

2023

An Investigation of Methods for Improving Spatial Invariance of Convolutional Neural Networks for Image Classification

David Noel

Nova Southeastern University, dn632@mynsu.nova.edu

Follow this and additional works at: https://nsuworks.nova.edu/gscis_etd



Part of the [Computer Sciences Commons](#), and the [Data Science Commons](#)

All rights reserved. This publication is intended for use solely by faculty, students, and staff of Nova Southeastern University. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, now known or later developed, including but not limited to photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the author or the publisher.

NSUWorks Citation

David Noel. 2023. *An Investigation of Methods for Improving Spatial Invariance of Convolutional Neural Networks for Image Classification*. Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, College of Computing and Engineering. (1185)
https://nsuworks.nova.edu/gscis_etd/1185.

This Dissertation is brought to you by the College of Computing and Engineering at NSUWorks. It has been accepted for inclusion in CCE Theses and Dissertations by an authorized administrator of NSUWorks. For more information, please contact nsuworks@nova.edu.

An Investigation of Methods for Improving Spatial Invariance of
Convolutional Neural Networks for Image Classification

by

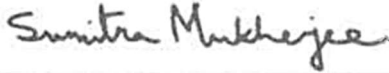
David Noel

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Computer Science

College of Computing and Engineering
Nova Southeastern University

2023

**We hereby certify that this dissertation, submitted by David Noel
Conforms to acceptable standards and is fully adequate in scope and
quality to fulfill the dissertation requirements for the degree of
Doctor of Philosophy.**



Sumitra Mukherjee, Ph.D.
Chairperson of Dissertation Committee

6/26/23
Date



Michael J. Laszlo, Ph.D.
Dissertation Committee Member

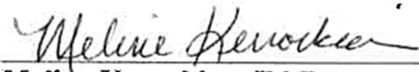
6/26/23
Date



Francisco J. Mitropoulos, Ph.D.
Dissertation Committee Member

6/26/23
Date

Approved:



Meline Kevorkian, Ed.D.
Dean, College of Computing and Engineering

6/26/23
Date

**College of Computing and Engineering
Nova Southeastern University**

2023

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

An Investigation of Methods for Improving Spatial Invariance of Convolutional Neural Networks for Image Classification

by
David Noel
May 2023

Convolutional Neural Networks (CNNs) have achieved impressive results on complex visual tasks such as image recognition. They are commonly assumed to be spatially invariant to small transformations of their input images. Spatial invariance is a fundamental property that characterizes how a model reacts to input transformations, i.e., its *generalizability* - and deep networks that can robustly classify objects placed in different orientations or lighting conditions have the property of invariance. However, several authors have recently shown that this is not the case, and that slight rotations, translations, or rescaling of their input images significantly reduce the network's predictive accuracy. Furthermore, incorrectly classified images can have disastrous consequences, such as fatalities from self-driving cars or raising concerns about racial discrimination.

Data augmentation is a mainstream technique used to improve invariance in CNNs by artificially increasing the amount of training data by generating new and diverse data points with additional properties extracted from the existing dataset. This research aimed to provide a rigorous comparative analysis of two novel data augmentation techniques used to improve spatial invariance and reduce overfitting in CNNs: RandAugment (a stochastic technique that applies geometric transforms to images) and Conditional Generative Adversarial Networks (generative models that create synthetic samples).

This work finetuned pre-trained ResNet50 and InceptionV3 networks on each augmentation method. It evaluated and compared combinations of these networks across a base model (NoelNet) developed for this work, using three benchmark image data sets taken from different perspectives: MNIST, FMNIST, and CIFAR-10 using the following metrics: training/validation loss, training/validation accuracy, test accuracy, precision, recall, and training latency. The experimental setup for each analysis was guided by five policies: Policy 1 (no data augmentation), Policy 2 (augmentation using RandAugment), Policy 3 (augmentation using GAN), Policy 4 (combine GAN generated samples with non-augmented samples), Policy 5 (apply RandAugment to GAN generated samples).

The main contribution of this work was to demonstrate how data augmentation improves the classification performance of modern CNNs by using Generative Adversarial Networks and Reinforcement Learning to increase their spatial invariance and minimize overfitting.

Acknowledgments

I am sincerely thankful to my advisor, Dr. Sumitra Mukherjee. His steady guidance, mentorship, and constructive feedback have been invaluable in shaping my research and academic growth. In our first meeting, he advised that the purpose of a Ph.D. is to inform and not impress. This wise advice has grounded me throughout this process. To the rest of the dissertation committee, Dr. Michael Laszlo, and Dr. Frank Mitropoulos, thank you for your insights, time, and valuable feedback in reviewing my work.

To my parents, thank you for your unwavering support throughout my academic journey. Your constant encouragement, guidance, and sacrifices have been instrumental in shaping me into the person I am today.

I am also thankful to my wife, Shari Taylor Noel, for her steady support and understanding during the challenging times of my Ph.D. journey. Her patience, encouragement, and sacrifices have been a source of strength and motivation for me.

I am especially grateful to my 4-year-old son, Liam Noel, who has been a constant source of inspiration, joy, and laughter throughout this journey. His innocent presence, curious mind, and playful nature have reminded me of what truly matters in life.

Finally, I thank my colleagues, friends, and loved ones who have supported and encouraged me throughout this endeavor. Your kindness and encouragement have meant a great deal to me.

Table of Contents

Abstract	iii
Acknowledgments	iv
List of Tables	vii
List of Figures	viii

Chapters

1. Introduction	1
Problem Statement	3
Dissertation Goal	5
Relevance and Significance	9
Barriers and Issues	12
Assumptions, Limitations, and Delimitations	13
Definition of Terms	14
List of Acronyms	21
Summary	22
2. Literature Review	24
Neural Networks	24
Deep Neural Networks	29
Deep Learning	30
Convolutional Neural Networks	31
Spatial Invariance in Convolutional Neural Networks	39
Data Augmentation	44
Related Works	49
RandAugment	52
Generative Adversarial Networks	53
Network Architectures	55
Summary	61
3. Methodology	64
Models	64
Optimization	67
Experimental Design	68
Evaluation Metrics	73
Experiments	75
Data Analysis	77
Results Format	78
Resource Requirements	78
Summary	79
4. Results	81
Conditional DCGAN Training	82
MNIST on NoelNet	85

Fashion MNIST on NoelNet	93
CIFAR-10 on NoelNet	101
MNIST on ResNet-50	108
Fashion MNIST on ResNet-50	116
CIFAR-10 on ResNet-50	123
MNIST on InceptionV3	130
Fashion MNIST on InceptionV3	139
CIFAR-10 on InceptionV3	146
Dataset Latency Across Different Models	155
Dataset Accuracy Across Different Models	157
Data Analysis	159
Summary	164
5. Conclusions, Implications, Recommendations, and Summary	172
Conclusions	172
Implications	174
Recommendations	175
Summary	176
Appendices	
A. Algorithms	189
B. Hyperparameters	194
C. Third-Party Libraries	196
D. Datasets	199
E. Architectures	202
F. Experimental Results	209
G. Dissertation Source Code	213
References	216

List of Tables

Tables

1. Training Latencies for the Conditional DCGAN 84
2. Performance Test Metrics of each Policy for MNIST on NoelNet 86
3. Performance Test Metrics of each Policy for FMNIST on NoelNet 94
4. Performance Test Metrics of each Policy for CIFAR-10 on NoelNet 102
5. Performance Test Metrics of each Policy for MNIST on ResNet-50 86
6. Performance Test Metrics of each Policy for FMNIST on ResNet-50 95
7. Performance Test Metrics of each Policy for CIFAR-10 on ResNet-50 104
8. Performance Test Metrics of each Policy for MNIST on InceptionV3 107
9. Performance Test Metrics of each Policy for FMNIST on InceptionV3 121
10. Performance Test Metrics of each Policy for CIFAR-10 on InceptionV3 130
- B1. Hyperparameters That Were Used Throughout This Dissertation 177
- D1. Type of Dataset Used by Each of the Dissertation Experiments 182
- D2. FashionMNIST Classes 182
- D3. CIFAR-10 Classes 182
- D4. MNIST Classes 182
- F1. Summary of Policies and Expected Metrics Relative to the Baseline 193
- F2. Summary of Policies and Actual Metrics Relative to the Baseline 194

List of Figures

Figures

1. Illustration of the Model of a Neuron and a Three-Layer Neural Network 25
2. The Architecture of Lenet-5 CNN Developed to Classify Hand-Written Digits 33
3. Example of a Convolution in the First Hidden Layer of the Lenet-5 ConvNet 34
4. A Max Pooling Operation With 2x2 Filters and a Stride of 2 38
5. Jagged Predictions of CNNs Caused by One-Pixel Perturbations in Input Images 42
6. Graphs Showing Signs of Overfitting Using Training and Validation Loss 45
7. Gap Between Training and Validation Accuracy Denotes Amount of Overfitting 46
8. The Architecture of a Simple Conditional Generative Adversarial Network 54
9. The Residual Blocks of ResNet34 and ResNet50, Respectively 56
10. The ResNet Architecture Inspired by the VGG-19 Model 57
11. The Inception Modules Used by GoogleNet and InceptionV3 59
12. The GoogleNet (InceptionV1) Convolutional Network 60
13. Sample Images of Unaugmented, Augmented, and GAN-Generated MNIST 85
14. Accuracy and Loss Graphs of each Policy for MNIST on NoelNet 87
15. Loss and Validation Accuracy of All Policies for MNIST on NoelNet 91
16. Training Latency of All Polices for MNIST on NoelNet 92
17. Sample Images of Unaugmented, Augmented, and GAN-Generated FMNIST 95
18. Accuracy and Loss Graphs of each Policy for FMNIST on NoelNet 89
19. Loss and Validation Accuracy of All Policies for FMNIST on NoelNet 99
20. Training Latency of All Polices for FMNIST on NoelNet 100
21. Sample Images of Unaugmented, Augmented, and GAN-Generated CIFAR-10 102

22. Accuracy and Loss Graphs of each Policy for CIFAR-10 on NoelNet	104
23. Loss and Validation Accuracy of All Policies for CIFAR-10 on NoelNet	107
24. Training Latency of All Polices for CIFAR-10 on NoelNet	108
25. Accuracy and Loss Graphs of each Policy for MNIST on ResNet-50	87
26. Training Loss and Validation Accuracy of All Policies for MNIST on ResNet-50	91
27. Training Latency of All Polices for MNIST on ResNet-50	92
28. Accuracy and Loss Graphs of each Policy for FMNIST on ResNet-50	97
29. Loss and Validation Accuracy of All Policies for FMNIST on ResNet-50	100
30. Training Latency of All Polices for FMNIST on ResNet-50	101
31. Accuracy and Loss Graphs of each Policy for CIFAR-10 on ResNet-50	105
32. Loss and Validation Accuracy of All Policies for CIFAR-10 on ResNet-50	108
33. Training Latency of All Polices for CIFAR-10 on ResNet-50	109
34. Accuracy and Loss Graphs of each Policy for MNIST on InceptionV3	114
35. Loss and Validation Accuracy of All Policies for MNIST on InceptionV3	118
36. Training Latency of All Polices for MNIST on InceptionV3	119
37. Accuracy and Loss Graphs of each Policy for FMNIST on InceptionV3	123
38. Loss and Validation Accuracy of All Policies for FMNIST on InceptionV3	126
39. Training Latency of All Polices for FMNIST on InceptionV3	127
40. Accuracy and Loss Graphs of each Policy for CIFAR-10 on InceptionV3	131
41. Loss and Validation Accuracy of All Policies for CIFAR-10 on InceptionV3	135
42. Training Latency of All Polices for CIFAR-10 on InceptionV3	136
43. Comparing Dataset Training Times Across Different Models	138
44. Comparing Dataset Accuracy Across Different Models	140

A1. Adaptive Moment Estimation (Adam) Algorithm	172
A2. RandAugment Algorithm in Python	175
A3. Algorithm to Partition a Dataset Into Training, Validation, and Test Splits	175
A4. Algorithm to Resize Image and One-Hot Encode Label	176
A5. Algorithm to Generate a Performant Data Input Pipeline for TensorFlow	176
D1. Sample Images From (a) MNIST, (b) FMNIST, and (c) CIFAR-10	183
D2. Sample Images at Various Epochs Generated Using the Conditional DCGAN	183
D3. CDCGAN Images and Loss at Epoch 1000 (a) MNIST, (b) FMNIST, and (c) CIFAR-10	184
E1. The Conditional GAN Architecture Used: (a) Discriminator, (b) Generator	185
E2. Architectural Components of Various ResNet ConvNets	186
E3. Model of ResNet50 Used for This Dissertation	187
E4. Model of InceptionV3 Used for This Dissertation	188
E5. Data Input Pipeline Used by Each Policy to Evaluate Their Respective Datasets	189

Chapter 1

Introduction

This dissertation investigated methods for improving the spatial invariance of Convolutional Neural Networks (ConvNets, or CNNs; LeCun et al., 1989, 1998) for image classification using advanced data augmentation techniques. Image classification has been a critical focus of research in computer vision for at least two decades. There have been some notable methods used to represent images so that they can be classified, such as Scale Invariant Feature Change (SIFT; Lowe, 2004), Histogram of Oriented (HOG; Dalal & Triggs, 2005), and Fisher Vectors (Sánchez et al., 2013). The latest generation is Convolutional Neural Networks. ConvNets are specialized deep neural networks that aim to learn feature representations of their input instead of relying on hand-engineered features generated using methods like SIFT, HOG, and Fisher Vectors. They have achieved impressive results on many complex visual tasks such as image recognition, semantic segmentation, image captioning, video captioning, and action recognition, amongst others (Antoniou et al., 2016; Jaderberg et al., 2016; Krizhevsky et al., 2012). This success is because CNNs exploit the natural translational symmetries in data through weight sharing and translation invariance.

In this work, *spatial invariance* is a highly desirable property of CNNs in which the network can disentangle the poses and deformations of objects from their texture and shape (Jaderberg et al., 2016). It is a crucial property of image representation where the objects in an image are invariant to viewpoint changes (Lenc & Vedaldi, 2015). In this work, viewpoint represents one's perspective of a 3D object. In contrast, image transformation represents geometric or color changes to an image, such as rotation,

translation, and brightness changes. For example, the image of a cat rotated by 10 degrees should still be a cat. According to Goodfellow et al. (2009), a deep network that can robustly classify objects despite the objects being placed in different orientations or lighting conditions is said to have the property of invariance. They further state that invariance implies a selectivity for complex, high-level input features but robustness to irrelevant input transformations, suggesting the balance between selectivity and robustness makes learning invariant features paramount but nontrivial. Regarding image recognition, an invariant feature should only respond to one stimulus despite changes in its translation, scale, rotation, illumination, or other geometric properties.

Despite their impressive performance on various computer vision tasks, several authors have demonstrated that modern ConvNets are surprisingly brittle such that slight perturbations in their input can result in large changes in the accuracy of their predicted output. For instance, Azulay & Weiss (2018) showed a 30% change in the accuracy of predictions made by a CNN when its input was shifted by one pixel. The latter can result in severe consequences in systems using CNNs, such as autonomous vehicles or facial recognition systems where reliability, security, and unbiasedness are critical. Data augmentation and architectural modifications are the most common techniques to imbue spatial invariance to ConvNets (Shorten & Khoshgoftaar, 2019). Data augmentation artificially increases the size of the training dataset using affine or elastic transforms such as rotations, translations, and gaussian noise. The intuition for doing so is that this creates more diverse samples, thus greater sources of invariances which the CNN can then learn during training (Krizhevsky et al., 2012). According to Shorten & Khoshgoftaar (2019), data augmentation imbues invariances within the data, forgoing the laborious process of

hand-engineering features, manual geometric transforms, or inflexible mutations of network architecture. In contrast, architectural modifications modify the CNN's convolution layers and other internal structures using techniques such as dropout (Srivastava et al., 2014) and Batch normalization (Ioffe & Szegedy, 2015). Furthermore, architectural techniques assume significant knowledge of invariances a priori.

As such, this dissertation focused on data augmentation using Generative Adversarial Networks (GANs; Goodfellow et al., 2014) and RandAugment (Cubuk et al., 2020) to improve the spatial invariance of Convolutional Networks. GANs use generative modeling to synthesize new and diverse training samples. In contrast, RandAugment uses reinforcement learning to stochastically select and apply up to 14 different geometric and color transform to each training sample. This research conducted 45 experiments using five augmentation strategies called *policies* applied to three benchmark datasets: MNIST (LeCun et al., 1998), FMNIST (Xiao et al., 2017), CIFAR-10 (Krizhevsky & Hinton, 2009). In addition, the datasets were used to train a base ConvNet called NoelNet, and two advanced ConvNets: ResNet50 (He et al., 2015) and InceptionV3 (Szegedy et al., 2015), whose performance was analyzed using training/validation loss, training/validation accuracy, test accuracy, precision, recall, f1 score, and training latency. Finally, a rigorous comparative analysis was conducted on how advanced data augmentation techniques can improve spatial invariance in ConvNets.

Problem Statement

Despite their “superhuman” performance on object recognition (He et al., 2015), CNNs suffer from a significant problem: they are invariant to some but not all input transformations (Azulay & Weiss, 2018; Chen et al., 2020; Engstrom et al., 2018;

Jaderberg et al., 2016; Szegedy et al., 2014). Lack of spatial invariance in CNNs, even by minimal amounts, can lead to unfortunate consequences such as fatalities from autonomous vehicles or racial bias from image recognition systems (Kendall & Gal, 2017). This behavior is an inductive bias of CNNs that dates to the “Neocognitron,” the predecessor to modern CNNs (Fukushima & Miyake, 1982).

The local max-pooling layers within CNNs have helped to satisfy spatial invariance somewhat. However, due to their small spatial support for max-pooling (e.g., 2x2 pixels) and convolutions (e.g., 9x9 kernels), this invariance is only achieved over very deep hierarchies of pooling and convolutions (Jaderberg et al., 2016). Moreover, the intermediate feature maps in CNNs are not invariant to significant transformations of their input data (Lenc & Vedaldi, 2015). This limitation of CNNs is due to their predefined and inadequate pooling mechanisms for handling variations in the spatial arrangement of data. This results in CNNs exhibiting poor invariance to spatial transformations (e.g., rotations, flips, scaling, et cetera) of their input images (Jaderberg et al., 2016). For example, Bowles et al. (2018) showed that CNNs could not easily learn rotationally invariant features without sufficient examples at different rotations in the training data.

Convolutional networks contain multiple non-linear hidden layers between their inputs and outputs, making them highly expressive models capable of learning complicated relationships. The weights of these hidden layers adapt to incoming stimuli, which enables the CNN to learn feature detectors and predict the correct output when given an input vector (Srivastava et al., 2014). However, with limited training data, many of these relationships result from sampling noise and random fluctuations leading to the

problem of overfitting, a phenomenon where a network learns a function with very high variance such that it models its training data perfectly (Krizhevsky et al., 2012; Shorten & Khoshgoftaar, 2019). Overfitting indicates poor invariance and leads to poor generalization on the test set, where the model performs poorly on unseen data. Most state-of-the-art applications of CNNs use large datasets (millions of images, e.g., ImageNet) in their training. In realistic settings, however, many tasks have only limited datasets (tens of thousands, e.g., CIFAR-10). In these cases, CNNs fall short because of the problem of overfitting (Antoniou et al., 2016).

Objects in realistic settings exhibit considerable variability, so learning to recognize them requires more extensive training samples (Azulay & Weiss, 2018; Shorten & Khoshgoftaar, 2019). Because of this limited generalization ability, the long tail of machine learning problems remains inaccessible. For instance, collecting large expert-annotated datasets for every imaging device or patient population in medical imaging and computer-aided diagnosis can be costly and legally prohibitive, rendering many niche applications uneconomical or impractical.

Dissertation Goal

Improving the generalizability of deep convolutional networks by increasing invariance is a nontrivial challenge (Antoniou et al., 2016; Goodfellow et al., 2009; Immer et al., 2022). Data augmentation is one of the most effective and powerful methods of achieving this (Cubuk et al., 2020; Krizhevsky et al., 2012; Shorten & Khoshgoftaar, 2019). According to these authors, data augmentation approximates the data probability space by artificially increasing the size and diversity of the training set with the assumption that more information can be extracted from the existing dataset.

Data augmentation is done either by data warping (label-preserving image perturbations in data space such as rotations) or oversampling (synthetic images are created and added to the training set).

Chen et al. (2020) state that data augmentation teaches invariance to a CNN and proves that it leads to variance reduction. This reduction in variance results in improved classification accuracy (Perez & Wang, 2017) and increased generalization across novel distributions of examples by acting as a regularizer (Arjovsky et al., 2020). Data augmentation forms a critical part of modern deep-learning pipelines. It is typically necessary to achieve state-of-the-art performance, e.g., in AlexNet (Krizhevsky et al., 2012) for image classification or in the medical field for Liver Lesion classification (Frid-Adar et al., 2018).

Several other regularization strategies have been developed to reduce overfitting and improve the invariance of CNNs, such as Dropout (Srivastava et al., 2014) and Batch Normalization (Shorten & Khoshgoftaar, 2019). However, these strategies focus on the model's architecture, leading to increasingly more complex networks from AlexNet (Krizhevsky et al., 2012) to InceptionV3 (Szegedy et al., 2015) because they try to build invariances and equivariances directly into the models. While they have drastically increased data efficiency and generalization, they require knowing the invariances a priori (Immer et al., 2022).

In contrast, data augmentation addresses the problem of overfitting and poor invariance from the root: the training dataset. In several discriminative examples like cat vs. dog, the image classifier must first overcome issues such as scale, viewpoints, occlusion, lighting, background, et cetera. Data augmentation aims to imbue these

invariances into the dataset so that resulting classifiers learn them during training - *instead of guessing what they are during model architecture* - and have robust performance despite these challenges (Shorten & Khoshgoftaar, 2019; Arjovsky et al., 2020).

This research aimed not to produce state-of-the-art results but to provide a rigorous comparative analysis of advanced data augmentation techniques to improve invariance and reduce overfitting in CNNs. First, we used two advanced augmentation techniques: RandAugment (Cubuk et al., 2020) and Conditional Generative Adversarial Networks (CGANs; Mirza & Osindero, 2014) to learn invariances from three baseline datasets: MNIST, FMNIST, CIFAR-10. Second, their performance was evaluated on NoelNet to establish baseline metrics that were used for subsequent and comparative analysis. NoelNet is a basic CNN the author developed, which obtained competitive accuracies and f1 scores on each dataset (at least 0.99 for MNIST, 0.92 for FMNIST, and 0.85 for CIFAR-10) without data augmentation. Finally, on two award-winning classifiers: ResNet50 (He et al., 2015) and InceptionV3 (Szegedy et al., 2015), using the following metrics: training/validation loss, training/validation accuracy, test accuracy, precision, recall, and training latency. The results obtained from ResNet50 and InceptionV3 were presented and analyzed to determine if similar patterns were observed relative to NoelNet.

This work sets itself apart because most prior methods employ simple augmentations, including cropping and flipping (Krizhevsky et al., 2012). However, we hypothesize that more advanced data augmentation techniques lead to stronger augmentations. This intuition comes from the fact that more advanced techniques generate more realistic

images with greater diversity since they can make more extensive modifications to the input images without changing their label (Xie et al., 2020).

Many studies, like those of (He et al., 2015; Krizhevsky et al., 2012), recommend performing data augmentations using affine transformations (e.g., rotation, scaling, and translation). While effective, they are time-consuming and require much expertise and effort to design policies that capture knowledge in each domain. RandAugment resolves these issues because it is a label-preserving stochastic augmentation technique that applies geometric transforms to images without requiring the transform to be learned beforehand. Instead, it randomly selects up to 14 affine transformations and applies them to each sample. As a result, it achieved state-of-the-art image classification results that are uniform across tasks and datasets (*85% accuracy on ImageNet, a 0.6% increase over the previous state-of-the-art*), matching or surpassing all previous methods. In addition, this random distribution of transforms makes learning trends in noise almost impossible, thus reducing overfitting and improving invariance.

Standard data augmentation techniques use a minimal set of known invariances that are easy to invoke. However, Antoniou et al. (2016) recognized that we could learn a model of a much larger invariance space by training a CGAN in a source domain, then applying it in a low-data target domain to synthesize training examples. These networks can generate new training data that results in better-performing classifiers, even after standard data augmentation. First proposed by Goodfellow et al. (2014), GANs represent classes of neural networks that learn to generate synthetic examples with the same characteristics as their training distribution.

According to Bowles et al. (2018), GANs offer a unique way to unlock additional information from a dataset by generating synthetic samples with the appearance of actual images but different properties than the data you already had. A significant advantage of using GANs for data augmentation is that they take away the user's many "handcrafted" decisions. An ideal GAN transforms the discrete distribution of training samples into a continuous distribution by applying augmentation to each source of variances within the dataset. For example, given enough training data at different orientations, a GAN automatically learns to produce samples at any orientation, replicating rotation augmentation across 360 degrees. Works by Frid-Adar et al. (2018) and Bowles et al. (2018) have achieved excellent results using GAN-generated samples.

Relevance and Significance

Modern CNNs used for image classification are commonly assumed to be invariant to small image transformations because of their convolutional architecture or because they were trained using data augmentation. Several authors, however, have shown that this is not the case because small transformations of the input image can significantly deteriorate the accuracy of the network's predictions (Azulay & Weiss, 2018; Shorten & Khoshgoftaar, 2019), which can have disastrous consequences. For example, May 2016 marked the first fatality from an automated driving system caused when the perception module confused the white side of a trailer for bright blue. In another example, an image classification system from Google erroneously identified two African Americans as gorillas, raising significant concerns about racial discrimination. If the classifiers in both cases had greater invariances, these two unfortunate incidents might have been averted (Kendall & Gal, 2017) and serve as the motivation for this dissertation.

The ability to learn symmetries is a foundational property of intelligent systems. Humans can discover patterns and regularities in data that provide representations of reality, such as translation, rotation, intensity, or scale symmetries (Fukushima & Miyake, 1982). Deep convolutional networks seek to replicate such behavior and have achieved impressive results in many areas, such as image recognition, semantic segmentation, image captioning, and video captioning (Shorten & Khoshgoftaar, 2019).

This success is in part due to the *inductive bias* of the network being used. Azulay & Weiss (2018) have demonstrated that the choice of network architecture significantly affects the network's inductive bias. Notably, the choice of pooling and convolution in CNNs is guided by the desire to imbue the networks with invariance to spatial transformations such as rotations, translations, flips, and other small deformations. Research by Fukushima & Miyake (1982) alluded that the reason all layers of the neocognitron are convolutional indicates that the response in the final layer "is not affected by the shift in the position of the stimulus pattern. Neither is it affected by a slight change of the shape or the size of the stimulus pattern."

The second source of inductive bias in convolutional networks is data augmentation (Krizhevsky et al., 2012), a technique that artificially increases the number of training examples. Data augmentation is frequently used to make models invariant to arbitrary transformations beyond the ones built into the network's architecture by allowing the model to be trained with the original data plus transformed data (Chen et al., 2020).

However, Azulay & Weiss (2018) have shown that convolutional networks are not invariant because they ignore the classical sampling theorem such that aliasing effects make their output variant. An alternative solution, evaluated by this research, is to

increase data augmentation using advanced techniques such as RandAugment and cGANs to show that they improve the invariance of many widely used classifiers such as ResNet and InceptionV3. The assumption for using these augmentation strategies is that they generate sufficient samples diverse enough in their attributes to result in robust image classifiers trained on them.

The essential premise of data augmentation is that in a real-world scenario, we may have a dataset of images taken in a limited set of conditions (Perez & Wang, 2017). However, our target application may exist under various conditions, such as orientation, location, scale, brightness, et cetera. Therefore, we use augmentation techniques to account for these situations by training neural networks with additional synthetically generated examples.

Data Augmentation cannot overcome all biases present in a small dataset. For example, in a dog breed classification task, if there are only German Shepherds and no instances of Bulldogs, no augmentation method will create an example of a Bulldog. However, several biases, such as lighting, occlusion, scale, viewpoints, and many more, are preventable or dramatically minimized with Data Augmentation (Shorten & Khoshgoftaar, 2019). The contributions of this research were as follows:

- We provided an empirical analysis of how Generative Adversarial Networks and Reinforcement learning-based data augmentation improves the classification performance of modern CNNs.
- We added empirical data to the field of deep convolutional networks to enable researchers to select the most appropriate data augmentation scheme for a given dataset and classifier.

- We demonstrated that data augmentation minimizes the problems of overfitting and poor invariance in CNNs caused by limited training data.
- We conducted extensive experiments that compare GAN-based data augmentation with stochastic geometric data augmentation and demonstrate which method or combination maximizes the invariance of CNNs for image classification.

Barriers and Issues

ConvNets are equivariant to translations of their input to some degree but not to other transformations. Data augmentation and architectural modifications are the two most common techniques to grant invariance to ConvNets. However, the mechanisms by which the models acquire invariant properties are poorly understood. The impact of modifications to their architectures on their efficiency and representational power is unknown. Moreover, using different augmentation strategies to gain invariance has yet to be extensively studied (Lenc & Vedaldi, 2015). Another challenge was that data augmentation methods are task-independent in that operations are performed simultaneously on the image data and labels. However, since label types are different under different tasks, augmentation strategies for image recognition tasks, for instance, cannot be directly applied to semantic segmentation tasks, resulting in inefficiency and low scalability.

Furthermore, there is no quantitative measure of the optimal size of training datasets. Moreover, in practice, the size of the training datasets is usually designed according to personal experience and extensive experiments (Yang et al., 2022). The latter implies that data augmentation should be done carefully to ensure that the

augmented data is still within the support of the original data distribution (Perez & Wang, 2017).

Assumptions, Limitations, and Delimitations

This dissertation assumed that all datasets are from the same distribution, with their validation sets being of sufficient size to prevent high variance and overfitting (Lorraine et al., 2020). A limitation of this study was the lack of significant quantitative analysis of spatial invariance in CNNs. This shortage results from the minimal studies on how CNNs quantitatively learn invariances during training. Another limitation was the readiness of software packages, such as the one used to implement RandAugment. Design choices within that package result in significant performance drops when applying geometric transforms to images. As an alternative, RandAugment was implemented using TensorFlow's Vision Library (Abadi et al., 2016). This addition increased the runtime overhead of the code, but it was a necessary compromise while the Keras team resolved RandAugment issues. Finally, to limit the scope of this study and make it manageable, experiments were conducted using only two CNN architectures, three baseline datasets, and five evaluation policies for a total of 45 experiments (3 CNN * 3 Datasets * 5 Policies). Furthermore, training was over a maximum of 100 epochs with early stopping, and the author assessed the quality of synthesized images (*as opposed to being algorithmically determined*) as being adequate for use as training samples.

Definition of Terms

Activation Function: This function is applied by neural networks to learn nonlinear relationships between input features and their labels. ReLU and Sigmoid are typical activation functions used.

Adaptive Moment Estimation (Adam): A gradient-based optimization algorithm for stochastic objective functions. It uses randomly selected subsets of data to create a stochastic approximation instead of the entire dataset.

Backpropagation: A mathematical method of computing gradients of expressions through the recursive application of the chain rule.

Batch: A set of examples used in one training iteration.

Batch Normalization: A regularization technique that normalizes the set of activations in a layer.

Batch size: The number of examples a model processes in a single iteration.

Bias: A parameter in machine learning models representing an origin offset customarily used to minimize overfitting.

Convergence: A state reached during training where the loss changes very little or not during an iteration.

Convolution: A mathematical operation performed by taking the dot product of a filter and a slice of an input matrix, where the slice of the input matrix has the same size and rank as the filter. It is also called a *convolutional operation*.

Convolutional layer: This is a layer in a deep neural network where a kernel strides along an input matrix, performing a convolutional operation at each stride to produce an activation or feature map.

Convolutional Neural Network (CNN): A neural network with at least one convolutional layer. It typically consists of some combination of the following layers: convolutional layers, pooling layers, and dense layers. They are also referred to as ConvNets.

Cross-Entropy: A generalization of the log loss function used in binary classification to multi-class classification problems.

Data Augmentation: A technique to artificially increase the number of training examples by transforming existing examples.

Dataset: A collection of raw data, typically organized in a spreadsheet or CSV (comma-separated values) format.

Deep Learning: A machine learning technique in which a model learns to perform classification tasks directly from sound, images, or text, where the neural network typically has two or more hidden layers.

Deep Neural Network: This neural network contains two or more hidden layers. They are also referred to as a deep model or a deep network.

Depth: The sum of all layers (hidden, output, embedding) in a neural network except for the input layer.

Discriminator: This subsystem within a Generative Adversarial Network determines whether examples are real or fake, where examples are typically images.

Dropout: A regularization technique that inactivates the activation values of randomly chosen neurons during training. This typically involves setting the activation values to zero.

Early Stopping: A regularization technique where a model's training is intentionally stopped before its loss increases.

Epoch: A complete pass over the entire training set during training such that each example is processed once.

Example: The values of one row of features and an optional label, i.e., an (x, y) pair where x is some input, and y is its label. An example is also referred to as a *sample*.

Feature: An input variable to a machine learning model.

Feature Map: The output of one filter or kernel applied to the previous layer.

Feed Forward Neural Network: Neural networks where the output from one layer is used as input to the next layer. These networks are acyclic since information is always fed forward and never backward.

Filter: A matrix having the same rank(dimension) as an input matrix but a smaller shape. For example, given a 28x28 input matrix, the filter could be a 3x3 matrix. It is also called a *kernel*.

Forward Pass: The flow of information from the input to the neural network's output.

Fully Connected Layer: A hidden layer in which each node (*or neuron*) is connected to every node in the subsequent hidden layer. Also known as a dense layer.

Fully Connected Neural Network: A neural network consisting of fully connected layers.

Generative Adversarial Network: A system that synthesizes new data by using a Generator to create data and a Discriminator that determines if the created data is real or fake.

Generator: A subsystem within a generative adversarial network that synthesizes new examples. Examples created are typically images.

Gradient Descent: An algorithm that minimizes loss by iteratively adjusting weights and biases to find the best combination that results in a minimal loss.

Hidden Layer: This represents a layer between a neural network's input and output layers.

Hyperparameter: Variables that are tuned during successive iterations when training a model.

Image Classification: A process that classifies objects, patterns, or concepts as images.

InceptionV3: An award-winning 42-layer image recognition ConvNet developed by Google.

Inductive Bias: The set of assumptions a learner uses to predict outputs of given inputs that it has not encountered.

Inference: Making predictions by applying a trained model to unlabeled examples.

Input Layer: The layer of a neural network containing the feature vector, i.e., provides examples for training or inference.

Iteration: A single update of a model's parameters during training.

Keras: A Python machine learning API that typically runs on TensorFlow.

Label: The answer or result portion of an example.

Labeled Example: An example that contains at least one feature and a label.

Learning Rate: A floating-point number the gradient descent algorithm uses to adjust weights and biases on each iteration.

Loss: A measure of how far a model's prediction is from its label.

Loss Function: A function that computes the loss on a batch of examples. Typical loss functions include Mean Squared Error (MSE) and Log-Loss. It is also known as the cost or objective function.

Machine Learning: A program or system that trains a model from input data such that the trained model can make actionable and practical predictions from unseen data, typically from the same distribution as the training data.

Model: For this dissertation's context, a model is the set of parameters and structures a system needs to make predictions.

Neural Network: A model containing one or more hidden layers called an Artificial Neural Network.

Neuron: A distinct unit within a hidden layer of a neural network. Each neuron performs a two-step action: (1) calculates the sum of its input values multiplied by their respective weights, and (2) passes the weighted sum as input to an activation function.

NoelNet: A basic ConvNet developed by the author to serve as a base model for evaluations and to establish competitive baseline metrics on F/MNIST and CIFAR-10.

One-Hot Encoding: A technique used to represent categorical data as vectors in which one element is set to 1, and all others are set to 0.

Optimizer: An implementation of the gradient descent algorithm, e.g., Adam

Output Layer: The final layer in a neural network containing the predictions or classifications.

Overfitting: The phenomenon where a model matches its training data so closely that the model fails to make correct predictions on unseen data.

Pooling: An operation that reduces the size of a matrix created by an earlier convolutional layer but preserves the rank of the resulting smaller matrix. Pooling typically involves taking the maximum or average value across a pooled area.

Pooling Layer: The layer in a CNN that is responsible for the pooling operation.

Pre-trained Model: A model(s) that have already been trained. These models are typically used in transfer learning scenarios.

RandAugment: A stochastic data augmentation technique that sequentially applies a specified number of uniformly sampled transformations with specified strength of distortions.

Receptive Field: A local region of an input volume equivalent to the filter size.

Regularization: A technique used to reduce overfitting, such as early stopping and dropout.

Reinforcement Learning: A set of algorithms that learn an optimal policy to maximize a return or reward when interacting with an environment.

ResNet50: An award-winning 50-layer image recognition ConvNet developed by Microsoft.

Softmax: A function that computes the probabilities for each class in a multi-class classification model; the probabilities sum to precisely 1.0.

Stochastic Gradient Descent: An algorithm that implements Gradient Descent.

Stride: The spatial extent to which a filter/kernel is moved across an input volume. E.g., a kernel having a stride of 1 means it is moved along an input volume one pixel at a time.

Supervised Learning: Training a machine learning model from features and corresponding labels. Analogous to studying a set of questions and their answers.

TensorFlow: A distributed and large-scale machine learning platform.

Testing Dataset: A dataset used to evaluate the model after completing the training. It measures the model's performance on unseen data using accuracy, precision, & recall.

Training: The process used by a model to determine its ideal parameters(weights and biases).

Training Dataset: A dataset used to train a neural network that allows the model to learn hidden patterns, features, and other distortions in the data. More diversity within this dataset leads to more robust models.

Transfer Learning: Transferring information from one machine learning task to another, such as sharing knowledge from the solution of a simpler task to a more complex one or transferring knowledge from a task with more data to one with less data.

Translational Invariance: A property of a neural network that allows it to be robust to changes in horizontal or vertical shifts of its input.

Validation Dataset: The dataset used to validate the model performance during training. It is separate from the training dataset or test dataset.

Weight: A randomly initialized value that a model multiplies by another value. A model's ideal weight is found by training, after which those weights are used to make inferences.

List of Acronyms

Adam: Adaptive Moment Estimation

ANN: Artificial Neural Network

CIFAR-10: Dataset of color images in 10 categories

CNN: Convolutional Neural Network

CONV: Abbreviation for Convolution

ConvNet: Convolutional Neural Network

DAG: Directed Acyclic Graph

CDCGAN: Conditional Deep Convolutional GAN

DNN: Deep Neural Network

FC: Fully Connected

FCNN: Fully Connected Neural Network

FFNN: Feed Forward Neural Network

FMNIST: Dataset of various fashion items in 10 categories

GAN: Generative Adversarial Network

ImageNet: Large dataset with more than 14 million images in 1000 categories

MNIST: Dataset of hand-written digits in 10 categories (from zero to nine)

MSE: Mean Squared Error

NLP: Natural Language Processing

NN: Neural Network

ReLU: Rectified Linear Unit

SGD: Stochastic Gradient Descent

Summary

ConvNets have achieved remarkable success on many visual tasks, such as image classification. They are assumed to be spatially invariant because of their unique architectures. However, several authors have shown that contemporary ConvNets are not spatially invariant to small perturbations in their input images. This lack of robustness significantly reduces their predictive abilities, which could lead to fatalities, racial discrimination, or other unexpected outcomes. Data augmentation is one of two standard solutions for teaching spatial invariance to CNNs. Data augmentation artificially increases the training dataset by generating new and diverse data points extracted from existing training data. And in so doing, it directly imbues sources of invariances within the training dataset. While simple augmentation techniques are effective, they require domain expertise, must be hand-engineered, and do not capture enough of the axes of variations within the training data.

In contrast, advanced augmentation techniques that use deep learning, such as RandAugment and Generative Adversarial Networks, learn invariances from data by sampling from a continuous distribution space, leading to more diverse and varied training samples. The goal of this dissertation was to provide a comparative study on using advanced augmentation techniques such as RandAugment and GANs to improve the spatial invariance in CNNs. These techniques were applied to the MNIST, FMNIST, and CIFAR-10 benchmark datasets via five augmentation strategies and evaluated on NoelNet, ResNet50, and InceptionV3 ConvNets using metrics such as loss, accuracy, precision, recall, and training latency. The remainder of this dissertation report was organized as follows: Chapter 2 reviewed the literature that directly influenced this

research. Chapter 3 presented the proposed methodology for studying advanced data augmentation techniques to improve spatial invariance in CNNs. Chapter 4 summarizes the results of this research. Finally, chapter 5 confers conclusions, implications, and recommendations based on this research.

Chapter 2

Literature Review

This chapter presents a review of the literature that directly influenced the goal of this research. As such, the following areas of literature were the most relevant to this dissertation:

- Neural Networks
- Deep Neural Networks
- Deep Learning
- Convolutional Neural Networks
- Spatial Invariance
- Data Augmentation
- Related Works
- RandAugment
- Generative Adversarial Networks
- Network Architectures

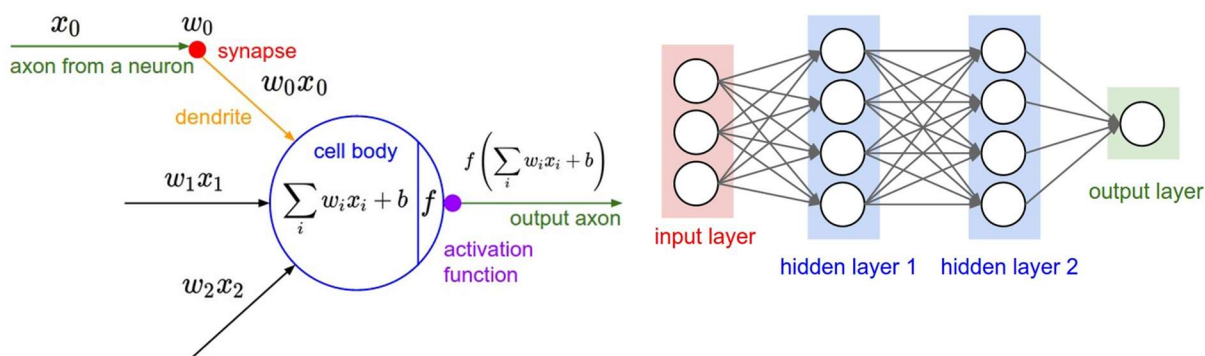
Neural Networks

Neural Networks (NNs) are biologically inspired computational models at the core of deep learning algorithms such as Google's search algorithm and allow the classification and clustering of data very quickly. They are usually comprised of neurons organized in one or more hidden layers to approximate some function f^* , that maps an input x to a category y . That is, neural networks define a mapping $y = f(x; \theta)$ and learn the value of the parameters θ that result in the best function approximation (Goodfellow et al., 2016, p.164). McCulloch & Pitts (1943) introduced neural networks to understand

how the human brain could produce complex patterns through the interactions of connected brain cells called neurons (McCulloch & Pitts, 1943). Their research led to the mathematical model of a neuron, as seen in Figure 1, intended to mimic how neurons in the human brain signal to one another.

Figure 1

Illustration of the Model of a Neuron and a Three-Layer Neural Network



Note. Left: Diagram showing the biological inspiration behind a neuron. Inputs x_i interact multiplicatively with synapses having weights w_i ; the neuron body accumulates the sum (*together with a bias*), passes it to an activation function to add non-linearity, then fires as an output signal. **Right:** Example of a 3-layer neural network. The neurons in one layer are connected to all neurons in the previous layer but are not connected to each other. This arrangement efficiently evaluates neuron activations in one layer with matrix multiplication. Adapted from “Connecting images and natural language,” by A. Karpathy, 2016, Stanford Digital Repository. Copyright 2016 by Andrej Karpathy.

In Figure 1 (left; Karpathy, 2016), a signal x_i enters a neuron via its synapse, with the synapse having weight w_i . The interaction is multiplicative, $w_i x_i$, where the weights (*randomly initialized vectors of small values*) are learnable and control the influence that one neuron has on another: excitatory (*positive weight*) or inhibitory

(*negative weight*). All the signals entering the neuron get multiplied by their respective weights, then summed, and a bias (*initialized to zero or small positive values*) added. This weighted input is passed through an activation function f , which determines the neuron's output. If the output exceeds a given threshold (*the bias*), the neuron “fires” or activates, passing its data to the neurons in the next layer. Otherwise, the neuron does not activate, and no data is passed from the neuron to the next layer (Li et al., 2015).

Formally, let w_{jk}^l denote the weight of the connection from the k^{th} neuron in $(l - 1)^{th}$ layer to the j^{th} neuron in l^{th} layer, b_j^l for the bias of the j^{th} neuron in the l^{th} layer, and a_j^l for the activation of the j^{th} neuron in the l^{th} layer, we have:

$$a_j^l = f \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right) \quad (1)$$

where the sum is over all the neurons k in the $(l - 1)^{th}$ layer. Equation 1 can be re-written more generally as:

$$y = f(\langle x, W \rangle + b) \quad (2)$$

, where y denotes the output of a neuron, $\langle \cdot, \cdot \rangle$ denotes the dot product, $f(\cdot)$ is a non-linear activation function, and W and b are the weights and biases of the neuron, respectively (Nielsen, 2015, p. 61). The activation function is added for non-linearity and models the neuron's firing rate. Several activation functions can be used, but the Rectified Linear Unit (ReLU; Nair & Hinton, 2010) is usually recommended. ReLU computes the function:

$$f(x) = \max(0, x) \quad (3)$$

, which has shown a 6x improvement in the convergence of models using it (Krizhevsky et al., 2012). As such, ReLU was used as the activation function for this research.

Neural Networks combine multiple nonlinear processing layers in an acyclic graph that consists of an input layer, zero or more hidden layers, and an output layer. The layers are interconnected via neurons, where each neuron connects to another, and the connections between neurons do not form a cycle, i.e., information processes in only one direction. Such networks are called Feed Forward Neural Networks, and their structure makes them easy to evaluate using matrix-vector operations. In a regular NN, the most common type of layer is the fully connected (FC) layer, where neurons between adjacent layers are fully pairwise connected, but neurons within a layer do not share any connection, e.g., Figure 1 (*right*; Karpathy, 2016). Fully Connected Neural Networks (FCNN) have significant representational power and are shown to be universal function approximators in that they can approximate any continuous function (Cybenko, 1989; Nielsen, 2015, p. 176).

Learning in neural networks is the process whereby a network uses training examples to infer rules for recognizing patterns within those examples. By increasing the number of samples, a network can learn more about these patterns and improve its accuracy on predictions involving similar but new patterns (Goodfellow et al., 2016, p.97). Consider a neural network with weights w_{ij} connecting neuron j to neuron i . Each neuron has total input x , non-linear activation function $f(x)$, and output $y = f(x)$. The goal of the network is to learn the weights from the training data such that the predicted output \hat{y} is close to the target y for all inputs x . A loss function L measures how far the prediction is from the target.

For multi-class (*greater than two classes*) classification problems, the softmax activation function (Goodfellow et al., 2016, pp. 180-184) is usually applied to the

outputs on the final layer to compute the probabilities over the classes for each input. Finally, the categorical cross-entropy loss function is used to determine the final loss on the batch. See Appendix A, “Softmax,” for information on computing the softmax probabilities and the cross-entropy loss function.

Training a neural network involves multiple iterations of a two-pass cycle: the forward pass (forward propagation) and the backward pass (backpropagation). During the forward pass, an example (x_{input}, y_{target}) enters the input layer, where its output is equal to the input since neurons in the input layer do not have activation functions, i.e., $y_{out} = x_{input}$. The first hidden layer is then updated by taking the output of the neurons in the previous layer and using the weights to compute the input x of the nodes in the next layer. Finally, the activation function $f(x)$ computes the output of the neurons in the first hidden layer, as seen in (2). This process is propagated forward for the rest of the network until the final output layer.

During the backward pass, the network tries to find parameters that minimize the loss from the loss function. It does so by adjusting the weights of all the neurons in the hidden layers, using the backpropagation and gradient descent algorithms, respectively (LeCun et al., 1998; Rumelhart et al., 1986). The intuition is that a neural network contains many neurons across several hidden layers, where each neuron contributes to the overall loss of the network in separate ways. The backpropagation algorithm (See Appendix A, “Backpropagation”) determines whether to increase or decrease the weights applied to each neuron by using the chain rule to calculate the partial derivative of the loss function with respect to each weight in the batch. Once the error derivatives are calculated, a gradient descent algorithm such as Adam (Kingma & Ba, 2014; Appendix

A, “Adam Optimizer”) or Stochastic Gradient Descent (Appendix A, “Stochastic Gradient Descent”) is used to perform the optimization. The optimization rule is detailed further in Appendix A, “Weight Optimization.”

Deep Neural Networks

Deep Neural Networks are excellent at discovering intricate structures in high-dimensional data using deep learning techniques. They are powerful learning models that have made considerable progress in discriminative tasks (Szegedy et al., 2014; LeCun et al., 2015) such as Computer vision (Krizhevsky et al., 2012), Speech recognition (Hinton et al., 2012), and Natural Language Processing (Torfi et al., 2020). Advancements in deep network architectures, powerful computing, and access to big data have fueled their success (Shorten & Khoshgoftaar, 2019).

First, the scale of a network’s architecture is proportional to its generalization ability. For instance, due to its increased depth, the 152 layers of ResNet (He et al., 2015) gain significant accuracy compared to shallower networks.

Second, the advancements in computing have significantly impacted deep learning because powerful computing makes it possible to design models with deeper architectures that can be trained in hours instead of weeks.

Third, the availability of free and open datasets like ImageNet (Russakovsky et al., 2015), MNIST (LeCun et al., 1998), and CIFAR-10 (Krizhevsky & Hinton, 2009) are crucial to the training and further development of deep learning models (Yang et al., 2022). Finally, regarding image classification, the success of deep neural networks is attributed to the development of Convolutional Neural Networks.

Deep Learning

For decades, image classification tasks required significant domain expertise and relied heavily on hand-engineered features using Scale Invariant Feature Change (SIFT; Lowe, 2004), Histogram of Oriented (HOG; Dalal & Triggs, 2005), or Fisher Vectors (Sánchez et al., 2013). These features were used to transform raw data into a suitable internal representation from which a classifier could classify patterns in the input (Krizhevsky et al., 2012; LeCun et al., 2015).

Deep learning is a set of techniques to automate deep network feature detection and pattern recognition. They allow a machine to be fed raw data, e.g., pixels from an image, and automatically discover the features needed for classification or detection. The latter occurs because deep networks have multiple levels of representation, obtained by creating simple non-linear modules that transform the representation at one level (*starting with the input*) into slightly more abstract representations at higher levels. As a result, deep networks can learn complex functions by creating enough such modules (LeCun et al., 2015).

Higher layers amplify input features important for discrimination and discard irrelevant variations for classification tasks. For example, an image is composed of an array of pixel values. The first layer's learned features typically represent the presence or absence of edges at different orientations and locations in the image. The second layer typically detects patterns by recognizing arrangements of edges, regardless of slight variations in the edge positions. Finally, the third layer may assemble patterns into larger combinations corresponding to parts of familiar objects, and subsequent layers would detect objects as combinations of these parts. The key aspect of deep learning is that

human engineers do not design these layers of features. Instead, they are learned from data using general-purpose learning procedures where a human does not have to manually define all necessary abstractions or provide vast sets of hand-labeled examples (Bengio et al., 2009; LeCun et al., 2015).

Convolutional Neural Networks

Convolutional Neural Networks (ConvNets, or CNNs; LeCun et al., 1989, 1998) are specialized feed-forward neural networks used to extract features from data that comes in the form of multiple arrays such as 1D for signals, 2D for images, and 3D for video (LeCun et al., 2015). They are the predominant networks used to classify images because they utilize parameterized, sparsely connected kernels that preserve the spatial characteristics of images (Shorten & Khoshgoftaar, 2019).

ConvNets were inspired by the structure of the biological visual system through models proposed by Hubel and Wiesel (1962), whose research showed that individual neurons respond to stimuli only in a restricted region of the visual field called the *receptive field*. When a collection of such fields overlaps, it covers the entire visual area (Hubel & Wiesel, 1962). The first computation model based on these visual structures was the Neocognitron by Fukushima (1980). He realized that translational invariance was obtained when neurons with the same parameters were applied on patches of the previous layer at distinct locations (Fukushima, 1980).

This idea was later followed up by LeCun, who trained variants of the Neocognitron using error gradient (backpropagation). He eventually obtained state-of-the-art performance on several vision tasks, including handwritten character recognition (LeCun et al., 1989, 1998); *LeCun's CNN is a standard and machine learning benchmark*

today. The CNN described by (LeCun et al., 1998) has seven layers. When randomly initialized, this number of layers would be almost impossible to train and optimize properly in an equivalent Fully Connected Neural Network (Bengio et al., 2009). The ubiquitous use of CNNs over Fully Connected Neural Networks for image classification is due to their unique architecture.

CNN Architecture

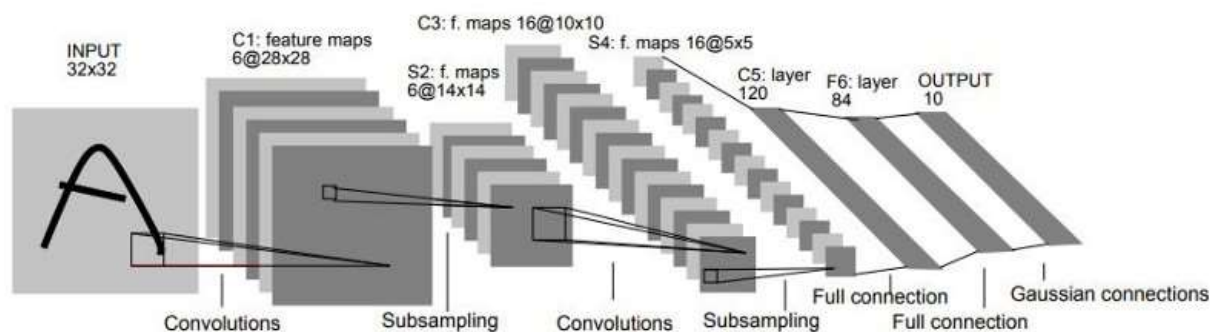
ConvNets are like regular neural networks in that they are both made up of neurons with learnable weights and biases. They still express a single differentiable score function and apply a loss function like softmax on the fully-connected output layer. However, CNNs make the explicit assumption that their inputs are images, which allows the encoding of specific properties into the architecture, making the forward function more efficient to implement and significantly reducing the number of parameters in the network (Li et al., 2015). Unlike CNNs, regular neural networks do not consider images' spatial structure and treat input pixels far apart and close together with the same significance (Nielsen, 2015, p. 228).

Furthermore, neural networks do not scale well to full images (LeCun et al., 1998; LeCun & Bengio, 1995). For example, the images in CIFAR-10 are only $32 \times 32 \times 3$ in size (width, height, and depth, respectively). Using a regular neural network to represent such an image would require a single fully connected neuron in the first hidden layer to have $32 \times 32 \times 3 = 3072$ parameters which may seem manageable. However, an image of a more respectable size, e.g., $256 \times 256 \times 3$, would require each neuron to have 196608 parameters. And a typical network would need to have several such neurons, which would make the number of parameters add up very quickly, rapidly increasing the network size and

promptly leading to overfitting and significant computing storage (Li et al., 2015). As shown in Figure 2 (LeCun et al., 1998), a ConvNet architecture typically comprises three main layers: Convolutional Layer, Pooling Layer, and a Fully Connected Layer.

Figure 2

The Architecture of Lenet-5 CNN Developed to Classify Hand-Written Digits



Note. From “Gradient-based learning applied to document recognition,” by Y. LeCun, L.

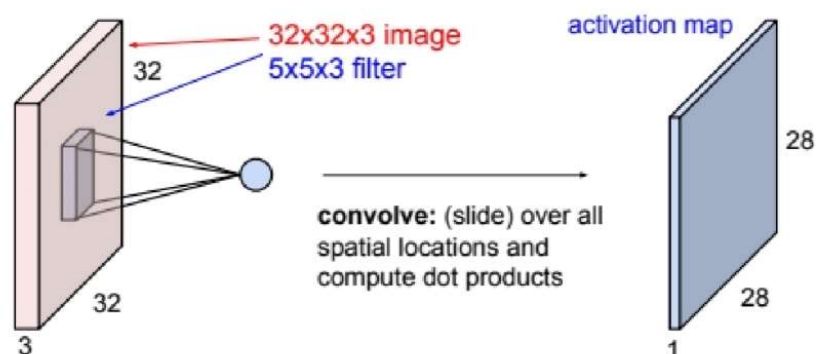
Bottou, Y. Bengio and P. Haffner, 1998, Proceedings of the IEEE, 86(11), pp. 2278-2324

Convolutional Layer

The convolutional layer is the core building block of a CNN, which does the computational heavy lifting. This layer aims to learn the feature representations of the input. Figure 2 (LeCun et al., 1998) shows that a convolutional layer comprises several convolution kernels used to compute different feature maps. Specifically, each neuron of a feature map connects to a local region (*receptive field*) of neighboring neurons in the previous layer. A new feature map is obtained by convolving the input with a learned kernel and applying an element-wise non-linear activation function on the convolved results. To generate each feature map, the kernel weights are shared by all spatial locations of the input. The complete set of feature maps is obtained by using several different kernels.

Figure 3

Example of a Convolution in the First Hidden Layer of the Lenet-5 Convnet



Note. Convoluting a 5x5 filter over a 32x32x3 image with stride 1 and zero input padding. The filters are always spatially smaller than the input but span the full depth of the input. The convolution operation produces a 28x28x1 activation map where each neuron in the feature map results from the dot product between the filter and input. A convolution layer typically has multiple filters (e.g., 6) applied independently, resulting in their respective activation maps. Finally, the activation maps are stacked along the depth to produce the output layer, e.g., 28x28x6. From “Connecting images and natural language,” by A. Karpathy, 2016, Stanford Digital Repository. Copyright 2016 by Andrej Karpathy.

Figure 3 (Karpathy, 2016) shows a convolution as a mathematical operation that takes the dot product between two matrices. In this context, one matrix is the kernel, and the other is the receptive field. The kernel is spatially smaller than the input image. Still, it extends the entire *depth* of the input, e.g., if the input is a CIFAR-10 image having size 32x32x3, and the filter (kernel) size is 5x5, then each neuron in the convolutional layer will have weights to a 5x5x3 region in the input volume. During a forward pass, the kernel slides across the input image (*left to right, top to bottom*), producing a two-

dimensional representation of the image known as a *feature* or *activation map*. The sliding size of the filter is called the *stride*. And it is sometimes convenient to *zero-pad* the input volume with zeros around the border to preserve information that may otherwise be lost (Nielsen, 2015, pp. 228-233). A convolution layer that accepts an input of size $W_1 \times H_1 \times D_1$, with K filters, where each filter has spatial size F , stride S , and amount of zero padding P , will produce an output volume of size $W_2 \times H_2 \times D_2$ where: $W_2 = \frac{W_1 - F + 2P}{S} + 1$, $H_2 = \frac{H_1 - F + 2P}{S} + 1$, and $D_2 = K$. The size of the output volume corresponds to the number of neurons in that volume (Karpathy, 2016).

More formally, the k^{th} output feature map Y_k is defined as:

$$Y_k = f(W_k * x) \quad (4)$$

, where x denotes an input activation such as an image, W_k denotes the k^{th} convolutional filter, the $*$ symbol denotes the 2D convolutional operator, and $f(\cdot)$ is the non-linear activation function (Nielsen, 2015, pp. 230-233). By repeating (4) many times across the convolutional layers of a ConvNet, the ConvNet can learn progressive levels of features. The features extracted from the lower levels (*i.e., closer to the input layer*) are simple features like corners, lines, and edges, which are progressively combined in the higher levels to form motifs, motifs assemble into parts, and parts assemble to form objects (LeCun et al., 2015)

Convolution leverages three essential ideas: sparse interactions, parameter sharing, and *equivariant representations* (Goodfellow et al., 2016, p.329-335; Karpathy, 2022). The layers in a traditional neural network use matrix multiplication by a matrix of parameters describing interactions between input and output neurons, meaning that every output neuron interacts with every input neuron. However, CNNs use sparse interaction,

achieved by making the kernel smaller than the input, e.g., using a 5x5 kernel to process a 32x32x3 input. Using these kernels to process images with thousands or millions of pixels, we can detect meaningful information that is tens or hundreds of pixels. This is significant because this method allows storing fewer parameters, reducing memory requirements, and improving the model's statistical efficiency (Goodfellow et al., 2016, p.329-330).

An assumption made by CNNs to dramatically reduce the number of parameters is that if one feature is useful to compute at a spatial position (x_1, y_1) then it should also be useful to compute at a different position (x_2, y_2) . This means that when creating an activation map, neurons in a CNN are constrained to use the same set of weights, unlike in a traditional neural network where each element of the weight matrix is used only once and never revisited. These shared parameters perform the same operation on different parts of the image. This has several advantages such as multiple features can be extracted at each location in parallel, the network becomes easier to train, and model complexity reduces (Goodfellow et al., 2016, p.331-332; Gu et al., 2018; Rumelhart et al., 1986).

As a result of parameter sharing, the layers of a ConvNet possess a property called equivariance to translation. A function is equivariant means that if the input changes, the output changes in the same way, i.e., function $f(x)$ is equivariant to function g if $f(g(x)) = g(f(x))$. Once detected, this means the exact location of a feature becomes less significant so long as its approximate position relative to other features remains preserved (Goodfellow et al., 2016, p.334; LeCun et al., 2015).

Non-Linearity Layers

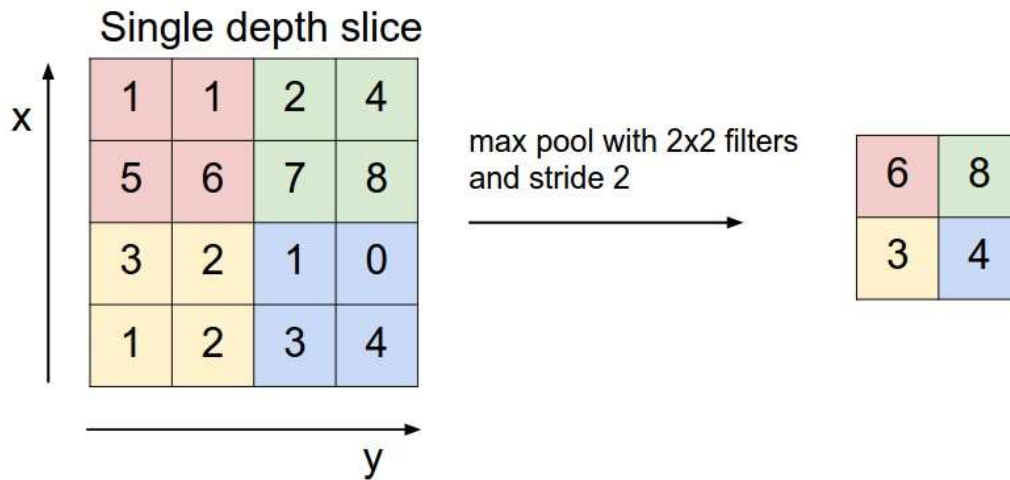
A convolution is a linear operation, but images are far from linear. Therefore, non-linearity layers are often placed directly after a convolutional layer to introduce non-linearity to the feature maps; *They do not change the values in the feature maps*. Non-linearities are desirable for multi-layer networks to detect non-linear features (Gu et al., 2018). The sigmoid and hyperbolic tangent functions have traditionally been used, but recently ReLU has become the most popular choice for many applications (LeCun et al., 2015).

Pooling Layer

The role of the pooling layer is to merge semantically similar features into one (LeCun et al., 2015) by progressively down-sampling the spatial size (*depth remains unchanged*) of the feature maps. As a result, it reduces the number of parameters and computations in the network and imbues the CNN with translational invariance (Goodfellow et al., 2016, pp:335-337). It is typically placed between successive convolutional layers where it computes a statistical summary of nearby outputs. The pooling operation is processed on every feature map individually. There are several pooling operations, but the most typical are average pooling (Wang et al., 2012) and max pooling (Zhou & Chellappa, 1988). Figure 4 (Li et al., 2015) shows an example of max pooling using 2x2 filters with a stride of 2. The max value is taken over four numbers (each 2x2 square).

Figure 4

A Max Pooling Operation With 2x2 Filters and a Stride of 2



Note. From *Convolutional neural networks for visual recognition*, by F. Li, A. Karpathy, & J. Johnson, 2015 (<http://cs231n.github.io/convolutional-networks>)

Fully Connected Layer

This is the last layer of the CNN and is an output layer. The neurons in this layer have full connectivity with all neurons in the previous and subsequent layers, akin to a regular FCNN. Softmax is typically used in this layer for classification and to calculate the overall loss of the CNN. See Appendix A, “Softmax,” for more information on the operation.

Spatial Invariance In Convolutional Neural Networks

In the context of this work, spatial invariance is a highly desirable property of ConvNets in which the network can disentangle the poses and deformations of objects from their texture and shape (Jaderberg et al., 2016; LeCun et al., 2015). Moreover, it is a fundamental property of image representation where the objects in an image are invariant to viewpoint changes, i.e., more significance is placed on a feature being present in an image over where exactly it is within the image (Lenc & Vedaldi, 2015). For example, when determining if a face is present in an image, the location of the eyes with pixel-perfect accuracy is not necessary, only that there is an eye on the left and right side of the face.

ConvNets have achieved impressive performance on many complex visual tasks and are the most prevalent technique used for image classification. Much of that success is because of their ability to exploit the natural translational symmetries in data through weight sharing and pooling (Goodfellow et al., 2016, pp. 336-337, LeCun et al., 2015). The introduction of local pooling layers, specifically max pooling, has helped to satisfy the property of spatial invariance by allowing CNNs to be somewhat invariant to the position of features within an input volume (Jaderberg et al., 2016). According to LeCun et al. (2015), the pooling layer merges semantically similar features into one. Because the relative positions of the features forming a motif can vary, detecting the motif can be done by coarse-graining the position of each feature. Max pooling computes the maximum of a local patch of neurons in one feature map then neighboring pooling neurons take input from patches that are shifted by more than one row or column. This

has the effect of reducing the dimension of the representation and creating invariance to small shifts and distortions.

Weight sharing has also helped satisfy spatial invariance in CNNs to some degree. The assumption is that if one feature is useful to compute at a spatial position (x_1, y_1) , it should also be useful to compute at a different position (x_2, y_2) . This is a reasonable assumption because local groups of values in images are often highly correlated, forming easily detected patterns. In other words, if an artifact could appear in one part of the image, it could appear anywhere within that image, hence the idea of neurons in different locations of a feature map sharing the same weights and biases (LeCun et al., 1998, 2015). By ensuring all neurons in a feature map share a single weight and bias, they are constrained to perform the same operation on different parts of the image, thus providing some translation equivariance. For instance, if the neurons in one feature map are tuned to look for vertical lines, they can be activated if a vertical line is found anywhere within the image, irrespective of location (Goodfellow et al., 2016, p.334-335).

Despite their “superhuman” performance on object recognition (He et al., 2015), CNNs suffer from a significant problem: they are invariant to some but not all input transformations (Engstrom et al., 2018; Szegedy et al., 2014; Jaderberg et al., 2016; Azulay & Weiss, 2018; Chen et al., 2020). Lack of spatial invariance in CNNs, even by minimal amounts, can lead to unfortunate consequences such as fatalities from autonomous vehicles or racial bias from image recognition systems (Kendall & Gal, 2017).

Szegedy et al. (2014) observed that applying an imperceptible (*to humans*) non-random perturbation to a test image arbitrarily changed the prediction of state-of-the-art

CNNs that should be robust to small perturbations in the input, i.e., small perturbations should not change the category of an image. These perturbations are found by optimizing the input to maximize the prediction error and were termed “adversarial examples.” To minimize the effects of adversarial examples, the authors suggest that back-feeding adversarial examples to the training might improve the generalization of the resulting models, a data augmentation approach (Engstrom et al., 2018; Madry et al., 2017; Szegedy et al., 2014).

Azulay & Weiss (2018) demonstrated that the chance that a CNN output on a randomly chosen image would change after translating downward by a single pixel can be as high as 30%. This raises serious concerns about using CNNs where reliability, dependability, security, and unbiasedness are concerned. According to Azulay & Weiss (2018), the two sources of inductive bias in a CNN are the convolutional architecture and data augmentation. However, they are still insufficient to achieve spatial invariance for the following reasons:

First, the architecture ignores the subsampling operation (or the *stride* operation), which causes a failure of translation invariance in systems with subsampling, first observed by Simoncelli et al. (1992). They stated that translation invariance could not be expected in a system based on convolution and subsampling unless the translation is a multiple of each subsampling factor. Since CNNs often consist of many subsampling operations, the subsampling factor of the deep layers may be significant, so translation invariance holds only for very specific translations. For example, in InceptionResnetV2, the subsampling factor is 60, so translation invariance can only hold for $\frac{1}{60^2}$ Possible

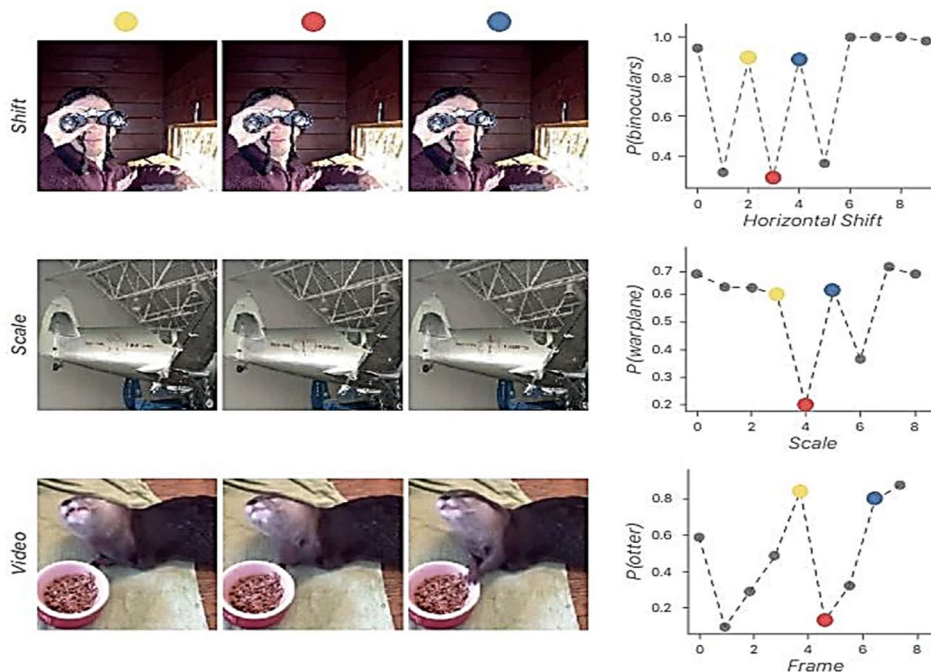
translations. The latter was corroborated by Zhang et al. (2019), who showed that invariance holds when a multiple of the subsampling factors shifts the input pattern.

Second, data augmentation causes the network to learn invariance only for images like those seen during training. This occurs because the distribution of training images is highly biased, leading to a lack of invariance for images disobeying the bias.

Azulay & Weiss (2018) have further shown that modern CNNs are surprisingly brittle when the input is translated, rescaled, or perturbed via small transforms. Figure 5 (Azulay & Weiss, 2018) illustrates examples of such failures on a modern CNN. A one-pixel shift (top row), one-pixel rescaling (middle row), and an imperceptible change in image posture (bottom row) lead to dramatic changes in the network’s predictions.

Figure 5

Jagged Predictions of CNNs Caused by One-Pixel Perturbations in Input Images



Note. Jagged predictions of a modern CNN caused by a one-pixel horizontal shift, one-pixel scale, and imperceptible change in posture of the input image. From “Why do deep

convolutional networks generalize so poorly to small image transformations?,” by A. Azulay, and Y. Weiss, 2018, arXiv.

Jaderberg et al. (2016) attribute the lack of spatial invariance in CNNs to the small spatial support for max-pooling (e.g., using 2x2 pixels). And that spatial invariance is only realized over very deep hierarchies of max-pooling and convolutions. Furthermore, Lenc & Vedaldi (2015) has demonstrated that the intermediate feature maps in a CNN are not variant to large transformations of the input data due to CNNs having limited, pre-defined pooling mechanisms to deal with variations in spatial data. As a result, they introduced a Spatial Transformer module, a dynamic tool that spatially transforms an image or feature map by producing an appropriate transformation for each input sample. This transformation is then performed on the entire feature map, allowing networks to select the most relevant regions of an image and to transform those regions to an expected post to simplify recognition in the subsequent layers, an architectural approach. (Jaderberg et al., 2016). Azulay & Weiss (2018) propose three practical solutions to improve the spatial invariance of CNNs: Antialiasing - suggested by (Zhang, 2019), Increased Data Augmentation - suggested by (Szegedy et al., 2014; Engstrom et al., 2017), and Reducing subsampling - suggested by (Jaderberg et al., 2016)

Antialiasing and subsampling reduction are architectural approaches.

Unfortunately, these have led to increasingly more complex networks because designers attempt to build invariances into the model instead of letting it learn them from its input. Also, this requires knowledge of every invariance beforehand (Immer et al., 2022), akin to the traditional hand-engineered techniques discussed at the beginning of this chapter. As a result, this dissertation sought to improve spatial invariance in CNNs using data

augmentation via two advanced techniques: RandAugment (Cubuk et al., 2020) and Generative Adversarial Networks (GANs; Goodfellow et al., 2014).

Data Augmentation

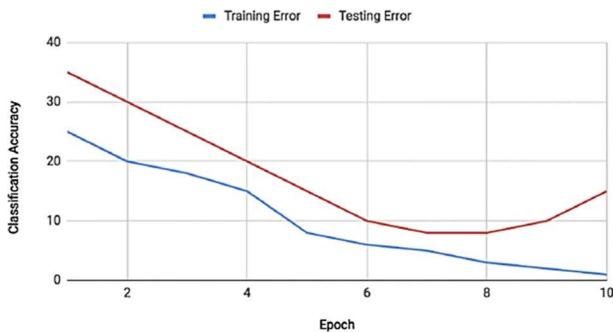
Data augmentation is a technique used to artificially increase the size of a training set by creating novel and realistic-looking examples by applying a transformation to an example without changing its label. Formally, let $q(\hat{x}|x)$ be the augmentation transform from which to draw an augmented example \hat{x} , based on some original example x . For a valid augmentation, it is required that an example $\hat{x} \sim q(\hat{x}|x)$ drawn from the distribution shares the same ground-truth label as x (Xie et al., 2020).

Improving the spatial invariance of CNNs increases their generalizability and statistical efficiency. Generalizability is defined as the performance delta of a CNN when evaluated on previously seen data (training data) versus data it has never seen before (testing data) (Shorten & Khoshgoftaar, 2019). A model is completely invariant to transforms if the training error is zero when trained from massive amounts of data under all conditions (Arjovsky et al., 2019). CNNs that overfit their training data exhibit poor generalizability and thus have poor spatial invariance. A very reliable way of discovering overfitting is to plot the training and validation loss at each epoch during the training of a classifier.

Figure 6

Graphs Showing Signs of Overfitting Using Training and Validation Loss

Signs of Overfitting (Training Error and Testing Error)



Desired convergence of Training and Testing Error

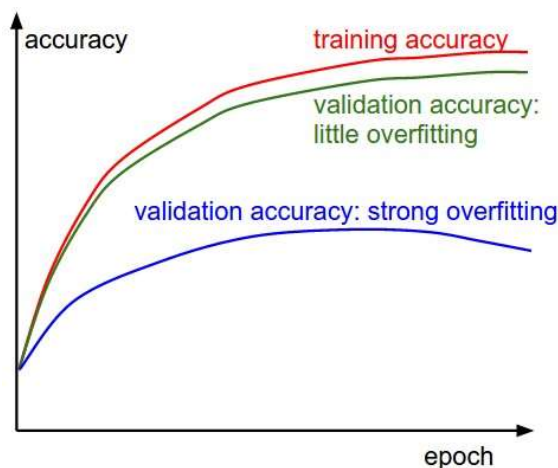


Note. From “A survey on Image Data Augmentation for Deep Learning,” by C. Shorten and T. Khoshgoftaar, 2019, *Journal of Big Data*, 6(1), p. 60

The leftmost plot in Figure 6 (Shorten & Khoshgoftaar, 2019) shows an inflection point where the validation error increases while the training error decreases. This implies that the model is overfitting the training data and is performing poorly on the testing (unseen) data. In stark contrast, the rightmost plot of Figure 6 (Shorten & Khoshgoftaar, 2019) shows a model with the desired relationship between the training and testing error, i.e., minimal overfitting, thus high spatial invariance (Shorten & Khoshgoftaar, 2019). Furthermore, Arjovsky et al. (2019) demonstrated that robust learning was equivalent to minimizing a weighted average of environment training errors, supporting using training and validation errors as indicators to check levels of invariance in CNNs.

Figure 7

Gap Between Training and Validation Accuracy Denotes Amount of Overfitting



Note. From *Convolutional neural networks for visual recognition*, by F. Li, A. Karpathy, & J. Johnson, 2015 (<http://cs231n.github.io/convolutional-networks>)

Similarly, the gap between the training and validation accuracy indicates the amount of overfitting. Figure 7 (Li et al., 2015) shows two possibilities. The blue curve shows minimal validation accuracy compared to the training accuracy, which indicates strong overfitting. In practice, this implies that an increase in the model's regularization may be required, such as adding more data augmentation or increasing dropout. The other case where the green curve tracks the training accuracy suggests little overfitting but implies the model's capacity may not be high enough, and the model may need to be made larger by increasing its parameters (Li et al., 2015).

Improving the generalizability of deep convolutional networks, thereby, their spatial invariance is a nontrivial challenge (Antoniou et al., 2016; Goodfellow et al., 2009; Immer et al., 2022). However, several authors have shown that data augmentation is one of the most effective and powerful methods of achieving this (Cubuk et al., 2020; Krizhevsky et al., 2012; LeCun et al., 1998; Shorten & Khoshgoftaar, 2019; Simard et al.,

2003). According to these authors, data augmentation approximates the data probability space by artificially increasing the size and diversity of the training set with the assumption that more information can be extracted from the existing dataset. Data augmentation is done either by data warping (label-preserving image perturbations in data space such as rotations) or oversampling (synthetic images are created and added to the training set).

Several other techniques are used to improve deep networks' generalizability and spatial invariance. Some of these include Dropout (Srivastava et al., 2014), Batch normalization (Ioffe & Szegedy, 2015), Transfer learning (Weiss, 2016), One-shot (Palatucci et al., 2009) and Zero-shot learning (Xian et al., 2018). However, despite their respective levels of success, they attempt to solve the problem of overfitting at the architectural level, in contrast to data augmentation that seeks to resolve the issue from the root, the training dataset. Instead, data augmentation aims to bake invariances into the dataset, such as rotations, lighting, occlusion, and scale. This way, models trained using that dataset learn these invariances and perform well despite transformations of their input (Shorten & Khoshgoftaar, 2019).

Furthermore, contemporary CNNs, with their deep architectures, are often over-parametrized, and over-parametrization carries subtle consequences. Unfortunately, spurious correlations and biases are often simpler to detect than the true phenomenon of interest. This implies that networks with vast numbers of parameters and limited data will quickly learn noise and other randomness, making them overfit. For these networks to learn invariance and reduce overfitting, it is imperative to increase the size and diversity of samples used to train them (Arjovsky et al., 2019).

One of the earliest demonstrations showing the effectiveness of data augmentations came from simple (traditional) transformations such as horizontal flipping, color space augmentations, and random cropping by Krizhevsky et al. (2012) to train AlexNet; their award-winning, state-of-the-art CNN on the ImageNet dataset. Data augmentation allowed them to increase their dataset size by 2048, reducing the error rate of AlexNet by over 1% (Krizhevsky et al., 2012).

The simple transformations described above typically refer to traditional affine and elastic transformations: creating new images by performing rotation or reflection of the original image, zooming in and out, shifting, applying distortion, and changing the color palette. They are popular and have proven to be an effective practice for data augmentation. However, despite their many advantages, there are some cases where classical operations are not enough to improve the accuracy of deep networks significantly or overcome the overfitting problem (Perez & Wang, 2017). Moreover, current research into adversarial attacks on CNNs showed that deep neural networks could be easily fooled into misclassifying images simply by partial rotations and image translation (Engstrom et al., 2018). Or adding noise to images (Madry et al., 2017) and even changing one skillfully selected pixel in the image (Su et al., 2019).

Increasing the dataset size via more advanced data augmentation techniques and using it to train a CNN would generally make that model more robust and less vulnerable to adversarial attacks and other perturbations of their inputs (Xie et al., 2020). Furthermore, Xie et al. (2020) have shown that using stronger data augmentations produced by advanced augmentation methods like RandAugment leads to the superior

performance of CNNs because of the greater diversity of the training samples and the increased sources of invariances from which to learn.

Xie et al. (2020) provide some intuitions on how advanced data augmentation is more advantageous over simpler ones from two aspects: First is *valid noise*, where advanced augmentation strategies generate realistic augmented examples that share the same ground-truth labels as the original example. Second is *diverse noise*, where advanced strategies generate diverse examples because they can extensively modify the input example without changing its label. In contrast, simple Gaussian noise only makes local changes.

Related Works

CNNs are formidable mechanisms for automatically learning features for image recognition tasks due to their multiple layers of simple computational elements. By combining the output of lower layers with higher layers, CNNs can represent progressively more complex input features. Several approaches have been used to imbue deep networks with spatial invariance. Hinton et al. (2006) introduced the deep belief network (DBN), where each layer comprises a restricted Boltzmann machine. Ranzato et al. (2007) explored sparsity regularization in autoencoding energy-based models and sparse convolutional DBNs with probabilistic max-pooling. After training, these networks achieved excellent performance on handwritten digit recognition tasks. However, their effectiveness was limited to simple datasets like MNIST and not more complex datasets like FMNIST or CIFAR-10. Models such as the Neocognitron (Fukushima & Miyake, 1982), HMAX model (Riesenhuber & Poggio, 1999), and

Convolutional Network (LeCun et al., 1998) achieve invariance by alternating layers of feature detectors with local pooling and subsampling of the feature maps.

More recently, several strategies have been introduced that focus on the model's architecture itself. Dropout (Srivastava et al., 2014) is a regularization technique that zeros out the activation values of randomly chosen neurons during training to force the network to learn more robust features instead of relying solely on a small subset of neurons. Tompson et al. (2015) extended this idea by dropping entire feature maps instead of individual neurons. Batch normalization (Ioffe & Szegedy, 2015) is another regularization technique that normalizes the set of activations in a layer by subtracting the batch mean for each activation and dividing by the batch standard deviation. More quantitative approaches have been taken by Goodfellow et al. (2009) and Azulay & Weiss (2018), and specialized deep network architectures have been proposed to improve invariance to transformations such as rotation, scaling, and arbitrary deformations (Dai et al., 2017; Cohen & Welling, 2016). Hosseini et al. (2017) presented a body of work studying the robustness of CNNs to image corruptions, while a parallel line of work is adversarial training, where specially designed perturbations are added to input images to yield significant changes in the output of a CNN (Szegedy et al., 2014; Madry et al., 2017). It was demonstrated by van der Wilk et al. (2018) that modeling invariance learning as a Bayesian model selection problem in Gaussian processes selects helpful invariances using the marginal likelihood of the model with training data alone. This approach was shown by Schwöbel et al. (2022) to be successful for small neural networks, but it failed to learn invariances on more complex datasets requiring deep neural networks.

More related to this work are those from Perez & Wang (2017), who used GANs and traditional transformations. However, the traditional transforms were manually selected, and the GANs were used to synthesize images using neural style transfer. Other closely related works are those from Shijie et al. (2017), who compared GANs, WGANs, flipping, cropping, shifting, PCA jittering, color jittering, adding noise, rotation, and some combinations on the CIFAR-10 and ImageNet datasets. They found that the combinations of the two types of augmentation resulted in better performance. While combining augmentation techniques can result in massively inflated dataset sizes, this is not guaranteed to be advantageous in minimal data settings, which could result in further overfitting. Shorten & Khoshgoftaar (2019) presented some existing methods and promising developments in data augmentation but did not provide an evaluation of the effectiveness of augmentation for various tasks and lacks some newly proposed methods, such as CutMix (Yun et al., 2019), and RandAugment (Cubuk et al., 2020). Yang et al. (2022) use CutMix and other recent augmentation techniques in their augmentation strategies but do not include experimentation using RandAugment or GANs.

The approach taken by this dissertation focused on using GANs (generative modeling) and RandAugment (reinforcement learning) to improve the spatial invariance of Convolutional Networks. This approach imbued invariances within the data, forgoing the laborious process of hand-engineering features, manual geometric transforms, or inflexible mutations of network architecture requiring significant knowledge of invariances a priori.

RandAugment

Highly effective data augmentation policies require significant manual work and expertise to design policies that capture prior knowledge in each domain. Learning policies to automate the design of such strategies have recently emerged and have shown enormous potential in addressing some of the weaknesses of traditional data augmentation methods (Cubuk et al., 2020). One such approach is RandAugment. It is an advanced data augmentation technique inspired by AutoAugment (Cubuk et al., 2018). AutoAugment is a reinforcement learning algorithm (Sutton & Barto, 2018) that searches for an optimal augmentation policy amongst a constrained set of geometric transformations with various levels of distortions (Shorten & Khoshgoftaar, 2019). An example of such a policy could be ‘rotateX 20 degrees’. Unfortunately, this search involves several levels of stochasticity, which is computationally expensive (Yang et al., 2022).

RandAugment improves upon this by eliminating the algorithm’s search phase and sampling from $K = 14$ available transformations with uniform probability $\frac{1}{K}$. RandAugment supports the following 14 transformations: *identity*, *autoContrast*, *equalize*, *rotate*, *solarize*, *color*, *posterize*, *contrast*, *brightness*, *sharpness*, *shear-x*, *shear-y*, *translate-x*, *translate-y*. Each transformation has a global magnitude ranging from 0 (*no distortion*) to 10 (*maximum distortion*). Given N transformations for a training image, RandAugment expresses K^N potential policies. Several bodies of work (Cubuk et al., 2020; Xie et al., 2020; Yang et al., 2022) have shown RandAugment to not only be faster than previous approaches like AutoAugment but improve the outcomes

significantly, as measured by the testing accuracies, and validation error losses of the models.

Generative Adversarial Networks

Generative adversarial networks (GANs) were first proposed by Goodfellow et al. (2014) and represent a class of neural networks which learn to generate synthetic samples with the same characteristics as the original set. For images, this involves learning to produce images via a Generator, which are so visually like a set of authentic images that an adversary, the Discriminator, cannot detect them (Bowles et al., 2018).

A GAN consists of two adversarial models: a generator model G that captures a data distribution and a discriminative model D that estimates the probability that a sample came from the training set instead of G . The two networks, G and D , are trained simultaneously in a minimax (or *zero-sum*) game, having value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (5)$$

, where G tries to minimize $\log(1 - D(G(z)))$ and D tries to minimize $\log D(X)$, i.e., both functions try to improve their loss function. x is a sample from the real dataset distribution $p_{data}(x)$ and z is sampled from the latent space distribution $p_z(z)$ (Goodfellow et al., 2014).

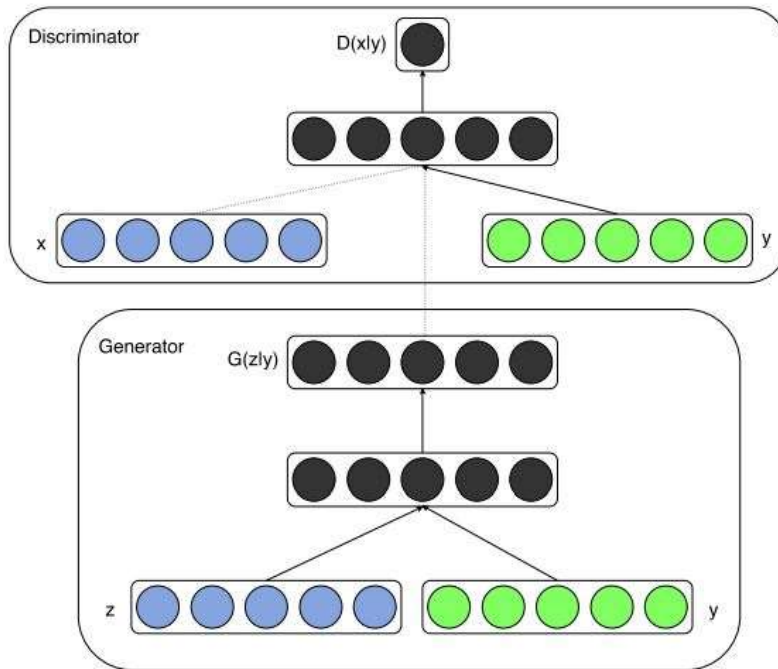
Figure 8 (Mirza & Osindero, 2014) shows that GANs can be extended to a model called Conditional Generative Adversarial Networks (CGAN; Mirza & Osindero, 2014) if both the generator and discriminator are conditioned on an extra input, y , which is typically the class label. This allows considerable flexibility in GANs because, once trained, the Generator can synthesize any number of samples per specified class. E.g., a CGAN can generate 1000 samples of the number 9, “9” *being the class label*

representing the digit 9. Moreover, the objective function for a CGAN is only a slight modification to the one for a GAN:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)} [\log D(x|y)] + E_{z \sim p_z(z)} [\log(1 - D(G(z|y)))] \quad (6)$$

Figure 8

The Architecture of a Simple Conditional Generative Adversarial Network



Note. From “Conditional Generative Adversarial Nets,” by M. Mirza and S. Osindero, 2014, arXiv.

GANs represent a deep learning approach to data augmentation that is widely used across several domains with impressive results. For example, CNNs have been successfully used in the medical field to detect skin lesions (Esteva et al., 2017) and liver lesions (Frid-Adar et al., 2018). These CNNs must be trained using extensive datasets since larger datasets result in better deep-learning models (Sun et al., 2017). However, assembling large datasets in the medical field can be very daunting due to patient privacy,

the rarity of diseases, the medical expertise required for labeling, and the expense and labor needed to collect data. Such factors have led many medical professionals to use GAN-based oversampling for limited datasets (Shorten & Khoshgoftaar, 2019).

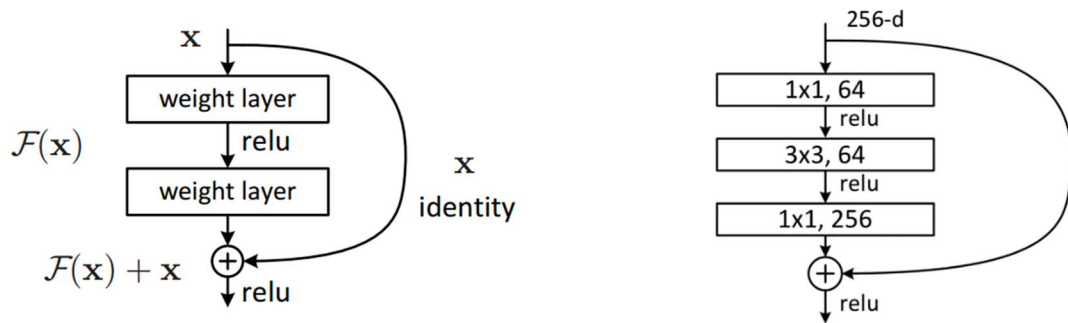
Network Architectures

ResNet

The Residual Network (ResNet; He et al., 2015) is a state-of-the-art CNN introduced in 2015 that achieved a top-5 error rate of 3.57% on ImageNet, beating human-level performance on this dataset and securing first place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015 competition. It was created to address the problem of degradation observed in deeper networks: as network depth increases, accuracy unsurprisingly gets saturated and degrades rapidly. However, what was unexpected was that this degradation was not caused by overfitting, and adding more layers to the network resulted in higher training errors (Srivastava et al., 2015; He et al., 2015). The basic idea of the solution was that a deeper model should not produce a higher training error than its shallower counterpart. The latter implies constructing deeper models by copying the learned layers from the shallower model and setting additional layers to an identity mapping, i.e., instead of attempting to learn a new representation, learn only the residual.

Figure 9

The Residual Blocks of ResNet34 and ResNet50, Respectively



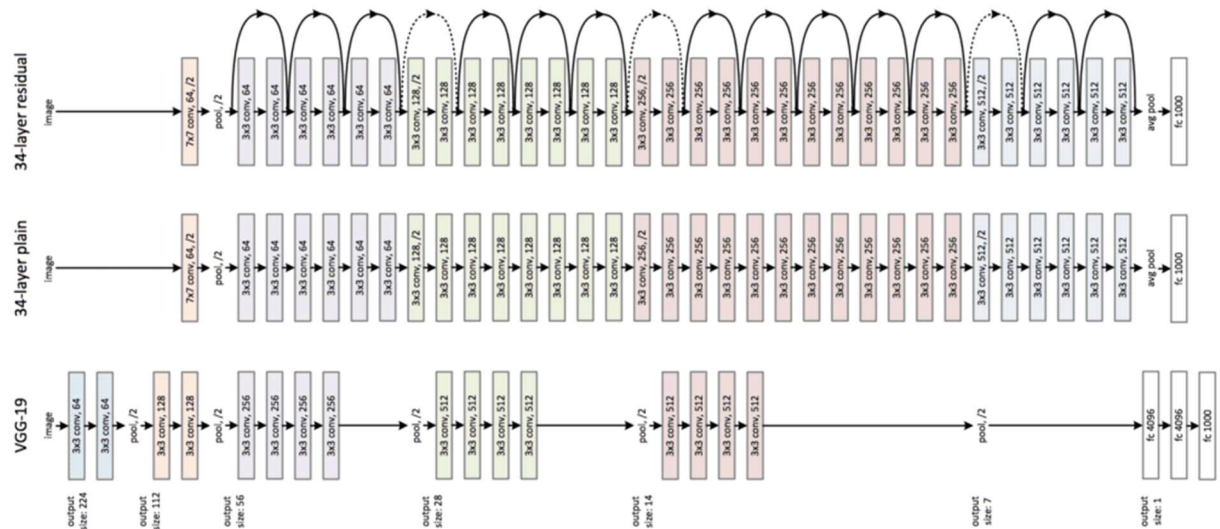
Note. Adapted from “Deep Residual Learning for Image Recognition,” by K. He, X.

Zhang, S. Ren, and J. Sun, 2015, arXiv.

Key to the model design was the “residual block,” as seen in Figure 9 (left; He et al., 2015), that made use of shortcut connections (*connections skipping one or more layers*). Shortcut connections perform identity mapping, and their outputs are added to the outputs of the stacked layers. A residual block comprises of two convolutional layers with ReLU activation, where its output is combined with its input via shortcut connections. This forces a new layer to learn something different from what its input has already encoded. Typically, the size of the input shape to the shortcut connection is the same as the size of the output shape of the residual block. If the shapes are different, 1x1 convolutions, called *projections*, are applied to the input of the shortcut connection.

Figure 10

The ResNet Architecture Inspired by the VGG-19 Model



Note. **Top:** ResNet-34 model. **Middle:** 34-layer plain CNN. **Bottom:** VGG-19 model.

Adapted from “Deep Residual Learning for Image Recognition,” by K. He, X. Zhang, S. Ren, and J. Sun, 2015, arXiv.

To build ResNet, as shown in Figure 10 (**top**; He et al., 2015), the authors started with a VGG-19 (Simonyan & Zisserman, 2014) inspired plain network as seen in Figure 10 (**middle**; He et al., 2015) which has small 3x3 filters, grouped convolutional layers without pooling between them, and average pooling before the fully connected output layer with Softmax activation. This plain network is then modified to create a residual network by adding shortcut connections to define the residual blocks. Furthermore, it was found that building deep networks using ResNets significantly reduces the vanishing gradient problem (Glorot & Bengio, 2010).

ResNet50 (He et al., 2015) is an advanced variant of the ResNet-34 model described above. It is a 50-layer (48 Convolution, 1 MaxPool, 1 Average Pool) deep

CNN that uses 3-layer residual blocks, as seen in Figure 9 (**right**; He et al., 2015), instead of 2-layer residual blocks, which ensures improved accuracy and lesser training time. See Figure E2 (He et al., 2015) for more information on the different architectures of ResNet.

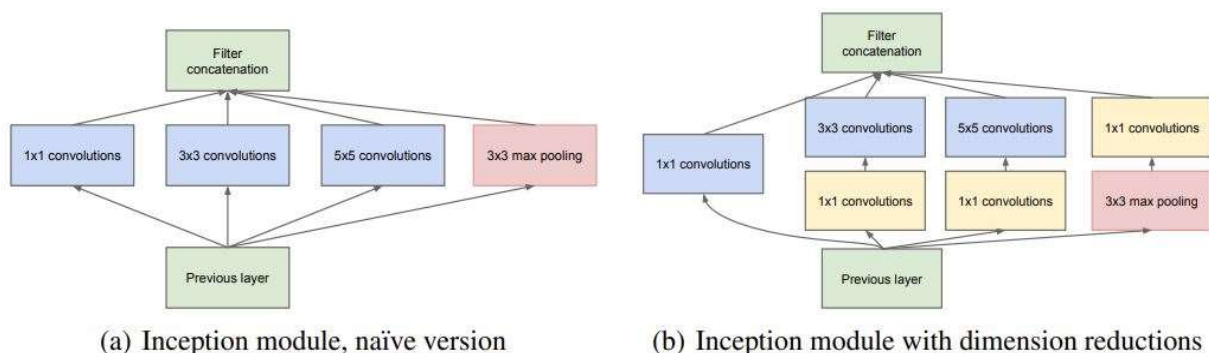
Inception Network

GoogleNet (InceptionV1; Szegedy et al., 2014) is a CNN developed in 2014 by Szegedy et al. (2014) and the winner of ILSVRC 2014. The premise was that salient parts on images could have significant variations in size, e.g., three images of a dog can each have the dog appear in various positions and scales within them. Because of this variation in the location of the relevant information, the kernel size chosen for convolutions becomes critical. Larger kernels are preferred for information distributed more globally, while smaller kernels are preferred for more local distributions. Furthermore, deep networks are subjected to overfitting and degradation, and successive layers of convolution operations are computationally expensive.

The solution was to have filters of multiple sizes operating at the same level. This would make the network “wider” instead of “deeper.” This intuition led the authors to create the “inception module,” as seen in Figure 11 (Szegedy et al., 2014), and the key module in GoogleNet.

Figure 11

The Inception Modules Used by GoogleNet and InceptionV3

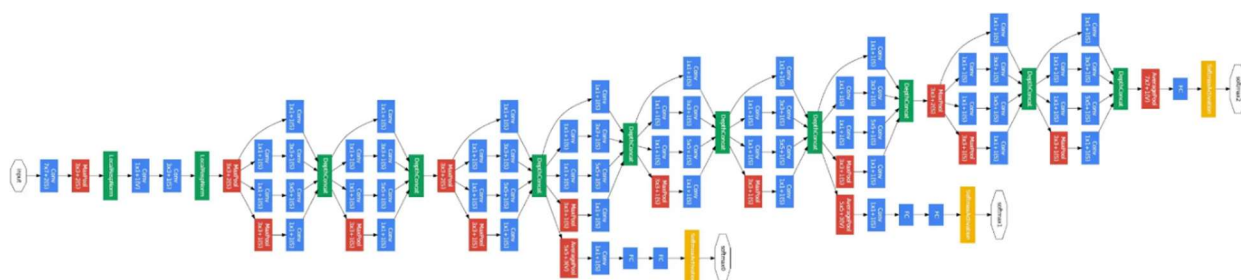


Note. From “Going deeper with convolutions,” by C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, & A. Rabinovich, 2014, arXiv.

Figure 11 (a; Szegedy et al., 2014) shows the “naïve” inception module that performs convolution on input with three different filters (1x1, 3x3, 5x5) and max pooling on the same level. The results are then concatenated and sent to the next inception module. To reduce the computational expense, the authors added an extra 1x1 convolution before the 3x3 and 5x5 convolutions and after the max pooling layer because 1x1 convolutions are cheaper than 5x5 convolutions, and they reduce the number of input channels (dimensions). This can be seen in Figure 11 (b; Szegedy et al., 2014), which shows the inception module with dimension reductions that were used to build GoogleNet (InceptionV1).

Figure 12

The GoogleNet (InceptionV1) Convolutional Network



Note. From “Going deeper with convolutions,” by C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, & A. Rabinovich, 2014, arXiv.

InceptionV1 is shown in Figure 12 (Szegedy et al., 2014) and is 22 layers deep, making it vulnerable to the vanishing gradient problem (Glorot & Bengio, 2010). To address this problem, the authors introduced two auxiliary classifiers; *see the middle of Figure 12*, applied softmax to the output of two inception modules, and computed an auxiliary loss over the same labels. The loss function for the network was then computed as a weighted sum of the auxiliary loss and the real loss (Szegedy et al., 2014).

InceptionV3 (Szegedy et al., 2015) is a 48-layer CNN later developed by the same authors as improvements over InceptionV1 and InceptionV2 by making four main upgrades: RMSProp optimizer, Factorized 7x7 convolutions, Batch Normalization in the Auxiliary Classifiers, and Label Smoothing. As a result, InceptionV3 has achieved impressive performance on image classification tasks and was the 1st Runner Up in ILSVRC 2015. In addition, it is more computationally efficient and can achieve very low error rates on ImageNet compared to its contemporaries (Szegedy et al., 2015).

NoelNet

NoelNet, shown in Figure E6, is a basic but performant CNN the author developed and used to establish baseline metrics from which comparative analysis was made with similar metrics from ResNet50 and InceptionV3. NoelNet comprises 1,091,594 parameters from 9 convolutional layers stacked in three groups, with kernel sizes of 3 and 5, respectively. BatchNormalization follows each CONV layer to minimize overfitting. MaxPooling (*for downsampling*) and Dropout (*to minimize overfitting*) follow each group. Batch Normalization and Dropout also immediately precede the 10-node output layer with Softmax activation.

Summary

Neural Networks are biologically inspired computational models that combine multiple nonlinear processing layers in an acyclic graph that consists of an input layer, zero or more hidden layers, and an output layer. The layers are interconnected via neurons, where neurons in adjacent layers are fully pairwise connected, but neurons within a layer do not share any connection. Several authors have observed that going deeper than three layers in a deep network rarely increase its representational power, which led to a class of neural networks known as deep neural networks, typically having two or more hidden layers. Deep neural networks are excellent at discovering intricate structures in high-dimensional data using deep learning techniques; these techniques automate feature detection in deep models, unlike in traditional networks where these features must be painstakingly hand-engineered by domain experts. Deep neural networks have made considerable progress in many areas, such as computer vision, speech recognition, and natural language processing. Advances in deep network

architectures, powerful computation, and open access to extensive training datasets have fueled that success.

CNNs are specialized neural networks that extract features from data with grid-like topologies, such as images. They typically comprise three main layers: Convolutional layers, Pooling Layers, and a Fully Connected Layer. Neurons in the convolutional layer are combined in feature maps, where each neuron is connected to local patches in the feature maps of the previous layer through a set of weights called a filter (or kernel). Different feature maps use different filters because local groups of values in images are typically highly correlated, making it easy to detect their patterns. Since the same pattern can appear anywhere within an image, neurons within the same feature map share the same weight. The latter imbues convolutional networks with some degree of translational invariance. The pooling layer merges semantically similar features into one via a down-sampling operation (typically max-pooling), which further adds translational invariance and reduces the number of parameters within the network. The fully connected layer is the output layer, where all neurons connect to all the neurons in the previous layer. After passing through the fully connected layers, the final layer uses the softmax activation function to classify the input.

Spatial invariance is a highly desirable property of ConvNets because it allows a network to disentangle the poses and deformations of objects from their texture and shape. For example, this implies that the picture of a cat rotated 10 degrees should still be a cat. Because of their weight-sharing and pooling operations, CNNs are thought to have the property of spatial invariance. Still, several authors have shown this not to be the case, i.e., CNNs are invariant to some but not all transformations. For example,

experiments have demonstrated that the chance of misclassification after translating a random image downward by a single pixel can be as high as 30%, raising serious concerns such as fatalities from autonomous vehicles or racial discrimination by image recognition systems.

One way to improve spatial invariance in CNNs, thereby enhancing their generalizability and reducing overfitting, is data augmentation, a powerful technique used to artificially increase the training set size to capture more sources of invariance in the data. Traditional transformations, such as horizontal flipping, have proven very effective for simple cases but not enough to significantly improve accuracy, reduce overfitting or minimize adversarial attacks in deep networks. However, using advanced data augmentation techniques such as RandAugment and Generative Adversarial can significantly improve the robustness of CNNs because they provide stronger augmentations leading to more diverse samples and greater sources of invariance from which to learn. This dissertation validated these approaches using NoelNet (a base CNN) and two state-of-the-art CNNs: Resnet50 (a 50-layer CNN) and InceptionV3 (a 42-layer CNN), across three benchmark datasets.

Chapter 3

Methodology

This chapter describes the experiments conducted to provide a deep comparative analysis of how data augmentation improves the classification performance of CNNs by using Conditional GANs and RandAugment to increase their spatial invariance. The study conducted 45 experiments. Each experiment represented a data augmentation strategy (a *policy*) performed on a benchmark dataset and validated on a CNN. There were five such policies (Policies 1 through 5) that were implemented using performant data pipelines provided by TensorFlow, three benchmark datasets (MNIST, FMNIST, CIFAR-10), two advanced CNNs (ResNet50 and InceptionV3), and one basic CNN (NoelNet). The datasets, augmentation techniques, CNNs, and policies were developed and configured per the specifications outlined in this chapter. The performance of each trained network was evaluated, *per policy*, using the following metrics: training/validation loss, training/validation accuracy, test accuracy, precision, recall, f1 score, and training latency. Finally, Python was used as the primary programming language for all experiments undertaken by this research.

Models

This section outlines the design specifications for the two deep and one basic network used for experimentation.

CNN Model

One rudimentary ConvNet(NoelNet) and two state-of-the-art ConvNets (ResNet50 and InceptionV3) were used to conduct the experiments in this dissertation. NoelNet, whose architecture is shown in Figure E6, was designed and developed by the author

using Keras (refer to Table B1 for hyperparameters). ResNet50 and InceptionV3 were imported via Keras Applications API and were already pre-trained over the millions of images in ImageNet, an instance of transfer learning. Their initial weights were saved to file before first use, so reusing the model for a new policy was initialized to the same weights. This ensures consistency across comparisons amongst the five policies in this research since each policy had the same starting point with respect to the network weights. The default hyperparameters outlined for each network in their respective papers were used. However, the following modifications were made (refer to Table B1 for hyperparameters)

- The final output layer was removed and replaced with a 10-neuron layer and Softmax activation. This was necessary because ResNet50 and InceptionV3 were trained on 1000 classes on ImageNet, while the baseline datasets used in this research had only 10 classes.
- Image inputs were resized to the minimum required by each network: 32x32x3 for ResNet50 and 75x75x3 for InceptionV3.
- Adam optimizer with an annealed learning rate of 0.001 was used with a categorical cross-entropy loss function, batch size 512, and trained for 100 epochs.

Based on the architectural design and modifications described above for ResNet50 and InceptionV3, Figure E3 and Figure E4 show the resulting architecture and total parameters of each network that this dissertation implemented.

Conditional GAN Model

The CGAN model combined the Generator and Discriminator models into a larger model, which was used to train the weights in the Generator by using the output and error computed by the Discriminator model. The Discriminator model was trained separately, so its weights were marked as non-trainable within the CGAN model. The latter ensured that only the weights of the generator model were updated. This model took the Generator, Discriminator, and point in latent space as input; the Generator produced an image using that point and fed it to the Discriminator, who computed the output as real or fake. The CGAN used the Adam optimizer with a learning rate of 0.0002 and momentum of 0.9 for both the Generator and Discriminator, as Radford et al. (2015) suggested, and applied the binary cross-entropy loss function. The following sub-components were built and combined to create the final CGAN model: Discriminator, Generator, and Embedding Layer.

- **Discriminator.** The Discriminator was implemented as a moderate CNN using GAN best practices such as LeakyReLU activation function with a slope of 0.2, 2x2 stride to downsample, and Adam optimizer with a learning rate of 0.0002 and momentum 0.9. Each discriminator took as input one 28x28 grayscale image (MNIST, FMNIST) or one 32x32 color image (CIFAR-10) and an integer class label of the image and output a binary classification of whether the image was real (*class=1*) or fake (*class=0*). The CIFAR-10 model had three CONV layers with 5x5 kernels, a stride of 2, and same padding. It had 64 kernels in CONV1, 128 in CONV2, and 256 in CONV3. LeakyReLU followed each CONV layer. Similarly, Batch Normalization followed each CONV layer except for the first layer. Finally, a dropout of 40% was used before the one-unit output layer. The

F/MNIST model had the same setup, except two CONV layers were used, both with 128 filters. See Figure E1(a) for design specifications.

- **Generator.** The Generator model was implemented as a moderate CNN and took as input a point in latent space and the number of classes and output a single 28x28 (MNIST, FMNIST) or 32x32 (CIFAR-10) image. Following the input layer was a fully connected layer to interpret the point in latent space with $7 * 7 * 128$ activations for F/MNIST or $4 * 4 * 256$ activations for CIFAR-10. These activations were then reshaped into many copies of a low-resolution version of the output image (in this case, 128 for F/MNIST and 256 for CIFAR-10). For CIFAR-10, these activations were up-sampled using three sequential transpose convolutional layers with 256, 128, and 64 5x5 kernels, respectively, with a stride of 2 and the same padding. LeakyReLU followed each CONV layer with a slope of 0.2 and Batch normalization with a momentum of 0.9. Finally, the output layer consisted of a convolutional layer with one 5x5 kernel, tanh activation, and same-padding. See Figure E1(b) for design specifications.
- **Embedding Layer.** As a best practice, an embedding layer was used to encode class labels into the Discriminator and Generator models. A fully connected layer followed this layer with linear activation that scales the embedding to the image size before concatenating it in the model as an additional feature map (Denton et al., 2015).

Optimization

During the training of each ConvNet, several callback functions provided by Keras API were utilized, such as: early stopping if the validation loss had not improved

after 40 epochs, annealing of the learning rate by a factor between 2 and 10 (Smith, 2017), if no improvement in validation loss after 10 epochs and saving the best-performing weights (*measured by the minimum validation loss across all epochs*) of the model to file under filename `<dataset_model_policy{policy_number}_model>.h5`, e.g., `mnist_resnet_policy1_model.h5`. See Appendix G, “Dissertation Source Code,” for information on the location and access of the trained model files in *h5* format.

Experimental Design

Datasets

This research used six datasets for experimentation, with the following three as benchmark datasets: MNIST, FMNIST, and CIFAR-10 imported from TensorFlow Datasets API. The other three datasets were GAN-synthesized from the benchmark datasets and shared the same image size, structure, classes, and number of samples as their benchmark counterparts. See Figure D1 for sample images of the benchmark datasets.

- MNIST: a dataset of handwritten digits consisting of 10 categories. It has a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image associated with a label from 10 classes (0 to 9). See Table D4 for class labels and descriptions.
- FMNIST(FashionMNIST): a more complicated version of MNIST consisting of fashion categories. It has a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image associated with a label from 10 classes (0 to 9). See Table D2 for class labels and descriptions.

- CIFAR-10: a dataset of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images. See Table D3 for class labels and descriptions.
- MNIST_GAN: a dataset of 60,000 GAN-synthesized MNIST samples in 10 classes, with 6,000 images per class. It had a training set of 60,000 examples, and its test dataset was the 10,000 test samples from the benchmark MNIST dataset.
- FMNIST_GAN: a dataset of 60,000 GAN-synthesized FMNIST samples in 10 classes, with 6,000 images per class. It had a training set of 60,000 examples, and its test dataset was the 10,000 test samples from the benchmark FMNIST dataset.
- CIFAR10_GAN: a dataset of 50,000 GAN-synthesized CIFAR-10 samples in 10 classes, with 5,000 images per class, and its test dataset was the 10,000 test samples from the benchmark CIFAR-10 dataset.

Data Preparation

The datasets consist of pairs (x, y) where x is an input example, and y is its label. The training set was used to optimize the model's parameters with backpropagation. The validation set was used for hyperparameter optimization during training. Finally, the test set was used *after* training to validate the model's performance on unseen data. The benchmark datasets provide only training and testing datasets by default, but no validation set. As a result, a custom data partition function was used to split the training set in a 90:10 ratio: 90% of the training set for training and 10% for validation. The GAN-synthesized datasets followed the same training/validation/test splits. See Figure A3 for the data partition algorithm.

Data Preprocessing

Preprocessing the datasets can assist in improving the convergence of neural networks (LeCun et al., 1998). The benchmark datasets used for this research contain size-normalized and centered samples but require a few modifications before being used by each CNN. First, a custom preprocessing function converted each image to float32 datatype, normalized in the range [0,1], and resized to the minimum input volume required by ResNet50 and Inception: 32x32x3 and 75x75x3. Finally, the class labels of each dataset were one-hot encoded. See Figure A4 for the preprocessing algorithm.

Data Augmentation Methods

This dissertation used two advanced data augmentation techniques: Conditional GANs and RandAugment. cDCGAN augmentation generated synthetic samples, and RandAugment stochastically chose and applied various geometric transforms to the dataset. Augmentation was **ONLY** applied to the training dataset, leaving the validation and test sets unperturbed. This was because only the training dataset tunes the model's parameters during training. The testing and validation datasets evaluate the model's performance during inference (*after training*) using metrics like accuracy, precision, recall, and f1 score (Shorten & Khoshgoftaar, 2019).

- **RandAugment.** Several Python libraries have implemented RandAugment, such as Keras (Chollet, 2015) and TensorFlow (Abadi et al., 2016). RandAugment was imported via TensorFlow's Vision library for this body of research and added to each network's data input pipeline when implementing Policy 2 and Policy 5. It took two required parameters: N (number of random transforms to apply sequentially) and M (strength of each transform). Each sub-policy consisted of two operations. The transformation name, application probability at each layer,

and magnitude specific to that operation represented each operation. For example, a sub-policy can be [(Rotate, 0.6, 2), (Translate, 0.3, 9)]. For each operation, we randomly sampled a transformation from 14 possible transformations, a magnitude in [1, 10], and fixed the probability to 0.5. We followed the approach Cubuk et al. (2020) suggested and set the hyperparameter N to equal two transforms with magnitude nine, i.e. ($N = 2$ and $M = 9$). Specifically, we sampled from the following 14 transformations: *identity*, *autoContrast*, *equalize*, *rotate*, *solarize*, *color*, *posterize*, *contrast*, *brightness*, *sharpness*, *shear-x*, *shear-y*, *translate-x*, *translate-y*. See Figure A2 for an implementation of the RandAugment algorithm in Python.

- **Conditional DCGAN.** The cDCGAN was built using Keras Sequential API as described in this chapter's *Conditional GAN Model* sub-section and trained separately on each benchmark dataset. A total of 1000 epochs was used for training, as this number was observed to produce visually plausible samples to the original (Perez & Wang, 2017). To minimize the effects of overfitting, one-sided label smoothing was implemented as recommended by (Szegedy et al., 2015; Salimans et al., 2016). Samples after each training epoch and samples after convergence demonstrated that the CGAN model was not simply producing samples via memorizing training examples (Denton et al., 2015). At the end of the training, the generator model was saved to a file with the filename `<dataset>_gan.h5`, where `<dataset>` represents the benchmark dataset used to train the GAN, and h5 represents TensorFlow's file format for saving models. For example, `mnist_gan.h5` would indicate a cDCGAN trained on MNIST. This

model was then loaded to generate new, class-specific, but plausible samples for the respective dataset. The proposed cDCGAN was adapted from Paul (2021). Refer to Table B1 for the hyperparameters used in building and training the conditional DCGAN.

Data Pipeline

A performant data input pipeline (see Figure E5) was developed to set up and configure each dataset, train the model on this dataset, evaluate the model's performance, then save the evaluation metrics to file. This pipeline was used by each policy and consisted of the following stages:

1. A dataset was loaded via TensorFlow's datasets library or synthesized using the cDCGAN generator.
2. The training dataset was split as described in the *Data Preparation* sub-section of this chapter.
3. Each dataset sample was mapped to a sample preprocessing function that prepared the sample for input to the CNN. See Figure A4 for the sample preprocessing algorithm.
4. Each dataset was passed to a dataset preprocessing function to generate its data input pipeline. The preprocessing function also accepted a *shuffle* and *augment* parameter that indicated whether the dataset should be shuffled or augmented. See Figure A5 for the dataset preprocessing algorithm.
5. The CNN was imported via Keras Applications API and configured per the specifications defined in the *CNN Model* sub-section of this chapter.

6. The CNN was trained using the pipeline's training and validation datasets from stage 4. After training, the result was stored in an object variable and used for metrics computations. In addition, the model was enclosed within a timer function that started when training began and ended when training stopped. The difference between the stop and start time was recorded as the *training latency*.
7. The training result from stage 6 was used to plot two graphs: *training/validation loss* and *training/validation accuracy*. These graphs were used to analyze model overfitting and were saved to file.
8. The trained model was then evaluated on the test dataset, and its result was saved to an object variable for use in the next stage.
9. The result from stage 8 was used to compute the model's test accuracy, precision, recall, and f-1 score. Together with the training latency, these metrics were exported to a CSV file for later analysis. The exported file had filename `<dataset_model>_metrics.csv`, for example, `mnist_resnet_metrics.csv`. See Appendix F for a complete list of exported fields and graphs.

Evaluation Metrics

The following metrics were used in this research to analyze the performance, and robustness of ResNet50, and InceptionV3, across the five augmentation policies. These metrics have been used in extensive bodies of work to measure invariances, such as those by Arjovsky et al. (2019), Goodfellow et al. (2009), Krizhevsky et al. (2012), LeCun et al. (1998), and Szegedy et al. (2014).

Arjovsky et al. (2019) have postulated that robust learning is equivalent to minimizing training errors or maximizing training accuracies, which leads to spatially

invariant deep networks. Consequently, networks with higher training or testing accuracies are more spatially invariant than networks with lower accuracies. Several authors, including Chen et al. (2020) and Gowal et al. (2020), corroborated the latter using testing accuracy to measure invariance in deep networks. Their experiments provided state-of-the-art performance in deep networks, accomplished across many datasets, using sophisticated data augmentation techniques such as RandAugment. As a result, this dissertation used testing accuracy as the primary measure of spatial invariance in ConvNets. In addition, if two policies had equal testing accuracy, the f1 score, which combined precision and recall, was used to perform a more granular analysis.

- **Training/Validation Loss.** As shown in Figure 6 (Shorten & Khoshgoftaar, 2019), this represents the gap between the training and validation loss of the model. Larger gaps indicate strong overfitting; *may require more regularization, such as increased augmentation or dropout*. Conversely, smaller gaps indicate little overfitting and may require increasing the number of parameters to make the models larger.
- **Training/Validation Accuracy.** This metric is like training/validation loss, except the training and validation accuracy is measured. It also gives insights into the amount of overfitting in the model. See Figure 7 (Li et al., 2015) for an overview.
- **Test Accuracy.** This metric describes how well a given model performs across all classes, and it is useful when all classes are equally important as they are for this research. It measures the ratio between the number of correct predictions to the total number of predictions.

- **Precision.** This metric attempts to answer the question concerning the correct proportion of identifications by measuring a model’s accuracy in classifying a sample as positive.
- **Recall.** This metric attempts to answer the question concerning what proportion of actual positives was identified correctly by measuring a model’s ability to detect positive samples.
- **Training Latency.** This metric measures the time it takes to train a model over a given dataset for a specified number of epochs. Since training a deep neural network can be challenging, time-consuming, and expensive, quickly trained models are generally regarded as more performant.
- **F1 score.** This metric conveys the relative balance between precision and recall and thus is typically used to measure the relative performance of one model against another. It is the harmonic balance between the precision and recall of a model, i.e.,

$$F1Score = 2 * \left(\frac{precision * recall}{precision + recall} \right) \quad (7)$$

Experiments

The following is a detailed description of each of the five policies evaluated by this dissertation. See Table F1 for an overview of the expected results.

1. **Policy 1** served as the baseline for each experiment. It established a reference for loss, accuracy, precision, recall, f1-score, and training latency for all experiments because it did not use any form of data augmentation. For a given dataset and CNN architecture, this policy executed all stages of the input pipeline without any pre- or post-modifications to any stage. The training,

validation, and testing datasets generated in stage 2 of the pipeline were saved as TensorFlow datasets to be used by the following policies.

2. **Policy 2** evaluated data augmentation using geometric transformations applied to a given dataset. However, instead of using manual “hand-coded” transforms, it used RandAugment to generate transformation policies stochastically. First, it set the “augment” flag in Stage 4 of the pipeline to “true” to activate the RandAugment function within that stage. Once activated, the input pipeline generated augmented samples fed into the chosen CNN. The implementation of RandAugment used for this policy was described earlier in this chapter.
3. **Policy 3** evaluated data augmentation using a deep generative model called a Conditional DCGAN to synthesize additional training examples for a given dataset. The implementation of the CGAN for this policy was described earlier in this chapter. The CGAN was first trained using the pipeline's training dataset generated in Stage 2. Once trained, its generator was used to synthesize labeled training examples equal to the number of samples in the training dataset. Finally, these synthesized samples were saved as a TensorFlow dataset, used as the training dataset for stage 3 of the pipeline, and later used in Policies 4 and 5.
4. **Policy 4.** Policy 3 + Policy 1. This policy evaluated data augmentation by concatenating the GAN-generated training dataset from Policy 3 with the training dataset from Policy 1. The resulting dataset was then used as the pipeline's training dataset for stage 3.

5. **Policy 5.** Policy 3 + Policy 2. This policy evaluated data augmentation by stacking different augmentation styles. It applied RandAugment to the GAN-generated training dataset from Policy 3. The GAN-generated training dataset was used as the training dataset for stage 3 of the pipeline, and the augment flag in stage 4 was set to “true” to activate the RandAugment function.

Data Analysis

The metrics recorded from each policy were used to perform a rigorous comparative analysis using advanced data augmentation techniques to improve spatial invariance and reduce overfitting in CNNs. A minimum of the following questions was addressed as part of this analysis.

1. Does data augmentation lead to more spatially invariant and robust networks?
2. Are deep architectures more spatially invariant than wide architectures? Did a particular architecture perform better?
3. Does generative data augmentation lead to more diverse samples than reinforcement learning techniques? Policy 2 vs. Policy 3.
4. Do larger datasets increase spatial invariance? (Policy 2,4,5 vs. Policy 1 and 3)
5. Do stacking data augmentation methods lead to more robust networks? Policy 5.
6. Does combining synthesized samples with original unaugmented samples improve invariance in CNNs? Policy 4
7. Do synthesized samples improve invariance? Policy 3
8. Do Stochastic techniques based on Reinforcement learning techniques improve invariance? Policy 2

9. Does the number of parameters in a CNN affect its spatial invariance as measured by its test accuracy?

Results Format

This dissertation reported the results from each experiment using graphs, tables, and charts. Line graphs showed the training/validation loss, as seen in Figure 6 (Shorten & Khoshgoftaar, 2019), and the training/validation accuracy, as seen in Figure 7 (Li et al., 2015). The performance metrics of the model were computed using the metrics module from scikit-learn (Pedregosa et al., 2011). The metrics were aggregated in tabular format using Pandas (Wes, 2010), then exported to a CSV file with fields described in Appendix F. In addition, the accuracy, f1 score, precision, and recall were displayed as percentages with two decimal places. Training latencies were measured in seconds. Finally, bar charts provided relative metrics comparisons across policies, datasets, and architectures.

Resource Requirements

This dissertation used standard, readily available, open-source, and off-the-shelf personal computer hardware and software components. A personal computer with an Intel® i7-7700K quad-core liquid-cooled CPU, 32GB DDR4 RAM, 8GB GDDR5X NVIDIA GeForce GTX 1080 GPU, and 512GB NVMe SSD hard drive was available for this research. Furthermore, Python 3.9 (Van Rossum, 1995) served as the primary programming language, in conjunction with scikit-learn for predictive data analysis, TensorFlow (Abadi et al., 2016) for machine learning, Keras high-level API (Chollet, 2015) as an interface to TensorFlow, Matplotlib (Hunter et al., 2007) for visualization, NumPy (Harris et al., 2020) and Pandas for processing and manipulating large datasets.

Nova Southeastern University bore no financial expense for this research. For a complete list of third-party libraries used in this dissertation, refer to Appendix C.

Summary

This chapter delineated five data augmentation strategies called policies: Policy 1 (baseline policy with no data augmentation), Policy 2 (data augmentation using RandAugment applied to the unaugmented samples), Policy 3 (data augmentation using Conditional GAN to synthesize training samples), Policy 4 (data augmentation using CGAN samples combined with unaugmented samples), Policy 5 (data augmentation by applying RandAugment on GAN generated samples). Their results were used to perform a deep comparative analysis of how advanced data augmentation techniques improve the predictive robustness of CNNs by improving their spatial invariance using the following metrics: training/validation accuracy, training/validation loss, test accuracy, recall, precision, f1 score, and training latency. The experiments used three benchmark datasets imported via TensorFlow: MNIST, FMNIST, and CIFAR-10. Of the two advanced data augmentation techniques, RandAugment was imported via Keras API, and the Conditional GAN was built using Keras Sequential API with a modestly designed Generator and Discriminator. Finally, NoelNet was built using Keras, and the two ImageNet pre-trained CNNs, ResNet50 and InceptionV3, were imported via Keras API.

RandAugment is an augmentation technique that stochastically applies two geometric transforms with a magnitude of distortion of 9 to each dataset. The Conditional GAN was first trained on each dataset using their respective training datasets. Its generator then synthesized equivalently sized training datasets, which were used by policies 3, 4, and 5 for training the CNNs. Both network architectures used their default

hyperparameters with the following changes: 10-neuron output layer instead of 1000, Adam optimizer with $1e-3$ learning rate, and categorical cross-entropy loss function. Image samples were resized to the minimum size required by each network and were normalized in the $[0,1]$ range. The networks were trained for 100 epochs, using 512 batch sizes, where training stopped after 40 epochs if there was no improvement in validation loss.

Additionally, the learning rate was reduced by a factor between 2 and 10 if there was no improvement in the validation loss after ten epochs. Each policy was implemented over nine stages using a highly performant data input pipeline from TensorFlow's data API: Stage 1 (load the dataset), Stage 2 (partition the dataset into train/validation/test splits), Stage 3 (preprocess dataset samples), Stage 4 (configure dataset using batching, prefetching, shuffling, augmenting), Stage 5 (import and configure CNN), Stage 6 (train CNN using training dataset), Stage 7 (evaluate CNN performance on training/validation data), Stage 8 (evaluate CNN on the testing dataset), and Stage 9 (compute and save performance metrics to file). The results of each experiment were reported in graphical and tabular formats using percentages for scalar metrics and seconds to measure time. A personal computer with highly performant components (liquid-cooled quad-core CPU, 8GB GTX 1080 GPU, 32GB RAM, 512GB SSD storage) was used to conduct each experiment in conjunction with Python 3.9, TensorFlow, and Keras providing the software components.

Chapter 4

Results

This chapter presents the 45 experiments conducted using six datasets (three benchmarks and three synthetics), three ConvNets (NoelNet, ResNet50, and InceptionV3), and five augmentation policies to support improvements in the spatial invariance of ConvNets using advanced augmentation techniques.

NoelNet is a simple (*under 2 million parameters*) but performant CNN with 9 Convolution layers developed by the author and was used first to achieve competitive accuracies and f1 scores on each dataset (at least 0.99 for MNIST, 0.92 for FMNIST, and 0.85 for CIFAR-10) without data augmentation (Policy 1). It was then used to evaluate the other augmentation policies (Policies 2 through 5) and present their results. Finally, Policies 1 through 5 were evaluated using ResNet50 and InceptionV3, and their results were presented and analyzed to determine if similar patterns were observed relative to NoelNet.

The results are shown in tabular and graphical formats and are used to address the data analysis questions presented in Chapter 3. The results were also used for hyperparameter optimization while training the DCGAN, ResNet50, and InceptionV3. Parameter tuning was necessary to prevent issues like gradient explosion; where the training losses spike to very large values, thus skewing the loss and accuracy curves and mode collapse; where a GAN prefers a subset of its training data instead of the entire dataset.

The former resulted in training ResNet50 and InceptionV3 at least 20 times each until the effects of gradient explosion were minimized. The latter resulted in training the

DCGAN at least 40 times and choosing the model with the highest test accuracy and most visually appealing synthesized images. The final set of hyperparameters used across all experiments is provided in Appendix B, "Hyperparameters."

The author's GitHub account shows all experiments' source code and raw data. Refer to Appendix G, "Dissertation Source Code," for more information on accessing the source code and raw data. In addition, the raw data is saved in CSV format, including performance metrics and training history. Refer to Appendix F, "Experimental Results," for a more detailed description of the data points captured and the results format.

Conditional DCGAN Training

Three of the six datasets used for experimentation were synthesized using a conditional DCGAN, and Policies 3 through 5 use these synthetic datasets. As a result, a conditional DCGAN was built and then trained on the three benchmark datasets. Figure E1 shows the final cDCGAN architecture used by each dataset, and Appendix B, "Hyperparameters," details the final hyperparameters used in both the generator and discriminator of the GAN.

The GAN model was trained at least 30 times on CIFAR-10 and 10 times on F/MNIST for 200 epochs using various combinations of hyperparameter values to select the most optimal hyperparameters. The model that resulted in the highest training accuracy and most visually appealing images for each dataset was saved and then retrained for 1000 epochs on their respective dataset. After 1000 training epochs, the resulting model was saved to file and used as the final model to synthesize samples for training NoelNet, ResNet50, and InceptionV3.

Figure D2 shows sample images generated by the cDCGAN at epochs 1, 10, 100, and 1000. At epoch 1 for each dataset, the images contain mostly unrecognizable blurbs and noise. From epochs ten and onwards, F/MNIST synthesized samples are easily recognizable and have a relatively equivalent quality to their training set. For CIFAR-10, more defined shapes, colors, and edges begin to take shape by epoch 10. At epoch 100, images such as automobiles, horses, cars, and airplanes become more recognizable. Finally, by epoch 1000, most images within each class are recognizable.

The progression in image quality from epoch 1(worst quality) to epoch 1000 (best quality) is significant because it shows that the GAN is not simply memorizing the images. If that were the case, the images at Epoch 1 would have the same high quality as those at Epoch 1000. Instead, the GAN is learning to generate images during training and without hand-engineered features from external sources. Also evident from Figure D2 is that the quality of the generated CIFAR-10 images is poorer than those of the generated F/MNIST images. The latter is mainly because CIFAR-10 images are 32x32 color images with more complex features, while F/MNIST are 28x28 grayscale images with simpler features.

Figure D3 shows a sample of the cDCGAN generated images and their respective loss after 1000 training epochs. From Figure D3, the DCGAN stabilized after approximately 500 epochs of training for each dataset, implying the generator and discriminator have achieved a Nash equilibrium, i.e., the generator has a 50% chance of fooling the discriminator, and the discriminator has a 50% chance of detecting fake images from the generator. This result agrees with the GAN loss function described by

Equation 6, and further training using the cDCGAN would likely not improve image quality.

cDCGAN Training Latency

According to the literature, training GANs is notoriously difficult and time-consuming. This assertion is supported by the long training latencies in Table 1 for the cDCGAN trained on each dataset for 1000 epochs.

Table 1

Training Latencies for the Conditional DCGAN

	latency (s)	latency (hrs.)	latency per epoch (s)
MNIST	32493	9.026	32.493
FMNIST	31189	8.664	31.189
CIFAR-10	41526	11.535	41.526
Total	105208	29.224	

Table 1 shows that the cDCGAN trained on CIFAR-10 had the longest training time, taking over 11.5 hours to train. This is followed by the DCGAN trained on MNIST, which took just over 9 hours to train. The DCGAN trained on FMNIST took the shortest time, at 8.664 hours, to train. A total of 29.22 hours of training was taken to generate the three required cDCGAN, one for each dataset. The GAN trained on CIFAR-10 took 33.14% longer to train than the one trained on FMNIST, while MNIST took 4.18% longer. CIFAR-10, having the longest train time, is in line with expectations since its images have more complex features than those of F/MNIST.

On the other hand, MNIST took approximately 4% longer than FMNIST, which could be due to the distinctness of its class labels and images compared to FMNIST class labels and images. For example, FMNIST has a few similar classes in description and

image, such as T-shirt/top and Shirt or Sneaker and Ankle Boot. This implies that the cDCGAN learns the features of FMNIST slightly faster than that of MNIST.

MNIST on NoelNet

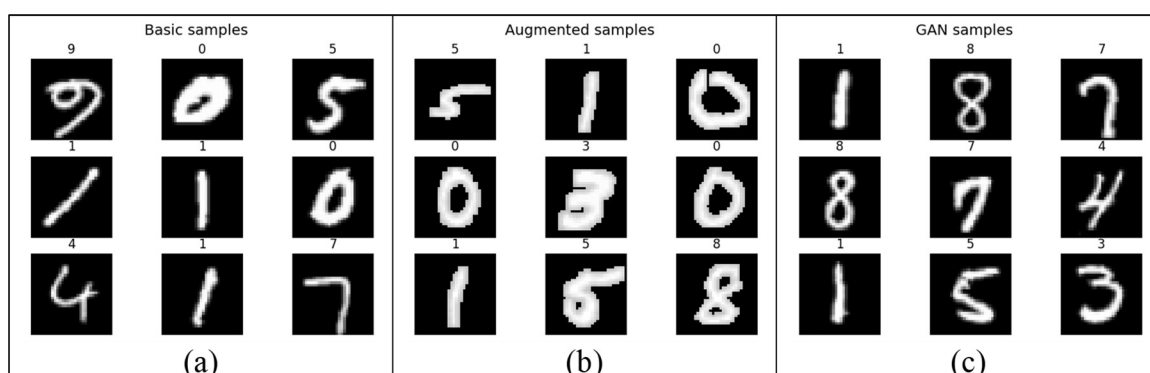
The results obtained after analyzing each policy for the MNIST dataset on NoelNet are as follows.

Dataset Images

Figure 13 shows samples of three variations of the MNIST dataset used in the experiments, either by themselves or in combinations thereof. Figure 13(a) shows the unperturbed dataset used by Policy 1 to establish baseline metrics. Figure 13(b) showed samples after being augmented by RandAugment and was used by Policy 2. Figure 13(c) shows the cDCGAN generated samples used by Policy 3. Finally, Policy 4 combined samples shown in Figure 13(a and c), and Policy 5 applied RandAugment to samples shown in Figure 13(c).

Figure 13

Sample Images of Unaugmented, Augmented, and GAN-Generated MNIST



Note. Samples shown in (a) represent the benchmark or unperturbed dataset. (b) shows samples that were augmented using RandAugment. (c) shows samples generated using a conditional DCGAN.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 2

Performance Test Metrics of each Policy for MNIST on NoelNet

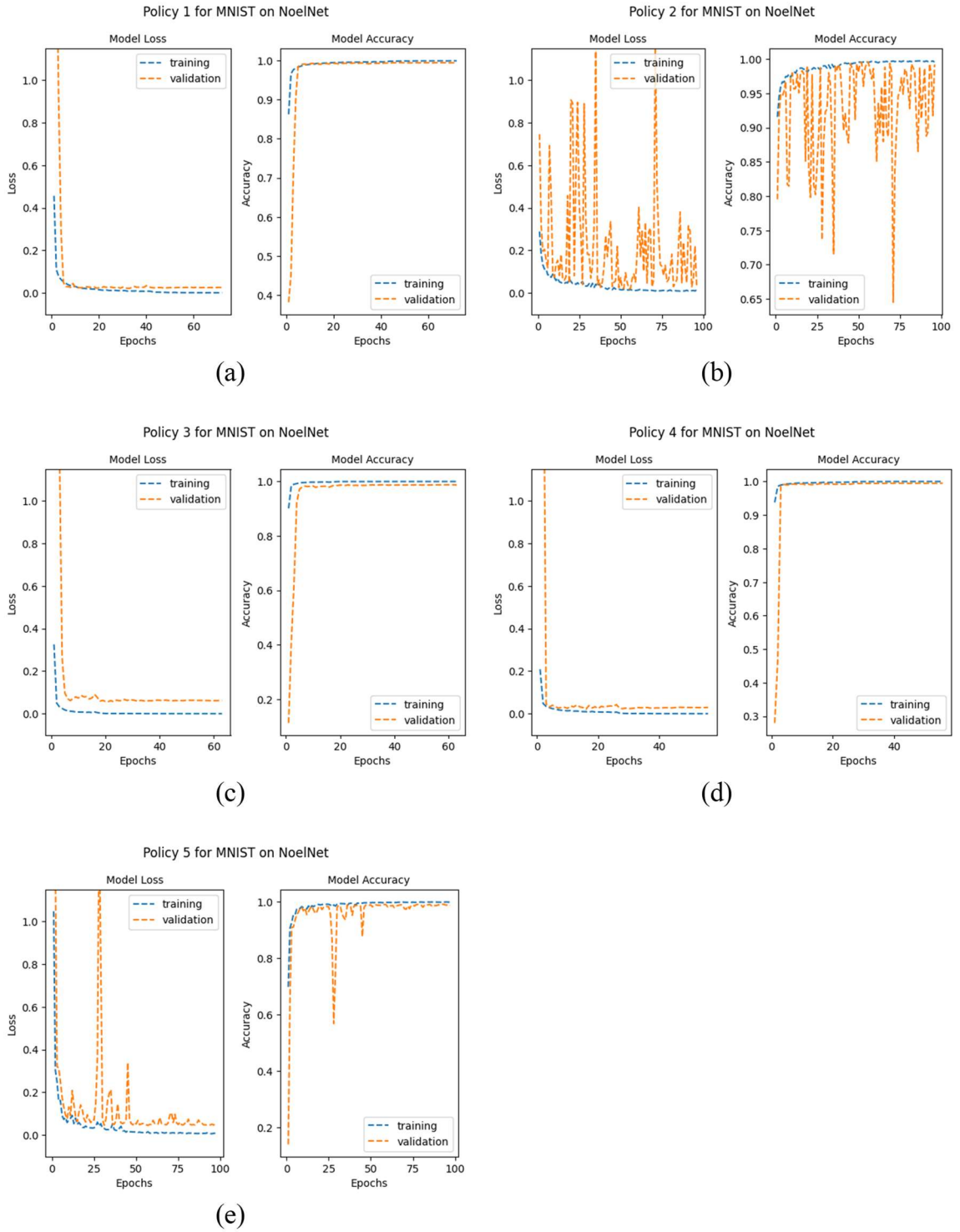
dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
mnist	noelnet	1	0.9943	0.9944	0.9942	0.9943	0
mnist	noelnet	2	0.9961	0.9961	0.9961	0.9961	0.18
mnist	noelnet	3	0.9894	0.9894	0.9894	0.9894	-0.49
mnist	noelnet	4	0.9944	0.9944	0.9943	0.9944	0.01
mnist	noelnet	5	0.9913	0.9915	0.9913	0.9914	-0.3

Note. The table displays the performance metrics for the test samples, where the delta column shows the percentage change in accuracy for each policy relative to Policy 1. Higher percentages imply improved performance and generalizability.

Table 2 summarizes the performance of each policy for MNIST on NoelNet, where Policy 1 established a baseline. Policy 2 and Policy 4 had accuracies of 99.61% and 99.44%, respectively, which translated to improvements of 0.18% and 0.01%, respectively, relative to the baseline. In contrast, policies 3 and 5 had accuracies less than the baseline, 98.94% and 98.92%, respectively, which translated to decreases of 0.49% and 0.51%, respectively. The veracity of the accuracy for each policy is further confirmed by the f1 score being, on average, within 0.002% of each policy's accuracy. From Table 2, it can be surmised that Policies 2 and 4 increased the spatial invariance of the model, while Policies 3 and 5 reduced the spatial invariance of the model.

Figure 14

Accuracy and Loss Graphs of each Policy for MNIST on NoelNet



Policy 1

The purpose of Policy 1 was to establish a baseline for the metrics by training the model on the unperturbed dataset. This baseline was used as the control over which all other policies and metrics were compared to determine if there were any improvements in spatial invariance. Figure 14(a) shows the loss and accuracy curves for Policy 1. The chart shows that the validation graphs follow the training graphs very closely and are coordinated, so the gap between the graphs is minimal, suggesting negligible overfitting. The model reached convergence at epoch 72, when the validation loss reached its minimum and training was terminated. From Table 2, Policy 1 achieved an accuracy of 99.43% and an f1 score of 99.43%. The f1 score is the same as the accuracy, which suggests that the class distribution is balanced and corroborates the model's accuracy as a "true" accuracy, i.e., the results produced by the model correctly reflect the performance of the model and the predictions made by the model should be at least 99% accurate.

Policy 2

The purpose of Policy 2 was to analyze the effect of augmenting the dataset using RandAugment to improve spatial invariance within the ConvNet. Figure 14(b) shows the loss and accuracy graphs for Policy 2. Despite the very noisy validation curves, the accuracy and loss graphs show a very tight correlation in their relative patterns. This noisiness is due to the increased stochastic perturbations generated by RandAugment. The gap between each curve is smaller or tighter than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is smaller than in Policy 1. The model also had a slower convergence rate (~1.3x slower) compared to Policy 1 in that training terminated at Epoch 96 instead of Epoch 72 as it was for Policy 1.

From Table 2, Policy 2 achieved an accuracy of 99.61% and an f1 score of 99.61%. This accuracy represents a 0.18% improvement over Policy 1 and the unperturbed dataset. This improvement in accuracy may be negligible but significant in that almost any sensible ConvNet can quickly learn the MNIST dataset. These ConvNets, when trained on MNIST, can consistently achieve over 99% accuracy without optimizations. As a result, any techniques leading to further improvements in performance are noteworthy.

Policy 3

Policy 3 was used to synthesize samples of the dataset using a conditional DCGAN and use these samples to train the model to improve the model's spatial invariance. Theoretically, the cDCGAN samples should possess increased invariances, which can reduce the level of overfitting in a model and increase its generalizability. Figure 14(c) shows the loss and accuracy graphs for Policy 3. The graphs can be seen to have more significant gaps as compared to policies 1 and 2, suggesting the synthetic samples cause higher levels of overfitting. The model converged at epoch 63, which was faster than that of policies 1 and 2. From Table 2, Policy 3 achieved an accuracy of 98.94% and an f1 score of 98.94%, representing a 0.49% decrease from Policy 1. The latter implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

The purpose of Policy 4 was to demonstrate that by combining the cDCGAN synthesized samples with the unaugmented samples, an improvement in spatial invariance can be achieved. Figure 14(d) shows the loss and accuracy graphs for Policy 4.

The gaps in the charts are comparable to those of Policies 1 and 2 and smaller than Policy 3, indicating that Policy 4 had similar overfitting levels to Policies 1 and 2 but less overfitting than Policy 3. The model converged at epoch 56, faster than Policies 1, 2, and 3. From Table 2, Policy 4 achieved both accuracy and an f1 score of 99.44%, representing a 0.01% improvement in accuracy over Policy 1. Though negligible, this improvement over the baseline model suggests that Policy 4 improves spatial invariance. This improvement directly results from the additional invariances provided by the GAN samples when combined with the baseline dataset. When used by themselves and not merged with any other dataset, the synthesized samples from Policy 3 led to a decrease in spatial invariance. However, when combined with their baseline counterpart, as seen in Policy 4, it increases the spatial invariance of the trained model.

Policy 5

The purpose of Policy 5 was to demonstrate that stacking or combining different augmentation techniques can lead to improvements in spatial invariance in ConvNets. This policy applied RandAugment to the cDCGAN synthesized samples, and the resulting loss and accuracy graphs are shown in Figure 14(e). The validation curves are less noisy than Policy 2 but still benefit from the increased stochastic perturbations generated by RandAugment. The gap between each graph is smaller (has less overfitting) than those of Policy 3 but slightly wider than those of Policies 1, 2, and 4. This suggests that applying RandAugment to synthesized samples can improve spatial invariance compared to using the synthesized samples in isolation. This observation is corroborated by Table 2, where Policy 5 has an accuracy of 99.13% and an f1 score of 99.14%, while Policy 3 has an accuracy of 98.94%. Although the accuracy of this policy is 0.30% lower

than the baseline, thus not increasing the model's spatial invariance, it is still 0.19% higher than Policy 3. The model convergence of Policy 5, at 97 epochs, was slower than Policies 1, 3, and 4 but like Policy 2, implying that RandAugment introduced more sources of invariances than Policies 1, 3, and 4, thus taking the model longer to converge.

Figure 15

Loss and Validation Accuracy of All Policies for MNIST on NoelNet

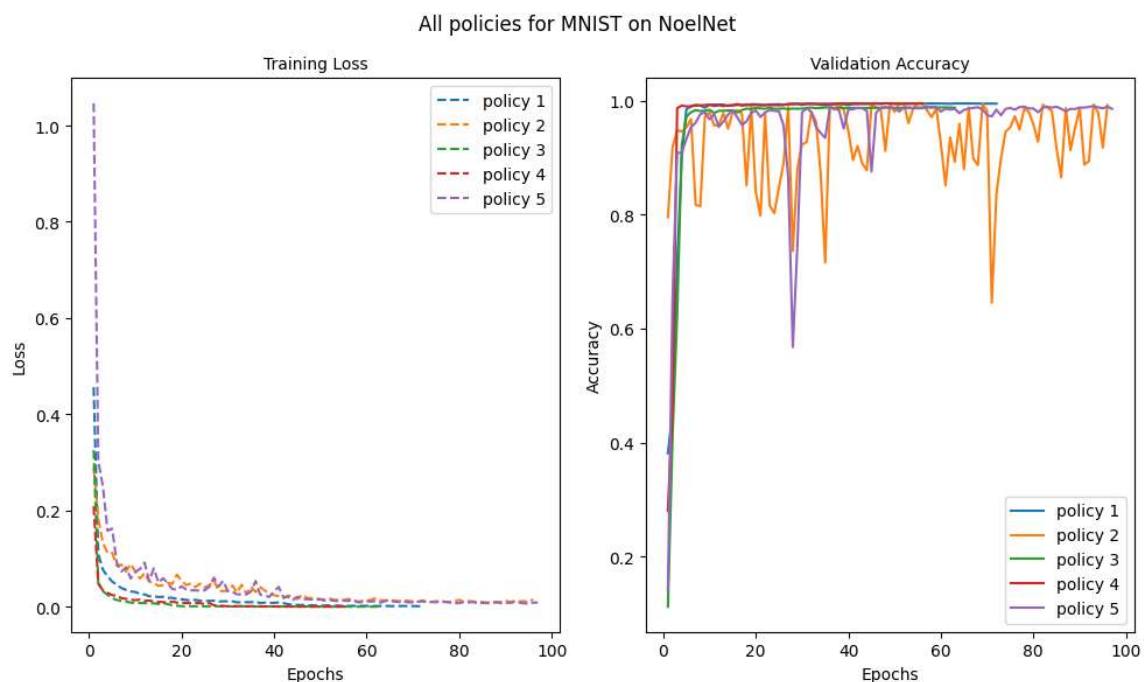
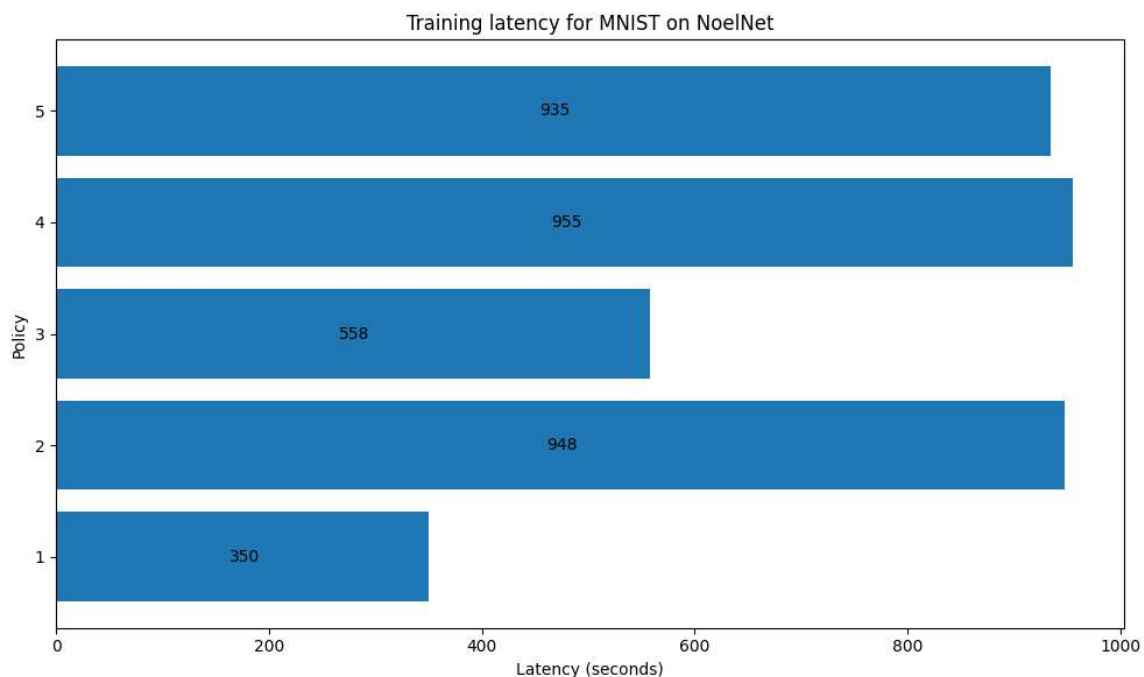


Figure 15 summarizes the training loss and validation accuracy of all policies for MNIST on NoelNet. Policy 1 established the baseline metrics and converged after 72 epochs. Policies 2 and 5 had the longest convergence at 96 and 97 epochs, respectively, which can be attributed to the stochasticity of RandAugment used by those two policies. The stochastic nature of RandAugment introduces a wide variety of invariances that takes the models longer to learn (evidenced by the spikes in the validation curves) and, thus, longer to converge. Policy 2 had the highest accuracy, followed by Policy 4. Policy 3 had the fastest convergence but the lowest accuracy of all policies.

Figure 16

Training Latency of All Policies for MNIST on NoelNet



The training latencies of all policies for MNIST on NoelNet are seen in Figure 16. Policy 4 had the longest training time, taking 955 seconds, approximately three times as long as Policy 1 and twice as long as Policy 3. Policies 2 and 5 took almost the same amount of time which was expected since they both use RandAugment. Policy 1 had the shortest training time, followed by Policy 3, mainly because they are not augmented or combined with any other dataset.

To summarize the performance of MNIST on NoelNet, Policies 2 and 4 had the highest accuracies and were higher than Policy 1, implying that these two policies increased the spatial invariance of the model. However, both policies also had the highest training times. In contrast, Policy 3 had the lowest accuracy, followed by Policy 5, with

accuracies less than Policy 1, suggesting that Policies 3 and 5 did not improve the spatial invariance of the model.

Fashion MNIST on NoelNet

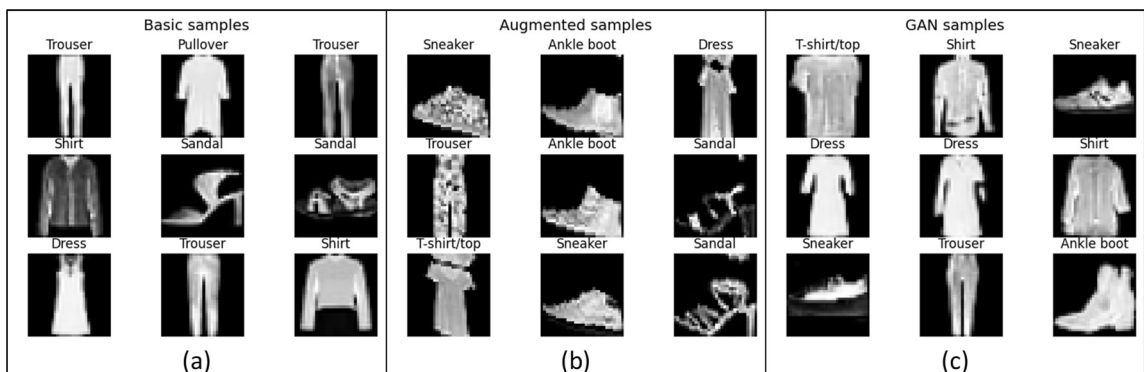
The results obtained after analyzing each policy for the FMNIST dataset on NoelNet are as follows.

Dataset Images

Figure 17 shows samples of three variations of the FMNIST dataset used in the experiments, either by themselves or in combinations thereof. Figure 17(a) shows the unperturbed dataset used by Policy 1 to establish baseline metrics. Figure 17(b) showed samples after being augmented by RandAugment and was used by Policy 2. Figure 17(c) shows the cDCGAN generated samples used by Policy 3. Finally, Policy 4 combined samples shown in Figure 17(a and c), and Policy 5 applied RandAugment to samples shown in Figure 17(c).

Figure 17

Sample Images of Unaugmented, Augmented, and GAN-Generated FMNIST



Note. Samples shown in (a) represent the benchmark or unperturbed dataset. (b) shows samples that were augmented using RandAugment. (c) shows samples generated using a conditional DCGAN.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 3

Performance Test Metrics of each Policy for FMNIST on NoelNet

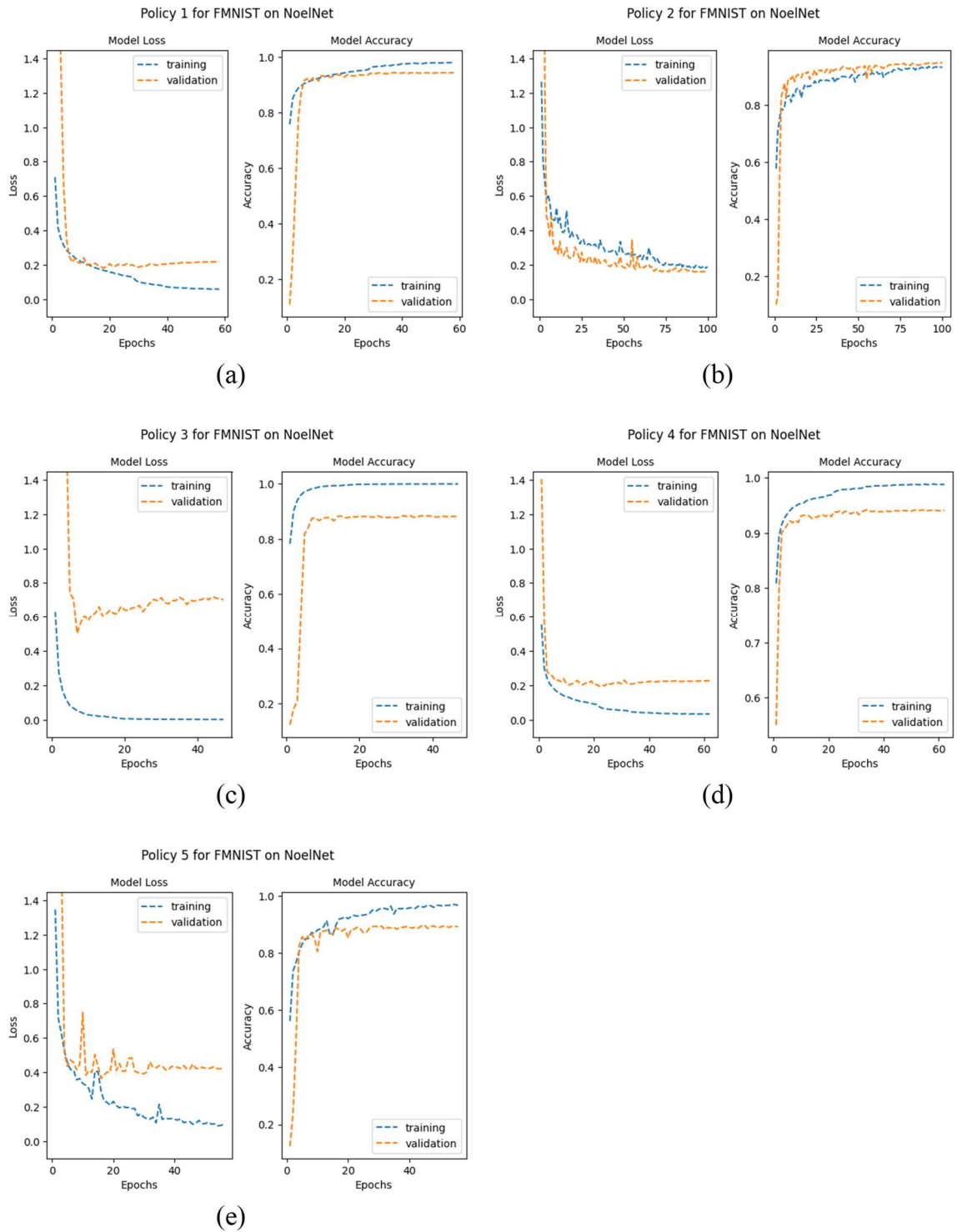
dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
fmnist	noelnet	1	0.9288	0.9292	0.9288	0.9287	0
fmnist	noelnet	2	0.9424	0.9422	0.9424	0.9421	1.46
fmnist	noelnet	3	0.8634	0.8635	0.8634	0.8617	-7.04
fmnist	noelnet	4	0.9307	0.9303	0.9307	0.9304	0.2
fmnist	noelnet	5	0.8724	0.8719	0.8724	0.871	-6.07

Note. The table displays the performance metrics for the test samples, where the delta column shows the percentage change in accuracy for each policy relative to Policy 1. Higher percentages imply improved performance and generalizability.

Table 3 summarizes the performance of each policy for FMNIST on NoelNet, where Policy 1 established a baseline. Policy 2 and Policy 4 had accuracies of 94.24% and 93.07%, respectively, which translated to improvements of 1.46% and 0.2%, respectively, relative to the baseline. In contrast, policies 3 and 5 had accuracies less than the baseline, 86.34% and 87.24%, respectively, which translated to decreases of 7.04% and 6.07%, respectively. The veracity of the accuracy for each policy is further confirmed by the f1 score being, on average, within 0.087% of each policy's accuracy. From Table 3, it can be surmised that Policies 2 and 4 increased the spatial invariance of the model, while Policies 3 and 5 reduced the spatial invariance of the model.

Figure 18

Accuracy and Loss Graphs of each Policy for FMNIST on NoelNet



Policy 1

Figure 18(a) shows the loss and accuracy curves for Policy 1. The chart shows that the training and validation curves start diverging around epoch 20, such that the gap between the graphs quickly becomes more prominent, suggesting a quick increase in overfitting is occurring. The model reached convergence at epoch 58, when the validation loss reached its minimum and training was terminated. This convergence was faster than that for Policy 1 on MNIST, which suggests that it is easier to train FMNIST on NoelNet. From Table 3, Policy 1 achieved an accuracy of 92.88% and an f1 score of 92.87%. The f1 score is approximately the same as the accuracy, which suggests that the class distribution is balanced, and the results produced by the model correctly reflect its performance, i.e., predictions made by the model should be at least 92% accurate.

Policy 2

Figure 18(b) shows the loss and accuracy graphs for Policy 2. The accuracy and loss graphs show a much closer correlation in relative trends than Policy 1. The gap between each curve is also smaller than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is much less than in Policy 1. The model had a slower convergence rate ($\sim 1.7x$ slower) compared to Policy 1 in that training terminated at Epoch 100 instead of Epoch 58 as it was for Policy 1. From Table 3, Policy 2 achieved an accuracy of 94.24% and an f1 score of 94.21%. This accuracy represents a 1.46% improvement over Policy 1 and the unperturbed dataset, a noteworthy improvement in the spatial invariance of the model by using RandAugment.

Policy 3

Figure 18(c) shows the loss and accuracy graphs for Policy 3. The charts can be seen to have much wider gaps as compared to policies 1 and 2, suggesting the synthetic samples caused much higher levels of overfitting very early in the training process. The model converged at epoch 47, which was faster than that of policies 1 and 2. From Table 3, Policy 3 achieved an accuracy of 86.34% and an f1 score of 86.17%, representing a 7.04% decrease from Policy 1. Also, the slightly smaller f1 score indicates slight imbalances in the class distributions. The reduction in accuracy relative to Policy 1 implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

Figure 18(d) shows the loss and accuracy graphs for Policy 4. The gaps in the graphs are slightly more significant than those of Policies 1 and 2 but smaller than Policy 3, indicating that Policy 4 may lead to more overfitting than Policies 1 and 2 but less overfitting than Policy 3. The model converged at epoch 62, which was faster than that of Policy 2 but slower than Policies 1 and 3. From Table 3, Policy 4 achieved an accuracy of 93.07% and an f1 score of 93.04%, representing a 0.2% improvement over Policy 1. In a similar fashion to Policy 4 for MNIST on NoelNet, it is observed that combining the synthesized samples from Policy 3 with their baseline counterpart led to improved spatial invariance in the trained model.

Policy 5

Figure 18(e) shows the loss and accuracy graphs for Policy 5. The gap between the loss curves is smaller than Policy 3 (has less overfitting) but broader than all the other

policies. In comparison, the gap between the accuracy curves is wider than policies 1 and 2 but not policies 3 and 4. This suggests that applying RandAugment to synthesized samples can improve spatial invariance compared to using the synthesized samples in isolation. This observation is corroborated by Table 3, where Policy 5 has an accuracy of 87.24% and an f1 score of 87.10%, while Policy 3 has an accuracy of 86.34%. Although the accuracy of this policy is 6.07% lower than the baseline, thus not increasing the model's spatial invariance, it is still 0.97% higher than Policy 3. The model convergence of Policy 5, at 56 epochs, was slower than Policy 3 but faster than Policies 1, 2, and 4.

Figure 19

Loss and Validation Accuracy of All Policies for FMNIST on NoelNet

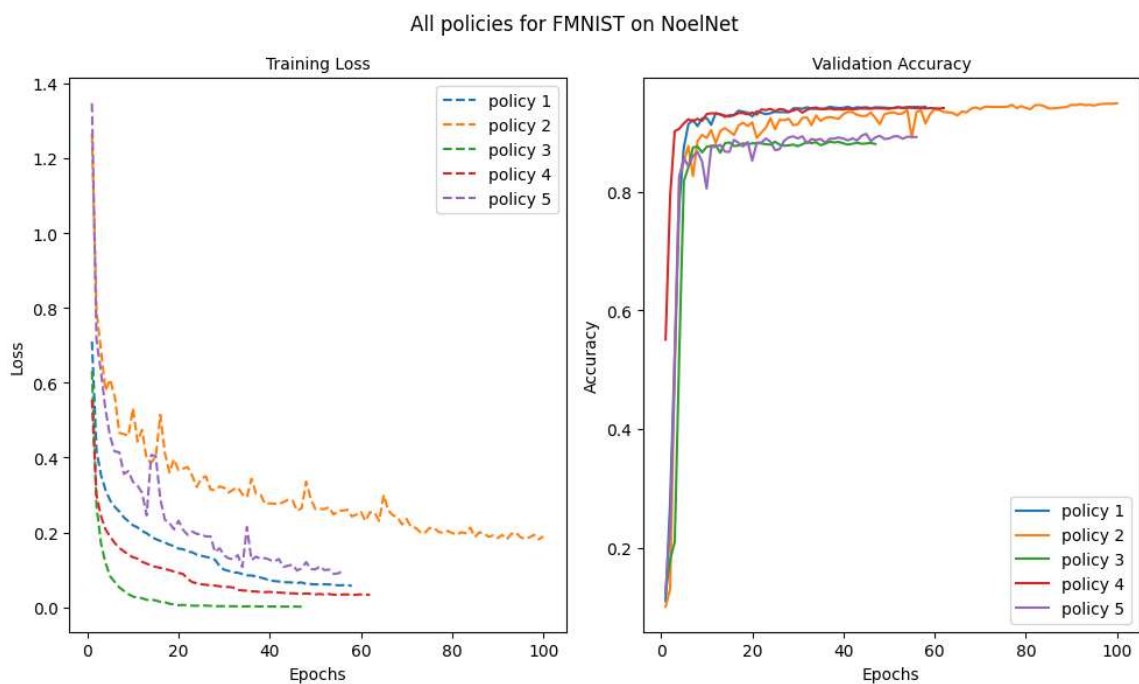
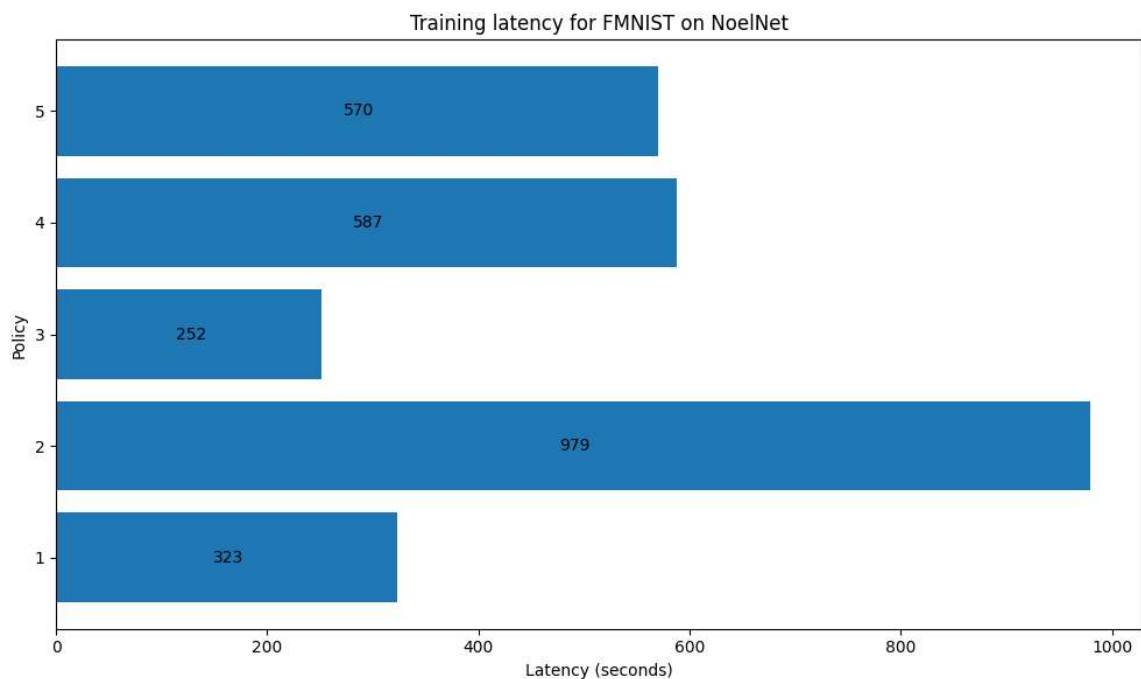


Figure 19 summarizes the training loss and validation accuracy of all policies for FMNIST on NoelNet. Policy 1 established the baseline metrics and converged after 58 epochs. Policy 2 had the longest convergence at 100 epochs, which can be attributed to the stochasticity of RandAugment, used by that policy. The stochastic nature of

RandAugment introduces a wide variety of invariances that takes the models longer to learn and, thus, longer to converge. Policy 2 also had the highest accuracy, followed by Policy 4. Policy 3 had the fastest convergence but the lowest accuracy of all policies.

Figure 20

Training Latency of All Policies for FMNIST on NoelNet



The training latencies of all policies for FMNIST on NoelNet are seen in Figure 20. Policy 2 had the longest training time, taking 979 seconds, approximately three times as long as Policy 1 and four times as long as Policy 3. Policies 4 and 5 took almost the same amount of time which was unexpected since Policy 5 used RandAugment and was expected to have similar latencies to Policy 2. Policy 3 had the shortest training time, followed by Policy 1, mainly because they are not augmented or combined with any other dataset.

To summarize the performance of FMNIST on NoelNet, despite having the longest training latency, Policy 2 had the highest accuracy, followed by Policy 4. They were both higher than Policy 1, implying that they increased the spatial invariance of the model. In contrast, Policy 3 had the lowest accuracy albeit the fastest training time, followed by Policy 5, with accuracies less than Policy 1. This suggests that Policies 3 and 5 did not improve the spatial invariance of the model. Furthermore, the precision and recall for each policy closely match the trend in their accuracies, implying that the accuracy reflects the model's "true" performance and that the classes are evenly distributed.

CIFAR-10 on NoelNet

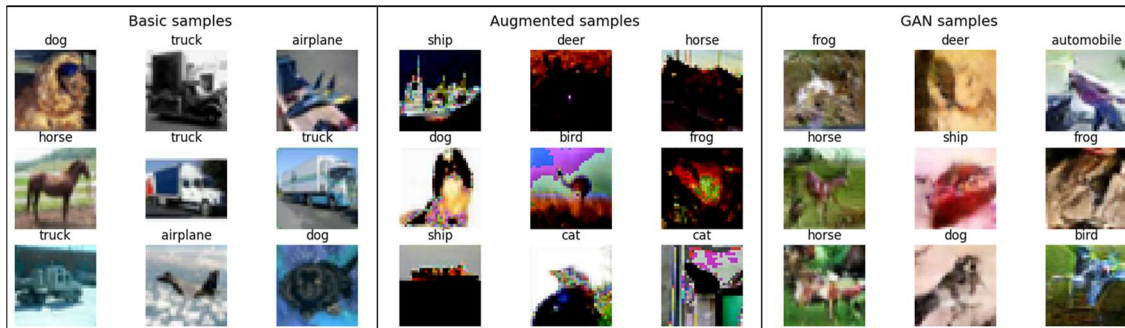
The results obtained after analyzing each policy for the CIFAR-10 dataset on NoelNet are as follows.

Dataset Images

Figure 21 shows samples of three variations of the CIFAR-10 dataset used in the experiments, either by themselves or in combinations thereof. Figure 21(a) shows the unperturbed dataset used by Policy 1 to establish baseline metrics. Figure 21(b) shows samples after being augmented by RandAugment and was used by Policy 2. Figure 21(c) shows the cDCGAN generated samples used by Policy 3. Finally, Policy 4 combined samples shown in Figure 21(a and c), and Policy 5 applied RandAugment to samples shown in Figure 21(c).

Figure 21

Sample Images of Unaugmented, Augmented, and GAN-Generated CIFAR-10



Note. Samples shown in (a) represent the benchmark or unperturbed dataset. (b) shows samples that were augmented using RandAugment. (c) shows samples generated using a conditional DCGAN.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 4

Performance Test Metrics of each Policy for CIFAR-10 on NoelNet

dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
cifar10	noelnet	1	0.8516	0.8517	0.8516	0.8509	0
cifar10	noelnet	2	0.8895	0.8913	0.8895	0.8901	4.45
cifar10	noelnet	3	0.5783	0.5992	0.5783	0.5814	-32.09
cifar10	noelnet	4	0.8364	0.837	0.8364	0.8364	-1.78
cifar10	noelnet	5	0.6859	0.6963	0.6859	0.6892	-19.46

Note. The table displays the performance metrics for the test samples, where the delta column shows the percentage change in accuracy for each policy relative to Policy 1. Higher percentages imply improved performance and generalizability.

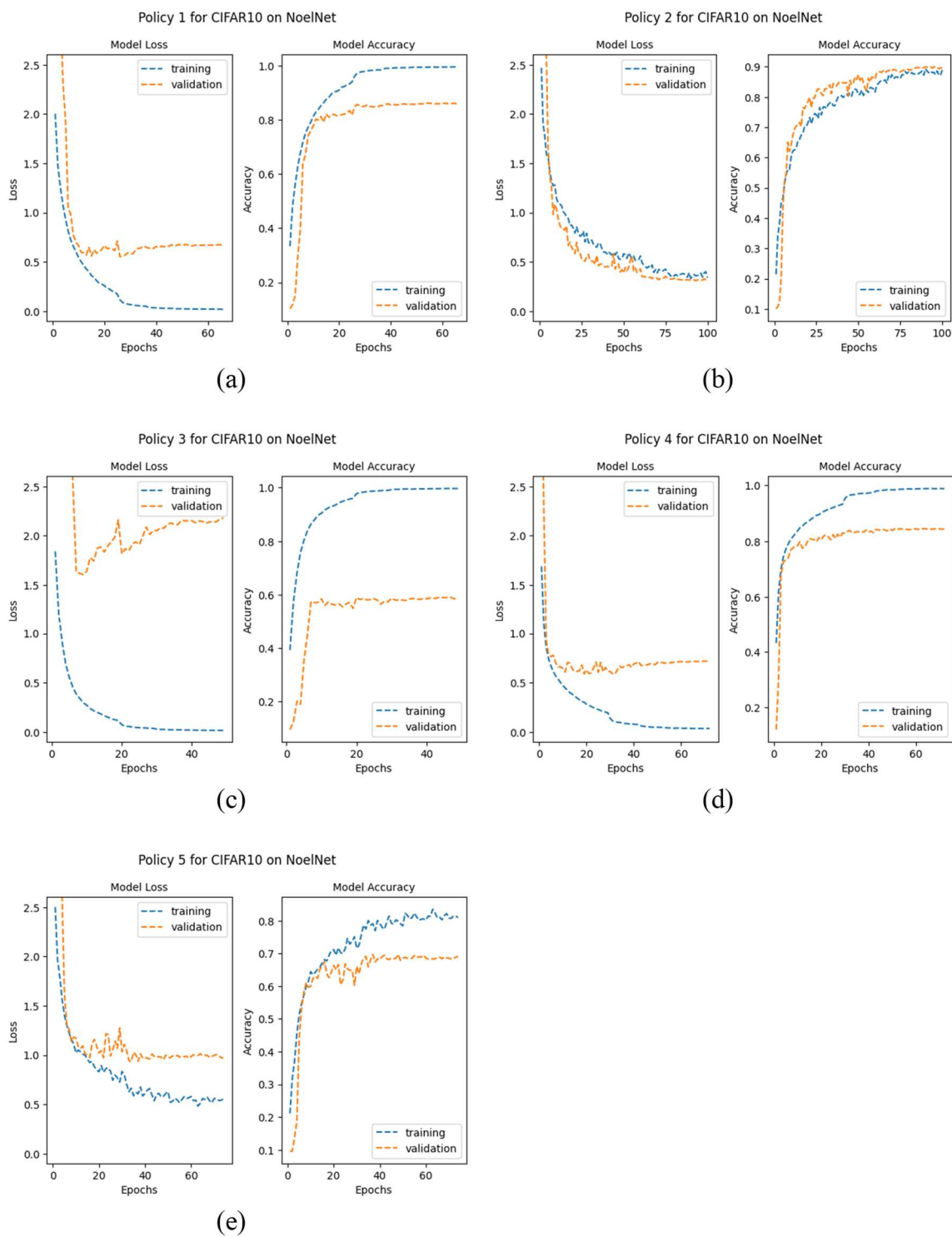
Table 4 summarizes the performance of each policy for CIFAR-10 on NoelNet, where Policy 1 established a baseline. Policies 2 and 4 had accuracies of 88.95% and

83.64%, respectively, which translated to an improvement of 4.45% using Policy 2 and a decrease of 1.78% using Policy 4. The latter differs from Policy 4 on F/MNIST for NoelNet, where the performance of Policy 4 was consistently higher than the baseline. Conversely, policies 3 and 5 had accuracies less than the baseline, 57.83% and 68.59%, respectively, which translated to decreases of 32.09% and 19.46%, respectively.

The major decrease in accuracy observed by Policy 3 could be attributed to the poor quality of the GAN samples or the simplicity of NoelNet architecture in learning the invariances in the GAN-generated samples for CIFAR-10 or a combination of both. Since Policy 5 directly used the GAN-generated samples, its performance was similarly degraded, albeit better than Policy 3. From Table 4, it can be surmised that Policy 2 increased the spatial invariance of the model, while Policies 3, 4, and 5 reduced its spatial invariance.

Figure 22

Accuracy and Loss Graphs of each Policy for CIFAR-10 on NoelNet



Policy 1

Figure 22(a) shows the loss and accuracy curves for Policy 1. The chart shows that the training and validation curves start diverging around epoch 10, such that the gap between the graphs becomes more prominent, suggesting an increase in overfitting is occurring. The model reached convergence at epoch 66, when the validation loss reached its minimum and training was terminated. This convergence was faster than that for Policy 1 on MNIST but slower than Policy 1 on FMNIST. From Table 4, Policy 1 achieved an accuracy of 85.16% and an f1 score of 85.09%. The f1 score was competitive (i.e., within 0.08%) with accuracy, which suggested that the class distribution is balanced and that results produced by the model correctly reflected its performance, i.e., predictions made by the model should be at least 85% accurate.

Policy 2

Figure 22(b) shows the loss and accuracy graphs for Policy 2. The accuracy and loss graphs show a much closer correlation in their relative trends than Policy 1, a correlation that was maintained throughout the training process. This implies the model was learning the increased invariances in the augmented data and thus was generalizing better on the unseen (i.e., the validation) data. The gap between each curve is also smaller than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is less than in Policy 1. The model had a slower convergence rate (~1.5x slower) compared to Policy 1 in that training terminated at epoch 100 instead of epoch 66 as it was for Policy 1. From Table 4, Policy 2 achieved an accuracy of 88.95% and an f1 score of 89.01%. This accuracy represents a 4.45% improvement over Policy 1

and the unperturbed dataset, a noteworthy improvement in the spatial invariance of the model by using RandAugment.

Policy 3

Figure 22(c) shows the loss and accuracy graphs for Policy 3. The graphs can be seen to have significantly wider gaps as compared to policies 1 and 2 and started diverging around epoch 10, suggesting that the synthetic samples caused high levels of overfitting in NoelNet. The model converged at epoch 49, much quicker than Policies 1 and 2, suggesting that the synthesized samples had less diversity. From Table 4, Policy 3 achieved an accuracy of 57.83% and an f1 score of 58.14%, representing a 32.09% decrease from Policy 1. Also, the slightly higher f1 score indicates imbalances in the class distributions. The reduction in accuracy relative to Policy 1 implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

Figure 22(d) shows the loss and accuracy graphs for Policy 4. The gaps in the loss graphs are slightly wider than Policy 1, wider than Policy 2, and much smaller than Policy 3, indicating that Policy 4 may lead to more overfitting than Policy 1 and 2 but much less overfitting than Policy 3. The model converged at epoch 72, faster than Policy 2 but slower than Policies 1 and 3. From Table 4, Policy 4 achieved an accuracy and an f1 score of 83.64%, representing a 1.78% decrease from Policy 1. The latter was dissimilar to Policy 4 for F/MNIST on NoelNet, where it was observed that combining the synthesized samples from Policy 3 with their baseline counterpart led to improved spatial invariance in the trained model.

Policy 5

Figure 22(e) shows the loss and accuracy graphs for Policy 5. The gap between the curves is smaller than policies 1, 3, and 4, which suggests Policy 5 has less overfitting compared to these policies. From Table 4, Policy 5 has an accuracy of 68.59% and an f1 score of 68.92%, while Policy 3 has an accuracy of 57.83%. Although the accuracy of this Policy is 19.46% lower than the baseline, thus not increasing the model's spatial invariance, it is 12.63% higher than Policy 3. The model convergence of Policy 5, at 74 epochs, was slower than Policies 1, 3, and 4 but, like Policy 2, implies that RandAugment introduced more sources of invariances than Policies 1, 3, and 4, thus taking the model longer to converge.

Figure 23

Loss and Validation Accuracy of All Policies for CIFAR-10 on NoelNet

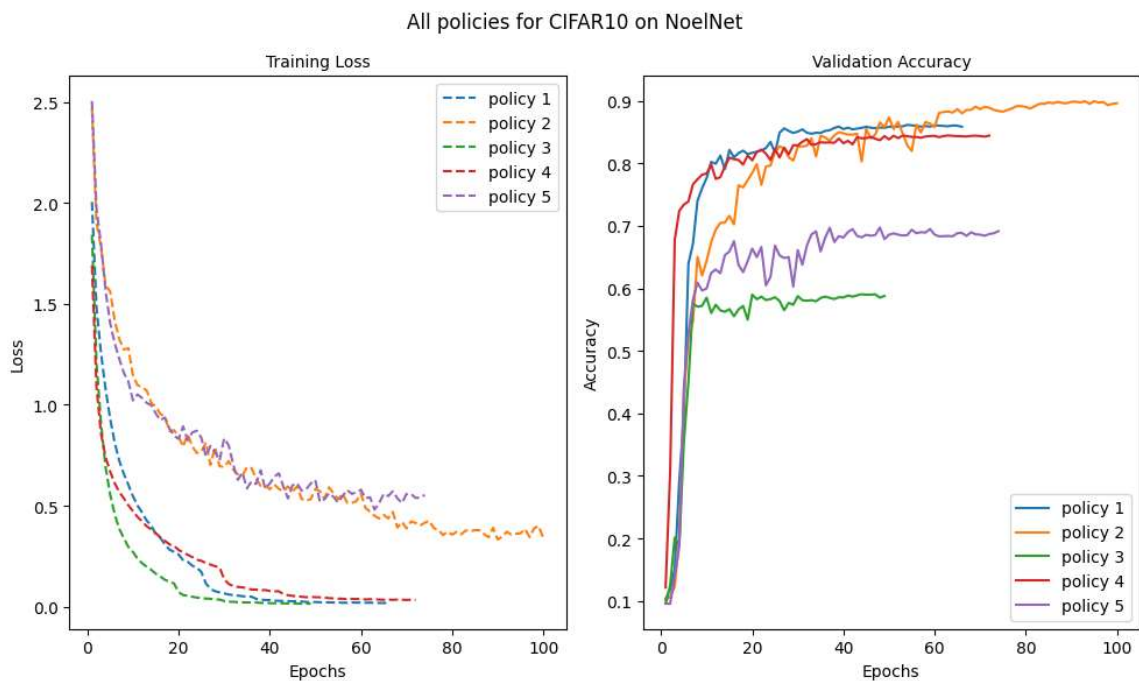
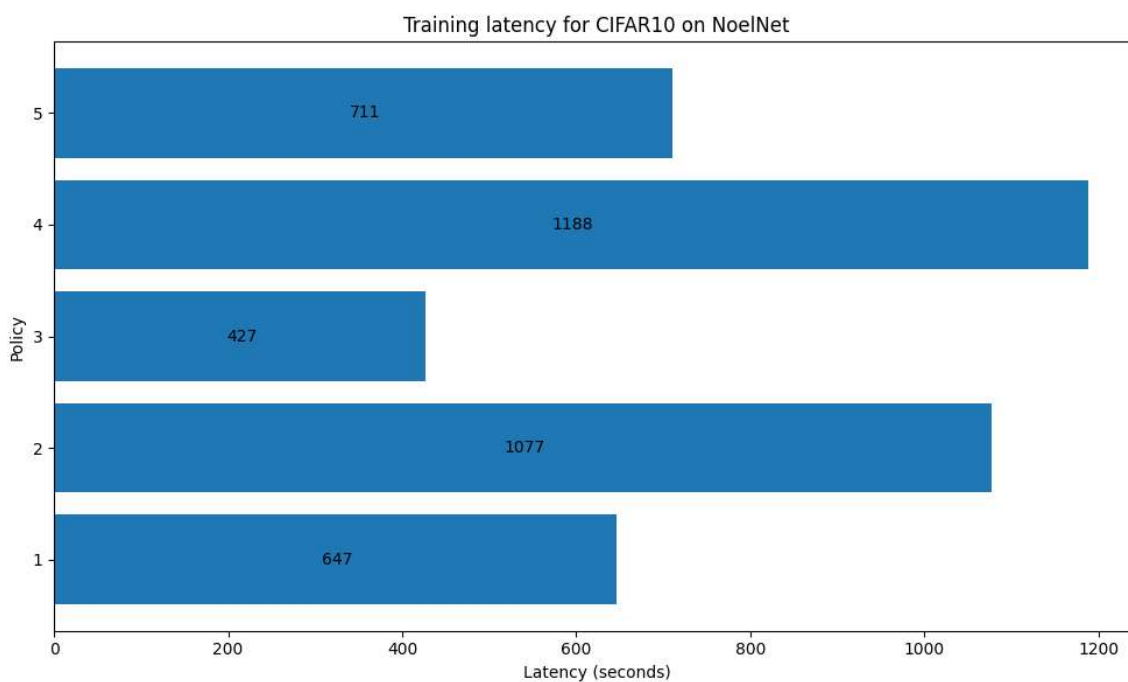


Figure 23 summarizes the training loss and validation accuracy of all policies for CIFAR-10 on NoelNet. Policy 1 established the baseline metrics and converged after 66

epochs. Policy 2 had the longest convergence at 100 epochs, while Policy 3 had the shortest at 49. Policy 2 had the highest accuracy, while Policy 3 had the fastest convergence but the lowest accuracy. The loss curves for Policies 2 and 5, the two policies implementing RandAugment, fit the profile of the smooth polynomial curves seen in models with minimal overfitting like Figures 7 and 8. This suggests that RandAugment is a very effective method of reducing overfitting in ConvNets.

Figure 24

Training Latency of All Policies for CIFAR-10 on NoelNet



The training latencies of all policies for CIFAR-10 on NoelNet are seen in Figure 24. Policy 4 had the longest training time of 1188 seconds, approximately 1.8x as long as Policy 1, more than 2.7x as long as Policy 3, and about 1.6x that of Policy 5. The large difference between the latencies of Policies 2 and 5 was unexpected since they used RandAugment and expected to have similar training times. The last observation may be

because the synthesized samples used in Policy 5 are poorer in quality and complexity than the samples used by Policy 2, and augmenting them with RandAugment did not increase their invariance to the levels seen in Policy 2. A similar observation was made in Policy 3, which had the shortest training time and smallest accuracy.

To summarize the performance of CIFAR-10 on NoelNet, Policy 2 had the highest accuracy despite having the longest training time; a similar pattern was observed for F/MNIST on NoelNet. In contrast, Policy 3 had the fastest training time but the lowest performance metrics, as was also seen by Policy 3 on F/MNIST for NoelNet. The lower metrics achieved by Policy 3 may be due to the simpler architecture of NoelNet not being able to learn the invariances from the GAN-generated CIFAR-10 samples sufficiently. Since Policies 4 and 5 also used the GAN-generated samples, they also suffered similar degradations in performance. This suggests that Policies 3, 4, and 5 did not improve the spatial invariance of the model.

MNIST on ResNet-50

The results obtained after analyzing each policy for the MNIST dataset on the ResNet50 model are as follows.

Dataset Images

Refer to Figure 13 for a description of and samples of three variations of the MNIST dataset used in the experiments, either by themselves or in combinations thereof.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 5

Performance Test Metrics of each Policy for MNIST on ResNet-50

dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
mnist	resnet	1	0.9914	0.9916	0.9913	0.9914	0
mnist	resnet	2	0.9937	0.9938	0.9936	0.9937	0.23
mnist	resnet	3	0.9874	0.9873	0.9874	0.9873	-0.40
mnist	resnet	4	0.9930	0.9931	0.9929	0.9930	0.16
mnist	resnet	5	0.9894	0.9895	0.9893	0.9893	-0.20

Note. The table displays the performance metrics for the test samples, where the delta column shows the percentage change in accuracy for each policy relative to Policy 1. Higher percentages imply improved performance and generalizability.

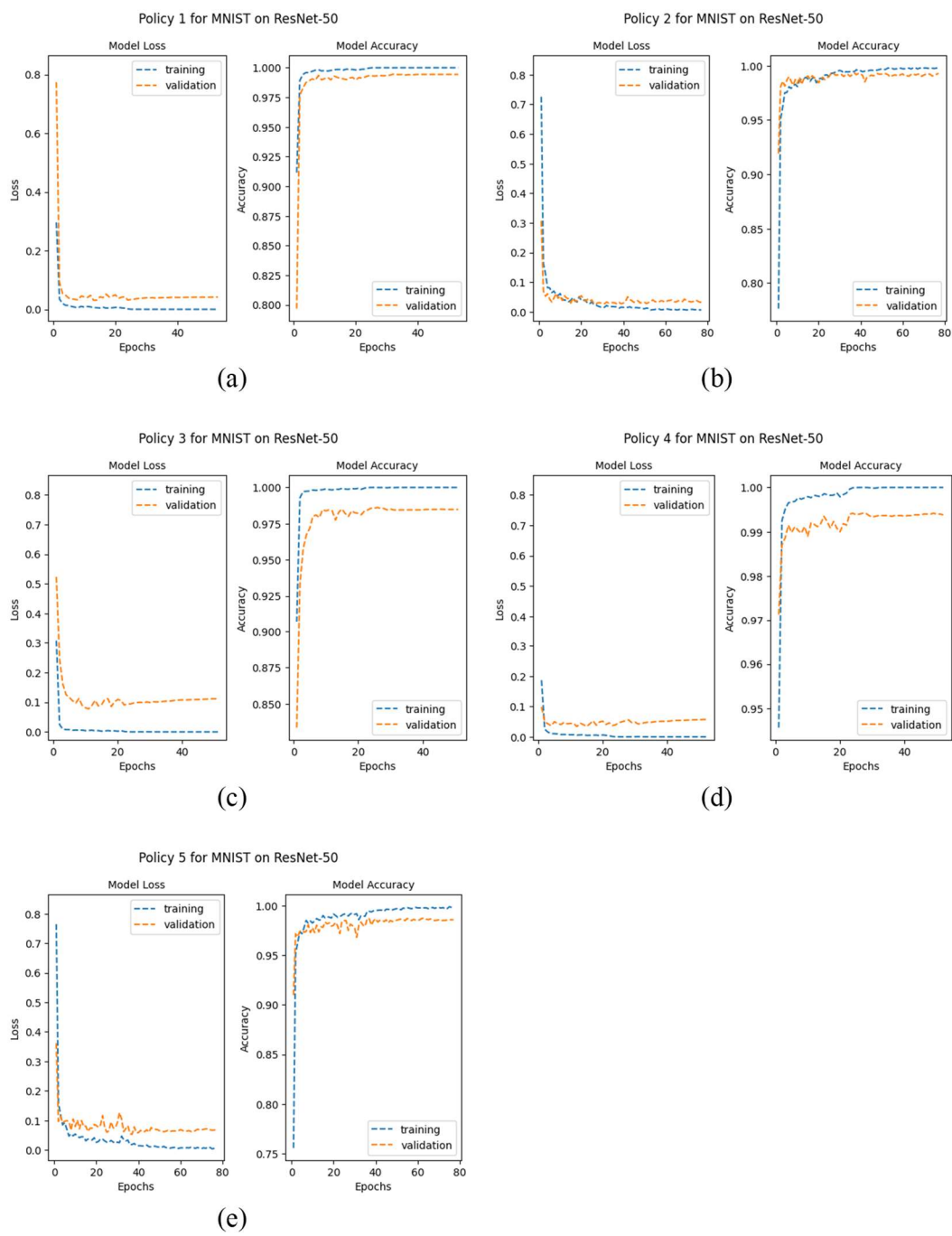
Table 5 summarizes the performance of each policy for MNIST on ResNet-50, where Policy 1 established a baseline. Policy 2 and Policy 4 had accuracies of 99.37% and 99.30%, respectively, which translated to improvements of 0.23% and 0.16%, respectively, relative to the baseline. On the other hand, both policies 3 and 5 had accuracies less than the baseline, 98.74% and 98.94%, respectively, which translated to decreases of 0.4% and 0.2%, respectively. The veracity of the accuracy for each policy is further confirmed by the f1 score being, on average, within 0.004% of each policy's accuracy. From Table 5, it can be surmised that Policies 2 and 4 increased the spatial invariance of the model, while Policies 3 and 5 reduced the spatial invariance of the model.

Similar patterns were observed in Table 2 for MNIST on NoelNet in that Policy 2 achieved the highest accuracy and f1 scores of 99.61%, followed by Policy 4 with both scores at 99.44%. Policy 1 (the unaugmented data policy) achieved scores of 99.43%, while Policy 5 achieved 99.13%. Finally, Policy 3 had the lowest accuracy and f1 scores

at 98.94%. Notably, each Policy on NoelNet for MNIST resulted in higher accuracy and f1 scores than on ResNet-50.

Figure 25

Accuracy and Loss Graphs of each Policy for MNIST on ResNet-50



Policy 1

The purpose of Policy 1 was to establish a baseline for the metrics by training the model on the unperturbed dataset. This baseline was used as the control over which all other policies and metrics were compared to determine if there were any improvements in spatial invariance. Figure 25(a) shows the loss and accuracy curves for Policy 1. The chart shows that the validation graphs follow the training graphs very closely and are coordinated, so the gap between the graphs is minimal, suggesting negligible overfitting. The model reached convergence at epoch 53, when the validation loss reached its minimum and training was terminated. From Table 5, Policy 1 achieved an accuracy of 99.14% and an f1 score of 99.14%. The f1 score is the same as the accuracy, which suggests that the class distribution is balanced and corroborates the model's accuracy as a "true" accuracy, i.e., the results produced by the model correctly reflect the performance of the model and the predictions made by the model should be at least 99% accurate.

Policy 2

The purpose of Policy 2 was to analyze the effect of augmenting the dataset using RandAugment to improve spatial invariance within the ConvNet. Figure 25(b) shows the loss and accuracy graphs for Policy 2. The accuracy and loss graphs show a very tight correlation in their relative patterns. The gap between each curve is smaller or tighter than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is smaller than in Policy 1. The model also had a slower convergence rate (~1.5x slower) compared to Policy 1 in that training terminated at Epoch 77 instead of Epoch 53 as it was for Policy 1. From Table 5, Policy 2 achieved an accuracy of 99.37% and an f1 score of 99.37%. This accuracy represents a 0.23% improvement over Policy 1 and the

unperturbed dataset. This improvement in accuracy may be small but significant in that off-the-shelf ConvNets quickly learn the MNIST dataset. These ConvNets, when trained on MNIST, can achieve over 99% accuracy consistently and without optimizations. The ResNet50 model used in this policy is highly optimized; thus, techniques leading to further improved accuracy in its performance on any dataset are noteworthy.

Policy 3

Policy 3 was used to synthesize samples of the dataset using a conditional DCGAN and use these samples to train the model to improve the model's spatial invariance. Theoretically, the cDCGAN samples should possess increased invariances, which can reduce the level of overfitting in a model and increase its generalizability. Figure 25(c) shows the loss and accuracy graphs for Policy 3. The graphs can be seen to have more significant gaps as compared to policies 1 and 2, suggesting the synthetic samples cause higher levels of overfitting. The model converged at epoch 51, which was faster than that of policies 1 and 2. From Table 5, Policy 3 achieved an accuracy of 98.74% and an f1 score of 98.73%, representing a 0.40% decrease from Policy 1. The latter implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

The purpose of Policy 4 was to demonstrate that by combining the cDCGAN synthesized samples with the unaugmented samples, an improvement in spatial invariance can be achieved. Figure 25(d) shows the loss and accuracy graphs for Policy 4. The gaps in the graphs are slightly more significant than those of Policies 1 and 2 but smaller than Policy 3, indicating that Policy 4 may lead to more overfitting than Policies

1 and 2 but less overfitting than Policy 3. The model converged at epoch 52, which was faster than that of Policies 1 and 2 but slower than Policy 3. From Table 5, Policy 4 achieved both accuracy and an f1 score of 99.30%, representing a 0.16% improvement in accuracy over Policy 1. This improvement over the baseline model suggests that Policy 4 improves spatial invariance. This improvement directly results from the additional invariances provided by the GAN samples when combined with the baseline dataset. When used by themselves and not combined with any other dataset, the synthesized samples from Policy 3 led to a decrease in spatial invariance. However, when combined with their baseline counterpart, as seen in Policy 4, it increases the spatial invariance of the trained model.

Policy 5

The purpose of Policy 5 was to demonstrate that stacking or combining different augmentation techniques can lead to improvements in spatial invariance in ConvNets. This policy applied RandAugment to the cDCGAN synthesized samples, and the resulting loss and accuracy graphs are shown in Figure 25(e). The gap between each graph is smaller (has less overfitting) than those of policies 3 and 4 but more prominent than those of policies 1 and 2. This suggests that applying RandAugment to synthesized samples can improve spatial invariance compared to using the synthesized samples in isolation. This observation is corroborated by Table 5, where Policy 5 has an accuracy of 98.94% and an f1 score of 98.93%, while Policy 3 has an accuracy of 98.74%. Although the accuracy of this policy is 0.20% lower than the baseline, thus not increasing the model's spatial invariance, it is still 0.2% higher than Policy 3. The model convergence of Policy 5, at 77 epochs, was slower than Policies 1, 3, and 4 but like Policy 2, implying

that RandAugment introduced more sources of invariances than Policies 1, 3, and 4, thus taking the model longer to converge.

Figure 26

Training Loss and Validation Accuracy of All Policies for MNIST on ResNet-50

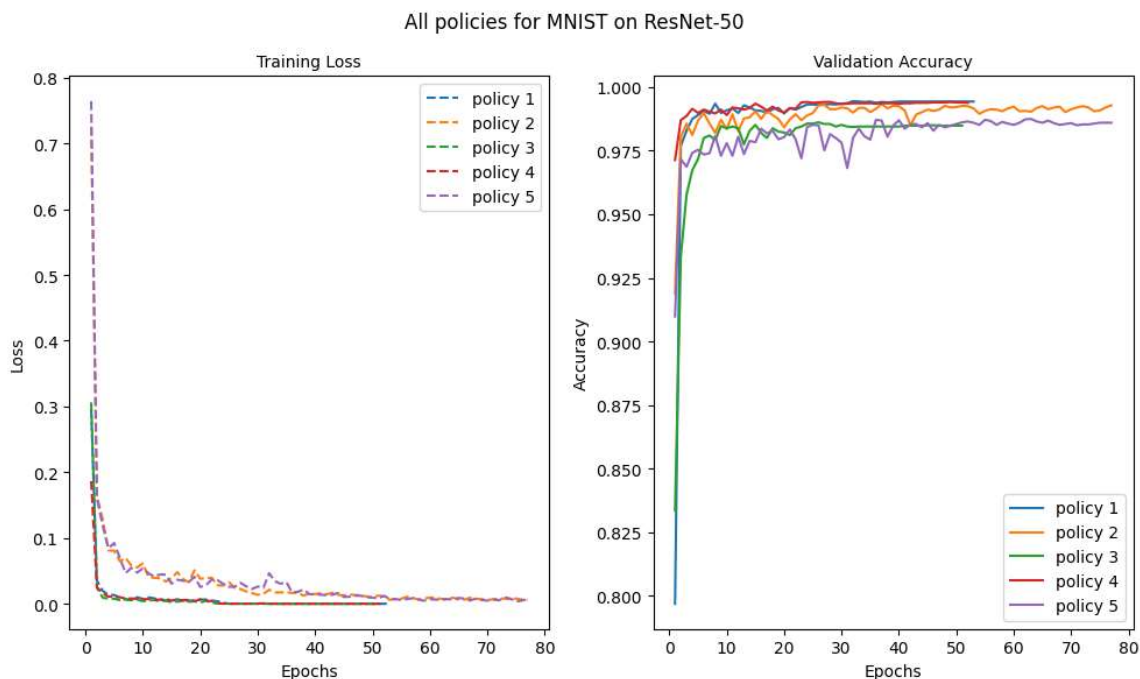
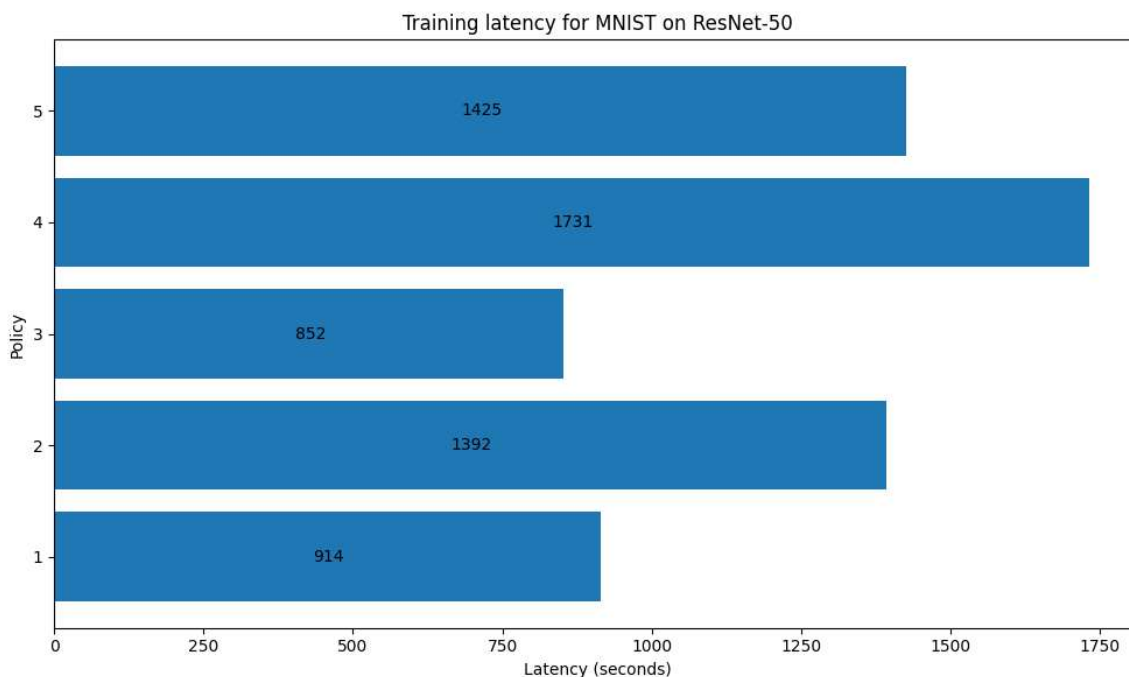


Figure 26 summarizes the training loss and validation accuracy of all policies for MNIST on ResNet-50. Policy 1 established the baseline metrics and had a relatively fast convergence. Policies 2 and 5 had the longest convergence at 77 epochs, which can be attributed to the stochasticity of RandAugment used by those two policies. The stochastic nature of RandAugment introduces a wide variety of invariances which takes the models longer to learn and, thus, longer to converge. Policy 2 also had the highest accuracy, followed by Policy 4. Policy 3 had the fastest convergence but the lowest accuracy of all policies.

Figure 27

Training Latency of All Policies for MNIST on ResNet-50



The training latencies of all policies for MNIST on ResNet-50 are seen in Figure 27. Policy 4 had the longest training time, taking 1731 seconds, approximately twice as long as policies 1 and 3. On the other hand, policies 2 and 5 took almost the same amount of time which was expected since they both use RandAugment. Policy 3 had the shortest training time, followed by Policy 1, mainly because they are not augmented or combined with any other dataset. In comparison, the training latency for MNIST was between 1.5 to 2 times faster on NoelNet than on ResNet-50.

In summary, for MNIST on ResNet-50, it was observed that Policies 2 and 4 had the highest accuracies and were higher than Policy 1, implying that these two policies increased the spatial invariance of the model. On the other hand, policy 3 has the lowest accuracy, followed by Policy 5, with accuracies less than Policy 1. The latter suggests

that policies 3 and 5 did not improve the spatial invariance of the model. Furthermore, Table 5 shows that the precision and recall for each policy closely match the trend in their accuracies, implying that the accuracy reflects the model's "true" performance, the classes are evenly distributed, and the model is not hallucinating. These same patterns were also observed in Table 2 for MNIST on NoelNet.

Fashion MNIST on ResNet-50

The results obtained after analyzing each policy for the FMNIST dataset on the ResNet50 model are as follows.

Dataset Images

Refer to Figure 17 for a description of and samples of three variations of the FMNIST dataset used in the experiments, either by themselves or in combinations.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 6

Performance Test Metrics of each Policy for FMNIST on ResNet-50

dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
fmnist	resnet	1	0.8985	0.9002	0.8985	0.8968	0
fmnist	resnet	2	0.9266	0.9279	0.9266	0.9269	3.13
fmnist	resnet	3	0.8267	0.8289	0.8267	0.8206	-7.99
fmnist	resnet	4	0.9159	0.917	0.9159	0.9162	1.94
fmnist	resnet	5	0.85	0.8582	0.85	0.8518	-5.4

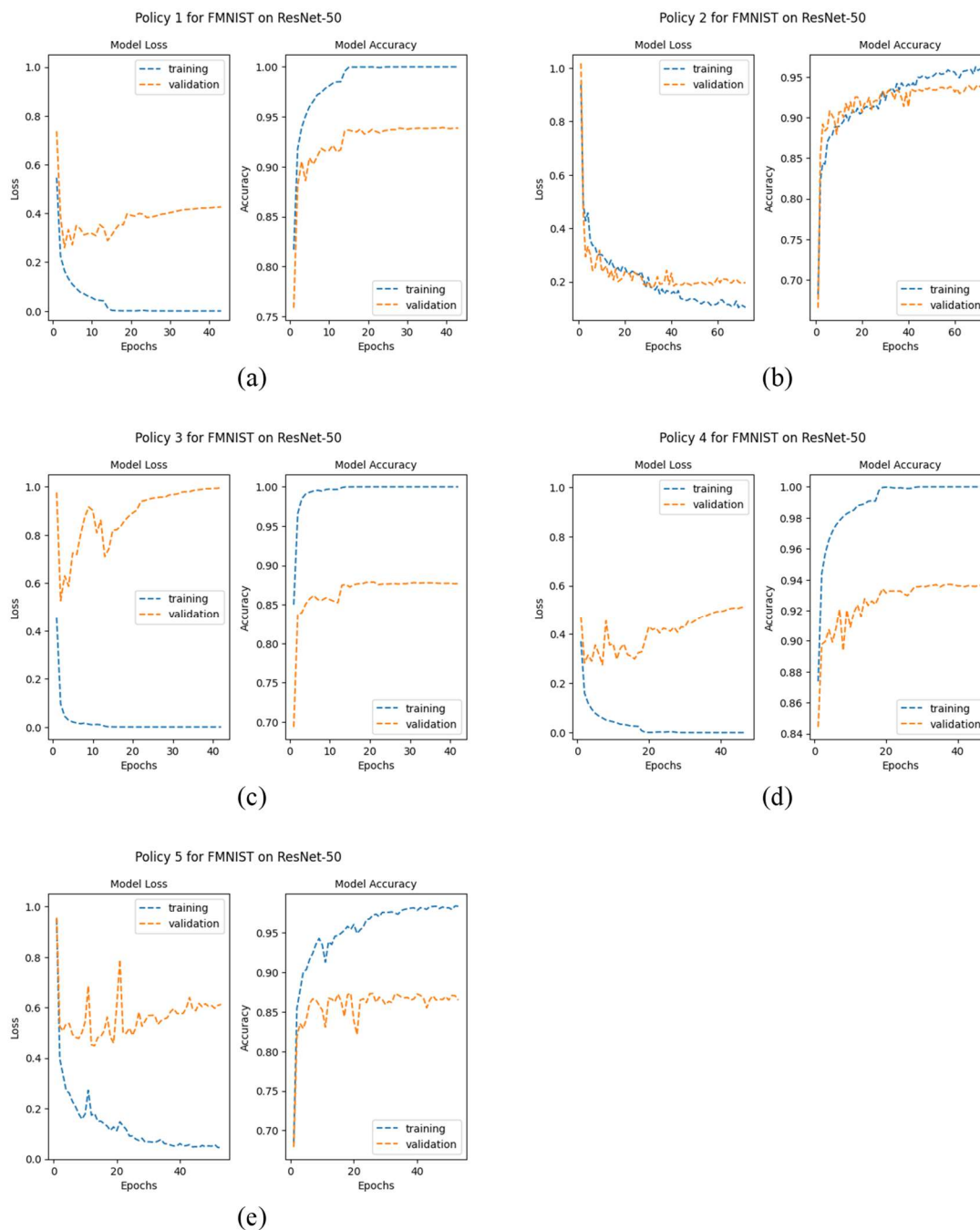
Note. The table shows the performance metrics for the test samples. The delta column shows the percentage change in accuracy for each policy relative to Policy 1. Higher percentages imply improved performance and generalizability.

Table 6 summarizes the performance of each policy for FMNIST on ResNet-50, where Policy 1 established a baseline. Policy 2 and Policy 4 had accuracies of 92.66% and 91.59%, respectively, which translated to improvements of 3.13% and 1.94%, respectively, relative to the baseline. On the other hand, both policies 3 and 5 had accuracies less than the baseline, 82.67% and 85%, respectively, which translated to decreases of 7.99% and 5.4%, respectively. The veracity of the accuracy for each policy is further confirmed by the f1 score being, on average, within 0.24% of each policy's accuracy. From Table 6, it can be surmised that Policies 2 and 4 increased the spatial invariance of the model, while Policies 3 and 5 reduced the spatial invariance of the model.

Similar patterns were observed in Table 3 for FMNIST on NoelNet in that Policy 2 achieved the highest accuracy and f1 scores of 94.24%, followed by Policy 4 with an accuracy of 93.07%. Policy 1 (the unaugmented data policy) achieved an accuracy of 92.88%, while Policy 5 achieved 87.24%. Finally, Policy 3 had the lowest accuracy and f1 scores at 86.34%. Notably, each Policy on NoelNet for FMNIST resulted in higher accuracy and f1 scores than on ResNet-50.

Figure 28

Accuracy and Loss Graphs of each Policy for FMNIST on ResNet-50



Policy 1

Figure 28(a) shows the loss and accuracy curves for Policy 1. The chart shows that the training and validation curves start diverging around epoch 5, such that the gap

between the graphs quickly becomes more prominent, suggesting a quick increase in overfitting is occurring. The model reached convergence at epoch 43, when the validation loss reached its minimum and training was terminated. This convergence was much faster than that for Policy 1 on MNIST, which suggests that it is easier to train FMNIST on ResNet50. From Table 6, Policy 1 achieved an accuracy of 89.85% and an f1 score of 89.68%. The f1 score is approximately the same as the accuracy, which suggests that the class distribution is balanced, and the results produced by the model correctly reflect its performance, i.e., predictions made by the model should be at least 89% accurate.

Policy 2

Figure 28(b) shows the loss and accuracy graphs for Policy 2. The accuracy and loss graphs show a much closer correlation in relative trends than Policy 1. The gap between each curve is also smaller than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is less than in Policy 1. The model had a slower convergence rate ($\sim 1.7x$ slower) compared to Policy 1 in that training terminated at Epoch 72 instead of Epoch 43 as it was for Policy 1. From Table 6, Policy 2 achieved an accuracy of 92.66% and an f1 score of 92.69%. This accuracy represents a 3.13% improvement over Policy 1 and the unperturbed dataset, a noteworthy improvement in the spatial invariance of the model by using RandAugment.

Policy 3

Figure 28(c) shows the loss and accuracy graphs for Policy 3. The graphs can be seen to have much wider gaps as compared to policies 1 and 2, suggesting the synthetic samples caused much higher levels of overfitting very early in the training process. The model converged at epoch 42, which was faster than that of policies 1 and 2. From Table

6, Policy 3 achieved an accuracy of 82.67% and an f1 score of 82.06%, representing an almost 8% decrease from Policy 1. Also, the slightly smaller f1 score indicates slight imbalances in the class distributions. The decrease in accuracy relative to Policy 1 implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

Figure 28(d) shows the loss and accuracy graphs for Policy 4. The gaps in the graphs are slightly more significant than those of Policies 1 and 2 but smaller than Policy 3, indicating that Policy 4 may lead to more overfitting than Policies 1 and 2 but less overfitting than Policy 3. The model converged at epoch 47, which was faster than that of Policy 2 but slower than Policies 1 and 3. From Table 6, Policy 4 achieved an accuracy of 91.59% and an f1 score of 91.62%, representing an almost 2% improvement over Policy 1. In a similar fashion to Policy 4 for MNIST on ResNet50, it is observed that combining the synthesized samples from Policy 3 with their baseline counterpart led to improved spatial invariance in the trained model.

Policy 5

Figure 28(e) shows the loss and accuracy graphs for Policy 5. The gap between the loss curves is smaller than Policy 3 (has less overfitting) but broader than all the other policies. In comparison, the gap between the accuracy curves is wider than policies 1 and 2 but not policies 3 and 4. This suggests that applying RandAugment to synthesized samples can improve spatial invariance compared to using the synthesized samples in isolation. This observation is corroborated by Table 6, where Policy 5 has an accuracy of 85% and an f1 score of 85.18%, while Policy 3 has an accuracy of 82.67%. Although the

accuracy of this policy is 5.4% lower than the baseline, thus not increasing the model's spatial invariance, it is still 2.6% higher than Policy 3. The model convergence of Policy 5, at 53 epochs, was slower than Policies 1, 3, and 4 but like Policy 2, implying that RandAugment introduced more sources of invariances than Policies 1, 3, and 4, thus taking the model longer to converge.

Figure 29

Loss and Validation Accuracy of All Policies for FMNIST on ResNet-50

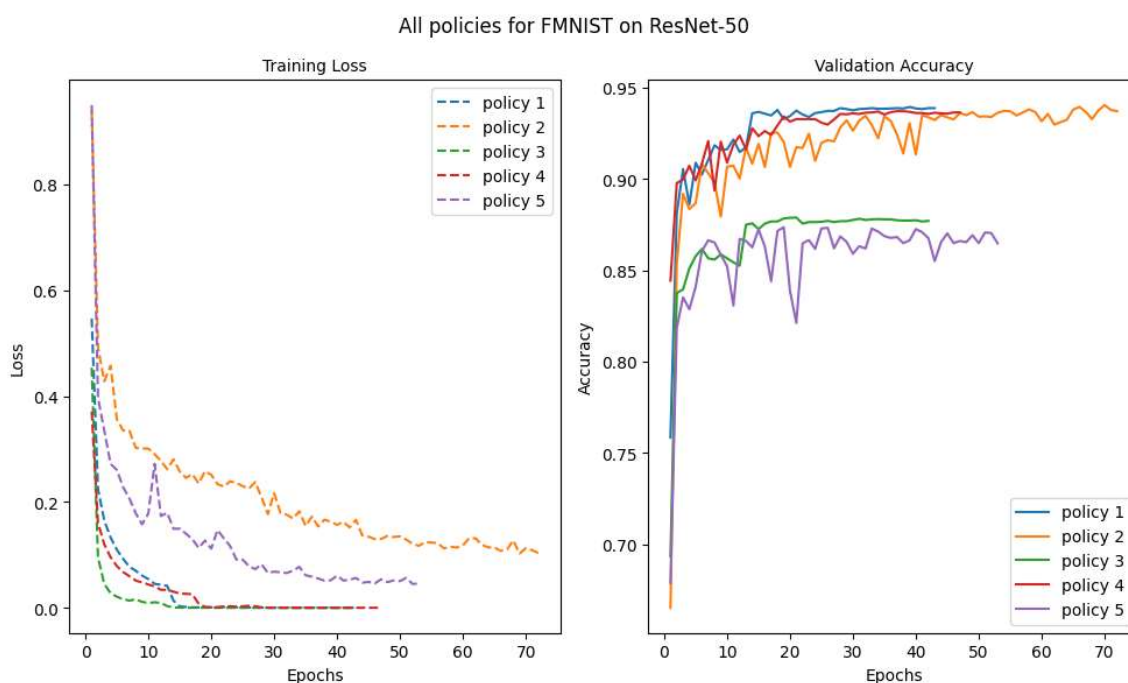
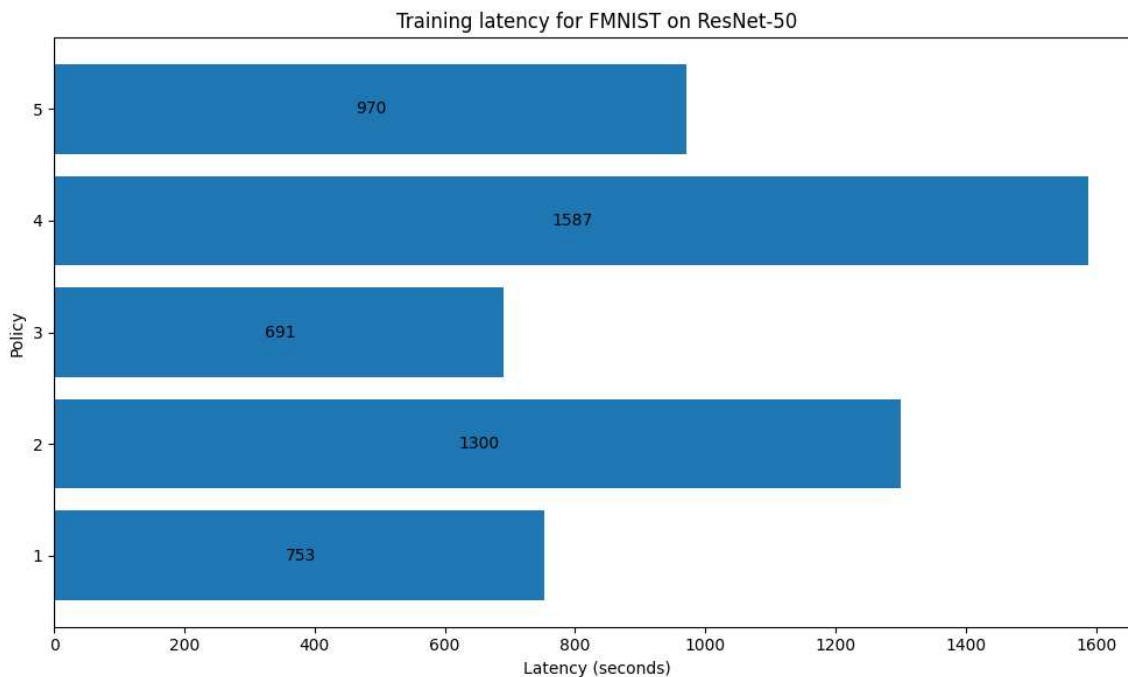


Figure 29 summarizes the training loss and validation accuracy of all policies for FMNIST on ResNet-50. Policy 1 established the baseline metrics and had a relatively fast convergence. Policy 2 had the longest convergence time at 72 epochs, followed by Policy 5, which converged at 53. This can be attributed to the stochasticity of RandAugment used by those two policies. Policy 2 had the highest accuracy, followed by Policy 4. Policy 3 had the fastest convergence but the lowest accuracy of all policies. The loss curves for policies 2 and 5, the two policies implementing RandAugment, fit the profile

of the smooth polynomial curves seen in models with minimal overfitting like Figures 7 and 8. This suggests that RandAugment is a very effective method of reducing overfitting in ConvNets.

Figure 30

Training Latency of All Policies for FMNIST on ResNet-50



The training latencies of all policies for FMNIST on ResNet-50 are seen in Figure 30. Policy 4 had the longest training time, taking 1587 seconds, more than twice as long as policies 1 and 3. On the other hand, Policy 2 took approximately 1.3x that of Policy 5, which was unexpected since they both used RandAugment and expected to have similar training times like policies 2 and 5 for MNIST on ResNet50. Policy 3 had the shortest training time, followed by Policy 1, mainly because they are not augmented or combined with any other dataset. In comparison, the training latency for FMNIST was circa 2 times faster on NoelNet than on ResNet.

In summary, for FMNIST on ResNet-50, it was observed that Policies 2 and 4 had the highest accuracies and were higher than Policy 1, implying that these two policies increased the spatial invariance of the model. On the other hand, policy 3 has the lowest accuracy, followed by Policy 5, with accuracies less than Policy 1. The latter suggests that policies 3 and 5 did not improve the spatial invariance of the model. Furthermore, Table 6 shows that the precision and recall for each policy closely match the trend in their accuracies, implying that the accuracy reflects the model's "true" performance, the classes are evenly distributed, and the model is not hallucinating. Similar patterns were observed in Table 3 for FMNIST on NoelNet.

CIFAR-10 on ResNet-50

The results obtained after analyzing each policy for the CIFAR-10 dataset on the ResNet50 model are as follows.

Dataset Images

Refer to Figure 21 for a description of and samples of three variations of the CIFAR-10 dataset used in the experiments, either by themselves or in combinations.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 7

Performance Test Metrics of each Policy for CIFAR-10 on ResNet-50

dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
cifar10	resnet	1	0.7339	0.7476	0.7339	0.7358	0
cifar10	resnet	2	0.8517	0.8535	0.8517	0.8523	16.05
cifar10	resnet	3	0.6286	0.6351	0.6286	0.6258	-14.35
cifar10	resnet	4	0.7671	0.7759	0.7671	0.7683	4.52
cifar10	resnet	5	0.6915	0.7006	0.6915	0.6937	-5.78

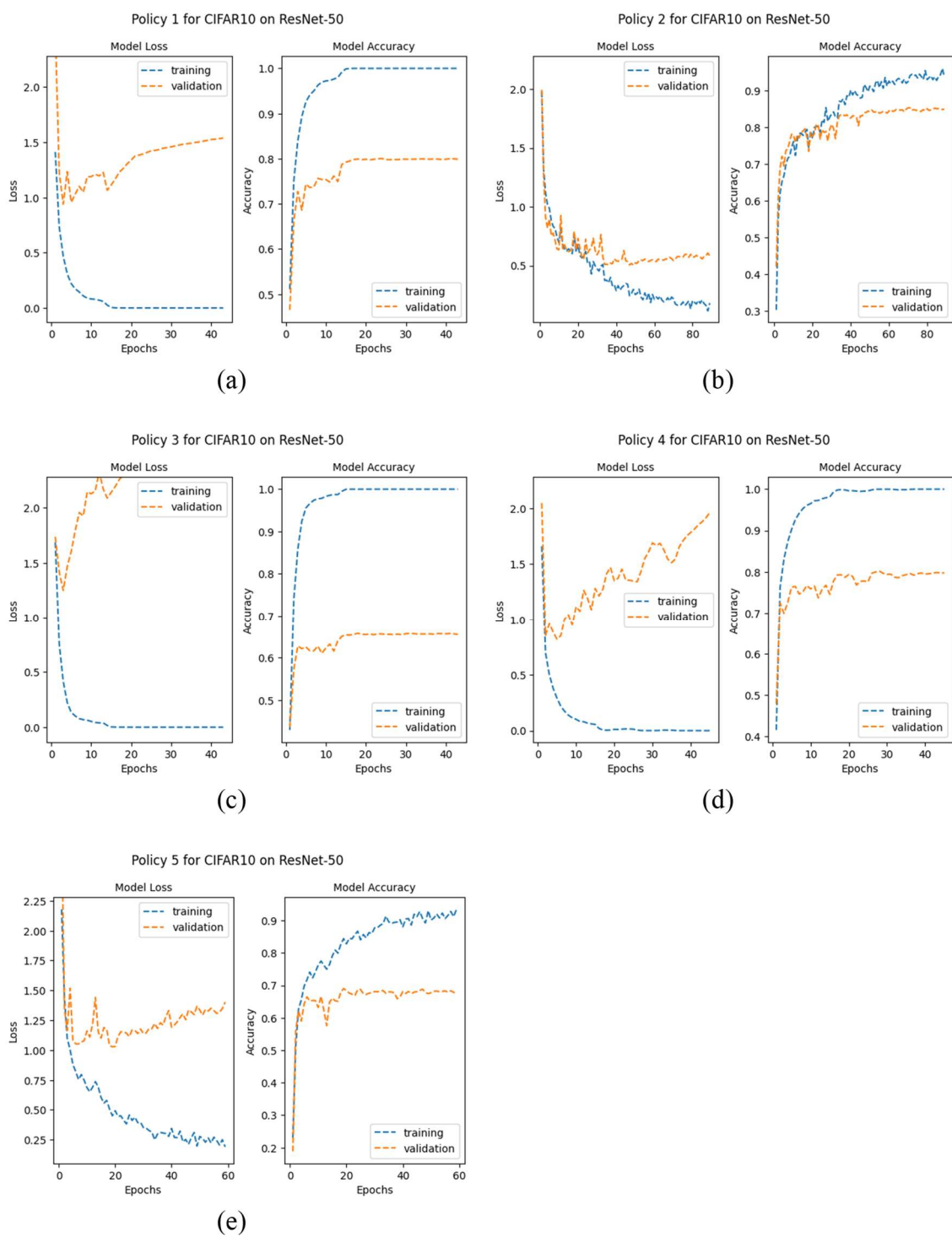
Note. The table displays the performance metrics for the test samples, where the delta column shows the percentage change in accuracy for each policy relative to Policy 1. Higher percentages imply improved performance and generalizability.

Table 7 summarizes the performance of each policy for CIFAR-10 on ResNet-50, where Policy 1 established a baseline. Policy 2 and Policy 4 had accuracies of 85.17% and 76.71%, respectively, which translated to improvements of 16.05% and 4.52%, respectively, relative to the baseline. On the other hand, both policies 3 and 5 had accuracies less than the baseline, 62.86% and 69.15%, respectively, which translated to decreases of 14.35% and 5.78%, respectively. The veracity of the accuracy for each policy is further confirmed by the f1 score being, on average, within 0.25% of each policy's accuracy. From Table 7, it can be surmised that Policies 2 and 4 increased the spatial invariance of the model, while Policies 3 and 5 reduced the spatial invariance of the model.

Similar patterns were observed in Table 4 for CIFAR-10 on NoelNet in that Policy 2 achieved the highest accuracy of 88.95%, while Policy 1 achieved a competitive accuracy of 85.16%. However, unlike previously, Policy 4 had a slightly lower accuracy at 83.64% compared to Policy 1. Finally, Policy 5 achieved 68.59%, while Policy 3 had the lowest accuracy at 57.83%. Nonetheless, Policies 1, 2, and 4 on NoelNet for CIFAR-10 resulted in higher accuracy and f1 scores than on ResNet-50.

Figure 31

Accuracy and Loss Graphs of each Policy for CIFAR-10 on ResNet-50



Policy 1

Figure 31(a) shows the loss and accuracy curves for Policy 1. The chart shows that the training and validation curves start diverging around epoch 7, such that the gap between the graphs quickly becomes more prominent, suggesting a quick increase in overfitting is occurring. The model reached convergence at epoch 43, when the validation loss reached its minimum and training was terminated. This convergence was faster than that for Policy 1 on MNIST but equal to Policy 1 on FMNIST, suggesting that it is easier to train CIFAR-10 and FMNIST on ResNet50 than to train MNIST. From Table 7, Policy 1 achieved an accuracy of 73.39% and an f1 score of 73.58%. The f1 score was competitive (i.e., within 0.07%) with accuracy, which suggested that the class distribution is balanced and that results produced by the model correctly reflected its performance, i.e., predictions made by the model should be at least 73% accurate.

Policy 2

Figure 31(b) shows the loss and accuracy graphs for Policy 2. The accuracy and loss graphs show a much closer correlation in relative trends than Policy 1. The graphs started diverging around epoch 25 instead of epoch 7, as with Policy 1. This implies the model was learning the increased invariances in the augmented data and thus was generalizing better on the unseen (i.e., the validation) data. The gap between each curve is also smaller than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is less than in Policy 1. The model had a slower convergence rate (~2x slower) compared to Policy 1 in that training terminated at Epoch 89 instead of Epoch 43 as it was for Policy 1. From Table 7, Policy 2 achieved an accuracy of 85.17% and an f1 score of 85.23%. This accuracy represents an impressive 16.05% improvement

over Policy 1 and the unperturbed dataset, a significant improvement in the spatial invariance of the model by using RandAugment.

Policy 3

Figure 31(c) shows the loss and accuracy graphs for Policy 3. The graphs can be seen to have much wider gaps as compared to policies 1 and 2, and they also diverged very early in the training process, around epoch 3, suggesting that the synthetic samples caused much higher levels of overfitting. The model converged at epoch 43, the same as Policy 1 and much quicker than Policy 2. This suggests Policy 3 imbued the dataset with smaller invariances than Policy 2. From Table 7, Policy 3 achieved an accuracy of 62.86% and an f1 score of 62.58%, representing a 14.35% decrease from Policy 1. Also, the slightly smaller f1 score indicates slight imbalances in the class distributions. The decrease in accuracy relative to Policy 1 implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

Figure 31(d) shows the loss and accuracy graphs for Policy 4. The gaps in the loss graphs are slightly more significant than Policy 1, much wider than Policy 2, and smaller than Policy 3, indicating that Policy 4 may lead to more overfitting than Policies 1 and 2 but less overfitting than Policy 3. The model converged at epoch 45, which was much faster than that of Policy 2 but slower than Policies 1 and 3. From Table 7, Policy 4 achieved an accuracy of 76.71% and an f1 score of 76.83%, representing a 4.52% improvement over Policy 1. In a similar fashion to Policy 4 for F/MNIST on ResNet50, it is observed that combining the synthesized samples from Policy 3 with their baseline counterpart led to improved spatial invariance in the trained model.

Policy 5

Figure 31(e) shows the loss and accuracy graphs for Policy 5. The gap between the curves is smaller than policies 1, 3, and 4, which suggests less overfitting and that applying RandAugment to synthesized samples can improve spatial invariance compared to using the synthesized samples in isolation. This observation is corroborated by Table 7, where Policy 5 has an accuracy of 69.15% and an f1 score of 69.37%, while Policy 3 has an accuracy of 62.86%. Although the accuracy of this Policy is 5.78% lower than the baseline, thus not increasing the model's spatial invariance, it is still 8.57% higher than Policy 3. The model convergence of Policy 5, at 59 epochs, was slower than Policies 1, 3, and 4 but, like Policy 2, implies that RandAugment introduced more sources of invariances than Policies 1, 3, and 4, thus taking the model longer to converge.

Figure 32

Loss and Validation Accuracy of All Policies for CIFAR-10 on ResNet-50

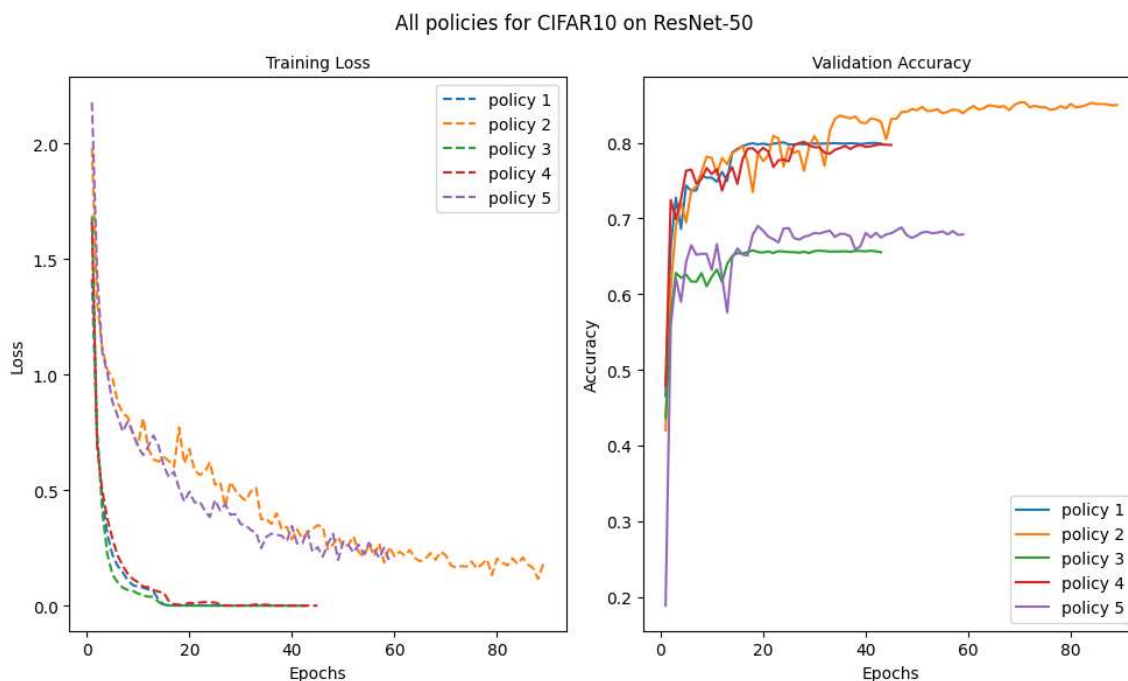
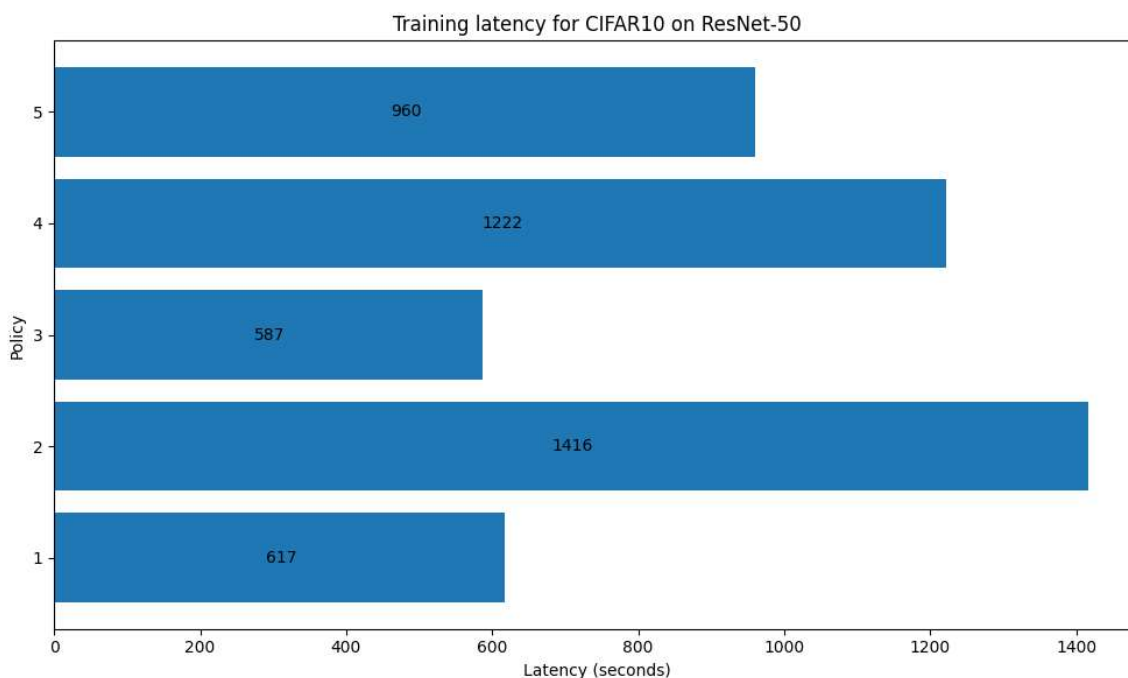


Figure 32 summarizes the training loss and validation accuracy of all policies for CIFAR-10 on ResNet-50. Policy 1 established the baseline metrics and had a relatively fast convergence. Policy 2 had the longest convergence time at 89 epochs, followed by Policy 5, which converged at epoch 59. This can be attributed to the stochasticity of RandAugment used by those two policies. Policy 2 had the highest accuracy, followed by Policy 4. Policy 3 had the fastest convergence but the lowest accuracy of all policies. The loss curves for policies 2 and 5, the two policies implementing RandAugment, fit the profile of the smooth polynomial curves seen in models with minimal overfitting like Figures 7 and 8. This suggests that RandAugment is a very effective method of reducing overfitting in ConvNets.

Figure 33

Training Latency of All Policies for CIFAR-10 on ResNet-50



The training latencies of all policies for CIFAR-10 on ResNet-50 are seen in Figure 33. Policy 2 had the longest training time, taking 1416 seconds, more than twice as long as policies 1 and 3 and approximately 1.5x that of Policy 5. The latter was unexpected since policies 2 and 5 used RandAugment and expected similar training times to policies 2 and 5 for MNIST on ResNet50. The last observation may be because the synthesized samples used in Policy 5 are poorer in quality and complexity than the samples used by Policy 2, and augmenting them using RandAugment did not increase their invariance to the levels seen in Policy 2. Policy 3 had the shortest training time, followed by Policy 1, mainly because they are not augmented or combined with any other dataset. In comparison, the training latency for CIFAR-10 was circa 1.2 times faster on NoelNet than on ResNet.

In summary, for CIFAR-10 on ResNet-50, it was observed that Policies 2 and 4 had the highest accuracies and were higher than Policy 1, implying that these two policies increased the spatial invariance of the model. On the other hand, policy 3 has the lowest accuracy, followed by Policy 5, with accuracies less than Policy 1. The latter suggests that policies 3 and 5 did not improve the spatial invariance of the model. Furthermore, Table 7 shows that the precision and recall for each policy closely match the trend in their accuracies, implying that the accuracy reflects the model's "true" performance, the classes are evenly distributed, and the model is not hallucinating. Except for Policy 4, these same patterns were observed in Table 4 for CIFAR-10 on NoelNet.

MNIST on InceptionV3

The results obtained after analyzing each policy for the MNIST dataset on the InceptionV3 model are as follows.

Dataset Images

Refer to Figure 13 for a description of and samples of three variations of the MNIST dataset used in the experiments, either by themselves or in combinations thereof.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 8

Performance Test Metrics of each Policy for MNIST on InceptionV3

dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
mnist	inception	1	0.9943	0.9944	0.9942	0.9943	0
mnist	inception	2	0.9958	0.9959	0.9957	0.9958	0.15
mnist	inception	3	0.9892	0.9894	0.9891	0.9892	-0.51
mnist	inception	4	0.9956	0.9956	0.9955	0.9956	0.13
mnist	inception	5	0.9911	0.9911	0.9911	0.9911	-0.32

Note. The table displays the performance metrics for the test samples, where the delta column shows the percentage change in accuracy for each policy relative to Policy 1. Higher percentages imply improved performance and generalizability.

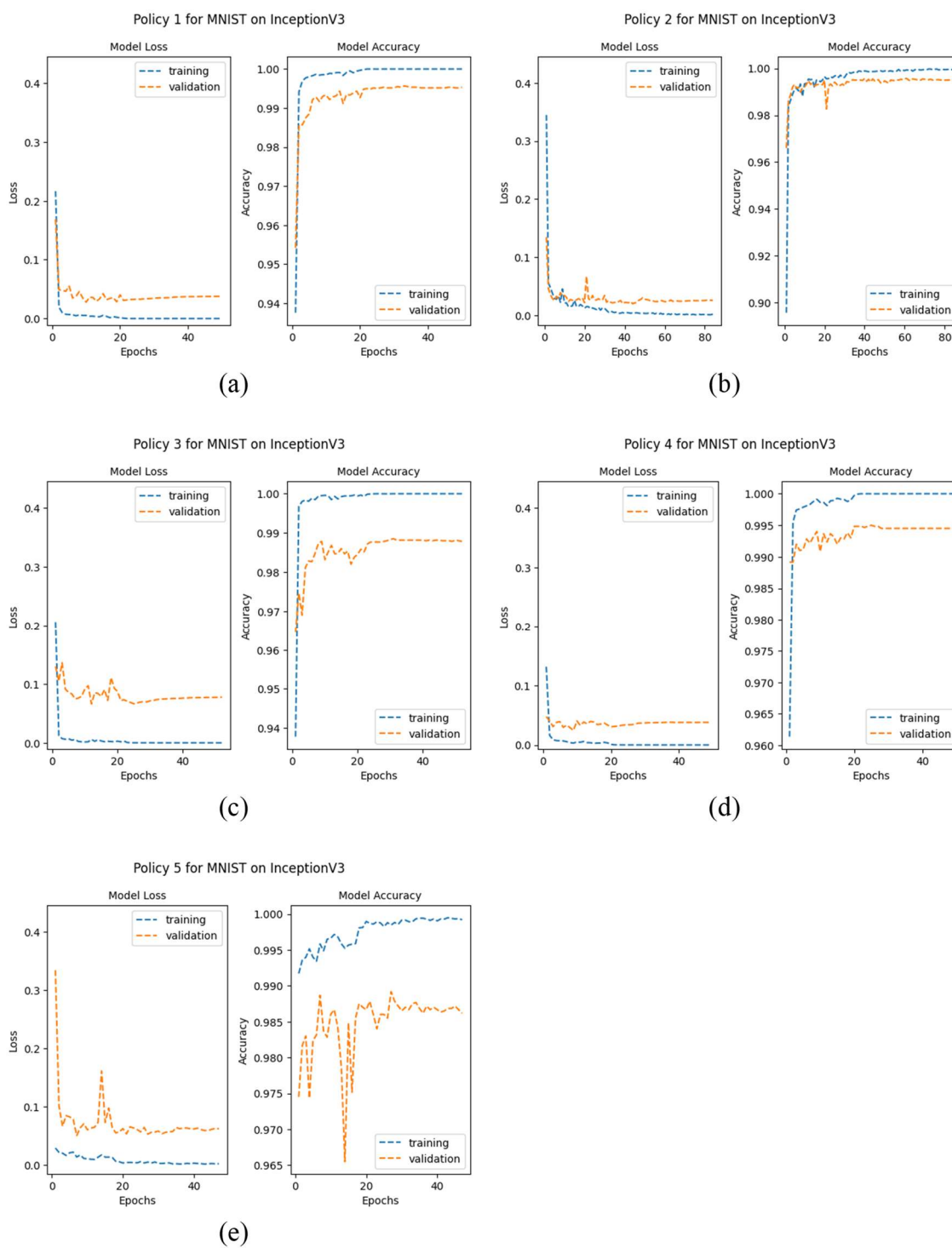
Table 8 summarizes the performance of each policy for MNIST on InceptionV3, where Policy 1 established a baseline. Policy 2 and Policy 4 had accuracies of 99.58% and 99.56%, respectively, which translated to improvements of 0.15% and 0.13%, respectively, relative to the baseline. The baseline had an accuracy of 99.43% which was a 0.3% improvement over the baseline established on ResNet-50. On the other hand, both policies 3 and 5 had accuracies less than the baseline, 98.92% and 99.11%, respectively, which translated to decreases of 0.51% and 0.32%, respectively. The veracity of the accuracy for each policy is further confirmed by the f1 score being equal to each policy's accuracy. This represents another difference from ResNet-50, where the f1 scores were

non-zero. From Table 8, it can be surmised that Policies 2 and 4 increased the spatial invariance of the model, while Policies 3 and 5 reduced the spatial invariance of the model. Furthermore, the accuracies, precision, and recall for MNIST are higher on InceptionV3 than on ResNet-50.

Similar patterns were observed in Table 2 for MNIST on NoelNet in that Policy 2 achieved the highest accuracy and f1 scores, followed by Policy 4. Policy 1 (the unaugmented data policy) achieved the same accuracy as MNIST on InceptionV3, while Policy 5's accuracy was lower than Policy 1. Finally, Policy 3 had the lowest accuracy and f1 scores. Notably, each Policy on NoelNet for MNIST had competitive accuracies and f1 scores relative to InceptionV3.

Figure 34

Accuracy and Loss Graphs of each Policy for MNIST on InceptionV3



Policy 1

Figure 34(a) shows the loss and accuracy curves for Policy 1. The chart shows that the validation graphs follow the training graphs very closely, so the gap between the graphs is minimal, suggesting negligible overfitting. The model reached convergence at epoch 50, when the validation loss reached its minimum and training was terminated. From Table 8, Policy 1 achieved an accuracy of 99.43% and an f1 score of 99.43%. This was a 0.3% improvement over the performance of MNIST on ResNet-50. The f1 score is the same as the accuracy, which suggests that the class distribution is balanced and corroborates the model's accuracy as a "true" accuracy, i.e., the results produced by the model correctly reflect the performance of the model and the predictions made by the model should be at least 99% accurate.

Policy 2

Figure 34(b) shows the loss and accuracy graphs for Policy 2. The accuracy and loss graphs show a very tight correlation in their relative patterns. The gap between each curve is smaller than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is smaller than in Policy 1. The model also had a slower convergence rate ($\sim 1.7x$ slower) compared to Policy 1 in that training terminated at Epoch 84 instead of Epoch 50 as it was for Policy 1. The increased convergence rate suggests increased invariance produced by RandAugment, which the model is taking longer to learn. From Table 8, Policy 2 achieved an accuracy of 99.58% and an f1 score of 99.58%. This accuracy represents a 0.15% improvement over Policy 1 and the unperturbed dataset. This improvement in accuracy may be small but significant in that off-the-shelf ConvNets quickly learn the MNIST dataset. These ConvNets, when trained

on MNIST, can achieve over 99% accuracy consistently and without optimizations. The InceptionV3 model used in this Policy is highly optimized; thus, techniques leading to further improved accuracy in its performance on any dataset are noteworthy.

Policy 3

Figure 34(c) shows the loss and accuracy graphs for Policy 3. The graphs can be seen to have more significant gaps as compared to policies 1 and 2, suggesting the synthetic samples cause higher levels of overfitting. The model converged at epoch 52, faster than Policy 2 but slower than Policy 1. This contrasts with Policy 1 on ResNet-50, where Policy 3 had the fastest convergence of all policies. This is suggestive that InceptionV3, being a wider network with several different kernels being applied in parallel, can detect more invariances within the dataset. From Table 8, Policy 3 achieved an accuracy of 98.92% and an f1 score of 98.92%, representing a 0.51% decrease from Policy 1. The latter implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

Figure 34(d) shows the loss and accuracy graphs for Policy 4. The gaps in the graphs are slightly wider than those of Policies 1 and 2 but smaller than Policy 3, indicating that Policy 4 may lead to more overfitting than Policies 1 and 2 but less overfitting than Policy 3. The model converged at epoch 49 faster than policies 1, 2, and 3. From Table 8, Policy 4 achieved both accuracy and an f1 score of 99.56%, representing a 0.13% improvement in accuracy over Policy 1. This improvement over the baseline model suggests that Policy 4 improves spatial invariance. This improvement directly results from the additional invariances provided by the GAN samples when

combined with the baseline dataset. When used by themselves and not combined with any other dataset, the synthesized samples from Policy 3 led to a decrease in spatial invariance. However, when combined with their baseline counterpart, as seen in Policy 4, it increases the spatial invariance of the trained model. The same effect is observed in Policy 4 for MNIST on ResNet-50.

Policy 5

Figure 34(e) shows the loss and accuracy graphs for Policy 5. The gap between the accuracy graphs is larger than that of all other policies, and the gap between the loss graphs is more extensive than all other policies except Policy 3. This suggests that Policy 5 leads to more overfitting than Policies 1, 2, and 4 but less overfitting than Policy 3. Applying RandAugment to the synthesized samples can improve spatial invariance compared to using the synthesized samples in isolation. This observation is corroborated by Table 8, where Policy 5 has an accuracy and an f1 score of 99.11%, while Policy 3 has an accuracy of 98.92%. Although the accuracy of Policy 5 is 0.32% lower than the baseline, thus not increasing the model's spatial invariance, it is still 0.19% higher than Policy 3. Furthermore, the model convergence of Policy 5, at 47 epochs, was faster than the other policies.

Figure 35

Loss and Validation Accuracy of All Policies for MNIST on InceptionV3

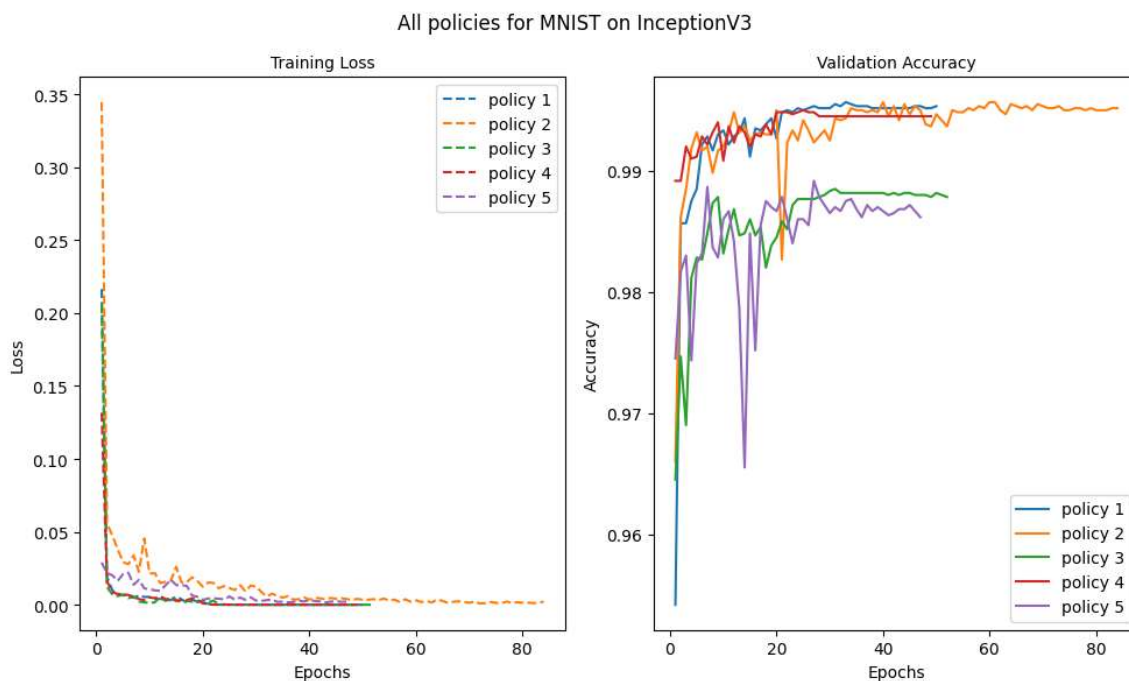
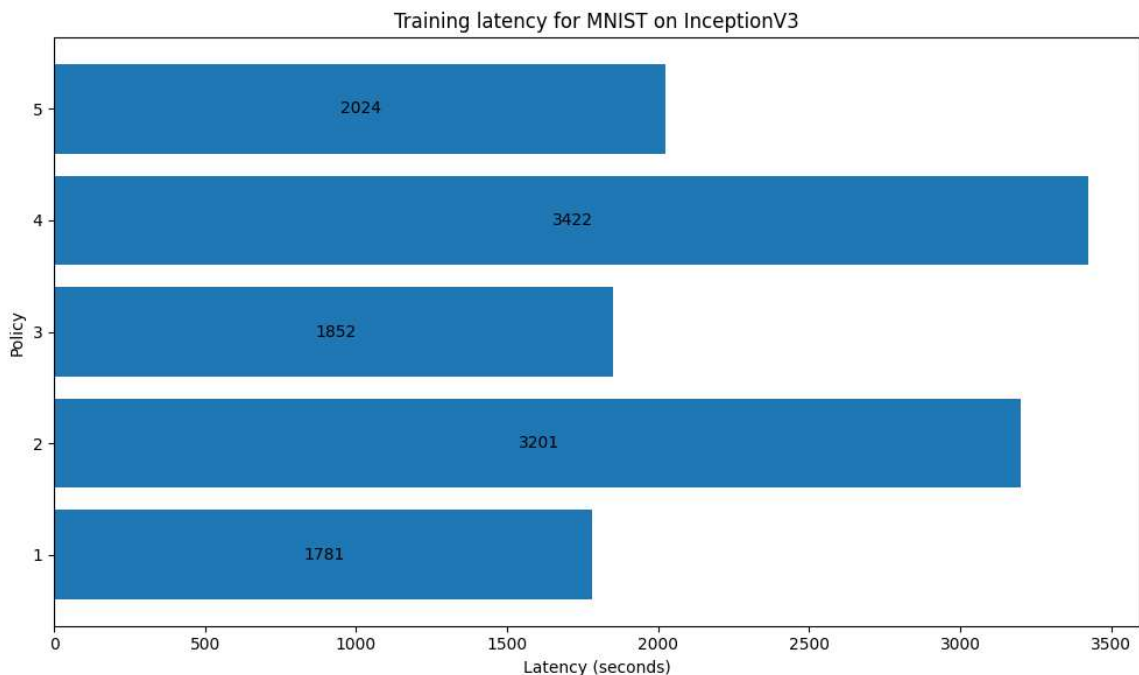


Figure 35 summarizes the training loss and validation accuracy of all policies for MNIST on InceptionV3. Policy 1 established the baseline metrics and had a relatively fast convergence. Policy 2 had the longest convergence time at 84 epochs (attributed to the stochasticity of RandAugment), followed by Policy 3, which converged at epoch 52. Policy 2 had the highest accuracy, followed by Policy 4. Policy 3 had a higher convergence than policies 1, 4, and 5 but the lowest accuracy of all policies. The loss curves for policies 2 and 5, the two policies implementing RandAugment, fit the profile of the smooth polynomial curves seen in models with minimal overfitting like Figures 7 and 8. This suggests that RandAugment is a very effective method of reducing overfitting in ConvNets.

Figure 36

Training Latency of All Policies for MNIST on InceptionV3



The training latencies of all policies for MNIST on InceptionV3 are seen in Figure 36. Policy 4 had the longest training time, taking 3422 seconds, more than twice as long as policies 1 and 3. Policy 2 took approximately 1.7x that of Policy 5, which was unexpected since policies 2 and 5 used RandAugment. The last observation may be because the synthesized samples used in Policy 5 are poorer in quality and complexity than the samples used by Policy 2, and augmenting them using RandAugment did not increase their invariance to the levels seen in Policy 2. Policy 1 had the shortest training time, followed by Policy 3, mainly because they are not augmented or combined with any other dataset. In comparison, the training latency for MNIST was approximately 3 times faster on NoelNet than on InceptionV3.

In summary, for MNIST on InceptionV3, it was observed that Policies 2 and 4 had the highest accuracies and were higher than Policy 1, implying that these two policies increased the spatial invariance of the model. On the other hand, policy 3 has the lowest accuracy, followed by Policy 5, with accuracies less than Policy 1. The latter suggests that policies 3 and 5 did not improve the spatial invariance of the model. Furthermore, Table 8 shows that the precision and recall for each policy closely match the trend in their accuracies, implying that the accuracy reflects the model's "true" performance, the classes are evenly distributed, and the model is not hallucinating. These same patterns were observed in Table 2 for MNIST on NoelNet.

Fashion MNIST on InceptionV3

The results obtained after analyzing each policy for the FMNIST dataset on the InceptionV3 model are as follows.

Dataset Images

Refer to Figure 17 for a description of and samples of three variations of the FMNIST dataset used in the experiments, either by themselves or in combinations thereof.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 9

Performance Test Metrics of each Policy for FMNIST on InceptionV3

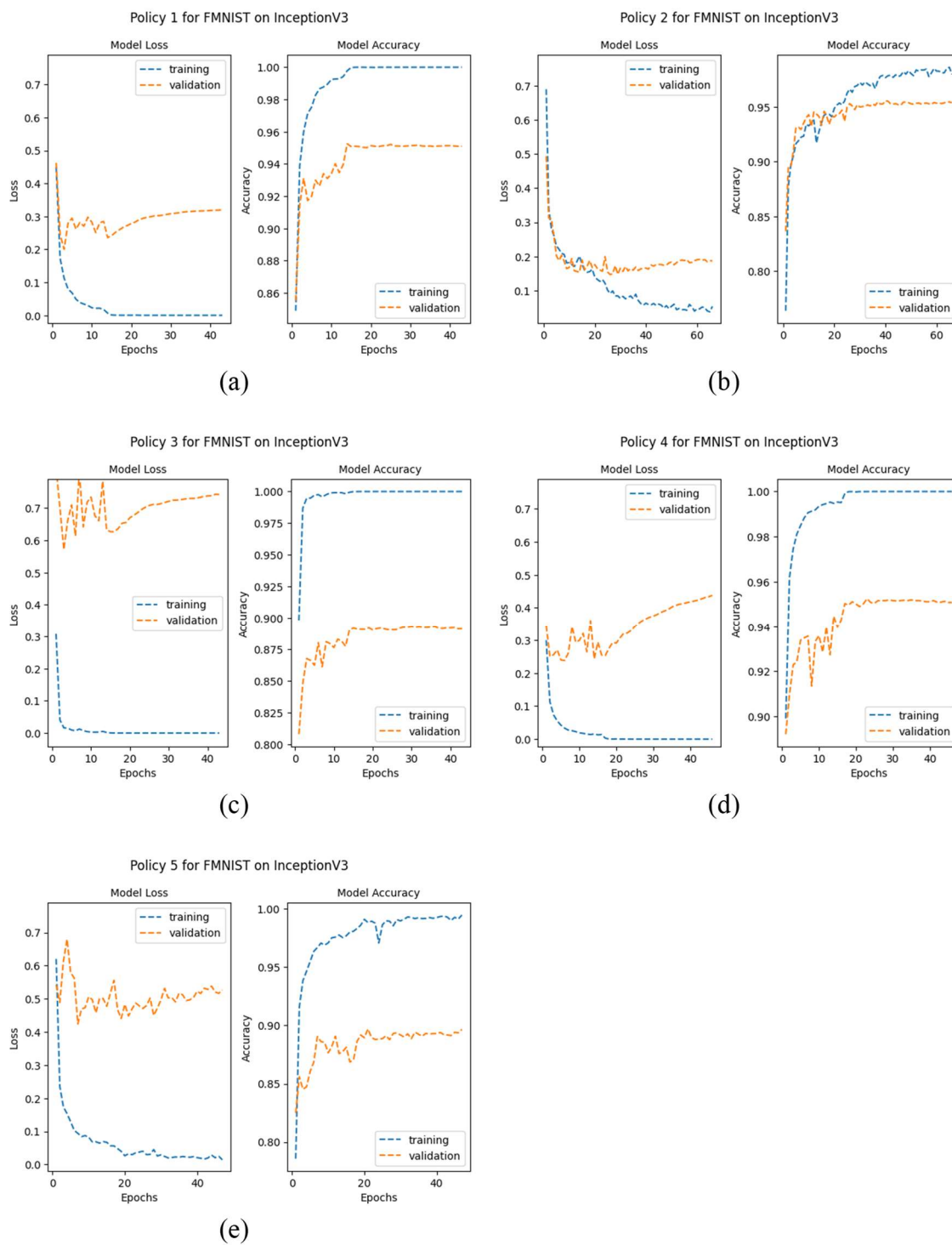
dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
fmnist	inception	1	0.9167	0.9166	0.9167	0.9163	0
fmnist	inception	2	0.9482	0.9484	0.9482	0.9482	3.44
fmnist	inception	3	0.8527	0.8595	0.8527	0.8543	-6.98
fmnist	inception	4	0.9353	0.9364	0.9353	0.935	2.03
fmnist	inception	5	0.8825	0.8831	0.8825	0.8821	-3.73

Note. The table displays the performance metrics for the test samples, where the delta column shows the percentage change in accuracy for each policy relative to Policy 1. Higher percentages imply improved performance and generalizability.

Table 9 summarizes the performance of each policy for FMNIST on InceptionV3, where Policy 1 established a baseline. Policy 2 and Policy 4 had accuracies of 94.82% and 93.53%, respectively, which translated to improvements of 3.44% and 2.03%, respectively, relative to the baseline. On the other hand, both policies 3 and 5 had accuracies less than the baseline, 85.27% and 88.25%, respectively, which translated to decreases of 6.98% and 3.73%, respectively. The veracity of the accuracy for each policy is further confirmed by the f1 score being, on average, within 0.06% of each policy's accuracy. From Table 9, it can be surmised that Policies 2 and 4 increased the spatial invariance of the model, while Policies 3 and 5 reduced the spatial invariance of the model. The former and latter were also observed in Table 6 for FMNIST on ResNet-50 and Table 3 for FMNIST on NoelNet. Furthermore, the accuracies, precision, and recall for FMNIST are higher on InceptionV3 than on ResNet-50 but competitive with NoelNet.

Figure 37

Accuracy and Loss Graphs of each Policy for FMNIST on InceptionV3



Policy 1

Figure 37(a) shows the loss and accuracy curves for Policy 1. The chart shows that the training and validation curves start diverging around epoch 5, such that the gap between the graphs quickly becomes more prominent, suggesting a quick increase in overfitting is occurring. The model reached convergence at epoch 43, when the validation loss reached its minimum and training was terminated. This convergence was much faster than that for Policy 1 on MNIST, which suggests that it is easier to train FMNIST on InceptionV3. From Table 9, Policy 1 achieved an accuracy of 91.67% and an f1 score of 91.63%. This was a 2% improvement over the performance of FMNIST on ResNet-50. The f1 score is approximately the same as the accuracy, which suggests that the class distribution is balanced, and the results produced by the model correctly reflect its performance, i.e., predictions made by the model should be at least 91% accurate.

Policy 2

Figure 37(b) shows the loss and accuracy graphs for Policy 2. The accuracy and loss graphs show a much closer correlation in relative trends than Policy 1. The gap between each curve is also smaller than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is less than in Policy 1 primarily because of the increased levels of invariances imbued by RandAugment. The model had a slower convergence rate ($\sim 1.5x$ slower) compared to Policy 1 in that training terminated at Epoch 66 instead of Epoch 43 as it was for Policy 1. From Table 9, Policy 2 achieved an accuracy and f1 score of 94.82%. This accuracy represents a 3.44% improvement over Policy 1 and the unperturbed dataset, a notable improvement in the spatial invariance of the model by using RandAugment.

Policy 3

Figure 37(c) shows the loss and accuracy graphs for Policy 3. The graphs can be seen to have much wider gaps as compared to policies 1 and 2, suggesting the synthetic samples caused much higher levels of overfitting very early in the training process. The model converged at epoch 43, which was faster than that of Policy 2 but the same as Policy 1. From Table 9, Policy 3 achieved an accuracy of 85.27% and an f1 score of 85.43%, representing an almost 7% decrease from Policy 1. Also, the slightly higher f1 score indicates slight imbalances in the class distributions. The decrease in accuracy relative to Policy 1 implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

Figure 37(d) shows the loss and accuracy graphs for Policy 4. The gaps in the graphs are wider than those of Policies 1 and 2 but smaller than Policy 3, indicating that Policy 4 may lead to more overfitting than Policies 1 and 2 but less overfitting than Policy 3. The model converged at epoch 46, faster than Policy 2 but slower than Policies 1 and 3. From Table 9, Policy 4 achieved an accuracy of 93.53% and an f1 score of 93.50%, representing an almost 2% improvement over Policy 1. In a similar fashion to Policy 4 for FMNIST on ResNet-50, it is observed that combining the synthesized samples from Policy 3 with their baseline counterpart led to improved spatial invariance in the trained model.

Policy 5

Figure 37(e) shows the loss and accuracy graphs for Policy 5. The gap between the curves is smaller than Policy 3 (has less overfitting) but wider than all the other

policies. This suggests that applying RandAugment to synthesized samples can improve spatial invariance compared to using the synthesized samples in isolation. This observation is corroborated by Table 9, where Policy 5 has an accuracy of 88.25% and an f1 score of 88.21%, while Policy 3 has an accuracy of 85.27%. Although the accuracy of Policy 5 is 3.73% lower than the baseline, thus not increasing the model's spatial invariance, it is still 3.25% higher than Policy 3. The model convergence of Policy 5, at 47 epochs, was slower than Policies 1, 3, and 4 but faster than Policy 2, implying that RandAugment introduced more sources of invariances than Policies 1, 3, and 4, thus taking the model longer to converge.

Figure 38

Loss and Validation Accuracy of All Policies for FMNIST on InceptionV3

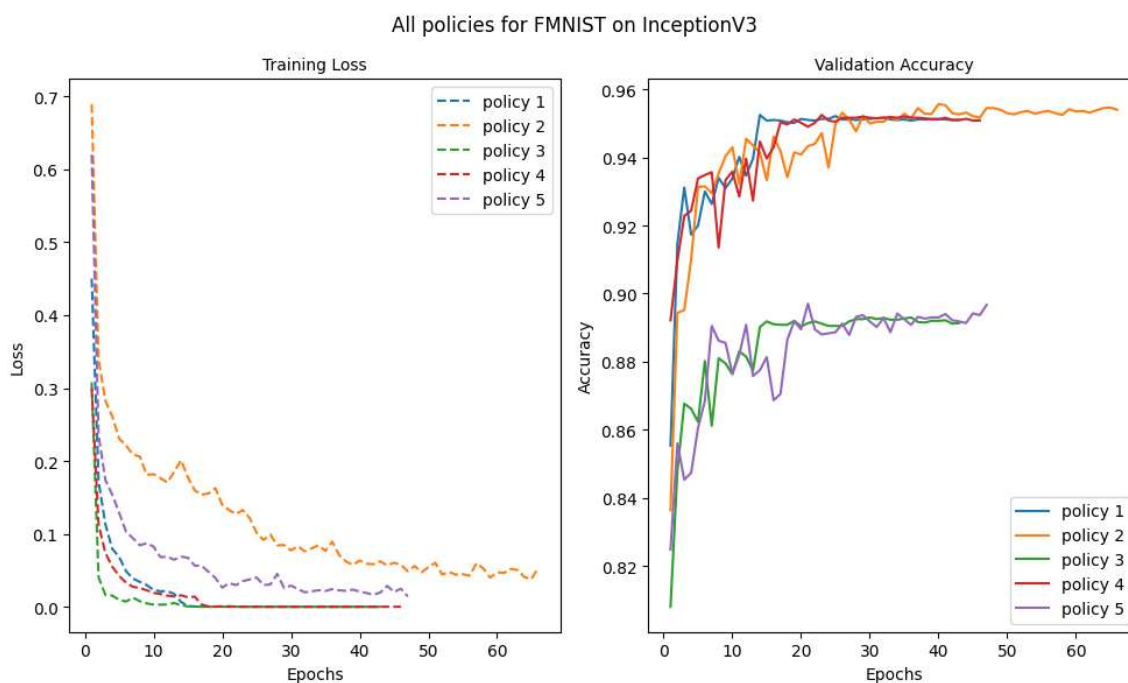


Figure 38 summarizes the training loss and validation accuracy of all policies for FMNIST on InceptionV3. Policy 1 established the baseline metrics and had a relatively

fast convergence. Policy 2 had the longest convergence time at 66 epochs, followed by Policy 5, which converged at 47. This can be attributed to the stochasticity of RandAugment used by those two policies. Policy 2 had the highest accuracy, followed by Policy 4, as observed for all datasets on ResNet-50 and InceptionV3. Policy 3 and Policy 1 had the fastest convergence, but Policy 3 had the lowest accuracy of all policies. The loss curves for policies 2 and 5, the two policies implementing RandAugment, fit the profile of the smooth polynomial curves seen in models with minimal overfitting like Figures 7 and 8. This suggests that RandAugment is a very effective method of reducing overfitting in ConvNets.

Figure 39

Training Latency of All Policies for FMNIST on InceptionV3

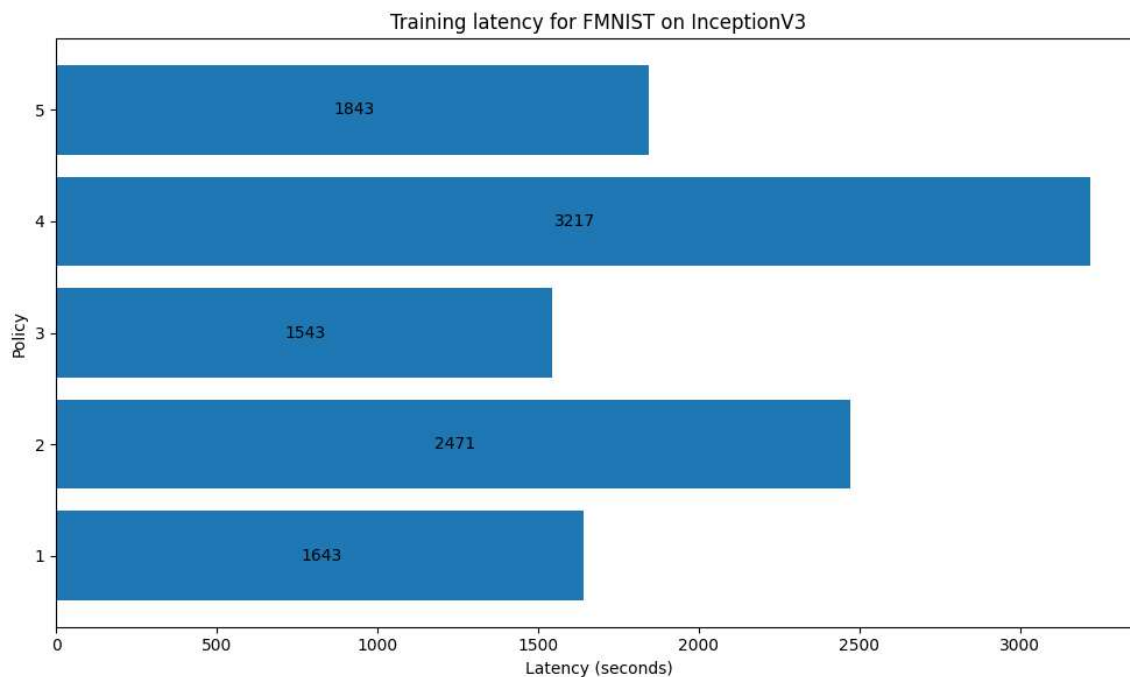


Figure 39 shows the training latencies of all policies for FMNIST on InceptionV3. Policy 4 had the longest training time, taking 3217 seconds, more than twice as long as

policies 1 and 3. On the other hand, Policy 2 took approximately 1.3x that of Policy 5, which was unexpected since they both used RandAugment and expected similar training times. Policies 3 and 1 had the shortest training time, mainly because they were not augmented or combined with any other dataset. In comparison, the training latency for FMNIST was approximately 4 times faster on NoelNet than on ResNet.

In summary, for FMNIST on InceptionV3, it was observed that Policies 2 and 4 had the highest accuracies and were higher than Policy 1, implying that these two policies increased the spatial invariance of the model. On the other hand, policy 3 has the lowest accuracy, followed by Policy 5, with accuracies less than Policy 1. The latter suggests that policies 3 and 5 did not improve the spatial invariance of the model. However, the precision for Policy 3 was slightly higher than the recall and accuracy. This implies that the accuracy for Policy 3 may not reflect the model's "true" performance or that the class distribution for Policy 3 was not evenly distributed. These same patterns were observed in Table 3 for FMNIST on NoelNet.

CIFAR-10 on InceptionV3

The results obtained after analyzing each policy for the CIFAR-10 dataset on the InceptionV3 model are as follows.

Dataset Images

Refer to Figure 21 for a description of and samples of three variations of the CIFAR-10 dataset used in the experiments, either by themselves or in combinations.

Policy Results

The following presents the metrics and charts obtained from training each policy.

Table 10

Performance Test Metrics of each Policy for CIFAR-10 on InceptionV3

dataset	model	policy	accuracy	precision	recall	f1score	delta(%)
cifar10	inception	1	0.8196	0.8229	0.8196	0.8192	0
cifar10	inception	2	0.9134	0.9142	0.9134	0.9135	11.44
cifar10	inception	3	0.6483	0.668	0.6483	0.6441	-20.9
cifar10	inception	4	0.8292	0.8389	0.8292	0.829	1.17
cifar10	inception	5	0.6954	0.7068	0.6954	0.6981	-15.15

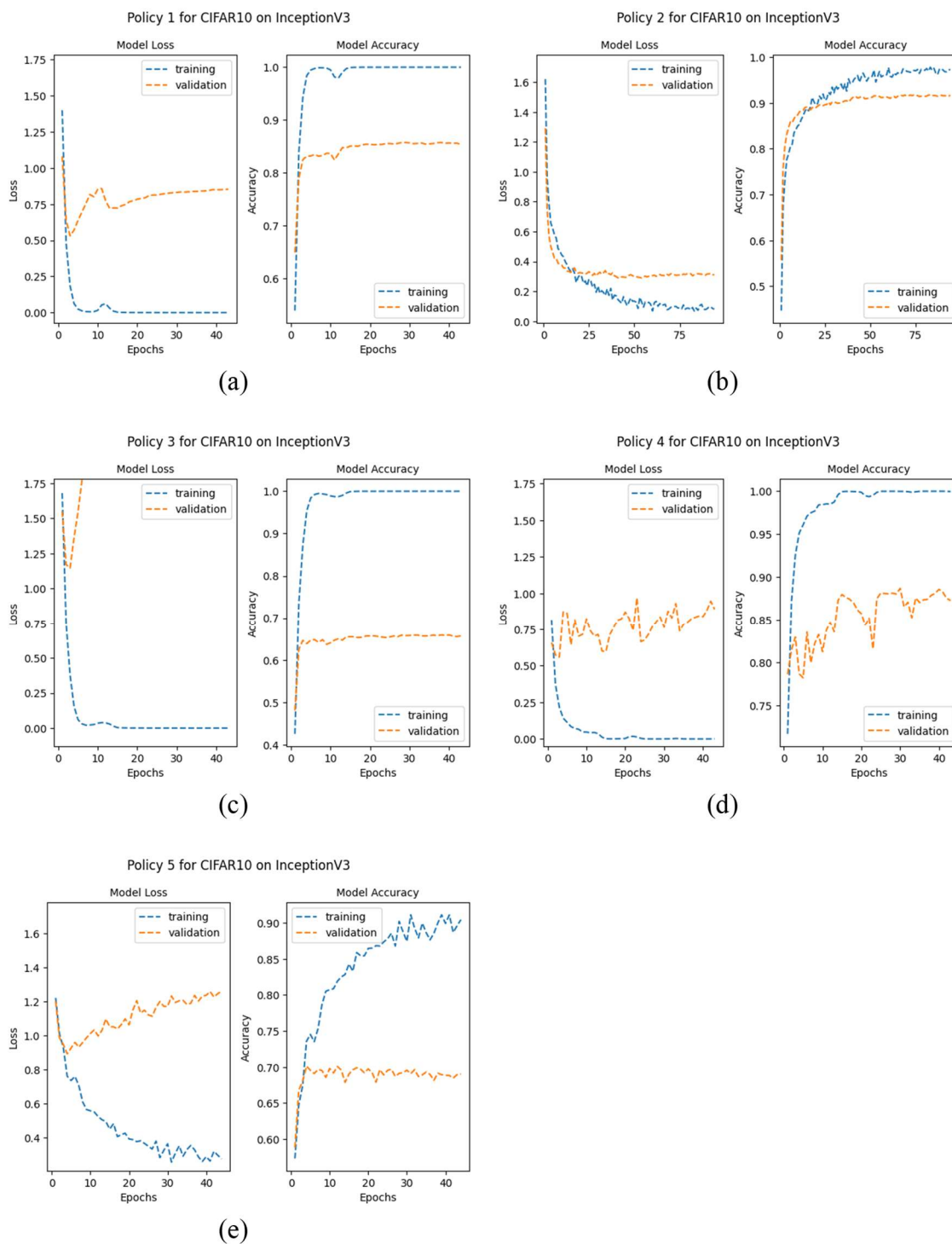
Note. The table displays the performance metrics for the test samples, where the delta column shows the percentage change in accuracy for each policy relative to Policy 1.

Higher percentages imply improved performance and generalizability.

Table 10 summarizes the performance of each policy for CIFAR-10 on InceptionV3, where Policy 1 established a baseline. Policy 2 and Policy 4 had accuracies of 91.34% and 82.92%, respectively, which translated to improvements of 11.44% and 1.17%, respectively, relative to the baseline. On the other hand, both policies 3 and 5 had accuracies less than the baseline, 64.83% and 69.54%, respectively, which translated to decreases of 20.9% and 15.15%, respectively. The veracity of the accuracy for each policy is further confirmed by the f1 score being, on average, within 0.22% of each policy's accuracy. From Table 10, it can be surmised that Policies 2 and 4 increased the spatial invariance of the model, while Policies 3 and 5 reduced the spatial invariance of the model. The former and latter were also observed for MNIST and FMNIST in Tables 5 and 6 for ResNet-50 and in Tables 2 and 3 for NoelNet.

Figure 40

Accuracy and Loss Graphs of each Policy for CIFAR-10 on InceptionV3



Policy 1

Figure 40(a) shows the loss and accuracy curves for Policy 1. The chart shows that the training and validation curves start diverging around epoch 5, such that the gap between the graphs quickly becomes more prominent, suggesting a quick increase in overfitting is occurring. The model reached convergence at epoch 43, when the validation loss reached its minimum and training was terminated. This convergence was equal to Policy 1 on FMNIST but less than Policy 1 on MNIST, suggesting that it is easier to train CIFAR-10 and FMNIST on InceptionV3 than to train MNIST. From Table 10, Policy 1 achieved an accuracy of 81.96% and an f1 score of 81.92%. This was a 11.7% improvement over the performance of CIFAR-10 on ResNet-50. Furthermore, the f1 score was competitive (i.e., within 0.05%) with accuracy, which suggested that the class distribution is balanced and that results produced by the model correctly reflected its performance, i.e., predictions made by the model should be at least 81% accurate.

Policy 2

Figure 40(b) shows the loss and accuracy graphs for Policy 2. The accuracy and loss graphs show a much closer correlation in relative trends than Policy 1. The graphs started diverging around epoch 25 instead of epoch 5, as with Policy 1. This implies the model was learning the increased invariances in the augmented data and thus was generalizing better on the unseen (i.e., the validation) data. The gap between each curve is also smaller than the gap seen in the curves for Policy 1. This suggests that the level of overfitting in Policy 2 is less than in Policy 1. The model had a slower convergence rate (~2x slower) compared to Policy 1 in that training terminated at Epoch 94 instead of Epoch 43 as it was for Policy 1. From Table 10, Policy 2 achieved an accuracy of 91.34%

and an f1 score of 91.35%. This accuracy represents an impressive 11.44% improvement over Policy 1 and the unperturbed dataset, a significant improvement in the spatial invariance of the model by using RandAugment.

Policy 3

Figure 40(c) shows the loss and accuracy graphs for Policy 3. The graphs can be seen to have much wider gaps as compared to policies 1 and 2, and they also diverged very early in the training process, around epoch 5, suggesting that the synthetic samples caused much higher levels of overfitting. The model converged at epoch 43, the same as Policy 1 and much quicker than Policy 2. This suggests Policy 3 imbued the dataset with smaller invariances than Policy 2. From Table 10, Policy 3 achieved an accuracy of 64.83% and an f1 score of 64.41%, representing an approximately 21% decrease from Policy 1. Also, the slightly smaller f1 score indicates slight imbalances in the class distributions. The decrease in accuracy relative to Policy 1 implies that Policy 3 led to a model with decreased spatial invariance and generalizability.

Policy 4

Figure 40(d) shows the loss and accuracy graphs for Policy 4. The gaps in the loss graphs are slightly wider than Policy 1, much wider than Policy 2, and less wide than Policy 3, indicating that Policy 4 may lead to more overfitting than Policies 1 and 2 but less overfitting than Policy 3. The model converged at epoch 43, which was much faster than that of Policy 2 and equal to Policies 1 and 3. From Table 10, Policy 4 achieved an accuracy of 82.92% and an f1 score of 82.90%, representing a 1.17% improvement over Policy 1. In a similar fashion to Policy 4 for F/MNIST on InceptionV3, it is observed that

combining the synthesized samples from Policy 3 with their baseline counterpart led to improved spatial invariance in the trained model.

Policy 5

Figure 40(e) shows the loss and accuracy graphs for Policy 5. The gap between the curves is wider than Policies 1, 2, and 4 and less wide than Policy 3, which suggests less overfitting when compared to Policy 3 and that applying RandAugment to synthesized samples can improve spatial invariance compared to using the synthesized samples in isolation. This observation is corroborated by Table 10, where Policy 5 has an accuracy of 69.54% and an f1 score of 69.81%, while Policy 3 has an accuracy of 64.83%. Although the accuracy of Policy 5 is 15.15% lower than the baseline, thus not increasing the model's spatial invariance, it is still 5.75% higher than Policy 3. The model convergence of Policy 5, at 44 epochs, was slower than Policies 1, 3, and 4 but less than Policy 2, which implies that RandAugment introduced more sources of invariances than Policies 1, 3, and 4, thus taking the model longer to converge.

Figure 41

Loss and Validation Accuracy of All Policies for CIFAR-10 on InceptionV3

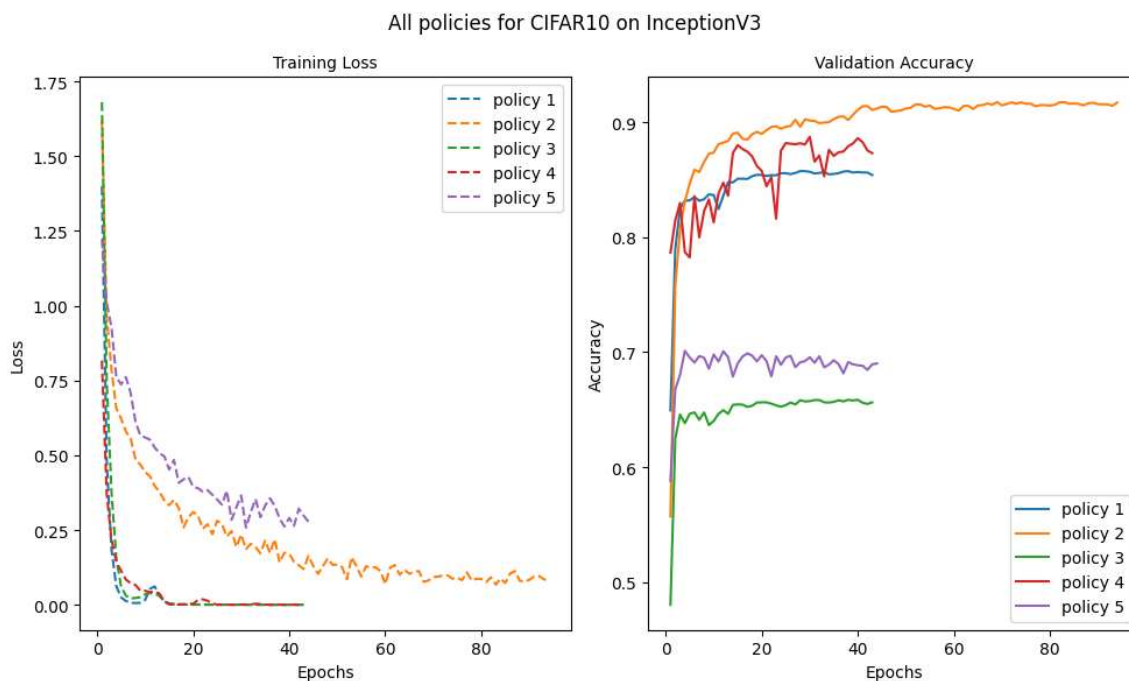
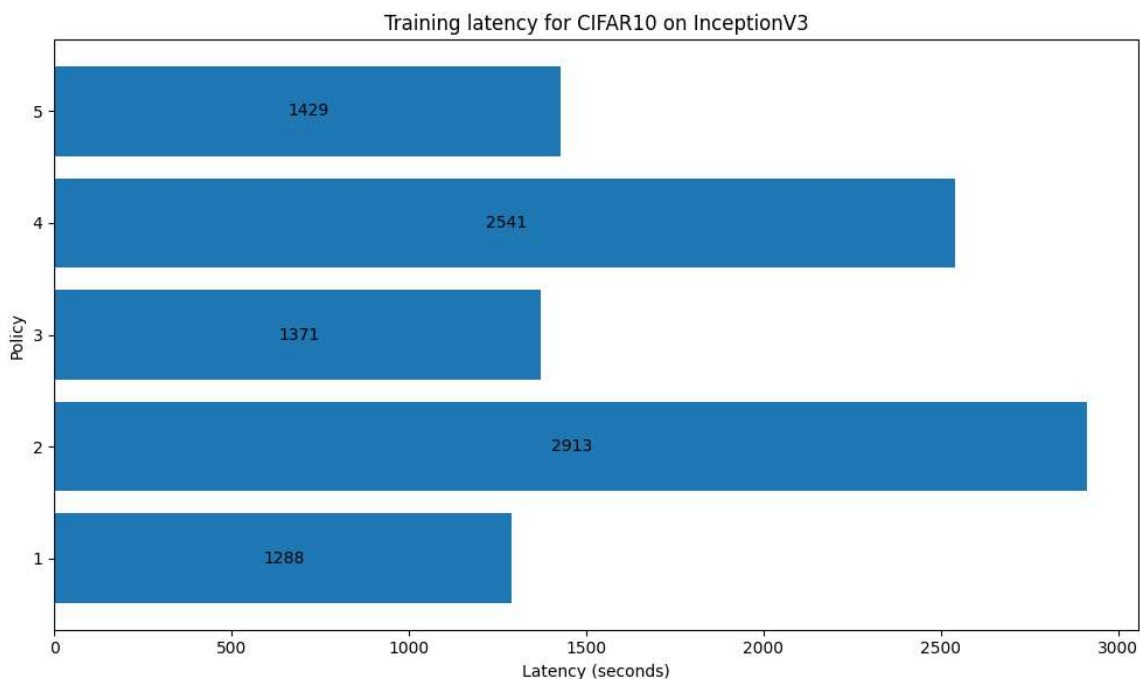


Figure 41 summarizes the training loss and validation accuracy of all policies for CIFAR-10 on InceptionV3. Policy 1 established the baseline metrics and had a relatively fast convergence. Policy 2 had the longest convergence time at 94 epochs, followed by Policy 5, which converged at epoch 44. This can be attributed to the stochasticity of RandAugment used by those two policies. Policy 2 had the highest accuracy, followed by Policy 4. Policy 3 converged at the same epoch as policies 1 and 4 but had the lowest accuracy. The loss curves for policies 2 and 5, the two policies implementing RandAugment, fit the profile of the smooth polynomial curves seen in models with minimal overfitting like Figures 7 and 8. This suggests that RandAugment is a very effective method of reducing overfitting in ConvNets.

Figure 42

Training Latency of All Policies for CIFAR-10 on InceptionV3



The training latencies of all policies for CIFAR-10 on InceptionV3 are seen in Figure 42. Policy 2 had the longest training time, taking 2913 seconds, more than twice as long as policies 1 and 3 and approximately 2x that of Policy 5. The latter was unexpected since policies 2 and 5 used RandAugment and expected to have similar training times. The last observation may be because the synthesized samples used in Policy 5 are poorer in quality and complexity than the samples used by Policy 2, and augmenting them using RandAugment did not increase their invariance to the levels seen in Policy 2. Policy 1 had the shortest training time, followed by Policy 3, mainly because they are not augmented or combined with any other dataset. Policy 4 had the second longest training time because it combines the unaugmented and synthesized samples. In

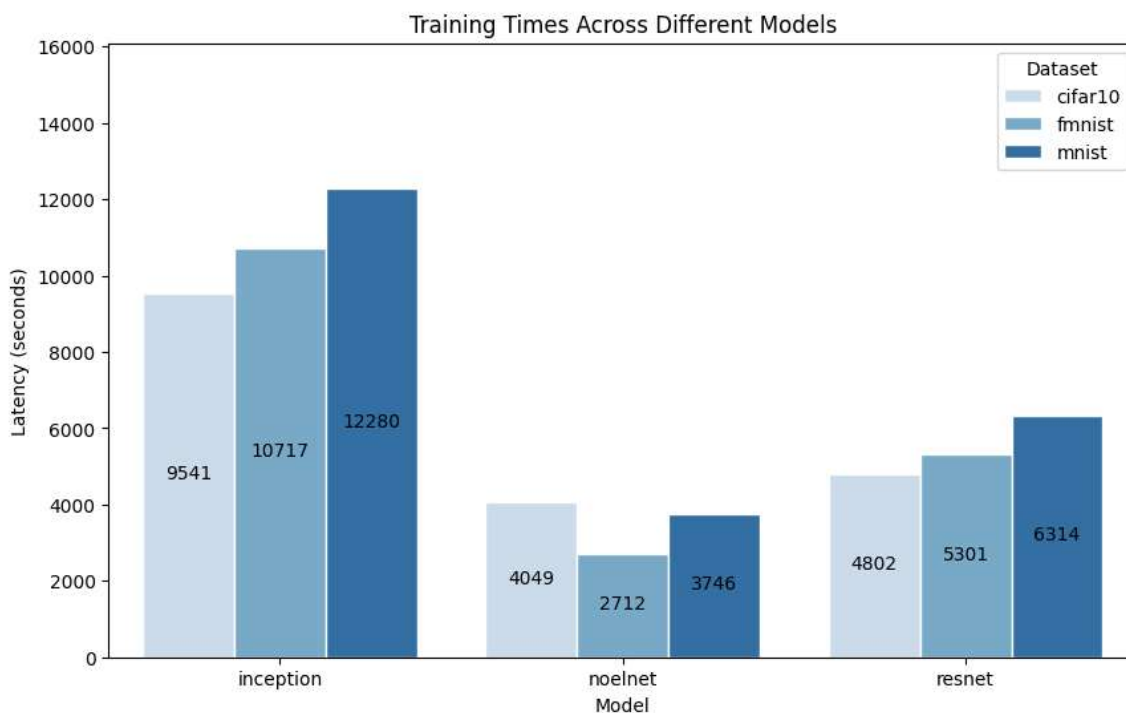
comparison, the training latency for CIFAR-10 was between 2 to 2.5 times faster on NoelNet than on InceptionV3.

In summary, for CIFAR-10 on InceptionV3, it was observed that Policies 2 and 4 had the highest accuracies and were higher than Policy 1, implying that these two policies increased the spatial invariance of the model. On the other hand, policy 3 had the lowest accuracy, followed by Policy 5, with accuracies less than Policy 1. The latter suggests that policies 3 and 5 did not improve the spatial invariance of the model. However, the precision for Policy 3 was slightly higher than the recall and accuracy. This implies that the accuracy for Policy 3 may not reflect the model's "true" performance or that the class distribution for Policy 3 was not evenly distributed. These same patterns were observed in Table 4 for CIFAR-10 on NoelNet.

Dataset Latency Across Different Models

Figure 43

Comparing Dataset Training Times Across Different Models



Note. The total time shown on each bar of the chart represents the sum of the latency for each policy using the dataset on that model. For example, cifar10 on inception took 9,541 seconds; this final value is the sum of the individual times taken by each policy using CIFAR-10 on InceptionV3.

Figure 43 shows the training times of each dataset across the different models, and it is observed that NoelNet, when compared to InceptionV3, was approximately twice as fast in training CIFAR-10, four times as fast in training FMNIST, and three times as fast in training MNIST. NoelNet was approximately 19% faster in training CIFAR-10 and twice as fast in training F/MNIST when compared to ResNet-50. The training time on ResNet-50 is approximately two times faster than on InceptionV3, which

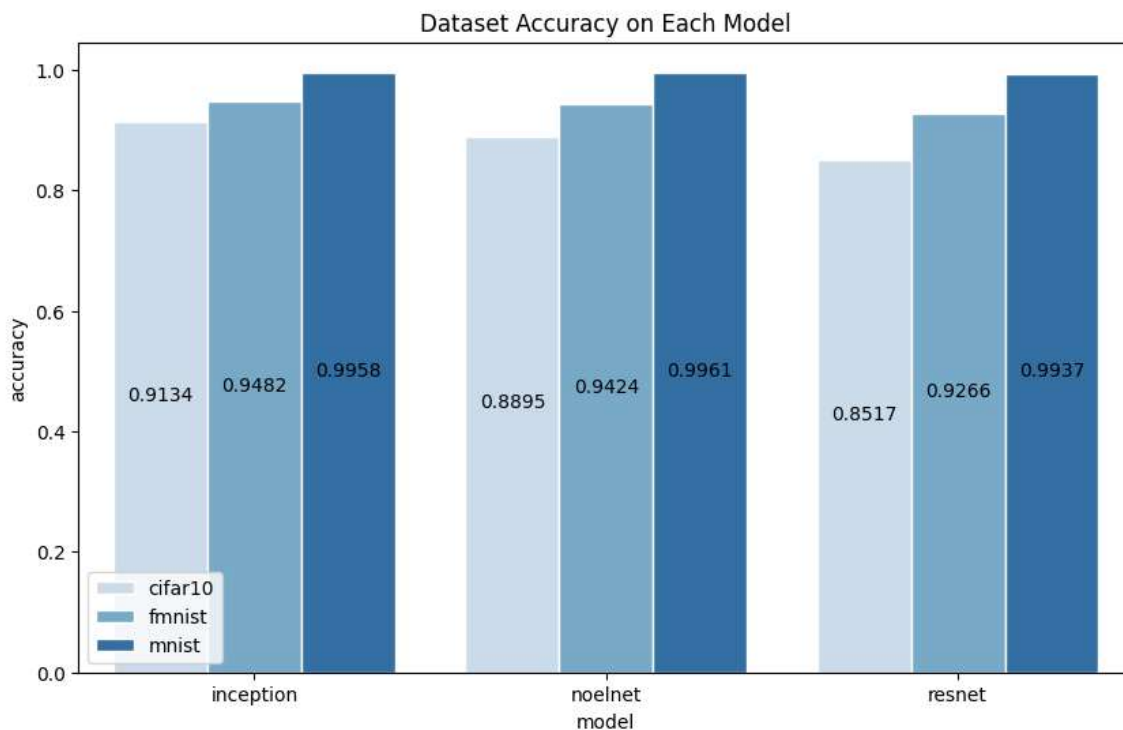
is observed for each dataset. On ResNet-50, the MNIST dataset took 6,314 seconds, while InceptionV3 took 12,280 seconds. Similarly, FMNIST on ResNet-50 took 5,301 seconds, while InceptionV3 took 10,717 seconds. Finally, CIFAR-10 took 4802 seconds on ResNet-50 and 9541 seconds on InceptionV3. The total training time on NoelNet was 10,507 seconds (2.9183 hours). On InceptionV3, it was 32,538 seconds (9.0383 hours); on ResNet-50, it was 16,417 seconds (4.560 hours).

Training NoelNet is roughly 3 times faster than training InceptionV3 and 1.5 times faster than training ResNet-50. Furthermore, it takes twice as long to train InceptionV3 than ResNet-50. The additional time needed to train InceptionV3 is because it simultaneously uses kernels of three different sizes (see Figure 11) to learn or extract diverse features within an image. The combined time spent training the three models was 59,462 seconds (*or* 16.517 hours) of continuous training time. *It must be noted that the author conducted at least twelve weeks of prior training to find the most optimal balance of hyperparameters used in the final models.*

Dataset Accuracy Across Different Models

Figure 44

Highest Dataset Accuracy Across Different Models



Note. The values shown on each bar of the chart represent the accuracy achieved by Policy 2, which was the most performant Policy, having consistently achieved the highest accuracies and f1 scores for all experiments.

Figure 44 shows the accuracies achieved by Policy 2, which consistently produced the highest accuracies across all models and datasets. Concerning datasets, MNIST had the highest accuracy, followed by FMNIST, then CIFAR-10. MNIST had an accuracy of 99.58% on InceptionV3, 99.61% on NoelNet, and 99.37% on ResNet-50. FMNIST had an accuracy of 94.82% on InceptionV3, 94.24% on NoelNet, and 92.66% on ResNet-50. Finally, CIFAR-10 achieved 91.34% on InceptionV3, 88.95% on NoelNet, and 85.17% on ResNet-50.

Several bodies of work support the distribution of accuracies seen in Figure 44 in that MNIST is one of the less challenging datasets to train, and most models trained on MNIST achieved very high accuracies. FMNIST has the same structure as MNSIT (28x28 grayscale images) but is a slightly more complex dataset than MNIST. In contrast, CIFAR-10 is the most complicated dataset of the three since its structure is 32x32 color images.

From the chart, InceptionV3 resulted in the highest accuracy across the three datasets, closely followed by NoelNet, then by ResNet-50. Policy 2 achieving the highest accuracies and f1 scores relative to all Policies, including Policy 1 (the unaugmented data policy), means that it has consistently improved the spatial invariance of each model. Notably, NoelNet, which is a basic CNN (~1 million parameters), consistently achieved higher accuracies than ResNet-50 (~23.6 million parameters) and competitive accuracies with InceptionV3 (~21.8 million parameters).

Data Analysis

This chapter addressed the following questions posed in Chapter 3 based on the experimental results obtained and presented.

1. Does data augmentation lead to more spatially invariant and robust networks?

Of the five Policies used in this research, four Policies: 2, 3, 4, and 5 implemented a form of data augmentation. Policies leading to test accuracies and f1 scores higher than those achieved by Policy 1 (baseline) are considered to have increased the spatial invariance of the network. First, Policies 2 and 4 have consistently achieved higher accuracy and f1 scores than Policy 1 on a basic model, NoelNet, as seen in Tables 2, 3, and 4. Similar patterns were observed on ResNet-50 (Tables 5, 6, and 7) and InceptionV3 (Tables 8, 9, and 10) for all three datasets. For instance, from the results of CIFAR-10 on ResNet-50 shown in Table 7, Policy 2 achieved a 16.05% improvement over the baseline, while Policy 4 achieved a 4.52% improvement over the baseline. The results from Policies 2 and 4 on each model indicate that data augmentation does lead to more spatially invariant networks.

Secondly, Figure 44 shows that NoelNet has consistently achieved higher accuracies than ResNet-50 and competitive accuracies with InceptionV3 across all datasets, even though it is a basic CNN (~1 million parameters) without optimizations. ResNet-50 has over 23 million parameters, InceptionV3 has over 21 million, and both are state-of-the-art ConvNets with novel and innovative architecture. NoelNet being competitive with the more advanced models occurs because augmentation imbues invariances within the training data, which models with simple or sophisticated architectures can then learn. This shows that advanced augmentation techniques are more effective in improving the

robustness and spatial invariance of ConvNets as opposed to architectural modifications, as stated by one of the goals of this research.

Thirdly, the loss and accuracy curves of Policies 2 and 4 for each experiment consistently yielded narrower gaps than those of Policy 1, as shown in Figures 14, 18, 22, 25, 28, 31, 34, 37, and 40, suggesting less overfitting of the models and increases in their generalizability. Furthermore, the accuracies achieved have been verified as “true” reflections of the model’s performance by the f1 scores being competitive (i.e., within 0.2% in most cases) with the measured accuracies. As such, data augmentation does lead to more spatial invariance and robust networks.

2. Are deep architectures more spatially invariant than wide architectures? Did a particular architecture perform better?

NoelNet (Figure E6) and ResNet-50 (Figure 10) are considered deep models because they consist of two or more sequentially stacked layers. In contrast, InceptionV3 is considered a wide model because of its many inception modules (Figure 11). Nevertheless, throughout the experiments, each policy implemented on InceptionV3 consistently had slightly higher accuracies (Figure 44), precision, and recall than ResNet-50 and NoelNet, especially on more complicated datasets like CIFAR-10.

However, training ResNet-50 and NoelNet was much faster than training InceptionV3, as seen in Figure 43. Therefore, the wider model had higher accuracies and f1 scores and was more spatially invariant than the deeper models, while the deeper models were at least twice as fast to train.

3. Does generative data augmentation lead to more diverse samples than reinforcement learning techniques? Policy 2 vs. Policy 3.

Policy 2 consistently resulted in higher accuracies and f1 scores throughout the experiments than Policy 3. Policy 2 had the highest accuracy of each policy (Figure 44), while Policy 3 consistently had the lowest accuracy and f1 scores. Policy 3 implemented and used only cDCGAN synthesized samples, which, when used independently, resulted in the lowest accuracies, precisions, and recalls across both models.

However, when RandAugment was applied to the synthesized samples (Policy 5), each model's accuracy and f1 scores increased notably. Therefore, the samples produced by the reinforcement learning technique (RandAugment) led to more diverse samples than those produced by the generative technique (GAN).

4. Do larger datasets increase spatial invariance? (Policy 2, 4, 5 vs. Policy 3)

Policy 2 and 4 resulted in higher accuracies and f1 scores relative to Policy 1 (the baseline), while Policy 3 and 5 resulted in scores lesser than the baseline. Policy 2 and 5 artificially increased the datasets by augmenting each sample using RandAugment online, while Policy 4 physically increased the dataset by uniformly combining the unaugmented samples with the GAN-generated samples. Policy 5 did not increase the accuracy or f1 score of the models, thus their spatial invariance, primarily because the synthesized images did not possess more significant amounts of invariances when compared to the unaugmented samples. However, policies 2 and 4 consistently improved the measured metrics of the models; thus, it can be inferred that larger datasets do increase spatial invariance in their trained models.

5. Do stacking data augmentation methods lead to more robust networks?

Policy 5 applied RandAugment to the GAN-synthesized samples from Policy 3. As seen in Tables 2 through 10, the accuracy and f1 scores of Policy 5 consistently rank

higher than those of Policy 3. Therefore, according to the results, stacking data augmentation methods increases models' accuracy and f1 score, leading to more robust networks.

6. *Does combining synthesized samples with original unaugmented samples improve invariance in CNNs? Policy 4*

Policy 4 has consistently achieved higher accuracies and f1 scores than Policy 1, the baseline policy, across all datasets and models, as shown by Tables 2 through 10. In addition, the gaps between its loss and accuracy curves for each model have also mainly been smaller than those for Policy 1. These increases in accuracy and f1 scores were due to the increased invariance of the GAN samples added to the unaugmented dataset.

Independently, the GAN samples needed more invariances to improve the model's spatial invariance, but when combined with the unperturbed samples, it created a more diverse dataset. As such, combining the synthesized samples with the original unaugmented samples improved the invariance of the models.

Furthermore, a reasonable question posed by the author was how does one know that an increase in accuracy and f1 scores are not just because of additional samples? Because the loss/accuracy graphs should have a smaller gap compared to Policy 1, indicating overfitting improved. On the other hand, additional samples that brought no value (i.e., had no additional sources of invariance) would have caused wider gaps between the curves, thus increasing the levels of overfitting.

7. *Do synthesized samples improve invariance? Policy 3*

Tables 2 through 10 show that Policy 3 consistently had the lowest accuracies of each policy and never resulted in accuracies, precisions, or recalls higher than the baseline

(Policy 1). Furthermore, the loss and accuracy graphs for Policy 3, across each dataset and model, had the widest gaps compared to the other policies. The former and latter implies that the synthesized samples led to significant overfitting, thus not improving the invariance of the models. Independently, the GAN samples did not improve the model's spatial invariance. However, combined with the unperturbed samples (as done by Policy 4), it improved the spatial invariance of the models because it created a more diverse dataset that led to accuracy and f1 scores higher than the baseline.

8. Do stochastic techniques based on Reinforcement learning improve invariance?

RandAugment is a stochastic technique based on reinforcement learning and was implemented by Policies 2 and 5. Policy 2 has consistently achieved the highest accuracy and f1 scores across each model and dataset, in addition to having the narrowest gaps in its loss and accuracy curves, thus leading to improved invariance of the models. Refer to questions (3) and (4) above for a complete analysis.

9. Does the number of parameters in a CNN affect its spatial invariance as measured by its test accuracy?

InceptionV3 has 21,823,274 parameters, NoelNet has 1,091,594, and ResNet-50 has 23,585,290. From Figure 44, the maximum accuracy on InceptionV3 was 99.58%; on NoelNet, it was 99.61%; and on ResNet-50, it was 99.37%. These accuracies were achieved using MNIST, but similar patterns were seen when using FMNIST or CIFAR-10. Interestingly, InceptionV3 and NoelNet had higher accuracy and f1 scores for all three datasets than ResNet-50, even though ResNet-50 has the highest number of parameters. Furthermore, NoelNet, which has the lowest number of parameters, i.e., ~23 times smaller than ResNet-50, and ~21 times smaller than InceptionV3, achieved very

competitive scores with InceptionV3. The results show that having more parameters does not necessarily indicate better predictive abilities or increased spatial invariances, as intuition may suggest.

Summary

This chapter described and presented the results of a series of 45 experiments that were conducted using six datasets, three ConvNets (NoelNet, ResNet50, and InceptionV3), and five augmentation policies. NoelNet was a basic model developed by the author and used to establish baseline metrics against which the results from ResNet50 and InceptionV3 would be analyzed for similar patterns. The results of the experiments were presented in tabular and graphical formats. They were used for analyzing improvements in spatial invariance in ConvNets and for hyperparameter optimization during training of the conditional DCGAN, ResNet50, and InceptionV3. Parameter tuning was required to avoid issues such as gradient explosion and mode collapse, which were minimized by training ResNet50 and InceptionV3 at least 20 times each and DCGAN at least 40 times. The final set of hyperparameters used for all experiments is provided in Appendix B, "Hyperparameters." The author's GitHub account, described in further detail in Appendix G, "Dissertation Source Code," contains the complete source code and raw data for all experiments; the raw data was saved in CSV format and included performance metrics and training history. A more detailed description of the data points captured and the results format is provided in Appendix F, "Experimental Results."

Five policies were used to analyze spatial invariance on ConvNets. Policy 1 established a baseline for the metrics by training the model on the unperturbed dataset.

Policy 2 analyzed the effect on invariance by augmenting the dataset using RandAugment. Policy 3 used synthesized samples of the dataset using a conditional DCGAN and used these samples to train the model. Policy 4 combined the cDCGAN synthesized samples with the unaugmented samples, and finally, Policy 5 combined different augmentation techniques by applying RandAugment to the synthesized samples.

The conditional DCGAN, seen in Figure E1, synthesized three of the six datasets used for the experimentations. The GAN model was trained 30 times on CIFAR-10 and 10 times on F/MNIST for 200 epochs to select the most optimal hyperparameters. These hyperparameters, detailed in Appendix B, were then used to train the final GAN model for 1000 epochs. The generated images improved in quality from epoch 1 to epoch 1000, and stabilization was achieved after approximately 500 epochs of training. This progression of image quality was significant because it demonstrated that the GAN was not simply memorizing the images but learning their salient features to synthesize additional samples. The generated images for F/MNIST were of higher quality than those for CIFAR-10 due to the simpler features of the former. From Table 1, training the cDCGAN took approximately 29.22 hours; MNIST took 9.02 hours to train, FMNIST took 8.66 hours, and CIFAR-10 took 11.53 hours. Samples of the synthesized images can be seen in Figures D2 and D3.

Figure 13 shows samples of the MNIST dataset used to train NoelNet. As shown in Table 2, the training resulted in Policy 1 having an accuracy of 99.43%, Policy 2 achieving 99.61%, Policy 3 achieving 98.94%, Policy 4 achieving 99.44%, and Policy 5 achieving 99.13%. The f1 scores confirmed the veracity of the accuracies by being, on average, within 0.002% of each policy's accuracy. Policy 2 achieved the highest accuracy

and f1 scores among all the policies, followed by Policy 4. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 14, which indicated it resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that Policies 2 and 4 improved the spatial invariance of the model, while Policies 3 and 5 reduced the model's spatial invariance. Finally, Figure 16 shows that training the five policies for MNIST on NoelNet took 3,746 seconds, with Policy 4 having the longest train time of 955 seconds and Policy 1 having the shortest train time of 350 seconds.

Figure 17 shows samples of the FMNIST dataset used to train NoelNet. Table 3 shows that the training resulted in Policy 1 having an accuracy of 92.88%, Policy 2 achieving 94.24%, Policy 3 achieving 86.34%, Policy 4 achieving 93.07%, and Policy 5 achieving 87.24%. The f1 scores confirmed the veracity of the accuracies by being, on average, within 0.087% of each policy's accuracy. Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 18, which indicated it resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that policies 2 and 4 improved the spatial invariance of the model, while policies 3 and 5 reduced the model's spatial invariance. Finally, Figure 20 shows that training the five policies for FMNIST on NoelNet took 2,712 seconds, with Policy 2 having the longest train time of 979 seconds and Policy 3 having the shortest train time of 252 seconds.

Figure 21 shows samples of the CIFAR-10 dataset used to train NoelNet. Table 4 shows that the training resulted in Policy 1 having an accuracy of 85.16%, Policy 2 achieving 88.95%, Policy 3 achieving 57.83%, Policy 4 achieving 83.64%, and Policy 5 achieving 68.59%. Policy 2 achieved the highest accuracy and f1 scores among all the policies; however, unlike F/MNIST on NoelNet, Policy 4 decreased by 1.78%. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 22, which indicated it resulted in the lowest overfitting and highest generalizability. Policies 3, 4, and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that Policy 2 improved the spatial invariance of the model, while Policies 3, 4, and 5 reduced the model's spatial invariance. Finally, Figure 24 shows that training the five policies for CIFAR-10 on NoelNet took 4,050 seconds, with Policy 4 having the longest train time of 1188 seconds and Policy 3 having the shortest train time of 427 seconds.

Table 5 shows the results of training MNIST dataset on ResNet-50, where Policy 1 achieved an accuracy of 99.14%, Policy 2 achieved 99.37%, Policy 3 achieved 98.74%, Policy 4 achieved 99.30%, and Policy 5 achieved 98.94%. The f1 scores confirmed the veracity of the accuracies by being, on average, within 0.004% of each policy's accuracy. Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 25, which indicated it resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that policies 2 and 4 improved the spatial invariance of the model, while policies 3 and 5 reduced the model's spatial

invariance. Finally, Figure 27 shows that training the five policies for MNIST on ResNet-50 took 6,314 seconds, with Policy 4 having the longest train time of 1731 seconds and Policy 3 having the shortest train time of 852 seconds.

Table 6 shows the results of training FMNIST dataset on ResNet-50, where Policy 1 achieved an accuracy of 89.85%, Policy 2 achieved 92.66%, Policy 3 achieved 82.67%, Policy 4 achieved 91.59%, and Policy 5 achieved 85%. The f1 scores confirmed the veracity of the accuracies by being, on average, within 0.24% of each policy's accuracy. Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 28, which indicated it resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that policies 2 and 4 improved the spatial invariance of the model, while policies 3 and 5 reduced the model's spatial invariance. Finally, Figure 30 shows that training the five policies for FMNIST on ResNet-50 took 5,301 seconds, with Policy 4 having the longest train time of 1587 seconds and Policy 3 having the shortest train time of 691 seconds.

Table 7 shows the results of training CIFAR-10 on ResNet-50, where Policy 1 achieved an accuracy of 73.39%, Policy 2 achieved 85.17%, Policy 3 achieved 62.86%, Policy 4 achieved 76.71%, and Policy 5 achieved 69.15%. The f1 scores confirmed the veracity of the accuracies by being, on average, within 0.25% of each policy's accuracy. Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 31, which indicated it resulted in the lowest overfitting and highest

generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that policies 2 and 4 improved the spatial invariance of the model, while policies 3 and 5 reduced the model's spatial invariance. Finally, Figure 26 shows that training the five policies for CIFAR-10 on ResNet-50 took 4,802 seconds, with Policy 2 having the longest train time of 1416 seconds and Policy 3 having the shortest train time of 587 seconds.

Table 8 shows the results of training MNIST dataset on InceptionV3, where Policy 1 had an accuracy of 99.43%, Policy 2 achieved 99.58%, Policy 3 achieved 98.92%, Policy 4 achieved 99.56%, and Policy 5 achieved 99.11%. The f1 scores confirmed the veracity of the accuracies by being equal to each policy's accuracy. Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 34, which indicated it resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that policies 2 and 4 improved the spatial invariance of the model, while policies 3 and 5 reduced the model's spatial invariance. Finally, Figure 36 shows that training the five policies for MNIST on InceptionV3 took 12,280 seconds, with Policy 4 having the longest train time of 3422 seconds and Policy 1 having the shortest train time of 1781 seconds.

Table 9 shows the results of training FMNIST dataset on InceptionV3, where Policy 1 had an accuracy of 91.67%, Policy 2 achieved 94.82%, Policy 3 achieved 85.27%, Policy 4 achieved 93.53%, and Policy 5 achieved 88.25%. The f1 scores confirmed the veracity of the accuracies by being, on average, within 0.06% of each

policy's accuracy. Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 37, which indicated it resulted in the lowest overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that policies 2 and 4 improved the spatial invariance of the model, while policies 3 and 5 reduced the model's spatial invariance. Finally, Figure 39 shows that training the five policies for FMNIST on InceptionV3 took 10,717 seconds, with Policy 4 having the longest train time of 3217 seconds and Policy 3 having the shortest train time of 1543 seconds.

Table 10 shows the results of training the CIFAR-10 dataset on InceptionV3, where Policy 1 had an accuracy of 81.96%, Policy 2 achieved 91.34%, Policy 3 achieved 64.83%, Policy 4 achieved 82.92%, Policy 5 achieved 69.54%. The f1 scores confirmed the veracity of the accuracies by being, on average, within 0.22% of each policy's accuracy. Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also had the smallest gaps between its loss and accuracy curves, as shown in Figure 40, which indicated it resulted in the lowest overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, with Policy 3 having the lowest scores. This implies that policies 2 and 4 improved the spatial invariance of the model, while policies 3 and 5 reduced the model's spatial invariance. Finally, Figure 42 shows that training the five policies for CIFAR-10 on InceptionV3 took 9,542 seconds, with Policy 2 having the longest train time of 2913 seconds and Policy 1 having the shortest train time of 1288 seconds.

Figure 43 shows that the total training time on InceptionV3 was 9.0383 hours; NoelNet was 2.9183 hours; ResNet-50 was 4.560 hours, meaning NoelNet was three times as fast to train than InceptionV3 while ResNet-50 was twice as fast to train than InceptionV3. This increased training time has led to InceptionV3 achieving the highest accuracies across all datasets used by the policies, as shown in Figure 44.

In summary, Policies 2 and 4 improved the spatial invariance of the models, while Policies 3 and 5 did not improve the spatial invariance of the models. The latter was unexpected and differed from the expectations summarized in Table F1 before experimentation. Therefore, Table F2 is a revised version of Table F1 with the updated results after experimentation.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

The results of this dissertation demonstrated that advanced data augmentation does lead to more spatially invariant and robust networks. This was directly observed by Policies 2 and 4 consistently achieving higher accuracies, f1 scores, and less overfitting compared to Policy 1, the baseline, across one basic and two advanced ConvNets. Regarding overfitting, Policy 2 consistently had much narrower gaps in its loss and accuracy curves compared to all other policies, which suggested minimal overfitting and improved generalizability of the models. These results proved that RandAugment, which is applied by Policy 2, is a highly effective advanced data augmentation technique for improving the spatial invariance of ConvNets. Policy 5 further corroborated the effectiveness of RandAugment by applying it to the synthesized cDCGAN samples. The resulting accuracies and f1 scores were consistently higher and with less overfitting than those of Policy 3, which used the synthesized samples independently.

Policies 3 and 5 decreased the spatial invariance of the models for each experiment, with Policy 3 consistently having the lowest accuracies, f1 scores, and widest gaps in its loss-accuracy curves. This implied that the cDCGAN synthesized samples, when trained *independently* of other samples and augmentations, did not possess sufficient diversity and invariances to improve the spatial invariance of the models. However, when the synthesized samples were *combined* with the unaugmented samples, as demonstrated by Policy 4, the spatial invariance of the resulting model consistently

increased, as seen by increases in the accuracies, f1 scores, and narrower gaps in the loss-accuracy curves compared to Policy 1.

Regarding architecture, NoelNet has consistently achieved higher accuracies than ResNet-50 and competitive accuracies with InceptionV3 across all datasets, even though it is a basic CNN (~1 million parameters). ResNet-50 has over 23 million parameters, InceptionV3 has over 21 million, and both are state-of-the-art ConvNets. This shows that advanced augmentation techniques are more effective in improving the robustness and spatial invariance of ConvNets as opposed to architectural modifications, as stated by the goals of this research.

The study also compared the spatial invariance of deep and wide models and found that the wider model (InceptionV3) frequently had higher accuracies and, thus, was more spatially invariant than the deeper models (NoelNet, and ResNet-50). However, ResNet-50 was at least twice as fast to train as InceptionV3, while NoelNet was at least three times faster, a significant consideration for resource or time-constrained environments and use cases. The considerable increase in training time on InceptionV3 was because it uses kernels of three different sizes in parallel (Inception module) to extract salient and diverse features within an image. In contrast, the deeper models are absent of parallel computations.

Additionally, the study compared the effectiveness of generative data augmentation and reinforcement learning techniques and found that the samples produced by reinforcement learning techniques were more diverse than those produced by generative techniques. Regarding the effect of an increased number of parameters on

spatial invariance, it was found that a higher number of parameters do not indicate higher accuracies or spatial invariance.

Finally, the study examined the impact of larger datasets by artificially or physically oversampling the training set and found that the larger datasets consistently led to more spatially invariant networks. However, physically increasing the dataset size, as opposed to artificial means, was found to have longer training latencies.

Implications

Spatial invariance is an essential property of deep convolutional neural networks (CNNs) that allows them to recognize objects or patterns regardless of their location or orientation within an image. Not improving spatial invariance in deep CNNs can have several implications, including:

1. **Poor performance on new images:** Without spatial invariance, deep CNNs may not be able to recognize objects in new images that are slightly rotated, translated, or scaled from the training images. This can result in poor performance on real-world datasets, where images can vary significantly in spatial properties.
2. **Large model size:** Deep ConvNets typically use pooling layers to achieve spatial invariance, reducing the feature maps' spatial resolution. If spatial invariance is not improved, larger ConvNets with more parameters may be needed to achieve the same level of performance. This can increase the memory and computational requirements of the model, making it harder to deploy on resource-constrained devices.
3. **Overfitting:** Without spatial invariance, deep CNNs may become too sensitive to small variations in the input images, leading to overfitting. This can occur if the

model learns to memorize the spatial position of the features in the training data rather than learning to recognize the features themselves.

4. Limited interpretability: A lack of spatial invariance can also make interpreting the features learned by deep CNNs harder. The location of the features in an input image can be important for their interpretation. Without spatial invariance, it may be harder to understand how different features contribute to the final decision of the model.

In summary, improving spatial invariance is crucial for achieving optimal performance, smaller model sizes, better generalization, and improved interpretability in deep CNNs.

Recommendations

Improving spatial invariance in ConvNets is essential to evolve their robustness and generalizability, and the results of this dissertation spurred additional areas of research to continue in this pursuit. The following represents some potential areas of research (in no specific order):

- To synthesize additional training samples, a different GAN, such as a Wasserstein GAN (WGAN; Arjovsky et al., 2017), can be implemented instead of a Deep Convolutional GAN (DCGAN; Radford et al., 2015). In this study, Policy 3 used a cDCGAN to generate training samples. However, this policy consistently achieved accuracies and f1 scores lower than the baseline dataset (Policy 1). Furthermore, while impressive, the samples generated for CIFAR-10 were still not competitive (via visual inspection) with the quality of the samples from the training dataset. The output of DCGANs tends to be hindered by the problem of

vanishing gradients and mode collapse primarily because they use a binary cross-entropy loss function. The WGAN uses a Wasserstein loss to prevent vanishing gradients and mode collapse (Arjovsky et al., 2017), leading to higher-quality generated outputs.

- Implement different generative augmentation techniques, such as Neural Style Transfer (Gatys et al., 2015; Perez & Wang, 2017), which take the style of one sample and apply it to another without degrading the quality of the affected sample.
- Perform studies using contemporary alternatives to RandAugment, such as Cutout (DeVries & Taylor, 2017), CutMix (Yun et al., 2019), and GridMask (Chen et al., 2020), and on datasets like ImageNet with samples of more respectable image sizes.

Summary

Introduction

ConvNets have achieved great success on many visual tasks, such as image classification. They are assumed to be spatially invariant because of their unique architectures. However, several authors have shown that contemporary ConvNets are not spatially invariant to small perturbations in their input images. This lack of robustness significantly reduces their predictive abilities, which could lead to fatalities, racial discrimination, or other unexpected outcomes. Data augmentation is one of two standard solutions for teaching spatial invariance to CNNs. Data augmentation artificially increases the training dataset by generating new and diverse data points extracted from existing training data, and in so doing, it directly imbues sources of invariances within the

training dataset. While simple augmentation techniques are effective, they require domain expertise, must be hand-engineered, and do not capture enough of the axes of variations within the training data.

In contrast, advanced augmentation techniques that use deep learning, such as RandAugment and Generative Adversarial Networks, learn invariances from data by sampling from a continuous distribution space, leading to more diverse and varied training samples. This dissertation aimed to provide a comparative study on using advanced augmentation techniques such as RandAugment and GANs to improve the spatial invariance in CNNs. These techniques were applied to the MNIST, FMNIST, and CIFAR-10 benchmark datasets via five augmentation strategies and evaluated on NoelNet, ResNet50, and InceptionV3 using loss, accuracy, precision, recall, and training latency metrics. The dissertation report is organized as follows: Chapter 2 reviewed the literature that directly influenced this research. Chapter 3 presents the proposed methodology for studying advanced data augmentation techniques to improve spatial invariance in CNNs. Chapter 4 summarizes the results of this research. Finally, chapter 5 confers conclusions, implications, and recommendations based on this research.

Literature Overview

Neural Networks are biologically inspired computational models that combine multiple nonlinear processing layers in an acyclic graph that consists of an input layer, zero or more hidden layers, and an output layer. The layers interconnect via neurons, where neurons in adjacent layers are fully pairwise connected, but neurons within a layer do not share any connection. Several authors have observed that going deeper than three layers in a deep network rarely increase its representational power, which led to a class of

networks known as deep neural networks, typically having two or more hidden layers. Deep neural networks are excellent at discovering intricate structures in high-dimensional data using deep learning techniques; these techniques automate feature detection in deep models, unlike in traditional networks where domain experts must painstakingly hand-engineered these features. Deep neural networks have made significant progress in many areas, such as computer vision, speech recognition, and natural language processing, due to advances in deep network architectures, powerful computation, and open access to extensive training datasets have fueled that success.

CNNs are specialized neural networks that extract features from data with grid-like topologies, such as images. They typically comprise three main layers: Convolutional layers, Pooling Layers, and a Fully Connected Layer. Neurons in the convolutional layer combine in feature maps, where each neuron connects to local patches in the feature maps of the previous layer through a set of weights called a filter (or kernel). Different feature maps use different filters because local groups of values in images are typically highly correlated, making it easy to detect their patterns. Since the same pattern can appear anywhere within an image, neurons within the same feature map share the same weight. The latter imbues convolutional networks with some degree of translational invariance. The pooling layer merges semantically similar features into one via a down-sampling operation (typically max-pooling), which adds translational invariance and reduces the number of parameters within the network. The fully connected layer is the output layer, where all neurons connect to all the neurons in the previous layer. After passing through the fully connected layers, the final layer uses the softmax activation function to classify the input.

Spatial invariance is a highly desirable property of ConvNets because it allows a network to disentangle the poses and deformations of objects from their texture and shape. For example, this implies that the picture of a cat rotated 10 degrees should still be a cat. Because of their weight-sharing and pooling operations, CNNs are thought to have the property of spatial invariance. Still, several authors have shown this not to be the case, i.e., CNNs are invariant to some but not all transformations. For example, experiments have demonstrated that the chance of misclassification after translating a random image downward by a single pixel can be as high as 30%, raising serious concerns such as fatalities from autonomous vehicles or racial discrimination by image recognition systems.

One way to improve spatial invariance in CNNs, thus enhancing their generalizability and reducing overfitting, is data augmentation, a powerful technique used to artificially increase the training set size to capture more sources of invariance in the data. Traditional transformations, such as horizontal flipping, have proven very effective for simple cases but not enough to significantly improve accuracy, reduce overfitting or minimize adversarial attacks in deep networks. However, using advanced data augmentation techniques such as RandAugment and Generative Adversarial can significantly improve the robustness of CNNs because they provide stronger augmentations leading to more diverse samples and more significant sources of invariance from which to learn. This dissertation validated these approaches using five augmentation policies, NoelNet (a basic CNN developed by the author), two state-of-the-art CNNs: Resnet50 and InceptionV3, across three benchmarks and three conditional DCGAN synthesized datasets.

Methodology

This dissertation conducted 45 experiments using five data augmentation strategies called policies: Policy 1 was the baseline policy with no data augmentation, Policy 2 implemented data augmentation using RandAugment applied to the unaugmented samples, Policy 3 implemented data augmentation using a Conditional DCGAN (cDCGAN) to synthesize training samples, Policy 4 implemented data augmentation using the cDCGAN samples combined with the unaugmented samples, and Policy 5 applied RandAugment on the cDCGAN generated samples. Their results were used to perform a deep comparative analysis of how advanced data augmentation techniques improve the predictive robustness of CNNs by improving their spatial invariance using the following metrics: training/validation accuracy, training/validation loss, test accuracy, recall, precision, f1 score, and training latency. The experiments used three benchmark datasets imported via TensorFlow: MNIST, FMNIST, and CIFAR-10. Of the two advanced data augmentation techniques, RandAugment was imported via Keras API, and the Conditional GAN was built using Keras Sequential API with a modestly designed Generator and Discriminator. Finally, the two ImageNet pre-trained CNNs, ResNet50 and InceptionV3, were imported via Keras API.

RandAugment is an augmentation technique that stochastically applies two geometric transforms with a magnitude of distortion of 9 to each dataset. The Conditional GAN was first trained on each dataset using their respective training datasets. Its generator then synthesized equivalently sized training datasets used by policies 3, 4, and 5 for training the CNNs. Both network architectures used their default hyperparameters with the following changes: 10-neuron output layer instead of 1000, Adam optimizer

with $1e-3$ learning rate, and categorical cross-entropy loss function. Image samples were resized to the minimum size required by each network and were normalized in the $[0,1]$ range. The networks were trained for 100 epochs, using 512 batch sizes, where training stopped after 40 epochs if there was no improvement in validation loss.

Additionally, the learning rate was reduced by a factor between 2 and 10 if there was no improvement in the validation loss after ten epochs. Each policy was implemented over nine stages using a highly performant data input pipeline from TensorFlow's data API: Stage 1 (load the dataset), Stage 2 (partition the dataset into train/validation/test splits), Stage 3 (preprocess dataset samples), Stage 4 (configure dataset using batching, prefetching, shuffling, augmenting), Stage 5 (import and configure CNN), Stage 6 (train CNN using training dataset), Stage 7 (evaluate CNN performance on training/validation data), Stage 8 (evaluate CNN on the testing dataset), and Stage 9 (compute and save performance metrics to file). The results of each experiment were reported in graphical and tabular formats using percentages for scalar metrics and seconds to measure time. A personal computer with highly performant components (liquid-cooled quad-core CPU, 8GB GTX 1080 GPU, 32GB RAM, 512GB SSD storage) was used to conduct each experiment in conjunction with Python 3.9, TensorFlow, and Keras providing the software components.

Results

A total of 45 experiments were conducted using six datasets, three ConvNets (NoelNet, ResNet50, and InceptionV3), and five augmentation policies. The results were presented in tabular and graphical formats and used for analyzing improvements in spatial invariance in ConvNets and for hyperparameter optimization during training of

the conditional DCGAN, ResNet50, and InceptionV3. The final set of hyperparameters used for all experiments is provided in Appendix B, "Hyperparameters." In addition, the author's GitHub account, described in Appendix G, "Dissertation Source Code," contains the complete source code and raw data for all experiments; the raw data was saved in CSV format and included performance metrics and training history. Finally, Appendix F, "Experimental Results.", provides a more detailed description of the data points captured and the results format.

The conditional DCGAN, seen in Figure E1, synthesized three of the six datasets used for the experimentations. The GAN model was trained for 200 epochs on each dataset to select the most optimal hyperparameters. These hyperparameters, detailed in Appendix B, were then used to train the final cDCGAN model for 1000 epochs. The generated images improved in quality from epoch 1 to epoch 1000, demonstrating that the GAN was not simply memorizing the images but learning their salient features to synthesize additional samples. From Table 1, training the cDCGAN took approximately 29.22 hours. Samples of the synthesized images can be seen in Figures D2 and D3.

Figure 13 shows samples of the MNIST dataset used to train NoelNet. As shown in Table 2, the training resulted in Policy 2 achieving the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, which implied they reduced the model's spatial invariance, while Policies 2 and 4 improved the spatial invariance of the model. Training the five policies for MNIST on NoelNet took 3,746 seconds, with Policy 4 having the longest train time and Policy 1 having the shortest train time.

Figure 17 shows samples of the FMNIST dataset used to train NoelNet. As shown in Table 3, the training resulted in Policy 2 achieving the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, which implied they reduced the model's spatial invariance, while Policies 2 and 4 improved the spatial invariance of the model. Training the five policies for FMNIST on NoelNet took 2,712 seconds, with Policy 2 having the longest train time and Policy 3 having the shortest train time.

Figure 21 shows samples of the CIFAR-10 dataset used to train NoelNet. As shown in Table 4, the training resulted in Policy 2 achieving the highest accuracy and f1 scores among all the policies. Policy 2 also resulted in the lowest amount of overfitting and highest generalizability. Policies 3, 4, and 5 had lower accuracies and f1 scores than Policy 1, which implied they reduced the model's spatial invariance, while Policies 2 improved the spatial invariance of the model. Training the five policies for CIFAR-10 on NoelNet took 4,050 seconds, with Policy 4 having the longest train time and Policy 3 having the shortest train time.

Table 5 shows the results of training MNIST on ResNet-50; the training resulted in Policy 2 achieving the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, which implied they reduced the model's spatial invariance, while Policies 2 and 4 improved the spatial invariance of the model. Training the five policies for MNIST on ResNet-50 took 6,314 seconds.

Table 6 shows the results of training FMNIST on ResNet-50; the results show that Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also resulted in the lowest amount of overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, which implied they reduced the model's spatial invariance, while Policies 2 and 4 improved the spatial invariance of the model. Training the five policies for FMNIST on ResNet-50 took 5,301 seconds.

Table 7 shows the results of training CIFAR-10 on ResNet-50; the results show that Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also resulted in the lowest overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, which implied they reduced the model's spatial invariance, while Policies 2 and 4 improved the spatial invariance of the model. Training the five policies for CIFAR-10 on ResNet-50 took 4,802 seconds.

Table 8 shows the results of training the MNIST dataset on InceptionV3, where Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also resulted in the lowest amount of overfitting and highest generalizability. On the other hand, policies 3 and 5 had lower accuracies and f1 scores than Policy 1, implying that Policies 2 and 4 improved the spatial invariance of the model, while Policies 3 and 5 reduced the model's spatial invariance. The total training time for MNIST on InceptionV3 was 12,280 seconds.

Table 9 shows the results of training the FMNIST dataset on InceptionV3, where Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by

Policy 4. Policy 2 also resulted in the lowest overfitting and highest generalizability. On the other hand, policies 3 and 5 had lower accuracies and f1 scores than Policy 1, implying that Policies 2 and 4 improved the spatial invariance of the model, while Policies 3 and 5 reduced the model's spatial invariance. The total training time for FMNIST on InceptionV3 took 10,717 seconds.

Table 10 shows the results of training the CIFAR-10 dataset on InceptionV3, where Policy 2 achieved the highest accuracy and f1 scores among all the policies, followed by Policy 4. Policy 2 also resulted in the lowest overfitting and highest generalizability. Policies 3 and 5 had lower accuracies and f1 scores than Policy 1, implying that Policies 2 and 4 improved the spatial invariance of the model, while Policies 3 and 5 reduced the model's spatial invariance. The total training time for CIFAR-10 on InceptionV3 took 9,542 seconds.

The total training time on InceptionV3 was 9.0383 hours; NoelNet was 2.9183 hours; ResNet-50 was 4.560 hours, meaning NoelNet was three times as fast to train than InceptionV3 while ResNet-50 was twice as fast to train than InceptionV3. This increased training time has led to InceptionV3 achieving the highest accuracies across all datasets used by the Policies. In comparing accuracies, Policy 2 has consistently achieved the highest accuracies across all Policies for each dataset and model, followed by Policy 4. In contrast, Policies 3 and 5 have consistently achieved the lowest accuracies, with Policy 3 achieving the lowest.

Finally, InceptionV3 and NoelNet frequently achieved higher accuracies and f1 scores than ResNet-50 even though they had fewer parameters, suggesting that a higher number of parameters do not indicate higher accuracies or spatial invariance.

Implications, Conclusion, and Recommendations

Spatial invariance is an essential property of ConvNets, and models not implementing strategies to increase spatial invariance can have wide-ranging consequences, such as poor performance on new images, large model size, overfitting, and limited interpretability.

In conclusion, the study found that RandAugment is an effective data augmentation technique for improving spatial invariance and reducing overfitting. Policy 2, which used RandAugment, consistently outperformed other policies regarding accuracy, f1 scores, and overfitting. Furthermore, advanced augmentation techniques were found to be more effective than architectural innovations, as seen by NoelNet consistently achieving higher accuracies than ResNet-50 and competitive accuracies to InceptionV3.

The study also found that combining synthesized and unaugmented samples led to increased spatial invariance of the resulting model. Additionally, wider models such as InceptionV3 were found to be more spatially invariant than deeper models such as ResNet-50 and NoelNet, although deeper models were faster to train.

The study also compared the effectiveness of generative data augmentation and reinforcement learning techniques and found that reinforcement learning produced more diverse samples than generative. Finally, the study found that larger datasets consistently led to more spatially invariant networks, but physically increasing the dataset size resulted in longer training latencies.

As part of future research to improve spatial invariance, and prompted by this dissertation, we could use GANs with more versatile loss functions such as Wasserstein

GANs, generative techniques like Neural Style Transfer, augmentation strategies like CutMix, Cutout, and GridMask, all applied to more respectable datasets like ImageNet.

Based on the deep comparative analysis performed on the results of the extensive experimentation conducted by this dissertation, it is concluded that this research achieved its goal of demonstrating that advanced data augmentation techniques improved spatial invariance and reduced overfitting in deep convolutional networks.

Appendices

Appendix A

Algorithms

Adam Optimizer

Figure A1

Adaptive Moment Estimation (Adam) Algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters θ

Initialize 1st and 2nd moment variables $\mathbf{s} = \mathbf{0}$, $\mathbf{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1) \mathbf{g}$

Update biased second moment estimate: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$

Compute update: $\Delta \theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \delta}}$ (operations applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta \theta$

end while

Note. From *Deep Learning* (p. 306), by I. Goodfellow, Y. Bengio, and A. Courville,

2016, MIT Press.

Softmax

Softmax is an activation function used in multi-class classification problems, specifically at the output layer of a network, where it normalizes the non-negative predictions in the range $[0, 1]$ to get a probability distribution of the classes (Gu et al., 2018). Given a training set $\{(x^i, y^i); i \in 1, \dots, N, y^i \in 1, \dots, K\}$, where x^i is the i^{th} input

image among N images, and y^i is the target class among K classes, the softmax prediction p_j^i of the j^{th} class for the i^{th} input is given by:

$$p_j^i = \frac{e^{z_j^i}}{\sum_{l=1}^K e^{z_l^i}} \quad (8)$$

where z_j^i is the activations of the densely connected output layer. These probabilistic predictions are then used to compute the categorical cross-entropy loss, i.e., the softmax loss, as follows:

$$L_{softmax} = -\frac{1}{N} \left[\sum_{i=1}^N \sum_{j=1}^K \{y^i = j\} \log p_j^i \right] \quad (9)$$

Stochastic Gradient Descent

The backpropagation algorithm (Rumelhart et al., 1986) is used in a neural network to minimize a loss function such as (9) by computing the gradient of the loss with respect to each weight(parameter) in the network and using gradient descent to update the parameters. The standard gradient descent algorithm updates parameters θ of the loss function $L(\theta)$ as $\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} E[L(\theta_t)]$, where $E[L(\theta_t)]$ is the expectation of $L(\theta)$ over the full training set, and α is the learning rate (Gu et al., 2018). Stochastic Gradient Descent does not compute $E[L(\theta_t)]$ but estimate the gradients using backpropagation on a randomly picked example (x^t, y^t) such that:

$$\theta_{t+1} = \theta_t - \alpha_t \nabla_{\theta} L(\theta_t; x^t, y^t) \quad (10)$$

Backpropagation

The backpropagation algorithm (Rumelhart et al., 1986) is used in a neural network to minimize the loss function $L(\theta)$ by computing the gradient of the loss function $\nabla L(\theta)$ with respect to every parameter in the network. This is accomplished

using the chain rule from calculus. Let $\{f^i(x_i, y_i, \theta_i)\}_{i=1}^N = \{f^i\}_{i=1}^N$ be the layers in a neural network, $L(f^i)$ be the loss at the layer f^i over training set $\{x \in \mathbb{R}^d, y \in \mathbb{R}\}$, and θ_{jk}^i be the weight(parameter) mapping output j in layer $i - 1$ to input k in layer i . Using the backpropagation algorithm, the gradient of the loss function $L(f^i)$ with respect to θ_{jk}^i is computed as:

$$\frac{\partial L(f^i)}{\partial \theta_{jk}^i} = \frac{\partial L(f^N)}{\partial L(f^{(N-1)})} * \frac{\partial L(f^{(N-1)})}{\partial L(f^{(N-2)})} * \dots * \frac{\partial L(f^{(N-(i+1))})}{\partial \theta_{jk}^i} \quad (11)$$

, where each element in the chain denotes the gradient of the weight in that layer with respect to the previous layer (Nielsen, 2015, pp. 78-81).

Weight Optimization

Consider a neural network with weight w_{ij} connecting neuron j to neuron i , and a loss function E that computes the network's loss after each training batch. During backpropagation, w_{ij} is updated using a gradient descent algorithm like Adam or SGD that applies the following weight update or optimization rule:

$$w_{ij} = w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}} \quad (12)$$

, where α is a positive constant that is called the learning rate. It is a hyper-parameter that needs to be fine-tuned empirically. The update rule is such that: if the error goes down when the weight increases, i.e., $\frac{dE}{dw_{ij}} < 0$, then increase the weight. Otherwise, if the error goes up when the weight increases, i.e., $\frac{dE}{dw_{ij}} > 0$, then decrease the weight. A neural network learns by using the gradient descent algorithm to update its weights, employing gradients computed by the backpropagation algorithm (LeCun et al., 1998; Goodfellow et al., 2016, p.149).

Figure A2*RandAugment Algorithm in Python*

```

transforms = [
    'Identity', 'AutoContrast', 'Equalize',
    'Rotate', 'Solarize', 'Color', 'Posterize',
    'Contrast', 'Brightness', 'Sharpness',
    'ShearX', 'ShearY', 'TranslateX', 'TranslateY']

def randaugment(N, M):
    """Generate a set of distortions.

    Args:
        N: Number of augmentation transformations to
            apply sequentially.
        M: Magnitude for all the transformations.
    """

    sampled_ops = np.random.choice(transforms, N)
    return [(op, M) for op in sampled_ops]

```

Note. From “RandAugment: Practical automated data augmentation with a reduced search space,” by E. Cubuk, B. Zoph, J. Shlens, and Q. Le, 2020, Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops (pp. 702-703).

Figure A3*Algorithm to Partition a Dataset Into Training, Validation, and Test Splits*

```

def partition_dataset(dataset, train_split, val_split, test_split):
    """Partition dataset into train, validation, and test datasets"""
    # 1. shuffle dataset using same seed
    # 2. compute training size
    # 3. compute validation size
    # 4. extract training size samples from dataset
    # 5. extract validation size samples from dataset
    # 6. extract test size samples from dataset
    return train_dataset, val_dataset, test_dataset

```

Figure A4

Algorithm to Resize Image and One-Hot Encode Label

```
def resize_and_rescale(image, label):  
    """Resize an image sample and one-hot encode its label"""  
    # 1. convert image to RGB (3 channels)  
    # 2. resize image with padding  
    # 3. one-hot encode label  
    return image, label
```

Figure A5

Algorithm to Generate a Performant Data Input Pipeline for TensorFlow

```
def prepare_dataset(dataset, shuffle=False, augment=False):  
    """Create dataset input pipeline"""  
    # 1. Resize and rescale all images in dataset.  
    # 2. If shuffle is true, shuffle dataset  
    # 3. Batch dataset  
    # 4. If augment is true, map each image in dataset to augment function  
    # 5. Prefetch and buffer dataset  
    # 5. return prefetched and buffered dataset  
    return dataset
```

Appendix B

Hyperparameters

This appendix describes the global hyperparameters used to configure and tune the models and algorithms used throughout this dissertation.

Table B1

Hyperparameters That Were Used Throughout This Dissertation

Hyperparameter	Value	Description
Epochs	100	Training iterations
Batch Size	512	Training samples seen by the model, per iteration
Channels	3	Depth dimension for ResNet, Inception inputs
Optimizer	Adam	The amount by which network weights are updated per batch
cDCGAN	Epochs: 1000 Batch size: 128 Num. channels: <ul style="list-style-type: none"> • 1 (F/MNIST) • 3 (CIFAR-10) Num. classes: 10 Latent dim: 100 Learning rate: 0.0002 Momentum: 0.9 Beta1: 0.5 Beta2: 0.9 Kernel size: 5x5 Activation: LeakyReLU (alpha = 0.2) Loss Function: Binary Cross Entropy	These are the Generator and Discriminator hyperparameters used in the Conditional DCGAN model.
ResNet50	Min. Input Size: 32x32x3 Output Layer: 10 neurons Loss function: Categorical Cross Entropy Learning rate: <ul style="list-style-type: none"> • 5e-4 (CIFAR-10) • 3e-4 (F/MNIST) ReduceLRonPlateau: <ul style="list-style-type: none"> • factor: $\frac{1}{3}$ (F/MNIST) 	ResNet50 hyperparameters used in this research

	<ul style="list-style-type: none"> • factor: $\frac{1}{2}$ (CIFAR-10) • patience: 10 <p>EarlyStopping:</p> <ul style="list-style-type: none"> • patience: 40 • restore_best_weights: True <p>ModelCheckpoint:</p> <ul style="list-style-type: none"> • save_weights_only: True • save_best_only: True 	
InceptionV3	<p>Min. Input Size: 75x75x3 Output Layer: 10 neurons Loss function: Categorical Cross Entropy Learning rate:</p> <ul style="list-style-type: none"> • 1e-4 (CIFAR-10) • 3e-4 (F/MNIST) <p>ReduceLROnPlateau:</p> <ul style="list-style-type: none"> • factor: $\frac{1}{3}$ (F/MNIST) • factor: $\frac{1}{2}$ (CIFAR-10) • patience: 10 <p>EarlyStopping:</p> <ul style="list-style-type: none"> • patience: 40 • restore_best_weights: True <p>ModelCheckpoint:</p> <ul style="list-style-type: none"> • save_weights_only: True • save_best_only: True 	InceptionV3 hyperparameters used in this research
RandAugment	<p>N (number of transforms): 2 M (strength of transforms): 9 exclude: Cutout translate_const: 4</p>	Augmenter used in this research. Available from TensorFlow Models Library.
NoelNet	<p>CONV Layers: 9 Kernel Size: 3, 5 Input Size: 32x32x3 Output Layer: 10 neurons Loss function: Categorical Cross Entropy Learning Rate: 1e-3 Dropout: 0.2, 0.3, 0.4, 0.5</p>	A basic CNN used to establish competitive baseline metrics on F/MNIST and CIFAR-10.

Appendix C

Third-Party Libraries

This appendix lists several third-party libraries for the Python programming language used in this dissertation. The libraries are listed with their current version used and without modifications to their source code except via configuration settings.

TensorFlow

TensorFlow 2.10.1 is a free and open-source platform for machine learning that Google developed. It provides a comprehensive selection of tools, libraries, workflows, pipelines, and community resources to support machine learning-powered applications. It allows developers to create machine learning models in numerous languages, which can be deployed to servers, cloud, mobile, edge devices, and JavaScript-powered platforms like browsers. It was used in this dissertation to support all experiments requiring machine learning models and pipelines.

Keras

Keras 2.10.0 is an open-source, deep-learning API developed by Google and written in Python to run on top of the machine-learning platform TensorFlow. It enables the fast development of complex machine learning workflows that are both industry performant and scalable. It was used in this dissertation to build and configure several Convolutional Neural Networks and monitor and report their performance.

Pandas

Pandas 1.5.3 is an open-source, flexible, powerful, easy-to-use, fast data analysis and manipulation library built on Python. It was used in this dissertation to process and

export the performance metrics datasets of the used Convolutional Networks to CSV format.

SciPy

SciPy 1.9.2 is an open-source Python library for mathematics, science, and engineering. It contains modules for optimization, numerical integration, interpolation, linear algebra, FFT, ODE solvers, signal processing, and other everyday tasks in science and engineering. This dissertation used SciPy as a requirement for Scikit-Learn in generating classification reports for each ConvNet.

Scikit-Learn

Scikit-Learn 1.2.1 is a free machine learning library developed for Python that provides efficient tools for various tasks such as classification, regression, clustering, and dimensionality reduction via a consistent Python interface. It was built upon NumPy, SciPy, and Matplotlib and was used in this dissertation to generate classification reports on the performance of ConvNets.

Numpy

NumPy 1.24.1 is an open-source library extension to Python consisting of multidimensional array objects and an extensive collection of performant high-level routines for processing these structures. It was used in this dissertation to perform concatenation, aggregation, and reshaping of several datasets.

Matplotlib

Matplotlib 3.6.3 is a comprehensive library for creating static, dynamic, and interactive data visualizations in Python. It provides an Object-Oriented API for embedding plots into applications using Python GUI toolkits such as PyQt/PySide,

PyGObject, TK+, and wxPython. It can be used in Python, IPython shells, Jupyter notebooks, and web applications. This dissertation used Matplotlib to generate several plots to report the performance metrics of ConvNets.

Appendix D

Datasets

This dissertation used three benchmark datasets and three synthetic datasets. The synthetic datasets were generated by a Conditional DCGAN, using the three benchmark datasets as training examples. Table D1 shows a complete list of these datasets and the experiments (*policies*) using them.

Table D1

Type of Dataset Used by Each of the Dissertation Experiments

Dataset type	Dataset name	Policy
Benchmark(<i>real</i>)	MNIST	1, 2, 4
Benchmark(<i>real</i>)	FMNIST	1, 2, 4
Benchmark(<i>real</i>)	CIFAR-10	1, 2, 4
Synthetic	MNIST_GAN	3,4,5
Synthetic	FMNIST_GAN	3,4,5
Synthetic	CIFAR10_GAN	3,4,5

Table D2

FashionMNIST Classes

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Table D3

CIFAR-10 Classes

Label	Description
0	airplane
1	automobile
2	bird
3	cat
4	deer
5	dog
6	frog
7	horse
8	ship
9	truck

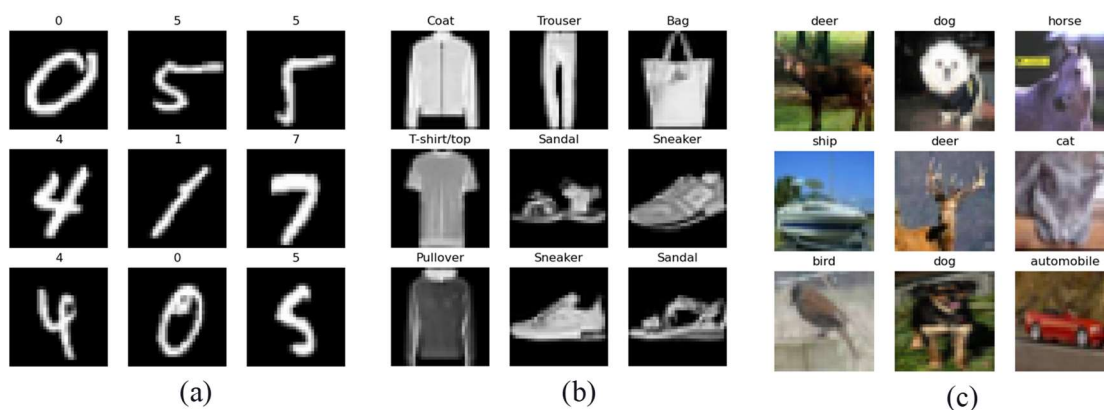
Table D4

MNIST Classes

Label	Description
0	Zero
1	One
2	Two
3	Three
4	Four
5	Five
6	Six
7	Seven
8	Eight
9	Nine

Figure D1

Sample images from (a) MNIST, (b) FMNIST, and (c) CIFAR-10

**Figure D2**

Sample Images at Various Epochs Generated Using the Conditional DCGAN

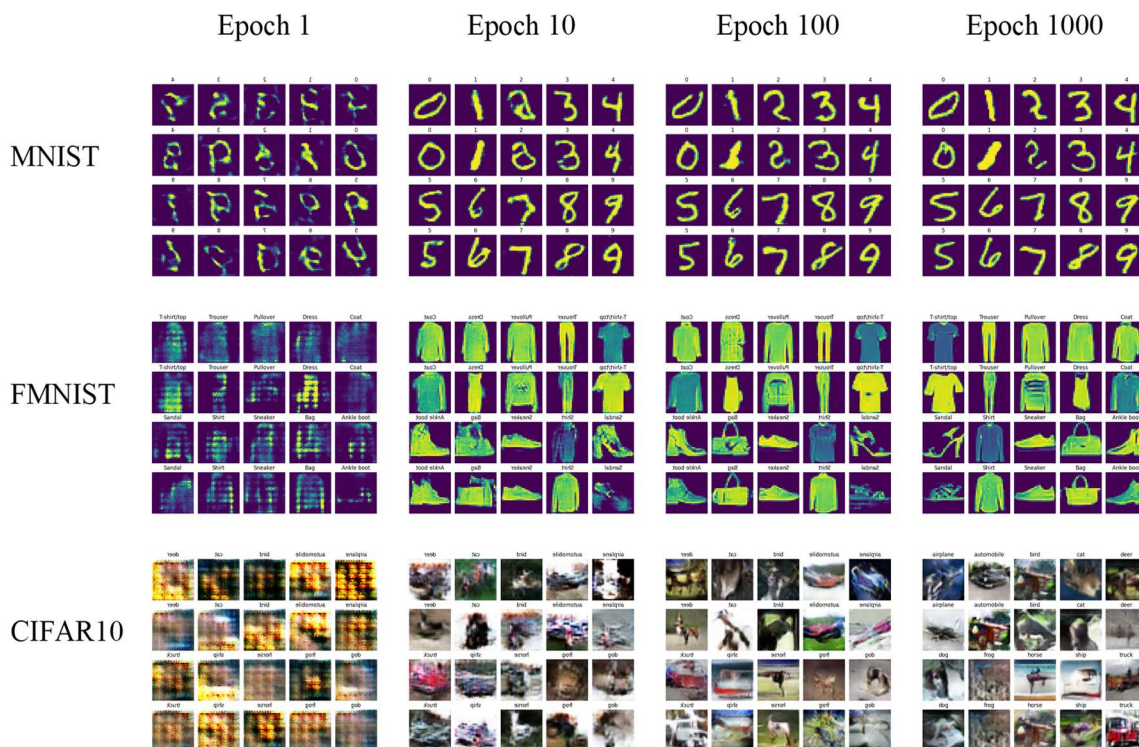
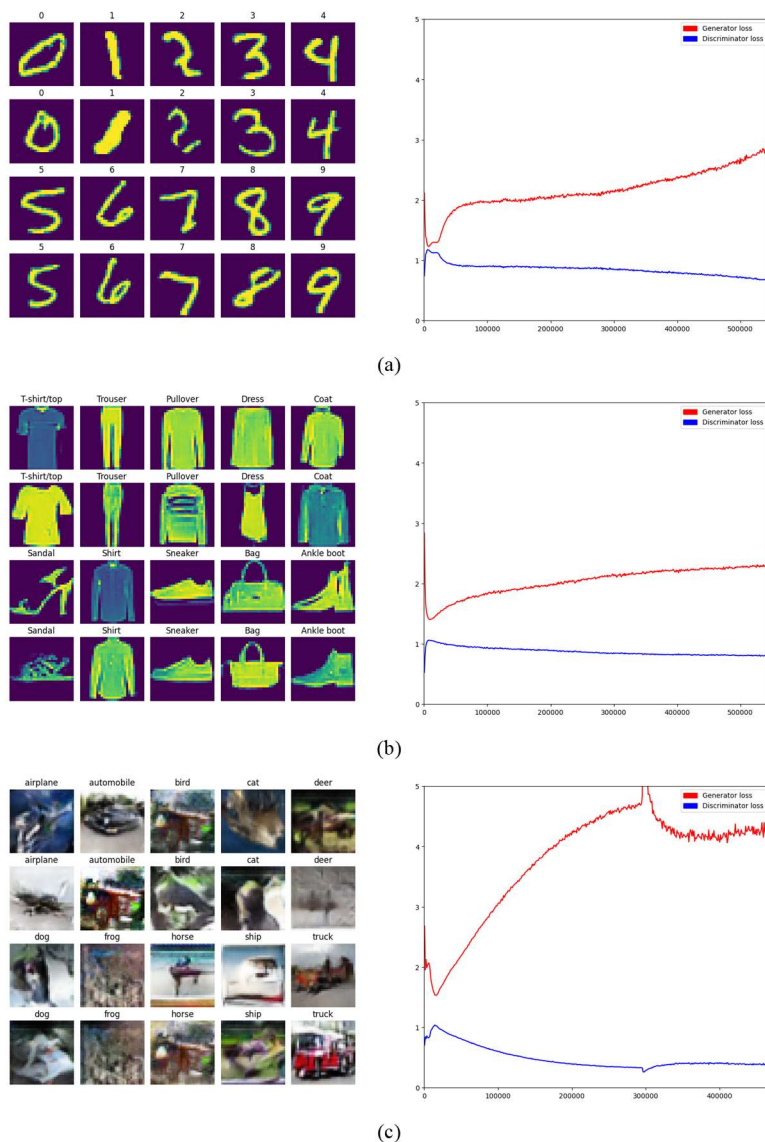


Figure D3

CDCGAN Images and Loss at Epoch 1000 (a) MNIST, (b) FMNIST, and (c) CIFAR-10



Note. The mean loss of the generator and discriminator is shown alongside each generated sample. The losses stabilize after approximately 500 epochs. At this point, further improvements in loss are minimal, which implies that the generator's ability to fool the discriminator is roughly equivalent to the discriminator's ability to detect fake samples. No significant improvement in the quality of images can be expected by further training.

Appendix E

Architectures

This appendix outlines the architectures used to build each model, data input pipeline, or other system throughout this dissertation.

Figure E1

The Conditional DCGAN Architecture used for MNIST, FMNIST and CIFAR-10.

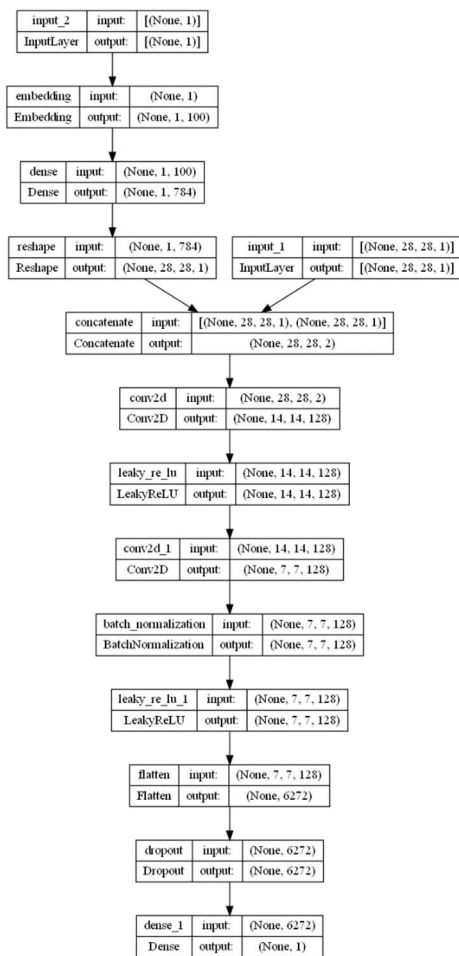
(a) Discriminator, (b) Generator

F/MNIST cDCGAN

Total params: 502,969

Trainable params: 502,713

Non-trainable params: 256

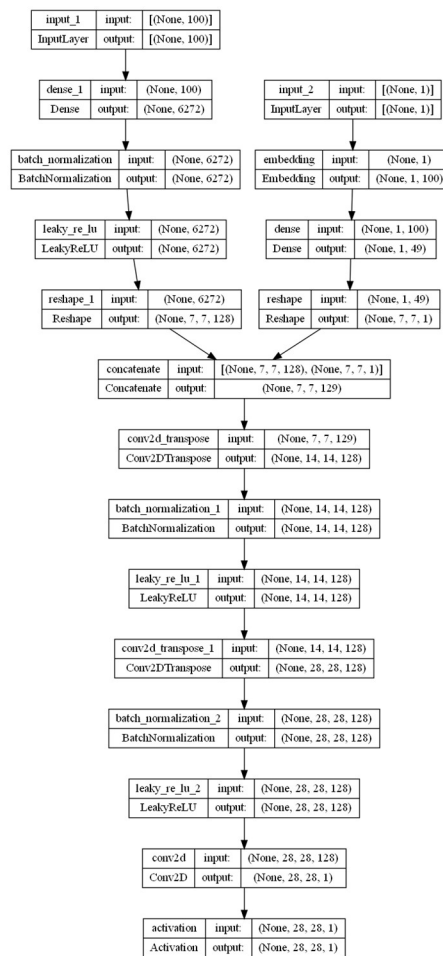


(a)

Total params: 1,484,862

Trainable params: 1,471,806

Non-trainable params: 13,056



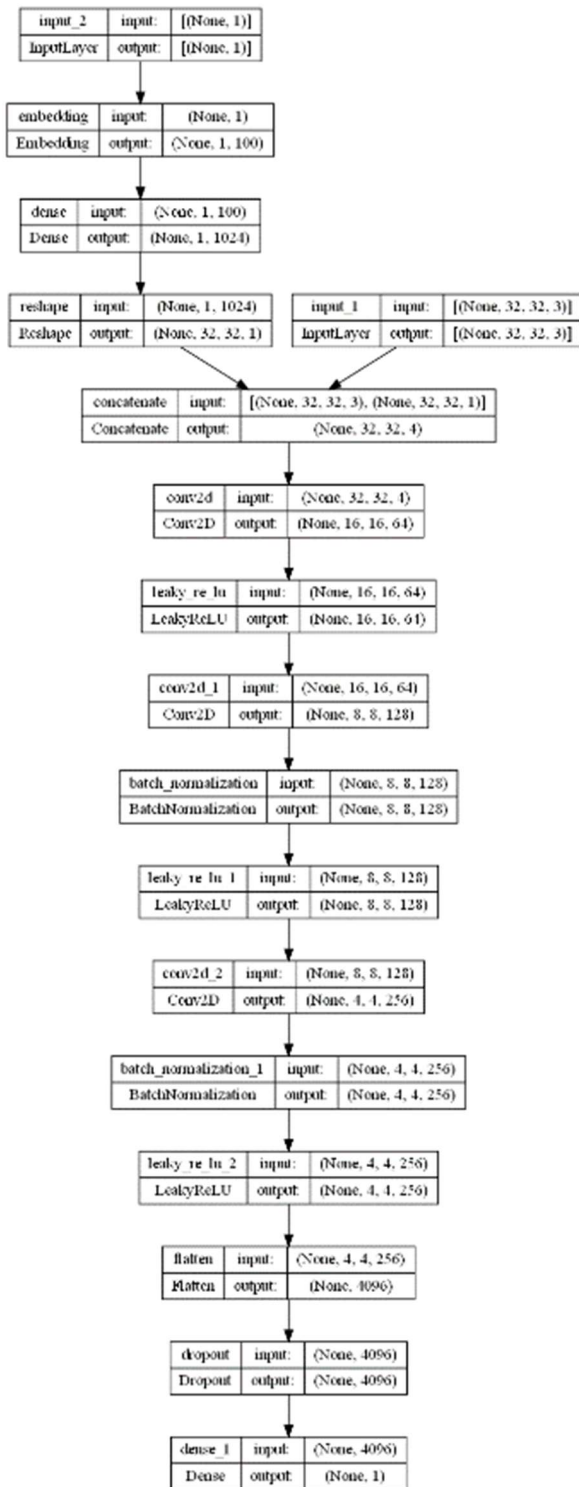
(b)

CIFAR-10 cDCGAN

Total params: 1,140,457

Trainable params: 1,139,689

Non-trainable params: 768

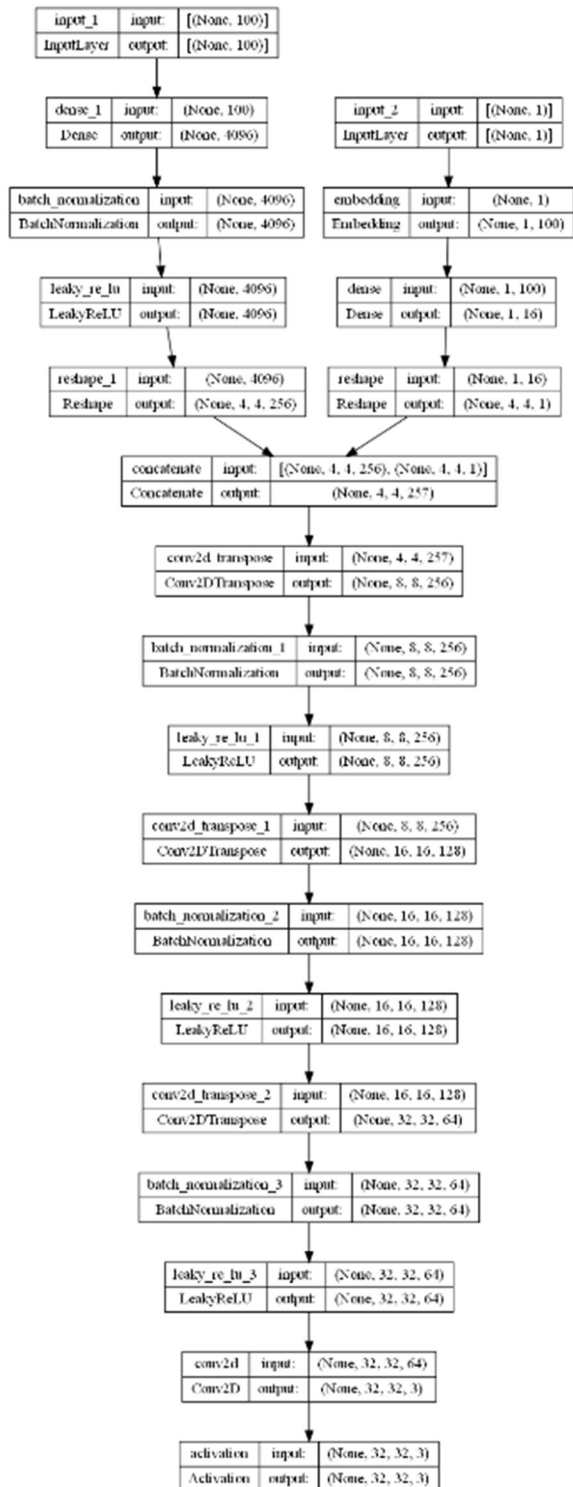


(a)

Total params: 3,103,995

Trainable params: 3,094,907

Non-trainable params: 9,088



(b)

Figure E2*Architectural Components of Various ResNet ConvNets*

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Note. Building blocks are shown in brackets and the number of stacked blocks. Adapted from “Deep Residual Learning for Image Recognition,” by K. He, X. Zhang, S. Ren, and J. Sun, 2015, arXiv.

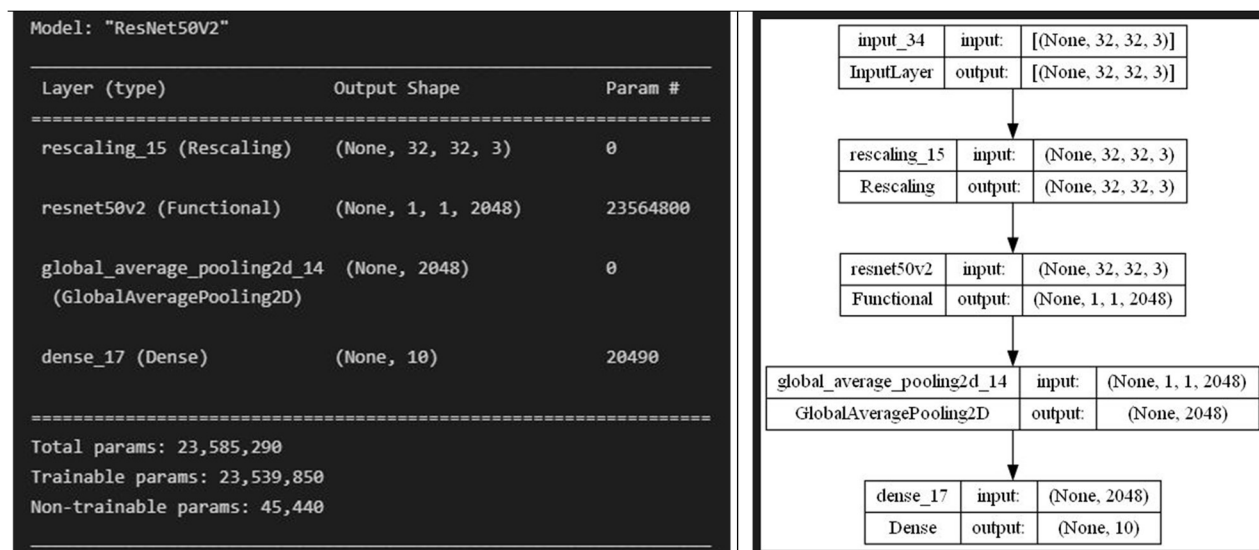
Figure E3*Model of ResNet50 Used for This Dissertation*

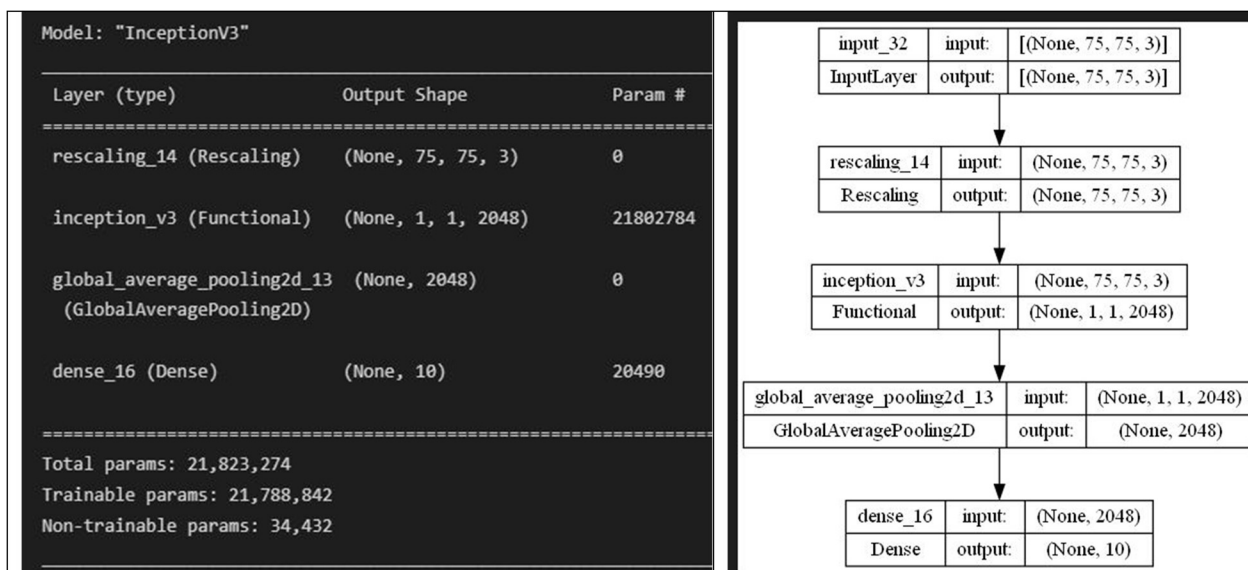
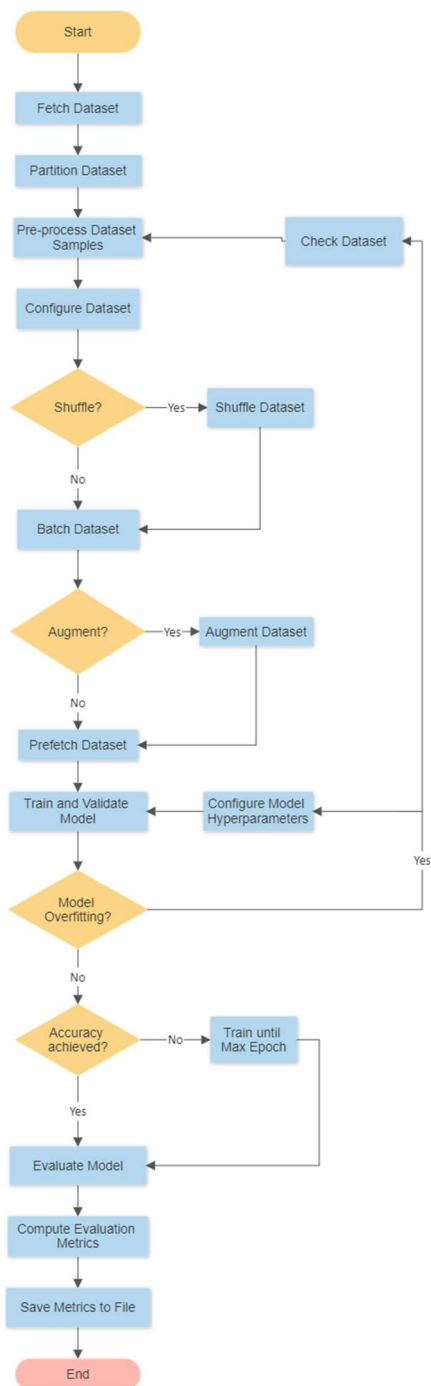
Figure E4*Model of InceptionV3 Used for This Dissertation*

Figure E5

Data Input Pipeline Used by Each Policy to Evaluate Their Respective Datasets



Note. The pipeline loaded & partitioned the dataset into train/validation/test splits, preprocessed dataset samples, configured the dataset using batching, prefetching,

shuffling, augmenting, trained the CNN using the training dataset, evaluated the CNN performance on training/validation data, evaluated the CNN on the testing dataset, and computed and save performance metrics to file.

Appendix F

Experimental Results

The results collected from the experiments were in graphical and tabular formats, and the data is available on the author's GitHub repository. Information about their access is detailed in Appendix G, "Dissertation Source Code." The tabular data is stored as CSV files, and the graphical data is stored as PNG files. This dissertation produced 12 CSV files containing the raw data of each model's performance metrics and history.

The format for the metrics filenames is `<dataset_model_metrics>.csv`, where *dataset* is the dataset being used, and *model* is the CNN being used. For example, when evaluating the MNIST dataset on ResNet50, the output filename is `mnist_resnet_metrics.csv`. Similarly, the training history is saved to filenames in the format `<dataset_model_history>.csv`

Format for `<dataset_model_metrics>.csv`

The fields contained in each metrics file are as follows:

- "dataset" – The name of the dataset being used.
- "model" – The convolutional network being used; *ResNet50* or *InceptionV3*.
- "params" – The total number of parameters in the convolutional network.
- "policy" – The name of the policy being evaluated; *Policy 1* to *Policy 5*.
- "accuracy" – The accuracy of the model over the test set.
- "precision" – The precision of the model over the test set.
- "recall" – The recall(sensitivity) of the model over the test set.
- "f1score" – The harmonic average of the precision and recall.
- "latency" – The total time it took to train the model measured in seconds.

Format for <dataset_model_history>.csv

The fields contained in each history file are as follows:

- "loss" – The training loss of the model when evaluated on the training dataset.
- "accuracy" – The training accuracy of the model evaluated on the training dataset.
- "val_loss" – The validation loss of the model evaluated on the validation dataset.
- "val_accuracy" – The validation accuracy of the model on the validation dataset.
- "lr" – The learning rate used by the optimizer.
- "policy" – The number of the policy being evaluated.

The graphical results collected include the training/validation accuracy graphs, the training/validation loss graphs, training latency charts, testing accuracy, precision, recall, and f1score graphs. Their configuration is as follows:

Format of Training/Validation Accuracy Graphs

- “x-axis”: the training epoch
- “y-axis”: the training and validation accuracy values
- “title”: the title of the graph

Format of Training/Validation Loss Graph

- “x-axis”: the training epochs
- “y-axis”: the training and validation loss values
- “title”: the title of the graph

Format of the Latency Graphs

- “x-axis”: the training latency in seconds
- “y-axis”: the policy number

- “title”: the title of the graph

Summary of Expected Results

Table F1

Summary of Policies and Expected Metrics Relative to the Baseline

	Loss	Accuracy	Precision	Recall	Latency
Policy 1	baseline	baseline	baseline	baseline	baseline
Policy 2	smaller	higher	higher	higher	higher
Policy 3	smaller	higher	higher	higher	similar
Policy 4	smallest	highest	highest	highest	highest
Policy 5	smaller	higher	higher	higher	higher

Note. Policy 1 is a baseline policy that establishes a reference for loss, accuracy, precision, recall, and training latency for all experiments. Policy 2 uses traditional augmentation using stochastic mechanisms and is expected to have more minor losses, higher accuracies, precision, and recall compared to Policy 1. In addition, its training latency has been higher since augmentation was online. Policy 3 is expected to have lesser losses and higher accuracies, precision, and recall than Policies 1 and 2 since GANs are supposed to generate samples with greater diversity in their feature distributions. In addition, its training time is expected to be like Policy 1 but less than Policy 2. Policy 4 is expected to have the smallest losses and the highest accuracy, precision, and recall of all five policies. It is expected to have the highest training time since it combines two datasets (baseline and GAN samples). Policy 5 is expected to have similar or slightly better metrics than Policy 2 but with higher training time than Policy 1.

Table F2

Summary of Policies and Actual Metrics Relative to the Baseline

	Loss	Accuracy	Precision	Recall	Latency
Policy 1	baseline	baseline	baseline	baseline	baseline
Policy 2	lowest	highest	highest	highest	higher
Policy 3	highest	lowest	lowest	lowest	lowest
Policy 4	lower	higher	higher	higher	highest
Policy 5	higher	lower	lower	lower	higher

Note. Table F2 shows the actual results after experimentation. After conducting the experiments, some results were different than expected. The results showed that Policy 2 led to the highest accuracy, precision, recall, and lowest loss but had the second-highest training time. It was followed by Policy 4, which had the second-best metrics but had the highest train time. Policy 3 had the fastest training time but had the lowest accuracy, precision, recall, and highest loss. Finally, except for training time, Policy 5 had better metrics than Policy 3 but was still not higher than the baseline. In summary, Policies 2 and 4 improved the spatial invariance of the models, while Policies 3 and 5 did not improve the spatial invariance of the models.

Appendix G

Dissertation Source Code

The source code for this dissertation is available from the author's GitHub repository¹. The files in this dissertation consist primarily of Python source code, raw data CSV files, and other generated files. The repository comprises three main folders: **analysis**, **gan**, and **invariance**. Each folder is subdivided into the three datasets used and a Jupyter notebook with the source code for each section. The repository layout is as follows:

gan:

- CIFAR10: contains sample images, loss data, and trained cDCGAN models for the CIFAR-10 dataset. The *cifar10_gan.h5* model can synthesize CIFAR-10 images for specific classes.
- FMNIST: contains sample images, loss data, and trained cDCGAN models for the FMNIST dataset. The *fmnist_gan.h5* model can synthesize FMNIST images for specific classes.
- MNIST: contains sample images, loss data, and trained cDCGAN models for the MNIST dataset. In addition, the *mnist_gan.h5* model can synthesize MNIST images for specific classes.
- Conditional DCGAN.ipynb: contains Python source code that creates a Conditional Deep Convolutional GAN (cDCGAN) for each dataset. The GAN was then trained on each dataset for 1000 epochs. After each training epoch, sample images and losses were displayed, and the corresponding generator and

¹ Available from: <https://github.com/davidanoel/phd>

discriminator model for that epoch was saved to the appropriate dataset file. The generator file for each dataset can then synthesize images of specific classes for that dataset.

invariance:

- `cifar10`: contains the raw performance metrics and training history data in CSV format for CIFAR-10 on InceptionV3 and ResNet50. It also contains the loss and accuracy charts for CIFAR-10 on each model.
- `fmnist`: contains the raw performance metrics and training history data in CSV format for FMNIST on InceptionV3 and ResNet50. It also contains the loss and accuracy charts for FMNIST on each model.
- `mnist`: contains the raw performance metrics and training history data in CSV format for MNIST on InceptionV3 and ResNet50. It also contains the loss and accuracy charts for MNIST on each model.
- `cifar10_gan.h5`: the trained CIFAR-10 GAN generator model.
- `fmnist_gan.h5`: the trained FMNIST GAN generator model.
- `mnist_gan.h5`: the trained MNIST GAN generator model.
- `spatial_invariance.ipynb`: notebook containing Python code to create the data input pipeline described in Figure E5. It generates GAN samples of each dataset, creates NoeNet, ResNet50, and InceptionV3 models, trains each model on the benchmark and synthesized datasets, plots training performance, and saves performance metrics and history to file.

The trained NoeINet, ResNet50, and InceptionV3 models for each dataset and policy are available from Google Drive² since their large file sizes prevent storage on GitHub.

analysis:

- `cifar10`: contains the raw performance metrics and training history data in CSV format for CIFAR-10 on InceptionV3 and ResNet50. It also contains the loss and accuracy charts for CIFAR-10 on each model.
- `fmnist`: contains the raw performance metrics and training history data in CSV format for FMNIST on InceptionV3 and ResNet50. It also contains the loss and accuracy charts for FMNIST on each model.
- `mnist`: contains the raw performance metrics and training history data in CSV format for MNIST on InceptionV3 and ResNet50. It also contains the loss and accuracy charts for MNIST on each model.
- `analysis.ipynb`: contains Python source code that generates analytics, statistics, charts, and graphs for each dataset on each model to analyze the effects of spatial invariance.

To run all experiments, install the libraries and environments outlined in Appendix C, then clone the GitHub repository¹ to a folder on the local machine. Next, open the notebook of interest and run it using Python. To replicate the experiments, execute the **`spatial_invariance.ipynb`** notebook. To generate and train the cDCGAN, execute **`Conditional DCGAN.ipynb`**. Furthermore, to perform analytics on the results, execute **`analysis.ipynb`**. It should be noted that running the experiments or training the cDCGAN could take several hours, depending on the computer's speed.

² Available from [Google Drive](#)

References

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.
- Antoniou, A., Storkey, A., & Edwards, H. (2017). Data Augmentation Generative Adversarial Networks. *arXiv*. <https://doi.org/10.48550/arXiv.1711.04340>
- Arjovsky, M., Bottou, L., & Gulrajani, I. (2019). Invariant Risk Minimization. *arXiv*. <https://doi.org/10.48550/arXiv.1907.02893>
- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. In International conference on machine learning (pp. 214-223). PMLR.
- Azulay, A., & Weiss, Y. (2018). Why do deep convolutional networks generalize so poorly to small image transformations?. *arXiv*. <https://doi.org/10.48550/arXiv.1805.12177>
- Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1), 1-127.
- Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19.
- Bowles, C., Chen, L., Guerrero, R., Bentley, P., Gunn, R., Hammers, A., ... & Rueckert, D. (2018). GAN Augmentation: Augmenting Training Data using Generative Adversarial Networks. *arXiv*. <https://doi.org/10.48550/arXiv.1810.10863>
- Chen, P., Liu, S., Zhao, H., & Jia, J. (2020). Gridmask data augmentation. arXiv preprint arXiv:2001.04086.
- Chen, S., Dobriban, E., & Lee, J. H. (2020). A group-theoretic framework for data augmentation. *The Journal of Machine Learning Research*, 21(1), 9885-9955.
- Chollet, F. (2015). Keras. <https://github.com/fchollet/keras>
- Cohen, T., & Welling, M. (2016). Group equivariant convolutional networks. In International conference on machine learning (pp. 2990-2999). PMLR.
- Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*.

- Cubuk, E. D., Zoph, B., Shlens, J., & Le, Q. V. (2020). RandAugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops* (pp. 702-703).
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4), 303-314.
- Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 764-773).
- Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)* (Vol. 1, pp. 886-893).IEEE.
- Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). IEEE.
- Denton, E., Chintala, S., Szlam, A., & Fergus, R.. (2015). Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks.
- DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.
- Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., & Madry, A. (2018). A rotation and a translation suffice: Fooling cnns with simple transformations.
- Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639), 115-118.
- Frid-Adar, M., Diamant, I., Klang, E., Amitai, M., Goldberger, J., & Greenspan, H. (2018). GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing*, 321, 321-331.
- Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets* (pp. 267-285). Springer, Berlin, Heidelberg.
- Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576.

- Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Amsterdam University Press.
- Goodfellow, I. J., Mirza, M., Xu, B., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. *arXiv*. <https://doi.org/10.48550/arXiv.1406.2661>
- Goodfellow, I., Lee, H., Le, Q., Saxe, A., & Ng, A. (2009). Measuring invariances in deep networks. *Advances in neural information processing systems*, p. 22.
- Gowal, S., Qin, C., Huang, P. S., Cemgil, T., Dvijotham, K., Mann, T., & Kohli, P. (2020). Achieving robustness in the wild via adversarial mixing with disentangled representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 1211-1220).
- Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, 77, 354-377.
- Harris, C.R., Millman, K.J., van der Walt, S.J. et al. (2020). Array programming with NumPy. *Nature* 585, 357–362.
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv*. <https://doi.org/10.48550/arXiv.1512.03385>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026-1034).
- Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7), 1527-1554.
- Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... & Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6), 82-97.
- Hosseini, H., Xiao, B., & Poovendran, R. (2017). Google's cloud vision api is not robust to noise. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)* (pp. 101-105). IEEE.

- Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1), 106.
- Hunter, J. (2007). Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3), 90–95.
- Immer, A., Rättsch, G., & Fortuin, V. (2022). Invariance Learning in Deep Neural Networks with Differentiable Laplace Approximations. *arXiv*. <https://doi.org/10.48550/arXiv.2202.10638>
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.
- Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015). Spatial Transformer Networks. *arXiv*. <https://doi.org/10.48550/arXiv.1506.02025>
- Karpathy, A. (2016). Connecting images and natural language [Doctoral dissertation, Stanford University]. Stanford Digital Repository.
- Kendall, A., & Gal, Y. (2017). What uncertainties do we need in Bayesian deep learning for computer vision? *Advances in neural information processing systems*, p. 30.
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*. 25.
- LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. doi:10.1038/nature14539
- LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems*. Morgan-Kaufmann.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

- Lenc, K., & Vedaldi, A. (2015). Understanding image representations by measuring their equivariance and equivalence. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 991-999.
- Li, F. F., Karpathy, A., & Johnson, J. (2015). Convolutional neural networks for visual recognition. Available at <http://cs231n.github.io/convolutional-networks>.
- Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2), 91-110.
- Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.
- McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 115-133.
- Mirza, M., & Osindero, S. (2014). Conditional Generative Adversarial Nets. *arXiv*. <https://doi.org/10.48550/arXiv.1411.1784>
- Nair, V., & Hinton, G. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (pp. 807–814). Omnipress.
- Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination Press.
- Palatucci, M., Pomerleau, D., Hinton, G. E., & Mitchell, T. M. (2009). Zero-shot learning with semantic output codes. *Advances in neural information processing systems*, 22.
- Paul, S. (2021). Conditional GAN. GitHub. Retrieved January 23, 2023, from https://github.com/keras-team/keras-io/blob/master/examples/generative/conditional_gan.py
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *The Journal of machine Learning research*, 12, 2825-2830.
- Perez, L., & Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *arXiv*. <https://doi.org/10.48550/arXiv.1712.04621>
- Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

- Ranzato, M. A., Boureau, Y. L., & Cun, Y. (2007). Sparse feature learning for deep belief networks. *Advances in neural information processing systems*, 20.
- Riesenhuber, M., & Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11), 1019-1025.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408.
<https://doi.org/10.1037/h0042519>
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088), 533-536.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3), 211-252.
- Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. *Advances in neural information processing systems*, 29.
- Sánchez, J., Perronnin, F., Mensink, T., & Verbeek, J. (2013). Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3), 222-245.
- Schwöbel, P., Jørgensen, M., Ober, S. W., & Van Der Wilk, M. (2022). Last Layer Marginal Likelihood for Invariance Learning. In *International Conference on Artificial Intelligence and Statistics* (pp. 3542-3555). PMLR.
- Shijie, J., Ping, W., Peiyi, J., & Siping, H. (2017). Research on data augmentation for image classification based on convolution neural networks. In *2017 Chinese automation congress (CAC)* (pp. 4165-4170). IEEE.
- Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1), 60. doi:10.1186/s40537-019-0197-0
- Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Icdar* (Vol. 3, No. 2003).
- Simoncelli, E. P., Freeman, W. T., Adelson, E. H., & Heeger, D. J. (1992). Shiftable multiscale transforms. *IEEE transactions on Information Theory*, 38(2), 587-607.
- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

- Smith, L. N. (2017). Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)* (pp. 464-472). IEEE.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, *15*(1), 1929–1958.
- Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.
- Su, J., Vargas, D. V., & Sakurai, K. (2019). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, *23*(5), 828-841.
- Sun, C., Shrivastava, A., Singh, S., & Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision* (pp. 843-852).
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.
- Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). *Going deeper with convolutions*. arXiv.org. Retrieved January 19, 2023, from <https://arxiv.org/abs/1409.4842>
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818-2826).
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.
- Tompson, J., Goroshin, R., Jain, A., LeCun, Y., & Bregler, C. (2015). Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 648-656).
- Torfi, A., Shirvani, R. A., Keneshloo, Y., Tavaf, N., & Fox, E. A. (2020). Natural language processing advancements by deep learning: A survey. *arXiv preprint arXiv:2003.01200*.

- van der Wilk, M., Bauer, M., John, S. T., & Hensman, J. (2018). Learning invariances using the marginal likelihood. *Advances in Neural Information Processing Systems*, 31.
- Van Rossum, G. (1995). Python reference manual. Department of Computer Science [CS]. CWI.
- Wang, T., Wu, D. J., Coates, A., & Ng, A. Y. (2012, November). End-to-end text recognition with convolutional neural networks. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)* (pp. 3304-3308). IEEE.
- Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big data*, 3(1), 1-40.
- Wes McKinney (2010). Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference* (pp. 56 - 61).
- Xian, Y., Lampert, C. H., Schiele, B., & Akata, Z. (2018). Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 41(9), 2251-2265.
- Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.
- Xie, Q., Dai, Z., Hovy, E., Luong, T., & Le, Q. (2020). Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems*, pp. 33, 6256–6268.
- Yang, S., Xiao, W., Zhang, M., Guo, S., Zhao, J., & Shen, F. (2022). Image Data Augmentation for Deep Learning: A Survey. *arXiv preprint arXiv:2204.08610*.
- Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., & Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 6023-6032).
- Zhang, R. (2019). Making convolutional networks shift-invariant again. In *International conference on machine learning* (pp. 7324-7334). PMLR.
- Zhou, Y. T., & Chellappa, R. (1988). Computation of optical flow using a neural network. In *ICNN* (pp. 71-78).