
2.14	Illustration of the Tucker decomposition in which an input \mathbf{T} is approximated into a compressed core \mathbf{G} . This compressed core is then multiplied by the matrices representing rank-one tensors; A , B , and C	29
2.15	Illustration of a 5th order tensor decomposition into respective cores of the Tensor-Train (TT) format.	31
3.1	The CIFAR-10 dataset containing various classes ranging from airplanes to trucks [8].	35
3.2	The SVHN dataset containing digits 0 through 9 in a similar manner to MNIST, but in natural scenery [9].	35
3.3	The UCM dataset containing aerial imagery of 21 classes [10, 11].	36
3.4	The MSTAR dataset containing 10 unique vehicle classes [12].	37
3.5	The small NORB dataset containing 5 classes ranging from four-legged animals to cars [13].	37
3.6	The FMNIST dataset containing 10 classes of fashion items ranging from pullovers to boots [14].	38

3.7	Diagram depicting the operations required for SGD. (a) illustrates the forward pass as well as the error calculation and gradient calculation for the backward pass. (b) illustrates the weight update as well as calculations for regularization and momentum [15].	41
3.8	Illustration of the CELM architecture which makes use of an initial convolution layer as a feature extractor and then 2 FC layers. Skipped connections are utilized to observe their effectiveness in the possibility of increasing performance.	43
3.9	Illustration of the CRVFL-FC architecture. A convolutional layer is used to extract features that are sent to a pooling layer to reduce the spatial size of the feature maps. A normalization layer is used to decrease covariance shift and then the output is fed to the last 2 Fully Connected (FC) layers. A skipped connection is used from the input to the first FC layer as this was used in the original architecture. . . .	44
3.10	Illustration of an FC layer and its representation in a TT-FC layer format. The initial layer's inputs are decomposed into several TT cores that act as unfolded auxiliary matrices to the output.	45
4.1	Plot of each ResNet-20 configuration on the Canadian Institute for Advanced Research (CIFAR)-10 dataset averaged over 10 runs with a confidence interval of 99%. Fixed-random refers to the case of trained and fixed-random filters and downsized refers to the case of fewer filter models. The case of Linear refers to the increase of trained weights with each convolutional layer in the network. It's noted that models with trained and fixed-random filters outperform the downsized versions of the models across all cases. This difference is more noticeable at smaller fractional amounts of trained filters. This is due to the fixed-random filters enabling for a greater feature space to be extracted from in comparison to the downsized models.	49

4.2	Statistics plot of each ResNet-20 configuration on the CIFAR-10 dataset averaged over 10 runs with a confidence interval of 99%. The configurations consist of fixed-random filters in the convolutional layer and downsized models with fewer filters in the convolutional layer. It can be seen that the downsized filter models utilize fewer Giga-FLOPs (GFLOPs) in comparison to the fixed-random and fully trained models. This is because the fixed-random filter models need to backpropagate the entire network back due to their mix of trained and random filters. In contrast, downsized models demand fewer resources because less filters are utilized in these models as well as fewer operations on the last FC layer.	51
4.3	Image of a frog from CIFAR-10.	52
4.4	Image of an automobile from CIFAR-10.	52
4.5	Activated outputs of each convolutional layer in ResNet-20 on the input image of a frog from CIFAR-10. Features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	53
4.6	Activated outputs of each convolutional layer in ResNet-20 on the input image of an automobile from CIFAR-10. Similarly to the frog, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	53
4.7	Plot of each ResNet-20 configuration on the Street View House Numbers (SVHN) dataset averaged over 10 runs with a confidence interval of 99%. Fixed-random refers to the case of trained and fixed-random filters and downsized refers to the case of fewer filter models. The case of Linear refers to the increase of trained weights with each convolutional layer in the network. Unlike the results from CIFAR-10, random projection shows a surprisingly high performance, while the other models show more comparable performances to one another. This is most likely due to the higher number of training samples and possibly more simple imagery in comparison to CIFAR-10.	55

4.8	Statistics plot of each ResNet-20 configuration on the SVHN dataset averaged over 10 runs with a confidence interval of 99%. The configurations consist of fixed-random filters in the convolutional layer and downsized models with fewer filters in the convolutional layer. The downsized models utilize fewer GFLOPs as less operations are occurring in these networks. Additionally, the downsized models require fewer operations in their forward passes leading to a greater decrease in the number of operations and an additional decrease in memory size as fewer parameters are available in these models. The fixed-random models utilize a similar amount of GFLOPs to the fully trained models due to backpropagation occurring across all layers.	57
4.9	Plot of each DenseNet100-BC configuration on the CIFAR-10 dataset averaged over 5 runs with a confidence interval of 99%. Fixed-random refers to the case of trained and fixed-random filters and downsized refers to the case of fewer filter models. The case of Linear refers to the increase of trained weights with each convolutional layer in the network. The performance of random projection shows surprisingly that DenseNet's concatenating skipped connectivity can be beneficial with random weights as features are extracted and trained on at the last FC layer of the architecture.	58
4.10	Statistics plot of each DenseNet100-BC configuration on the CIFAR-10 dataset averaged over 5 runs with a confidence interval of 99%. The configurations consist of fixed-random filters in the convolutional layer and downsized models with fewer filters in the convolutional layer. Similarly to the previous statistic plots, the downsized models show a decrease in GFLOPs and memory size over the fixed-random models. In addition to this, the fixed-random weights for random projection is shown to actually have approximately 12% of the weights trained. This is due to the concatenation skipped connectivity leading to more features trained on the output layer.	61
4.11	Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of a frog from CIFAR-10. Features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	62

4.12	Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of an automobile from CIFAR-10. Similarly to the frog, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	63
4.13	Plot of each DenseNet100-BC configuration on the SVHN dataset averaged over 5 runs with a confidence interval of 99%. Fixed-random refers to the case of trained and fixed-random filters and downsized refers to the case of fewer filter models. The case of Linear refers to the increase of trained weights with each convolutional layer in the network. In contrast to the results from CIFAR-10, random projection shows high performance comparable to that of the other models. This may be due to the additional samples available to train on for the few trained layers of the network. In addition to this, the other architectures illustrate near comparable performance to the fully trained model.	64
4.14	Statistics plot of each DenseNet100-BC configuration on the SVHN dataset averaged over 5 runs with a confidence interval of 99%. The configurations consist of fixed-random filters in the convolutional layer and downsized models with fewer filters in the convolutional layer. Similarly to the previous statistic plots, the downsized models show a decrease in GFLOPs and memory size over the fixed-random models. The random projection model with only fixed-random convolutional layers is shown to have approximately 12% of the weights trained due to concatenated features trained on the output layer.	66
4.15	Plot of each ResNet-50 transfer learning configuration on the SVHN dataset averaged over 10 runs with a confidence interval of 99%. Fixed refers to the case of trained and fixed filters, from the pretrained model, and Linear refers to the increase of trained weights with each convolutional layer in the network. Fine-tuning the entire network shows increased performance over partial fine-tuning, however, near comparable accuracies can be achieved with partial fine-tuning and fewer resources are utilized as fewer memory writes occur. Additionally, the results show that performance over that of Scott et al.'s work [16] can be achieved with the partial fixed networks.	67

4.16	Performance of the random projection architectures on the Moving and Stationary Target Acquisition and Recognition (MSTAR) dataset. The CELM network with skipped connectivity to both the FC layers shows the highest test accuracy as more features are available on the output to train on.	70
4.17	Statistics of the random projection architectures on the MSTAR dataset. The architectures utilizing additional layers require more GFLOPs and additional memory to store all of their parameters. It's important to note that the Tensor-Train Extreme Learning Machine (TT-ELM) and TT-Random Vector Functional Link Neural Network (RVFL) are able to achieve a smaller model with the cost of more GFLOPs due to the additional operations occurring in the layer.	71
4.18	Performance of the random projection architectures on the small-NYU Object Recognition Benchmark (NORB) dataset. Similarly to the MSTAR dataset, the Convolutional ELM (CELM) with skipped connectivity to both FC layers achieves the highest performance as additional features are trained in this layer.	72
4.19	Statistics of the random projection architectures on the small-NORB dataset. The architectures utilizing additional layers require more GFLOPs and additional memory to store all of their parameters. The TT networks achieve a smaller model at the cost of additional GFLOPs due to the additional operations occurring in the layer.	73
4.20	Performance of the random projection architectures on the Fashion-MNIST (FMNIST) dataset. As with the other datasets, the CELM with skipped connectivity to both FC layers achieves the highest performance. However, there is less of a distinguishable gap in this more complex dataset.	74
4.21	Statistics of the random projection architectures on the FMNIST dataset. The architectures utilizing additional layers require more GFLOPs and additional memory to store all of their parameters. The TT networks achieve a smaller model at the cost of additional GFLOPs due to the additional operations occurring in the layer.	75
5.1	Image of an airplane from CIFAR-10.	87
5.2	Image of a dog from CIFAR-10.	87
5.3	Image of a truck from CIFAR-10.	87

5.4	Activated outputs of each convolutional layer in ResNet-20 on the input image of an airplane from CIFAR-10. Features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	87
5.5	Activated outputs of each convolutional layer in ResNet-20 on the input image of a dog from CIFAR-10. Similarly to the frog, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	88
5.6	Activated outputs of each convolutional layer in ResNet-20 on the input image of a truck from CIFAR-10. Similarly to the frog, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	88
5.7	Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of an airplane from CIFAR-10. Similarly to the other activated images, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	89
5.8	Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of a dog from CIFAR-10. Similarly to the other activated images, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	90
5.9	Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of a truck from CIFAR-10. Similarly to the other activated images, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.	91

List of Tables

2.1	Performance achieved when learning only a fraction of filters for the CIFAR-10 dataset for training epochs of 200 on WRN and 300 on densenets. Eff. Params indicates the number of trained parameters, while * indicates the performance when the parameters for random weights were zeroed out [7].	21
2.2	Performance on the Tiny-ImageNet dataset using the WRN model with various percentages of trained weights [7].	22
4.1	Performance on the CIFAR-10 dataset using the ResNet-20 architecture. A dividing line is used to separate networks with trained and fixed-random filters and downsized networks with fewer filters. The network is averaged over 10 runs and a confidence interval of 99% is used. From the results, the fixed-random networks are able to achieve more comparable accuracies to the fully trained model as opposed to the downsized network cases.	50
4.2	Performance on the SVHN dataset using the ResNet-20 architecture. A dividing line is used to separate networks with trained and fixed-random filters and downsized networks with fewer filters. The network is averaged over 10 runs and a confidence interval of 99% is used. The results illustrate clearly how the fixed-random models outperform the downsized models even if marginally. Additionally, the random projection network shows a surprisingly high accuracy. This may be attributed to additional samples and possibly more simple imagery.	56
4.3	Performance on the CIFAR-10 dataset using the DenseNet100-BC architecture. A dividing line is used to separate networks with trained and fixed-random filters and downsized networks with fewer filters. The network is averaged over 5 runs and a confidence interval of 99% is used. From the results, the fixed-random networks are able to achieve more comparable accuracies to the fully trained model as opposed to the zeroed out network cases. In addition to this, the random projection network with a 0 trained fractional amount is able to achieve over 88% accuracy.	59

4.4	Performance on the SVHN dataset using the DenseNet100-BC architecture. A dividing line is used to separate networks with trained and fixed-random filters and downsized networks with fewer filters. The network is averaged over 5 runs and a confidence interval of 99% is used. The results show surprisingly that both fixed-random and downsized models lead to performance extremely similar to the fully trained model. The random projection case even shows performance akin to the fully trained model. This may be attributed to a greater number of samples available as well as simpler imagery when compared to CIFAR-10.	65
4.5	Performance on the CIFAR-10 dataset after training each deep CNN for additional epochs up until 500 total trained epochs.	65
4.6	Performance on the UCM dataset by transfer learning the ResNet-50 architecture. The network is averaged over 10 runs and a confidence interval of 99% is used. A dividing line is used to separate the fixed networks fine-tuned in this work compared to that of Scott et al. [16]. Near comparable accuracies can be achieved with partial fine-tuning in comparison to entire fine-tuning of the network. In all cases, except for only training the FC layer of the network, the accuracy is shown to surpass that of the work of Scott et al. [16].	68

Acronyms

ANN

Artificial Neural Network

ANNs

Artificial Neural Networks

CELM

Convolutional ELM

CIFAR

Canadian Institute for Advanced Research

CNN

Convolutional Neural Network

CNNs

Convolutional Neural Networks

CP

canonical polyadic

CPUs

Central Processing Units

CRVFL

Convolutional Random Vector Functional Link Neural Network

DNN

Deep Neural Network

DNNs

Deep Neural Networks

ELM

Extreme Learning Machine

ESN

Echo State Network

FC

Fully Connected

FLOPs

Floating Point Operations per Second

FMNIST

Fashion-MNIST

GFLOPs

Giga-FLOPs

GPUs

Graphics Processing Units

LSM

Liquid State Machine

MLP

Multi-Layer Perceptron

MNIST

Mixed National Institute of Standards and Technology

MPP

Moore-Penrose Pseudoinverse

MSTAR

Moving and Stationary Target Acquisition and Recognition

NORB

NYU Object Recognition Benchmark

PCA

Principle Component Analysis

ReLU

Rectified Linear Unit

RVFL

Random Vector Functional Link Neural Network

SGD

Stochastic Gradient Descent

SVHN

Street View House Numbers

TT

Tensor-Train

TT-ELM

Tensor-Train Extreme Learning Machine

UAVs

Unmanned Aerial Vehicles

UCM

UC Merced Land Use

VGG

Visual Geometry Group

WRN

Wide ResNet

Chapter 1

Introduction

Deep Neural Networks (DNNs) are achieving state-of-the-art performance in several application domains such as image classification [17], target detection [18], and forecasting tasks [19]. The backpropagation algorithm [20] in these networks can train the weighted parameters to learn important features from their inputs. This enables the network to generalize to new data and classify the data to their respective classes.

1.1 Research Motivation

Unfortunately, the iterative update of these weighted parameters is a computationally intensive process with long training times. For example, convolutional neural networks (CNNs) (eg: AlexNet [2] and ResNet [3]), require lengthy training times for complex tasks, such as ImageNet [21]. More recent studies show that training AlexNet on ImageNet for 100 epochs and a batch size of 512 on the powerful DGX-1 station requires 6 hours and 10 minutes [22]. Training ResNet-50 on ImageNet for 90 epochs and a batch size of 256 on a DGX-1 station requires 21 hours [22]. Alternatively, with enough computational resources, such as 3,456 Tesla V100 GPUs, ResNet-50 was trained in 2 minutes [23].

Such long training times and massive demand on resources are impractical for a growing subset of applications, such as lifelong learning systems [24] and edge applications [25, 26]. For example, in [27], the authors focus on utilizing Convolutional

Neural Networks (CNNs) for on-device object detection in Unmanned Aerial Vehicles (UAVs). They observe that conventional DNNs incur high computational costs to process frames of a video on-device. Additionally, in many UAVs, Central Processing Units (CPUs) are used over Graphics Processing Units (GPUs) due to power concerns. This work showcases lightweight networks designed for resource constrained applications. A critical design constraint is to design algorithms that can be trained on-device, in real-time, to adapt to dynamic environments. Another example is learning from few samples through methodologies such as meta-learning [28]. In [28], the authors illustrate how specialized training techniques can enable for networks to train on few samples with minimal gradient calculations. This leads to a network that can train quickly for new tasks. This type of training methodology is ideal for continual lifelong learning systems that require learning new tasks quickly in a computationally efficient fashion. In contrast to faster training models, faster inference models with low latency features are available to accelerate the post-deployment phase of DNNs [29, 30]. However, the training time bottleneck for large and small datasets, is still an active research problem [31, 32]. With the evolving need to move training to the edge, it is critical to study compute-lite architectures with fast training times.

1.2 Research Objectives

The central objective of this research is to study how fixed-random weights can potentially reduce resources and training time in Artificial Neural Networks (ANNs) and deep CNNs while retaining high performance. In this context, fixed-random weights refer to weights that are initialized to a random value and then left untrained, or fixed, throughout the network’s training process. The utilization of fixed-random weights is one methodology for decreasing training time and resource utilization. Other methodologies include the usage of compression [33] or pruning [34] to decrease the number of parameters in a pre-trained network or while training the network. This can lead

to a decrease in the number of Floating Point Operations per Second (FLOPs) for the forward pass of the network, which can be beneficial for inference. However, these techniques do not address the amount of time needed to train these networks as additional computations can occur from the compression or pruning overhead [35, 36, 37]. Alternatively, the use of fixed-random weights can potentially allow for reduced training time while retaining the network’s large parameter space. Maintaining the large parameter space can be beneficial as the input’s initial dimensionality can be extracted into a larger dimensionality allowing for separability between the data [38]. This can enable the network to optimize its trained weights while leveraging the features extracted from the random weights. To evaluate the use of fixed-random weights, various methodologies are explored in this work, such as setting entire layers of networks to fixed-random weights, setting portions of layers to fixed-random weights, and setting the first layer of a network to fully random, or 100% random, and then slowly decreasing the percentage of random weights in layers further down throughout the network’s architecture.

In addition to the above methodologies, the second objective of this work is to extend the architecture of random projection networks trained using the Moore-Penrose Pseudoinverse (MPP) [39] to approximate the least squares solution [40, 41, 42, 43, 44, 45]. These networks are shown to train in orders of magnitude faster than stochastic gradient descent approaches. In [43] and [44], a CNN is first trained end to end with a softmax classification layer and then transfer learned as a feature extractor for a random projection network that is used in place of the fully-connected layers from the original CNN. This method enabled their models to achieve high performance on digestive organ classification for wireless capsule endoscopy and traffic sign recognition. Unfortunately, these works don’t highlight the memory requirements for the transfer learned models. In [45], the authors train a random projection network with convolution and pooling layers. The network utilized random weights in all layers and

only trained on the output layer. The authors showed that high performance could be achieved on handwritten digit recognition, but didn't address the memory required with the addition of each additional layer. Additionally, few works explore the use of skipped connectivity in random projection literature [46, 47, 48]. The use of skipped connectivity was shown to increase performance in these works but this connection was limited to skipped connectivity from the input to the output layer and did not further explore the use of greater connectivity. Increased skipped connectivity may be beneficial to explore as these connections have been shown to boost the performance of many DNNs today [3, 4]. To address these concerns, random projection networks are extended further with additional layers, containing fixed-random weights, as well as concatenating skipped connectivity, similar to that of DenseNet [4]. The use of additional random projection layers are shown to lead to transformations in the data that are beneficial for the trained portion of these networks. Likewise, the use of skipped concatenating connectivity allows for earlier layer features to be utilized in the training process to obtain optimal weights.

The third objective is to use a tensor decomposition technique to support compressed layers in an Extreme Learning Machine (ELM) and thereby reduce the model size. Tensor decomposition allows for tensors, or multi-dimensional arrays, to be transformed into their respective low-rank model enabling for an approximation of the original model. This can potentially lead to a decrease in parameters and lead to savings in resource utilization. This idea is utilized in a random projection network to decrease the demand on memory and to assess if a high performance model can be achieved with little to no performance drop as compared to its uncompressed counterpart.

To summarize, the main objectives of this work are the following:

1. Empirically evaluate the performance of networks utilizing fixed-random weights and assess the benefits of using this technique in comparison to smaller equiv-

alent networks.

2. Extensions of feed-forward random projection networks to enable high performing models at fast training times.
3. Exploration of tensor decomposition in random projection networks to minimize the parameter space while retaining performance.

Chapter 2

Background & Related Work

In this chapter a brief overview of ANNs is initially discussed. The practice of utilizing random weights in ANNs as a means to decrease the number of trained parameters and the training time of these network is then reviewed. The chapter concludes with the usage of pruning and compression techniques in Artificial Neural Network (ANN) literature.

2.1 The Perceptron and Multi-Layer Perceptrons

An ANN is a computational model that learns an approximation for a function given a series of inputs [49]. ANNs make use of artificial neurons for computations with synaptic connections between these neurons. These artificial neurons are typically of the Perceptron model introduced by Frank Rosenblatt in [50]. The Perceptron model consists of a neuron with synaptic connections, or weights, connected to its respective inputs. The neuron takes a weighted sum of its inputs and utilizes an activation function to obtain an output. An illustration of the Perceptron can be seen in Figure 2.1 as well as its respective calculation in (2.1).

$$\hat{\mathbf{y}} = \mathbf{f}\left(\sum_{i=1}^n \mathbf{x}_i * \mathbf{w}_i + \mathbf{b}\right) \quad (2.1)$$

Depicted in both the figure and equation are the terms \mathbf{x} , \mathbf{w} , \mathbf{b} , and $\hat{\mathbf{y}}$. \mathbf{x} is the

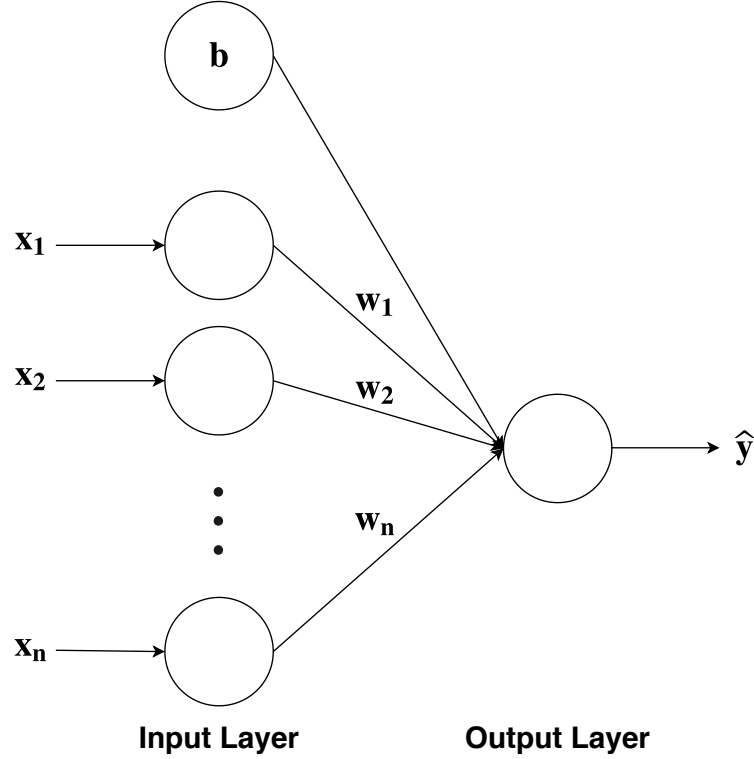


Figure 2.1: Illustration of a Perceptron taking the weighted sum of its inputs and an additional bias term.

input to the neuron with its weights, \mathbf{w} , and a bias term \mathbf{b} , which is added on to offset the boundary decision of the algorithm. $\hat{\mathbf{y}}$ is the predicted output after the weighted sum of the inputs is sent through an activation function $\mathbf{f}()$. This activation function can be as simple as a threshold value to perform binary classification on an input. To update the weights of the model, a stochastic learning rule was employed as shown in (2.2).

$$\mathbf{w}^{(t+1)} = \mathbf{w}^t + \eta(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)})\mathbf{x}^{(i)} \quad (2.2)$$

\mathbf{t} is the current training epoch of the model, η is the learning rate to adjust how much the weights would be changed, and $\mathbf{y}^{(i)}$ is the ground truth observation for sample \mathbf{i} . The Perceptron by itself could only perform linear separability between its inputs. Therefore, to handle problems that were not linearly separable, the Multi-

Layer Perceptron (MLP) was created. The MLP, as illustrated in Figure 2.2, makes use of a number of Perceptrons as artificial neurons in an ANN structure. The artificial neurons in this network create a hidden layer that takes the weighted sum of its inputs which is then sent to a nonlinear activation function. A set of common activation functions utilized in ANNs include sigmoid, hyperbolic tangent, or tanh, and Rectified Linear Unit (ReLU)) [51]. The equations for these activation functions can be seen in (2.3), (2.4), and (2.5).

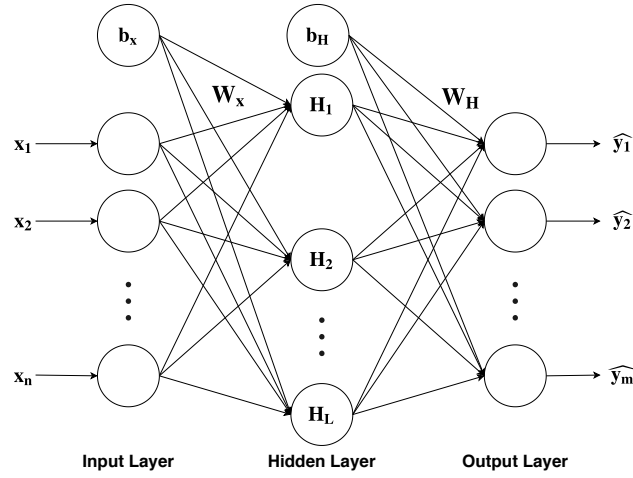


Figure 2.2: Illustration of an MLP. In this architecture each progressive layer takes the weighted sum of its inputs.

$$\text{sig}(z) = \frac{1}{1 + e^{-z}} \quad (2.3)$$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (2.4)$$

$$\text{ReLU}(z) = \max(0, z) \quad (2.5)$$

Through these activation functions the input's value is transformed to be within a smaller number range. In the sigmoid activation function the values would be between

a range of $[0, 1]$, while the tanh function sets the value to be between a range of $[-1, 1]$. Alternatively, the ReLU activation function keeps the original value if it's equal to or above 0. These activation functions enable ANNs to learn more complex structures of its inputs. After taking the activated weighted sum of its inputs the network can perform a softmax activation function on the output layer to calculate which class the data is classified to be. The softmax operation can be seen in (2.6).

$$g(z_c) = \frac{e^{z_c}}{\sum_{i=1}^c e^{z_i}} \quad (2.6)$$

The softmax operation takes its inputs, z_c and calculates the probability of which class the input would be associated with. The highest probability would be considered the classified class for the given input. To configure the network such that it classifies its input correctly, a cost, or loss, function needs to be minimized as well as an adjustment to the weights in the network. A number of loss functions have been introduced, but in this work the log loss, or cross-entropy loss function, seen in (2.7), is focused on as it's commonly used in image classification tasks.

$$\text{Loss}^{(i)} = -\log\left(\frac{e^{o_{yi}^i}}{\sum_{j=1}^c e^{o_c^i}}\right) \quad (2.7)$$

In this loss function, o is the output of the last layer and yi is the correct class, or ground truth, of the sample i across all classes c . To minimize this loss the weights in a network are updated with learning rules, such as Stochastic Gradient Descent (SGD) through the backpropagation algorithm [20]. In this algorithm, each weight of the network is adjusted such that the loss would decrease over the training time. Through the use of multiple layers an ANN could learn to map its inputs to its selected outputs such that the data would be separated.

2.2 Convolutional Neural Networks

In this section CNNs are discussed as these networks are further explored in this work. Initially introduced in [52], CNNs are ANNs that consist of convolutional layers as well as dense, or Fully Connected (FC), layers used in MLPs. The convolutional layers used in these network are made up of filters with receptive fields, or filter sizes, that convolve over input images at a given stride. For each filter a single value is outputted by taking a weighted sum of its inputs. This allows for a similar operation to the FC layers, as a weighted sum of the inputs is taken, but in a sparse manner. Similarly to FC layers, a non-linear activation is typically applied to the output of a convolutional layer.

The sparse manner of the convolutional layer to the FC layer is illustrated through the calculation of the input weights of both the FC and convolutional layers. The comparison of the number of weights used in an FC layer as opposed to a convolutional layer is illustrated in (2.8) and (2.9) respectively.

$$\mathbf{W}_{\text{in}} = \mathbf{w} * \mathbf{h} * \mathbf{d} * \mathbf{N} \quad (2.8)$$

$$\mathbf{W}_{\text{in}} = \mathbf{K}^2 * \mathbf{F} * \mathbf{d} \quad (2.9)$$

The number of weights, denoted by \mathbf{W}_{in} , to be calculated in an FC layer would be equal to the product of the input's width, height, and depth, denoted by \mathbf{w} , \mathbf{h} , and \mathbf{d} , respectively, and the number of neurons in the FC layer, \mathbf{N} . Alternatively, a convolutional layer would only require the product of its filter size, K , squared, the number of filters, F , and the depth of the input. This means for a given input image of dimensions $32 \times 32 \times 3$ and 64 neurons, a FC layer would require $32 * 32 * 3 * 64 = 196,608$ input weights to be trained as opposed to a convolutional layer with 64 filters and

a filter size of 3×3 requiring only $3 * 3 * 64 * 3 = 1,701$ weights to be trained. Additionally, unlike the FC layers, which require that the input image is flattened down to a vector form, convolutional layers can convolve over an input while retaining the given width, height, and depth allowing for spatial features to be preserved from the input.

In addition to convolutional layers, CNNs typically make use of pooling layers. Pooling layers are sub-sampling layers that decrease the dimensionality of an input. The most common pooling layers are max pooling layers and average pooling layers. In max pooling layers, the highest valued pixel, given a receptive field, is selected as the output. This operation occurs across the entire image to obtain a maximally activated output from an input. In contrast, the average pooling operation takes the average pixels in its receptive field. This allows for a low pass filtering operation on the input. This type of layer is commonly used after convolutional layers to transform and decrease the dimensionality of the input.

[52] illustrated how an end to end Convolutional Neural Network (CNN) could be trained with the backpropagation algorithm. In [1], the LeNet-5 CNN architecture, shown in Figure 2.3, was introduced as an improved version of the original design. This architecture consisted of 2 convolutional layers, 2 pooling layers, and 3 FC layers. Through the use of successive convolutional and pooling layers hierarchical features were able to be extracted from input images and then densely connected through the FC layers. This network was able to effectively learn representations from its input without the need for hand crafted features. It also paved the way for further research into CNNs.

2.2.1 AlexNet

Although LeNet-5 showed the initial potential of CNNs, CNNs were not widely adopted as they required large amounts of data to train on and were expensive in

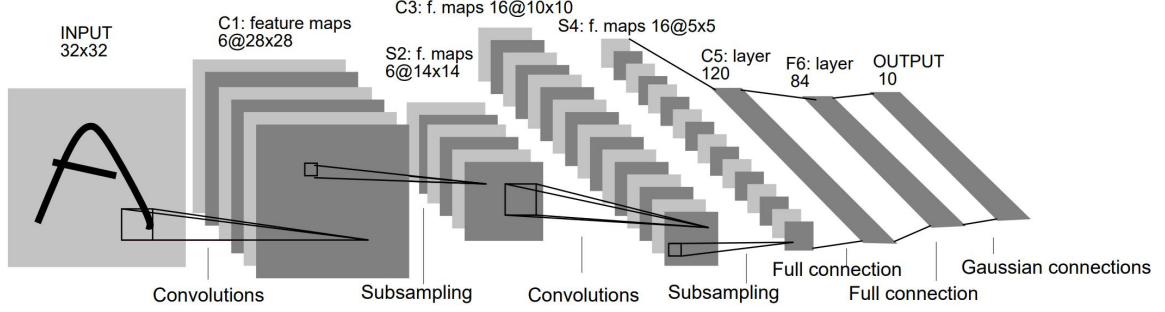


Figure 2.3: The LeNet-5 shallow CNN architecture consisting of convolutional, pooling, and FC layers [1].

terms of resources due to the tuning process of the weights, or parameters, in their networks. It wasn't until the introduction of AlexNet, depicted in Figure 2.4 that CNNs became popularized and widely used. AlexNet was merely a variant on the architecture of LeNet-5. It utilized convolutional, pooling, and FC layers just as LeNet-5 did. However, the authors of the AlexNet paper emphasized how depth played an important role in achieving high performance. They illustrated this by using an 8 layer network, containing 5 convolutional layers and 3 FC layers, as well as an additional 3 max pooling layers after some of the convolutional layers, that achieved state-of-the-art performance on the ImageNet dataset. This led to the beginning of the deep learning era as more researchers began to adopt deep CNN architectures to evaluate the effectiveness of these models on other aspects of computer vision.

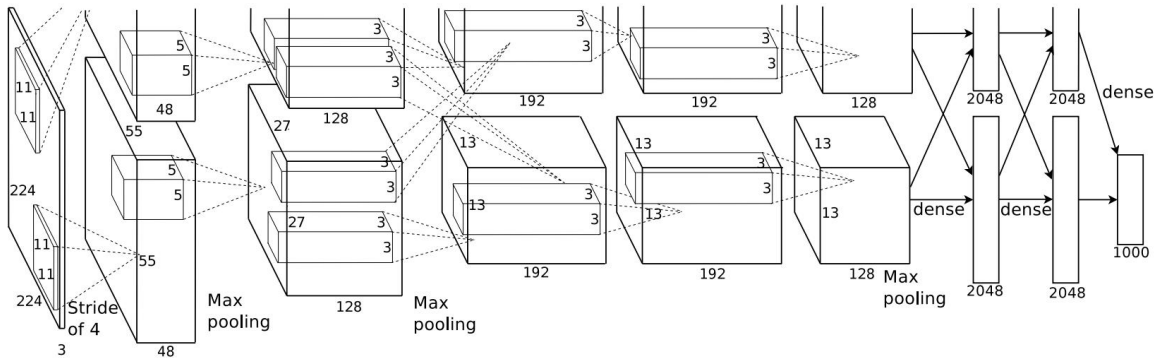


Figure 2.4: The AlexNet CNN architecture consisting of multiple convolutional, pooling, and FC layers illustrating the impact of deep neural networks [2].

In addition to introducing a deeper CNN architecture. The authors make use

of techniques, such as utilizing ReLU activation functions, data augmentation, and dropout [53]. The method of using ReLU activation functions led to faster convergence as opposed to activation functions, such as sigmoid and tanh. While data augmentation and dropout assisted in regularizing the network. All 3 of these techniques have since been utilized on various networks.

2.2.2 VGG

Following AlexNet was the Visual Geometry Group (VGG) Network which achieved state-of-the-art results on ImageNet surpassing AlexNet’s performance [54]. In these VGG networks, many 3×3 filter sizes were used as well as additional convolutional layers leading to variants of the architecture, such as VGG-16 and VGG-19, where the number after the hyphen denotes the number of layers not including the pooling layers. With the additional layers used in this architecture, the number of trained parameters increased as well as the number of resources needed to compute.

2.2.3 ResNet

As research into CNNs continued, researchers found that creating deeper networks didn’t necessarily always increase performance and in most cases would actually lead to decreased performance. This was due to the vanishing and exploding gradient problem [55, 56]. In the vanishing gradient problem, gradients calculated using back-propagation grow closer to 0 in the earlier layers due to saturation in the non-linear activation functions of these deep networks. This leads to little changes to the parameters space in early layers. In contrast, the exploding gradient problem is where large gradients occur in the network leading to large changes in the parameter space. This can lead to an unstable network that does not converge properly. The ResNet architecture alleviates these problems through the introduction of residual connections. These residual connections are skipped connections from previous layers that map to

later layers enabling for identity mapping, depicted in Figure 2.5. More specifically, this skipped connection is an addition operation on a previous input and a current layer’s output. With the usage of these residual connections, the ResNet architecture was able to obtain state-of-the-art performance on numerous datasets, including ImageNet and the MS COCO datasets [57]. With these residual connections, much deeper networks could be created while enjoying an increase in performance rather than a degradation.

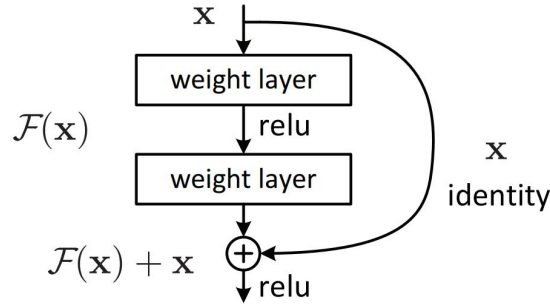


Figure 2.5: A ResNet residual block consisting of the use of skipped additive connectivity to alleviate the vanishing and exploding gradient problems [3].

2.2.4 DenseNet

Taking inspiration from networks with skipped connectivity, such as ResNet, the DenseNet architecture was formed [4]. This architecture enables for a large number of features to be extracted as all layers were connected through feature map concatenation, as opposed to ResNet’s summing of feature maps. This leads to $\frac{L(L+1)}{2}$ connections in a network as opposed to L connections, where L denotes the number of layers in a network. In DenseNet architectures few filters are used to extract features for each layer, but as more features are concatenated to the entire amount of feature maps collected a substantial amount of salient feature can be extracted from these dense blocks, as shown in Figure 2.6.

As this dense connectivity can lead to a large amount of parameters, measures are taken to decrease the parameter space. The dense blocks of the network employed the

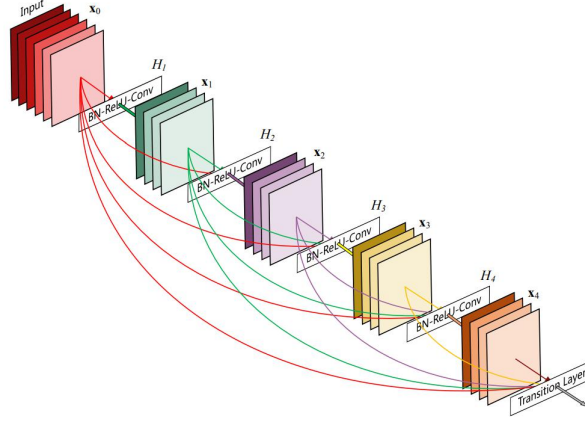


Figure 2.6: A DenseNet dense block consisting of the use of skipped concatenating connectivity to increase the number of features extracting in each layer [4].

use of bottleneck layers, introduced in [58, 59]. Bottleneck layers consist of applying 1×1 filters prior to 3×3 filters to decrease the number of parameters in the network as opposed to taking direct 3×3 filters on the input. In addition to the bottleneck layers, the number of feature maps between dense blocks were reduced. This meant that the intermediate, or transition, layers could output fewer feature maps to be fed into their next dense block. Through these techniques and the dense connectivity of the network design, a high performing model could be achieved with fewer parameters. This led to state-of-the-art performance surpassing the ResNet on datasets, such as ImageNet. Few networks were able to surpass DenseNet and of those that did many employed the use of large amounts of data augmentation, regularization, or an increased parameter space leading to higher resource utilization.

2.3 Random Weights in ANNs

In this section the use of fixed-random weights in ANNs is discussed. More specifically this section highlights the impact that the architecture has on the performance of a network with and without random weights. It additionally illustrates that comparable accuracies can be achieved in networks utilizing fixed-random weights on a number of tasks.

2.3.1 The Importance of Architecture Design

A number of works have explored the usage of fixed-random weights. In this section a brief overview is given for a few of these works. In [60], the authors show that a shallow CNN with random weights and normalization can achieve comparable performance to a trained system on the Caltech-101 dataset [61]. The shallow networks consisted of convolution layers with 64 filters in the first layer and 256 filters in the second layer, both with 9×9 kernel sizes, that are then passed to a linear classifier. The performance was found to be sub-par in comparison to a trained network when evaluated on a dataset with greater labeled samples, such as the NYU Object Recognition Benchmark (NORB) dataset [13]. However, this work illustrates how random weights still enable features to be extracted from the input to enable fairly high performance.

The authors in [62] delve deeper into the work of [60]. They highlight the fact that random weights only result in a slightly worse performance than trained weights and explore the reasoning for this. The authors find that CNNs with random weights in their convolution and pooling layers can be frequency selective and translation invariant. Through a comparison of various trained and random CNN architectures, it's found that there are cases where an architecture containing random weights can outperform a different architecture that has been trained. Therefore the authors conclude that the architecture alone plays an important role in key feature extraction from inputs and that the learning algorithm used in these networks may not necessarily need to be the focus in yielding a high performance model.

2.3.2 Random Weights in DNNs Achieving Comparable Performance

In [5], the authors illustrate the usage of fixed-random weights in image reconstruction, synthesizing textures, and utilizing neural style transfer [6]. They make use of a VGG-19 CNN as their chosen network and replace the maximum pooling layers of the

architecture with average pooling. All weights of the architectures were initially set to random values from a Gaussian distribution with zero mean and a standard deviation of 0.015. After random weights were initialized they stack the network layer by layer with new random weights in a greedy manner. In their new constructed network, they first sample images with an array of random weight combinations to find a set that minimizes the loss for one layer. They then repeat the process until the network leads to ideal random weights. A comparison of the reconstructions of images for the randomized VGG, denoted by ranVGG, and a fully trained VGG net on ImageNet, denoted by VGG, can be seen in Figure 2.7.

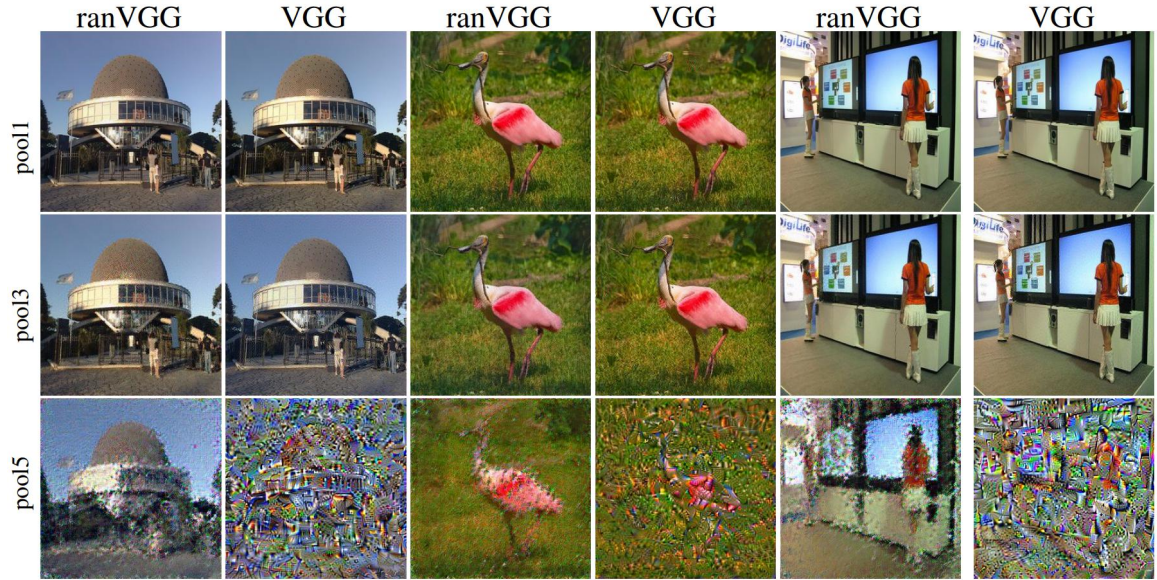


Figure 2.7: Reconstructions of 3 images when utilizing a fully random VGG net and a fully trained VGG net [5]. The visualization from the pool5 layer exhibits certain features learned in the trained net that inhibits reconstruction.

A clear difference can be seen perceptually between the fully random and trained VGG nets for the pool5 layer. The image reconstruction for the random network is shown to be more visually appealing. However, the authors take a greedy approach in constructing the ranVGG network. Additional experiments from the author show that textures can be generated as shown in Figure 2.8. Additionally, neural style transferred images can be generated as shown in Figure 2.9.

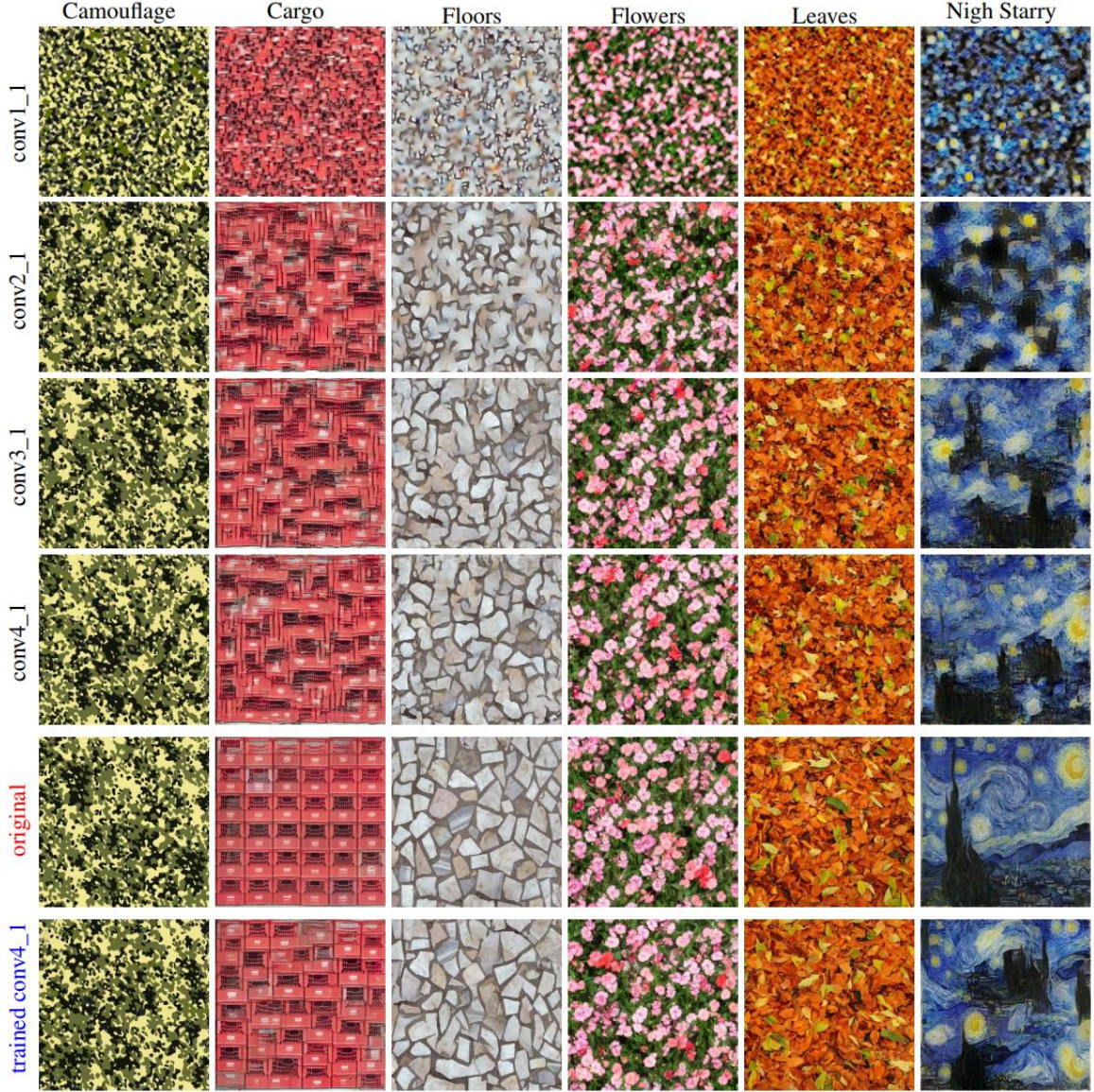


Figure 2.8: Generated textures from several images when utilizing a fully random VGG net and a fully trained VGG net [5]. In this figure each row corresponds to a convolutional layer in the random VGG net while a comparison is made between the 4th convolutional layer of both the trained and random nets.

A difference can be seen between both the images of the textures and the neural style transferred images. For example, in the texture images, Figure 2.8, the conv4_1 of the random VGG for the floors shows textures that are not as sharp as trained conv4_1. However, across the other textures, this is visually unnoticeable. Although the textures are not as distinct, in comparison to the trained net, the random net is



Figure 2.9: Neural style transfer of images when utilizing a fully random VGG net and a fully trained VGG net [5]. The first row depicts the original image, while the second and third rows show the random VGG net and trained VGG net’s content images with the applied style transfer. Note: The reference [13] in this image refers to [6]’s paper.

still able to extract textures from the given input.

This can be additionally seen in the neural style transferred images as the trained net depicts sharper more noticeable changes in the transferred content image for Der Schrei. However, this is not as visually noticeable when utilizing the other style images. These transferred content images are still able to obtain features from both style and content as the architecture enables for various features to be extracted from the input.

To explore the impacts of random weights even further, the authors in [7] experiment on training only portions of the weights in CNNs while leaving the untrained weights to their initialized random values. Many previous works explore the usage of layers containing random weights, but this work explores the use of layers containing a mixture of trained and random weights. They observe the effects of training convolutional layers with a percentage of trained filters while the rest are left untrained. Additionally, these experiments included tests on leaving only a value of k filters trained per convolutional layer. For these experiments the FC layers of each architecture consisted of weights that were trained. These experiments on both the Canadian

Institute for Advanced Research (CIFAR)-10 [8] and CIFAR-100 [8] datasets can be seen in Figure 2.10.

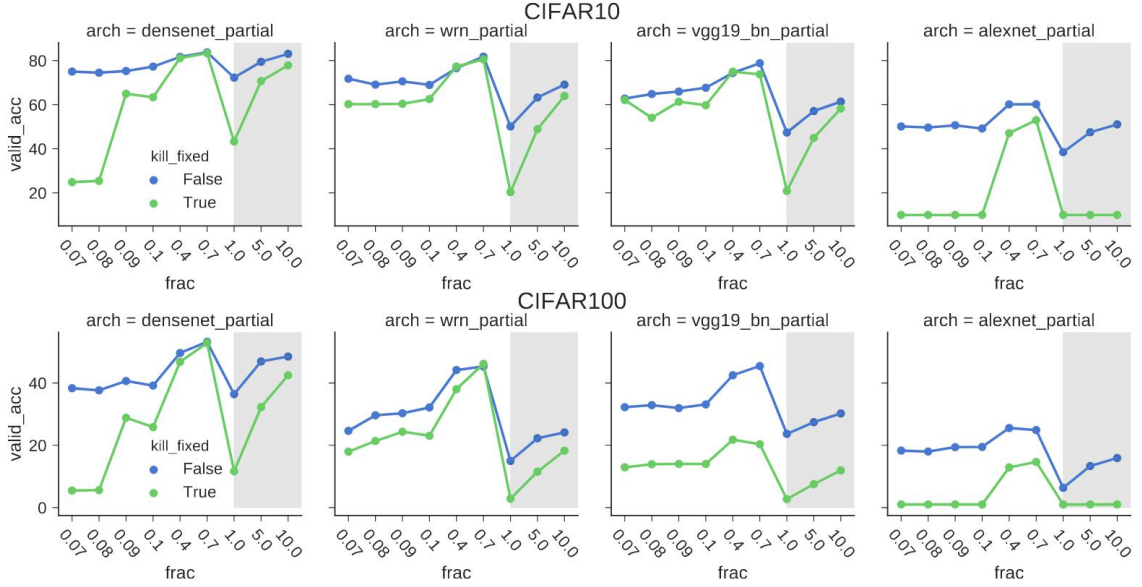


Figure 2.10: Experiments utilizing a percentage and a k amount of trained weights in deep CNN architectures. The white area indicates the percentage of weights trained, while the gray area indicates training on only k number of filters per convolutional layer. The blue line indicates the rest of the weights in the layer were left untrained, while the green line indicates the rest of the weights in the layer were zeroed out [7].

The architectures used in this experiment, from left to right in Figure 2.10, include DenseNet with a depth of 100 and a growth rate of 12, Wide ResNet (WRN) [63] with 28 layers and a widen factor of 4, VGG-19, and AlexNet. The experiments show that each architecture was able to achieve fairly high performance even with a large portion of weights left random. Additionally, across many of the architectures the random weights a small increase in performance over leaving the weights zeroed out. It is also seen that even learning a subset of k filters while leaving the rest untrained can lead to fairly high performance most notably in the DenseNet architecture.

Unfortunately, in this experiment the authors fail to illustrate the accuracy achieved by a fully trained model after the 10 epochs of training. It's also unclear if a greater number of training epochs would have substantial effects on the accuracies achieved by these networks with random weights. Fortunately, the authors conduct an experi-

ment where they train the WRN model and DenseNet model for 200 and 300 epochs, respectively, as shown in Table 2.1.

Table 2.1: Performance achieved when learning only a fraction of filters for the CIFAR-10 dataset for training epochs of 200 on WRN and 300 on densenets. Eff. Params indicates the number of trained parameters, while * indicates the performance when the parameters for random weights were zeroed out [7].

Method	Fraction	Eff. Params x 106	Perf	Perf*
WRN	0.1	3.66	94.12	91.53
WRN	0.4	14.6	95.75	95.49
densenets	0.1	0.09	88.73	82.11
densenets	0.4	0.3	93.33	92.46

In this experiment the WRN and DenseNet models were compared with small fractions of trained convolutional weights and large fractions of random or zeroed out convolutional weights on the CIFAR-10 dataset. It can be seen that the random weights are more influential when a small fraction of weights (0.1) is trained. However, the experiments of zeroing out the weights for the 0.4 trained fraction amount indicates that the random weights only assist the performance marginally ($<1\%$). Another experiment comparing the two networks for 200 and 300 epochs of training, respectively, on the CIFAR-10 and CIFAR-100 datasets can be seen in Figure 2.11.

This figure illustrates how even learning 70% of the weights is a substantial amount in enabling for a model that reaches performance comparable to its respective trained equivalent for both the CIFAR-10 and CIFAR-100 datasets. The authors evaluate on one more dataset, namely the Tiny-ImageNet dataset [64]. This dataset is an alternative to the ImageNet dataset and consists of 200 classes with 500 training and 50 validation images for each class. Each image is of RGB color and size 64×64 . The experiments conducted on this dataset consisted of training only the WRN model for 45 epochs with an initial learning rate of 0.1 which was decreased by a factor of 10 every 15 epochs. The result of this experiment is shown in Table 2.2.

It can be seen that even if 40% or 70% of the weights are used to train the network

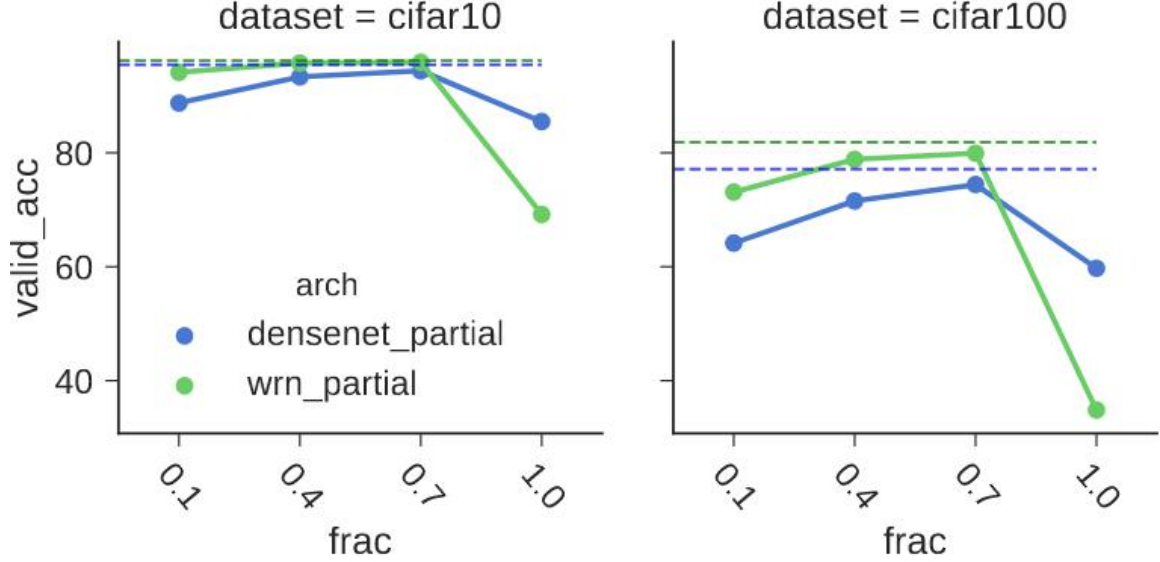


Figure 2.11: The WRN and DenseNet networks compared on both the CIFAR-10 (left) and CIFAR-100 (right) datasets at various training fractions per convolutional layer. Green and blue solid lines indicate networks with a fraction of trained weights and the rest left random. Green and blue dashed lines indicate the performance of the WRN and DenseNet networks with trained weights. Note: when $\text{frac} = 1.0$, this is actually when only a single filter is trained per convolutional layer [7].

Table 2.2: Performance on the Tiny-ImageNet dataset using the WRN model with various percentages of trained weights [7].

% Params	Top-1 Accuracy (%)	# Params
10	21.75	.83M
40	30.13	2.58M
70	33.22	4.33M
100	35.54	6.1M

while all other weights are left random high accuracies can be achieved. This work illustrates how much of an impact the architecture can have on the performance of deep CNNs as initially discussed in [62]’s work.

Overall, the research into the field of fixed-random weights in ANNs illustrates how these networks may not necessarily need to be fully tuned. However, the architectural design of the ANN plays an important role in extracting meaningful features from the input. Additionally, a purely random network can yield comparable performance in some cases to fully tuned networks. This brings into question, though, of how many

parameters need to truly be tuned in a network? It's not necessarily true that all parameters need to be tuned as a purely random network illustrates the impact of the architecture. On the other hand, if only portions of a network are tuned then a network can still reach comparable performance to its respective trained model.

2.4 Random Projection Networks

Aside from the works discussed above, many other works highlight the use of random weights in ANNs. In the work of [40], the extreme learning machine (ELM) architecture is introduced. This architecture is a feed-forward architecture, similar in design to a single hidden layer MLP, that makes use of random projection from its inputs to a hidden layer and conducts training on the output layer weights. A similar concept is utilized in the form of recurrent ANNs. This concept, better known as reservoir computing, is explored in two works by [41] and [42] through the introduction of the Echo State Network (ESN) and Liquid State Machine (LSM), respectively. In both works the input to hidden layer weights are left as a random projection along with a reservoir layer containing recurrent neurons with random weights and connectivity to other neurons. Through the random nature of the reservoir layer, feedback signals are introduced from temporal inputs enabling for an innate form of fading memory. Similarly to the ELM, the ESNs and LSMs are trained only on their output layer weights.

2.4.1 ELM

The ELM approach to fixed-random weights has become increasingly adopted in the literature [65]. The ELM architecture is a shallow feedforward ANN that makes use of two fully connected (FC) layers, similar to that of the structure of an MLP illustrated previously in Figure 2.2. In this architecture, the input to hidden layer weights are kept fixed and random. This allows for a random projection from the input space to

a larger more separable space as the hidden layer nodes in the network are typically greater than the number of input nodes. The hidden to output layer weights are then trained using an alternative training paradigm to SGD, namely taking the MPP [39] of the output weights given all training samples. By taking the inverse of a matrix a least squares solution can be calculated. However, the matrix must be square to take the inverse. Unfortunately, it is rare for this to occur given most data. Therefore, by using the MPP, a least squares approximation of the output weights can be computed by taking the generalized inverse of the matrix. The evaluation of the output layer's weights in an ELM is calculated using (2.10), (2.11), and (2.12).

$$\mathbf{H}^\dagger = (\mathbf{H}^T \mathbf{H} + \lambda \mathbb{I})^{-1} \mathbf{H}^T \quad (2.10)$$

$$\mathbf{W}_{\text{out}} = \mathbf{H}^\dagger \mathbf{y} \quad (2.11)$$

$$\hat{\mathbf{y}} = \mathbf{H} \mathbf{W}_{\text{out}} \quad (2.12)$$

Initially the MPP of \mathbf{H} is taken using (2.10), where \mathbf{H}^\dagger is the MPP of \mathbf{H} , \mathbf{H} is the output of the hidden layer, and λ acts as a regularization term multiplied by an identity matrix, \mathbb{I} . Equation (2.11) then highlights the calculation of the least squares approximation. In this equation the output weight matrix, \mathbf{W}_{out} , is calculated by multiplying \mathbf{H}^\dagger by \mathbf{y} , which is the ground truth labels. Once the output weights are calculated, the output predictions, $\hat{\mathbf{y}}$, are calculated in (2.12) by multiplying the output of the hidden layer and the output weights. The output predictions are then used to compare to the ground truth labels to evaluate the network's performance.

2.4.2 Random Vector Functional Link Network

Prior to the introduction and popularization of the ELM, there was another network that introduced random projection, namely the Random Vector Functional Link Neural Network (RVFL) [46]. In this network, hidden layer nodes were called enhancement nodes. The weighted connection between the input and enhancement nodes were set to fixed-random values during the training process while the output weights were trained. The training paradigms used in this network would consist of either a gradient based approach or training in one step through the utilization of the MPP. Many similarities can be seen between the ELM and this network, however, this network makes use of an additional link, or skipped connection, from its input to its output. This skipped connection was a concatenation of the input-to-output weights to the enhancement-to-output weights. Therefore, in the calculation of the MPP of the output layer's weights of the network, the \mathbf{H} matrix of (2.10) would consist of the additional initial inputs. Through the use of this skipped connection, the RVFL was shown to be able to approximate functions well in comparison to a normal MLP.

An in depth evaluation of the skipped connection was studied in [47] for this network. The authors tested this connection on 121 datasets from the UCI machine learning repository [66] to assess the direct input-to-output connection. They found that the connection led to an increase in accuracy for the datasets when compared to an RVFL lacking this connection. They concluded that this link acted as a regularizer for the randomized weights and enhanced the performance of the network.

The authors in [48] introduce the Convolutional Random Vector Functional Link Neural Network (CRVFL), which is a combination of the concepts used in CNN architectures and the RVFL architecture. This architecture made use of a convolutional layer, an average pooling layer, a normalization layer, and a FC layer, as well as the input-to-output skipped connection, to perform visual tracking. Similarly to the RVFL, the CRVFL left the weighted connections in the intermediate layers fixed and

random. Training only occurred on the output layer and the training paradigm consisted of a recursive least squares approach as visual tracking was a task that occurred over time as opposed to image classification which is purely spatial. Using this architecture, the authors showed results on a 51 video sequence of tracking objects and showed comparable performance to CNN backpropagation methods.

2.5 Tensor Decomposition

In many ANNs today the flow of data is typically represented in terms of vectors or matrices. For example, an MLP with a single hidden layer of H neurons has weighted connections from its input to a successive layer. The H neurons are represented in the form of a vector, while the weighted connections are represented in the form of a matrix. In mathematics there is another method of representing these matrices through the use of tensors. Tensors are multidimensional arrays where each dimension is a vector space [67]. The dimension of a tensor goes by many names, such as its order, number of ways, or number of modes [67]. These terms are used interchangeably in the literature, however, clarification is given to ensure readability of this work. A clear example of the dimensionality of tensors can be seen in Figure 2.12.

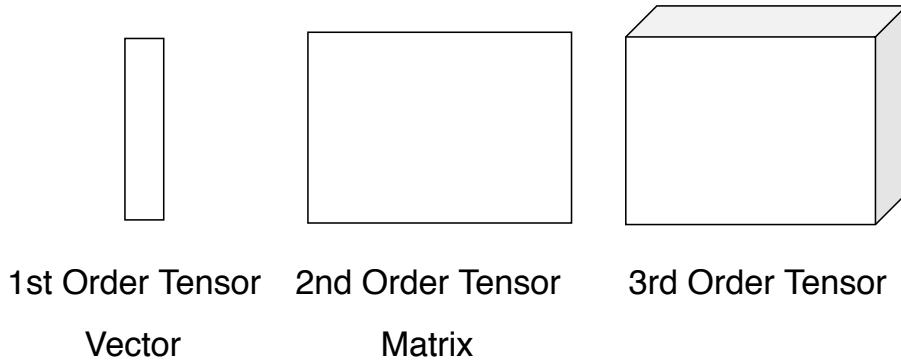


Figure 2.12: Illustration of tensors in varying orders, or dimensions.

In this figure tensors in their respective dimensions are shown. A tensor of $d = 1$ is a vector or 1st order tensor, a tensor of $d = 2$ is a matrix or 2nd order tensor, a

tensor of $d = 3$ is a 3rd order tensor, and lastly a tensor of $d = N$ is an N th order tensor. Additionally, a tensor of $d = 0$ is merely a scalar. In this work, tensors of order 3 or higher are denoted with bold upper-case letters, such as \mathbf{X} , as opposed to matrices denoted by italicized upper-case letters, such as X . Vectors are denoted by lower case scripted letters, such as a .

Decomposing data, or compressing it, to a smaller or approximate form is beneficial as it decreases the dimensionality of the data. Many applications, such as signal processing [68], image compression [69], machine learning [70], etc., require the use of compression techniques to decrease the number of resources used. Therefore, the tensor decomposition has become popularized as it presents effective methods in decomposing high dimensional data [67].

In order to understand tensor decompositions, it's important to first understand matrix decomposition. In matrix decomposition problems, an initial matrix X is approximated by a low-rank model M . M is calculated by taking the sum of a subset of rank-one matrices as shown in (2.13).

$$X \approx M = \sum_{l=1}^r a_l \otimes b_l = AB^T = [AB] \quad (2.13)$$

The outer product, denoted by \otimes , of vectors a and b is taken with l as the index into the vectors across all r , which is the total number of ranks. This is equivalent to AB^T , where B^T is the transpose of B .

Under tensor representations, an N th order rank-one tensor can be represented as the outer products of its respective 1st order tensors, or vectors. Rank-one tensors refer to an N th order tensor that can be decomposed into the outer product of N 1st order tensors. For example, suppose we have a rank-one tensor of order 3, $\mathbf{T} \in \mathbb{R}^{m \times n \times p}$, this tensor can be represented by the outer product of its 1st order

tensors, $a \in \mathbb{R}^m$, $b \in \mathbb{R}^n$, and $c \in \mathbb{R}^p$, as shown in (2.14).

$$\mathbf{T} = a \otimes b \otimes c \quad (2.14)$$

Aside from the example of a rank-one tensor, the rank of a tensor is defined as the summation of the minimum number of rank-one tensors whose summation produce the original tensor [71]. However, determining the rank required to produce an original tensor is found to be an NP-hard problem [72, 73]. Although, determining the rank is an NP-hard problem, the use of tensor decomposition still holds benefits over matrix decomposition as it extends to higher orders.

To gain an understanding of tensor decomposition, one of the more well known and original tensor decomposition is the canonical polyadic (CP) decomposition [74]; also known as the CANDECOMP/PARAFAC decomposition [75, 76]. In this method of decomposing a tensor, a tensor is approximated by respective rank-one tensors. Figure 2.13 illustrates an example of CP decomposition on a 3rd order tensor, \mathbf{T} , which is approximated by rank-one tensors, a , b , c . Equation (2.15) presents the calculation for an N th order tensor.

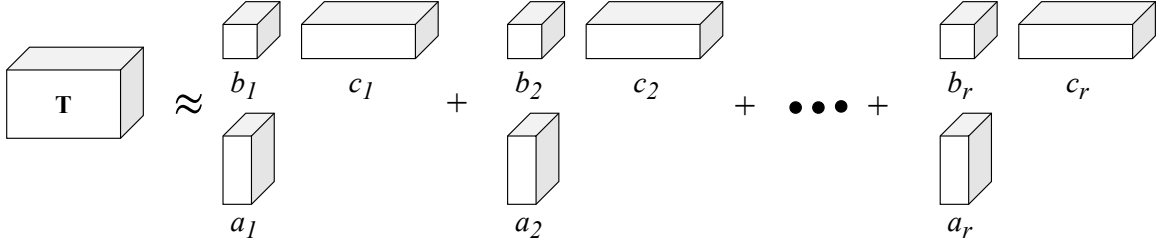


Figure 2.13: Illustration of a tensor decomposition in which an input \mathbf{T} is approximated by the summation of rank-one tensors. These rank-one tensors are calculated from the outer product of 1st order tensors a , b , and c .

$$\mathbf{T} \approx \sum_{l=1}^r a_l \otimes b_l \otimes c_l = [ABC] \quad (2.15)$$

In this decomposition, the outer product of a single instance of $a_l \otimes b_l \otimes c_l$ is called

a component and the matrices $[ABC]$ are referred to as factor matrices that describe the 3rd tensor \mathbf{T} . By summing the rank-one tensors across r ranks the original tensor \mathbf{T} can be approximated. In contrast to matrix decomposition, tensor decomposition holds the benefit of enabling for less rigidity in unique solutions. The matrix decomposition to approximate a low-rank model typically doesn't hold a unique solution unless additional constraints are made to the matrices. For example, various matrices A and B can enable for an approximate model M to be formed [77]. However, tensor decomposition can approximate a low-rank model with less rigid constraints as deterministic approaches can be made to either compute or enable uniqueness in the decomposition [71]. As the mathematical proofs for this uniqueness is out of the scope of this work, the readers are referred to [71] for additional information on this topic. Additionally, a survey on tensor decomposition techniques is discussed in detail in [67].

A number of tensor decomposition techniques exist in the literature [74, 78, 79, 80], however, the most commonly used techniques consist of CP and Tucker decomposition [74, 78]. The CP decomposition was illustrated above through a tensor, \mathbf{T} , being approximated by rank-one tensors. While the Tucker decomposition is illustrated in Figure 2.14 and calculated in (2.16).

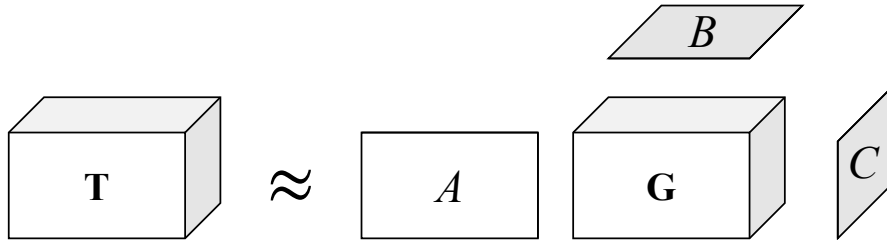


Figure 2.14: Illustration of the Tucker decomposition in which an input \mathbf{T} is approximated into a compressed core \mathbf{G} . This compressed core is then multiplied by the matrices representing rank-one tensors; A , B , and C .

$$\mathbf{T} \approx \mathbf{G} \times_1 A \times_2 B \times_3 C = \sum_{p=1}^P \sum_{q=1}^Q \sum_{r=1}^R g_{pqr} a_p \otimes b_q \otimes c_r = [\mathbf{G}; ABC] \quad (2.16)$$

The Tucker decomposition decomposes an original tensor, \mathbf{T} into a core tensor, \mathbf{G} , which is multiplied by factor matrices, A , B , and C , representing the rank-one tensors. The original tensor's dimensionality is of $\mathbf{T} \in \mathbb{R}^{I \times J \times K}$, the core tensor is of dimensionality, $\mathbf{G} \in \mathbb{R}^{P \times Q \times R}$, and lastly the factor matrices are of dimensionality, $A \in \mathbb{R}^{I \times P}$, $B \in \mathbb{R}^{J \times Q}$, and $C \in \mathbb{R}^{K \times R}$. The Tucker decomposition is considered a higher-order Principle Component Analysis (PCA) where the factor matrices are the principle components, while the core tensor represents the intractability between each component [67]. When the components, or number of columns, P , Q , and R , are less than I , J , and K , then the core tensor \mathbf{G} is considered to be a compressed form of the original tensor \mathbf{T} [67, 81].

2.5.1 Tensor-Train Decomposition

Although both the CP and Tucker decomposition enable for a decomposed form of their input tensors, they have drawbacks. As previously discussed above, the calculation for the ranks of tensors is NP-hard [67, 82]. Additionally, approximation methods for the ranks can fall into local minima and therefore don't always converge to a global minimum [82]. Although stable, the parameter space for Tucker decomposition leads to an exponential increase relative to the order of the original tensor and is therefore preferred for smaller dimensionalities [82].

In this work, the Tensor-Train (TT) decomposition [82] is used as it offers the benefits of stability and a decreased number of parameters. TT decomposition is a tensor decomposition method that makes use of auxiliary unfolded matrices to approximate the original tensor. A tensor, \mathbf{T} , of N th order can be approximated by

N unfolded matrices, G . These unfolded matrices can be represented in the form of 3rd order tensors of dimensionality $r_{k-1} \times n_k \times r_k$, where n refers to a single dimension in the total number of dimensions N and k is an indexing variable relative to a max of N . The first and last tensors would consist of r_0 and r_N and are equal to the value of 1 as boundary conditions [82]. An illustration of the TT format, or the decomposed tensor format, of a 5th order Tensor can be seen in Figure 2.15 and the calculations for the case of an N th order tensor can be seen in (2.17).

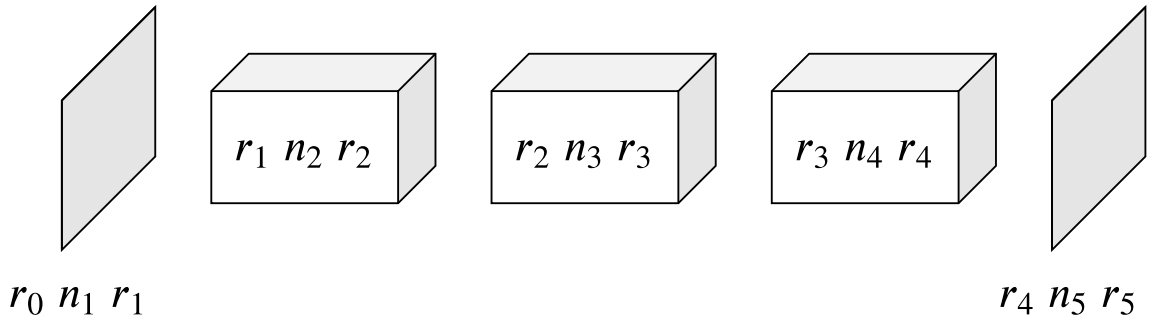


Figure 2.15: Illustration of a 5th order tensor decomposition into respective cores of the TT format.

$$\mathbf{T} \approx \sum_{k=1}^N \mathbf{G}_{r_0, n_1, r_1} \mathbf{G}_{r_1, n_2, r_2} \cdots \mathbf{G}_{r_{N-1}, n_N, r_N} \quad (2.17)$$

In this figure, a 5th order tensor is decomposed into 5 tensors of varying dimensions. When decomposed, these tensors are referred to as cores. The 2 cores at the beginning of the TT format consist of the dimensionality cases of $r_0 = r_N = 1$, leading to 2nd order tensors for the first and last tensors, while the central cores consist of 3rd order tensors. Each core is summed up across all dimensions N for each separate element to approximate the original tensor \mathbf{T} . These cores, when unfolded, are referred to as unfolded matrices, which, as discussed above, are represented as 3rd order tensors. To best approximate the original tensor the following condition in

(2.18) must be satisfied.

$$\|\mathbf{T} - \mathbf{M}\|_F \leq \epsilon \|\mathbf{M}\|_F \quad (2.18)$$

Where \mathbf{M} is the approximation of the original tensor \mathbf{T} , F is the Frobenius norm and ϵ is the target error desired for the approximation. To best approximate the model, an ϵ of 0% would be desired, however, a more compressed tensor can be acquired with lower ranks. The authors in [82] present an empirical method to determine the ranks using the singular value decomposition, however, this results in a costly operation as this empirical method requires an iterative method to determine the ranks across the dimensionality of the tensor [82]. Once the ranks are determined, the core dimensions are acquired as the ranks, r_0 to r_N , to determine the dimensionality of each core given the input tensors dimensionalities, n_k .

Chapter 3

Methodology

In this chapter, a hypothesis is initially discussed to illustrate the main goals of the experiments performed in this work. An explanation of the experiments is then conducted as well as an intuitive analysis of why these specific experiments were executed.

3.1 Hypothesis

As discussed in Chapter 2, previous research has highlighted how ANNs utilizing fixed-random weights are able to extract rich features from their inputs relative to their architecture and their training paradigm. Therefore, in this work, it is hypothesized that fixed-random weights can reduce the demand on computational resources while retaining performance akin to their trained network counterparts. This is due to fewer operations occurring in the backwards pass, or backpropagation, through the network. Only trained weights would need to have their weights updated, while random weights would retain their values. Additionally, in this work, it is hypothesized that the employment of fixed-random weights enable for networks to learn supplemental features to increase performance over downsized DNNs. This is hypothesized as the parameter space of the original network is retained leading to a greater feature extraction across each layer. To clarify, downsized models refer to fully trained smaller versions of the original architecture. For example, if a trained

model had a convolution layer with 16 filters and the trained fractional amount was 50%, then in the random case there would be 8 filters trained and 8 filters would be left with initialized fixed-random weights while in the downsized case there would be just 8 filters trained. The downsized case is meant to further highlight that the fixed-random weights can still extract supplemental features from the input leading to high performance along with a decrease in training time.

3.2 Evaluation

To assess each network and their performances, a number of image classification datasets are employed. The first dataset employed is the CIFAR-10 dataset, which consists of 32×32 RGB images of various classes. These classes included airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks as illustrated in Figure 3.1. The dataset includes 60,000 images, where 50,000 images are used for training, while 10,000 images are used for testing. The images are normalized with means 0.4914, 0.4822, 0.4465 and standard deviations of 0.247, 0.243, 0.261. Additional data augmentations of random horizontal flips and a random crop with padding of 4 are used. This dataset adds a layer of complexity as the images are RGB rather than grayscale. Therefore, this dataset is utilized to evaluate fixed-random weights with more complex data.

In addition to the CIFAR-10 dataset, the Street View House Numbers (SVHN) dataset [9] is used as an additional test to the CIFAR-10 dataset. This dataset consists of 32×32 RGB real-world imagery, shown in Figure 3.2, that is similar to the Mixed National Institute of Standards and Technology (MNIST) dataset [83] in which digits are classified with labels from 0 to 9. However, these images are of digits in a natural scenery, such as on streets or on other structures. Additionally, the RGB channels in the dataset increase the difficulty in contrast to the MNIST dataset. The dataset is split such that 73,257 images are used for training and 26,032 images are used for

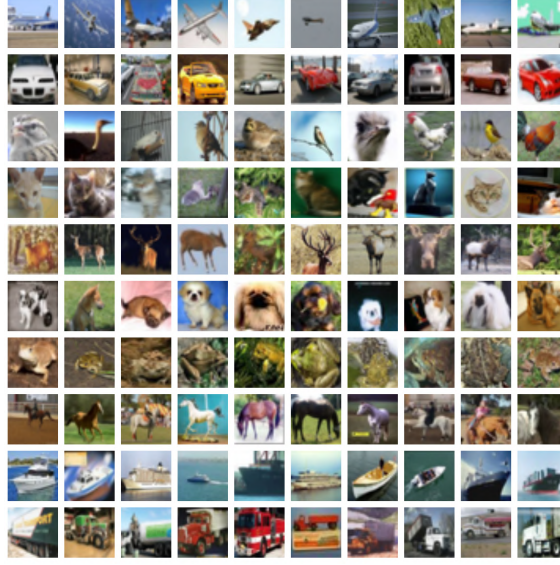


Figure 3.1: The CIFAR-10 dataset containing various classes ranging from airplanes to trucks [8].

testing. For this dataset, the images are normalized with means of 0.5, 0.5, 0.5 and standard deviations of 0.5, 0.5, 0.5.



Figure 3.2: The SVHN dataset containing digits 0 through 9 in a similar manner to MNIST, but in natural scenery [9].

The next dataset utilized is the UC Merced Land Use (UCM) dataset [10]. This dataset consists of 256×256 RGB aerial imagery, as seen in Figure 3.3. The 21 classes of the dataset range from agriculture and beach to airplane and buildings. The dataset is made up of 2,100 images of 21 classes, which are split into 80% training, or 1,680 training samples, and 20% testing, or 420 testing samples. The data is

normalized with means of 0.48678258, 0.49174392, 0.45313206 and standard deviations of 0.21827406, 0.20301312, 0.19686365.

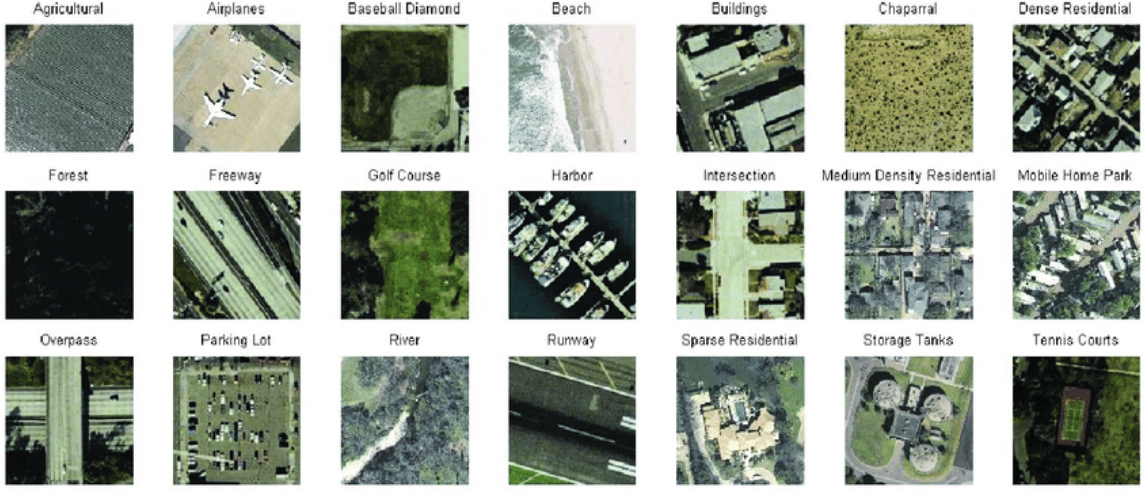


Figure 3.3: The UCM dataset containing aerial imagery of 21 classes [10, 11].

The last three datasets used are the Moving and Stationary Target Acquisition and Recognition (MSTAR) dataset [84], small NORB dataset, and the Fashion-MNIST (FMNIST) dataset [14]. The MSTAR dataset contains 6,894 128×128 grayscale synthetic aperture radar (SAR) images of vehicles captured at varying angles. Specifically, in this work, the vehicles are at 15 and 17-degree depression angles. The dataset is split into randomly shuffled 80% training and 20% testing set splits, leading to 5,499 images for training and 1,395 images for testing. The images are downsampled to 28×28 and normalized with a mean of 0.14816141 and a standard deviation of 0.14638391. The dataset consists of 10 unique vehicle classes as shown in Figure 3.4. From this figure it can be seen that this dataset offers noisy images in comparison to other cleaner datasets, such as MNIST.

The small NORB dataset consists of 96×96 grayscale images of 5 classes. The classes are made up of four-legged animals, human figures, airplanes, trucks, and cars at various lighting conditions and elevations as presented in Figure 3.5. The dataset consists of 24,300 image pairs which can be split into training and test splits of varying sizes. For these experiments, the images are downsampled to 28×28 and normalized

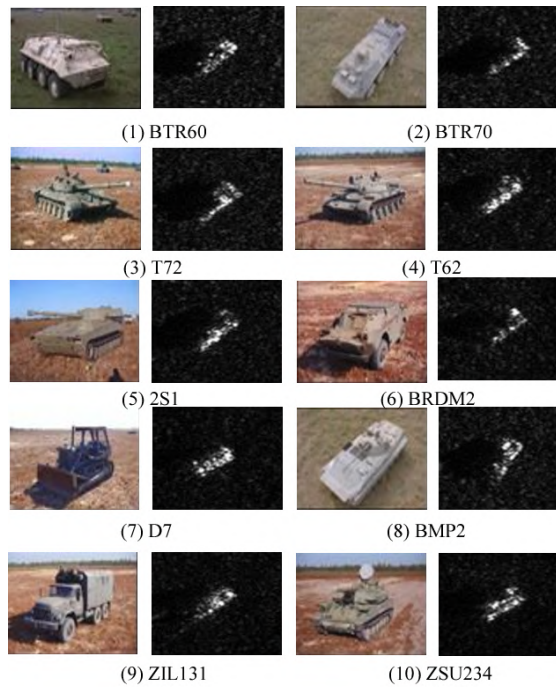


Figure 3.4: The MSTAR dataset containing 10 unique vehicle classes [12].

with a mean of 0.5 and a standard deviation of 0.5. The train split is 80% while the test split is 20%.



Figure 3.5: The small NORB dataset containing 5 classes ranging from four-legged animals to cars [13].

The FMNIST dataset is meant to be a drop in replacement for the MNIST dataset

as researchers were able to obtain near 100% accuracy on the MNIST dataset [85]. Therefore, to add a layer of complexity the authors in [14] downscaled images of fashion items and set their color channels from RGB to grayscale. The dataset consists of 10 fashion items at grayscale image sizes of 28×28 as depicted in Figure 3.6. Similarly to MNIST, 60,000 images are used for training and 10,000 images are used for testing. Due to the nature of the classes and their features, such as pullover and coat, the dataset was shown to be more difficult when compared to the MNIST dataset. For these experiments, the images are normalized with a mean of 0.5 and a standard deviation of 0.5.

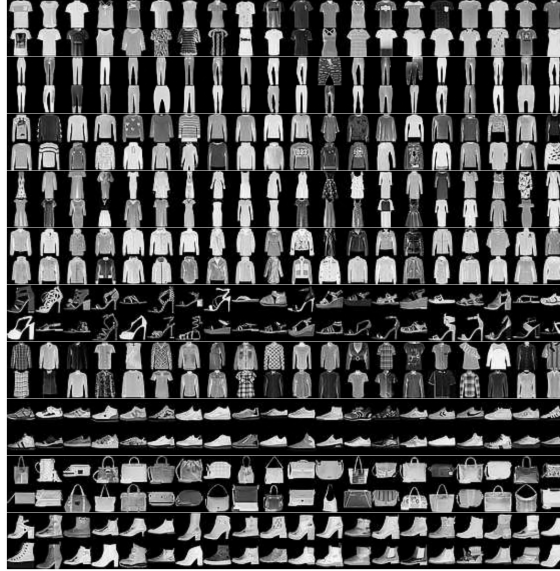


Figure 3.6: The FMNIST dataset containing 10 classes of fashion items ranging from pullovers to boots [14].

3.3 Semi-Random CNNs

To assess the hypotheses stated above, a number of experiments are conducted. The first set of experiments consists of utilizing fixed-random weights in the convolutional layer of deep CNNs, similarly to [7]’s work. These networks are denoted as Semi-Random CNNs. In these experiments 2 deep CNNs are looked at, namely ResNet-

20, DenseNet-BC-100. The ResNet and DenseNet architectures are chosen as they illustrate widely used deep architectures that make use of skipped connectivity to enhance performance over networks lacking this skipped connectivity. The use of fixed-random weights in these architectures allow for a comparison of how effective the different skipped connectivities, summation or concatenation, perform.

In these experiments, each convolutional layer of the selected architecture would consist of a fraction of trained filters vs fixed-random or omitted filters. The other layers in the network are left trained as these architectures primarily make use of convolutional layers meaning that these layers consist of a large percentage of the total number of parameters. These fractional amounts would consist of utilizing full random projection, various quarter fractions of trained filters, as well as a linear increase in trained weights. For each architecture, their initialized random distributions are left the same and each layer's weights would be fixed-randomly depending on the specified percentage of trained weights. To clarify, in the instances of the quarter of trained filters, let's assume we had the example of 64 filters in a convolutional layer. If 75% of the filters are set to trained, then 48 filters would be trained, while 16 filters are either set to fixed-random or would be omitted during the training process. If the filters are left fixed-random then we have a fixed-random version of the original architecture, while if the filters were omitted then we have a shrunken, or downsized architecture. Thus in total, for that layer there would be either 64 filters in total or 48 filters in total, but in both cases only 48 filters would be trained.

If the fractional amount is set to 'Linear', then the network would initially contain random filters in its first convolutional layers and as the network's deeper layers were reached, the number of random filters in the deeper layers would be decreased. To clarify, if the architecture contains 20 convolutional layers, then the first convolutional layer would have only fixed-random filters, while the second convolutional layer would have 5% of the number of filters set to trained. This pattern would continue until the

last convolutional layer, where the last convolutional layer would contain all trained filters in its layer as opposed to its predecessor layers containing random filters. This methodology was adopted from the use of random projection in networks, such as the ELM. The intuition behind this idea of random projection in the initial layers of a network extracting smaller features while the trained weights in the deeper layers would consist of more finely tuned features. An additional experiment that is explored in these random deep CNNs is training them for a greater number of epochs in comparison to their trained equivalent models. In [7]’s work, the authors only show training for a set number of epochs and only compare to the trained equivalent models for 1 set of experiments. This test is to see if a partially random model can achieve the same or better performance to its trained equivalent model after a longer period of training.

The last experiment conducted on these Semi-Random DNNs is a transfer learning experiment. In this experiment, a ResNet-50 model is used for transfer learning on the UCM dataset. The ResNet-50 model is chosen to show a comparison with the work of [16]. In this work, the authors transfer learn using a ResNet-50 model on the UCM dataset by fine-tuning the network and applying data augmentation to the original dataset. The data augmentation included 7° rotations as well as transposes of the images. In this work, similar data augmentations are used, except instead of transposing the images, horizontal and vertical flips are applied as they are similar in nature to transposing the image. In this experiment, the ResNet-50 model is transfer learned and has quarter fractions of trained filters to assess how partial fine-tuning of the network performs against fine-tuning the entire network.

These experiments are aiming to explore how random weights can alleviate the demand on resources and training times while still retaining high performance in models. Additional statistics are gathered apart from the accuracies achieved from these models, such as an approximation of the number of Giga-FLOPs (GFLOPs), as

well as the memory to store the weights of each model, to assess if random weights can alleviate these resource concerns. The time to train these models wasn't used as this can be variable to optimizations in programs and the machine that is running these programs. However, by measuring the number of FLOPs, a more accurate analysis of training time can be achieved. In addition to the following measurements, the filters of the initial layers of these networks are visualized to assess if the fixed-random filters capture any meaningful features.

To calculate the number of FLOPs, the forward and backward passes of these networks needs to be calculated. In the work of [15], the authors illustrate how the number of FLOPs can be calculated relative to the forward pass of the network. The authors present the diagram shown in Figure 3.7.

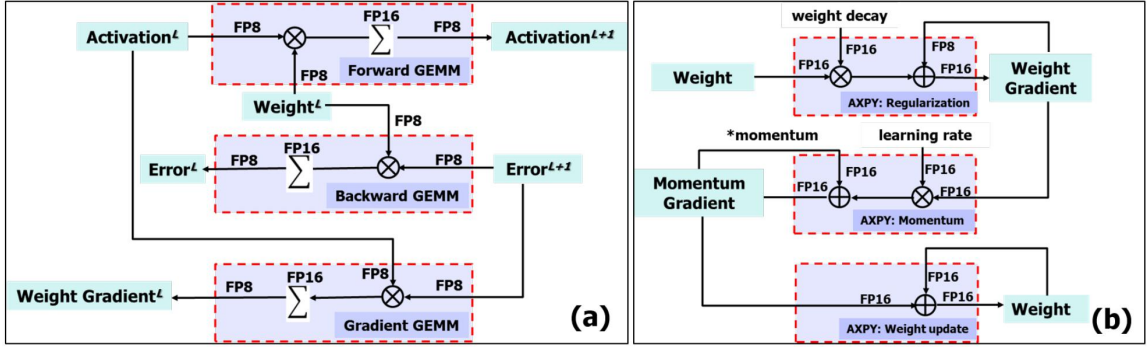


Figure 3.7: Diagram depicting the operations required for SGD. (a) illustrates the forward pass as well as the error calculation and gradient calculation for the backward pass. (b) illustrates the weight update as well as calculations for regularization and momentum [15].

In this figure it can be seen that the forward pass's operations are effectively equivalent to the error and gradient calculation. To clarify, a summation and multiplication are taken in each unit leading to a near equivalency in number of operations. Therefore, the calculations of the backward pass of SGD trained networks can be approximated to 3 times the forward pass's number of FLOPs, as shown in (3.1), if all weights in the network are trained; ignoring the additional weight update, regularization, and momentum calculations. In this equation, \mathbf{F} simply denotes the number

of FLOPs in the forward pass.

$$\text{FLOPs} = 3F \tag{3.1}$$

Similarly, to calculate the number of FLOPs in networks using fixed-random weights, the calculation would be equivalent. This is due to the fact that the network’s gradient update in the backwards pass would still require the need to look at all weight values to calculate the trained weights of the network. Even if a subset of the weights were left fixed and random, to propagate back to a layer with any trained weights would require the entire backwards pass. Therefore, the backward pass cannot be approximated with fewer operations in the cases of fixed-random weights. The only exception to this would be in the random projection cases, but only if all previous layers to the final trained layer are left fixed and random. Although a minimal effect, the use of fixed-random weights does alleviate the memory writing resources of these networks.

3.4 Extensions to the ELM

The second set of experiments consist of extending the ELM architecture. As the ELM is limited to a single hidden layer, the architecture is extended on with additional layers, such as convolution, pooling, normalization, and fully connected layers. However, with each additional layer, all weights are left randomly fixed and training occurs only on the output layer weights using the MPP, similar to the ELM’s training. It’s believed that through the addition of these layers a higher accuracy model can be achieved with insignificant additions to training time.

3.4.1 Convolutional ELM

The first architecture explored is the Convolutional ELM (CELM), which consists of a single convolutional layer as a feature extractor and 2 FC layers, as depicted in Fig. 3.8. In addition to the convolutional layer, an input-to-output connection is adopted, in which the input is appended onto the output of the last FC layer. This connection is taken as inspiration from both DenseNet and the RVFL networks. Typically a skipped connection like the following is utilized in DNNs as information is lost across the deeper layers of these architectures [3, 4]. In this work, the skipped connection is explored to evaluate if it is effective in a smaller architecture that makes use of the MPP training paradigm.

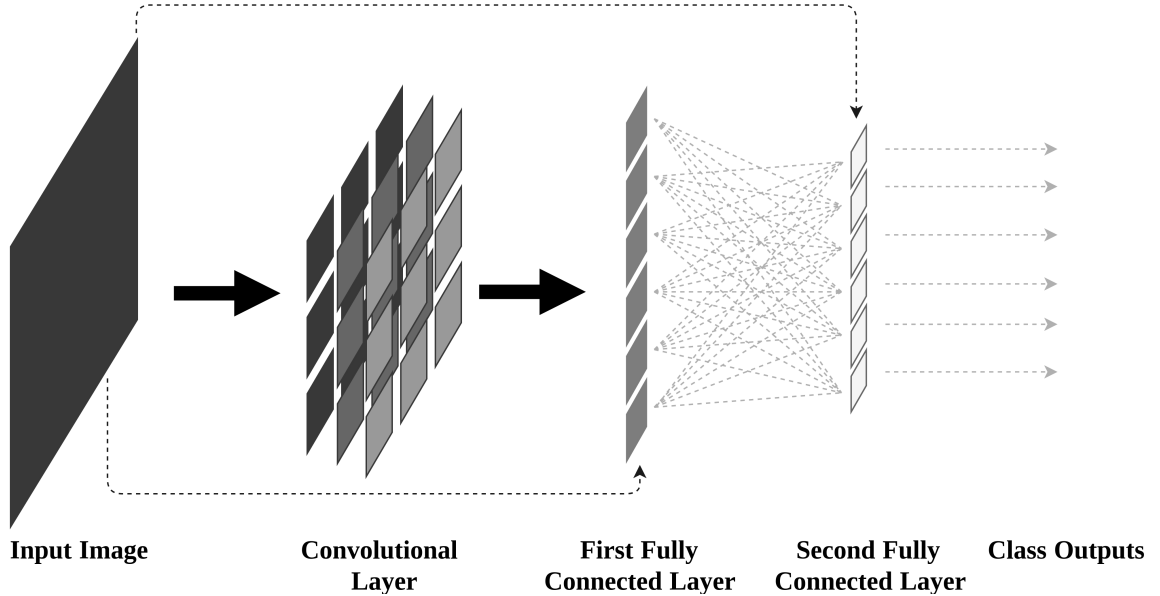


Figure 3.8: Illustration of the CELM architecture which makes use of an initial convolution layer as a feature extractor and then 2 FC layers. Skipped connections are utilized to observe their effectiveness in the possibility of increasing performance.

To observe the effects of the skipped connectivity even further with this architecture, the skipped connection is adopted in 3 different methodologies. A skipped connection from the input to the first FC layer (Skip 1), a skipped connection from the input to the second FC layer (Skip 2), and lastly a skipped connection from the

input to the first and second FC layers (Skip 1 & 2). Once again, to clarify, these connections are appending onto the respective layer and not performing an element-wise summation, such as in ResNet’s architecture.

3.4.2 Convolutional Random Vector Functional Link - Fully Connected

Another architecture that utilizes the ideas of the RVFL and ELM is the previously discussed CRVFL. The CRVFL consists of a convolutional layer, average pooling layer, normalization layer, and a single FC layer. An additional direct connection from the input to the last FC layer was also used in this architecture. The authors in [48] showed that the CRVFL was able to effectively perform visual tracking while benefiting from fast training times. Therefore, this architecture is adopted in this work to evaluate its performance as it enables for a greater transformation of the feature space across its layers. However, small modifications are made to the CRVFL network by adding in an additional FC layer as the number of outputs from the model approach memory limits. This problem will be further discussed in Chapter 4. With the additional FC layer, we refer to this architecture as CRVFL-FC, shown in Figure 3.9.

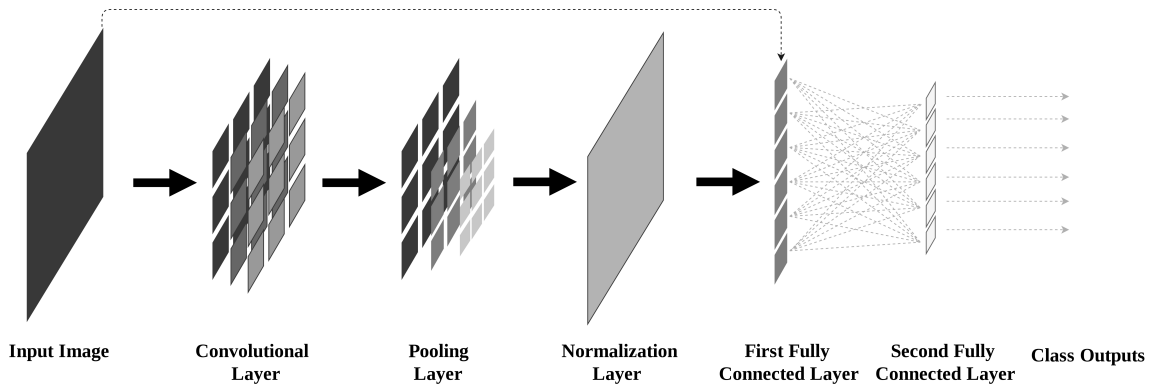


Figure 3.9: Illustration of the CRVFL-FC architecture. A convolutional layer is used to extract features that are sent to a pooling layer to reduce the spatial size of the feature maps. A normalization layer is used to decrease covariance shift and then the output is fed to the last 2 FC layers. A skipped connection is used from the input to the first FC layer as this was used in the original architecture.

3.4.3 Tensor-Train Extreme Learning Machine

The last architecture used in this work is an ELM with a TT-FC layer in its hidden layer as opposed to the use of a FC layer in the hidden layer. The origination of the TT-FC layer is discussed in [86]. The TT-FC is a TT decomposed, previously discussed in Chapter 2, FC layer into its respective TT format. An illustration comparing an FC layer and TT-FC layer can be seen in Figure 3.10.

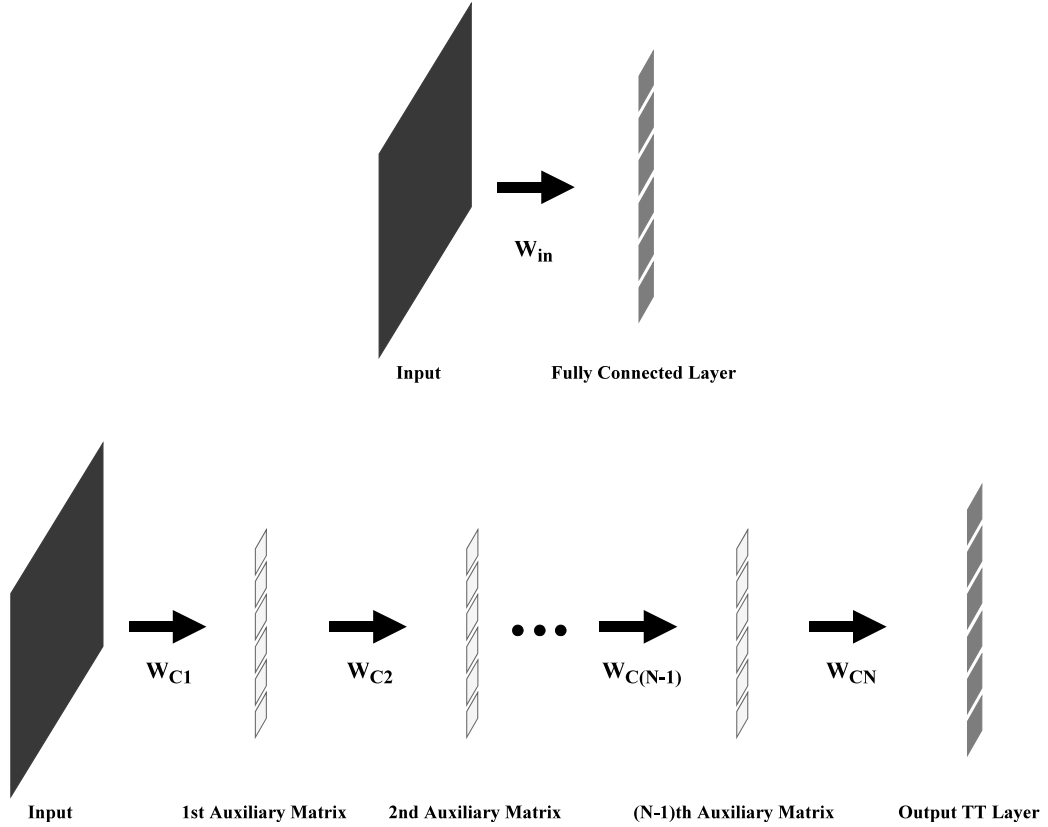


Figure 3.10: Illustration of an FC layer and its representation in a TT-FC layer format. The initial layer's inputs are decomposed into several TT cores that act as unfolded auxiliary matrices to the output.

In the FC layer, a dense weighted connection, denoted by \mathbf{W}_{in} , is taken from its input to a successive layer and then the layer outputs to either another layer or is passed to a classifier. In contrast to this, the TT-FC layer utilizes multiple weighted connections, denoted by \mathbf{W}_{C1} , \mathbf{W}_{C2} , \dots $\mathbf{W}_{C(N-1)}$, and \mathbf{W}_{CN} , between auxiliary

matrices, where $\mathbf{N} + 1$ is the total number of ranks used in the TT format and \mathbf{N} is the dimensionality of the input tensor. To clarify, in the TT-FC format, the input tensors can be of varying order. For example an image can either be by grayscale or rgb leading to 2nd or 3rd order tensors, however, the input can be reshaped into tensors of higher order. This enables for a higher number of ranks to be used as the ranks need to be $N + 1$. With the higher dimensionality of the original tensor input, more auxiliary matrices can be formed for the decomposition. This increase in auxiliary matrices is shown to empirically influence performance as the original tensor is approximated to a certain error, ϵ , as discussed in Chapter 2. It should be clarified that the output is set to the same order as the input for the multiplications to be performed properly in the neural network architecture.

In contrast to the dense FC layer, the weighted connections in the TT-FC layer are typically much smaller leading to a reduction in the total number of parameters stored for the network [86] while still enabling for the same amount of output neurons. To clarify, fixed-random weights are still used for the weighted connections of the new TT-FC layer. Therefore, to leverage the use of fixed-random weights and the TT format, the Tensor-Train Extreme Learning Machine (TT-ELM) is formed. The TT-ELM consists of a single TT-FC layer in its hidden layer and then a dense connection to the output classes, where training occurs on the output weights between the hidden and output layer. This enables for random projection through the auxiliary matrices of the TT-FC layer, while taking the outputs of the layer as part of the calculation for the MPP to find the output weights. A similar network can be constructed as the TT-RVFL, which is equivalent to the TT-ELM except for the added input-to-output concatenating skipped connection.

Similarly to the Semi-Random CNNs, the number of FLOPs is calculated in these ELM architectures. Aside from the FLOPs from the forward pass of the network, additional calculations are made in the MPP operation. As described in Chapter 2,

the MPP takes the last output of the hidden layer, \mathbf{H} , to calculate the output weights. Operations, such as matrix multiplication and the inverse, occur using \mathbf{H} and thus to approximate the number of FLOPs that occur in the MPP the number of FLOPs for these operations need to be calculated. For matrix multiplication, given matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, the number of FLOPs is calculated to equal $2mnp$ [87]. In [88], given a matrix $\mathbf{C} \in \mathbb{R}^{q \times q}$, the number of FLOPs in the matrix inverse operation is approximated to $\frac{2q^2+3q^2-5q}{6}$ and then furthermore approximated to $\frac{2q^3}{3}$, where the latter is used in the calculations for this work. The calculations for the total number of FLOPs for these architectures can then be presented in (3.2).

$$\mathbf{FLOPs} = \mathbf{F} + \mathbf{Mx} + \mathbf{I} \quad (3.2)$$

In this equation, \mathbf{F} denotes the number of FLOPs in the forward pass, while \mathbf{x} denotes the number of matrix multiplications that occur. \mathbf{M} denotes the number of FLOPs for matrix multiplication and \mathbf{I} denotes the number of FLOPs for the inverse operation. By taking the sum of all these elements, the number of FLOPs for the random projection architectures are approximated.

Chapter 4

Results & Discussion

4.1 Semi-Random CNNs

Initial experiments on the Semi-Random CNNs consisted of varying trained fractional amounts on the two DNNs, ResNet-20 and DenseNet100-BC. In this experiment, ResNet-20 was trained in a similar manner to the original paper by [3]. The network was trained on CIFAR-10 for 200 epochs with a batch size of 128 at an initial learning rate of 0.1 that decreased by a factor of 10x at epochs 80 and 120. SGD was used with a weight decay of 0.0001 and a momentum of 0.9. This network was averaged over 10 runs and only the convolutional layers' weights were modified to have fixed-random weights. In addition to the statistics discussed earlier, the size of storing the weights of the network, as well as the input data, is recorded in megabytes. A plot of accuracies to fractional amount of trained filters with respect to the original model on the ResNet-20 architecture for CIFAR-10 is presented in Figure 4.1. Additionally, Table 4.1 clarifies the accuracies shown in the plot.

From the results, it can be seen that the fully trained model achieves the highest test accuracy. However, both the fixed-random and downsized configurations with trained fractions of 0.75, 0.5, and Linear, illustrate comparable accuracies within 3% of the fully trained model. It's worthy to note that, in the comparison of the fixed-random and downsized configurations, the fixed-random weights perform better

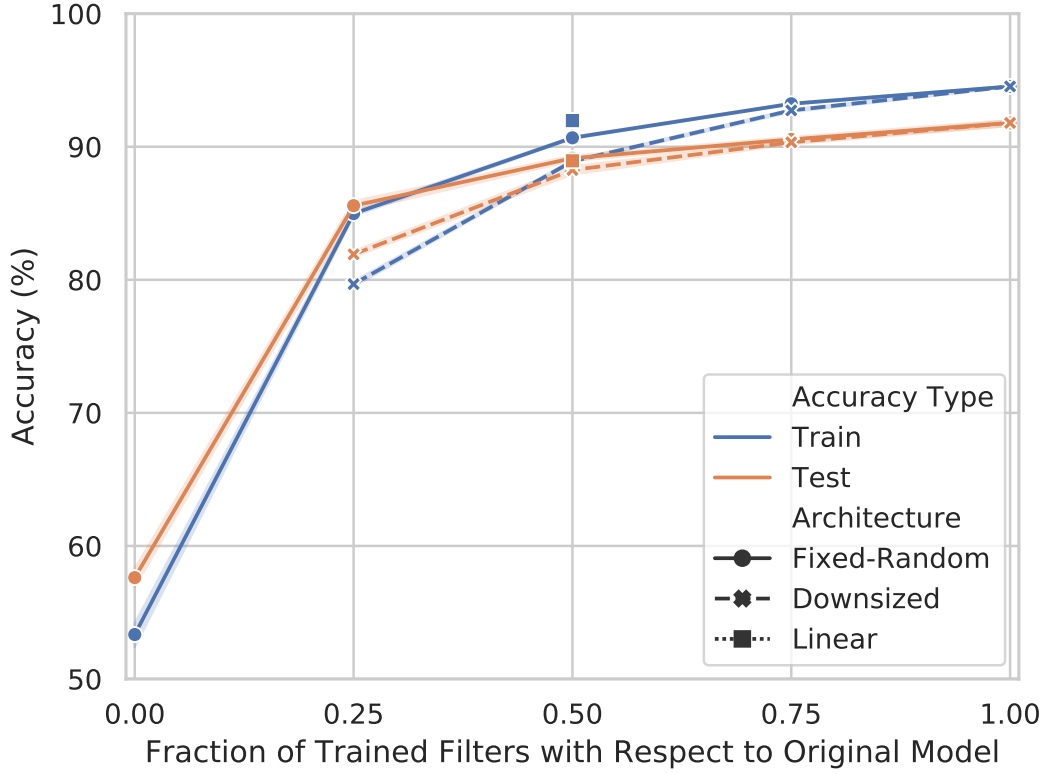


Figure 4.1: Plot of each ResNet-20 configuration on the CIFAR-10 dataset averaged over 10 runs with a confidence interval of 99%. Fixed-random refers to the case of trained and fixed-random filters and downsized refers to the case of fewer filter models. The case of Linear refers to the increase of trained weights with each convolutional layer in the network. It's noted that models with trained and fixed-random filters outperform the downsized versions of the models across all cases. This difference is more noticeable at smaller fractional amounts of trained filters. This is due to the fixed-random filters enabling for a greater feature space to be extracted from in comparison to the downsized models.

across all cases, even if slightly. This performance difference is more noticeable in the 50% and Linear cases, and even more so in the 25% case. Intuitively, the reasoning for this is due to the additional parameter space available in the fixed-random models as opposed to the downsized models. This parameter space enables for more features to be extracted from the input which can be trained on by the trained filters and other trained layers of the network. Unfortunately, the fully random projection network shows poor performance in comparison to all other configurations. This is due to less than only 0.8% of the network containing trained parameters leading to little

Table 4.1: Performance on the CIFAR-10 dataset using the ResNet-20 architecture. A dividing line is used to separate networks with trained and fixed-random filters and down-sized networks with fewer filters. The network is averaged over 10 runs and a confidence interval of 99% is used. From the results, the fixed-random networks are able to achieve more comparable accuracies to the fully trained model as opposed to the downsized network cases.

	Trained Fraction	Train Acc (%)	Test Acc (%)
Random	1	94.53 ± 0.04	91.79 ± 0.23
	0.75	93.23 ± 0.06	90.55 ± 0.21
	0.5	90.67 ± 0.08	89.14 ± 0.19
	0.25	84.98 ± 0.19	85.57 ± 0.42
	0	53.34 ± 0.95	57.62 ± 0.75
	Linear	91.97 ± 0.08	88.94 ± 0.30
Down-sized	0.75	92.73 ± 0.11	90.33 ± 0.13
	0.5	88.91 ± 0.09	88.25 ± 0.31
	0.25	79.68 ± 0.23	81.91 ± 0.25

actual training of the network to learn from its inputs. The random projection of the Linear method, though, enabled for performance comparable to the 50% random and downsized networks. In the Linear configuration, complete random projection is used in the first convolutional layer and then decreased over each convolutional layer. With the first few convolutional layers having a majority of random weights, lower level features are projected to become separable, while the network learns more complex features at its later layers from the untrained low level features. It could be stated that the use of partial random projection of the initial layer do not hinder the network’s performance substantially, however, another network or dataset needs to be discussed before claiming this. Additionally, the Linear configuration utilizes nearly the same amount of trained parameters in comparison to the 0.75 configuration, as highlighted in Figure 4.2.

This is due to a greater quantity of filters being used in the later layers of the architecture leading to a greater amount of trained filters. Setting 75% of each convolutional layer to trained weights led to a fairly distinct difference in performance of nearly 2% in comparison to the Linear case. Therefore, the use of random projection

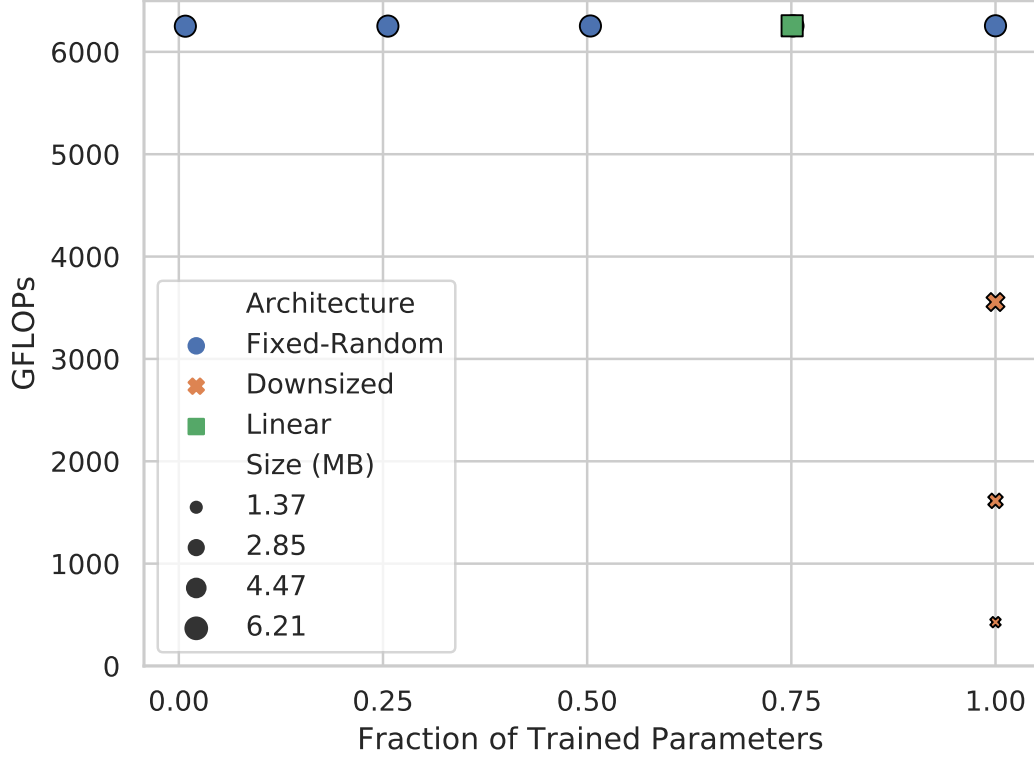


Figure 4.2: Statistics plot of each ResNet-20 configuration on the CIFAR-10 dataset averaged over 10 runs with a confidence interval of 99%. The configurations consist of fixed-random filters in the convolutional layer and downsized models with fewer filters in the convolutional layer. It can be seen that the downsized filter models utilize fewer GFLOPs in comparison to the fixed-random and fully trained models. This is because the fixed-random filter models need to backpropagate the entire network back due to their mix of trained and random filters. In contrast, downsized models demand fewer resources because less filters are utilized in these models as well as fewer operations on the last FC layer.

in the early layers of the architecture may actually have slightly hindered the features learned in the network.

Although entire convolutional random projection in the ResNet-20 architecture may not be beneficial, it can be seen that comparable performance can be achieved when using trained amounts of 0.75 and 0.5. Additionally, in terms of the number of resources used in these networks, resource utilization can be decreased minimally in the fixed-random case in comparison to the fully trained network for the backward pass as fewer memory writes occur in these models. However, networks with fewer

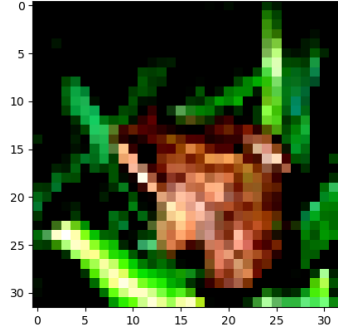


Figure 4.3: Image of a frog from CIFAR-10.

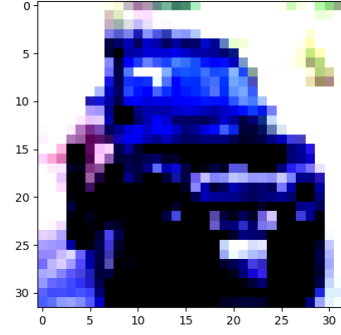


Figure 4.4: Image of an automobile from CIFAR-10.

filters leads to fewer FLOPs in both the forward and backward passes. This is highlighted in the 0.75 case, where fewer FLOPs occur in comparison to fixed-random weights. Furthermore, the 0.75 downsized case leads to a substantial decrease in resources over even the fixed-random weight case of 0.25 while retaining performance comparable to the fixed-random weight case of 0.75. Therefore, it's difficult to recommend the usage of fixed-random weights in this architecture type as a smaller version of the original trained network can utilize fewer FLOPs leading to a decrease in training time while resulting in a high performance model.

To gain a greater understanding as to why the performance of the fixed-random ResNet-20 performs fairly well, visualizations of the activations for each convolutional layer of the fixed-random network using 50% training and fixed-random filters were taken for two example images of a frog and automobile from CIFAR-10. The example images are shown in Figures 4.3 and 4.4. The layer by layer activations of a fixed-random filter and trained filter for the ResNet-20 model are presented in Figures 4.5 and 4.6.

For all images of the visualizations, the left images of each layer illustrates the fixed-random filter case while the right filtered image illustrates the trained filter case. From the visualization of the activations from the convolutional layers of the

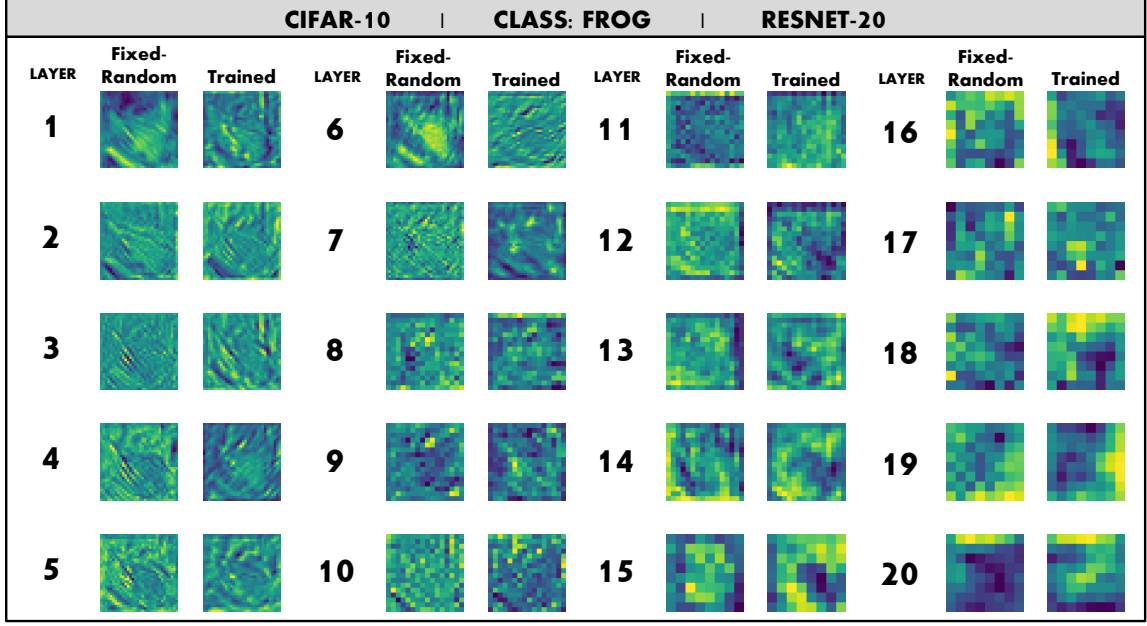


Figure 4.5: Activated outputs of each convolutional layer in ResNet-20 on the input image of a frog from CIFAR-10. Features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

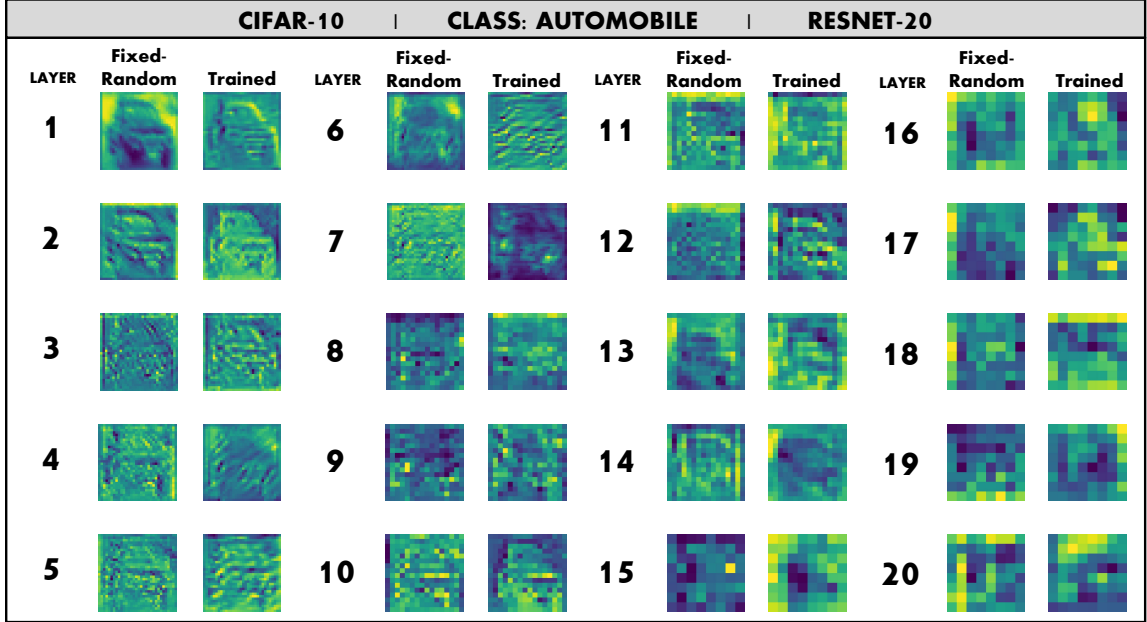


Figure 4.6: Activated outputs of each convolutional layer in ResNet-20 on the input image of an automobile from CIFAR-10. Similarly to the frog, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

frog image from CIFAR-10, it can be seen that distinct features are more visible in the initial layer activations for both the fixed-random and trained filter cases. These

visualizations are far less distinct in later layers where deeper representations are extracted. In the fixed-random filters, textures of the frog can be seen as well as activations on the frog’s body leading to a lighter green hue effect. In comparison to this, the trained weights illustrate a similar activation to the textures. In both filter cases, textures are the dominating visualization that can be seen. However, the trained filters illustrate more visible changes in the representation of the image. This is most likely due to the trained filters learning to become more activated on certain aspects of the image in contrast to other elements. This visualization supports [5]’s work where the authors discuss how fixed-random filters learn textures from their inputs. It also supports how fixed-random filters can be frequency selective, as discussed in [62], which can be seen in how the filters activate to certain portions of the images.

To illustrate the visualizations further, another image from CIFAR-10 is taken. This image is of an automobile which has less of a focus on the background. This can be seen more clearly as the filters are more activated on the body of the automobile, which is more distinct in the initial layers as opposed to the later layers. Similarly to the image of the frog, both the fixed-random and trained filters activate on the textures, which can be seen in how the automobile has mostly uniform activation. A contrast in the activations can be seen in the automobile’s wheels for the initial layers, which is darker in the original image. The features are less distinguishable in the deeper layers, but the trained filter is most likely learning representations that help it to distinguish this image from another class. In contrast, the fixed-random filter is extracting features that can be used in the trained layers of the training process. Additional visualizations can be seen in the Appendix at 5.2.

In order to further evaluate the effects of fixed-random weights, the SVHN dataset is trained on with the ResNet-20 architectures. The performance results for these experiments can be seen in Figure 4.7 and Table 4.2.

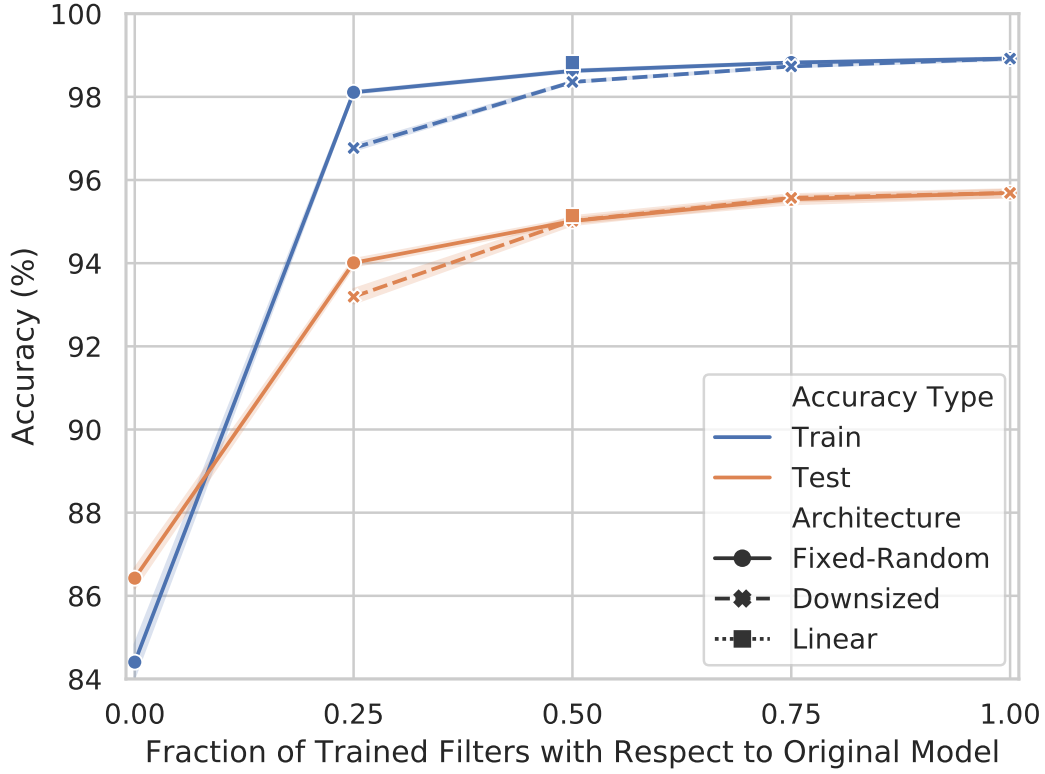


Figure 4.7: Plot of each ResNet-20 configuration on the SVHN dataset averaged over 10 runs with a confidence interval of 99%. Fixed-random refers to the case of trained and fixed-random filters and downsized refers to the case of fewer filter models. The case of Linear refers to the increase of trained weights with each convolutional layer in the network. Unlike the results from CIFAR-10, random projection shows a surprisingly high performance, while the other models show more comparable performances to one another. This is most likely due to the higher number of training samples and possibly more simple imagery in comparison to CIFAR-10.

From these results, a contrast can be seen in comparison to the CIFAR-10 dataset. Although both datasets consist of 32×32 RGB images, the SVHN dataset contains more samples leading to higher performing models. However, the largest contrast to CIFAR-10 is the accuracy of the random projection network having a test accuracy different of less than 10% with the fully trained model. This can be attributed to simpler imagery and a larger dataset to train on as the few weights left trained in the architecture can be tuned to obtain a greater classification distinction. Likewise to the CIFAR-10 results, the fixed-random weights achieve higher accuracies to the

Table 4.2: Performance on the SVHN dataset using the ResNet-20 architecture. A dividing line is used to separate networks with trained and fixed-random filters and downsized networks with fewer filters. The network is averaged over 10 runs and a confidence interval of 99% is used. The results illustrate clearly how the fixed-random models outperform the downsized models even if marginally. Additionally, the random projection network shows a surprisingly high accuracy. This may be attributed to additional samples and possibly more simple imagery.

	Trained Fraction	Train Acc (%)	Test Acc (%)
Random	1	98.92 ± 0.03	95.69 ± 0.10
	0.75	98.83 ± 0.02	95.54 ± 0.10
	0.5	98.62 ± 0.03	95.01 ± 0.06
	0.25	98.11 ± 0.03	94.01 ± 0.10
	0	84.41 ± 0.39	86.42 ± 0.26
	Linear	98.82 ± 0.02	95.14 ± 0.09
Down-sized	0.75	98.73 ± 0.02	95.57 ± 0.09
	0.5	98.36 ± 0.02	95.02 ± 0.11
	0.25	96.77 ± 0.08	93.19 ± 0.19

downsized models, albeit only slightly. This can be attributed to the additional feature space that is available to be trained on in the portions of the layers that contain trained weights. Similarly to the ResNet-20 CIFAR-10 results, additional statistics are gathered as depicted in Figure 4.8.

As the data consists of similar sized images to CIFAR-10, the memory size needed to store the parameters is equivalent to CIFAR-10. However, the number of GFLOPs needed to operate on SVHN for a single epoch of training is higher due to the additional number of samples available to train on. The trend of high GFLOPs utilized occurs for the fixed-random cases as the network needs to perform the entire backward pass to modify the trained portions of the network. However, the downsized cases decrease this as fewer filters are actually used in the architecture.

To further study the use of fixed-random weights in deep CNNs, the DenseNet architecture is evaluated on as a contrast to ResNet’s summation skipped connectivity. The DenseNet architecture’s results are presented in Figure 4.9 and Table 4.3. In this experiment, similar to how the ResNet-20 experiment was set up, DenseNet-BC with

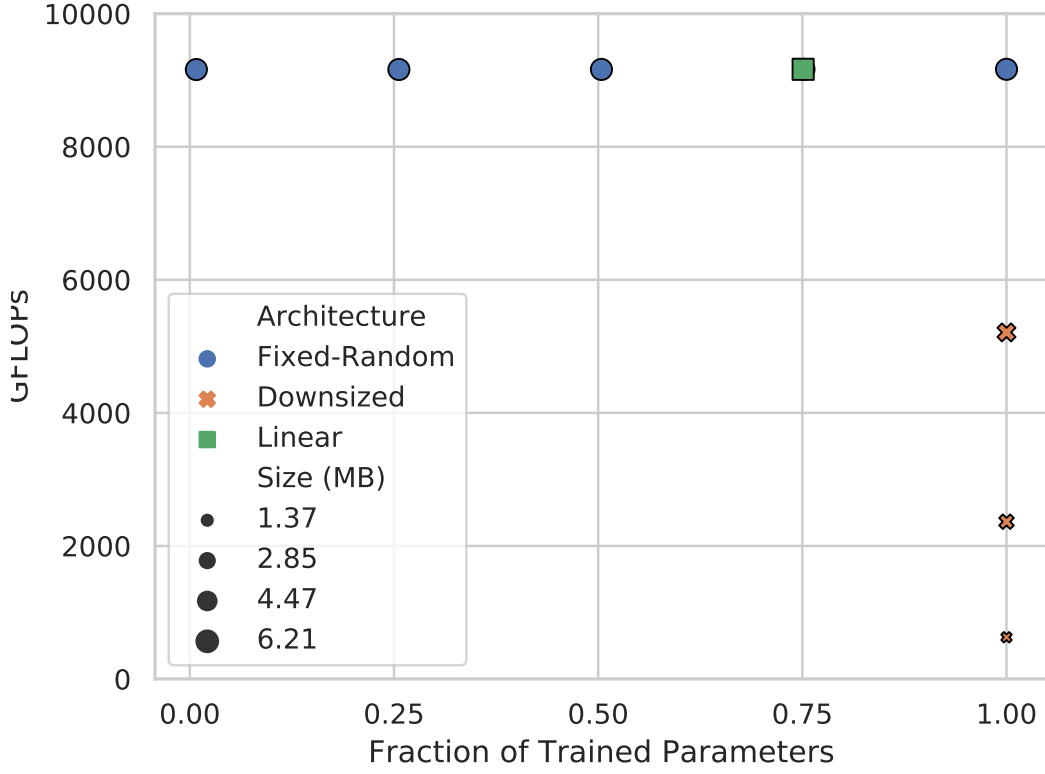


Figure 4.8: Statistics plot of each ResNet-20 configuration on the SVHN dataset averaged over 10 runs with a confidence interval of 99%. The configurations consist of fixed-random filters in the convolutional layer and downsized models with fewer filters in the convolutional layer. The downsized models utilize fewer GFLOPs as less operations are occurring in these networks. Additionally, the downsized models require fewer operations in their forward passes leading to a greater decrease in the number of operations and an additional decrease in memory size as fewer parameters are available in these models. The fixed-random models utilize a similar amount of GFLOPs to the fully trained models due to backpropagation occurring across all layers.

100 layers, denoted as DenseNet100-BC, was trained in a similar manner to its original paper in [4]. The network was trained on CIFAR-10 for 300 epochs with a batch size of 64 at an initial learning rate of 0.1 that decreased by a factor of approximately 10x at epochs 150 and 225. SGD was used with a weight decay of 0.0001 and a momentum of 0.9. This network was averaged over 5 runs and only the convolutional layers' weights were modified to have fixed-random weights.

From the results, it was expected that the fully trained model performs the best.

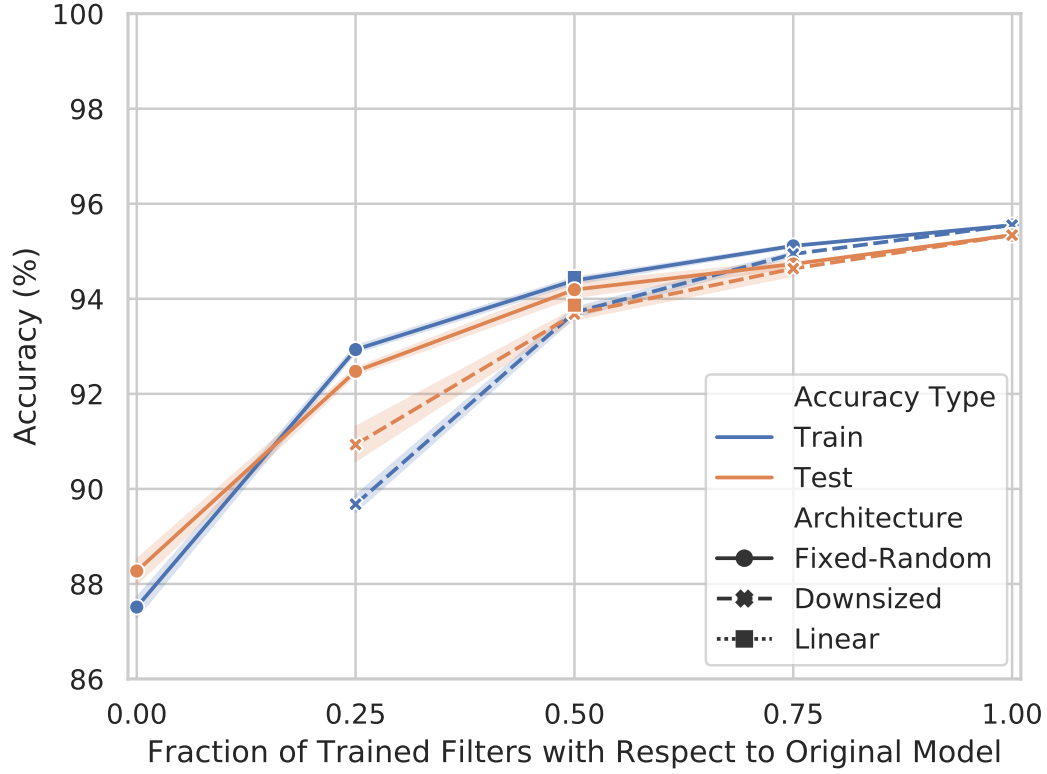


Figure 4.9: Plot of each DenseNet100-BC configuration on the CIFAR-10 dataset averaged over 5 runs with a confidence interval of 99%. Fixed-random refers to the case of trained and fixed-random filters and downsized refers to the case of fewer filter models. The case of Linear refers to the increase of trained weights with each convolutional layer in the network. The performance of random projection shows surprisingly that DenseNet’s concatenating skipped connectivity can be beneficial with random weights as features are extracted and trained on at the last FC layer of the architecture.

However, it’s intriguing to see that the fully random projection convolutional layers achieved performance that is only approximately 8% below the fully trained model. Only 12% of the network was trained in this instance as all convolutional layer filters were set to fixed-random values. In contrast to this performance, the ResNet-20 model’s use of random projection convolutional layers led to poor performance. To clarify, ResNet-20 with the fully random projection convolutional layers achieved an average test accuracy of 57.61% as highlighted in Table 4.1. That is an approximately 35% difference from its fully trained model, which achieved 91.87%. In contrast to

Table 4.3: Performance on the CIFAR-10 dataset using the DenseNet100-BC architecture. A dividing line is used to separate networks with trained and fixed-random filters and downsized networks with fewer filters. The network is averaged over 5 runs and a confidence interval of 99% is used. From the results, the fixed-random networks are able to achieve more comparable accuracies to the fully trained model as opposed to the zeroed out network cases. In addition to this, the random projection network with a 0 trained fractional amount is able to achieve over 88% accuracy.

	Trained Fraction	Train Acc (%)	Test Acc (%)
Random	1	95.55	95.34
	0.75	95.14 \pm 0.02	94.69 \pm 0.05
	0.5	94.33 \pm 0.08	94.05 \pm 0.17
	0.25	93.04 \pm 0.08	92.46 \pm 0.08
	0	87.51 \pm 0.23	88.27 \pm 0.29
	Linear	94.47 \pm 0.04	93.96 \pm 0.20
Down-sized	0.75	94.91 \pm 0.06	94.57 \pm 0.15
	0.5	93.72 \pm 0.10	93.81 \pm 0.12
	0.25	89.68 \pm 0.20	90.93 \pm 0.44

this, DenseNet only had an 8% difference. This illustrates how DenseNet’s use of connectivity may be beneficial in the process of classifying its inputs. Additionally, the use of extra layers in this architecture is most likely assisting the network’s ability to hierarchically feature extract from the input.

When comparing the accuracies of the fixed-random and downsized architectures, the fixed-random accuracies are comparable to the downsized versions. However, the use of fixed-random weights out performs the downsized cases even if negligibly. A more clear distinction between the performances can be seen in the 0.25 trained fraction amount. The difference in performance is around 1%, but this highlights how the fixed-random weights aren’t hindering the training process. Additionally, the random projection convolutional layers show high performance even with the network having a large majority of random weights. [62] discussed how important the architecture is in terms of achieving high performance regardless of trained or fixed-random weights, and the random projection case highlights this. The use of random projection is able to still extract features from the input which the 12% of trained parameters can

make use of to allow for separability in the data. As DenseNet’s structure concatenates the features of each successive layer, the random weights are actually assisting as features are still able to be obtained with each successive layer, without fine tuning during the backwards pass of the network. However, in comparison to the number of FLOPs, illustrated in Figure 4.10, required in this network, it’s preferable to utilize the downsized models that require fewer filters as they demand less FLOPs to achieve comparable performance to their fixed-random equivalents. Additionally, the downsized models can achieve greater performance to the random projection model in all cases meaning that training a smaller network can be beneficial as fewer resources are demanded on leading to a decrease in training time.

Similarly to ResNet-20, the DenseNet architecture has activated convolutional layer visualizations taken for the two CIFAR-10 images of the frog and automobile, shown above in Figures 4.3 and 4.4. The visualizations for DenseNet100-BC can be seen in Figures 4.11 and 4.12. To ensure that the images are more visible and clear, the activations for every 5 convolutional layers is taken. From the images of the frog, activations on the body of the frog can be seen in the first layer. However, after this initial layer, textures are the more prominent activation for the image. However, the 15th layer output can be seen to still capture the shape of the frog for both fixed-random and trained filters. The fixed-random filters show less distinct feature extraction in comparison to the trained filter in later layers. This can be seen more clearly in the automobile’s extraction. In layer 30 the car is still more noticeably activated on for the trained filter in comparison to the fixed-random filter which looks to be activated to the input in multiple places. In the deeper layers for both images, the features are less distinct to tell how the trained filter is learning from the input image as opposed to the fixed-random filter. Overall, in both cases, the trained filter has more distinct feature extraction over the fixed-random filter. Additional visualizations can be seen in the Appendix at 5.2.

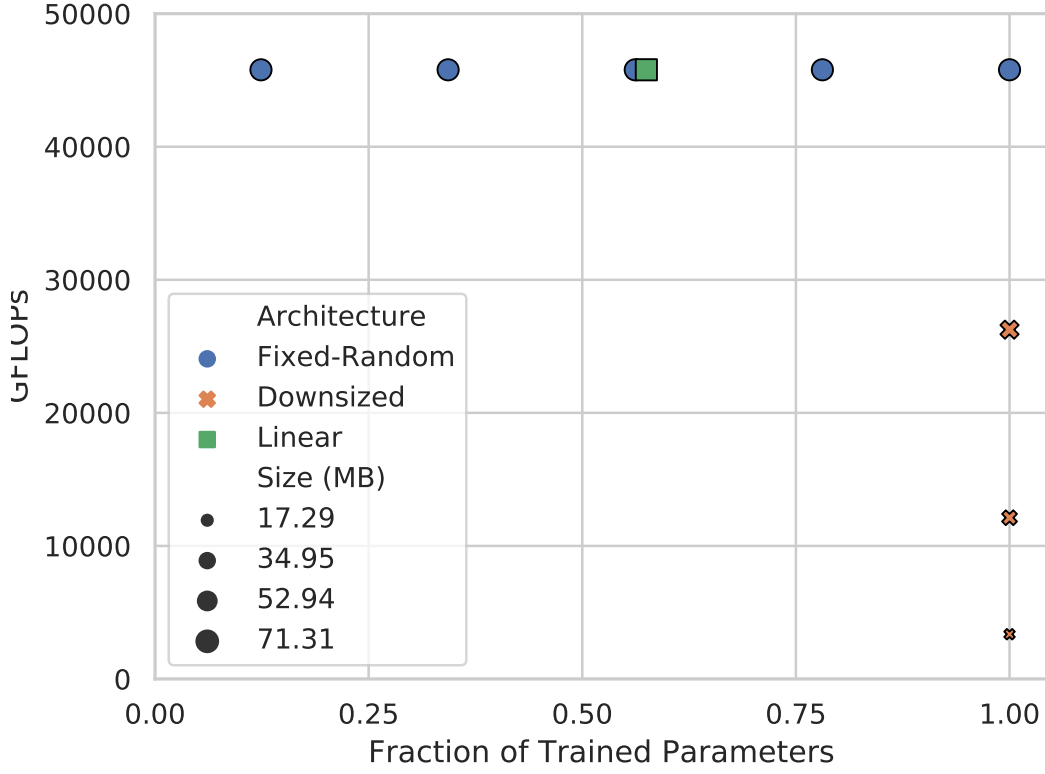


Figure 4.10: Statistics plot of each DenseNet100-BC configuration on the CIFAR-10 dataset averaged over 5 runs with a confidence interval of 99%. The configurations consist of fixed-random filters in the convolutional layer and downsized models with fewer filters in the convolutional layer. Similarly to the previous statistic plots, the downsized models show a decrease in GFLOPs and memory size over the fixed-random models. In addition to this, the fixed-random weights for random projection is shown to actually have approximately 12% of the weights trained. This is due to the concatenation skipped connectivity leading to more features trained on the output layer.

Likewise to ResNet-20, DenseNet100-BC was trained on the SVHN dataset to evaluate how a different dataset performed on the fixed-random and downsized architectures. The performance results of these experiments can be seen in Figure 4.13 and Table 4.4. Astoundingly, in these results the random projection network shows near comparable accuracies to all other networks. Only a small margin of around 2% can be seen in comparison to the fully trained model. The fixed-random weighted and downsized models obtain nearly the same accuracies. These results can be attributed to the dataset containing more samples to train on as well as simpler imagery in com-

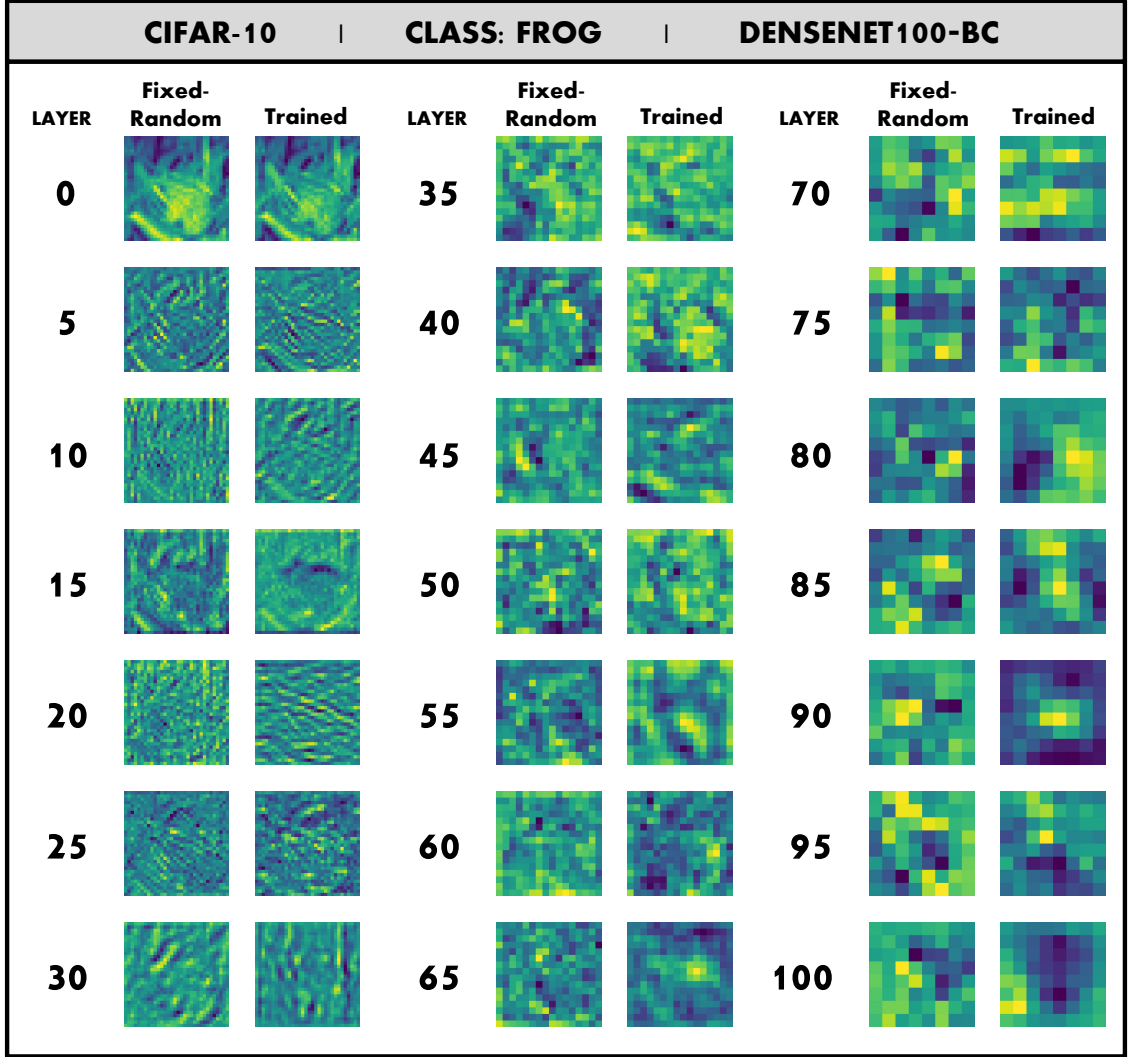


Figure 4.11: Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of a frog from CIFAR-10. Features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

parison to CIFAR-10. Similarly to CIFAR-10, the memory size and number of truly trained parameters are the same, as highlighted in Figure 4.14, because the input's dimensions are the same at $32 \times 32 \times 3$. However, in contrast to CIFAR-10, a greater amount of samples are available leading to an increase in the number of GFLOPs.

The next experiment conducted on these deep CNNs consisted of training their 75% trained/fixed-random model on CIFAR-10 for a greater number of epochs at their last learning rates and at a 10x decrease in the learning rate. The experiments

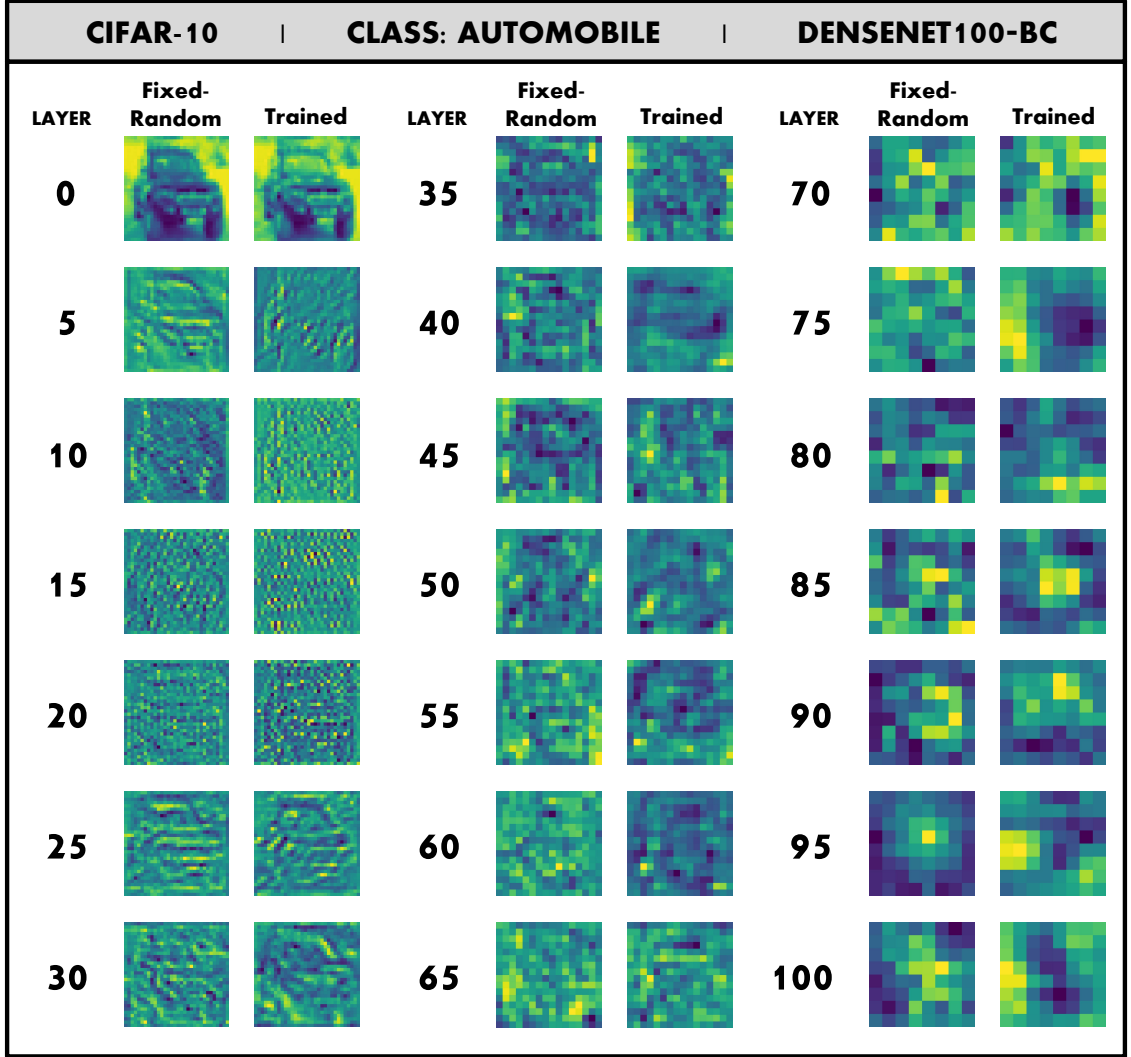


Figure 4.12: Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of an automobile from CIFAR-10. Similarly to the frog, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

were conducted such that the networks ran up until they reached a test accuracy of greater than or equal to their fully trained model's equivalents. This training occurred until 500 epochs of total training was reached, meaning that if the network was trained for 200 epochs then it would train for 300 more epochs. This experiment was meant to evaluate if a network with a mixture of trained and fixed-random filters could achieve a similar performance to its trained equivalent with more training. The

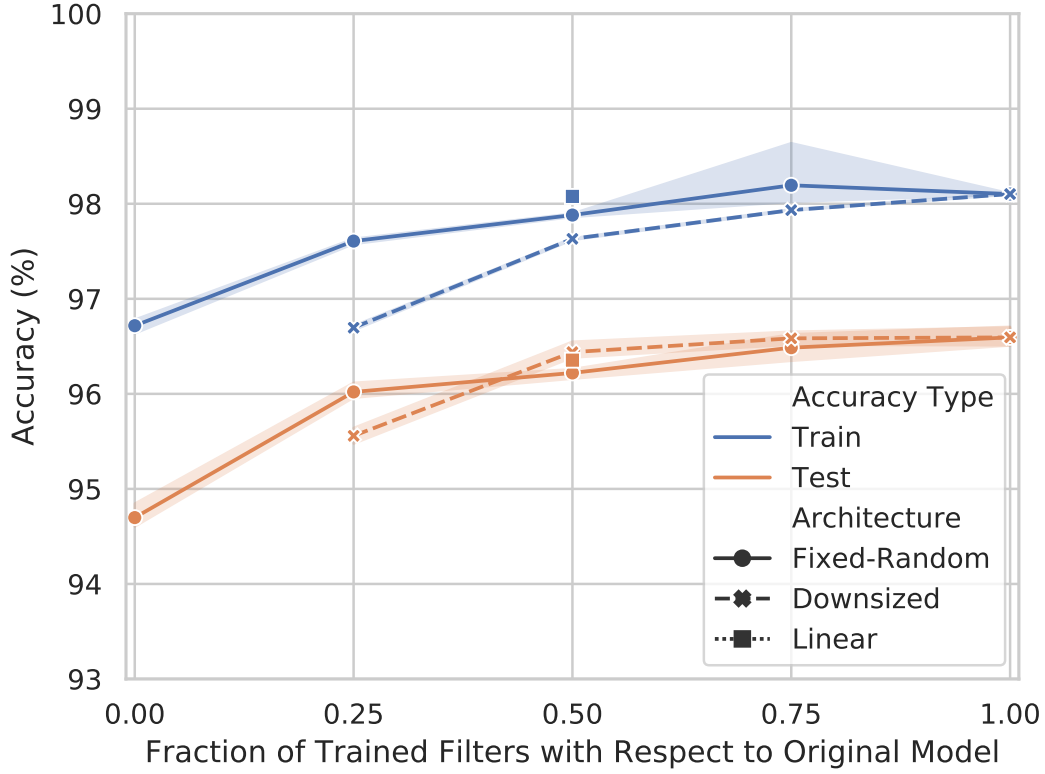


Figure 4.13: Plot of each DenseNet100-BC configuration on the SVHN dataset averaged over 5 runs with a confidence interval of 99%. Fixed-random refers to the case of trained and fixed-random filters and downsized refers to the case of fewer filter models. The case of Linear refers to the increase of trained weights with each convolutional layer in the network. In contrast to the results from CIFAR-10, random projection shows high performance comparable to that of the other models. This may be due to the additional samples available to train on for the few trained layers of the network. In addition to this, the other architectures illustrate near comparable performance to the fully trained model.

75% trained/fixed-random weight models were chosen as they consist of a majority of trained filters meaning that they may be tuned further with additional epochs to achieve equivalent or greater test accuracy. The results of this experiment can be seen in Table 4.5.

The results of this experiment show that even training a network with a larger percentage of trained weights for longer does not converge to the same as a fully trained equivalent network. To put in perspective, the trained accuracies for ResNet-20 and DenseNet100-BC were averaged to be 91.79 and 95.34, respectively. The

Table 4.4: Performance on the SVHN dataset using the DenseNet100-BC architecture. A dividing line is used to separate networks with trained and fixed-random filters and down-sized networks with fewer filters. The network is averaged over 5 runs and a confidence interval of 99% is used. The results show surprisingly that both fixed-random and down-sized models lead to performance extremely similar to the fully trained model. The random projection case even shows performance akin to the fully trained model. This may be attributed to a greater number of samples available as well as simpler imagery when compared to CIFAR-10.

	Trained Fraction	Train Acc (%)	Test Acc (%)
Random	1	98.11 ± 0.01	96.55 ± 0.12
	0.75	98.79 ± 0.39	96.46 ± 0.16
	0.5	97.90 ± 0.02	96.24 ± 0.05
	0.25	97.61 ± 0.03	96.02 ± 0.10
	0	96.72 ± 0.09	94.70 ± 0.13
	Linear	98.11 ± 0.04	96.54 ± 0.16
Down-sized	0.75	97.93 ± 0.01	96.56 ± 0.08
	0.5	97.63 ± 0.01	96.41 ± 0.10
	0.25	96.70 ± 0.03	95.56 ± 0.10

Table 4.5: Performance on the CIFAR-10 dataset after training each deep CNN for additional epochs up until 500 total trained epochs.

Model	Trained Fraction	Learning Rate	Train Acc (%)	Test Acc (%)
ResNet-20	0.75	0.001	98.98	90.75
DenseNet100-BC	0.75	0.001	99.98	94.76
ResNet-20	0.75	0.0001	98.71	90.75
DenseNet100-BC	0.75	0.0001	99.98	94.81

accuracies achieved from training longer are within the same accuracies as previously reported for the 0.75 cases of each deep network meaning they may have reached a minima, whether a global or local minima. Therefore, unfortunately, training for longer did not allow for fixed-random weights to act as a supplementary factor for enabling the trained weights to adjust towards the more ideal solution.

The last experiment conducted on these Semi-Random CNNs is the use of fixed weights in transfer learning with fine-tuning. The ResNet-50 model is utilized for this experiment to show a comparison between this work and the work of Scott et al. [16], who perform transfer learning with ResNet-50 on the UCM dataset. Similarly

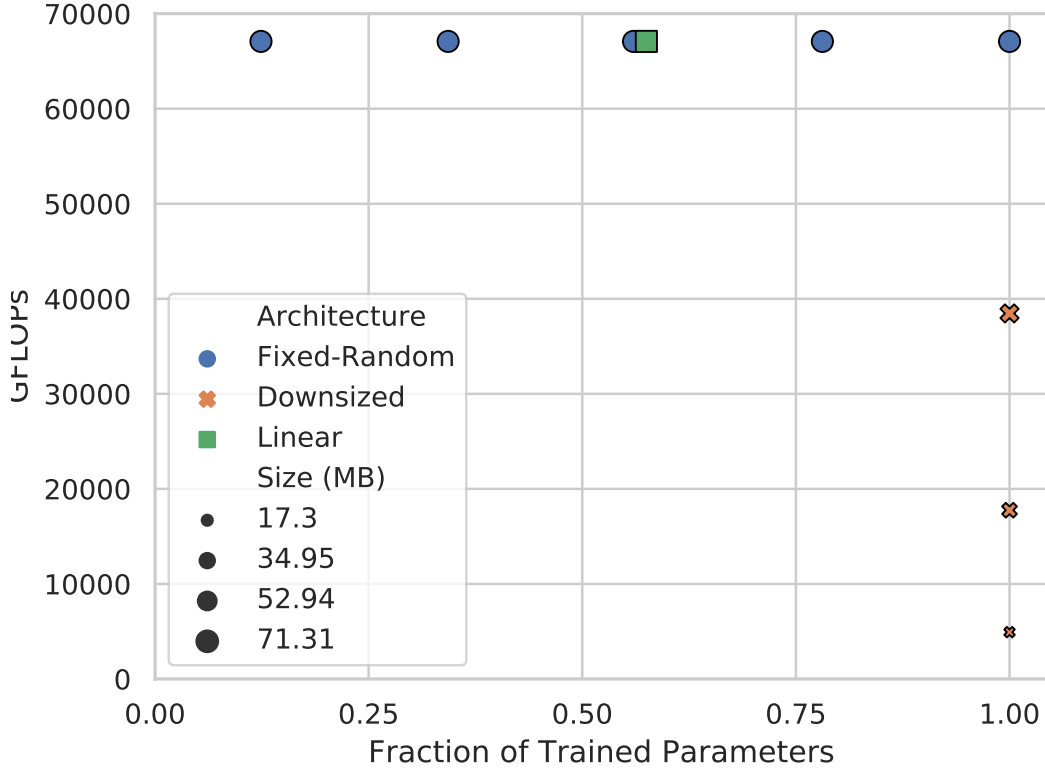


Figure 4.14: Statistics plot of each DenseNet100-BC configuration on the SVHN dataset averaged over 5 runs with a confidence interval of 99%. The configurations consist of fixed-random filters in the convolutional layer and downsized models with fewer filters in the convolutional layer. Similarly to the previous statistic plots, the downsized models show a decrease in GFLOPs and memory size over the fixed-random models. The random projection model with only fixed-random convolutional layers is shown to have approximately 12% of the weights trained due to concatenated features trained on the output layer.

to their setup, the input data of the UCM dataset is augmented with 7° rotations and random horizontal and vertical flips are utilized. Additionally, a learning rate of 0.001 and a momentum of 0.9 is utilized as stated in their work. As the number of training epochs is not specified in this work, 100 epochs of training with a batch size of 64 is used as this led to convergence for the model. The network is averaged over 10 runs with a confidence interval of 99%. The results for the transfer learning experiment can be seen in Figure 4.15 and Table 4.6, to clarify the accuracy values achieved.

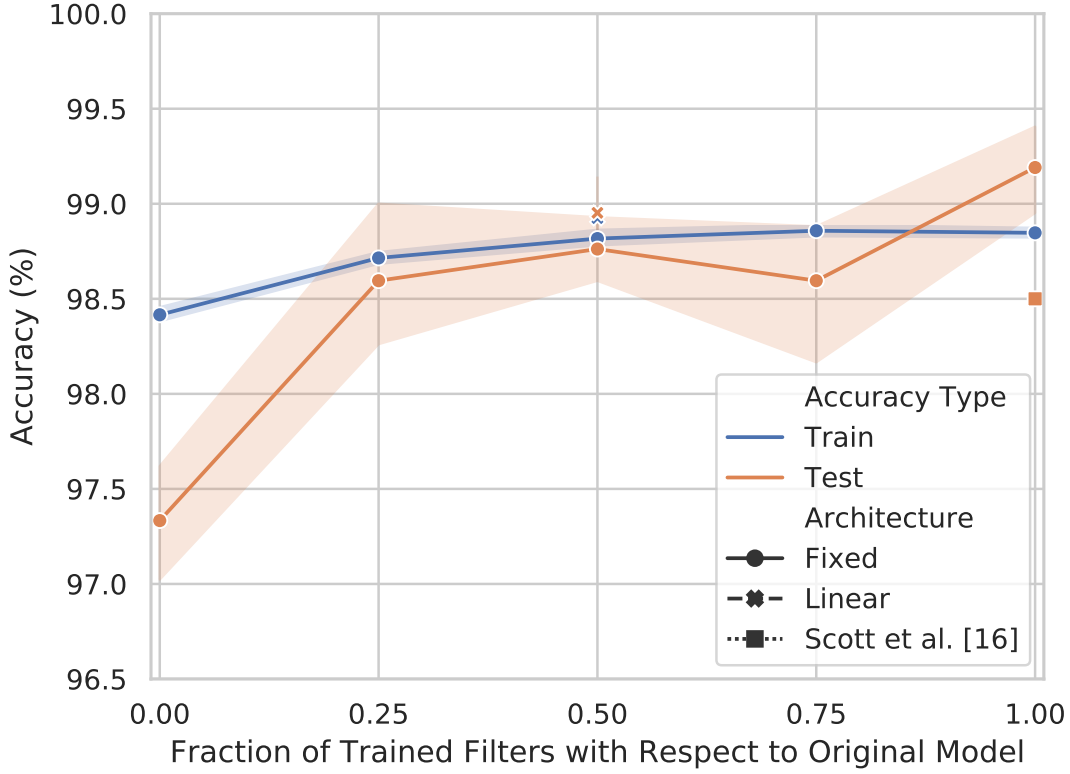


Figure 4.15: Plot of each ResNet-50 transfer learning configuration on the SVHN dataset averaged over 10 runs with a confidence interval of 99%. Fixed refers to the case of trained and fixed filters, from the pretrained model, and Linear refers to the increase of trained weights with each convolutional layer in the network. Fine-tuning the entire network shows increased performance over partial fine-tuning, however, near comparable accuracies can be achieved with partial fine-tuning and fewer resources are utilized as fewer memory writes occur. Additionally, the results show that performance over that of Scott et al.’s work [16] can be achieved with the partial fixed networks.

From the results of this experiment, it can be seen that the use of partial fixed weights is effective in obtaining high performance on the UCM dataset. In this experiment, the ImageNet transferred weights are useful as it gives the networks pretrained weights that have been trained on a variety of imagery. Fine-tuning of the entire network shows that changing the weights for the new task is still the best way to achieve the highest accuracy. However, the partial fine-tuned networks, or fixed networks, are able to achieve performance akin to the fully fine-tuned model. These models, aside from the model that had a trained fraction of 0, were able to

Table 4.6: Performance on the UCM dataset by transfer learning the ResNet-50 architecture. The network is averaged over 10 runs and a confidence interval of 99% is used. A dividing line is used to separate the fixed networks fine-tuned in this work compared to that of Scott et al. [16]. Near comparable accuracies can be achieved with partial fine-tuning in comparison to entire fine-tuning of the network. In all cases, except for only training the FC layer of the network, the accuracy is shown to surpass that of the work of Scott et al. [16].

	Trained Fraction	Train Acc (%)	Test Acc (%)
Fixed	1	98.85 ± 0.03	99.19 ± 0.25
	0.75	98.86 ± 0.03	98.60 ± 0.36
	0.5	98.82 ± 0.04	98.76 ± 0.18
	0.25	98.71 ± 0.03	98.60 ± 0.38
	0	98.42 ± 0.04	97.33 ± 0.34
	Linear	98.93 ± 0.01	98.95 ± 0.21
	1 [16]	-	98.50 ± 1.40

outperform the work of Scott et al. [16]. The pretrained weights from ImageNet most likely allowed for this with partial fine-tuning as it effectively had been trained on a wide variety of imagery leading to models that could distinguish between the classes of the UCM dataset. This can also be seen by the use of entire fixed layers for the trained fraction of 0. Only the FC layer was fine-tuned and the architecture is still able to achieve performance within 2% of the fully fine-tuned model.

Overall, the experiments on both the deep CNNs with and without transfer learning illustrate how the architecture is the most important part as previously discussed in Chapter 2. The DenseNet architecture highlights how using random projection convolutional layers still enabled the network to learn to classify the different classes of both the CIFAR-10 and SVHN datasets as features were concatenated over each layer. Of the 2 architectures shown, DenseNet illustrated how its architectural design enables for features to be extracted in an effective manner. Additionally, results from transfer learning show how freezing many of the weights of a previously trained ImageNet model leads to networks with nearly the same performance as entire fine-tuning trained models.

4.2 Extensions to the ELM

In this section extensions to the ELM are explored to evaluate if increased performance can be achieved using random projection across multiple layers with the use of concatenating skipped connectivity similar to that of DenseNet's. Additionally, the use of TT is explored in the TT-ELM network. For each network evaluated on, the last FC layer is left at a constant 1,000 neurons, the lambda regularization rate is adjusted to values of 0.1 and 0.01, and the number of filters are evaluated on either 16 or 32 filters. Each network is averaged over 10 runs and trained on up to 10,000 training samples. To denote skipped connectivity for the CELMs, a 'Skip' to their respective layer is depicted. For example, 'CELM Skip 1' refers to the architectures that feeds the input image to a convolutional layer and additionally concatenates the input to the first FC layer. To evaluate these networks, their training and testing accuracies, training time in minutes, GFLOPs, and memory size in megabytes are recorded. The training time is recorded relative to a machine containing an Intel Core i5-6500 CPU @ 3.20 GHz, 16 GB RAM, with an NVIDIA Titan X GPU. The PyTorch framework [89] is used for each test of this process. The reasoning as to why training time is recorded in this sections as opposed to the previous section is to highlight the rapid training that can be achieved with these networks with the ability to obtain high performance. The number of GFLOPs will also highlight why these networks are able to train so quickly.

The MSTAR dataset is the first to be experimented on to evaluate these architectures. The train set of 5,499 training images is used to train these networks and the results of these experiments are shown in Figures 4.16 and 4.17.

In these plots the accuracies, training times, GFLOPs, and memory size required to store the parameters of the model are presented. In terms of the time to train each model, the difference is negligible across all models. As these times are fractions of

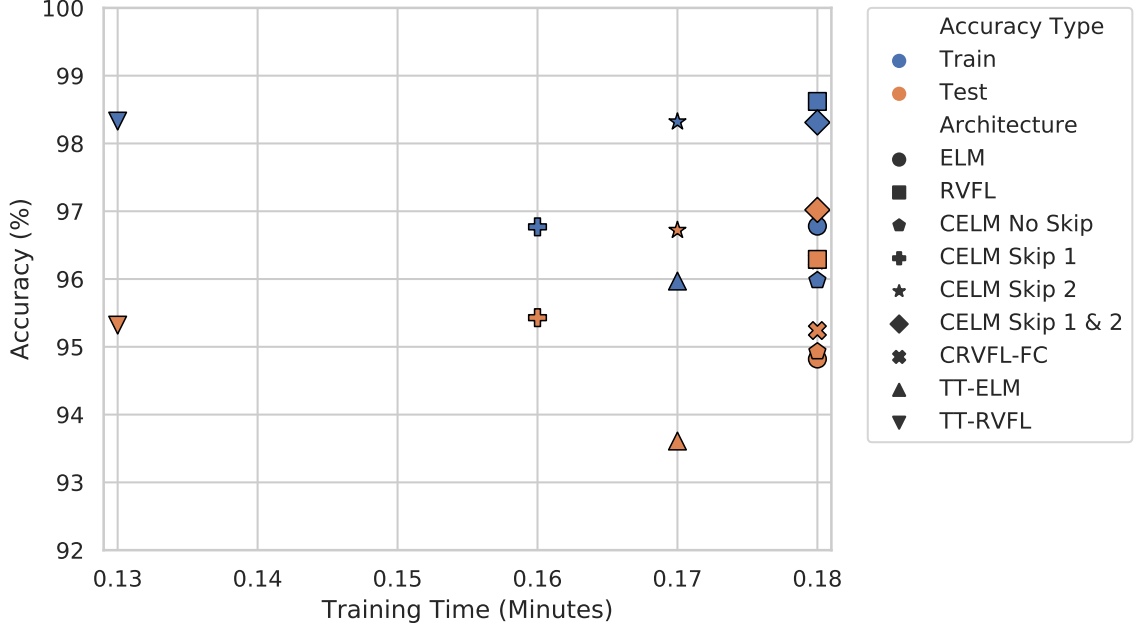


Figure 4.16: Performance of the random projection architectures on the MSTAR dataset. The CELM network with skipped connectivity to both the FC layers shows the highest test accuracy as more features are available on the output to train on.

a minute, the difference between 0.13 and 0.18 is only 3 seconds. This fast training time is due to less than 2.5 GFLOPs required to train the network. In addition to the quick training time, high performance of over 90% in all cases can be obtained with these networks on the grayscale 28×28 images. The network that achieved the highest performance is most notably the CELM model with skipped connectivity to the 1st and 2nd FC layers. This performance increase is attributed to the additional feature space that is available to train on in this model. The original input as well as features from each successive layer is utilized in a large feature space that can become separable. This claim is also supported when observing how the RVFL and CELM with a skipped connectivity to the 2nd FC layer also achieved comparable accuracies.

It’s also important to note how each network surpasses the performance of the original ELM except for the TT-ELM. This is due to the TT-FC layer’s approximation of an FC layer. Although, there is an accuracy difference, the model size of the network is substantially smaller by approximately 40x as fewer parameters are stored.

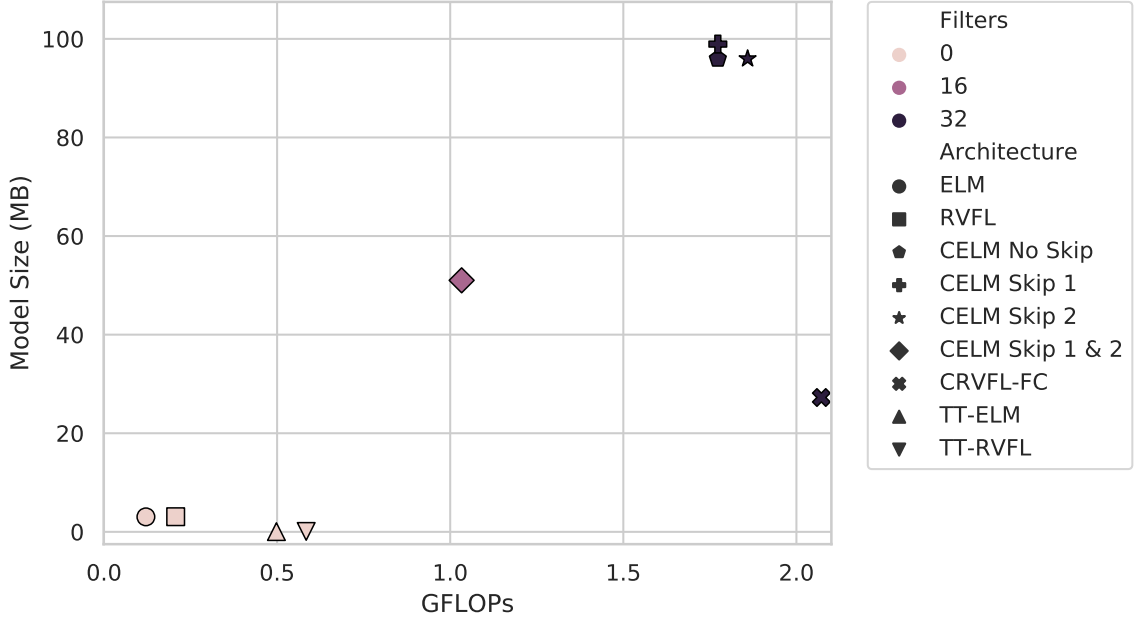


Figure 4.17: Statistics of the random projection architectures on the MSTAR dataset. The architectures utilizing additional layers require more GFLOPs and additional memory to store all of their parameters. It’s important to note that the TT-ELM and TT-RVFL are able to achieve a smaller model with the cost of more GFLOPs due to the additional operations occurring in the layer.

To illustrate why the parameter space is smaller, the ELM consisted of 1,000 hidden neurons in its FC layer, while the input was of size 28×28 . This leads to an input matrix of $784 \times 1,000$, or 7,840,000 parameters. In contrast to this, the TT-ELM changes the original inputs tensor size to decrease the parameter space. The input modes, or dimensions, of the input tensor would be reformed from 784 to $\langle 4, 7, 7, 4 \rangle$, which when reshaped to a 1st order tensor obtains 784. To clarify the notation used here, the use of ' $\langle \rangle$ ' denotes a list of dimensions that the tensor takes the form of. To approximate the 1,000 output neurons, output modes are set to values of $\langle 4, 8, 8, 4 \rangle$, which when reshaped to a 1st order tensor obtains 1,024. The tensor train cores are then formed relative to rank modes $\langle 1, 2, 2, 2, 1 \rangle$ which were determined to be these values as the authors in [82] present favorable results with this configuration in their FC net. In addition to this, the number of ranks must be equal to $N+1$, where N is the number of input dimensions, and the ranks must consist of $r_0 = r_N$

$= 1$ to enable the TT compression. From these ranks, cores are obtained such that weighted parameters between cores consisted of the following matrices: 4×4 , 14×14 , 14×14 , and lastly 8×8 . These add to a total number of 472 parameters in comparison to the 7,840,000 parameters of the original ELM's FC layer. Therefore, although a small accuracy drop can be seen, the TT-ELM presents a useful architecture that achieves rapid training time with little memory usage. However, to fully make this claim, it's important to test each of these networks more thoroughly. For this reason, experiments on the small-NORB and the FMNIST datasets can be seen in Figures 4.18, 4.19, 4.20, and 4.21.

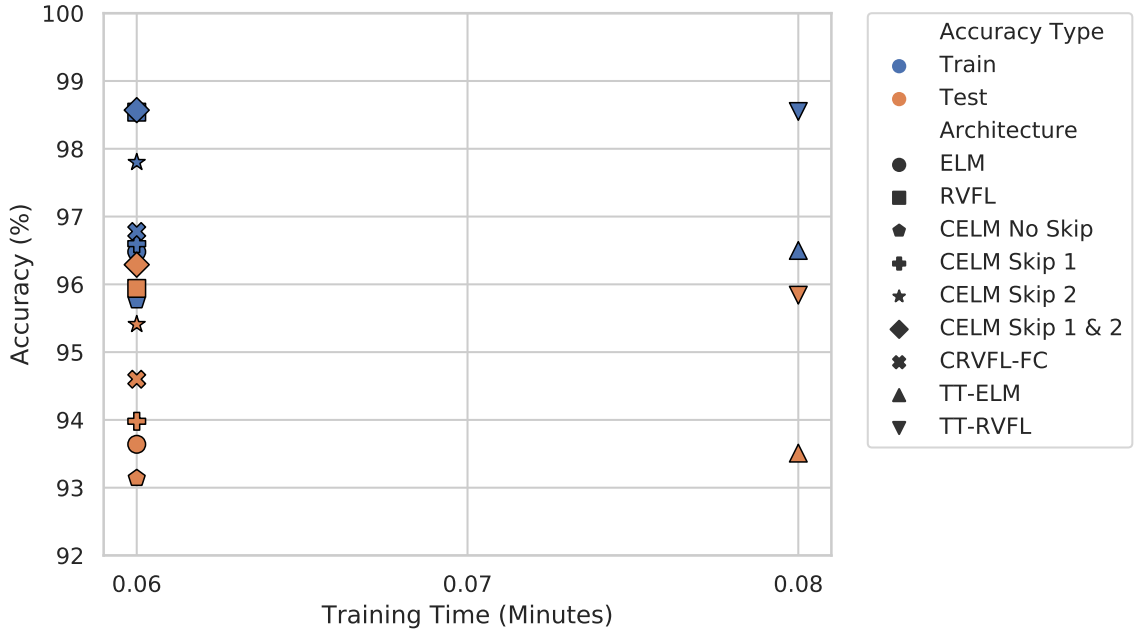


Figure 4.18: Performance of the random projection architectures on the small-NORB dataset. Similarly to the MSTAR dataset, the CELM with skipped connectivity to both FC layers achieves the highest performance as additional features are trained in this layer.

For both datasets, 10,000 samples are trained on to ensure that each network would run within the GPU's memory limits when performing the MPP operation. For the small-NORB experiments, the CELM consisting of skipped connectivity to both FC layers achieves the highest accuracy due to its additional parameter space. In comparison, the other networks show gaps in performance between one another.

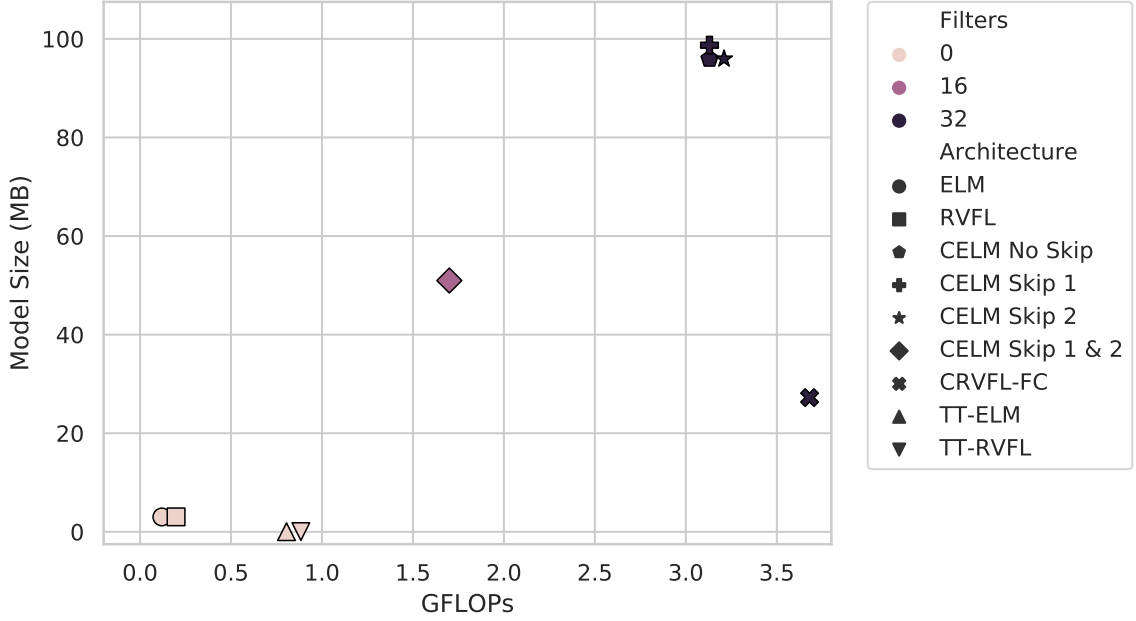


Figure 4.19: Statistics of the random projection architectures on the small-NORB dataset. The architectures utilizing additional layers require more GFLOPs and additional memory to store all of their parameters. The TT networks achieve a smaller model at the cost of additional GFLOPs due to the additional operations occurring in the layer.

This may mean that the dataset has certain classes that can't be distinguished as well without the use of skipped connectivity to the last FC layer. Both the RVFL and TT-RVFL networks impressively surpass the CELM with skipped connectivity. It's difficult to understand the reasoning for this, but the weights of the convolutional layer are also fully random with 32 filters in this CELM case. It may be that the transformed input is hindering the separability between classes in comparison to the use of the FC layer. However, with skipped connectivity from the input to both FC layers the features are still available for the first FC layer to make use of. This allowed for the CELM with skipped connectivity to both FC layers to surpass the other models.

Although the usage of additional layers and skipped connectivity enable for a greater performance, this comes at the cost of a demand on resources and memory to store the parameters. Similarly to what was seen in the MSTAR results, the deeper

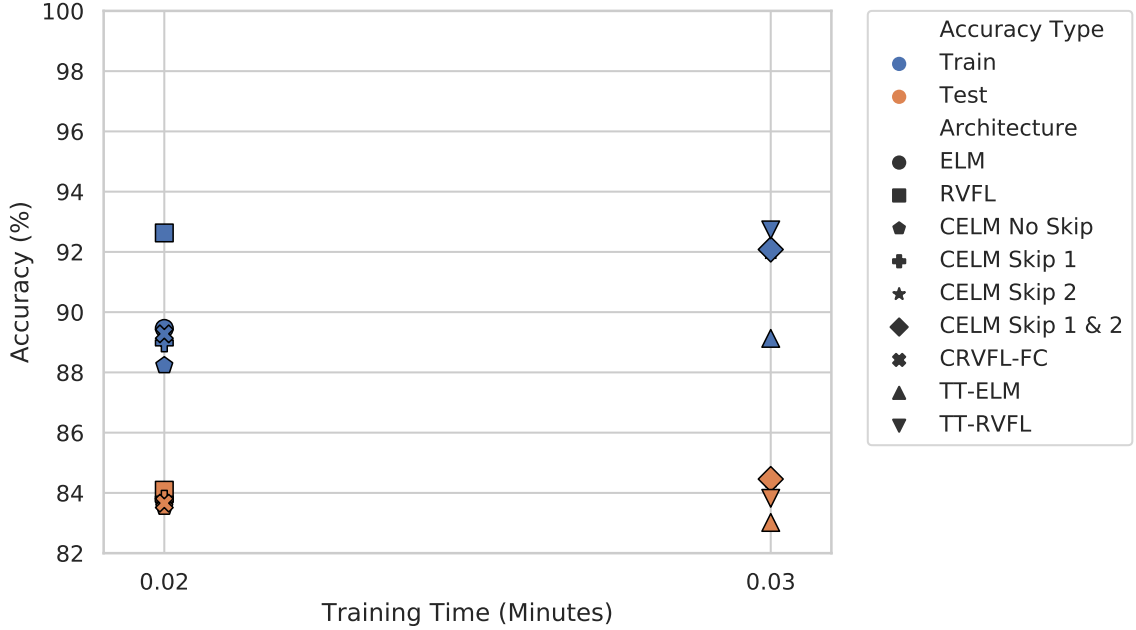


Figure 4.20: Performance of the random projection architectures on the FMNIST dataset. As with the other datasets, the CELM with skipped connectivity to both FC layers achieves the highest performance. However, there is less of a distinguishable gap in this more complex dataset.

architectures utilize more memory to store their parameters, while the ELM, RVFL, and their TT equivalents use trivial amounts.

The last experiment to assess these networks is shown in the results of FMNIST. In contrast to the other two datasets, this dataset showed complexity in its imagery of fashion items leading to testing accuracies below 90%. In addition to this, many of the networks show near comparable accuracies and training times to one another. However, similarly to the previous datasets, the CELM with skipped connectivity to both FC layers achieved the highest test accuracy. Although, this comes at the cost of demand on resources and memory size. The TT-RVFL presents the most favorable results with the smallest memory footprint and near comparable accuracies to the other networks.

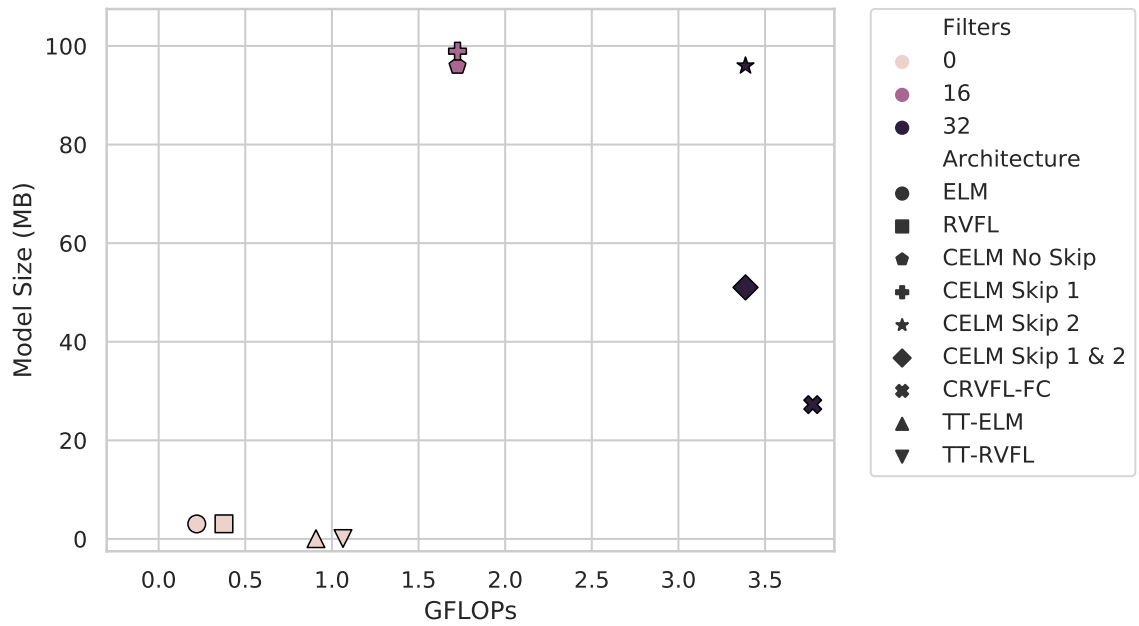


Figure 4.21: Statistics of the random projection architectures on the FMNIST dataset. The architectures utilizing additional layers require more GFLOPs and additional memory to store all of their parameters. The TT networks achieve a smaller model at the cost of additional GFLOPs due to the additional operations occurring in the layer.

Chapter 5

Conclusion & Future Work

This work presents how networks with fixed-random weights can achieve performance akin to their trained equivalent models. Both deep networks with and without transfer learning are evaluated on, where, in both cases, fixed-random weights were shown to enable effective learning due to the architecture. Additionally, smaller, or downsized, versions of the original trained networks are evaluated on to assess how effective the fixed-random weights were to the learning process. From these networks, accuracies comparable to the fixed-random weight models were achieved. However, the fixed-random weighted models show a slight increase over the downsized models. On the other hand, the downsized models were able to use less resources in most cases compared to the fixed-random weighted models. This illustrates how smaller models of the original trained architecture could be more beneficial in terms of the demand on resources and training time. However, this does not discount the effectiveness of fixed-random weights achieving a higher performance over these smaller models. Therefore, the fixed-random weights may be beneficial to alleviate some of the demand on resources in comparison to trained equivalent models, but downsized models may be even more beneficial in more resource constraint applications. Likewise, for transfer learned weights, the use of fixed weights for partial fine-tuning shows performance akin to that of entire fine-tuning of a network except with fewer memory writes occurring as fewer weights had to be updated in this process.

In addition to the experiments discussed above, the ELM random projection network is extended on to illustrate how the use of additional layers and skipped connectivity enables the network to train just as quickly with higher performance. This performance difference, though, does come at the price of a higher demand on resources. However, the tensor decomposed ELM presents how the use of random projection and tensor decomposition can be combined to enable for a fast training network with near comparable accuracies to their original networks, such as the ELM and RVFL. The TT-ELM and TT-RVFL present networks that decrease the memory needed to store the weights of the model by approximately 40x and still enables for a high performance model.

Overall, the main contributions of this research can be summarized as the following. The use of fixed-random weights in networks is shown to not hinder the training process significantly as high performing models can still be obtained with these weights in comparison to downsized models. The use of additional layers and skipped connectivity enable for higher performing models when using the ELM training paradigm. Lastly, the use of TT-FC layers allow for fewer parameters to be utilized in ELMs leading to a decrease in memory size at an insignificant cost in resources and performance. This research also opens up an avenue for gaining a greater understanding of how simple techniques can be utilized to obtain less resource demanding models with faster training times.

5.1 Future Work

Many aspects of the use of random weights can be further extended from this work. In this work, CNNs are the chosen network type to evaluate how fixed-random filters perform in the network’s learning. However, other types of ANNs can be explored, such as recurrent neural networks and generative adversarial networks. Additionally, different datasets can be evaluated on that illustrate a higher complexity or a noisier

input. Likewise with the other types of networks, different tasks can be evaluated on. As previously discussed in Chapter 2, the use of fixed-random weights in neural style transfer and image generation illustrated the intriguing properties of fixed-random weights on these tasks.

In addition to exploring the use of fixed-random weights in other types of networks, the extension of the ELM is discussed in this work. With changes to the architectural design, the base ELM design can be further extended to enable for a higher performing model that can also be lightweight in terms of memory size. TT was used as the tensor decomposition to obtain a lightweight version of the ELM and RVFL architectures. However, many other tensor decompositions exist that can potentially enable for a greater compression and smaller reduction in the performance gap. Additionally, the properties of the tensor decomposition can be explored in the use of other layers aside from only the FC layer explored in this work. With the use of tensor decomposition and fixed-random weights, other concepts, such as pruning, can be further explored to analyze how randomness can potentially help while decreasing the demand on resources and training time.

Bibliography

- [1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.
- [2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2999134.2999257>
- [3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.
- [4] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 2261–2269.
- [5] K. He, Y. Wang, and J. Hopcroft, “A powerful generative model using random weights for the deep image representation,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 631–639. [Online]. Available: <http://papers.nips.cc/paper/6568-a-powerful-generative-model-using-random-weights-for-the-deep-image-representation.pdf>
- [6] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image style transfer using convolutional neural networks,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 2414–2423.
- [7] A. Rosenfeld and J. K. Tsotsos, “Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing,” *CoRR*, vol. abs/1802.00844, 2018. [Online]. Available: <http://arxiv.org/abs/1802.00844>
- [8] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009.
- [9] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
- [10] Y. Yang and S. Newsam, “Bag-of-visual-words and spatial extensions for land-use classification,” in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS

- '10. New York, NY, USA: ACM, 2010, pp. 270–279. [Online]. Available: <http://doi.acm.org/10.1145/1869790.1869829>
- [11] V. Jovanovic and V. Risojevi, “Aggregated color descriptors for land use classification,” *Telfor Journal*, vol. 7, pp. 91–96, 01 2015.
- [12] H. Wang, S. Chen, F. Xu, and Y.-Q. Jin, “Application of deep-learning algorithms to mstar data,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*. IEEE, 2015, pp. 3743–3745.
- [13] Y. LeCun, Fu Jie Huang, and L. Bottou, “Learning methods for generic object recognition with invariance to pose and lighting,” in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 2, June 2004, pp. II–104 Vol.2.
- [14] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *CoRR*, vol. abs/1708.07747, 2017. [Online]. Available: <http://arxiv.org/abs/1708.07747>
- [15] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, “Training deep neural networks with 8-bit floating point numbers,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 7675–7684. [Online]. Available: <http://papers.nips.cc/paper/7994-training-deep-neural-networks-with-8-bit-floating-point-numbers.pdf>
- [16] G. J. Scott, M. R. England, W. A. Starms, R. A. Marcum, and C. H. Davis, “Training deep convolutional neural networks for landcover classification of high-resolution imagery,” *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 4, pp. 549–553, April 2017.
- [17] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2018, pp. 7132–7141.
- [18] B. Singh, M. Najibi, and L. S. Davis, “Sniper: Efficient multi-scale training,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 9310–9320. [Online]. Available: <http://papers.nips.cc/paper/8143-sniper-efficient-multi-scale-training.pdf>
- [19] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO, “Convolutional lstm network: A machine learning approach for precipitation nowcasting,” in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 802–810. [Online]. Available: <http://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting.pdf>

- [20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Online]. Available: <https://doi.org/10.1038/323533a0>
- [21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.
- [22] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, “Imagenet training in minutes,” in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: ACM, 2018, pp. 1:1–1:10. [Online]. Available: <http://doi.acm.org/10.1145/3225058.3225069>
- [23] H. Mikami, H. Suganuma, P. U.-Chupala, Y. Tanaka, and Y. Kageyama, “Imagenet/resnet-50 training in a flash,” *CoRR*, vol. abs/1811.05233, 2018. [Online]. Available: <http://arxiv.org/abs/1811.05233>
- [24] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, “Continual lifelong learning with neural networks: A review,” *Neural Networks*, vol. 113, pp. 54 – 71, 2019. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608019300231>
- [25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: <http://arxiv.org/abs/1704.04861>
- [26] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [27] C. Kyrkou, G. Plastiras, T. Theodoridis, S. I. Venieris, and C. Bouganis, “Dronet: Efficient convolutional neural network detector for real-time uav applications,” in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 967–972.
- [28] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1126–1135.
- [29] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.
- [30] X. Dong, J. Huang, Y. Yang, and S. Yan, “More is less: A more complicated network with less inference complexity,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [31] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *ICML*, 2015.

- [32] R. Anil, G. Pereyra, A. Passos, R. Ormándi, G. E. Dahl, and G. E. Hinton, “Large scale distributed neural network training through online distillation,” *CoRR*, vol. abs/1804.03235, 2018. [Online]. Available: <http://arxiv.org/abs/1804.03235>
- [33] R. Zhao, Y. Hu, J. Dotzel, C. D. Sa, and Z. Zhang, “Improving neural network quantization without retraining using outlier channel splitting,” *CoRR*, vol. abs/1901.09504, 2019.
- [34] X. Dong, S. Chen, and S. Pan, “Learning to prune deep neural networks via layer-wise optimal brain surgeon,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4857–4867. [Online]. Available: <http://papers.nips.cc/paper/7071-learning-to-prune-deep-neural-networks-via-layer-wise-optimal-brain-surgeon.pdf>
- [35] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1135–1143. [Online]. Available: <http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf>
- [36] J.-H. Luo, J. Wu, and W. Lin, “Thinet: A filter level pruning method for deep neural network compression,” 10 2017, pp. 5068–5076.
- [37] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” *arXiv preprint arXiv:1710.09282*, 2017.
- [38] H. Xie, J. Li, and H. Xue, “A survey of dimensionality reduction techniques based on random projection,” *CoRR*, vol. abs/1706.04371, 2017. [Online]. Available: <http://arxiv.org/abs/1706.04371>
- [39] R. Penrose, “A generalized inverse for matrices,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 51, no. 3, p. 406413, 1955.
- [40] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew, “Extreme learning machine: a new learning scheme of feedforward neural networks,” in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, vol. 2, July 2004, pp. 985–990 vol.2.
- [41] H. Jaeger, “The” echo state” approach to analysing and training recurrent neural networks-with an erratum note’,” *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, 01 2001.

- [42] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 593 – 616, 2004. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0022000004000406>
- [43] J. Yu, J. Chen, Z. Q. Xiang, and Y. Zou, "A hybrid convolutional neural networks with extreme learning machine for wce image classification," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec 2015, pp. 1822–1827.
- [44] Y. Zeng, X. Xu, Y. Fang, and K. Zhao, "Traffic sign recognition using extreme learning classifier with deep convolutional features," 2015.
- [45] S. Pang and X. Yang, "Deep convolutional extreme learning machine and its application in handwritten digit classification," *Computational Intelligence and Neuroscience*, vol. 2016, pp. 1–10, 01 2016.
- [46] Y. . Pao and Y. Takefuji, "Functional-link net computing: theory, system architecture, and functionalities," *Computer*, vol. 25, no. 5, pp. 76–79, May 1992.
- [47] L. Zhang and P. Suganthan, "A comprehensive evaluation of random vector functional link networks," *Information Sciences*, vol. 367-368, pp. 1094 – 1105, 2016. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0020025515006799>
- [48] L. Zhang and P. N. Suganthan, "Visual tracking with convolutional random vector functional link network," *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3243–3253, Oct 2017.
- [49] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251 – 257, 1991. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/089360809190009T>
- [50] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.
- [51] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 807–814. [Online]. Available: <http://dl.acm.org/citation.cfm?id=3104322.3104425>
- [52] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec 1989.
- [53] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors,"

- CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: <http://arxiv.org/abs/1207.0580>
- [54] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *International Conference on Learning Representations*, 2015.
- [55] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, March 1994.
- [56] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feed-forward neural networks,” in *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.
- [57] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *ECCV*, 2014.
- [58] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.
- [59] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.
- [60] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?” in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 2146–2153.
- [61] L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, Apr. 2006. [Online]. Available: <https://doi.org/10.1109/TPAMI.2006.79>
- [62] A. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng, “On random weights and unsupervised feature learning,” in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 1089–1096.
- [63] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *CoRR*, vol. abs/1605.07146, 2016.
- [64] Y. Le and X. J. Yang, “Tiny imagenet visual recognition challenge,” 2015.
- [65] G. Huang, G.-B. Huang, S. Song, and K. You, “Trends in extreme learning machines: A review,” *Neural Networks*, vol. 61, pp. 32 – 48, 2015. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608014002214>

- [66] D. Dua and C. Graff, “UCI machine learning repository,” 2017. [Online]. Available: <http://archive.ics.uci.edu/ml>
- [67] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Aug. 2009. [Online]. Available: <http://dx.doi.org/10.1137/07070111X>
- [68] A. C. Bovik, *Handbook of image and video processing*. Academic press, 2010.
- [69] K. Sayood, *Introduction to data compression*. Morgan Kaufmann, 2017.
- [70] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.
- [71] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, “Tensor decomposition for signal processing and machine learning,” *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, July 2017.
- [72] J. Hstad, “Tensor rank is np-complete,” *Journal of Algorithms*, vol. 11, no. 4, pp. 644 – 654, 1990. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/0196677490900146>
- [73] C. J. Hillar and L. Lim, “Most tensor problems are NP hard,” *CoRR*, vol. abs/0911.1393, 2009. [Online]. Available: <http://arxiv.org/abs/0911.1393>
- [74] F. L. Hitchcock, “The expression of a tensor or a polyadic as a sum of products,” *J. Math. Phys*, vol. 6, no. 1, pp. 164–189, 1927.
- [75] J. D. Carroll and J.-J. Chang, “Analysis of individual differences in multidimensional scaling via an n-way generalization of “eckart-young” decomposition,” *Psychometrika*, vol. 35, no. 3, pp. 283–319, Sep 1970. [Online]. Available: <https://doi.org/10.1007/BF02310791>
- [76] R. A. Harshman, “Foundations of the parafac procedure: Models and conditions for an ”explanatory” multimodal factor analysis,” 1970.
- [77] S. Rabanser, O. Shchur, and S. Günnemann, “Introduction to tensor decompositions and their applications in machine learning,” *arXiv preprint arXiv:1711.10781*, 2017.
- [78] L. R. Tucker, “Some mathematical notes on three-mode factor analysis,” *Psychometrika*, vol. 31, no. 3, pp. 279–311, Sep 1966. [Online]. Available: <https://doi.org/10.1007/BF02289464>
- [79] I. V. Oseledets and E. E. Tyrtysnikov, “Breaking the curse of dimensionality, or how to use svd in many dimensions,” *SIAM J. Scientific Computing*, vol. 31, pp. 3744–3759, 2009.

- [80] W. Hackbusch and S. Kühn, “A new scheme for the tensor representation,” *Journal of Fourier Analysis and Applications*, vol. 15, no. 5, pp. 706–722, Oct 2009. [Online]. Available: <https://doi.org/10.1007/s00041-009-9094-9>
- [81] T. G. Kolda, “Multilinear operators for higher-order decompositions.” 4 2006.
- [82] I. V. Oseledets, “Tensor-train decomposition,” *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, Sep. 2011. [Online]. Available: <http://dx.doi.org/10.1137/090752286>
- [83] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [84] E. R. Keydel, S. W. Lee, and J. T. Moore, “Mstar extended operating conditions: A tutorial,” in *Algorithms for Synthetic Aperture Radar Imagery III*, vol. 2757. International Society for Optics and Photonics, 1996, pp. 228–243.
- [85] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, “Regularization of neural networks using dropconnect,” in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1058–1066. [Online]. Available: <http://proceedings.mlr.press/v28/wan13.html>
- [86] A. Novikov, D. Podoprikin, A. Osokin, and D. Vetrov, “Tensorizing neural networks,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’15. Cambridge, MA, USA: MIT Press, 2015, pp. 442–450. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2969239.2969289>
- [87] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.
- [88] R. W. Farebrother, *Linear Least Squares Computations*. New York, NY, USA: Marcel Dekker, Inc., 1988.
- [89] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” in *NIPS-W*, 2017.

5.2 Appendix

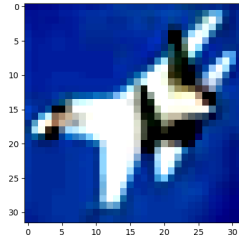


Figure 5.1: Image of an airplane from CIFAR-10.

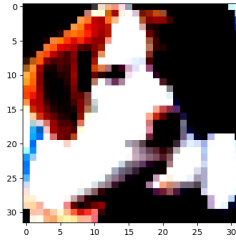


Figure 5.2: Image of a dog from CIFAR-10.

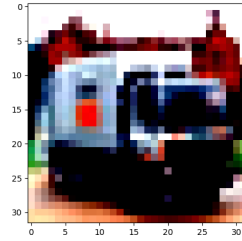


Figure 5.3: Image of a truck from CIFAR-10.









































CIFAR-10			CLASS: AIRPLANE			RESNET-20					
LAYER	Fixed-Random	Trained	LAYER	Fixed-Random	Trained	LAYER	Fixed-Random	Trained	LAYER	Fixed-Random	Trained
1			6			11			16		
2			7			12			17		
3			8			13			18		
4			9			14			19		
5			10			15			20		

Figure 5.4: Activated outputs of each convolutional layer in ResNet-20 on the input image of an airplane from CIFAR-10. Features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

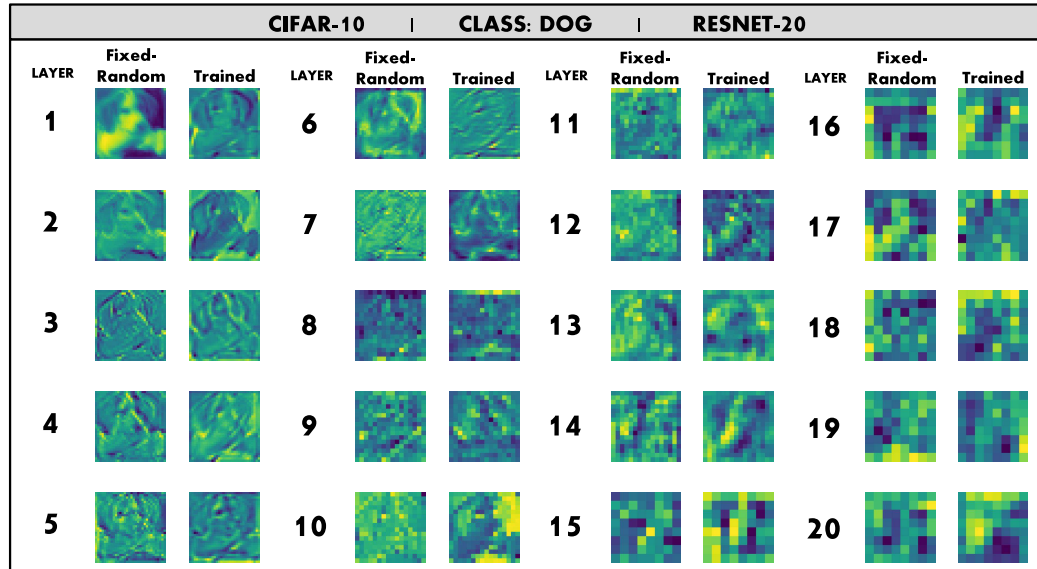


Figure 5.5: Activated outputs of each convolutional layer in ResNet-20 on the input image of a dog from CIFAR-10. Similarly to the frog, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

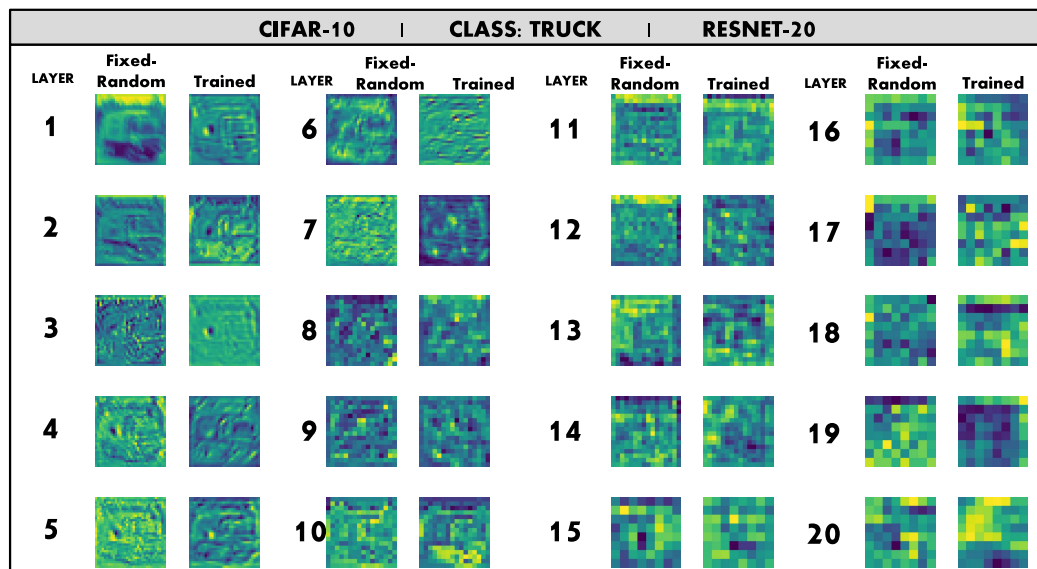


Figure 5.6: Activated outputs of each convolutional layer in ResNet-20 on the input image of a truck from CIFAR-10. Similarly to the frog, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

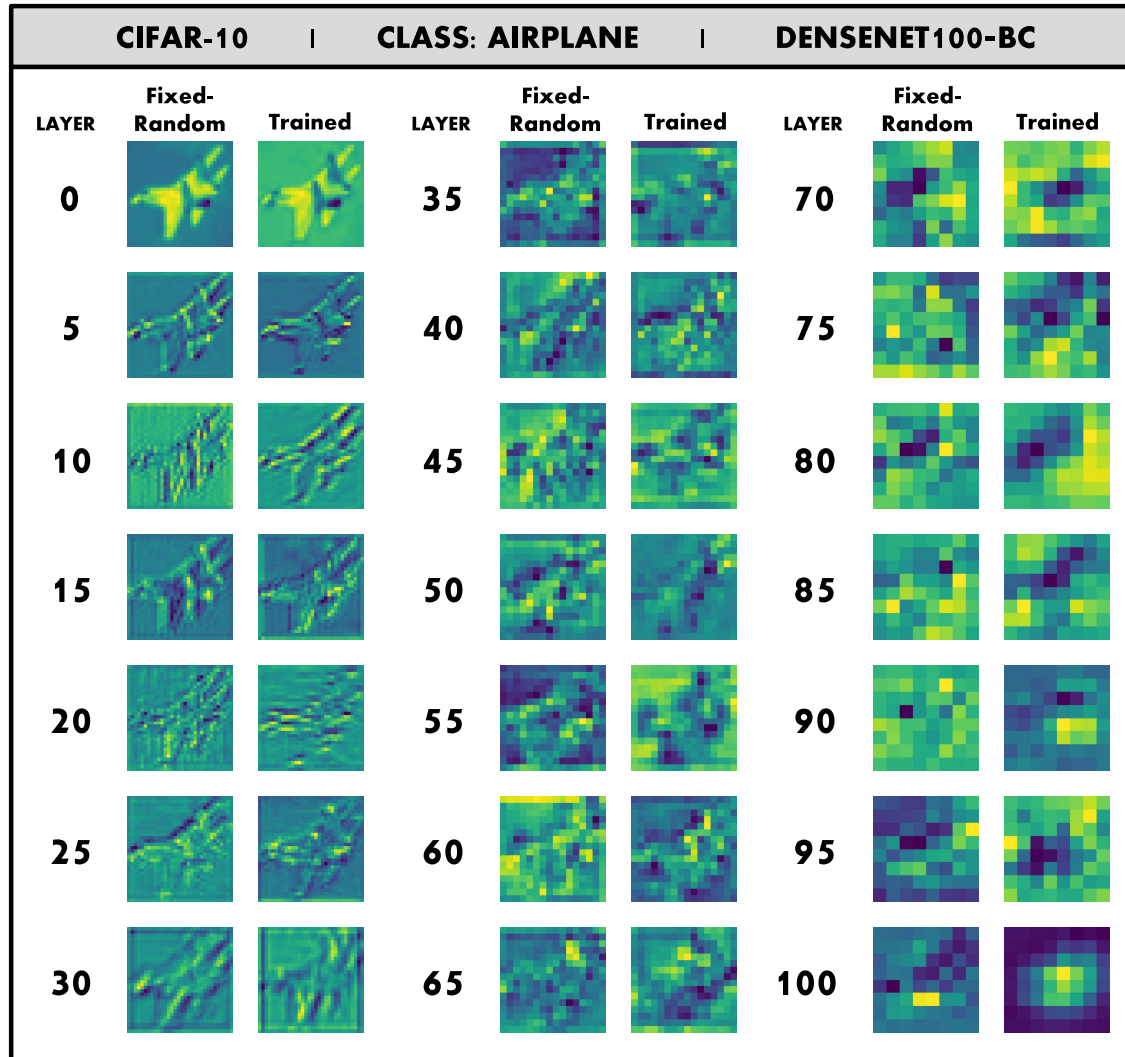


Figure 5.7: Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of an airplane from CIFAR-10. Similarly to the other activated images, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

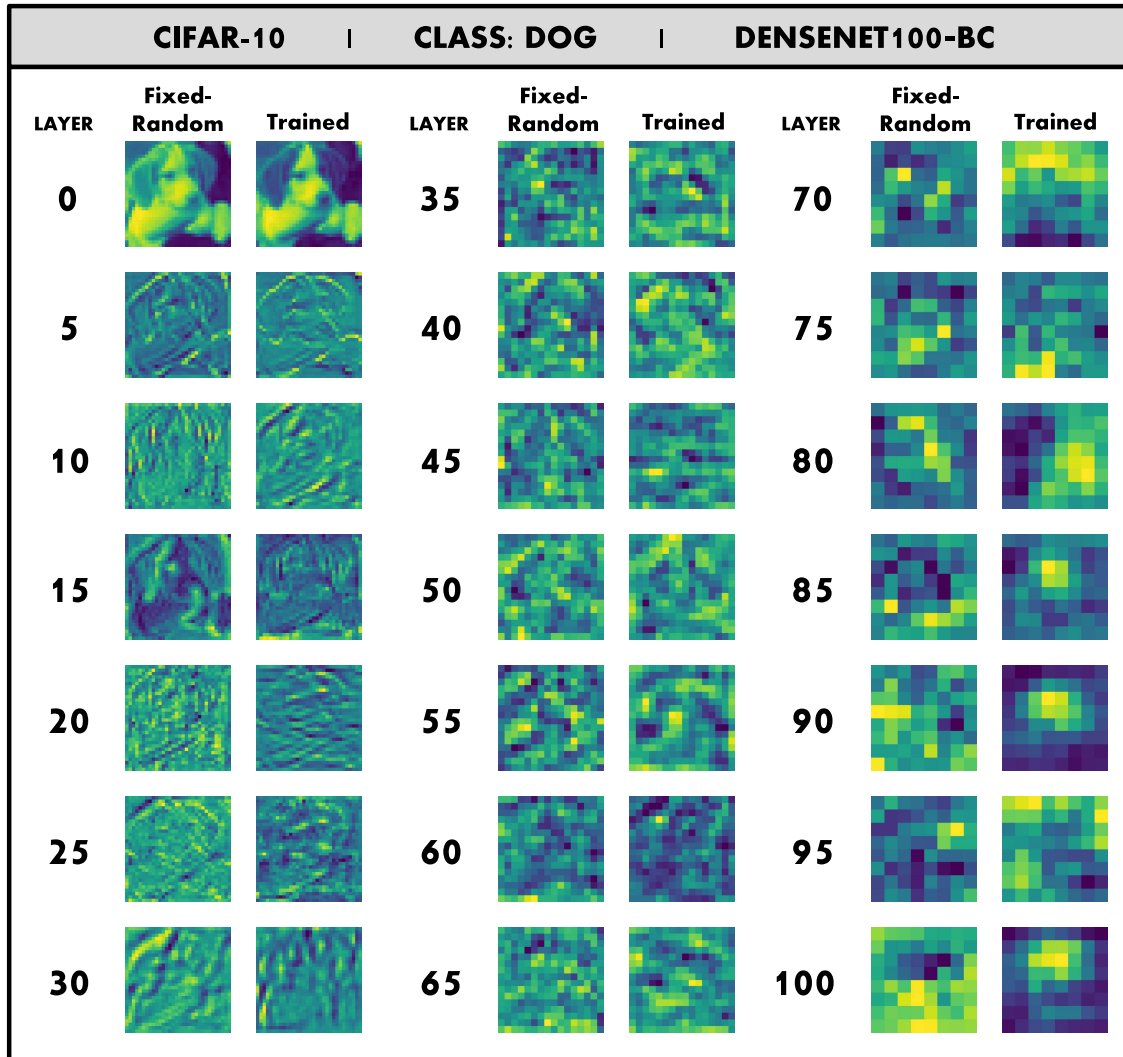


Figure 5.8: Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of a dog from CIFAR-10. Similarly to the other activated images, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.

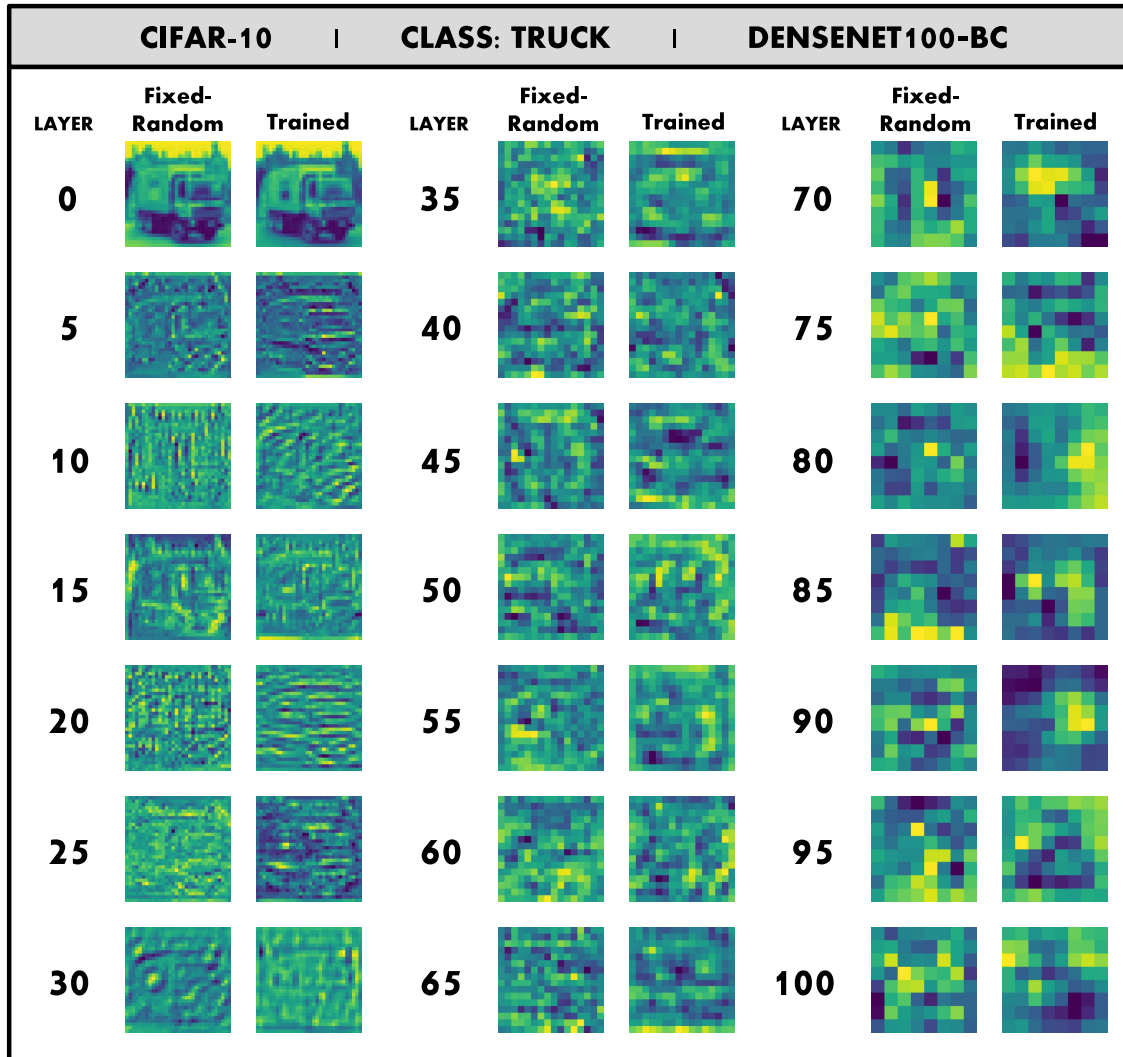


Figure 5.9: Activated outputs of every 5 convolutional layers of DenseNet100-BC on the input image of a truck from CIFAR-10. Similarly to the other activated images, features can be seen to be more distinguishable in the early layers as opposed to the latter layers of the architecture.