

Electronic Thesis and Dissertation Repository

---

2-16-2024 9:45 AM

## Attribution Robustness of Neural Networks

Sunanda Gamage, *Western University*

Supervisor: Samarabandu, Jagath, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Electrical and Computer Engineering

© Sunanda Gamage 2024

Follow this and additional works at: <https://ir.lib.uwo.ca/etd>



Part of the [Artificial Intelligence and Robotics Commons](#), and the [Computer Engineering Commons](#)

---

### Recommended Citation

Gamage, Sunanda, "Attribution Robustness of Neural Networks" (2024). *Electronic Thesis and Dissertation Repository*. 9956.

<https://ir.lib.uwo.ca/etd/9956>

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact [wlsadmin@uwo.ca](mailto:wlsadmin@uwo.ca).

# Abstract

While deep neural networks have demonstrated excellent learning capabilities, explainability of model predictions remains a challenge due to their black box nature. Attributions or feature significance methods are tools for explaining model predictions, facilitating model debugging, human-machine collaborative decision making, and establishing trust and compliance in critical applications. Recent work has shown that attributions of neural networks can be distorted by imperceptible adversarial input perturbations, which makes attributions unreliable as an explainability method. This thesis addresses the research problem of attribution robustness of neural networks and introduces novel techniques that enable robust training at scale.

Firstly, a novel generic framework of loss functions for robust neural net training is introduced, addressing the restrictive nature of existing frameworks. Secondly, the bottleneck issue of high computational cost of existing robust objectives is addressed by deriving a new, simple and efficient robust training objective termed “cross entropy of attacks”. It is 2 to 10 times faster than existing regularization-based robust objectives for training neural nets on image data while achieving higher attribution robustness (3.5% to 6.2% higher top-k intersection).

Thirdly, this thesis presents a comprehensive analysis of three key challenges in attribution robust neural net training: the high computational cost, the trade-off between robustness and accuracy, and the difficulty of hyperparameter tuning. Empirical evidence and guidelines are provided to help researchers navigate these challenges. Techniques to improve robust training efficiency are proposed, including hybrid standard and robust training, using a fast one-step attack, and optimized computation of integrated gradients, yielding 2x to 6x speed gains.

Finally, an investigation of two properties of attribution robust neural networks is conducted. It is shown that attribution robust neural nets are also robust against image corruptions, achieving accuracy gains of 3.58% to 11.94% over standard models. Empirical results suggest that robust models do not exhibit resilience against spurious correlations.

This thesis also presents work on utilizing deep learning classifiers in multiple application domains: an empirical benchmark of deep learning in intrusion detection, an LSTM-based pipeline for detecting structural damage in physical structures, and a self-supervised learning pipeline to classify industrial time-series in a label efficient manner.

**Keywords:** Attribution robustness, attribution attacks, feature significance, network intrusion detection, self-supervised learning, industrial time-series classification

## Summary for Lay Audience

Deep learning, a branch of artificial intelligence, has transformed how machines learn from data, making significant advancements in technology. However, understanding how these systems make their decisions has been a challenge, mainly because their processes are not always transparent. This issue of transparency is crucial, especially when these systems are used in areas where trust and reliability are paramount, like in healthcare or self-driving cars.

My research focuses on improving the transparency and reliability of deep learning systems. Specifically, I developed new methods to ensure that the explanations provided by these systems about their decisions remain accurate, even when faced with misleading information that could cause errors. This is important because, just like in human decision-making, understanding why a decision was made is often as critical as the decision itself.

One of the main achievements of my work is a novel approach to training these artificial intelligence systems that is both faster and produces more reliable explanations than previous methods. This means that we can now make these large AI systems more understandable and trustworthy more efficiently. Additionally, my research has shown that making these systems' decision-making processes more transparent not only helps in understanding their choices but also improves their performance in handling poor-quality data, such as blurry images.

Beyond enhancing the transparency and reliability of AI decisions, my thesis applies these advanced deep learning techniques to solve practical problems in various fields, from detecting security breaches in computer networks to assessing structural damage in buildings and efficiently analyzing industrial data.

In summary, my thesis aims to make deep learning AI systems more transparent, understandable, and trustworthy, ensuring that as these technologies become more integrated into our lives, they can be relied upon to make decisions in a way that we can understand and trust.

# Co-Authorship Statements

## Co-Authorship Statement for the Chapter: Deep Learning Methods in Network Intrusion Detection

The chapter “Deep Learning Methods in Network Intrusion Detection: a Survey and an Objective Comparison” (Appendix B) is based on the following publication [19].

- Gamage, Sunanda, and Jagath Samarabandu. "Deep learning methods in network intrusion detection: A survey and an objective comparison." *Journal of Network and Computer Applications* 169 (2020): 102767.

### Contributions by the author of the thesis

As the first author, I made the following contributions. The percentages indicate the degree of my contribution for each segment. The second author is my Ph.D. supervisor, and provided the critical research supervision, guidance and revisions for the publication.

- **Conceptualization (95%)**. Identifying the need for a survey paper and a comprehensive empirical benchmark in computer network intrusion detection with deep neural networks. Developing the taxonomy for neural network models and training paradigms in the intrusion detection application domain.
- **Design and implementation (100%)**. Identifying the models and datasets for the benchmark, and complete design and implementation.
- **Experimentation (100%)**. Running all experiments of the project, compiling and analyzing the results.
- **Writing (95%)**. Writing the full paper, engaging in the peer review process and making the revisions.

# Co-Authorship Statement for the Chapter: Structural Damage Detection and Localization

The chapter “Structural Damage Detection and Localization using Neural Networks on Vibration Signals” (Appendix D) is based on the following collaborative publication [62] with researchers from the Civil and Environmental Engineering department at Western University. The original manuscript has been slightly modified to highlight the machine learning aspects of the work.

- Sony, S., Gamage, S., Sadhu, A., & Samarabandu, J. (2022, January). Vibration-based multiclass damage detection and localization using long short-term memory networks. In Structures (Vol. 35, pp. 436-451). Elsevier.

## Contributions by the author of the thesis

The structural damage detection problem and the public datasets were brought into this collaborative work by the Civil Engineering researchers. They were also instrumental in positioning the work in the relevant literature on the topic of structural health monitoring within civil engineering, and writing relevant parts of the paper catering to a civil engineering audience.

As the second author, I made the following contributions. The percentages indicate the degree of my contribution for each segment.

- **Conceptualization (80%)**. Formulating the problem as a machine learning classification problem. Discussing and putting together the methodology (data pipeline, models, evaluation) needed for solving the problem.
- **Design and implementation (100%)**. Designing, implementing and testing the data pipeline with novel pre-processing and post-processing steps, the machine learning model architectures and the evaluation setup. Techniques in the current literature were also implemented for comparison with the novel methods.
- **Experimentation (100%)**. Running all experiments of the project, including comprehensive hyperparameter tuning of the models and variants of the implemented techniques.

- **Writing (50%).** Writing a significant part of the paper with a focus on the methodology section and results analysis. Reviewing and editing the paper for initial submission, and making revisions during peer review for final submission.
- **Collaboration (50%):** Carrying out continuous discussions with all authors on method design, development, experimentation, results analysis, and writing.

# Co-Authorship Statement for the Chapter: Contrastive Predictive Coding in Time-Series Classification

The chapter “Experiences with Contrastive Predictive Coding in Industrial Time-Series Classification” (Appendix E) is based on the following publication [18] resulting from research conducted during a DAAD internship at the ABB Corporate Research Center in Ladenburg, Germany.

- Gamage, Sunanda, Benjamin Klopper, and Jagath Samarabandu. "Experiences with contrastive predictive coding in industrial time-series classification." ACM SIGKDD Explorations Newsletter 24.2 (2022): 114-123.

## Contributions by the author of the thesis

The ABB research lab was working on the problem of label efficient classification for industrial process signals due to the unavailability of labeled data. I proposed self-supervised learning as a potential way of utilizing large amounts of unlabeled data, a solution that was not explored before by the team and in the literature on the application domain.

The second author, Dr. Benjamin Klopper was the project supervisor, and he carried out high-level research supervision (providing access to proprietary datasets and computational resources, discussing experiments and results, providing general research guidance).

As the first author, I made the following contributions. The percentages indicate the degree of my contribution for each segment.

- **Conceptualization (100%).** Formulating the problem as a self-supervised learning task in order to utilize unlabeled data.
- **Design and implementation (100%).** Designing, implementing and testing the data pipeline, the experimental setup, model architectures and training procedures needed for the self-supervised pre-training, supervised fine-tuning and evaluating label efficiency.
- **Experimentation (100%).** Running all experiments of the project, compiling and analyzing results.

- **Writing (95%).** Writing the full paper from initial draft through final revised publication. The other co-authors provided comments and suggestions during the peer review process.
- **Conference presentation (100%).** I made an oral and poster presentation of the paper at the AI for Manufacturing Workshop in the European Conference on Machine Learning 2022 in France.



## **Dedication**

*To my loving parents, my mother Rohini Pathirathna and my father Piyadasa Gamage.*

# Acknowledgments

I extend my deepest gratitude to my Ph.D. supervisor, Dr. Jagath Samarabandu for his support and guidance throughout my time at Western, and advising all research projects from conceptualization to publication. I thank Dr. Katarina Grolinger and Dr. Miriam Capretz, whose teachings and advice have helped shape the research presented in this thesis.

I am grateful to my internship supervisor Dr. Benjamin Klöpper for his support during my stay in Germany and advising me on the internship research project. I also thank my research collaborators, Dr. Ayan Sadhu and Dr. Sandeep Sony.

I am grateful to my family; parents, brothers and their families for their unconditional love and support that carry me forward.

I fondly remember and thank all the friends who have been part of my life and supported me throughout this time. Hasitha, Rashinda, Menaka, Vinura, Santiago, José, Cesar, Dimuthu, Praveen, Bhagya, Dhananjaya, Dhanushka, Kasun and Isuru to name a few.

I thank the graduate coordinators and staff of the Electrical and Computer Engineering Department, the Faculty of Engineering, and the School of Graduate and Postdoctoral Studies for their service to the graduate students of the university.

The computationally intensive research presented in this thesis was only made possible by the high performance compute clusters provided by the Digital Research Alliance of Canada and their partner organizations Calcul Québec and Compute Ontario. Thank you for making these resources available to Canadian academic researchers.

My research has been generously supported by several organizations. I thank the University of Western Ontario, the Government of Canada, the Government of Ontario, the Government of Germany, and the people of Canada and Germany for their generous support of scientific research.

## **Funding sources:**

- Western Graduate Research Scholarship (WGRS)
- Natural Sciences and Engineering Research Council of Canada (NSERC)

- Ontario Graduate Scholarship (OGS)
- DAAD RISE Professional internship grant (German Academic Exchange Service)

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Summary for Lay Audience</b>	<b>iii</b>
<b>Co-Authorship Statements</b>	<b>iv</b>
<b>Dedication</b>	<b>ix</b>
<b>Acknowledgments</b>	<b>x</b>
<b>Table of Contents</b>	<b>xii</b>
<b>List of Figures</b>	<b>xvii</b>
<b>List of Tables</b>	<b>xx</b>
<b>List of Appendices</b>	<b>xxii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Contributions of the thesis . . . . .	5
1.2.1 Attribution robustness of neural networks . . . . .	5
1.2.2 Applications of deep neural network classifiers . . . . .	7
1.3 Thesis organization . . . . .	8
<b>2 Attribution Robustness of Neural Networks: Background</b>	<b>10</b>
2.0.1 Attributions . . . . .	10
2.0.2 Usefulness of attributions . . . . .	11
2.1 Methods for computing attributions . . . . .	13
2.1.1 Simple gradients . . . . .	13
2.1.2 SmoothGrad . . . . .	14
2.1.3 Integrated gradients (IG) . . . . .	14

2.2	Attribution attacks . . . . .	15
2.2.1	Iterative Feature Importance Attacks (IFIA) . . . . .	16
2.2.2	Other attribution attacks . . . . .	16
2.3	Attribution robustness . . . . .	20
2.3.1	Why is attribution robustness important? . . . . .	20
2.4	Methods for training attribution robust neural networks . . . . .	22
2.4.1	Frameworks for attribution robust training . . . . .	22
2.4.2	Objective functions and techniques for attribution robust training . . . . .	23
2.5	Evaluating attribution robustness . . . . .	25
2.5.1	On the difficulty of benchmarking attribution robustness . . . . .	28
2.6	Properties of attribution robust models . . . . .	28
2.7	Research problems and gaps on attribution robustness . . . . .	29
<b>3</b>	<b>A Novel Framework and Attribution Robust Training Objective</b>	<b>31</b>
3.1	Contributions of the chapter . . . . .	31
3.2	Loss functions for attribution robustness: a framework . . . . .	32
3.2.1	Loss function instantiations . . . . .	35
3.2.2	The “Cross entropy of IG attacks” objective and training loop . . . . .	39
3.3	Attribution robustness experiments . . . . .	42
3.3.1	Experiment details . . . . .	42
3.3.2	Results and discussion . . . . .	44
3.4	Conclusion . . . . .	49
<b>4</b>	<b>Challenges of Robust Training and Methods for Efficient Training</b>	<b>50</b>
4.1	Contributions of the chapter . . . . .	51
4.2	Challenges in training attribution robust neural networks . . . . .	52
4.2.1	Computational cost of attribution robust training . . . . .	52
4.2.2	Trade-off between attribution robustness and accuracy . . . . .	53
4.2.3	Hyperparameter tuning . . . . .	55
4.3	Efficient training of attribution robust neural networks . . . . .	57
4.3.1	Strength of the attribution attack generated in the training loop . . . . .	57
4.3.2	Combining standard and robust training . . . . .	60
4.3.3	Tuning the integrated gradients steps parameter . . . . .	62
4.4	Conclusion . . . . .	63
<b>5</b>	<b>Properties of Attribution Robust Neural Networks</b>	<b>64</b>
5.1	Contributions of the chapter . . . . .	64

5.2	Robustness to image corruptions . . . . .	65
5.2.1	Experiment details . . . . .	65
5.2.2	Results and discussion . . . . .	67
5.3	Resilience to spurious correlations . . . . .	70
5.3.1	Experiment details . . . . .	70
5.3.2	Results and discussion . . . . .	71
5.4	Conclusion . . . . .	72
5.4.1	Potential avenues for exploration . . . . .	72
<b>6</b>	<b>Conclusion</b>	<b>74</b>
6.1	Summary of thesis contributions . . . . .	74
6.1.1	Attribution robustness of neural networks . . . . .	74
6.1.2	Applications of deep neural networks . . . . .	75
6.2	Impact and implications . . . . .	76
6.2.1	Impact of the research on attribution robustness . . . . .	76
6.2.2	Impact of the research on applications . . . . .	76
6.3	Future work . . . . .	77
	<b>Bibliography</b>	<b>79</b>
<b>A</b>	<b>Attribution Robustness Experiment Details</b>	<b>86</b>
A.1	IFIA attack generation hyperparameters . . . . .	86
A.2	Benchmark datasets with common corruptions and perturbations . . . . .	87
A.3	Datasets used for spurious correlations experiments . . . . .	87
<b>B</b>	<b>Deep Learning Methods in Network Intrusion Detection</b>	<b>89</b>
B.1	Introduction . . . . .	91
B.1.1	Related work . . . . .	93
B.1.2	Contributions . . . . .	95
B.2	Preliminary discussion . . . . .	96
B.2.1	The network intrusion detection problem . . . . .	96
B.2.2	Datasets . . . . .	96
B.2.3	Evaluation metrics . . . . .	97
B.3	Taxonomy and survey of deep learning methods for intrusion detection . . . . .	100
B.4	Empirical comparison . . . . .	106
B.4.1	Overview of experiments: models, datasets and evaluation metrics . . . . .	106
B.4.2	Details of the experiments . . . . .	108

B.4.3	Results and analysis . . . . .	114
B.5	Issues in current research and future directions . . . . .	125
B.6	Conclusion . . . . .	127
<b>C</b>	<b>Network Intrusion Detection Results</b>	<b>136</b>
C.1	Evaluation metrics . . . . .	136
C.2	Results of experiments . . . . .	137
<b>D</b>	<b>Structural Damage Detection and Localization</b>	<b>144</b>
D.1	Introduction . . . . .	146
D.1.1	Structural Health Monitoring (SHM) . . . . .	146
D.1.2	Machine learning methods in structural health monitoring . . . . .	147
D.1.3	Contributions and organization . . . . .	148
D.2	Preliminaries of LSTM networks . . . . .	148
D.3	Damage detection using LSTM model . . . . .	151
D.3.1	Data pipeline . . . . .	151
D.3.2	The model, training and inference . . . . .	152
D.3.3	Hyperparameter tuning . . . . .	153
D.3.4	Damage localization using LSTM model . . . . .	154
D.3.5	Performance evaluation criteria . . . . .	154
D.4	Experiments and results . . . . .	156
D.4.1	Experimental study . . . . .	157
D.4.2	Full-scale study . . . . .	162
D.5	Conclusion . . . . .	174
<b>E</b>	<b>Contrastive Predictive Coding in Time-Series Classification</b>	<b>177</b>
E.1	Introduction . . . . .	179
E.2	Related Work . . . . .	180
E.2.1	Semi-supervised learning . . . . .	181
E.2.2	Self-supervised learning (SSL) . . . . .	181
E.2.3	Active learning . . . . .	182
E.2.4	Combining SSL and Active Learning . . . . .	182
E.3	Contrastive Predictive Coding . . . . .	183
E.3.1	CPC architecture and self-supervision task . . . . .	183
E.3.2	CPC classifier . . . . .	184
E.4	Industrial Datasets and Classification Problems . . . . .	185
E.5	Proposed Classification Pipeline . . . . .	187

E.6	Experimental Results . . . . .	189
E.6.1	Initial experiments . . . . .	189
E.6.2	Label efficiency experiments . . . . .	191
E.6.3	Active learning experiments . . . . .	192
E.7	Conclusion . . . . .	194
	<b>Curriculum Vitae</b>	<b>198</b>



# List of Figures

1.1	Interpretability as understanding the decision making process of an ML system	2
1.2	Visualization of an attribution attack or perturbation. . . . .	3
2.1	Illustration of attribution method or estimator . . . . .	11
2.2	Heatmaps of attributions obtained by some of the methods described . . . . .	15
2.3	IFIA attacks and their heatmaps in comparison to original samples and heatmaps	18
2.4	Projected gradient descent (PGD) optimization loop searching for an attribution attack. . . . .	18
2.5	Targeted attribution attack . . . . .	19
2.6	IFIA attacks are ineffective against models trained to have attribution robustness	21
2.7	The process of evaluating attribution robustness of a trained model. . . . .	27
3.1	Explanation of the terms in the proposed robust loss functions framework . . . .	34
3.2	Attribution robustness of the three training objectives on four setups . . . . .	45
3.3	Evolution of attribution robustness and test accuracy during training . . . . .	48
4.1	Trade-off between attribution robustness and test accuracy . . . . .	55
4.2	IG attack strength through the attack loop for 3 data batches (one-step vs 10-step attacks). . . . .	59
4.3	Attribution robustness and accuracy change with no. of iterations in the attack loop . . . . .	60
4.4	Switching the training objective during training: all-epochs-robust vs. last-few-epochs-robust . . . . .	62
5.1	Visualization of corruptions in RIMB-C and MNIST-C datasets . . . . .	67
5.2	Overall accuracy difference (Acc_Diff%) of robust models compared to standard models . . . . .	68
5.3	Distribution of accuracy difference of robust models compared to standard models	68
A.1	Waterbirds and CelebA dataset details and visualizations . . . . .	88
B.1	Percentage of large enterprises with reported attacks . . . . .	93

B.2	Taxonomy of deep learning models in intrusion detection . . . . .	100
B.3	Accuracy, precision and F1-score of the models on each dataset . . . . .	116
B.4	False positive and false negative rates of the models on each dataset . . . . .	118
B.5	Training epochs and training time to achieve the average goal accuracy of a model	119
B.6	Results of training semi-supervised models AE + ANN and DBN + ANN . . . . .	121
B.7	Validation loss history during the supervised training stage . . . . .	122
B.8	Accuracy and F1-score on IDS 2017 and IDS 2018 datasets . . . . .	123
B.9	Evaluation metrics from shallow ANN models with varying training set sizes . . . . .	124
D.1	The typical internal structure of an LSTM cell . . . . .	149
D.2	Data pipelines for training the LSTM network and obtaining predictions for a given acceleration response. . . . .	152
D.3	Extracting sequences of windows from the vibration signals and the LSTM network architecture. . . . .	153
D.4	QUGS testbed . . . . .	158
D.5	Performance evaluation of LSTM based on window size for QUGS. . . . .	160
D.6	Performance evaluation of 5 random weight initializations for QUGS . . . . .	160
D.7	Performance of the LSTM model on the QUGS data . . . . .	162
D.8	Damage localization probabilities in QUGS . . . . .	163
D.9	Schematic of the Z24 bridge. . . . .	164
D.10	Sensor placement for the data acquisition. . . . .	164
D.11	Performance evaluation of LSTM based on window size . . . . .	166
D.12	Performance evaluation of LSTM for random weight initialization . . . . .	167
D.13	Confusion matrices for the pier settlement damage problem . . . . .	168
D.14	Confusion matrices for the rupture of tendon damage problem . . . . .	169
D.15	Performance of the proposed LSTM model in the Z24 bridge for pier settlement	170
D.16	Performance of the proposed LSTM model by windowing the data of the Z24 bridge . . . . .	171
D.17	Damage localization for lowering of pier for three damage levels . . . . .	172
D.18	Damage localization for lowering of pier, where legend shows the amount of pier-settlement. . . . .	173
D.19	Damage localization for rupture of tendons, where, the legend shows number of tendon ruptures. . . . .	173
E.1	Training and testing pipeline for time-series classification with self-supervised learning. . . . .	181
E.2	Contrastive predictive coding neural architecture of the <i>CPC_Full_SSL_Model</i> . . . . .	184

E.3	Nominal behaviour of a batch in the simulated dataset . . . . .	186
E.4	UMAP visualization of CPC embeddings on the test sets . . . . .	189
E.5	Label efficiency curves of different types of training of the model . . . . .	191
E.6	Active learning curves of on two datasets . . . . .	193

# List of Tables

3.1	Attribution robustness of different training objectives . . . . .	44
4.1	Computational cost of training objectives . . . . .	53
4.2	Time taken per one epoch of training for different objectives . . . . .	54
4.3	Combined training results (standard and CE-of-attacks [CEA]). . . . .	61
4.4	Tuning the IG steps parameter ( $m_a$ ) in the CE-of-attacks objective. . . . .	63
5.1	Benchmark image datasets containing corruptions and perturbations . . . . .	66
5.2	Overall accuracy difference (Acc_Diff%) and improvement percent (Acc_Impro) of robust models compared to standard models on the corrupt datasets. . . . .	69
5.3	Datasets used in spurious correlations experiments. Worst-accuracy group is highlighted in bold. . . . .	71
5.4	Accuracy % of different training objectives on spurious correlations datasets. . .	72
A.1	IFIA attack generation hyperparameters . . . . .	86
A.2	Benchmark image datasets containing corruptions . . . . .	87
B.1	Summary of dataset characteristics. . . . .	97
B.2	Attack types present in the datasets. . . . .	98
B.3	Confusion matrix for a binary class problem . . . . .	98
B.4	List of survey papers on machine learning methods for intrusion detection . . .	102
B.5	List of papers on supervised instance classification models for intrusion detection	103
B.6	List of papers on supervised sequence classification models for intrusion detection	104
B.7	List of papers on semi-supervised instance classification models for intrusion detection . . . . .	105
B.8	List of research papers on transfer learning models for intrusion detection . . .	106
B.9	Hyper-parameter configurations/ ranges used in the tuning process . . . . .	110
B.10	Hyper-parameter configurations selected from the tuning process . . . . .	113
B.11	Comparison of model performance with prior works . . . . .	117
B.12	Comparing the performance of semi-supervised models with supervised ANN .	121

B.13	p-values of the 10-fold cross validation <i>t</i> -tests comparing ANNs with semi-supervised models. . . . .	122
C.1	Overall results of the models on the KDD 99 test set . . . . .	138
C.2	Overall results of the models on the NSL KDD test set . . . . .	138
C.3	Overall results of the models on the IDS 2017 test set . . . . .	139
C.4	Overall results of the models on the IDS 2018 test set . . . . .	139
C.5	Nemenyi comparison test on the accuracy figures of models . . . . .	140
C.6	Performance of shallow and wide neural network architectures . . . . .	140
C.7	Performance of deep neural network architectures . . . . .	141
C.8	Performance of narrowing deep neural network architectures . . . . .	142
C.9	Evaluation metrics from shallow ANN models trained on increasingly large subsets of the IDS 2017 and IDS 2018 datasets. . . . .	143
D.1	Confusion matrix for a binary damage classification problem. . . . .	155
D.2	Description of various performance metrics. . . . .	156
D.3	Hyperparameters used in LSTM for tuning by random search algorithm. . . . .	159
D.4	Optimal configuration of LSTM hyperparameters for QUGS experiment. . . . .	159
D.5	Multiclass classification problem description for two damage scenarios. . . . .	165
D.6	Optimal configuration of LSTM hyperparameters for the Z24 bridge dataset. . . . .	166
E.1	Dataset characteristics . . . . .	187
E.2	Classification accuracy on the three datasets (trained on all labeled data). . . . .	190

# List of Appendices

Appendix A Attribution Robustness Experiment Details . . . . .	86
Appendix B Deep Learning Methods in Network Intrusion Detection: a Survey and an Objective Comparison . . . . .	89
Appendix C Network Intrusion Detection Results . . . . .	136
Appendix D Structural Damage Detection and Localization using Neural Networks on Vibration Signals . . . . .	144
Appendix E Experiences with Contrastive Predictive Coding in Industrial Time-Series Classification . . . . .	177

# Chapter 1

## Introduction

### 1.1 Motivation

Deep neural networks have become increasingly prevalent in diverse fields, offering new solutions to complex problems. They provide state-of-the-art results in various problem domains such as machine vision (eg: classification and segmentation [40], video understanding [53]), speech (eg: speech-to-text [43]), natural language processing (eg: generative AI [7]), and natural sciences (eg: protein folding [27]). The ability of deep neural networks to learn from large datasets has enabled engineers to develop solutions for problems previously handled by expert-designed rule-based systems.

While deep neural networks excel in solving a variety of tasks, a significant challenge they pose is the lack of interpretability or the inability to understand the decision making process of a model [76]. Unlike rule-based systems designed and written by humans, neural networks comprise large weight matrices, effectively making them complex black boxes with no easy way to understand the rationale behind their predictions. This is the central consideration of the field of **neural network interpretability** [11]. It encompasses techniques for assessing, understanding and modifying or refining a neural network's decision making process [21].

The **topic of explainability** can be considered a sub field of the broader literature on interpretability [49]. It addresses the problem of providing an explanation for a model's prediction of a given input in terms of feature attributions (also known as feature significance, feature relevance or saliency) [41]. A high-level view of machine learning interpretability and ex-

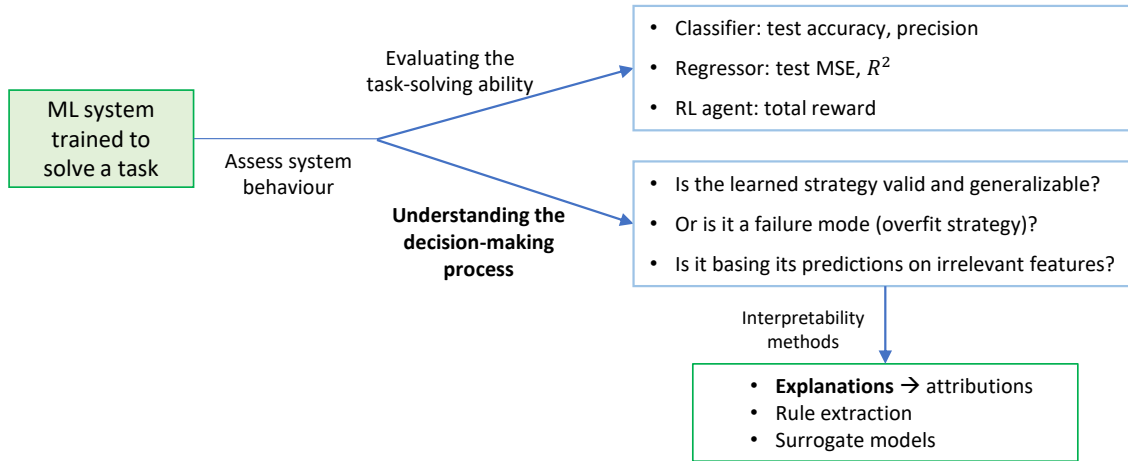


Figure 1.1: Interpretability as understanding the decision making process of a machine learning system, complementing a direct assessment of its task-solving ability.

plainability as methodology for assessing and understanding the decision making process of models is illustrated in Figure 1.1.

Intuitively, attributions represent the features of an input example that the model considers to be important for the prediction. For this reason, feature attributions produced by a model for a specific input are termed the model’s “explanation”. Attributions or explanations are useful outputs of a machine learning system. For example, they can promote human-machine collaborative decision-making systems by allowing human experts to inspect and understand which features or regions of an input are significant for a model prediction [55]. Attributions can also aid in identifying failure modes (eg: spurious correlations [59]) and debugging of trained models [6]. Several works have also explored the possibility of utilizing attribution maps in an automated manner to guide and improve the model training process [77, 72]. More broadly, coherent and well-aligned attributions or explanations of model predictions help establish trust and compliance of machine learning systems deployed in critical application settings such as healthcare [38, 75].

### The problem of fragility and robustness of attributions

Recent studies have exposed a critical vulnerability in attributions produced by neural networks: they are fragile and susceptible to manipulation [20, 9]. It is possible to find small, often imperceptible changes or perturbations to the input that can drastically alter the attributions, misleading the explanations given by the model. The perturbed examples are termed



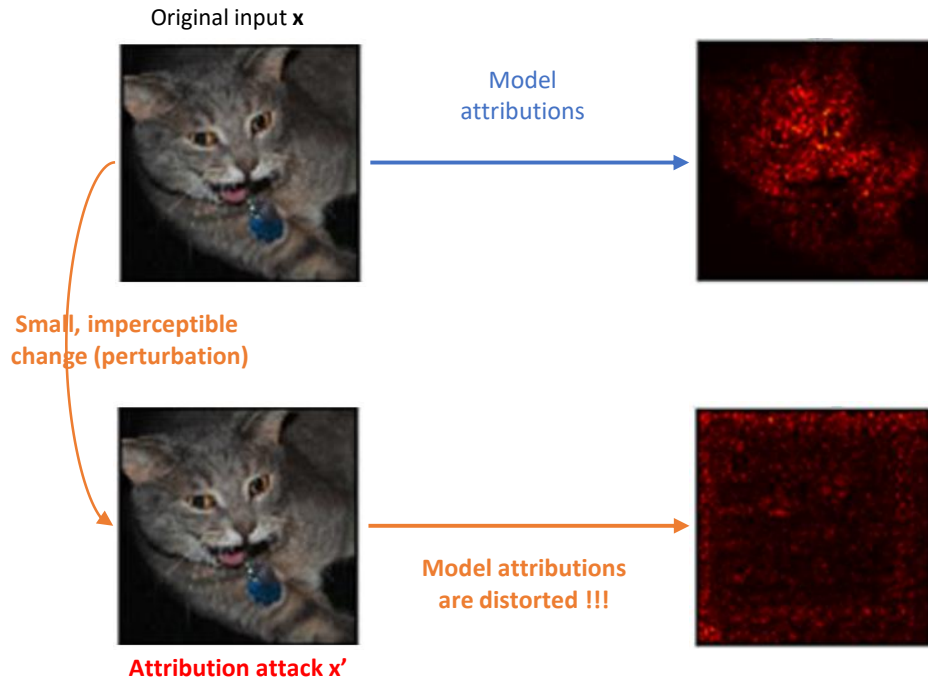


Figure 1.2: Visualization of an attribution attack or perturbation.

“**attribution attacks**”, and they are found by an iterative optimization algorithm such as projected gradient descent, which perturbs a natural example in the direction of maximal change attributions. Figure 1.2 illustrates an attribution attack that is visually similar to the natural example, but produces distorted attributions.

The fragility of neural network attributions makes them unreliable and untrustworthy as a tool of explanation in any setting where they are utilized, either for human expert inspection to support decision making or automated model improvement [13, 8]. The existence of such adversarial perturbations also reveals a failure mode of neural networks. These issues highlight the need for neural network models to not only be accurate but also have robust and reliable attributions [54, 58].

“**Attribution robustness**” of a model refers to the property that attributions or input feature significance scores produced by the model for a given input cannot be easily manipulated or distorted by imperceptible perturbations to the input [5].

Training attribution robust neural networks is typically done by modifying the standard loss function (eg: standard cross entropy for classification) to include a term that enforces a robustness constraint, resulting in robust training objectives. This requires computing an attribution

attack example (a perturbed input) for each natural example during training. For example, one robust objective termed “IG-Norm” [5] computes attribution attack examples by maximizing the distortion in integrated gradients between natural and perturbed versions. The robust objective is then created by adding to the cross entropy a regularization term that measures this maximal distortion. Attribution robust training objectives take the form of a min-max problem where the outer minimization of the loss function is performed with gradient descent (for neural net weight updates), and the loss function contains an inner maximization loop that computes the attribution attacks.

A major problem with existing methods for training attribution robust neural networks is their significantly high computational cost. For example, the IG-Norm robust objective [5] was measured to be up to 67 times slower than standard training, which is prohibitively expensive for large scale training. Furthermore, frameworks for deriving attribution robust objectives in the existing literature [5, 25] are designed to be too restrictive by specifying a given attribution computation method and coupling the inner attack generation to the regularization terms in the loss function. In addition to these research problems, the existing literature contains no investigation or information of the properties of neural networks trained with attribution robust training objectives.

**This thesis is a study of the problem of attribution robust training of neural networks.**

In this work, a novel, generic and flexible framework of loss functions is proposed, which allows for derivations of all well-known robust objectives as well as a new class of robust objectives. A new, simple and efficient attribution robust objective is proposed, with extensive empirical evidence demonstrating its effectiveness. A compilation of techniques are proposed to further improve the efficiency of robust training objectives. Furthermore, we investigate two key properties of attribution robust models: their robustness to common data corruptions and resilience in the presence of spurious correlations. The techniques and empirical results presented in this work will facilitate large scale training of attribution robust neural networks.

### **Applications of deep neural network classifiers**

This thesis also presents research on classification pipelines utilizing deep neural networks in multiple application domains. First, we conduct a comprehensive analysis of neural network classifiers for solving the problem of computer network intrusion detection. The second application is in structural health monitoring, where a novel data pipeline and a Long Short-Term

Memory (LSTM) architecture is used to detect damages in physical structures by analyzing vibration sensor data. The third application is in industrial process signal classification, where multi-variate sensor signals are used to identify various process states in a label efficient manner using self-supervised learning.

## 1.2 Contributions of the thesis

This section summarizes the key contributions of the thesis on both the central topic of attribution robustness of neural networks and the applications of deep neural network classifiers.

### 1.2.1 Attribution robustness of neural networks

#### **A novel framework of loss functions for attribution robust training**

We introduce a novel framework of loss functions as a principled approach to the problem of attribution robust training of neural networks. This framework is unique in its decoupling of the attribution attack generation from the regularization terms in the loss function and includes a cross-entropy of attacks term. It also offers flexibility in the choice of attribution methods and similarity functions that can be used to generate attribution attacks during training. To demonstrate the generalization capability of the proposed framework, we show that all well-known attribution robust objectives can be derived from it. In addition, derivations of several novel robust objectives that leverage the cross entropy of attacks-term are also provided.

#### **A novel attribution robust training objective: the “cross entropy of attacks”**

From the proposed framework of loss functions, a simple yet novel objective function termed “cross entropy of attacks” (CE-of-attacks) is instantiated. This method computes a batch of attribution attacks for each mini-batch of original data during training, and uses the cross entropy of model predictions on the attack batch as the loss value for gradient descent optimization of network weights.

Comprehensive empirical evidence on the performance of the CE-of-attacks objective for train-

ing convolutional neural networks (CNNs) on image datasets demonstrates that it achieves higher attribution robustness and accuracy than IG-Norm, a well-known robust objective [5]. As an example, a small 4-layer CNN trained on the CIFAR-10 dataset gives 72.1% accuracy with the CE-of-attacks objective compared to 70.7% accuracy with the IG-Norm objective, while achieving a 68.4% top-100 intersection (a metric of attribution robustness) compared to the 64.9% top-100 intersection of the IG-Norm objective.

More importantly, the proposed CE-of-attacks robust objective is significantly more efficient than existing robust loss functions (2 times to 10 times faster neural net training runs compared to IG-Norm), enabling large scale training of attribution robust neural networks.

### **A detailed analysis of the challenges of attribution robust training**

Attribution robust training of neural networks poses several unique challenges over standard training, an under-explored area in the literature on attribution robustness. We present a detailed analysis of three key issues: computational cost, the trade-off between attribution robustness and accuracy, and the difficulty of hyperparameter tuning.

Algorithmic and empirical analysis of computational cost of different attribution robust training objectives shows that robust training can be 4 to 67 times slower than standard loss functions. Attribution robust training with varying distance budgets for the generated attacks reveals that a 17.5% gain in top-100 intersection (a metric of attribution robustness) on the CIFAR-10 dataset with a small CNN results in an accuracy drop of 7.8%, highlighting the trade-off between the two properties. These two issues and the larger hyperparameter space make tuning of hyperparameters difficult. Techniques and guidelines are proposed to mitigate the issues.

### **Techniques for efficient attribution robust training of neural networks**

Several techniques are proposed to address the key issue of high computational cost of attribution robust training. They are: combining standard and robust training (2x - 4x speed gain), using a fast one-step attack in the training loop (3x or more speed gain), and tuning the number of steps in the integrated gradients computations (2x - 6x speed gain). Experimental results validate that the proposed techniques achieve significant efficiency gains with no performance degradation in terms of attribution robustness and accuracy.

## Properties of attribution robust neural networks

The goal of attribution robust training is to achieve a level of robustness in the input feature significance space. As a result, robust neural networks may demonstrate other desirable properties, a topic not addressed in the existing literature. We investigate two important properties of robust models.

We show that attribution robust neural networks are also robust against common image corruptions and perturbations (diverse noise types, weather effects, digital artifacts etc.) compared to well-trained standard neural nets. Robust models achieve accuracy gains in the range of 3.58% to 11.94% over standard trained models on three corrupt versions of popular image datasets: MNIST-C, CIFAR-10-C and ImageNet-C.

Prior work has hypothesized that attribution robust neural networks may be resilient to spurious correlations in training data [5]. We provide experimental evidence that this desirable property does not exist in robust models.

## Open source implementation

All code and artifacts needed to reproduce and build on our work on training attribution robust neural networks are made open source<sup>1</sup>

## 1.2.2 Applications of deep neural network classifiers

### A comprehensive survey and empirical benchmark of computer network intrusion detection with neural networks

We introduce a taxonomy of deep learning methods for intrusion detection, providing a structured overview of the current literature. Methods are categorized into supervised instance-based and sequence-based learning, and semi-supervised instance-based learning. A comprehensive benchmark is proposed to capture a range of deep learning models and datasets: both legacy small datasets and modern large datasets. Four major deep learning models (feed-

---

<sup>1</sup>[https://github.com/rcsunanda/attribution\\_robustness](https://github.com/rcsunanda/attribution_robustness)

forward neural networks, autoencoders, deep belief networks, and long short-term memory (LSTM) networks) are evaluated on this benchmark.

### **A novel pipeline with a LSTM network for damage detection in physical structures**

We propose a novel pipeline that utilizes a Long Short-Term Memory (LSTM) network for detecting structural damage using vibration signals. The input to the LSTM model is a sequence of sample windows, and a voting mechanism is used at the output to classify an entire vibration signal. This method facilitates data augmentation for cases with limited training data, and the voting mechanism significantly improves classification performance.

We propose a damage localization technique, where the damage class probabilities of vibration signals coming from different places in a physical structure are used to localize the damage. The system is evaluated and validated on two benchmark setups in the structural health monitoring domain: a physical simulation dataset and a real-world bridge dataset.

### **Using self-supervised learning methods for label efficient classification of industrial process signals**

We propose a system for label efficient classification of multi-variate time-series data in the industrial process domain. For this work, we utilize a self-supervised learning method called Contrastive Predictive Coding (a hybrid 1-dimensional CNN and LSTM architecture and a prediction pre-training task). The system is evaluated on public and proprietary process datasets demonstrating very high label efficiency (high classification accuracy with only 10's or 100's of labeled examples).

## **1.3 Thesis organization**

Chapter 2 introduces the reader to the concepts of attributions, attribution attacks, and the topic of attribution robustness. A comprehensive literature survey is conducted on the relevant topics with a focus on methods for attribution robust training of neural networks. Chapter 3 presents the novel framework of loss functions for attribution robust training, and the proposed robust

objective ‘CE-of-attacks’. Comprehensive experimental results evaluating the performance of the proposed methods are presented.

Chapter 4 presents the detailed analysis of three key challenges of robust training: computational cost, the trade-off between attribution robustness and accuracy, and the difficulty of hyperparameter tuning. Guidelines for mitigating the issues and conducting effective attribution robust training are discussed. Chapter 5 investigates the properties of attribution robust models, namely robustness to image corruptions and resilience to spurious correlations.

Chapter 6 concludes the study on attribution robustness of neural networks with a summary of the proposed novel methods, and provides concrete ideas for future directions based on this work.

The research on applications of deep neural network classifiers is presented in the appendices. Appendix B contains the survey and empirical benchmark of computer network intrusion detection with neural networks. Appendix D presents the work on damage detection in physical structures using neural networks. Appendix E contains the self-supervised learning based system for label efficient classification of industrial process signals.

**A note on thesis format.** This thesis comprises unpublished work on attribution robustness of neural networks (chapters 2 - 5) as well as modified manuscripts of the author’s publications (appendices B, D, E). Therefore, the thesis takes the form of a hybrid between the integrated articles format and a monograph.

# Chapter 2

## Attribution Robustness of Neural Networks: Background

In this chapter, the reader is first introduced to the preliminary notions of input attributions produced by a neural network classifier, attribution methods, fragility of attributions and attribution attacks. The research problem of attribution robustness is then discussed with a comprehensive review of the literature on methods for training attribution robust neural networks. Finally, several important research gaps on attribution robustness are identified for the benefit of the researchers of this topic.

### 2.0.1 Attributions

Among the many techniques in the explainability literature, feature attributions (also known as feature significance, feature importance, feature relevance, saliency or explanations) are a key class of methods used to explain predictions of machine learning models [41]. Attributions represent the degree of significance that a model places on each feature of a given input for making the prediction.

In the classification setting, attributions can be formally defined as follows (illustrated in Figure 2.1). Consider a model parameterized by  $\theta$  so that the model function  $f_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^C$  maps an input vector of dimensionality  $d$  into  $C$  categories (classes). The predicted class for an input  $x \in \mathbb{R}^d$  is then given by  $k = \operatorname{argmax}_j f_\theta(x)_j$ . An attribution method or an estimator is a function



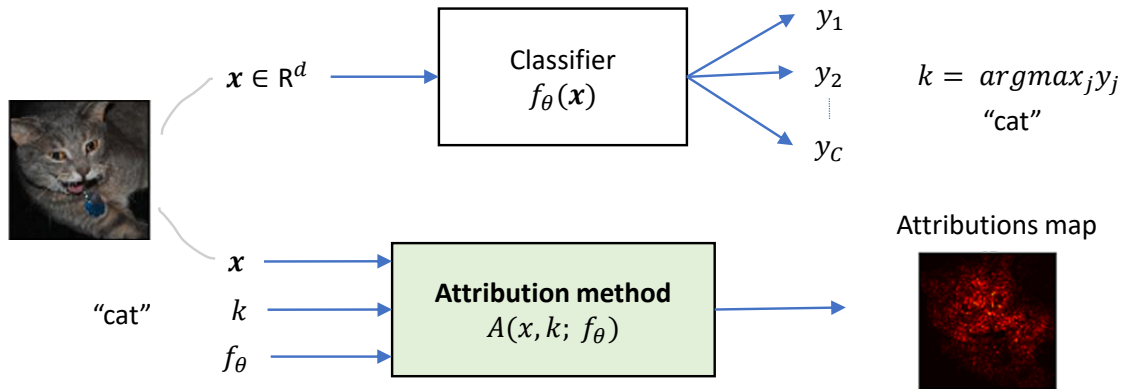


Figure 2.1: Illustration of attribution method or estimator

$A : \mathbb{R}^d \rightarrow \mathbb{R}^d$  that associates each feature of the input  $x \in \mathbb{R}^d$  to the significance that it has on the model prediction  $k$ . Note that the attributions vector has the same dimension  $\mathbb{R}^d$  as the input vector, and the attribution method  $A(x, k; f_\theta)$  is a function of the input  $x$ , predicted class  $k$  and the parameterized model  $f_\theta$ . i.e. attributions are produced by the attribution method for a given input, for the predicted class from the model.

## 2.0.2 Usefulness of attributions

**Attributions as a complementary output of a model.** The primary output of a machine learning model is its prediction relevant to the task. Feature attributions can be considered an additional output of a machine learning system for a given input, complementing the primary model output. Feature attributions can be visualized (eg: as heatmaps) so that a domain expert can inspect it to understand which features of the input the model is basing its prediction on or which features the model considers to be important. For example, a medical expert can look at the attributions heatmap of an X-Ray image classification of a model and spot which regions of the image have contributed more to the model prediction. Such information can be invaluable to the human expert in making informed decisions [67].

**Identifying failure modes and debugging a model.** Feature attributions can help us identify potential failure modes of a machine learning model. For example, inspecting attribution heatmaps for certain inputs could reveal that irrelevant features have high attribution scores (eg: edges of an image, artifacts such as copyright text on an image). This is an example of a model learning spurious correlations. We can then take corrective action based on this information to avoid the failure modes and train better models (eg: remove training samples with artifacts) [2,

59].

**Establishing causal relationships.** Feature attributions can aid in establishing causal relationships between model inputs and outputs. This is especially useful for machine learning models trained in domains with complex physical, chemical, or biological systems and massive datasets. In such scenarios, deriving analytical methods for the modeled data domain may be challenging. In these instances, feature attributions serve as a starting point for developing insights into experiments and discerning causal relationships within the data domain [4].

**Trust and compliance.** In a broad sense, feature attributions, when deployed as part of a machine learning system contribute to developing reliability and trust, and they help in building systems within ethical and legal constraints [39]. In some cases, explanations associated with a model prediction may be a legal requirement (eg: EU General Data Protection Regulation [44], EU Artificial Intelligence Act [45]).

**Improving the model training process.** The previous use cases are based on manual inspection of attribution maps. This can be error prone, as highlighted in critiques on the reliability and use of attribution methods for making critical decisions [34]. It is possible to incorporate attributions into the model training process with the aim of improving the models, which is an automatic or algorithmic way of using feature attributions (no manual inspection). An excellent review on this topic is found in [69]. Several concrete examples from the literature are briefly discussed below.

In [48], the authors modify the training loss function of neural networks to encourage certain irrelevant areas of the input (either marked by a human expert, or obtained in an unsupervised manner) to have smaller gradient attributions. The resulting trained models have been shown to have qualitatively different decision boundaries and higher out-of-distribution generalization performance than standard trained models. Another work [16] adds high-level priors such as smoothness and sparsity requirements to the training objective using a novel attribution method called expected gradients (an approximation of integrated gradients). Results on image and health care datasets show high generalization performance and robustness to noise. Closer to the central topic of this thesis, attributions can be incorporated into the training loss to improve attribution robustness [5, 25], noise robustness [58] and prediction or adversarial robustness [17].

## 2.1 Methods for computing attributions

Methods for computing attributions have been a central research problem in the literature on machine learning explainability. In this section, we introduce several gradient based methods [1] that can be used to generate attributions for neural networks. For a comprehensive review of attribution methods, we refer the reader to [41]. Figure 2.2 shows a comparison of attribution heatmaps obtained by some of the attribution methods.

### 2.1.1 Simple gradients

A simple technique to generate attributions is taking the gradient of the model output with respect to the input features [74]. Let  $\mathbf{x}$  be the input feature vector of an example,  $x_i$  be each input feature,  $f(\mathbf{x})$  be the model output for  $\mathbf{x}$  (eg: class logit or softmax value) and  $A_i$  be the attribution value for each feature  $x_i$  of the input. The simple gradient attributions can be written as in Equation 2.1.

$$A_i(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x_i} \quad (2.1)$$

Gradients represent the sensitivity of the model function to each input feature. For this reason, this technique is also called sensitivity analysis. Intuitively, a well-trained model must be more sensitive to core features (eg: pixels on an object under classification), and less sensitive to unimportant features (eg: background pixels).

One advantage of simple gradients is that it is very easy to compute when the model function  $f$  is differentiable: for neural networks, a backpropagation all the way to the input layer gives the gradient attributions. However, a downside is the noisy nature of gradient attributions, particularly for images, where they tend to highlight seemingly random pixels to the human eye.

### 2.1.2 SmoothGrad

SmoothGrad attributions method [61] was proposed as a solution to the problem of the noisy nature of gradients. In this technique,  $n$  noisy samples of the input image  $x$  are created by adding Gaussian noise ( $x^j = x + N(0, \sigma^2)$ ), and the average of gradients for each sample is taken as the final set of attributions (Equation 2.2).

$$A_i(\mathbf{x}) = \frac{1}{n} \sum_{j=1}^n \frac{\partial f(\mathbf{x} + N(0, \sigma^2))}{\partial x_i} \quad (2.2)$$

### 2.1.3 Integrated gradients (IG)

Integrated gradients (IG) [64] computes the path integral between a baseline input  $\mathbf{x}'$  and the given input example  $\mathbf{x}$ . Unlike other gradient-based attribution methods that solely rely on local gradients at a single point, IG integrates the gradient information over this path. Intuitively, it captures how much each feature contributes to the change in the function when the input varies from the baseline to the given input example.

The continuous path integral is shown in Equation 2.3. Practically, the discrete approximation in Equation 2.4 is used in all IG computations.

The choice of baseline can be made in a context-dependent manner. To compute attributions for an image, the zero baseline ( $\mathbf{x}' = \mathbf{0}$ vector) (all black pixels) is suitable. It is also important to select a suitable number of steps  $m$  for the discrete approximation: the authors of [64] suggest a number between 20 - 300. Depending on the application, this number needs to be found experimentally (trade off between efficiency and approximation accuracy).

$$A_i(\mathbf{x}) = (x_i - x'_i) \times \int_{\alpha=0}^1 \frac{\partial f(\mathbf{x}' + \alpha \cdot (\mathbf{x} - \mathbf{x}'))}{\partial x_i} d\alpha \quad (2.3)$$

$$A_i(\mathbf{x}) \approx (x_i - x'_i) \times \sum_{k=1}^m \frac{1}{m} \cdot \frac{\partial f(\mathbf{x}' + \frac{k}{m} \cdot (\mathbf{x} - \mathbf{x}'))}{\partial x_i} \quad (2.4)$$

In our work on attribution robust training methods, we use integrated gradients (IG) as the



Figure 2.2: Heatmaps of attributions obtained by some of the methods described [56]

preferred attribution computation method due to several reasons. First, IG offers a choice and good balance between the quality of attributions (less noise) and efficiency via the tunable parameter  $m$  ( $m = 4$  to  $m = 50$  depending on the location of the calculation in the training loop). Second, in our work, we often have to compute the difference between attributions of two input examples  $x_1, x_2$  (eg: natural and attack example). This can be done efficiently using the property in Equation 2.5, with only one IG computation (calculating IG of  $x_1$  with  $x_2$  as the baseline of the path integral). Finally, in nearly all our experiments, integrated gradients produce better results in comparison to other gradient based methods.

$$\begin{aligned} \text{Attribution difference}(x_1, x_2) &= IG(x_1, x_{baseline}) - IG(x_2, x_{baseline}) \\ &= IG(x_1, x_2) \end{aligned} \tag{2.5}$$

## 2.2 Attribution attacks

The existence and ability to generate attacks (perturbations) for a given input and machine learning model have been well studied research problems [3]. The most prominent category of attacks, known as adversarial attacks, involves perturbations to an input that alter the model’s prediction from the original (fooling the model prediction). Therefore, they are also called prediction attacks or prediction adversaries. The attack generation process targets the decision boundary of a model.

In [20], the authors proposed a different category of attacks. The perturbations of this attack category are generated to drastically change the attributions yielded by a machine learning model for a given input, while preserving the model predictions. (In contrast to the classic

adversarial or prediction attacks, which change the model prediction). This category of attacks can be broadly termed “attribution attacks”, as they aim to break (drastically change) the attributions of an input. Examples of attribution attacks are illustrated in Figure 2.3 and Figure 2.5.

### 2.2.1 Iterative Feature Importance Attacks (IFIA)

The optimization problem to solve for generating an attribution attack is shown in Equation 2.6 [20]. Given an attribution method  $A(\cdot)$  and a difference (dissimilarity) function  $Diff()$ , it attempts to find the perturbation  $\delta$  that maximally changes the attributions within a distance budget  $\epsilon$  (i.e.  $\|\delta\|_\infty \leq \epsilon$ ) while preserving the prediction of the original input.

$$\begin{aligned} & \underset{\delta}{\operatorname{argmax}} \operatorname{Diff}(A(x), A(x + \delta)) \\ & \text{subject to: } \|\delta\|_\infty \leq \epsilon, \\ & \operatorname{Prediction}(x + \delta) = \operatorname{Prediction}(x) \end{aligned} \tag{2.6}$$

The authors of [20] propose an algorithm to solve the optimization problem (Equation 2.6) in a projected gradient descent (PGD) loop and generate attribution attacks termed “Iterative Feature Importance Attacks” (IFIA). The process of a PGD loop searching for an attack within a small distance to the original example is illustrated in Figure 2.4.

Algorithm 1 describes a variant of the IFIA algorithm where we use integrated gradients (IG) as the attribution method, and the negative sum of attributions of  $k$  highest-attribution pixels as the difference function. This is called a “tok- $k$  IFIA attack” as it seeks to find a perturbation that maximizes the difference in attribution maps in the  $k$  initially most important pixels.

Figure 2.3 shows examples of IFIA attacks, and how they distort the attributions yielded by a neural network.

### 2.2.2 Other attribution attacks

Following the first work of [20] on the existence of attribution attacks, several more works have explored different types of attribution attacks.

---

**Algorithm 1** The top-k IFIA attack algorithm (Equation 2.6)

---

**Input:**

Input image  $\mathbf{x}$   
 Trained neural network

**Hyperparameters:**

No. of iterations ( $p$ ), step size ( $\alpha$ ), distance budget ( $\epsilon$ ), IG steps ( $m$ ), top-k pixels ( $k$ )

- 1:  $A \leftarrow IG(\mathbf{x}, ig\_steps = m)$
- 2:  $B \leftarrow$  subset of  $k$  pixels with highest  $A(\mathbf{x})$  values
- 3:  $\mathbf{x}' \leftarrow \mathbf{x}$

**Attack generation loop (PGD):**

- 4: **for**  $j \leftarrow 1$  to  $p$  **do**
- 5:    $A \leftarrow IG(\mathbf{x}', ig\_steps = m)$
- 6:    $D \leftarrow -\sum_{p \in B} A_p$     // *Difference function*
- 7:    $grad \leftarrow gradient(D, \mathbf{x}')$     // *Gradient of D w.r.t.  $\mathbf{x}'$*
- 8:    $\mathbf{x}' \leftarrow \mathbf{x}' + \alpha \cdot sign(grad)$     // *Gradient ascent update in PGD*
- 9:    $\mathbf{x}' \leftarrow clamp(\mathbf{x}', \epsilon)$     // *Projection in PGD*
- 10:   **if** Prediction( $\mathbf{x}'$ ) = Prediction( $\mathbf{x}$ ) **then**
- 11:      $attack = \mathbf{x}'$
- 12:   **end if**
- 13: **end for**

**Output:**

IFIA attribution attack:  $attack$

---

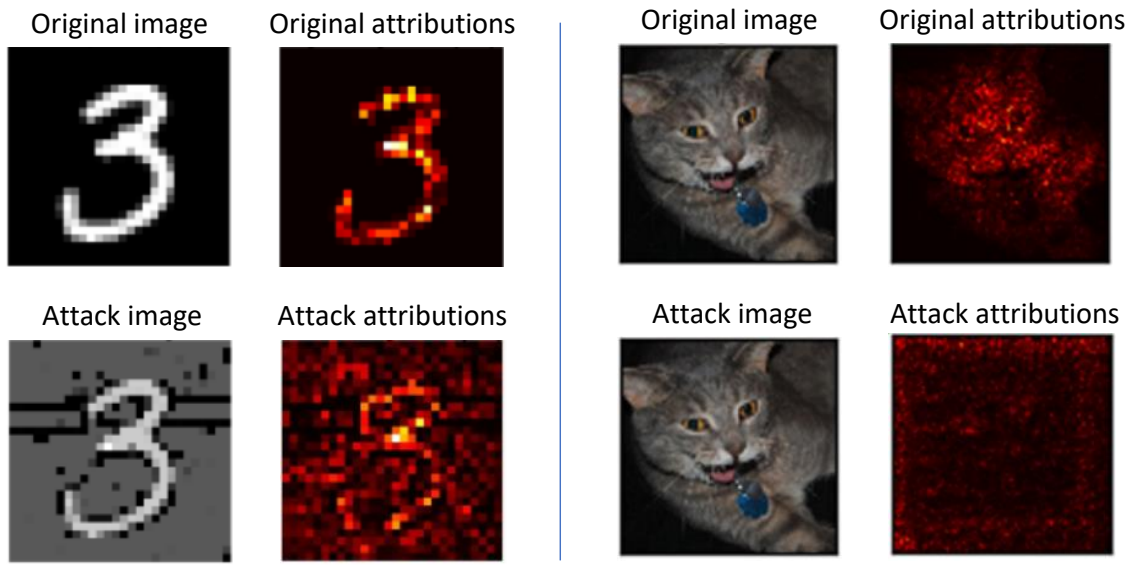


Figure 2.3: IFIA attacks and their heatmaps in comparison to original samples and heatmaps

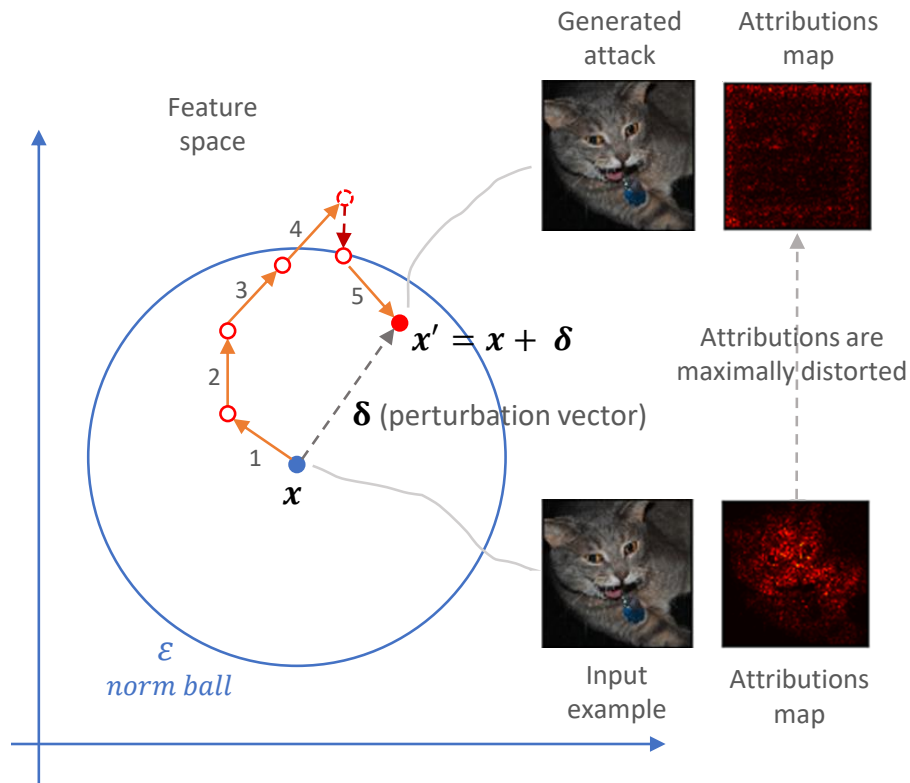


Figure 2.4: Projected gradient descent (PGD) optimization loop searching for an attribution attack.





Figure 2.5: Targeted attribution attack. Image taken from [9] for illustration.

**Targeted attacks.** In [9], the authors propose an attack that can manipulate model attributions map of an input image such that the manipulated image (attack) is imperceptible from the original image, and the attributions of the attack match an arbitrary target map. This is achieved by solving the optimization problem in Equation 2.7 where  $A(\cdot)$  is any attribution method,  $A^T$  is the target attribution map, and  $f(\cdot)$  denotes the model output. The first term makes the attack attribution as close to the target map as possible, and the second term ensures model outputs remains the same for the attack and the original input, while the parameter  $\gamma$  controls the relative weighing between the two objectives. An example of a targeted attack is shown in Figure 2.5.

$$\min_{x' \in N(x, \epsilon)} \|A(x') - A^T\|^2 + \gamma \|f(x') - f(x)\|^2 \quad (2.7)$$

**Attribution fragility with simple changes.** Recent work has shown that simply adding Gaussian noise to the input image in a small distance ball can change the attribution maps produced by a neural network [9]. Furthermore, simple and common preprocessing input transformations such as adding a constant shift to the input have been shown to heavily distort attribution maps [29].

**Adversarial model manipulation.** The methods discussed so far create attribution attacks by

modifying an input example. Other researchers have explored modifying model behavior (e.g., altering model parameters) to produce manipulated attributions without affecting the model’s outputs. In [24], the authors show that fine-tuning ImageNet pre-trained convolutional nets on a small dataset with manipulated attribution maps leads to the model learning weights that produce distorted attribution maps on entire test sets. Another work [60] proposes a technique to train an adversarial classifier that produces targeted attributions for LIME (Local Interpretable Model-agnostic Explanations [46]) and SHAP (SHapley Additive exPlanations [36]). In [8], the authors develop a simple fine-tuning loss function that can degrade the model sensitivity to an arbitrary target feature. [60] and [8] provide examples where such adversarial model manipulations effectively conceal a model’s reliance on undesirable feature dependencies such as learned harmful biases.

## 2.3 Attribution robustness

Attribution robustness is the property of a trained model where the attributions of an attack example do not drastically differ (or distort) from the attributions of the original, natural example. In other words, attribution attacks such as IFIA attacks are ineffective against robust models, or it is difficult to generate attacks for a model trained to have attribution robustness. This is illustrated in Figure 2.6.

Attribution robustness can be thought of as the counterpart of “adversarial robustness”; it is the robustness of a model’s input attributions (explanations), rather than the robustness of its predictions. At a high level, it can also be viewed as robustness in the learned rules space (or the decision-making space) in terms of feature significance scores or input gradients.

### 2.3.1 Why is attribution robustness important?

Attribution robustness of a machine learning model is an important research problem for several reasons.

**It improves the reliability of attributions or explanations.** In subsection 2.0.2, we discussed several use cases of attributions such as understanding a model’s learned rules, verifying that a model is relying on correct features, model debugging, and trust and compliance of machine

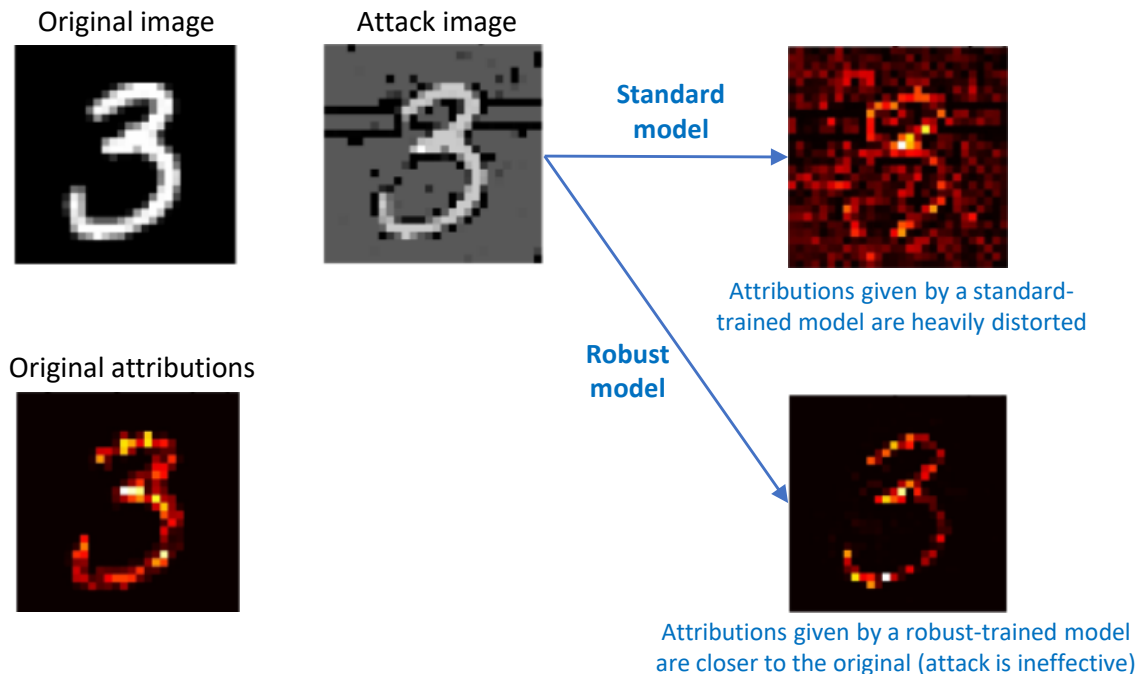


Figure 2.6: IFIA attacks are ineffective against models trained to have attribution robustness

learning systems. The ability to easily distort a model’s attributions with perturbations reduces the dependability and reliability of using attributions for such applications.

For example, in [13], the authors show that CNNs trained for MRI-based Alzheimer’s disease classification produce inconsistent attribution maps across different settings, making them unreliable as a tool of explanation. There is also evidence that attribution maps can be distorted in the presence of noise [10] without the need for smart or worst-case adversarial perturbations. This is especially problematic in critical applications or where model explanations are a legal requirement. The unreliability of attributions raises serious concerns for companies building machine learning products, regulators, and end-users who rely on attributions as a tool for interpretability [8]. Therefore, attribution robustness is an important quality for a machine learning model to have and an important research problem.

**The existence of attribution attacks reveals a failure mode of machine learning models.** Aside from the practical implications of attribution attacks, their existence implies that it is easy to modify a model’s decision making process, as indicated by the change in focus that it places on the different dimensions or elements of the input features space [8]. Rules learned by a human or the human decision making process would not change in this manner in the presence of small perturbations to the input. Therefore, the problem of attribution attacks and

training attribution robust models are important research questions to address.

**Attribution robustness could lead to a generic view of robust representation learning.** Robustness of models is a well studied problem in the machine learning literature, and the central concern has been the robustness of model predictions [8] (against adversarial attacks, noise, spurious correlations etc.). A model making correct predictions in the presence of such artifacts is considered to have learned robust representations. However, if a model is not robust in the input attribution space, the robustness of the learned representations can be questioned [70]. For a truly generic view of robustness in representation learning, we believe that attribution robustness is an important problem to investigate.

## 2.4 Methods for training attribution robust neural networks

As established in the previous section, the attribution robustness of machine learning models is an important research problem. In this section, we review the literature on training attribution robust neural networks.

### 2.4.1 Frameworks for attribution robust training

A principled and generic framework helps us understand the problem of attribution robustness broadly, compare it with other forms of robustness, and derive candidate solutions (robust training objectives). We discuss two prominent frameworks proposed in the current literature.

**Robust Attribution Regularization (RAR) [5].** In this work, the authors propose the framework in Equation 2.8. Here,  $l(x, y; \theta)$  is the standard cross entropy loss,  $s(\cdot)$  is a size function,  $IG(x, x')$  is integrated gradients, the outer minimization is the neural net training loop, and the inner maximization is the attack generation loop. The method relies on axiomatic properties of integrated gradients to achieve attribution robustness, leading to several theoretical connections to other forms of robustness (distributional and predication robustness).

$$\min_{\theta} \mathbb{E}_{(x, y)} \left[ l(x, y; \theta) + \lambda \max_{x' \in N(x, \epsilon)} s(IG(x, x')) \right] \quad (2.8)$$

Although this represents the first principled approach to attribution robustness in the literature, it is limited to using integrated gradients as the attribution method. This limitation exists for both the attack generation ( $x'$  via the inner maximization loop) and calculating the robustness term (second term) in the loss function. The authors also note difficulties in optimizing the robustness term with gradient descent during neural network training, as it remains at high values even after training completion.

**Framework for Attributional Robustness (FAR) [25].** This method removes the restriction of integrated gradients in [5] by making the attribution method arbitrary ( $S(\cdot)$ ). The FAR framework is shown in Equation 2.9, where  $d(\cdot)$  is a distance function that measures the difference between two attribution maps and  $S(\cdot)$  is any attribution method. Note that this framework allows for the ability to set any target attribution map  $S^T(x)$ , so that it can also be a manually generated or modified attribution map.

$$\min_{\theta} \mathbb{E}_{(x, y)} \left[ l(x, y; \theta) + \lambda \max_{x' \in N(x, \epsilon)} d(S^T(x), S(x')) \right] \quad (2.9)$$

The FAR framework is indeed more generic than the prior RAR framework; RAR can be derived from FAR. However, a crucial limitation in FAR is the coupling of the attribution attack generation and the robust regularization in the neural network loss function. i.e. the same term  $\max_{x'} d(S^T(x), S(x'))$  is used in both places. We believe that there is no fundamental reason for this to be the case: the attack generation process can be separate from the robust regularization term incorporated into the loss function, as proposed in our work.

## 2.4.2 Objective functions and techniques for attribution robust training

We now present several specific techniques for attribution robust training of neural networks and empirical results found in the literature. Some of these are robust objectives that can be instantiated from the RAR and FAR frameworks, while others take a different approach to robust training.

**“IG-Norm” and “IG-Sum-Norm” robust objectives.** The RAR paper [5] provides two attribution robust objectives termed IG-Norm and IG-Sum-Norm instantiated from the generic formulation. The IG-Norm objective computes an IG attack batch for each natural mini-batch and adds a robust regularization term to the loss function that is the norm of IG attribution

difference of natural and attack batches. The attack generated in IG-Sum-Norm training is a combination of prediction adversity and attribution adversity, intended to achieve both prediction and adversarial robustness in training. Small CNNs and Wide ResNet models are trained with the proposed robust objectives on several grayscale and natural color image datasets. Empirical results on small CNNs and Wide ResNet training show that both objectives achieve high attribution robustness (Kendall’s tau, top-k intersection metrics) compared to standard training and prediction robust objective of [37].

**“AAT” and “AdvAAT” robust objectives.** The FAR paper [25] also provides two attribution robust objectives termed AAT (adversarial attributional training) and AdvAAT instantiated from the generic formulation. The AAT objective computes an Iterative Feature Importance Attack (IFIA: see Algorithm 1) during training, which is a more generic attribution attack than the IG attack used in the previous IG-Norm loss. For the AdvAAT objective, a modified IFIA attack called Adversarial IFIA is generated, which is again a combination of prediction and attribution adversity. ResNets are trained on several grayscale and natural color image datasets, and the reported Kendall’s tau and top-k intersection results are slightly higher or comparable to the IG-Norm and IG-Sum-Norm results of [5].

**“ART” robust objective [58].** This paper proposes a robust objective termed ART (attributional robustness training) intended to minimize the upper bound of attribution vulnerability, defined in terms of spatial correlation between the input image and its attributions map. This is achieved through a soft-margin triplet loss that optimizes for high spatial correlation between the gradient-based attributions and the input image pixels values. The ART robust objective is an instantiation of the FAR generic loss formulation of [25].

The experimental setup and results of [58] are more comprehensive than other work on the topic of attribution robustness. Firstly, the authors train WideResNet-28-10 (28 layers deep, 10x wider than original ResNets), WideResNet-40-2 (40 layers deep) and ResNet-50 (50 layers deep) neural network architectures on several color dataset classification tasks. In comparison, the model architectures in previous works are only up to 18 layers deep. This is significant, considering the computational cost of generating attribution attacks during training and the challenges of computing input gradients for deep networks, such as vanishing gradients and stagnant loss values during training. Secondly, the authors show that attribution robust pre-training provides performances improvements in a downstream task of weakly supervised object localization in the CUB-200 dataset.

**Class attribution-based contrastive regularizer [54].** The authors of this work consider

attribution maps for both the positive class (ground truth) and negative classes of a classifier, and identify several conditions needed for attribution robustness. First, attribution map of the positive class must be sparse (localized to small pixel regions), and maps of the negative classes must be uniform (distributed across the image). Second, a change in attribution map due to an attack must be low for pixels with high attribution values in the original map. A novel robust loss function is proposed to enforce these properties, and empirical results with CNNs on image datasets demonstrate higher levels of attribution robustness than previous work [5].

**Geometric view of attribution robustness [9, 10].** The authors of these works present theoretical and empirical findings that a main cause of the fragility of attribution maps is the large curvature due to ruggedness of the hypersurface of constant network output. This implies that tiny input perturbations are able to drastically change the gradient vector on the hypersurface. Therefore, smoothing of the hypersurface should produce robust attributions. For ReLU neural networks, the authors propose a technique called  $\beta$ -smoothing. In this method, a neural net is first trained in the standard way with ReLU activations. Then, the ReLU activations are replaced with the softplus function  $\ln(1 + e^\beta x)/\beta$ , which is a smooth approximation of ReLU, parameterized by  $\beta$  (needs to be tuned). Robust attribution maps are then computed using this modified neural network.

The advantage of this method lies in its simplicity and minimal computational cost (no change to standard training), which makes it an excellent candidate for obtaining robust post-hoc explanations from trained ReLU-only models. However, for the purpose of training neural networks with the property of robustness, and studying the accompanying behavior and properties of such models, the after-the-fact softplus activation replacement is not a viable approach.

While the works discussed in this section achieve high attribution robustness compared to standard training, we note that it is difficult to compare attribution robustness performance numbers across research papers as discussed in subsection 2.5.1. This is largely due to the sensitivity of attributions to the many parameters of a training setup, as well as the lack of a uniform evaluation benchmark (datasets, models, metrics).

## 2.5 Evaluating attribution robustness

This section describes the methodology for evaluation of attribution robustness that is uniformly used in the literature, which is illustrated in Figure 2.7.

**Generating an attack dataset.** Measuring attribution robustness of a trained model involves comparing the similarity between attribution maps of a natural example and a corresponding attribution attack example. Attribution robust models should yield very similar attribution maps for a natural example and its perturbation (attribution attack). For this purpose, an attack dataset version of an out-of-sample test set is first generated using a standard-trained neural network classifier. The most common type of attribution attack used for this purpose is the Iterative Feature Importance Attack (IFIA) proposed in [20].

**Evaluation metrics.** To evaluate attribution robustness of a model, the following metrics are calculated. They are similarity metrics that measure similarity between a trained model’s attribution maps of the original (natural) test set and the attack set. The metrics are calculated for each example pair (original and attribution attack), and the average over all examples in the test set is taken as the overall measure of a model’s attribution robustness. In all metrics, higher values indicate higher levels of attribution robustness.

*Kendall’s tau* ( $\tau$ ) (Kendall’s rank correlation coefficient) [28]. Let the attribution values of pixels in the natural example be  $x_1, x_2, \dots$  and attribution values of pixels in the attack example be  $y_1, y_2, \dots$ . Any pair of values  $x_i, y_i$  and  $x_j, y_j$ , where  $i < j$ , are said to be “concordant” if the attribution sort order of the two pixels in the natural example ( $x_i, x_j$ ) has been preserved in the pixels of the attack example ( $y_i, y_j$ ). Otherwise, the pair is said to be “discordant”. Then Kendall’s rank correlation coefficient is calculated by Equation 2.10. Intuitively, this is a metric of how many pixel pairs (attribution value pairs) of the natural example have been preserved or distorted by the attack.

$$\tau = \frac{\text{number of concordant pairs} - \text{number of discordant pairs}}{\text{number of pairs}} \quad (2.10)$$

*Spearman’s rank correlation coefficient* ( $s$ ) [63]. The attribution values of the natural example ( $x_1, x_2, \dots$ ) and the attack example ( $y_1, y_2, \dots$ ) are first converted to rank values ( $R(x_1), R(x_2), \dots$  and  $R(y_1), R(y_2), \dots$ ). Then, the Spearman’s rank correlation coefficient is calculated as the standard Pearson correlation coefficient of the rank values (Equation 2.11).

$$s = \frac{\text{Cov}(R(X), R(Y))}{\sigma_{R(X)}\sigma_{R(Y)}} \quad (2.11)$$

*Top-k intersection.* In this metric, the  $k$  pixels with highest attribution values are selected (the



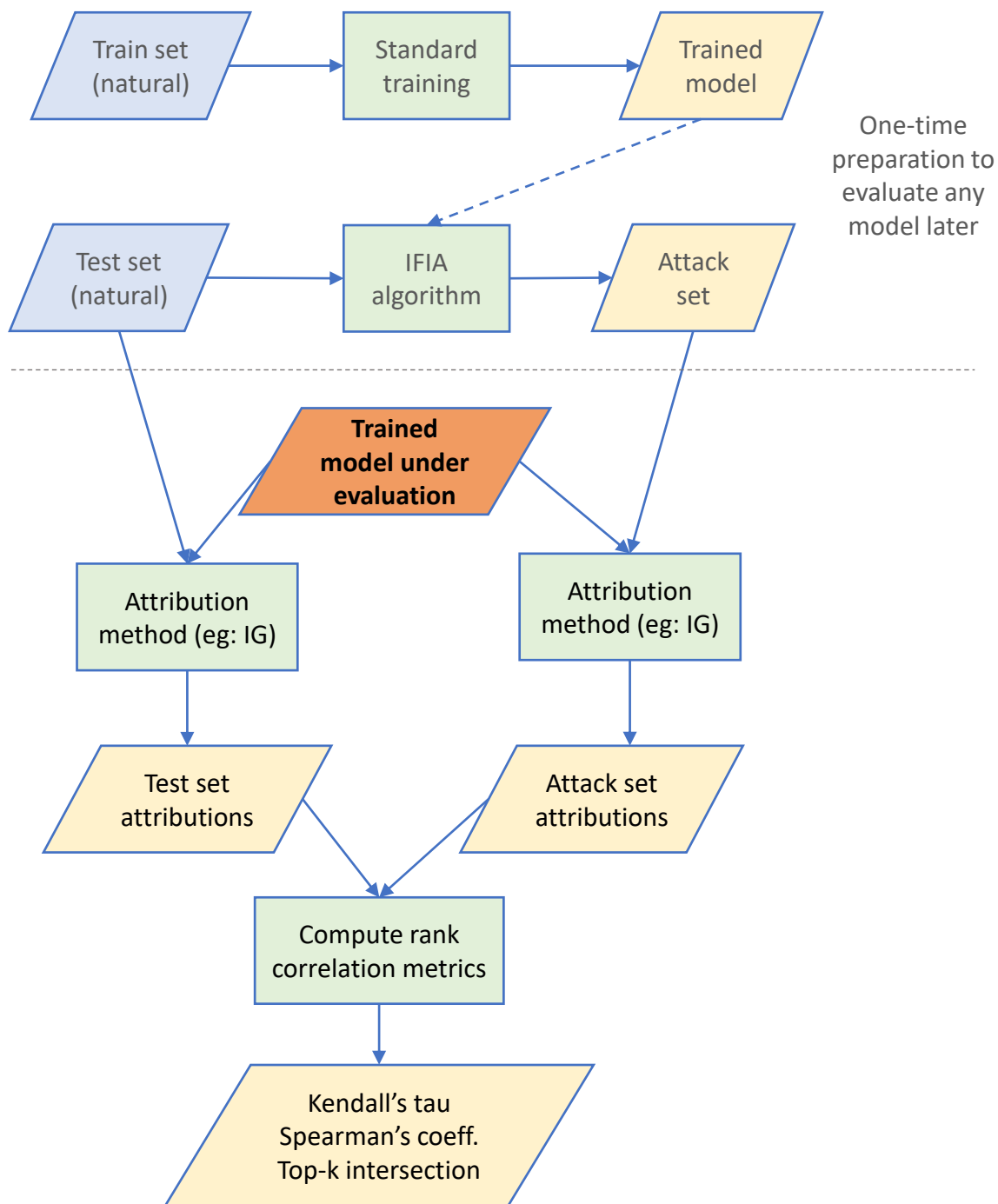


Figure 2.7: The process of evaluating attribution robustness of a trained model.

top- $k$  pixels) in the natural and attack examples. top- $k$  intersection is the percentage of pixels with overlap between the two attribution maps (no. of overlapping pixels/ $k$ ). The choice of  $k$  can be made based on the size of the input image;  $k = 200$  for small images (32 x 32) and  $k = 1000$  for large images (eg: 224 x 224) has been consistently used in the literature.

### 2.5.1 On the difficulty of benchmarking attribution robustness

It has been shown [25] that attribution maps (even robust) and therefore attribution robustness metrics are highly sensitive to the many hyperparameters in a robust training setup. Our experience with robust training of neural networks aligns with these findings; the outcomes are also sensitive to data preprocessing (eg: normalization) in addition to the training parameters. The evaluation setup as illustrated in Figure 2.7 includes an IFIA attack generation process, which also contains hyperparameters.

Additionally, the neural network architectures trained in different works are also diverse (small CNNs, wide and deep ResNets with different numbers of layers). This makes it difficult to compare reported attribution robustness results across different papers. In order to make a reliable benchmark and comparison between various robust training methods, a researcher would need to implement the relevant methods in her own training and evaluation setup and compare the results. For a practitioner trying to produce robust attribution maps within a machine learning system deployment, this would not be problematic, as the improvements of robust training methods can be measured against one another as implemented within that system.

## 2.6 Properties of attribution robust models

Since attribution robustness makes a model's decision making process in the input feature space more resilient (robust gradients), a good research question to ask is "what other properties do attribution robust models have?" Understanding such connections is important in developing unifying theories, connecting diverse research fields, and finding solutions to practical problems. This has been a common line of exploration for several other neural network training paradigms such as self-supervised learning and adversarial robust training. In this section, we present a brief summary of the literature on properties of attribution robust neural networks.

**Connection to adversarial robustness.** Intuition supports that models with robustness in the attribution space may also have robustness in the prediction boundary space (adversarial robustness). This can also be seen in the ability to derive adversarial robust objectives (eg: [37]) from the general attribution robustness frameworks (RAR [5], FAR [25] and our proposed framework).

Several recent works have explicitly established the connection between attribution robustness and adversarial (prediction) robustness of neural networks with empirical evidence. A theoretical framework linking adversarial robustness to the correlation between attributions  $A(x)$  and the image pixel values  $x$  itself (input-gradient alignment, a proxy for attribution robustness) is developed in [17]. This correlation is proven for linear models, and the observation is made that it is weakened for non-linear neural networks. The same relationship is established in [68] and the authors attribute it to the smoothness of decision boundaries in adversarial robust models. Empirical evidence on CNN classifiers on image datasets [58, 5] also suggest that the connection between the two forms of robustness exists both ways.

**Robustness to data noise or corruptions.** There is limited work in the literature exploring the connection between attribution robustness and model performance on noisy and corrupt data. In [58], the authors show that CNNs trained with the attribution robust objective give higher accuracy on CIFAR-10-C [23] (corrupted version of CIFAR-10 with various noise types and artifacts). They also show that the attribution robust models show higher performance on spatial adversarial perturbations [15] (small rotations and translations on CIFAR-10).

**Transfer learning performance on downstream tasks.** The same work [58] demonstrates another advantage of attribution robustness: pre-training a model with the robust objective for classification gives higher performance on downstream tasks of image segmentation (weakly supervised image localization). The authors provide empirical evidence of this by evaluating performance on two segmentation tasks on CUB-200 and Flower datasets.

## 2.7 Research problems and gaps on attribution robustness

The discussion so far has introduced the reader to the literature on attributions, attribution attacks and attribution robustness. Our impression is that the literature on attribution robustness is generally lacking in comparison to the well studied problem of adversarial or prediction robustness. We now identify several gaps or under-explored areas that could benefit from more

attention by researchers.

**Attribution attacks.** Several categories and variants of attribution attacks have been studied in the existing literature. However, we do not see a unifying framework of attribution attacks and threat models, as well as behaviors such as attack transferability across systems, and their effect on different types of attribution methods, models and data modalities.

**Efficient robust training methods and scalability.** Attribution robust training often involves an inner optimization problem to generate attribution attacks during training. This is even more expensive than prediction robust attack generation as it involves a large number of gradient computations and vector difference calculations. While techniques for efficient and scalable prediction robust training has been an active area of research [57, 71], there is no counterpart in the attribution robust training literature.

**Properties of attribution robust models.** We discussed the existing works on this topic in section 2.6 (prediction and noise robustness). However, given that attribution robustness models achieve robust gradients in the input space at high computational cost, more research is needed to understand behaviors both desirable and undesirable in robust models.

**Diverse set of tasks, models and data domains.** Nearly all research on attribution attacks and robustness has been done in the context of convolutional neural networks on image classification. This is likely due to the fact that attributions on images can be easily visualized. However, attribution attacks and robustness are general problems, and more work is needed to understand the impact of attribution robust training on other tasks, models and data domains.

**Benchmarks for attribution robustness.** The difficulty of comparing the performance of robust training methods was discussed in subsection 2.5.1. More work on standard benchmarks (large attack datasets and evaluation setups) could be highly useful to the research community.

In the remaining chapters of the thesis, we set out to address some of these problems to some extent and present our experiments and findings.

## Chapter 3

# A Novel Framework and Robust Objective for Training Attribution Robust Neural Networks

In the previous chapter, the significance of attribution robustness of neural networks was discussed, and several lacking areas in the literature were identified. A major problem with existing robust training methods is the very high computational cost, making it difficult to do large scale robust training of neural networks. In this chapter, we present our work addressing this issue: a novel and generic framework of loss functions for attribution robust training, and a new, efficient robust training objective derived from the framework.

### 3.1 Contributions of the chapter

The research contributions of this chapter are summarized below.

- **A framework for attribution robust training** (section 3.2). We present a novel framework of loss functions (training objectives) for attribution robust training of neural networks. The novelty of the proposed framework lies in decoupling attack generation from the regularization terms in the loss function, adding a cross entropy of attacks term in the loss function, and the flexibility to use any attribution method. This provides a principled

approach to think about attribution robust training, wherein a practitioner can choose different instantiations of the constituent components and run training sessions.

- **A new robust objective** (subsection 3.2.2). We derive a novel attribution robust objective from the proposed framework, termed the cross entropy of attacks (“CE-of-attacks”). This objective is simple to implement (generate an attack during training and compute the cross entropy of it) and significantly more computationally efficient than existing robust objectives. The CE-of-attacks objective is 2 to 10 times faster for robust training of neural networks compared to IG-Norm, a prominent existing method.
- **Empirical analysis of robust objectives** (section 3.3). We conduct comprehensive experiments on training attribution robust convolutional neural networks (small CNNs and ResNets) on image datasets (MNIST, CIFAR-10, ImageNet subset) with the proposed novel robust objective (CE-of-attacks) as well as existing robust objectives. The CE-of-attacks objective achieves higher attribution robustness than the IG-Norm objective [5] while simultaneously providing higher test accuracy. For example, a small 4-layer CNN trained on the CIFAR-10 dataset gives 72.1% accuracy with the CE-of-attacks objective compared to 70.7% accuracy with the IG-Norm objective, while achieving a 68.4% top-100 intersection compared to the 64.9% top-100 intersection of the IG-Norm objective. The results of the diverse experimental setup are novel and a valuable addition to the literature on attribution robustness of neural networks.

## 3.2 Loss functions for attribution robustness: a framework

In this section, we expand on the framework of loss functions for robust training of neural network classifiers proposed in [5]. The framework of [5] is termed the “Uncertainty Set Model” and we note that it is fairly generic and allows for multiple important instantiations (see section 2.4). The main limitation in their framework is the coupling of the optimization term to the regularization term in the loss function. Another limitation is its restriction to the use of integrated gradients to create the attribution attack during training. Additionally, the loss function contains only a size measurement of IG difference between the natural and attack example, which restricts the ability to easily include other potential candidate terms (eg: cross entropy of attack examples). Our proposed framework of attribution robust loss functions or training objectives for neural network classifiers can be expressed in the following form;

$$\min_{\theta} \mathbb{E}_{(x,y)} [\rho(x,y,\theta)] \quad (3.1)$$

$$\rho(x,y,\theta) = \alpha l(x,y,\theta) + \beta l(x',y,\theta) + \lambda \text{size}(\text{Diff}[A(x),A(x')]) \quad (3.2)$$

$$\text{where } x' = \max_{x' \in N(x,\epsilon)} \text{size}(\text{Diff}[A(x),A(x')]) \quad (3.3)$$

The terms in the above expressions are as follows (see Figure 3.1 for a quick visualization);

- Equation 3.1 is the usual minimization of a given loss function  $\rho(x,y,\theta)$ , which is done in the model training (optimization) loop.
  - $(x,y)$  are the (natural) training examples and the ground truth labels.
  - $\theta$  are the model parameters (weights of the neural network).
- Equation 3.2 is the attribution robust objective or the loss function.
  - The first term  $\alpha l(x,y,\theta)$  is the standard cross entropy loss of the natural examples with respect to the ground truth labels, weighted by a constant parameter  $\alpha$ .
  - The second term  $\beta l(x',y,\theta)$  is the standard cross entropy loss of the attribution attack examples (generated during training as shown in Equation 3.3) with respect to the ground truth labels, weighted by a constant parameter  $\beta$ .
  - The third term  $\lambda \text{size}(\text{Diff}[A(x),A(x')])$  is a measure of the difference between attributions given by the model for natural and attack examples.  $A$  is a function that computes attributions of the model, so that  $A(x)$  and  $A(x')$  refer to the attributions of natural and attack examples respectively.  $\text{Diff}[\ ]$  is a function that computes the difference between two vectors, and  $\text{size}()$  is a function that measures the size of a vector (eg: norms). The third term is weighted by a constant parameter  $\lambda$ .
  - Note that attribution function  $A()$  depends on the model parameters  $\theta$ , although it is omitted in the above expressions for clarity.
- Equation 3.3 is the attribution attack example computation. It is an optimization problem that finds a perturbed version of the original (natural) example within a given distance budget  $\epsilon$  such that the difference between attributions of the natural example and the perturbed version is maximized.

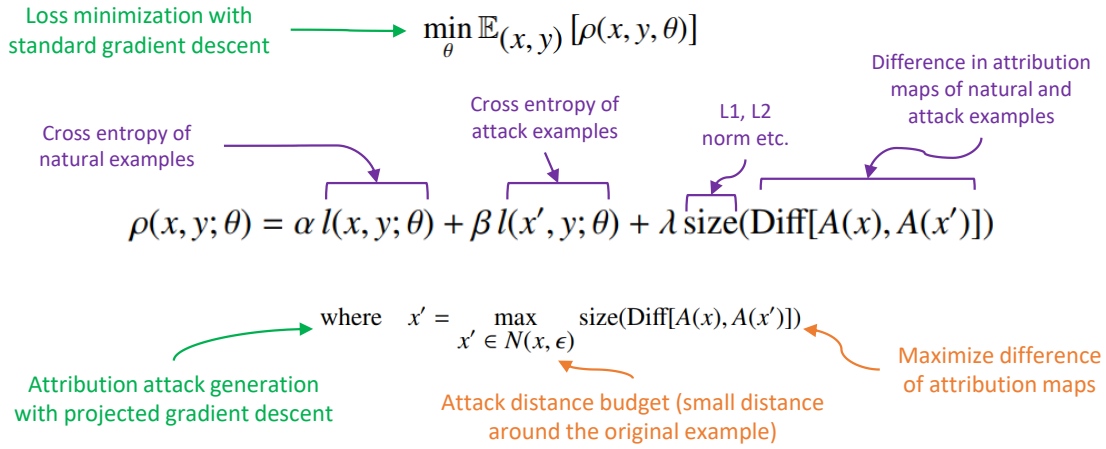


Figure 3.1: Explanation of the terms in the proposed robust loss functions framework

- $x'$  is the resulting attack example found by solving the optimization problem.
- $x' \in N(x, \epsilon)$  specifies that the attack example  $x'$  must be found within a distance budget of  $\epsilon$  around  $x$ .
- $\text{size}()$ ,  $\text{Diff}[]$ , and  $A()$  refer to the same functions as described before.

Note the following aspects of the proposed framework and the changes in comparison to prior frameworks in [5, 25].

- The attack generation (Equation 3.3) is decoupled from its use within the loss function (Equation 3.2). This simple change allows one to use any method to generate an attack, and use it in the loss function in a flexible manner. For example, the simple but novel robust objective “cross entropy of attacks” can be derived only because of this flexibility, which is not possible with prior frameworks.
- Two cross entropy loss terms are included; one for natural examples and one for attribution attack examples. This allows for flexible instantiations. It also adds two additional hyperparameters, but we can instantiate a loss function in a way that all three hyperparameters are not needed (eg: setting  $\alpha = 0$ ,  $\beta = 1$ ,  $\lambda = 0$  gives us the robust training objective “cross entropy of attacks”).
- The  $IG(x, x')$  term in the loss function of [5] is replaced by a more generic  $\text{Diff}[A(x), A(x')]$  term. This allows us to use any attribution method  $A()$  in place of integrated gradients



(IG), and to use suitable difference and size functions. For example, we can calculate gradients instead of IG and compute the L2 norm of the absolute value difference between the two gradient maps  $A(x)$  and  $A(x')$ .

- Replacing IG could come with the loss of certain theoretical properties inherent in IG. From an empirical point of view, that is not problematic, as we are searching for efficient and effective attribution robustness methods that work well in practice.
- The  $x'$  in this loss function is the attribution attack, which is generated during training for each natural example  $x$ . It is usually computed with projected gradient descent (PGD) within a distance budget of  $\epsilon$  around the natural example. The resulting  $x'$  is then used in the third term of the loss function (Equation 3.2).
- Since projected gradient descent (PGD) needs to compute the derivatives of the term  $\text{size}(\text{Diff}[A(x), A(x')])$  w.r.t  $x'$ , the functions  $\text{size}()$ ,  $\text{Diff}[]$  and  $A()$  must be differentiable. Other than that, there is no restriction in the choice of these functions. One can also consider derivative-free optimization techniques to bypass this restriction (not explored in this work).
- It is also possible to use different  $A()$  and  $\text{Diff}[]$  functions in the loss function's third term (Equation 3.2) and for the attack calculation Equation 3.3. This allows for more flexibility in instantiating an attribution robust objective.

### 3.2.1 Loss function instantiations

In this section, we discuss several instantiations of the proposed general framework. These instantiations are robust training objectives, and while some of them are well-known objectives, others have not been explored in the current literature to the best of our knowledge. These derivations demonstrate the generalization capability of the proposed framework.

The following are well-known robust objectives.

- Input Gradient Regularization objective [12, 47]
- Adversarial robustness objective [37]
- IG-Norm objective [5]

- IG-Sum-Norm objective [5]

The following robust objectives can be derived from the generic framework, and they are either novel or under-explored in the existing literature.

- Cross entropy of IG attacks
- Cross entropy of natural examples + Cross entropy of IG attacks
- Cross entropy between model outputs for natural examples and IG attacks
- Grad-Norm objective
- Objectives with layer-wise attacks

**The “Input Gradient Regularization” objective** [12, 47]. Let  $\alpha = 1$ ,  $\lambda = 0$ ,  $size(\cdot) = \|\cdot\|_2^2$  (square of  $L_2$  norm), and  $Diff[A(x), A(x')] = \nabla_x l(x, y; \theta)$  (gradients of loss w.r.t.  $x$ ). Then, we get the robust objective in Equation 3.4.

$$\min_{\theta} \mathbb{E}_{(x, y)} \left[ l(x, y; \theta) + \beta \|\nabla_x l(x, y; \theta)\|_2^2 \right] \quad (3.4)$$

Note that in this objective, an attack computation does not occur. Instead, the square of  $L_2$  norm of gradients of the loss function with respect to the input (input gradients) are added to the standard cross entropy loss. This was originally proposed in [12] as "double backpropagation" (due to the need for an extra backpropagation for the second term).

**The adversarial robustness objective** or the robust prediction objective [37]. Let  $\alpha = 0$ ,  $\beta = 1$ ,  $\lambda = 0$ ,  $size(\cdot) = sum(\cdot)$  (sum of all elements of a vector), and  $Diff[A(x), A(x')] = l(x, y; \theta)$  (integrated gradients of loss from a baseline of  $x'$  to  $x$ ). Then, we get the robust objective in Equation 3.5.

$$\min_{\theta} \mathbb{E}_{(x, y)} \left[ \max_{x' \in N(x, \epsilon)} l(x', y; \theta) \right] \quad (3.5)$$

This is the generic adversarial robust objective as proposed in [37]. Note that the inner optimization is maximizing the cross entropy loss, i.e. it is finding an attack that fools the model

prediction (an adversarial attack or a prediction attack), which is different from finding an attribution attack.

**The IG-Norm objective [5].** Let  $\alpha = 1, \beta = 0$ ,  $size(\cdot) = \|\cdot\|_1$  ( $L_1$  norm), and  $Diff[A(x), A(x')] = IG(x, x')$  (integrated gradients of loss from a baseline of  $x'$  to  $x$ ). Then, we get the robust objective in Equation 3.6.

$$\min_{\theta} \mathbb{E}_{(x, y)} \left[ l(x, y; \theta) + \lambda \max_{x' \in N(x, \epsilon)} \|IG(x, x')\|_1 \right] \quad (3.6)$$

In this robust attribution objective, the inner optimization (generating an attribution attack) is maximizing the difference of integrated gradients between the natural and attack example ( $\|IG(x, x')\|_1$ ), i.e. it is generating IG attack. The final loss function is the sum of cross entropy loss on natural attacks (learn to classify) and the difference of IG between the natural and the generated attack example (learn robust attributions).

**The IG-Sum-Norm objective [5].** Let  $\alpha = 0, \beta = 1$ ,  $size(\cdot) = \|\cdot\|_1$  ( $L_1$  norm), and  $Diff[A(x), A(x')] = IG(x, x')$  (integrated gradients of loss from a baseline of  $x'$  to  $x$ ). Then, we get the robust objective in Equation 3.7.

$$\min_{\theta} \mathbb{E}_{(x, y)} \left[ \max_{x' \in N(x, \epsilon)} l(x', y; \theta) + \lambda \|IG(x, x')\|_1 \right] \quad (3.7)$$

In this robust attribution objective, the inner optimization is maximizing both the cross entropy loss and the IG difference, i.e. it is finding an attack that fools model predictions as well as attributions.

**The ‘‘Cross entropy of IG attacks’’ objective.** Let  $\alpha = 0, \beta = 1, \lambda = 0$   $size(\cdot) = \|\cdot\|_1$  ( $L_1$  norm), and  $Diff[A(x), A(x')] = IG(x, x')$  (integrated gradients of loss from a baseline of  $x'$  to  $x$ ). Then, we get the robust objective in Equation 3.8.

$$\begin{aligned} & \min_{\theta} \mathbb{E}_{(x, y)} [l(x', y; \theta)] \\ \text{where } x' = & \max_{x' \in N(x, \epsilon)} \|IG(x, x')\|_1 \end{aligned} \quad (3.8)$$

Note that in this objective, an IG attribution attack is calculated, and the model is trained for the cross entropy loss of the attack example. In our experiments, this novel objective achieved the best performance in terms of attribution robustness, accuracy and computational efficiency.

**The “Cross entropy of natural examples + Cross entropy of IG attacks” objective.** Let  $\alpha = 1$ ,  $\lambda = 0$   $size(\cdot) = \|\cdot\|_1$  ( $L_1$  norm), and  $Diff[A(x), A(x')] = IG(x, x')$  (integrated gradients of loss from a baseline of  $x'$  to  $x$ ). Then, we get the robust objective in Equation 3.9.

$$\begin{aligned} & \min_{\theta} \mathbb{E}_{(x, y)} [l(x, y; \theta) + \beta l(x', y; \theta)] \\ \text{where } x' &= \max_{x' \in N(x, \epsilon)} \|IG(x, x')\|_1 \end{aligned} \quad (3.9)$$

Note that in this objective, an IG attribution attack is calculated, and the model is trained for the weighted sum of cross entropy loss of the natural example and the cross entropy loss of the attack example.

**The “Cross entropy between model outputs for natural examples and IG attacks” objective.** Let  $\alpha = 1$ ,  $\beta = 0$ ,  $\lambda = 0$   $size(\cdot) = \|\cdot\|_1$  ( $L_1$  norm), and  $Diff[A(x), A(x')] = IG(x, x')$  (integrated gradients of loss from a baseline of  $x'$  to  $x$ ). Let  $l(x, y; \theta) = l\{f_{\theta}(x), f_{\theta}(x'); \theta\}$  (cross entropy between model outputs for natural examples and attacks.  $f_{\theta}(\cdot)$  is the model function). Then, we get the robust objective in Equation 3.10.

$$\begin{aligned} & \min_{\theta} \mathbb{E}_{(x, y)} [l\{f_{\theta}(x), f_{\theta}(x'); \theta\}] \\ \text{where } x' &= \max_{x' \in N(x, \epsilon)} \|IG(x, x')\|_1 \end{aligned} \quad (3.10)$$

In this objective, an IG attribution attack is computed, and the model output vector (class logits) for the natural example and the attack is separately computed. Then, the cross entropy loss between the two output vectors is used as the training objective. The intuition behind this objective is that an attribution robust model must produce similar output for both natural and attack examples.

**The Grad-Norm objective.** The proposed framework allows the flexibility to use a different attribution method than IG to generate an attribution attack. As a concrete example, consider using input gradients as attributions instead of IG. Let  $\alpha = 1$ ,  $\beta = 0$ ,  $size(\cdot) = \|\cdot\|_1$  ( $L_1$  norm),  $A(x) = \nabla_x l(x, y; \theta)$  (input gradients of cross entropy loss), and  $Diff[\mathbf{a}, \mathbf{b}] = \mathbf{a} - \mathbf{b}$ . Then, we get

the robust objective in Equation 3.11.

$$\min_{\theta} \mathbb{E}_{(x,y)} \left[ l(x,y;\theta) + \lambda \max_{x' \in N(x,\epsilon)} \|\nabla_x l(x,y;\theta) - \nabla_{x'} l(x',y;\theta)\|_1 \right] \quad (3.11)$$

**Attacks on attributions of intermediate layers (layer-wise attacks).** In the instantiations discussed so far, the attributions are computed for the input layer of a neural network (input attributions). In all gradient-based attribution methods, it is also possible to compute attributions for any intermediate layer of a neural network, and therefore generate attribution attacks on the intermediate layers. This implies that any robust objective we have discussed so far can be instantiated with layer-wise attribution attacks. (Note: for clarity, we have omitted the layer-wise notation from the generic framework in Equation 3.2).

As a concrete example, consider a variation of IG-Norm objective (see Equation 3.6) with layer-wise attacks. Let  $\alpha = 1, \beta = 0$ ,  $size(\cdot) = \|\cdot\|_1$  ( $L_1$  norm), and  $Diff[A(x), A(x')] = IG_{\mathbf{h}}(x, x')$  (integrated gradients of loss at an intermediate layer  $\mathbf{h}$  from a baseline of  $x'$  to  $x$ ). Then, we get the robust objective in Equation 3.12

$$\min_{\theta} \mathbb{E}_{(x,y)} \left[ l(x,y;\theta) + \lambda \max_{x' \in N(x,\epsilon)} \|IG_{\mathbf{h}}(x, x')\|_1 \right] \quad (3.12)$$

**Other possible variants of robust objectives.** As demonstrated by the instantiations described in this section, the proposed framework for attribution robust objectives is generic and flexible. We have discussed some variants, but it is worth noting that it is possible to instantiate more robust objectives by choosing different  $A(\cdot)$  (attribution method),  $Diff[\cdot]$  (difference) and  $size(\cdot)$  functions.

### 3.2.2 The ‘‘Cross entropy of IG attacks’’ objective and training loop

The ‘‘Cross entropy of IG attacks’’ (CE-of-attacks) loss function initiation (Equation 3.8) is a novel attribution robust objective that has not been explored in the literature. A neural network training loop that implements this objective is shown in Algorithm 2.

The ‘‘CE-of-attacks’’ robust objective compares and contrasts with current popular attribution

robust objectives such as IG-Norm (Equation 3.6) and IG-Sum-Norm (Equation 3.7) [5] in the following ways;

- Similar to IG-Norm and IG-Sum-Norm, an attribution attack  $x'$  (integrated gradients attack) is computed for each natural example in the training loop. This is a solution to the inner maximization problem computed with projected gradient descent (PGD).
- Unlike IG-Norm and IG-Sum-Norm which calculates a  $\|IG(x, x')\|_1$  term for the outer minimization in the training loop (the model learns to produce similar IG attributions for the natural and the attack examples), the CE-of-attacks objective simply takes the cross entropy loss of the attacks  $l(x', y; \theta)$ .
- This essentially trains the model to classify attribution attack examples. The intuition is that the model will learn the patterns in the attacks, and therefore achieve some level of attribution robustness. Our experiments find that this is indeed the case. This may seem like an obvious or trivial technique; in fact, training on attacks is the typical adversarial or prediction robustness technique [37] (Equation 3.5). However, it is worth noting that none of the prior attribution robustness frameworks are able to derive this simple loss function due to the coupling of the regularization terms with the inner maximization.
- The benefit of the CE-of-attacks objective comes from not having to calculate the  $\|IG(x, x')\|_1$  term in the outer minimization loop, which is very expensive. Experimentally, the outer IG calculation of IG-Norm and IG-Sum-Norm has to be done with  $ig\_steps$  between 24 - 50 (that many backward passes or backpropagations to compute gradients). That is 24 - 50 more backprops than a standard training loop!
- In the CE-of-attacks objective, IG term calculation is only necessary for the inner maximization or attack generation. Experimentally, we find that  $ig\_steps = 4$  is adequate to produce good attribution robustness in convolutional neural networks on image datasets. Therefore, with practical hyperparameter values, the CE-of-attacks robust objective is several orders of magnitude more efficient than the currently popular IG-Norm and IG-Sum-Norm attribution robust objectives.

---

**Algorithm 2** Training loop with the “Cross entropy of IG attacks” robust objective:

$$\min_{\theta} \mathbb{E}_{(x, y)} [l(x', y; \theta)] \quad \text{where} \quad x' = \max_{x' \in N(x, \epsilon)} \|IG(x, x')\|_1$$


---

**Input:**

Training data  $\mathcal{D} = (X_{all}, Y_{all})$

Neural network weights  $\theta$

Outer training loop hyperparameters:

No. of training steps ( $N$ ), learning rate ( $\beta$ ), batch size ( $batch\_size$ )

Inner attack generation loop hyperparameters:

No. of iterations ( $iterations$ ), step size ( $\alpha$ ), distance budget ( $\epsilon$ ), IG steps: ( $ig\_steps$ )

Function :  $integrated\_gradients(A, B, Y, ig\_steps)$

Computes integrated gradients of cross entropy, w.r.t. A going from B

1: **Initialization:**

2: Initialize  $\theta$  with random weights

3: **Training loop:**

4: **for**  $i \leftarrow 1$  to  $N$  **do**     // Outer loss minimization loop

5:   Take mini-batch  $(X, Y)$  of size  $batch\_size$  from  $\mathcal{D}$

6:    $X\_attacks \leftarrow X$      // Initialize batch of attacks at the original examples

7:   **for**  $j \leftarrow 1$  to  $iterations$  **do**     // Inner maximization loop (compute attack with PGD)

8:      $IG \leftarrow integrated\_gradients(X, X\_attacks, Y, ig\_steps)$

9:      $IG\_norm \leftarrow L1\_norm(IG)$

10:      $grad \leftarrow gradient(IG\_norm, X\_attacks)$      // Gradient of  $IG\_norm$  w.r.t.  $X\_attacks$

11:      $X\_attacks \leftarrow X\_attacks + \alpha * grad$      // Gradient ascent update in PGD

12:      $X\_attacks \leftarrow clamp(X\_attacks, \epsilon)$      // Projection in PGD

13:   **end for**

14:    $loss = cross\_entropy(X\_attacks, Y; \theta)$      // Cross entropy loss of attack mini-batch

15:    $\theta \leftarrow \theta - \beta * \nabla_{\theta} loss$      // Gradient descent weight update on mini-batch

16: **end for**

**Output:**

Trained neural network weights  $\theta$  [with attribution robustness]

---

### 3.3 Attribution robustness experiments

We now take a selection of robust objectives from the previous section and train neural network classifiers on image datasets using these objectives. The aim is to determine if the robust training objectives, both from prior work and those proposed here, can effectively train attribution robust neural networks. The experiment setups, results and our findings are described in this section.

#### 3.3.1 Experiment details

##### Training objectives

The following three objective functions are used for training the neural network models. In all robust training setups, a one-step IG attack is used in the training to achieve a balance between efficiency and resulting attribution robustness (exception: on MNIST, a 5-step attack is used).

- Standard training (cross entropy loss on natural examples).
- CE-of-attacks or CEA (cross entropy of IG attacks): the novel attribution robust objective proposed in this work (Equation 3.8, subsection 3.2.2)
- IG-Norm [5]: the leading, theoretically grounded attribution robust objective in the current literature (Equation 3.6). This objective has a second term in the loss function (the attribution regularization term) that is expensive to compute (subsection 4.2.1).

##### Models and datasets

**Models.** Three convolutional neural network architectures are trained for classification tasks in the attribution robust training experiments.

- Small convolutional neural networks (Small-CNN). 4 pairs of ReLU-convolutional and max pooling layers. 32, 64, 128, 256 nodes in the 4 convolutional layers.



- Deep residual networks (ResNet-18 [22]). 18 ReLU-convolutional layers in 4 blocks with residual connections. 64 - 512 nodes in the convolutional layers.
- Wide residual networks (WRN-10-4 [73]). 10 ReLU-convolutional layers in 3 blocks with residual connections. 64 - 256 nodes in the convolutional layers.

**Datasets.** The neural network models are trained for classification tasks in the following datasets. These datasets represent a variety of image data modalities, including grayscale, and small and large natural color images. Each dataset includes a training set and a smaller test set for evaluation. The Small-CNN model is trained on the MNIST and CIFAR-10 datasets, while the all three architectures are trained on the RIMB dataset.

This combination of models and image modalities provides a diverse experimental setup, so that the attribution robustness results and insights derived from this work have broad applicability. All models are trained for a sufficient number of epochs until the test error converges.

- MNIST [32]. 60,000 grayscale digits, Image size: 28 x 28. Classes: 10 digit.
- CIFAR-10 [31]. 60,000 color natural images. Image size: 32 x 32. Classes: 10 object classes.
- RIMB (Restricted ImageNet Balanced) [14]. This is a small subset of the ImageNet dataset. It contains 37,400 images belonging to 14 classes (balanced). Image sizes in the range of 256 x 224.

### Evaluation data and metrics

**Attack datasets.** To measure attribution robustness of a model, similarity between attribution maps of natural and attack examples produced by the model is measured. Following the standard practice of the literature, we choose the Iterative Feature Importance Attack (IFIA) proposed in [20] (subsection 2.2.1) to create the attack data from the natural test sets in each dataset. Note that the IFIA attack used for evaluation is a stronger attack than the simple IG attacks computed during training (eg: up to 100 IFIA iterations vs. one-step IG attacks). See section A.1 for parameters of the IFIA attack generation process.

**Evaluation metrics.** To evaluate the similarity between attribution maps of natural and IFIA attack data, we choose two commonly used metrics in the literature: Kendall tau (Kendall's

rank correlation coefficient) and top-k intersection (top-100 for MNIST and CIFAR-10, top-1000 for RIMB).

See section 2.5 for a complete description of the attribution robustness evaluation setup and definitions of the metrics.

Additionally, we also evaluate the test accuracy on the original (natural) test set to verify that the models trained with robust objectives are not failing on the natural examples. Accuracy on the IFIA attack dataset is also calculated.

### 3.3.2 Results and discussion

Table 3.1: Attribution robustness (average over test set) of different training objectives. CEA = CE-of-attacks. Best performance in each metric is highlighted in bold.

Dataset	Model	Training method	Natural accuracy %	Attack accuracy %	Kendall tau	Top-k intersection %
MNIST	Small-CNN	Standard	<b>99.08</b>	13.25	0.35	58.92
		IG-Norm	98.93	<b>92.27</b>	0.41	67.63
		CEA	98.97	83.75	<b>0.43</b>	<b>73.78</b>
CIFAR-10	Small-CNN	Standard	<b>74.44</b>	54.46	0.57	57.26
		IG-Norm	70.74	63.73	0.63	64.86
		CEA	72.12	<b>64.49</b>	<b>0.66</b>	<b>68.35</b>
RIMB	Small-CNN	Standard	<b>82.64</b>	71.57	0.41	34.96
		IG-Norm	79.5	68.79	0.45	39.54
		CEA	76.71	<b>75.57</b>	<b>0.48</b>	<b>42.71</b>
RIMB	WRN-10-4	Standard	<b>81.29</b>	48.64	0.47	38.64
		IG-Norm	-	-	-	-
		CEA	79.07	<b>66.00</b>	<b>0.55</b>	<b>49.16</b>
RIMB	ResNet-18	Standard	<b>87.14</b>	69.57	0.37	33.23
		IG-Norm	67.93	50.64	0.37	<b>43.13</b>
		CEA	86.86	<b>82.21</b>	<b>0.46</b>	39.81

Table 3.1 shows the average attribution robustness metrics (over the test set), and accuracy on natural test set and IFIA attack set obtained by training the neural networks with the three different objective functions. Figure 3.2 illustrates the distribution of top-k intersection for each test example pair (natural and attack). As expected, both robust objectives (IG-norm and CE-of-attacks) produce higher attribution robustness than the standard cross entropy loss function.

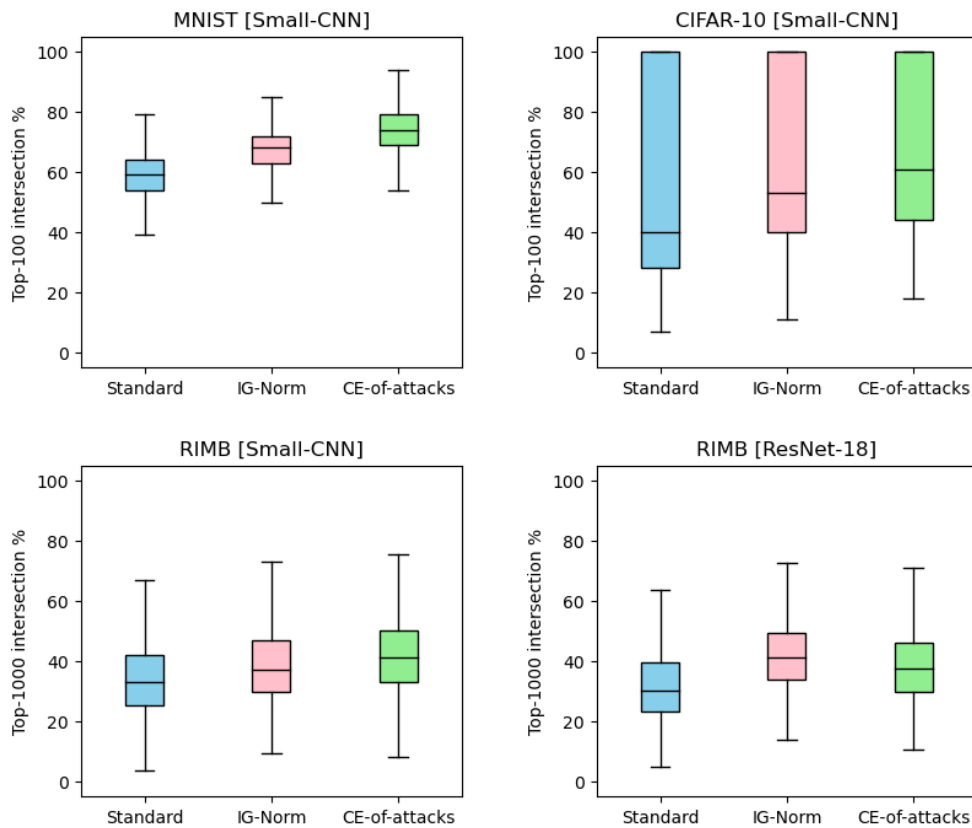


Figure 3.2: Attribution robustness of the three training objectives on four setups (top-k intersection distribution over all test examples). The middle horizontal bar is the median, box boundaries are the 25th and 75th percentiles and whiskers show the rest of the distribution.

**Comparing attribution robustness metrics.** The average Kendall tau and top-k intersection metrics over the test set in Table 3.1 show that the CE-of-attacks objective achieves high attribution robustness compared to the IG-Norm objective in most setups. This is also evident by the upward shift in the distribution of top-k intersection of the CE-of-attacks objective relative to IG-Norm (Figure 3.2). The one exception is the ResNet-18 model trained on the RIMB dataset, where the IG-Norm objective achieves higher top-k intersection. However, this comes with a considerable drop in accuracy (IG-Norm is 18.9% less accurate than CE-of-attacks with ResNet-18 on RIMB). Training the Wide ResNet (WRN-10-4) model on the RIMB dataset with the IG-Norm robust objective proved to be unsuccessful (test accuracy was too low for many training epochs and training was too slow).

It must be noted that the CE-of-attacks objective is also significantly faster than the IG-Norm objective (1.5 hours of CE-of-attacks training vs. 8 hours of IG-Norm training). The inefficiency of IG-Norm comes from computing the attribution regularization term in the loss function with many steps for the integrated gradients (24 steps) and backpropagating over it during gradient descent. A full comparison of computational cost is given in (subsection 4.2.1).

**Comparing natural accuracy.** In all setups in Table 3.1, the two robust objectives give lower accuracy on the test set (natural accuracy). This implies that there is a trade-off between attribution robustness and accuracy. This phenomenon has been noted in prior work as well [25, 10]. In fact, if one is willing to lose test accuracy, it is possible to tune both robust objectives to produce higher attribution robustness (eg: by generating a stronger IG attack during training or increasing the regularization strength parameter  $\lambda$  in IG-Norm). The challenge is to find a desired mid point for the two goals of attribution robustness and accuracy. This important effect is explored further in section subsection 4.2.2.

**Comparing attack accuracy.** In all scenarios setups in Table 3.1, model accuracy on the IFIA attack set is higher with the two robust objectives compared to standard training. On this metric as well, the CE-of-attacks objective performs better than the IG-Norm objective with the one exception on the MNIST dataset.

**Evolution of attribution robustness during training.** The discussion so far was based on final results of complete training sessions that reached convergence in test accuracy. It is also important to observe the evolution of the desired metrics over a training session. Figure 3.3 illustrates attribution robustness (top-k intersection) and test accuracy over training epochs on MNIST and CIFAR-10 datasets. Attribution robustness with the standard cross entropy loss clearly decreases in the initial half of training, and plateaus for the remaining epochs. This

implies that models trained with the standard loss are unable to produce any meaningful level of attribution robustness or that they are vulnerable to attribution manipulation. Both robust objectives increase in top-k intersection over time, signaling their capability to achieve the desired goal of improved attribution robustness. The novel CE-of-attacks objective proposed in this work maintains higher attribution robustness than the IG-Norm [5] objective throughout both training sessions in Figure 3.3.

We make another interesting observation with the top-k intersection and test accuracy curves on the CIFAR-10 dataset (Figure 3.3: right). While both robust objectives maintain higher top-k intersection than standard training, their test accuracy is consistently lower than standard training. This relative difference in test accuracy is present throughout the training epochs, which can be considered a strong indication of the previously discussed trade-off between attribution robustness and accuracy. On the test accuracy metric too, CE-of-attacks performs better than the IG-Norm objective through the CIFAR-10 training session.

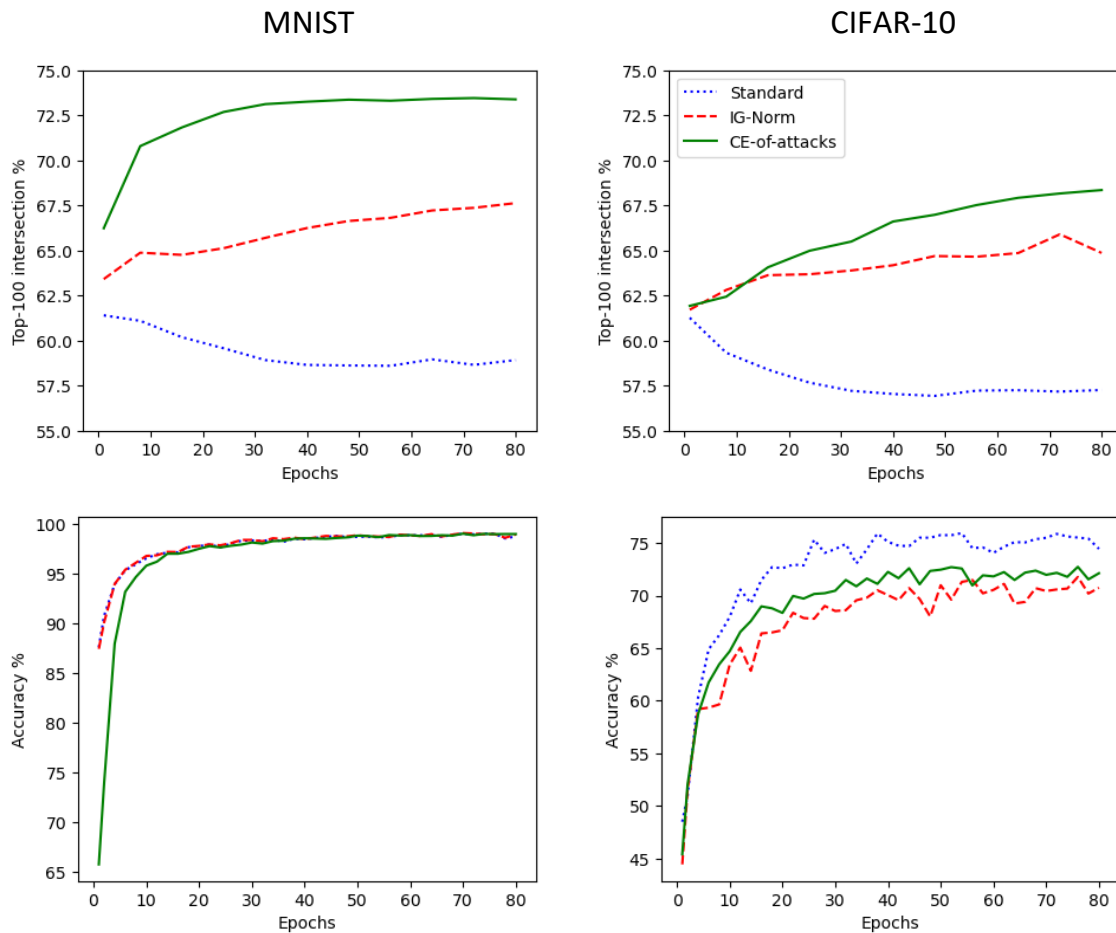


Figure 3.3: Evolution of attribution robustness (top-k intersection, top graphs) and test accuracy (bottom graphs) during training with the three objective functions (standard, IG-Norm, CE-of-attacks) on MNIST (left) and CIFAR-10 (right) datasets.

## 3.4 Conclusion

In this chapter, a novel, generic and flexible framework of attribution robust training objectives was introduced. A fast and effective attribution robust objective termed “cross entropy of attacks” (CE-of-attacks) was derived from this framework. Comprehensive experimental analysis confirmed that CE-of-attacks outperforms the existing IG-Norm robust objective in terms of both attribution robustness (Kendall correlation, top-k intersection) and accuracy.

## Chapter 4

# Challenges of Training Attribution Robust Neural Networks and Efficient Training Methods

Attribution robust training of neural networks is a more challenging task than standard training (cross entropy loss) due to several key issues: the high computational cost, the trade-off between attribution robustness and accuracy, and the difficulty of hyperparameter tuning. The existing literature on attribution robustness contains no meaningful analysis or discussion on these topics. To address this gap, we conduct a thorough analysis of the challenges of training attribution robust neural networks, and propose guidelines for mitigating these challenges to some extent.

The major challenge of robust training is its very high computational cost compared to standard training. The CE-of-attacks objective introduced in the previous chapter is 2 times faster than the existing IG-Norm objective. However, in its original form, the CE-of-attacks objective is still significantly slower than standard training (up to 25 times slower). In this chapter, we propose several techniques to significantly improve the efficiency of attribution robust training of neural networks with the CE-of-attacks objective.



## 4.1 Contributions of the chapter

The research contributions of this chapter are summarized below.

- **Computational cost analysis of attribution robust training** (subsection 4.2.1). Although it is obvious that attribution robust training is computationally more expensive than standard training, the existing literature contains no analysis on this important aspect. We provide a detailed algorithmic and empirical analysis of the computational cost of several attribution robust training objectives. Our analysis reveals that robust training with the prior IG-Norm objective is 16 to 67 times slower than standard training, while the proposed CE-of-attacks objective is only 2 to 35 times slower than standard training.
- **Trade-off between attribution robustness and accuracy** (subsection 4.2.2). Robust training experiments reveal that there is a fundamental trade-off between attribution robustness and test set accuracy. The existing literature lacks any analysis on this aspect of robust training. We provide empirical results that quantify the trade-off by conducting robust training under varying attack strengths. For example, training a small CNN on the CIFAR-10 dataset with varying distance budgets for the generated attacks yields a 17.5% gain in top-100 intersection at an accuracy drop of 7.8%.
- **Hyperparameter tuning of attribution robust training** (subsection 4.2.3). Hyperparameter tuning in robust training is difficult due to three reasons: the additional hyperparameters, the high computational cost and the trade-off between attribution robustness and accuracy. We provide a set of guidelines to mitigate these issues and find good hyperparameter configurations for attribution robust training of neural networks.
- **Techniques for efficient robust training** (section 4.3). We propose three efficiency improvement techniques to address the major problem of high computational cost of robust training: combining standard and robust training (2x - 4x speed gain), using a fast one-step attack in the training loop (3x or more speed gain), and tuning the number of steps in the integrated gradients computations (2x - 6x speed gain). Experiment results demonstrate that the efficiency gains are achieved with no significant performance degradation in terms of attribution robustness and accuracy.

## 4.2 Challenges in training attribution robust neural networks

### 4.2.1 Computational cost of attribution robust training

The main challenge of attribution robust training is its higher computationally cost compared to standard training. We consider the number of forward and backward (backpropagation or gradient computation) passes needed for one training step (one mini-batch weight update) of a neural network, as it is a practically useful measurement and is applicable to any neural architecture (Table 4.1).

The following notation is used in the analysis:  $f$  = no. of forward passes,  $b$  = no. of backward passes,  $p$  = no. of iterations in attack loop,  $m_a$  = IG steps in inner attack loop,  $m_n$  = IG steps in outer IG norm term. In robust training objectives, there are two sources of computational complexity, assuming integrated gradients (IG) is the choice of attribution computation method;

- The inner attack generation loop: a projected gradient descent loop ( $p$  iterations) is used to compute the attacks. Inside each iteration of the attack loop, integrated gradient ( $m_a$  IG approximations steps) computations are performed, resulting in  $p \cdot m_a$  forward and backward passes.
- The outer loss term computation and gradient descent minimization loop (only in the case of the IG-Norm objective). If the loss function has integrated gradients ( $m_n$  IG approximations steps), this step will incur additional cost of  $m_n$  forward and backward passes.

The computational cost of different training approaches is summarized in Table 4.1. For example calculations, we use two settings: computing a strong multi-step attack with  $p = 10$  iterations, and a weak one-step attack ( $p = 1$ ). For both cases we use  $m_a = 4$  and  $m_n = 40$ , values that were empirically found to produce well-trained (good test accuracy) attribution robust models (good Kendall’s tau and top-k intersection values).

The number of forward and backward passes in Table 4.1 indicate that IG-Norm and IG-Sum-Norm [5] robust objectives are extremely expensive compared to standard training in both the strong attack setting (81 times slower) and the weak attack setting (45 times slower). The CE-of-attacks objective proposed in this work is also very expensive in the strong attack setting

(41 times slower than standard training), but less so in the weak (one-step) attack setting (only 5 times slower than standard training).

Table 4.1: Computational cost of training objectives.  
 $p$  = no. of iterations in attack loop,  $m_a$  = IG steps in inner attack loop,  
 $m_n$  = IG steps in outer IG norm term. For example values  $m_a = 4$ ,  $m_n = 40$

Objective	No. forward ( $f$ ) and backward ( $b$ ) passes	Example values: multi-step attacks ( $p = 10$ )	Example values: one-step attacks ( $p = 1$ )
Standard	$f = 1$ $b = 1$	-	-
IG-Norm, IG-Sum-Norm	$f = 1 + p.m_a + m_n$ $b = 1 + p.m_a + m_n$	$f = 81$ $b = 81$	$f = 45$ $b = 45$
CE-of-attacks	$f = 1 + p.m_a$ $b = 1 + p.m_a$	$f = 41$ $b = 41$	$f = 5$ $b = 5$
Robust prediction [37]	$f = 1 + p$ $b = 1 + p$	$f = 11$ $b = 11$	$f = 2$ $b = 2$

**Measuring wall-clock time.** Implementation details (eg: batching etc.) and compute environment (eg: tensor-optimized GPUs) can heavily affect the actual run time of training sessions. Therefore, experiments were run to measure wall-clock time of different training objectives on a single A100 GPU environment. The results in Table 4.2 show that the observed slowdown in robust training is not as severe as suggested by the algorithmic analysis (Table 4.1) for the Small-CNN model. However, the ResNet-18 model approaches the theoretical slowdown multiples.

Overall, it is evident that attribution robust training is significantly slower than standard training (2x to 67x slower), and the IG-Norm objective is prohibitively inefficient for medium to large neural networks. **The proposed CE-of-attacks robust objective with a one-step attack ( $p = 1$ ) is the most efficient out of all training configurations (2x to 4x slower than standard training).** This implies that the CE-of-attacks objective is a good candidate for training attribution robust models at scale.

## 4.2.2 Trade-off between attribution robustness and accuracy

In the attribution robustness experiment results and discussion (subsection 3.3.2), we noted that there appears to be a trade-off between attribution robustness and accuracy, i.e. models trained

Table 4.2: Time taken (seconds) per one epoch of training for different objectives. The number within parenthesis is the slowdown factor for robust objectives relative to standard training. The fastest robust objective is highlighted in bold.  $p$  = no. of iterations in attack loop.

Dataset & model	Standard	IG-Norm		CE-of-attacks	
		$p = 1$	$p = 10$	$p = 1$	$p = 10$
CIFAR-10 [Small-CNN]	7.5	113 (16x)	306 (41x)	<b>23 (4x)</b>	72 (10x)
RIMB [Small-CNN]	38	708 (19x)	1511 (40x)	<b>74 (2x)</b>	606 (16x)
RIMB [ResNet-18]	36	1112 (31x)	2400 (67x)	<b>140 (4x)</b>	1240 (35x)

for high attribution robustness lose test accuracy.

To confirm the existence of this phenomenon, we conduct an experiment with the CE-of-attacks robust objective proposed in this work. The Small-CNN model is trained on the CIFAR-10 dataset using the CE-of-attacks objective with increasing attack strength. Attack strength is increased by setting larger values for the distance budget  $\epsilon$ , so that the attack examples can steer far away from the natural examples. A one-step attack is used with attack step size  $\alpha$  set to half the distance budget ( $\alpha = \epsilon/2$ ).

The results of this experiment are shown in Figure 4.1. It is clear that attribution robustness (measured by top-k intersection) and test accuracy have a negative correlation. Between the two models trained with the standard cross-entropy loss and the CE-of-attacks objective, a gain in attribution robustness of 17.5% top-100 intersection is achieved at the cost of a 7.8% drop in accuracy.

Note that this is not an isolated effect due to the increased distance budget. In our experiments, any technique used to increase attribution robustness by any amount always results in a drop in accuracy. This includes generating stronger attacks with more iterations (Figure 4.3) and increasing the regularization strength ( $\lambda$ ) in the IG-Norm objective. Similar findings have also been reported in the literature [25, 10]. **This broad empirical evidence suggests that there is a fundamental trade-off between attribution robustness and model accuracy.**

**Why is the trade-off a challenge and how to manage it?** Practically, the existence of the trade-off means that it is difficult to train models to have both high attribution robustness and very high (state-of-the-art) accuracy. Depending on the importance of attribution robustness within the application, a practitioner will need to decide the level of acceptable values for these two properties and tune the models towards those values. It is important to be cognizant of the existence of this trade-off, so that resources can be optimally allocated (not wasted on trying to

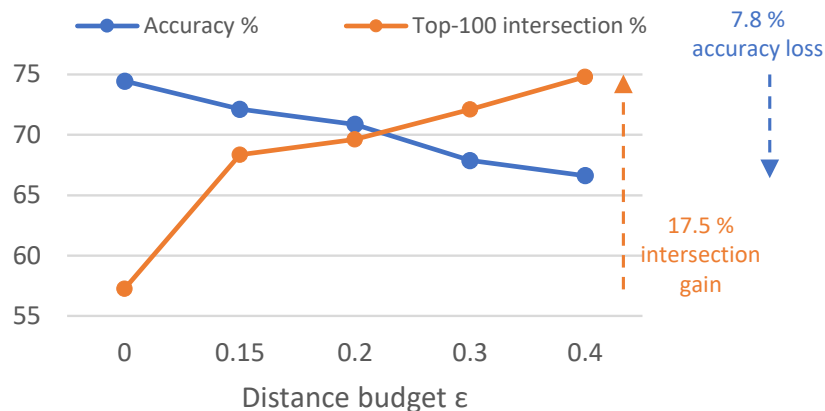


Figure 4.1: Trade-off between attribution robustness and test accuracy.  $\epsilon = 0$  is standard training, the other data points represent the CE-of-attacks objective.

improve both together at the edge).

### 4.2.3 Hyperparameter tuning

Hyperparameter tuning is especially challenging with attribution robust objectives due to three reasons: the additional hyperparameters involved (larger search space), the significantly high training time (hard to obtain results from many runs), and the trade-off between attribution robustness and accuracy (optimizing for one will sacrifice the other). These factors make it difficult to use common hyperparameter tuning algorithms such as random search or Bayesian optimization for the robust training case. We suggest the following approach to hyperparameter tuning for attribution robust training, in the hope that researchers and practitioners will find it a useful heuristic to guide the tuning process.

- Start by finding a good hyperparameter configuration for standard training. The relevant parameters may include neural network architecture, batch size, learning rate, number of training epochs needed for test error convergence etc. These parameters will form the basis for the outer gradient descent loop in robust training.
- Find a good set of values for the hyperparameters in the inner attack generation loop, separately from the training loop. The main parameters are: distance budget ( $\epsilon$ ), step size ( $\alpha$ ), no. of iterations ( $p$ ), no. of integrated gradient steps ( $m_a$ ). Some practical considerations are;

- It is a good idea to visualize the attacks generated and their heatmaps to verify that the attacks are not very dissimilar to the natural examples and that attribution maps have been distorted.
  - Distance budget ( $\epsilon$ ) and step size ( $\alpha$ ) tend to have the most impact on attack strength. Heuristics for these parameters may be found in the literature on attribution robustness as well as adversarial robustness. For one-step attacks we used  $\alpha = \epsilon/2$ , and for multi-step attacks  $\alpha = 2.\epsilon/p$  produced good attacks.
  - The other two parameters that affect attack strength and quality are the number of attack iterations ( $p$ ) and the number of integrated gradient steps ( $m_a$ ) in the attack loop. In this work, we have shown that a one-step attack ( $p = 1$ ) and a small number of IG steps ( $m_a = 4$ ) is sufficient for robust training. Similarly, attempt to find the most efficient values for these two parameters that will generate the attacks needed for robust training.
  - One can track the attack strength in the training loop to verify that the attack is strong and suitable for the task by measuring Kendall tau and the  $\|IG(x, x')\|_1$  terms as seen in Figure 4.2.
- 
- Once the hyperparameter value ranges have been found for the outer gradient descent loop and the inner attack generation loop, combine them and run training sessions. It is also possible to use a smaller subset of the dataset to run shorter sessions. Manual binary-like search within the hyperparameter ranges produced good results in our work.
  - At this point, it may be possible to do a random search or a Bayesian optimization within the constrained range of values.
  - Track all relevant metrics (training loss, test accuracy, attribution robustness metrics such as Kendall tau and top-k intersection) throughout training epochs (eg: using TensorBoard). This will highlight important effects even if a training run fails to achieve the desired end values for accuracy and attribution robustness.
  - Be mindful of the trade-off between attribution robustness and accuracy. This means that if tuning a parameter in a certain direction yields increasing attribution robustness, we must expect a drop in accuracy. This will help find the desired balance between the two.

## 4.3 Efficient training of attribution robust neural networks

The analysis of computational cost attribution robust training (subsection 4.2.1) highlights the crucial need for efficient attribution robust training methods, which is a lacking area in the current literature. We have made attempts to improve computational efficiency in the following ways.

- **Attack strength.** A major source of computational cost is the attribution attack generation inner loop. Creating a strong attack requires many iterations of the inner loop, leading to higher cost. We examine the possibility of using a weak one-step attack, and find that a one-step attack with proper hyperparameters is able to produce neural nets with high attribution robustness (subsection 4.3.1).
- **Combined training.** Since attribution robust training is inherently costly, a good question to ask is “what is the minimum amount of robust training needed to achieve a desired level of attribution robustness?” We find that running standard training followed by a relatively small number of robust training epochs is sufficient to achieve high levels of attribution robustness (subsection 4.3.2).
- **Tuning the IG steps parameter.** It is important to compute integrated gradients (IG) in the attack loop with the minimal number of approximation steps that still produce well-trained neural nets with high attribution robustness. The effect of the IG steps parameter on attribution robustness and the tuning process is described in subsection 4.3.3.

### 4.3.1 Strength of the attribution attack generated in the training loop

When training a neural network with attribution robust objectives (IG-Norm, IG-Sum-Norm, CE-of-attacks), a batch of attribution attacks is generated for each mini-batch of natural training data. Attack generation is the search for a solution with projected gradient descent (PGD) to the inner maximization seen in many robust objectives, as illustrated in Algorithm 3. Intuitively, generating stronger attacks should lead to increased attribution robustness. Two key parameters determine the strength of the generated attack: the distance budget  $\epsilon$  (the norm constraint for the attack around the natural example), and the number of PGD iterations  $p$ .

The distance budget  $\epsilon$  is set such that the attacks are not perceptually very different from the nat-

ural examples, and it does not affect the computational cost of training. However, as discussed in subsection 4.2.1, the computational cost of training increases linearly with the number of iterations  $p$  in the attack loop (and therefore training slows down significantly).

For the classic adversarial learning setting which also involves a generating an attack by an inner loss maximization loop, efficient training methods have been proposed that achieve the desired adversarial robustness with one-step attacks [71]. Inspired by this, we set out to find if weak one-step attacks in the robust objectives are sufficient to train neural networks with attribution robustness while being significantly faster than strong multi-step attacks. The key finding is that it is indeed possible to achieve attribution robustness with one-step attacks.

### Comparing the strength of one-step and 10-step attacks

---

**Algorithm 3** Attribution attack generation loop with PGD (inner maximization):

$$x' = \max_{x' \in N(x, \epsilon)} \|IG(x, x')\|_1$$


---

**Input:**

Mini-batch of natural examples:  $(X, Y)$

Attack generation loop hyperparameters:

No. of iterations ( $p$ ), step size ( $\alpha$ ), distance budget ( $\epsilon$ ), IG steps: ( $ig\_steps$ )

```

1:  $X\_attacks \leftarrow X$  // Initialize batch of attacks at the original examples
2: for  $j \leftarrow 1$  to  $p$  do
3:    $IG \leftarrow integrated\_gradients(X, X\_attacks, Y, ig\_steps)$ 
4:    $IG\_norm \leftarrow L1\_norm(IG)$ 
5:    $grad \leftarrow gradient(IG\_norm, X\_attacks)$  // Gradient of  $IG\_norm$  w.r.t.  $X\_attacks$ 
6:    $X\_attacks \leftarrow X\_attacks + \alpha * grad$  // Gradient ascent update in PGD
7:    $X\_attacks \leftarrow clamp(X\_attacks, \epsilon)$  // Projection in PGD
8: end for

```

**Output:**

Attribution attack batch  $X\_attacks$

---

As can be seen in Algorithm 3, the PGD loop finds a perturbed example batch (attack batch:  $X\_attacks$ ) that maximizes the L1 norm ( $IG\_norm$ ) of the integrated gradients between  $X$  and  $X\_attacks$ . In order to understand the evolution of attack strength in the attack generation loop, we observe the change in the  $IG\_norm$  and the Kendall correlation between intermediate



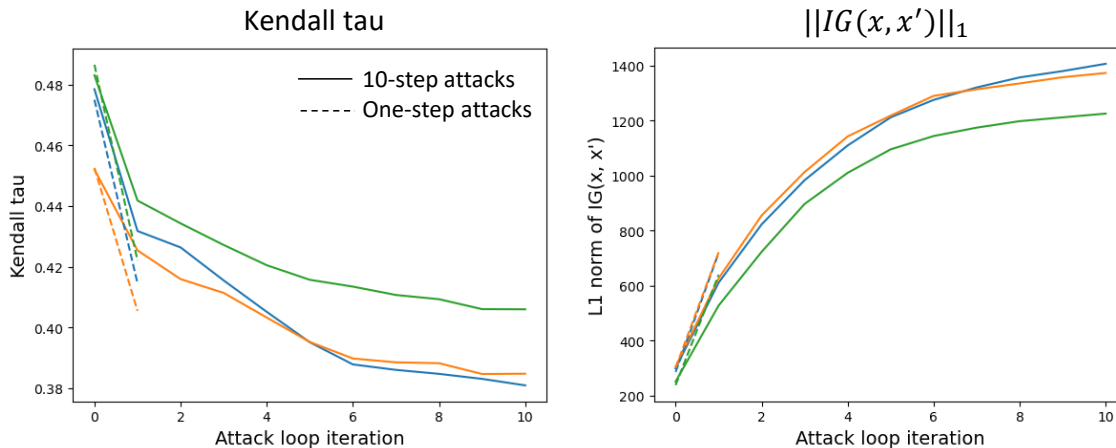


Figure 4.2: IG attack strength through the attack loop for 3 data batches (one-step vs 10-step attacks).

attacks and the natural example batch in a 10-step PGD loop (parameters:  $p = 10$ ,  $\epsilon = 0.1$ ,  $\alpha = \epsilon/5 = 0.02$ ). To compare with the attack strength of a one-step attack, we plot the same metrics for a one-step attack with a larger step size (parameters:  $p = 1$ ,  $\epsilon = 0.1$ ,  $\alpha = \epsilon/2 = 0.05$ ).

Several observations can be made from the results of this experiment Figure 4.2. The attack strength of the 10-step attack increases faster at first and slower towards the end of the loop (expected from a gradient descent optimization). The one-step attack with large step size achieves a significant percentage of the strength of the 10-step attack as measured by the  $IG\_norm$  (32% - 47% over test set) and Kendall tau (52% - 94% over test set) metrics. These observations hint that it may be possible to use the significantly cheap one-step attack in a robust objective and still achieve an acceptable level of attribution robustness.

### Attribution robust training with one-step and multi-step attacks

To verify the efficacy of the one-step attack with the CE-of-attacks robust objective, we train the Small-CNN model on the CIFAR-10 dataset with the attack iterations hyperparameter set to  $p = 1, 2, 4, 6, 8$ . The results are shown in Figure 4.3. The one-step attack in the CE-of-attacks method achieves a top-k intersection of 68.35%, which is only 2.65% less than the value achieved by the best multi-step attack (4 steps). **This suggests that a one-step attack is sufficient for the CE-of-attacks objective to train attribution robust neural networks.**

The immediate advantage of training with the one-step attack is that it is more than 3 times

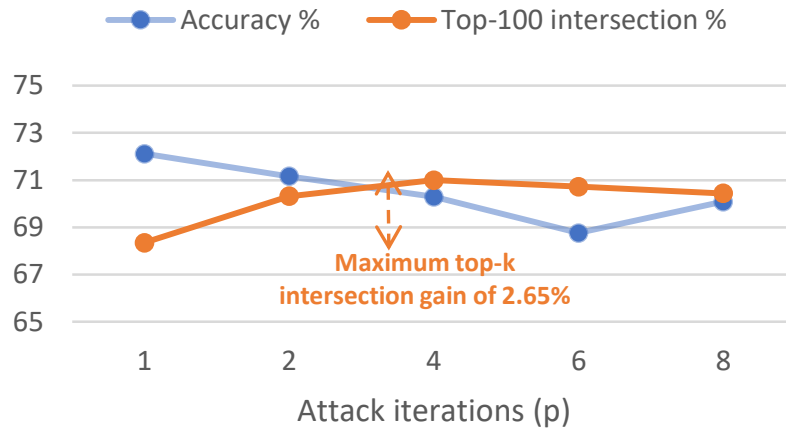


Figure 4.3: Attribution robustness and accuracy change with no. of iterations in the attack loop

faster than a 10-step attack (23 seconds per epoch vs. 72 seconds per epoch). For large neural networks and datasets, the CE-of-attacks objective with the one-step attack may be the only practical configuration for attribution robust training. Furthermore, it opens up the possibility of more efficiency improvement techniques used in the adversarial robustness literature (eg: gradient recycling [57]).

### 4.3.2 Combining standard and robust training

Given the computational expense of attribution robust training, we explore the possibility of doing a minimal amount of robust training to achieve the desired level of attribution robustness. Our findings indicate that conducting robust training for only 12.5% - 50% of epochs can achieve comparable robustness to full training, potentially speeding up the process by 2x - 4x.

**Combined training experiments.** First, the Small-CNN model is trained on the CIFAR-10 dataset with the standard cross entropy loss for many epochs. Next, the loss function is changed to the CE-of-attacks robust objective for the remainder of the training. We also perform experiments to observe the effect of alternating between standard and robust objectives every other epoch. The total no. of training epochs is 80 in this experiment setup.

This idea was inspired by the observation that attribution robustness soon reaches a high level and plateaus during robust training. Changing loss functions for different training regimes is also a common strategy in some machine learning settings. For example, in self-supervised

learning, models are pre-trained on large datasets with one loss function, followed by fine-tuning on downstream tasks with different loss functions.

The results of the experiments are shown in Table 4.3. It is clear that training the neural network with the robust objective in the last 10 to 40 epochs is sufficient to achieve a high level of attribution robustness as measured by top-k intersection. In fact, the highest top-k intersection value of 68.87% is produced by the “Last 50% robust” training configuration. The highest drop of top-k intersection is seen in the “Last 12.5% robust” setting, which is 2.93% less than full robust training (relatively small drop).

Table 4.3: Combined training results (standard and CE-of-attacks [CEA]).

Training configuration	Standard epochs	CEA epochs	Natural accuracy %	Top-100 intersection %
Full robust training	0	80	72.12	68.35
Last 50% robust	40	40	73.47	<b>68.87</b>
Last 25% robust	60	20	<b>73.61</b>	66.94
Last 12.5% robust	70	10	73.58	65.42
Alternating robust (50%)	40	40	72.23	68.12

In the Small-CNN on CIFAR-10 training sessions with 80 epochs, the speed gain of doing 50% to 87% less robust training may not seem significant. However, for large neural networks trained on large datasets with high-dimensional inputs, the robust training epochs dominate the overall training time. In such cases, the speed gain of combined training can be estimated to be in the range of 2x - 4x.

Figure 4.4 shows the evolution of top-k intersection during the different training runs of this experiment. The full-robust and alternating-robust training setups have a gradual increasing top-k intersection curve. In the three training configurations where standard cross entropy is used for the first set of epochs, the top-k intersection slowly decreases at first. When the loss function is switched to CE-of-attacks for the last n epochs (last 10, 20, 40 epochs), the curve shows a sharp increase towards the full-robust-training curves.

This effect of catching up to the full-robust curves in a small number of epochs is interesting, as it suggests that well-trained weights may be a better starting point than randomly initialized weights when training for attribution robustness. The top-k intersection evolution curves are a strong indication that combining objectives is an effective way to achieve attribution robustness while minimizing the cost of robust training.

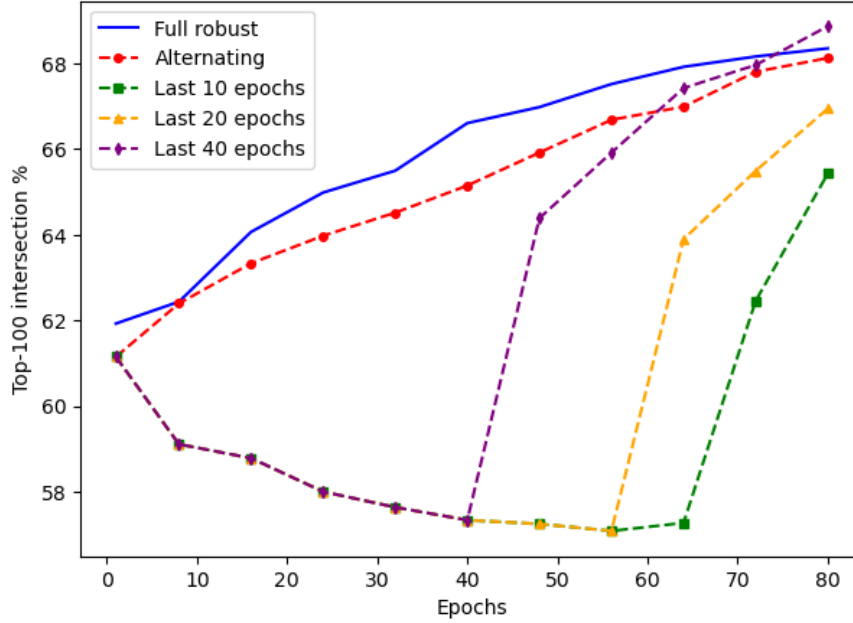


Figure 4.4: Switching the training objective during training: all-epochs-robust vs. last-few-epochs-robust

### 4.3.3 Tuning the integrated gradients steps parameter

As discussed in the computational cost analysis (subsection 4.2.1), the integrated gradients (IG) computation in the attack generation loop is a major source of cost in the overall training process. The IG computation involves an approximation of a path integral, where different points of the path are sampled (the IG steps). The typical number of IG steps needed for high quality attribution maps is 20 - 300 [64].

However, since the IG calculations happen in the inner attack generation loop, a large number of steps will significantly slow down the training process. Therefore, it is important to tune the IG steps parameter in the attack loop ( $m_a$ ): it must be low enough so the training is fast, but high enough that the IG approximation yields the desired attribution robustness.

To investigate the effect of the IG steps parameter ( $m_a$ ) on training time and attribution robustness, we train the Small-CNN model on the CIFAR-10 dataset with  $m_a$  values in the range of 2 - 24 for the CE-of-attacks robust objective.

The results are shown in Table 4.4. As expected, the lowest top-k intersection is given by  $m_a = 2$ , and the highest value is given by  $m_a = 24$  at the cost of a 6x slowdown in training. The

difference in the achieved top-k intersection between the two settings is 3.52%. This suggests that small values for  $m_a$  produce attribution maps sufficient for generating the IG attacks in the CE-of-attacks objective. With increasing  $m_a$ , the resulting increase in top-k intersection may not justify the corresponding slowdown in training. In all our experiments, we have chosen  $m_a = 4$  as an acceptable balance between training speed and attribution robustness.

Table 4.4: Tuning the IG steps parameter ( $m_a$ ) in the CE-of-attacks objective.

IG steps ( $m_a$ )	Natural accuracy %	Top-100 intersection %	Seconds per epoch	Speedup relative to $m_a = 24$
2	<b>72.90</b>	66.11	7	6.3x
4	72.12	68.35	10	4.4x
8	72.33	69.24	16	2.8x
16	71.43	68.99	30	1.5x
24	71.79	<b>69.62</b>	44	-

## 4.4 Conclusion

In this chapter, challenges of robust training was discussed in detail with extensive empirical results, including the problem of high computational cost, the trade-off between attribution robustness and accuracy and the difficulty of hyper-parameter tuning. Several techniques were proposed to further improve efficiency of the robust training loop. We believe that the techniques and the comprehensive empirical analysis on challenges of attribution robust training presented in this chapter is a valuable compilation of information for machine learning researchers and practitioners.

# Chapter 5

## Properties of Attribution Robust Neural Networks

Attribution robust models achieve robustness in the input feature space at high computational cost. With this knowledge, a good research question to ask is, "what other useful properties do attribution robust models have?" A similar theme of research in the adversarial or prediction robustness literature has uncovered several important properties of prediction robust models; for example, they are known to have higher transfer accuracy on downstream tasks [52, 66]. However, there is no exploration in the existing literature on the properties of attribution robust neural networks.

To address this gap, in this chapter we investigate two properties of attribution robust models: robustness to image corruptions and resilience to spurious correlations.

### 5.1 Contributions of the chapter

The key research contributions and findings presented in this chapter are as follows.

- Attribution robust neural networks show robustness to common image corruptions (eg: blurring, digital artifacts, etc.). This is demonstrated by the higher performance of attribution robust models compared to standard-trained models on corrupt images. The

robust models achieve accuracy gains in the range of 3.58% to 11.94% over standard trained models on three corrupt versions of popular image datasets: MNIST-C, CIFAR-10-C and RIMB-C.

- Prior work has hypothesized that attribution robust neural networks may be robust in the presence of spurious correlations [5]. Our experiments reveal that this is not the case, as measured by worst-group accuracy on datasets with spurious correlations. On the contrary, we find that the IG-Norm robust objective could result in perfect misclassification of minority groups.

## 5.2 Robustness to image corruptions

In this section, we describe the experiments and findings on the relationship between attribution robustness of a neural network classifier and its robustness to common corruptions.

### 5.2.1 Experiment details

**Models, training sets and training objectives.** We use the same neural network architectures (Small-CNN and ResNet-18) and hyperparameter configurations described in section 3.3 for training. Specifically, the Small-CNN model is trained on the MNIST, CIFAR-10 and RIMB datasets and the ResNet-18 model is trained on the RIMB datasets. The models are trained with the three objectives: standard, CE-of-attacks and IG-Norm.

**Evaluation metrics.** In order to evaluate the robustness of neural network classifiers to common corruptions and perturbations, we evaluate the accuracy of trained models on several corrupted datasets (described in the next section). As a measurement of corruption-robustness of attribution robust models, we calculate the accuracy difference between the robust model and the standard model (Equation 5.1) as well as the percentage improvement of the accuracy (Equation 5.2). Note that a negative value for these metrics indicate that the attribution robust model performs poorly in comparison to the standard model on the corrupted datasets.

$$\text{Accuracy Difference} = \text{Robust model accuracy} - \text{Standard model accuracy} \quad (5.1)$$

(Acc\_Diff)

$$\text{Accuracy Improvement } \% = \frac{\text{Robust model accuracy} - \text{Standard model accuracy}}{\text{Standard model accuracy}} \times 100\% \quad (5.2)$$

(Acc\_Improv)

**Benchmark datasets.** The corrupted datasets are taken from the existing literature, so that the benchmark is standard, and our work is easy for researchers to reproduce and compare with other techniques. The datasets are MNIST-C [42] ('C' stands for 'Corrupted'), CIFAR-10-C [23], and RIMB-C ('Restricted ImageNet Balanced - Corrupted' subset of the ImageNet-C in [23]). The CIFAR-10-C and RIMB-C datasets contain test set images corrupted with different types of noise and artifacts at multiple severity levels (1 to 5). Corruption types include the following;

- Noise: gaussian noise, shot noise, and impulse noise
- Blur: defocus blur, glass blur, motion blur, and zoom blur
- Weather: frost, snow, fog, and brightness
- Digital: contrast, elastic transform, pixelate, JPEG compression
- Other artifacts: speckle noise, spatter, gaussian blur, saturate, stripe

Details on dataset content are given in Table 5.1, and Figure 5.1 shows a subset of the corruptions. section A.2 provides details on the exact corruption types in each dataset and more visualizations.

Table 5.1: Benchmark image datasets containing corruptions and perturbations

Dataset	No. of corruptions	No. of examples
MNIST-C [42]	- 15 corruption types	- 10,000 examples in each corruption type - 150,000 total examples
CIFAR-10-C [23]	- 19 corruption types - 5 severity levels	- 10,000 examples in each corruption type - 950,000 total examples
RIMB-C [23]	- 19 corruption types - 5 severity levels	- 1,400 examples in each corruption type - 133,000 total examples



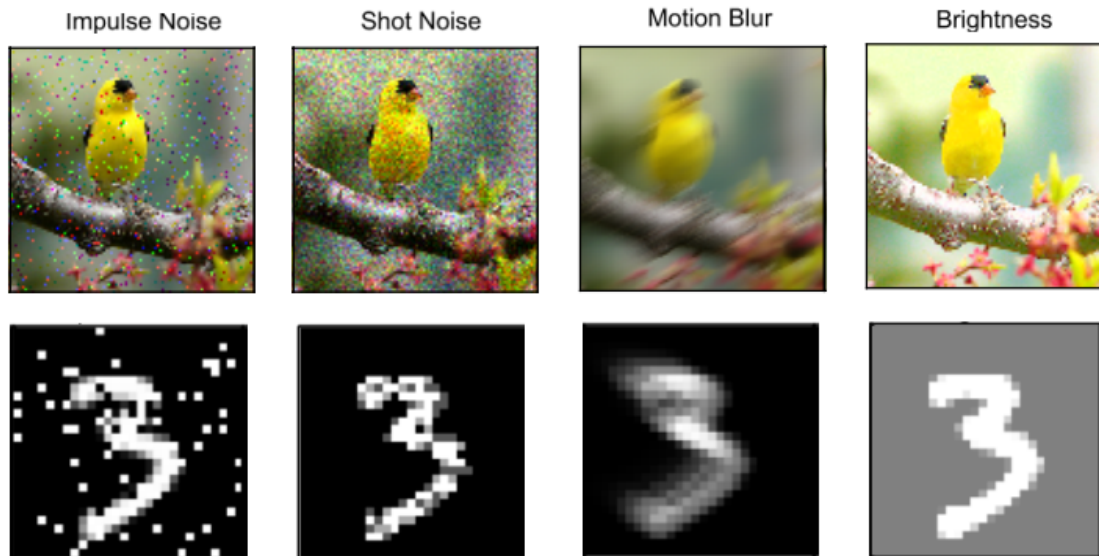


Figure 5.1: Visualization of corruptions in RIMB-C [23] and MNIST-C [42] datasets.

### 5.2.2 Results and discussion

**Overall robustness to corruptions.** Table 5.2 and Figure 5.3 show the accuracy improvement of two robust training objectives (IG-Norm, CE-of-attacks) compare to the standard cross-entropy loss on all corruption types and severity levels in each benchmark datasets. Neural nets trained with the CE-of-attacks objective clearly achieves a considerable accuracy gain over the standard-trained models on all datasets, ranging from a 3.58% gain in the RIMB-C [Small-CNN] case to a 11.94% gain on the MNIST-C dataset. These accuracy gains or differences correspond to large accuracy improvement percentages over the baseline standard models (6.1% to 19.58%)

On the MNIST-C and CIFAR-10-C datasets, the IG-Norm objective slightly outperforms the CE-of-attacks objective (gap of 0.86% on MNIST-C). However, on the RIMB-C dataset, the IG-Norm objective underperforms the standard training objective with both ResNet-18 and Small-CNN architectures. The worst case scenario is the ResNet-18 model, where the IG-Norm objective gives 11.45% less accuracy than the standard model (not illustrated in Figure 5.3). This behavior is expected, as the IG-Norm-trained ResNet-18 model has a standard accuracy (clean RIMB dataset) of 67.93% compared to standard-trained model accuracy of 87.14% (a 19.21% drop). In other words, IG-Norm training of the ResNet-18 model performs so poorly on clean data that it is unable to perform well on the corrupt data.

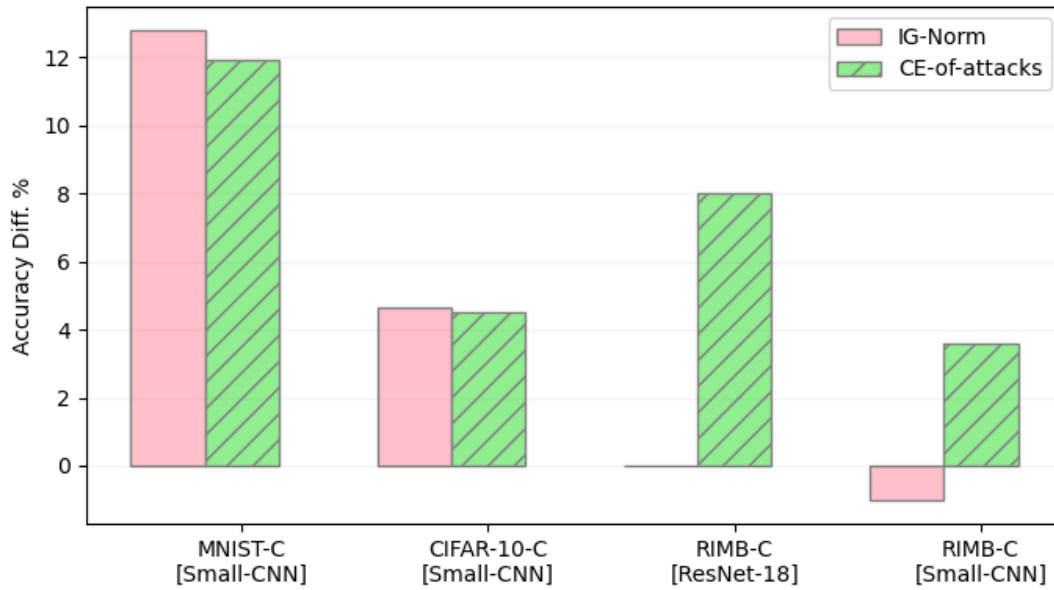


Figure 5.2: Overall accuracy difference (Acc\_Diff%) of robust models compared to standard models

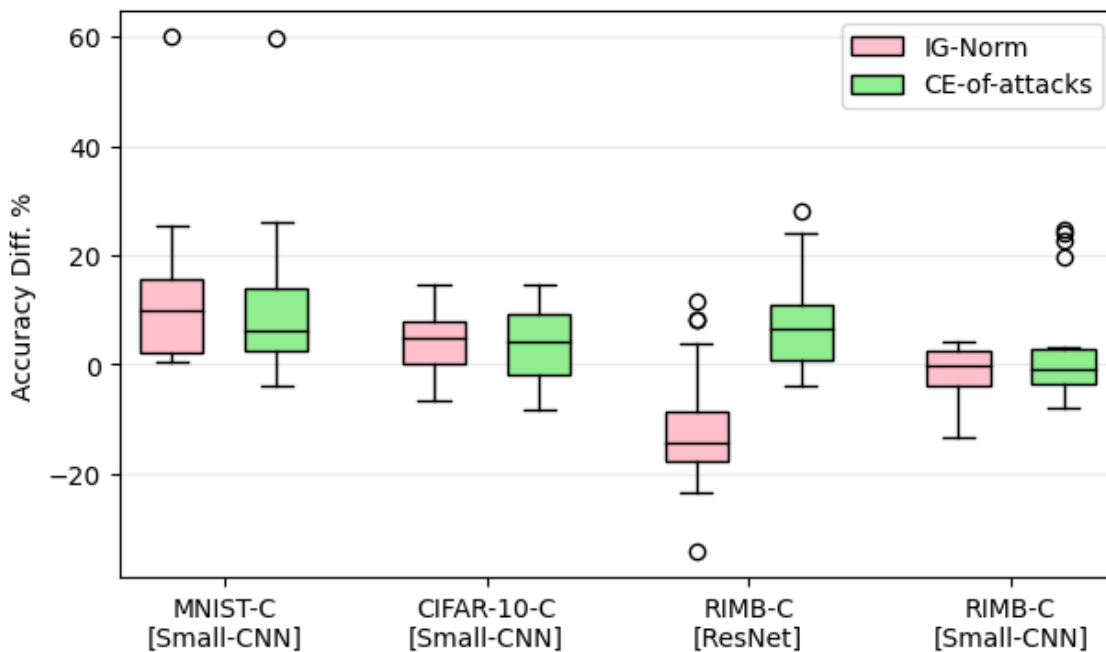


Figure 5.3: Distribution of accuracy difference (Acc\_Diff%) of robust models compared to standard models over different corruption types. The middle horizontal bar is the median, box boundaries are the 25th and 75th percentiles, and whiskers show the rest of the distribution with outliers shown as circles.

Table 5.2: Overall accuracy difference (Acc\_Diff%) and improvement percent (Acc\_Improv) of robust models compared to standard models on the corrupt datasets.

Dataset & model	Acc_Diff %		Acc_Improv %	
	IG-Norm	CEA	IG-Norm	CEA
MNIST-C [Small-CNN]	<b>12.8</b>	11.94	20.99	19.58
CIFAR-10-C [Small-CNN]	<b>4.65</b>	4.49	8.05	7.78
RIMB-C [ResNet-18]	-11.45	<b>8.03</b>	-17.91	12.56
RIMB-C [Small-CNN]	-1.02	<b>3.58</b>	-1.74	6.10

**Robustness to different corruption types.** So far, we looked at the accuracy of the robust models on the overall corruption datasets (all corruption types). Figure 5.3 illustrates the range or distribution of accuracy gains (or drops) given by robust training on different corruption types. While the accuracy difference distribution of the CE-of-attacks objective has a positive tilt (hence the average accuracy gain), the median and lower percentiles have crossed into the negative side. This indicates that on some corruption types, models trained with the CE-of-attacks have underperformed the standard-trained models.

However, there are no negative outliers, which shows that the accuracy drop of the CE-of-attacks model among different corruption types is limited. The positive outliers show that the CE-of-attacks objective performs extremely well on some corruptions (60% accuracy gain in one MNIST-C corruption, over 17% gain on 3 RIMB-C corruptions).

**These results strongly suggest that attribution robust training, especially with the CE-of-attacks objective proposed in our work, produce neural networks that are also robust against a multitude of image corruptions.**

Importantly, a high degree of robustness was achieved with no knowledge of the different corruption types and their generation process at training time (unknown corruption types in the evaluation data). In contrast, many methods need to have knowledge of the potential corruptions or slight distribution shifts and train for those specific corruptions (eg: with strong data augmentation where augmentations are made by making specific corruptions of the training data). In other words, attribution robust training is a generic technique that has the capability to build robustness to data corruptions into neural network classifiers.

## 5.3 Resilience to spurious correlations

Spurious correlations are features or shortcut patterns in data that a model may incorrectly learn to associate with a target variable. For example, an image classifier may learn to associate flowers in the background with the class “butterfly”. This may cause the model to classify any image with flowers as butterfly, and actual butterfly images without flowers into other classes. The problem of deep neural networks learning spurious correlations has been extensively studied in the machine learning literature [50, 26, 51].

It has been hypothesized that attribution robust models may be resilient in the presence of spurious correlations [5]. However, the relationship between attribution robustness, and resilience to spurious correlations has not been explored or established in the current literature. In this work, we set out to answer the question “does attribution robust training give neural network classifiers the ability to avoid spurious features in training data?”

### 5.3.1 Experiment details

**Models and training objectives.** We train ResNet-18 models (initialized with ImageNet pre-trained weights) with the three objective functions: standard cross entropy, IG-Norm and CE-of-attacks.

**Datasets.** We train and evaluate the models on two datasets used in the spurious correlations literature. Each containing a binary spurious attribute and a binary target variable. The composition of spurious attributes and target variables of the datasets is shown in Table 5.3. See section A.3 for visualizations of images and more details in these two datasets.

- Waterbirds [51]. Target: type of bird (waterbird vs. landbird). Spurious attribute: background (water and land). Minority groups: G1 (landbird on water backgrounds) and G2 (waterbirds on land backgrounds). Worst-accuracy group: G2.
- CelebA [35]. Target: hair color (blond vs. non-blond). Spurious attribute: gender: (male and female). Minority and worst-accuracy group: G3 (blond male). A 25% subset of the original CelebA dataset is used to train the models to save computational cost.

**Evaluation metrics.** To evaluate robustness to spurious correlations, we use a standard metric

Table 5.3: Datasets used in spurious correlations experiments. Worst-accuracy group is highlighted in bold.

Waterbirds dataset				CelebA dataset			
Target (class)	Spurious attribute	Group	Presence	Target (class)	Spurious attribute	Group	Presence
landbird (0)	land (0)	G0	73%	non-blond (0)	female (0)	G0	44%
landbird (0)	water (1)	G1	4%	non-blond (0)	male (1)	G1	41%
<b>waterbird (1)</b>	<b>land (0)</b>	<b>G2</b>	<b>1%</b>	blond (1)	female (0)	G2	14%
waterbird (1)	water (1)	G3	22%	<b>blond (1)</b>	<b>male (1)</b>	<b>G3</b>	<b>1%</b>

in the literature spurious correlations: **test set accuracy of the worst-accuracy group** [30, 51]. If a model has learned to rely heavily on a spurious attribute, it will perform poorly on the set of examples containing that attribute belonging to the minority group (eg: waterbirds on land background, males with blond hair). Therefore, higher accuracy on the worst-accuracy group implies higher resilience to spurious correlations.

### 5.3.2 Results and discussion

The results are shown in Table 5.4. On both datasets, the standard-trained model achieves higher worst-group accuracy than both robust objectives. The CE-of-attacks robust objective has a worst-group accuracy value closer to the standard model (0.62% less on Waterbirds dataset, 3.93% less on the CelebA dataset). **These results show that the two attribution robust objectives do not make the models resilient to spurious correlations.**

The model trained with the IG-Norm robust objective misclassifies all test examples in the two minority group of both datasets (0% accuracy). The overall accuracy of the IG-Norm models are high as the minority groups are heavily underrepresented in the test set. We identify this as a failure mode of the IG-Norm objective: it has very low accuracy on minority groups and perfect accuracy on majority groups. This could be an effect of group imbalance, or an effect of spurious correlations. Further research is warranted in order to understand this behavior.

Table 5.4: Accuracy % of different training objectives on spurious correlations datasets.

Training objective	Waterbirds		CelebA	
	Worst-group	Overall	Worst-group	Overall
Standard	<b>39.09</b>	<b>78.25</b>	<b>41.67</b>	94.77
IG-Norm	0.00	78.24	0.00	86.59
CE-of-attacks	38.47	71.06	37.74	<b>95.17</b>

## 5.4 Conclusion

The experiments described in this chapter demonstrate that neural networks trained with attribution robust objectives (IG-Norm and CE-of-attacks) are robust against image corruptions. Results show accuracy gains in robust models in the range of 3.58% to 11.94% over standard trained models evaluated on corrupt image datasets.

The experiments on spurious correlations show that both robust objectives underperform the standard models as measured by worst-group accuracy on datasets with spurious correlations. In addition, a failure mode of the IG-Norm objective was identified, where all minority group test examples are misclassified, and the majority groups are perfectly classified.

### 5.4.1 Potential avenues for exploration

We now discuss several potential paths of exploration on the topic of properties of attribution robust models with pointers to relevant research.

#### Attribution robustness and spurious correlations

The negative result on the relationship between attribution robustness and spurious correlations presented in this chapter should be treated as a preliminary result. The robust objectives used in the experiments (IG-Norm and CE-of-attacks) operate on generic attribution maps, and they do not incorporate any knowledge of the data domain or the spurious correlations present in the datasets. To be resilient to spurious correlations, we may need to incorporate knowledge of the spurious correlations into the training objectives. Two concrete ideas are discussed below;

- Annotate the images (pixel level annotations) so that relevant objects in the images and even the spurious parts to avoid are highlighted. Add a prior to the loss function based on annotations (focus on the correct parts, avoid the spurious parts) [48].
- When it is difficult to obtain a large amount of annotations, add sensible priors to the loss function that may be based on attributions [16]. For example, for the Waterbirds dataset, a prior that says attributions must be focused on regions, rather than dispersed could prove useful. This can be seen as domain specific attribution robustness (as opposed to the generic attribution robustness we have explored in this work).

### Studying a more comprehensive set of properties of attribution robust models

Since attribution robust training is a technique that makes the input feature space more robust than standard cross entropy loss, intuition supports that such robust models may have other benefits. We suggest two properties worth investigating.

- **Transfer learning capability.** Neural networks trained with attribution robust objectives on large source datasets may be capable of achieving higher transfer performance on secondary datasets. A similar effect is reported for the case adversarial or prediction robustness [52, 66]. Similarly, attribution robust models may perform better on diverse downstream tasks [58]. In these settings, attribution robust training can be thought of as a good pre-training task.
- **Training data efficiency.** For many tasks, large amounts of labeled data may not be available. In those cases, achieving high accuracy with less data is important. Prior work indicates that adversarial or prediction robust training can be data efficient, and even treated as a form of data augmentation [65]. Therefore, an investigation of the data efficiency of attribution robust training could be a worthwhile effort.

# Chapter 6

## Conclusion

### 6.1 Summary of thesis contributions

This thesis presented research on the topic of attribution robustness of neural networks (chapters 2 - 5) as well as applications of deep neural networks based on published work (appendices B, D, E). The research contributions of this thesis can be summarized as follows.

#### 6.1.1 Attribution robustness of neural networks

Attributions or feature significance maps are a method for explaining a model’s predictions for a given input. Attribution robustness refers to the property that attributions produced by a model cannot be easily manipulated by adversarial input perturbations. For reliability and trust of attributions and machine learning models in general, attribution robustness is an important research problem to solve.

**Novel framework of loss functions and a new robust objective.** We presented a new, generic and flexible framework of loss functions for training attribution robust neural networks, which allows us to derive well-known and new robust objectives. One such derivation led to the novel attribution robust objective, the “cross entropy of attacks”. It is simpler and significantly faster than the existing robust objectives, while simultaneously achieving higher attribution robustness and accuracy on several image datasets.



**Computational cost analysis of attribution robust training and techniques for efficiency improvement.** A detailed algorithmic and empirical analysis of the high computational cost of attribution robust training revealed that it can be 4 to 67 times slower than standard training in different settings. To address this critical issue, we introduced three efficiency improvement techniques: combining standard and robust training (2x - 4x speed gain), using a fast one-step attack in the training loop (3x or more speed gain), and tuning the number of steps in the integrated gradients computations (2x - 6x speed gain).

**An analysis of the challenges of attribution robust training.** We presented experiment results that confirm the existence of a trade-off between attribution robustness and accuracy. We also identified the critical challenge of hyperparameter tuning in robust training, and provided a guideline to approach the tuning process.

**Properties of attribution robust neural networks.** An empirical investigation of the properties of attribution robust neural networks revealed that they are robust to common image corruptions (higher accuracy than standard models on corrupt data), and that they are not resilient to spurious correlations as hypothesized in prior work.

## 6.1.2 Applications of deep neural networks

Neural network classification systems were developed using several training methodologies (supervised, semi-supervised and self-supervised) to solve real-world problems.

**Computer network intrusion detection.** We presented a comprehensive survey and empirical benchmark of deep learning models for the use case of network intrusion detection. We identified that deep feed-forward neural networks trained in a supervised manner outperform autoencoder, deep belief network and LSTM networks in terms of accuracy, F1-score and training and inference time. The diverse benchmark with legacy and modern intrusion detection datasets allows for standard comparison, which was a critical gap in the existing literature.

**Damage detection in physical structures.** We proposed a novel machine learning system utilizing an LSTM with multi-window input from vibration signals for the purpose of detecting and classifying damage in physical structures. We demonstrated that simple data augmentation techniques and a voting mechanism on the output of individual windows in a signal significantly improve the performance of the system.

**Label efficient classification of industrial process signals with self-supervised learning.** A critical problem in the domain of industrial process signal classification is the lack of labeled data despite the availability of vast amounts of unlabeled data. We used a well-known self-supervised learning method called Contrastive Predictive Coding (CPC) to pre-train a hybrid 1D CNN + LSTM architecture on unlabeled signals, followed by supervised fine-tuning on labeled data. This method produced excellent label efficiency, achieving high accuracy comparable to label-heavy supervised-only training methods with very little labeled training data (10's or few 100's of labeled examples).

## 6.2 Impact and implications

### 6.2.1 Impact of the research on attribution robustness

We approached this work with the goal of dissecting the attribution robust training problem and developing simple techniques to achieve robustness. **The methods proposed in this work are simple, efficient, backed by comprehensive empirical evidence and open source.** This allows researchers to easily build on our work without the need for vast computational resources. Robust attributions lead to reliable and trustworthy explanation methods and dependable machine learning systems in general. In domains where the reliability of explanations is critical, such as healthcare and autonomous systems, the advancements in attribution robustness are especially consequential.

The analysis of the properties of attribution robust models was intended to enhance the understanding of these training mechanisms and the resulting trained models, which are typically treated as black boxes and not broadly evaluated. We hope that both the positive and negative results presented in this work will be a valuable addition to the literature on attribution robustness of neural networks.

### 6.2.2 Impact of the research on applications

The three application projects introduced novel machine learning systems and techniques that the researchers and engineers in corresponding domains can use and build on. The research in network intrusion detection with neural networks will enhance the security of computer

systems and add value to the cybersecurity research community.

The structural damage detection project was a collaboration with a civil engineering research team. The research was conducted with a view of utilizing the learning capabilities of deep neural networks to enable the engineers in the field of structural health monitoring to develop better monitoring systems, and to enhance the safety of physical infrastructure of the world. The label efficient industrial process classification methods presented in this thesis address the critical issue of lack of labeled data in the field, and enable industrial engineers to become more productive. The published work on these topics have been well-received by the research community.

## 6.3 Future work

We now discuss several potential paths of exploration for future work on the topic of attribution robustness of machine learning models.

**Diverse neural architectures, tasks and data modalities.** The literature on attribution robustness is largely concentrated on convolutional neural network classifiers with image data. There is no fundamental reason for this to be the case; attribution attacks can be generated for any model and data type. We encourage more diversity on the topic of attribution robustness.

- Training attribution robust models for other data modalities (eg: time-series such as audio signals).
- Training other neural network architectures for attribution robustness (eg: very deep ResNets, LSTMs, transformers etc.) as well as other machine learning models.
- Investigating attribution robustness in task settings other than classification (eg: regression).

**Broad evaluation of properties of robust models.** Studying a more comprehensive set of properties of attribution robust models among multiple data modalities and neural network architectures (see also section 5.4.1).

- Properties such as transfer learning capability, label efficient classification.

- Performance on edge cases/ examples that are difficult to classify for standard models.

**Efficient robust training.** As seen in this work, attribution robust training of neural networks is computationally more expensive than standard training. We have introduced several techniques for improving efficiency, but more research in this direction will enable attribution robust training at scale (for large models and datasets). For example, it may be possible to replace integrated gradients with a faster attribution method, so that the attack generation during training will be fast.

**Improving our understanding of attribution robust training dynamics.** Existing work evaluate attribution robustness of trained models using metrics such as Kendall tau and accuracy after robust training sessions have been completed. However, we do not have a good idea on how the neural loss surfaces [33] of attribution robust objectives are different from standard modes. Such analysis will lead to a deeper understanding of attribution robust training, and how it poses a different optimization problem than the standard training objectives.

# Bibliography

- [1] Marco Ancona et al. “Gradient-Based Attribution Methods”. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Lecture Notes in Computer Science. 2019, pp. 169–191. doi: 10.1007/978-3-030-28954-6\_9.
- [2] Christopher J. Anders et al. “Finding and Removing Clever Hans: Using Explanation Methods to Debug and Improve Deep Models”. In: *Information Fusion 77* (Jan. 2022), pp. 261–295. doi: 10.1016/j.inffus.2021.07.015.
- [3] Tao Bai et al. “Recent Advances in Adversarial Training for Adversarial Robustness”. In: *Twenty-Ninth International Joint Conference on Artificial Intelligence*. Vol. 5. Aug. 2021, pp. 4312–4321. doi: 10.24963/ijcai.2021/591.
- [4] Aditya Chattopadhyay et al. “Neural Network Attributions: A Causal Perspective”. In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 981–990.
- [5] Jiefeng Chen et al. “Robust Attribution Regularization”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 14300–14310.
- [6] Valerie Chen et al. “Interpretable Machine Learning: Moving from Mythos to Diagnostics”. In: *Communications of the ACM 65.8* (July 2022), pp. 43–50. doi: 10.1145/3546036.
- [7] Gustavo H. de Rosa and João P. Papa. “A Survey on Text Generation Using Generative Adversarial Networks”. In: *Pattern Recognition 119* (Nov. 2021), p. 108098. doi: 10.1016/j.patcog.2021.108098.
- [8] Boty Dimanov et al. “You Shouldn’t Trust Me: Learning Models Which Conceal Unfairness From Multiple Explanation Methods”. In: *24th European Conference on Artificial Intelligence (2020)*, pp. 2473–2480. doi: 10.3233/FAIA200380.

- [9] Ann-Kathrin Dombrowski et al. “Explanations Can Be Manipulated and Geometry Is to Blame”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019, pp. 13589–13600.
- [10] Ann-Kathrin Dombrowski et al. “Towards Robust Explanations for Deep Neural Networks”. In: *Pattern Recognition* 121 (Jan. 2022), p. 108194. doi: 10.1016/j.patcog.2021.108194.
- [11] Finale Doshi-Velez and Been Kim. “Towards A Rigorous Science of Interpretable Machine Learning”. In: *arXiv* (Mar. 2017). doi: 10.48550/arXiv.1702.08608.
- [12] H. Drucker and Y. Le Cun. “Improving Generalization Performance Using Double Back-propagation”. In: *IEEE Transactions on Neural Networks* 3.6 (Nov. 1992), pp. 991–997. doi: 10.1109/72.165600.
- [13] Fabian Eitel and Kerstin Ritter. “Testing the Robustness of Attribution Methods for Convolutional Neural Networks in MRI-Based Alzheimer’s Disease Classification”. In: *Workshop on Interpretability of Machine Intelligence in Medical Image Computing*. 2019, pp. 3–11. doi: 10.1007/978-3-030-33850-3\_1.
- [14] L Engstrom et al. *Robustness (Python Library)*. Madry Lab. Dec. 2023. URL: <https://github.com/MadryLab/robustness> (visited on 12/17/2023).
- [15] Logan Engstrom et al. “Exploring the Landscape of Spatial Robustness”. In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 1802–1811.
- [16] Gabriel Erion et al. “Improving Performance of Deep Learning Models with Axiomatic Attribution Priors and Expected Gradients”. In: *Nature Machine Intelligence* 3.7 (July 2021), pp. 620–631. doi: 10.1038/s42256-021-00343-w.
- [17] Christian Etmann et al. “On the Connection Between Adversarial Robustness and Saliency Map Interpretability”. In: *Proceedings of the 36th International Conference on Machine Learning* (2019), p. 10.
- [18] Sunanda Gamage, Benjamin Klopper, and Jagath Samarabandu. “Experiences with Contrastive Predictive Coding in Industrial Time-Series Classification”. In: *ACM SIGKDD Explorations Newsletter* 24.2 (Dec. 2022), pp. 114–123. doi: 10.1145/3575637.3575654.
- [19] Sunanda Gamage and Jagath Samarabandu. “Deep Learning Methods in Network Intrusion Detection: A Survey and an Objective Comparison”. In: *Journal of Network and Computer Applications* 169 (Nov. 2020), p. 102767. doi: 10.1016/j.jnca.2020.102767.

- [20] Amirata Ghorbani, Abubakar Abid, and James Zou. “Interpretation of Neural Networks Is Fragile”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 3681–3688. doi: 10.1609/aaai.v33i01.33013681.
- [21] Leilani H. Gilpin et al. “Explaining Explanations: An Overview of Interpretability of Machine Learning”. In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. Oct. 2018, pp. 80–89. doi: 10.1109/DSAA.2018.00018.
- [22] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.
- [23] Dan Hendrycks and Thomas Dietterich. “Benchmarking Neural Network Robustness to Common Corruptions and Perturbations”. In: *International Conference on Learning Representations* (2019). doi: 10.48550/arXiv.1903.12261.
- [24] Juyeon Heo, Sunghwan Joo, and Taesup Moon. “Fooling Neural Network Interpretations via Adversarial Model Manipulation”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.
- [25] Adam Ivankay et al. “FAR: A General Framework for Attributional Robustness”. In: *British Machine Vision Conference 2021*. Mar. 2022. doi: 10.48550/arXiv.2010.07393.
- [26] Pavel Izmailov et al. “On Feature Learning in the Presence of Spurious Correlations”. In: *36th Conference on Neural Information Processing Systems*. 2022.
- [27] John Jumper et al. “Highly Accurate Protein Structure Prediction with AlphaFold”. In: *Nature* 596.7873 (Aug. 2021), pp. 583–589. doi: 10.1038/s41586-021-03819-2.
- [28] M. G. Kendall. “A New Measure of Rank Correlation”. In: *Biometrika* 30.1-2 (June 1938), pp. 81–93. doi: 10.1093/biomet/30.1-2.81.
- [29] Pieter-Jan Kindermans et al. “The (Un)Reliability of Saliency Methods”. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Berlin, Heidelberg: Springer-Verlag, Dec. 2022, pp. 267–280.
- [30] Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. “Last Layer Re-Training Is Sufficient for Robustness to Spurious Correlations”. In: *The Eleventh International Conference on Learning Representations*. Sept. 2022.
- [31] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. University of Toronto, 2009. URL: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf> (visited on 12/15/2023).
- [32] Yann LeCun. *The MNIST Database of Handwritten Digits*. URL: <https://yann.lecun.com/exdb/mnist/> (visited on 12/17/2023).

- [33] Hao Li et al. “Visualizing the Loss Landscape of Neural Nets”. In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018, pp. 6391–6401.
- [34] Zachary C. Lipton. “The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability Is Both Important and Slippery.” In: *Queue* 16.3 (June 2018), pp. 31–57. doi: 10.1145/3236386.3241340.
- [35] Ziwei Liu et al. “Deep Learning Face Attributes in the Wild”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015, pp. 3730–3738. doi: 10.1109/ICCV.2015.425.
- [36] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 4765–4774.
- [37] Aleksander Madry et al. “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *International Conference on Learning Representations*. Feb. 2018.
- [38] Aniek F. Markus, Jan A. Kors, and Peter R. Rijnbeek. “The Role of Explainability in Creating Trustworthy Artificial Intelligence for Health Care: A Comprehensive Survey of the Terminology, Design Choices, and Evaluation Strategies”. In: *Journal of Biomedical Informatics* 113 (Jan. 2021), p. 103655. doi: 10.1016/j.jbi.2020.103655.
- [39] Joao Marques-Silva and Alexey Ignatiev. “Delivering Trustworthy AI through Formal XAI”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.11 (June 2022), pp. 12342–12350. doi: 10.1609/aaai.v36i11.21499.
- [40] Shervin Minaee et al. “Image Segmentation Using Deep Learning: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (July 2022), pp. 3523–3542. doi: 10.1109/TPAMI.2021.3059968.
- [41] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. “Methods for Interpreting and Understanding Deep Neural Networks”. In: *Digital Signal Processing* 73 (Feb. 2018), pp. 1–15. doi: 10.1016/j.dsp.2017.10.011.
- [42] Norman Mu and Justin Gilmer. “MNIST-C: A Robustness Benchmark for Computer Vision”. In: *arXiv* (June 2019). doi: 10.48550/arXiv.1906.02337.
- [43] Ali Bou Nassif et al. “Speech Recognition Using Deep Neural Networks: A Systematic Review”. In: *IEEE Access* 7 (2019), pp. 19143–19165. doi: 10.1109/ACCESS.2019.2896880.
- [44] *Regulation (EU) 2016/679 of the European Parliament and of the Council*. of 27 April 2016 (General Data Protection Regulation). May 4, 2016. URL: <https://data.europa.eu/eli/reg/2016/679/oj> (visited on 03/13/2024).



- [45] *Regulation (EU) 2021/0106 of the European Parliament and of the Council*. Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act). 2021. URL: <https://artificialintelligenceact.eu/the-act> (visited on 03/13/2024).
- [46] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco California USA: ACM, Aug. 2016, pp. 1135–1144. doi: 10.1145/2939672.2939778.
- [47] Andrew Slavin Ross and Finale Doshi-Velez. “Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing Their Input Gradients”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. Apr. 2018, pp. 1660–1669.
- [48] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. “Right for the Right Reasons: Training Differentiable Models by Constraining Their Explanations”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI’17. Melbourne, Australia: AAAI Press, Aug. 2017, pp. 2662–2670.
- [49] Waddah Saeed and Christian Omlin. “Explainable AI (XAI): A Systematic Meta-Survey of Current Challenges and Future Opportunities”. In: *Knowledge-Based Systems* 263 (Mar. 2023), p. 110273. doi: 10.1016/j.knosys.2023.110273.
- [50] Shiori Sagawa et al. “An Investigation of Why Overparameterization Exacerbates Spurious Correlations”. In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, Nov. 2020, pp. 8346–8356.
- [51] Shiori Sagawa et al. “Distributionally Robust Neural Networks for Group Shifts: On the Importance of Regularization for Worst-Case Generalization”. In: *8th International Conference on Learning Representations*. Apr. 2020. doi: 10.48550/arXiv.1911.08731.
- [52] Hadi Salman et al. “Do Adversarially Robust ImageNet Models Transfer Better?” In: *34th Conference on Neural Information Processing Systems*. 2020. doi: 10.48550/arXiv.2007.08489.
- [53] Gabriel N. P. dos Santos et al. “Deep Learning Methods for Video Understanding”. In: *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*. WebMedia ’19. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 21–23. doi: 10.1145/3323503.3345029.
- [54] Anindya Sarkar, Anirban Sarkar, and Vineeth N. Balasubramanian. “Enhanced Regularizers for Attributional Robustness”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.3 (May 2021), pp. 2532–2540. doi: 10.1609/aaai.v35i3.16355.

- [55] Patrick Schramowski et al. “Making Deep Neural Networks Right for the Right Scientific Reasons by Interacting with Their Explanations”. In: *Nature Machine Intelligence* 2.8 (Aug. 2020), pp. 476–486. doi: 10.1038/s42256-020-0212-3.
- [56] Karl Schulz et al. “Restricting the Flow: Information Bottlenecks for Attribution”. In: *International Conference on Learning Representations*. 2020, p. 18. doi: 10.48550/arXiv.2001.00396.
- [57] Ali Shafahi et al. “Adversarial Training for Free!” In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 3358–3369.
- [58] Mayank Singh et al. “Attributional Robustness Training Using Input-Gradient Spatial Alignment”. In: *Computer Vision – ECCV 2020*. Lecture Notes in Computer Science. 2020, pp. 515–533. doi: 10.1007/978-3-030-58583-9\_31.
- [59] Sahil Singla and Soheil Feizi. “Salient ImageNet: How to Discover Spurious Features in Deep Learning?” In: *10th International Conference on Learning Representations*. Mar. 2022. doi: 10.48550/arXiv.2110.04301.
- [60] Dylan Slack et al. “Fooling LIME and SHAP: Adversarial Attacks on Post Hoc Explanation Methods”. In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. AIES ’20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 180–186. doi: 10.1145/3375627.3375830.
- [61] Daniel Smilkov et al. “SmoothGrad: Removing Noise by Adding Noise”. In: *arXiv* (June 2017). doi: 10.48550/arXiv.1706.03825.
- [62] Sandeep Sony et al. “Vibration-Based Multiclass Damage Detection and Localization Using Long Short-Term Memory Networks”. In: *Structures* 35 (Jan. 2022), pp. 436–451. doi: 10.1016/j.istruc.2021.10.088.
- [63] C. Spearman. “The Proof and Measurement of Association between Two Things”. In: *The American Journal of Psychology* 15.1 (1904), pp. 72–101. doi: 10.2307/1412159.
- [64] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic Attribution for Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR, July 2017, pp. 3319–3328.
- [65] Dimitris Tsipras et al. “Robustness May Be at Odds with Accuracy”. In: *7th International Conference on Learning Representations*. Sept. 2019. doi: 10.48550/arXiv.1805.12152.
- [66] Francisco Utrera et al. “Adversarially-Trained Deep Nets Transfer Better: Illustration on Image Classification”. In: *34th Conference on Neural Information Processing Systems*. 2020. doi: 10.48550/arXiv.2007.05869.

- [67] Linda Wang, Zhong Qiu Lin, and Alexander Wong. “COVID-Net: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases from Chest X-ray Images”. In: *Scientific Reports* 10.1 (Nov. 2020), p. 19549. doi: 10.1038/s41598-020-76550-z.
- [68] Zifan Wang, Matt Fredrikson, and Anupam Datta. “Robust Models Are More Interpretable Because Attributions Look Normal”. In: *Proceedings of the 39th International Conference on Machine Learning*. PMLR, June 2022, pp. 22625–22651.
- [69] Leander Weber et al. “Beyond Explaining: Opportunities and Challenges of XAI-based Model Improvement”. In: *Information Fusion* 92 (Apr. 2023), pp. 154–176. doi: 10.1016/j.inffus.2022.11.013.
- [70] Matthew Wicker et al. “Robust Explanation Constraints for Neural Networks”. In: *International Conference on Learning Representations*. 2023. doi: 10.48550/arXiv.2212.08507.
- [71] Eric Wong, Leslie Rice, and J. Zico Kolter. “Fast Is Better than Free: Revisiting Adversarial Training”. In: *Eighth International Conference on Learning Representations*. Apr. 2020. doi: 10.48550/arXiv.2001.03994.
- [72] Seul-Ki Yeom et al. “Pruning by Explaining: A Novel Criterion for Deep Neural Network Pruning”. In: *Pattern Recognition* 115 (July 2021), p. 107899. doi: 10.1016/j.patcog.2021.107899.
- [73] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: (June 2017). doi: 10.48550/arXiv.1605.07146.
- [74] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *Computer Vision – ECCV 2014*. Lecture Notes in Computer Science. 2014, pp. 818–833. doi: 10.1007/978-3-319-10590-1\_53.
- [75] Yiming Zhang, Ying Weng, and Jonathan Lund. “Applications of Explainable Artificial Intelligence in Diagnosis and Surgery”. In: *Diagnostics* 12.2 (Jan. 2022), p. 237. doi: 10.3390/diagnostics12020237.
- [76] Yu Zhang et al. “A Survey on Neural Network Interpretability”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 5.5 (Oct. 2021), pp. 726–742. doi: 10.1109/TETCI.2021.3100641.
- [77] Andrea Zunino et al. “Explainable Deep Classification Models for Domain Generalization”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. June 2021, pp. 3227–3236. doi: 10.1109/CVPRW53098.2021.00361.

# Appendix A

## Attribution Robustness Experiment Details

### A.1 IFIA attack generation hyperparameters

Table A.1 shows the hyperparameters used for generating the IFIA attacks for the test sets of different datasets. The symbols denote the following: distance budget =  $\epsilon$ , attack step size =  $\alpha$ , no. of attack iterations =  $iter$ , no. of steps in the IG calculation =  $ig\_steps$ , no. of top attribution pixels =  $k$ .

Table A.1: IFIA attack generation hyperparameters

Dataset	Hyperparameters				
	$\epsilon$	$\alpha$	$iter$	$ig\_steps$	$k$
MNIST	0.3	0.01	100	100	200
CIFAR-10	0.6	0.01	100	50	200
RIMB	0.2	0.005	50	40	1000
Waterbirds	0.2	0.005	50	40	1000
CelebA	0.1	0.0025	50	40	1000

## A.2 Benchmark datasets with common corruptions and perturbations

Table A.2 shows information about the different corruption types contained in the three corrupt image datasets MNIST-C, CIFAR-10-C and RIMB-C.

Table A.2: Benchmark image datasets containing corruptions

Dataset	No. of test examples	Corruption types
MNIST-C [42]	<ul style="list-style-type: none"> <li>- 10,000 examples in each corruption type</li> <li>- 15 corruption types</li> <li>- 150,000 total examples</li> </ul>	<ul style="list-style-type: none"> <li>Random noise: shot noise, impulse noise</li> <li>Blur: glass blur, motion blur</li> <li>Affine transformations: shear, translate, scale, rotate</li> <li>Weather: brightness, fog</li> <li>Artifacts: stripe, spatter, dotted line, zigzag, canny edges</li> </ul>
CIFAR-10-C [23]	<ul style="list-style-type: none"> <li>- 10,000 examples in each corruption type</li> <li>- 19 corruption types</li> <li>- 5 severity levels</li> <li>- 950,000 total examples</li> </ul>	<ul style="list-style-type: none"> <li>Noise: gaussian noise, shot noise, and impulse noise</li> <li>Blur: defocus blur, glass blur, motion blur, and zoom blur</li> <li>Weather: frost, snow, fog, and brightness</li> <li>Digital: contrast, elastic transform, pixelate, JPEG compression</li> <li>Other: speckle noise, spatter, gaussian blur, saturate</li> </ul>
RIMB-C [23]	<ul style="list-style-type: none"> <li>- 1,400 examples in each corruption type</li> <li>- 19 corruption types</li> <li>- 5 severity levels</li> <li>- 133,000 total examples</li> </ul>	Same corruptions types as CIFAR-10-C

## A.3 Datasets used for spurious correlations experiments

Figure A.1 illustrates the spurious attributes and groups in the Waterbirds and CelebA datasets used for spurious correlations experiments in section 5.3.


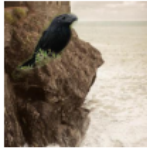
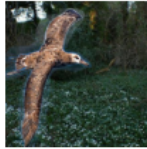





Waterbirds dataset				
	$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{G}_3$	$\mathcal{G}_4$
<b>Image Examples</b>				
<b>Description</b>	landbird on land	landbird on water	waterbird on land	waterbird on water
<b>Class Label</b>	0	0	1	1
<b># Train data</b>	3498 (73%)	184 (4%)	56 (1%)	1057 (22%)
<b># Val data</b>	467	466	133	133
<b>Target:</b> bird type; <b>Spurious feature:</b> background type; <b>Minority:</b> $\mathcal{G}_2, \mathcal{G}_3$				
CelebA hair color dataset				
	$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{G}_3$	$\mathcal{G}_4$
<b>Image Examples</b>				
<b>Description</b>	Non-blond woman	Non-blond man	Blond woman	Blond man
<b>Class Label</b>	0	0	1	1
<b># Train data</b>	71629 (44%)	66874 (41%)	22880 (14%)	1387 (1%)
<b># Val data</b>	8535	8276	2874	182
<b>Target:</b> hair color; <b>Spurious feature:</b> gender; <b>Minority:</b> $\mathcal{G}_4$				

Figure A.1: Waterbirds and CelebA dataset details and visualizations. Adapted from [30]

# Appendix B

## Deep Learning Methods in Network Intrusion Detection: a Survey and an Objective Comparison

### Synopsis

This chapter is based on a survey paper [21] written and published by the thesis author in the Journal of Network and Computer Applications.

The use of deep learning models for the network intrusion detection task has been an active area of research in cybersecurity. Although several excellent surveys cover the growing body of research on this topic, the literature lacks an objective comparison of the different deep learning models within a controlled environment, especially on recent intrusion detection datasets. In this chapter, we first introduce a taxonomy of deep learning models in intrusion detection and summarize the research papers on this topic. Then we train and evaluate four key deep learning models - feed-forward neural network, autoencoder, deep belief network and long short-term memory network - for the intrusion classification task on two legacy datasets (KDD 99, NSL-KDD) and two modern datasets (CIC-IDS2017, CIC-IDS2018). Our results suggest that deep feed-forward neural networks yield desirable evaluation metrics on all four datasets in terms of accuracy, F1-score and training and inference time. The results also indicate that two popular semi-supervised learning models, autoencoders and deep belief networks do not perform better

than supervised feed-forward neural networks. The implementation and the complete set of results have been released for future use by the research community. Finally, we discuss the issues in the research literature that were revealed in the survey and suggest several potential future directions for research in machine learning methods for intrusion detection.



## B.1 Introduction

Computer networks have expanded greatly in size, usage and complexity over the last decade. New types of devices and network architectures have emerged, such as cloud computing and the Internet of Things. With the expansion of these networks and systems, their security has become a key issue. According to the 2019 Cyberthreat Defense Report published by the CyberEdge group [12], attacks on global networks of large enterprises have been increasing during the past five years (Figure B.1). These attacks include, among others, denial of service attacks, malware and ransomware attacks and advanced persistent threats (APT). APTs can be particularly dangerous and costly, as these are targeted, long-term attacks launched by resourceful malicious actors against government and private organizations with the intent of exfiltrating data and sabotaging infrastructure. The cybersecurity report M-Trends 2019 [20] reveals that in 2018, these attacks had a mean dwell time of 184 days (the time that the attack is effective before it is detected).

The first line of defense against cyber attacks on a computer system is formed by a well-designed security framework that enforces standard security practices, which includes confidentiality management, access control, authentication and other security policies. However, attacks may still pose a threat due to operational mishaps, system vulnerabilities (especially, those targeted by zero-day exploits) and other reasons. Intrusion detection systems (IDS) perform the crucial task of detecting such attacks on computers and networks and alerting the system operators. In one case, an IDS in a SCADA system of a water utility helped detect a crypto-mining malware on the system servers by alerting the security team of anomalous HTTP traffic [60].

An IDS may be placed in individual hosts in a network, in a dedicated central location, or distributed across a network. IDS's that are designed to detect attacks on a network of computers rather than a single host are called network intrusion detection systems (NIDS). These systems monitor network functionality in the form of network telemetry, which may include network traffic, metadata of network flows (eg: protocols such as NetFlow) and activity logs from hosts, and attempt to detect attack occurrences.

Techniques used for intrusion detection can be broadly categorized into two groups: signature-based and anomaly detection methods [64]. Signature-based methods operate by matching the input telemetry data with a set of known attack patterns or signatures. These methods give accurate detections on previously known attacks, yet they fail at detecting unseen new attacks.

Anomaly detection methods on the other hand establish a notion of normal behavior in the data, and flag deviations from this behavior (anomalies) as attacks. This method of detection enables the detection of previously unseen attacks. However, since they flag any anomaly as an attack, these methods are known to yield a high number of false alarms.

Machine learning methods for intrusion detection have been studied by researchers for over 20 years [55]. The large volume of network telemetry and other types of security data has made the intrusion detection problem amenable to machine learning methods. Many modern commercial intrusion detection systems use machine learning based algorithms as part of their detection strategy (eg: security platforms Cisco Stealthwatch [11] and Microsoft Azure Sentinel). These methods generally fall under the anomaly detection category of intrusion detection technique. The machine learning models can be broadly categorized into two groups: shallow learning or classic models and deep learning models. Generally, deep learning models in current use are neural network models with a high number of hidden layers. These models are capable of learning highly complex non-linear functions, and the hierarchical arrangement of layers enables them to learn useful feature representations from input data. Deep learning methods have gained recent success in multiple domains such as image classification [44, 27], speech recognition [30, 4] and machine translation [70]. In the domain of intrusion detection, both classic machine learning models [9] and deep learning models [7] have been used with promising results.

Several recent survey papers cover the literature on deep learning methods for intrusion detection [7, 53]. However, the literature is lacking in an empirical evaluation of deep learning models evaluated on standard intrusion datasets; especially, newer datasets. In this chapter, we present a taxonomy and a broad survey of research papers on the topic of deep learning methods for intrusion detection. Four key deep learning models in the literature are trained and evaluated on two legacy datasets (KDD 99, NSL-KDD) and two modern datasets (CIC-IDS2017, CIC-IDS2018), and this benchmark implementation and its complete result set is made publicly available. The remainder of this chapter is organized as follows. The rest of Section B.1 briefly reviews the related literature and highlights the contributions of this chapter. Section B.2 is a preliminary discussion on the problem of intrusion detection, and section B.3 presents the taxonomy and the survey of the literature on deep learning methods for intrusion detection. Section B.4 describes in detail the series of experiments in the benchmark and the analysis of the results. Section B.5 is a discussion on issues in current research and future directions, and section B.6 gives concluding remarks.

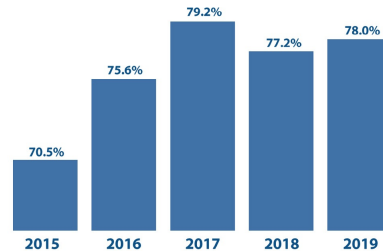


Figure B.1: Percentage of large enterprises with reported attacks [26]

### B.1.1 Related work

Several reviews of the literature on machine learning methods for intrusion detection have been published in recent years, some of them focusing on deep learning methods. One of the earlier surveys [76] review machine learning models in the intrusion detection domain published between 2000 and 2007. Since this is an older survey, it does not capture the research developments in intrusion detection during the past decade. It can be seen that during this period (2000 - 2007), classic machine learning models such as support vector machines, shallow artificial neural networks, and decision trees have been popular in research. The authors categorize the models into three groups: single classifiers, hybrid classifiers (cascade of different classifiers) and ensemble classifiers (combination of weak learners).

Several other surveys [9, 17] cover a wide range of machine learning algorithms in the intrusion detection domain. Use of classic algorithms such as SVMs, decision trees and random forests are reviewed in detail, as well as hidden Markov models and evolutionary algorithms. Both these surveys review research published before 2015 and they do not explore any class of algorithms in depth. In particular, deep learning methods in intrusion detection are not covered in these surveys. Several surveys published in 2018 and 2019 [53, 10, 69] review more recent research on this topic, which include some deep learning methods. However, the selection of deep learning algorithms in these articles is limited, as they are intended to be surveys of a broad set of machine learning models for intrusion detection.

Several survey papers focus on the topic of deep learning methods for intrusion detection. In [32], authors propose a taxonomy which accommodates deep learning methods by classifying them into two groups: generative and discriminative architectures. However, important models such as deep feed forward networks and recurrent neural networks are not comprehensively covered in this review. A detailed summary of a selection of seven research papers on deep learning models for intrusion detection is given in [46]. Other work [80] reviews several types

of deep learning models in cybersecurity: deep belief networks, recurrent neural networks and convolutional networks. Both these surveys lack breadth of coverage of deep learning models used in intrusion detection. A broad survey in [23] covers deep learning models in the domain of IoT security. However, few of the reviewed works are relevant to intrusion detection.

Of the recent survey papers, [7] provides perhaps the most comprehensive review of deep learning models used in cybersecurity tasks, including intrusion detection. The authors note that restricted Boltzmann machines, autoencoders and recurrent neural networks are particularly popular in the intrusion detection domain. They also note the difficulty in comparing the performance of different models, as researchers use different datasets and metrics for model evaluation. None of the survey papers mentioned above perform any benchmark or empirical analysis of the reviewed deep learning models themselves.

In [82], an empirical performance analysis of four deep learning models (multi-layer perceptron (MLP), restricted Boltzmann machine (RBM), Sparse autoencoder (SAE) and MLP with feature embeddings) is conducted on two intrusion datasets (NSL-KDD and UNSW-NB15). However, their experiments do not cover some of the popular models like recurrent neural networks, and no evaluation is done on newer intrusion datasets. Further, the authors report only accuracy, precision and recall of the experiments, and the analysis lacks depth and insight into the performance of the models. In a recent and concurrent work [18] the performance of seven deep learning models on CSE-CIC-IDS2018 and Bot-IoT datasets is analyzed. This benchmark evaluates the models on only two datasets and report three evaluation metrics: per-class detection rate, overall accuracy and training time. We were unable to verify important parameters of the deep network models such as the number of hidden layers. In contrast, the full implementation of our study is publicly available to encourage objective comparisons and to promote transparency. We train and evaluate both shallow and deep versions of neural network models and report a diverse range of evaluation metrics (accuracy, F1-score, false positive and negative rates, training and inference time etc). Furthermore, we conduct experiments to compare supervised vs. semi-supervised models, a thorough neural architecture search to understand the effects of network depth and width on performance, and an analysis of data efficiency of feed-forward neural networks.

Table B.4 lists prior surveys on the topic of deep learning methods in intrusion detection.

## B.1.2 Contributions

In this chapter, we attempt to address the shortcomings of the current surveys on deep learning models for intrusion detection as discussed above. The major contributions of this work are as follows:

- We present a taxonomy of deep learning methods used for intrusion detection and summarize the recent relevant literature under this taxonomy.
- We implement a benchmark of four key deep learning models used in the intrusion detection literature, and evaluate them on two legacy datasets (KDD 99, NSL-KDD) and two modern datasets (CIC-IDS2017, CIC-IDS2018). The deep learning models have been selected from the top three branches of the taxonomy (Figure B.2), and they are representative of three different approaches to building deep learning models: feed-forward neural networks that classify flow instances, LSTM networks that classify sequences of flows, autoencoder and deep belief networks that are trained in a semi-supervised manner with both unlabeled and labeled data. This empirical comparison aims to address the difficulty of comparing the models by results reported in research works due to differences in evaluation metrics, operations on datasets, and limited evaluation on recent datasets.
- The implementation<sup>1</sup> and the complete set of experiment results<sup>2</sup> are released for further use and analysis by the research community.
- We discuss the issues in current research on machine learning for intrusion detection and highlight several future research directions guided by our findings.

We believe that our survey of the literature, and the comprehensive empirical comparison of deep learning models provides a bird's-eye view of the field and insights to both new and experienced researchers in this field. Researchers may also find the open-source implementation of the benchmark useful as a reference point for their future work and experimentation.

---

<sup>1</sup>[https://github.com/sgamage2/dl\\_ids\\_survey](https://github.com/sgamage2/dl_ids_survey)

<sup>2</sup>[https://github.com/sgamage2/dl\\_ids\\_survey/paper\\_results](https://github.com/sgamage2/dl_ids_survey/paper_results)

## B.2 Preliminary discussion

### B.2.1 The network intrusion detection problem

The goal of an intrusion detection system (IDS) is to monitor the activity or behavior of a system, identify when attacks are occurring and generate alarms so that action can be taken to mitigate the consequences. In a computer system network, the input to the IDS (activity to be monitored) takes the form of network telemetry (traffic statistics, information extracted from packet headers and content) and information from hosts (process behavior, system call traces, application logs, file system changes). The output of the IDS could be a label or an attack score for each input or a sequence of inputs. The label can be binary: normal and attack, or multi-valued indicating different types of attacks.

From a machine learning perspective, this problem can be formulated as either an anomaly detection or a classification problem. To train an anomaly detection model, a dataset of normal inputs is sufficient. To train a classifier, a labeled dataset of normal and attack inputs is necessary (unlabeled data maybe helpful, but most classification models need some amount of labeled data for training). After a model is trained, it can be deployed to make detections on new data from the system.

### B.2.2 Datasets

A number of network intrusion detection datasets are available publicly, some of which have been used for training and evaluating the models in the works surveyed. It is not in the scope of this work to provide a comprehensive overview of all datasets (we direct the interested reader to [61]). This section contains a brief summary of the more frequently used datasets below, and the rationale for using them in our experiments for model comparison. Key characteristics of the datasets are summarized in Tables B.1 and B.2.

- KDD 99 [39]: This is one of the earliest public datasets for intrusion detection and the most frequently used in the reviewed literature. It is a flow-based dataset (augmented with additional metadata from hosts) with 41 features and attacks of four categories: denial of service (DoS), probe, remote to local (R2L), and user to root (U2R). Despite its frequent usage, the dataset contains many deficiencies such as duplicate records, simu-

lation artifacts and the fact that it does not reflect modern network traffic and attacks . In our experiments, we train and evaluate machine learning models on the KDD 99 dataset for two reasons: 1) as a first step to verify that the models are working as expected and 2) to be able to compare with previous results reported on this dataset.

- NSL-KDD [73]: The NSL-KDD dataset addresses some of the deficiencies in the KDD 99 set by removing duplicate records and selecting records that are difficult to classify. It is similar in structure to the KDD 99 dataset. Although this dataset is an improved version of the KDD 99 set, it is still not suitable for building intrusion detection models, and we use it in our experiments for the same reasons quoted for the KDD 99 dataset.
- CIC-IDS2017 [63]: This dataset was recorded in a small network environment with simulated normal traffic. Six categories of modern attacks are launched from a separate network. Both the raw packet capture and Netflows with 80 features are made available. A small number of research papers reviewed in this survey use this dataset. We believe that this dataset is representative of modern network traffic, and therefore we use it in our experiments.
- CSE-CIC-IDS2018 [63]: This dataset is similar to CIC-IDS2017 in structure and the contained attacks. However, it is prepared on a larger network representing a corporation as the victim network (420 client machines and 30 servers) and a larger attacker network (50 machines).

Table B.1: Summary of dataset characteristics.

Dataset	No. of instances	Type of data	No. of features	Features	No. of classes
KDD 99	Train: 4,898,431 Test: 311,029	Connection records	41 (3 categorical, 39 numeric)	duration, src_bytes, serror_rate, su_attempted etc.	24 original classes mapped to 5 classes
NSL-KDD	Train: 125,973 Test: 22,544				
IDS 2017	Total: 2,827,808	Bi-directional NetFlows	78 (1 categorical, 77 numeric)	duration, packet counts and sizes in both directions etc.	12 classes
IDS 2018	Total: 16,137,183				15 classes

### B.2.3 Evaluation metrics

There are several metrics and tools used by researchers to evaluate the performance of machine learning models in intrusion detection. These metrics measure different aspects of a model or an IDS system. A brief description of each metric and tool is given below.

Table B.2: Attack types present in the datasets.

Dataset	Attack types
KDD 99 and NSL-KDD	Probe: ipsweep, nmap, portsweep, satan
	DoS (Denial of Service): back, land, neptune, pod, smurf, teardrop
	U2R (User to Root): buffer_overflow, loadmodule, perl, rootkit
	R2L (Remote to Local): ftp_write, guesspasswd, imap, multihop, phf, spy, warezlient, warezmaster
IDS 2017	Bot, DDoS, DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS slowloris, FTP-Patator, PortScan, SSH-Patator, Web Attack Brute Force, Web Attack XSS
IDS 2018	Bot, Brute Force - Web, Brute Force - XSS, DDOS HOIC, DDOS LOIC-UDP, DDoS LOIC-HTTP, DoS GoldenEye, DoS Hulk, DoS SlowHTTPTest, DoS Slowloris, FTP-BruteForce, Infiltration, SQL Injection, SSH - Bruteforce

The confusion matrix is a tabulation of classifications made by a model, typically with the actual class (ground truth) on rows and predicted class on columns. It can be obtained for both binary classifications (Table B.3), and multi-class classifications. It shows the "classification distribution" of a model, and helps identify properties of the model, such as when it is consistently misclassifying one class as another. For example, a model may classify nearly all DDoS attempts as a data exfiltration. The researcher can then examine potential causes for this issue. This type of insight is not readily gleaned from other metrics.

Table B.3: Confusion matrix for a binary class problem: TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives

		Predicted class	
		Attack	Benign
Actual class	Attack	TP	FN
	Benign	FP	TN

Several other important metrics are calculated based on the values in the confusion matrix (Table B.3). They include overall accuracy, precision, recall, specificity, false positive rate and F1 score. See C.1 for definitions of the metrics.

For multi-class problems, these metrics can be calculated per-class, treating the classification as a one-vs-all problem. A final metric may be obtained by averaging the per-class metrics in one of three ways.

- Micro average: calculate the metrics using the sum of each type of classification (eg:  $TP = TP_{class1} + TP_{class2} + \dots$ ).
- Macro average: calculate the per-class metrics and average them (sum and divide by the



no. of classes)

- **Weighted macro average:** similar to macro average, except the per-class metrics are weighted by the number of instances in each class to obtain the final average.

Note that a natural trade-off exists between sensitivity (recall) and false positive rate (FPR); i.e. a highly sensitive model (desirable) is likely to have a high false positive rate (undesirable). This relationship is illustrated by a Receiver Operating Characteristic (ROC) curve, which has FPR in the horizontal axis, and sensitivity (recall) in the vertical axis. The ROC curve is obtained by changing the threshold for classification into the positive class (attacks), and recording the corresponding FPR and recall. One can then choose a point on the ROC curve that is suitable to the application. The area under the ROC curve (AUC) is another metric for classifiers that reflects the ability of the model to distinguish between the two classes.

In addition to the metrics that evaluate classification performance of a model, several other important metrics reflect model complexity. In a deep learning model, complexity is a function of the number of parameters (weights) and architecture.

- **Time to train:** This can be measured as the number of training epochs (iterations over the dataset taken by an algorithm like gradient decent) needed to reach a goal metric on out-of-sample data (eg: a goal accuracy or a goal F1-score). The time taken (in seconds or minutes) for those training epochs can also be measured. If the model is trained offline, a longer time to train it may be acceptable. However, if the model must be trained in real-time, fast training is required.
- **Time to predict (latency):** this can be measured as the time taken to make a prediction on a fixed-sized batch of input instances. Since predictions in an intrusion detection system are made online (real-time), low latency figures are preferred.
- **Throughput:** the number of predictions that can be made in a time period. This is a function of prediction latency and the manner the model is deployed in an IDS (eg: parallel deployments yield high throughput). Given the large volume of network telemetry data that an IDS has to examine, high throughput figures are required for proper operation.

### B.3 Taxonomy and survey of deep learning methods for intrusion detection

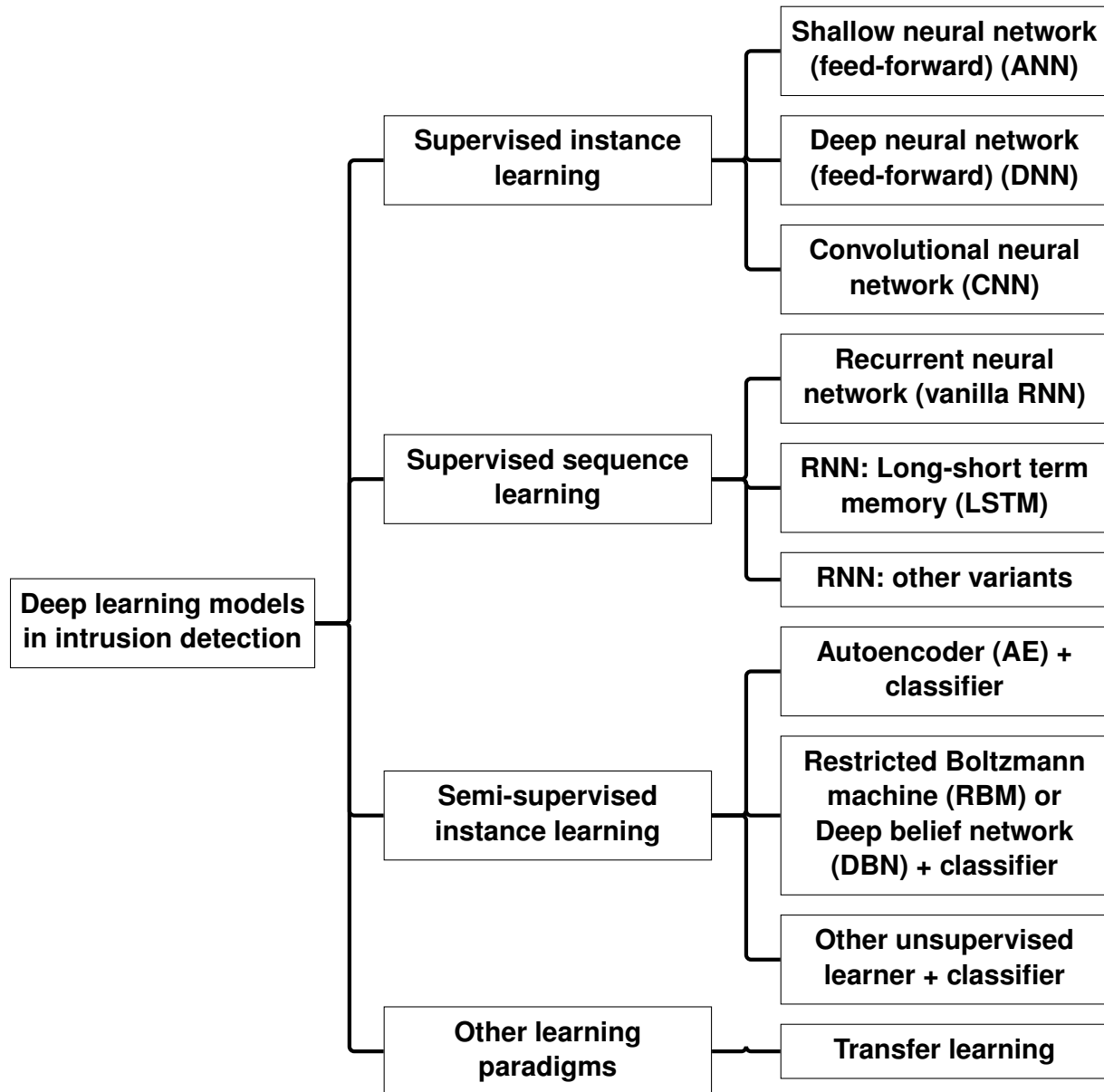


Figure B.2: Taxonomy of deep learning models in intrusion detection

Various taxonomies have been presented in previous surveys on the topic of deep learning models in intrusion detection [32, 46, 10]. The taxonomy we propose for this survey is shown in Figure B.2. For this taxonomy, we have considered the machine learning practitioner's point of view rather than conceptual classification of machine learning models. For example, previous taxonomies identify generative vs. discriminative models, which is an important conceptual

categorization. However, the distinctions we make are more likely to help practitioners in terms of data pre-processing and implementing a model building pipeline in addition to highlighting conceptual differences. For example, instance vs. sequence classification require different arrangements of the dataset, and semi-supervised methods typically involve a pre-training stage (with unlabeled data) in the machine learning pipeline. A transfer learning method requires downloading of a suitable pre-trained model from a repository.

We have surveyed the research literature on deep learning methods in intrusion detection and compiled a summary of the key works. Table B.4 shows a list of current survey papers on this topic and Tables B.5, B.6, B.7 and B.8 shows research papers categorized under each branch of the taxonomy. Finally, four key neural network models from the first three branches of the taxonomy were implemented for empirical comparison (section B.4).

Table B.4: List of survey papers on deep learning and other machine learning methods for intrusion detection. (DNN = deep neural network, RNN = recurrent neural network, CNN = convolutional neural network, RBM = restricted Boltzmann machine, DBN = deep belief network, AE = autoencoder, SOM = self-organizing map, IDS = intrusion detection system).

Survey paper	Coverage of machine learning models and remarks
[18]	Reviews 10 DNN papers, 7 RNN papers, 7 CNN papers, 9 RBM papers, 5 DBN papers and 7 AE papers. Includes an empirical comparison of seven deep learning models on two recent datasets.
[7]	Reviews 3 DNN papers, 11 AE papers, 2 CNN papers, 8 RNN papers and 7 RBM papers. Includes deep learning papers on other cybersecurity applications (eg: malware detection).
[53]	Reviews a selection of 38 papers that use different machine models for the intrusion detection problem. Contains an analysis of network attacks and features used in IDS.
[10]	Presents a taxonomy of machine learning algorithms in network intrusion detection, with a focus on the Internet of Things. A limited selection of prior research is reviewed in detail. Discusses other aspects such as IDS architectures and deployment methods.
[69]	Summarizes five papers that use deep learning for SDN-based network intrusion detection. Discusses challenges in intrusion detection in an SDN environment.
[80]	Reviews 7 DBN papers, 6 RNN papers and 5 CNN papers.
[32]	Gives a taxonomy of machine learning methods in intrusion detection. 15 papers on various neural network models are reviewed in detail (DNN, RBM, RNN, AE, CNN, SOM).
[46]	Gives a taxonomy of machine learning methods in intrusion detection. 7 deep learning IDS papers are reviewed in detail (includes DNN, AE, DBN). Introduces a 4-layer fully connected DNN with results on the NSL-KDD dataset.
[9]	Covers a range of machine learning methods, including decision trees, Bayesian networks and Ensemble methods. Neural network models are not given focus.
[17]	Covers a range of machine learning methods, including SVM, Naïve Bayes and decision trees. Neural network models are not given focus.

Table B.5: List of research papers on supervised instance classification models for intrusion detection. (Acc = accuracy, Prec = precision, Rec = recall, FAR = false alarm rate, F1 = F1-score)

Model type	Paper	Dataset and problem	Model details	Results
DNN	[77]	NSL-KDD, CICIDS 2017, UNSW-NB15, Kyoto, WSN-DS	ANN has 5 hidden layers with 1024, 768, 512, 256, 128 nodes. ReLU activation	In order NSL-KDD, IDS 2017: Acc = 78.5, 95.6%, Prec = 81.0, 96.2%, Rec = 78.5, 95.6%, F1 = 76.5, 95.7%
CNN	[84]	ISCX VPN-nonVPN, ISCX 2012. 5-class classification	Deep CNN: 2 1D convolutional layers, 1 fully connected layers	Acc = 99.85%
DNN	[78]	NSL-KDD, 5-class classification. Original training set is split to 90% training, 10% test sets	ANN has 4 hidden layers with 245 nodes each. ReLU activation	Acc = 86.35%, Prec = 81.86%, Rec = 77.32%, F1 = 73.89%, FAR = 0.1619
CNN	[45]	NSL-KDD, Kyoto, MAWILab, binary classification	Deep CNN: 3 1D convolutional, 2 max-pooling, 1 flatten, 3 fully connected layers	F1 = 79%
DNN	[15]	NSL-KDD, binary and 4-class classification	ANN has 3 hidden layers with 150, 120, 50 nodes	Acc = 98.27%, Rec = 96.5%, FAR = 0.0257
DNN	[37]	KDD 99 (10% subset), binary classification	ANN has 4 hidden layers with 100 nodes each. ReLU activation	Acc = 99.01%, Rec = 99.81%, FAR = 0.0047
CNN	[79]	Custom dataset (USTC-TFC2016), binary, 10-class and 20-class classification	Traffic flows are converted to gray-scale images. CNN has 2 convolutional, 2 max-pooling, 2 fully-connected layers	20-class classifier: Acc = 99.17%, Prec = 99%, Rec = 98%, F1 = 98%
DNN	[72]	NSL-KDD, binary classification	6 features are manually selected. ANN has 3 hidden layers with 12, 6, 3 nodes	Acc = 75.75%, Prec = 83%, Rec = 76%, F1 = 75%
DNN	[52]	Subsets of KDD 99 and NSL-KDD, 5-class classification	Dataset is clustered using spectral clustering. Multiple ANNs are trained (one per cluster). Result is aggregated	Acc = 72.64%, Prec = 57.48%
DNN	[31]	Simulated IoT network data, binary classification	ANN has one hidden layer with 3 nodes. Sigmoid activation, MSE loss function	Acc = 99%
ANN	[33]	NSL-KDD, binary and 5-class classification	Reduce features to 29. ANN has single hidden layer with 29 nodes	Acc = 79.9%

Table B.6: List of research papers on supervised sequence classification models for intrusion detection. (Acc = accuracy, Prec = precision, Rec = recall, FAR = false alarm rate, F1 = F1-score, GRU = Gated Recurrent Unit)

Model type	Paper	Dataset and problem	Model details	Results
LSTM	[14]	ISCX 2012, AWID. Binary classification	30 embedding layers, 10 LSTM layers, and sigmoid output layer	ISCX dataset: Acc = 99.91%, Prec = 99.85%, Rec = 99.96%
GRU	[81]	KDD 99, NSL-KDD, 5-class classification	GRU and Bidirectional GRU (BGRU) nets. Model has one layer with 128 GRU nodes, 3 feed-forward layers with 48 nodes	BGRU gives best results with fast convergence. On NSL-KDD: Acc = 99.24%, Rec = 99.31%, FAR = 0.84%
LSTM	[59]	ISCX IDS, anomaly detection by predicting future tokens (unsupervised)	Token embedding layer, 2 bidirectional LSTM layers, 2 dense layers	AUC = 0.84, ROC curves for attack classes given
LSTM	[36]	NSL-KDD, binary classification	Training multiple LSTM nets (one hidden layer) for different features extracted (channels). Majority voting.	Acc = 98.94%, Rec = 99.23%, FAR = 9.86%
LSTM	[51]	Vehicle network dataset (custom) with 3 attack types	3 LSTM layers with 100, 800, 1000 nodes	Acc = 86.9%
RNN	[83]	NSL-KDD, binary and 5-class classification	RNN has 1 hidden layer with 80 nodes, learning rate = 0.5	5-class classifier: Acc = 81.29%
LSTM	[8]	KDD 99 (converted to time-series), anomaly detection by predicting future packets	One hidden LSTM layer with 23 nodes. Input time steps = 3. High prediction errors in a time window (12 steps) indicate attacks	Rec = 94 - 98%, FAR = Variable
LSTM	[48]	KDD 99, 5-class classification	LSTM network details are unspecified	Acc = 97.54%, Prec = 97.69%, Rec = 98.95%, FAR = 9.98%
LSTM	[74]	CAN bus data (custom), anomaly detection by predicting next word	Network has 2 dense layers, 2 LSTM layers. Input seq. length is 20 words	AUC ranging from 94.4 to 99.8
LSTM	[40]	KDD 99, 5-class classification	One hidden LSTM layer with 80 nodes. Input time steps = 100. MSE loss function	Acc = 96.93%, Rec = 98.88%, FAR = 10.04%
LSTM	[68]	KDD 99, 5-class classification	LSTM network has 20 hidden nodes and variable no. of cell memory blocks	Acc = 93.82%. Comprehensive experiments and results are reported.

Table B.7: List of research papers on semi-supervised instance classification models for intrusion detection. (Acc = accuracy, Prec = precision, Rec = recall, FAR = false alarm rate, F1 = F1-score. AE = Autoencoder, RF= Random Forest, SM = Softmax, DBN = Deep Belief Network)

Model type	Paper	Dataset and problem	Model details	Results
AE + RF	[65]	KDD 99 (10% subset), NSL-KDD, 5-class and 13-class classification	2 stacked AE's (each with 3 hidden layers: 14, 28, 28 nodes): unsupervised training. Followed by RF classifier	5-class classifier on NSL-KDD: Acc = 85.42%, Prec = 100%, Rec = 85.42%, F1 = 87.37%, FAR = 14.58%
AE + SM	[58]	NSL-KDD, binary and 5-class classification	AE with 2 layers (20, 10) is pre-trained layer-wise, followed by fine-tuning a softmax layer	Overall results are not given
AE + SM	[35]	NSL-KDD, 2-class, 5-class and 13-class classification	Sparse AE for unsupervised training followed by softmax regression classifier	2-class classifier: Acc = 88.39%, Prec = 85.44%, Rec = 95.95%, F1 = 90.94%
AE + SM	[1]	NSL-KDD, binary classification	Stacked AE (3 layers with 150, 120, 50 nodes) followed by softmax layer	Acc = 99.2%, Rec = 99.27%, FAR = 0.85%
AE + DBN	[50]	KDD 99 (10% subset), binary classification	AE training round (5 layers with 41, 300, 150, 75 nodes), followed by a RBM-style training of each layer, followed by fine-tuning with a final softmax layer	Acc = 92.1%, Rec = 92.2%, FAR = 1.58%
DBN	[2]	NSL-KDD (40% subset), 5-class classification	DBN model details are unspecified	Acc = 97.45%
DBN	[22]	KDD 99, 5-class classification	DBN has 4 layers with 122, 150, 90, 50 nodes, followed by softmax layer (fine-tuned)	Acc = 93.49%, Rec = 92.33%, FAR = 0.76%
DBN	[29]	Custom IEEE 118-bus dataset, binary classification	Conditional DBN (5 hidden layers with 50 nodes each). Time window = 5	Acc = 96.16%, Rec = 95.89%, FAR = 3.57%
DBN + LR	[3]	KDD 99 (10% subset), 5-class classification	DBN has 4 layers with 72, 52, 40, 5 nodes (pretrained), followed by logistic regression (softmax: fine-tuned)	Acc = 97.9%, Rec = 97.5%, FNR = 2.47%
DBN + DNN	[38]	Custom CAN dataset, 3-class classification	DBN pre-trained weights are used to initialize DNN	Acc = 97.8%, FAR = 1.6%, FNR = 2.8%
DRBM	[19]	KDD 99 (10% subset) and custom dataset, binary classification	Discriminative RBM (parameters are unspecified)	On custom dataset: Acc = 94%

Table B.8: List of research papers on transfer learning models for intrusion detection. (TL = Transfer Learning)

Model type	Paper	Dataset and problem	Model details	Results
TL	[43]	Transfer models between UNSW NB-15 and CICIDS2017 datasets (train on one, transfer to other)	Adversarial Siamese neural network	Outperforms non-TL baseline and the CORAL TL method
TL	[85]	NSL-KDD, create source sets and target sets with attacks not present in the source set	Find a latent space where known and new attacks have minimum distance. Agnostic of the ML model	Outperforms all non-TL baselines (models trained only on target domain) and several other TL methods
TL	[71]	Transfer models from KDD 99 to Kyoto2006	Uses similarity measures and manifold alignment between the two feature spaces	Outperforms non-TL baseline (SVM)

## B.4 Empirical comparison

### B.4.1 Overview of experiments: models, datasets and evaluation metrics

The intrusion detection problem is frequently formulated as a classification problem in the literature reviewed in this chapter. For empirical analysis, four neural network classifiers were selected from the different categories in the model taxonomy (Figure B.2), and they were trained and evaluated on four popular intrusion detection datasets. For each deep learning model type, a shallow model (single hidden layer) and a deep model (multiple hidden layers) were implemented. These models were selected as they are representative of the deep learning models frequently used in the intrusion detection research literature (applicability), and they represent supervised, semi-supervised and sequential types of models (diversity and coverage). The following models were selected for the empirical comparison:

1. *Artificial neural network: feed-forward (ANN)*. Feed-forward neural networks yield state-of-the-art performance on classification tasks in many domains, and it is one of the key models in the deep learning literature [49]. For network intrusion detection, both shallow and deep feed-forward neural networks (DNNs) have been used since the KDD 99 competition [33, 77]. Therefore, feed-forward neural networks were selected as the main



model for experimentation on the intrusion detection datasets.

2. *Autoencoder followed by ANN (AE + ANN)*. In this semi-supervised learning mechanism, first an autoencoder with a symmetric encoder-decoder architecture is trained with unlabeled data (a portion of the training set). This unsupervised training phase is expected to capture feature transformations from the data. Next, the labeled data of the training set are transformed using the autoencoder, and the output of the final encoding layer (feature transformation) is input to a supervised classification model. In our experiments, a feed-forward neural network (ANN) after the final encoding layer of the autoencoder acts as the classification model. Semi-supervised approaches using autencoders for network intrusion detection have been explored in [65, 35].
3. *ANN initialized with deep belief network (DBN + ANN)*. In this semi-supervised method, first a deep belief network (DBN) is trained with unlabeled data (a portion of the training set). Next, the weights learned by the DBN are used as the initial weights for training a neural network (DNN) of the same architecture. The ANN is then trained with the available labeled data in a supervised fashion. This is a different method of semi-supervised learning to the autoencoder; it employs weight transfer instead of data transformation [16]. Semi-supervised approaches using DBNs for network intrusion detection have been explored in [2, 3].
4. *Long-short term memory network (LSTM)*: Network traffic is sequential in nature, which makes recurrent neural networks such as LSTMs good candidates for the network intrusion detection problem [8, 14]. In our experiments, input sequences to the LSTM model have been formed by grouping adjacent flows or records of the datasets, as they occur in a temporal sequence. The output of the model is a label sequence; one label for each flow in the input sequence. The expectation is that the LSTM network will capture temporal relationships among adjacent flows.
5. *Random forest (RF)*: In addition to the 4 neural network models, a random forest was also trained and evaluated on the datasets. It represents the classical machine learning category of models, and it is commonly used in the intrusion detection literature. It provides a benchmark for comparison with the deep learning models.

We used two legacy datasets (KDD 99, NSL-KDD) and two modern datasets (CIC-IDS2017, CIC-IDS2018) for training and testing the machine learning models. A description of the datasets used in the empirical evaluation is given in section B.2.2.

The following metrics are calculated for each model evaluation. The definition of the metrics are given in section B.2.3.

- Per-class metrics: accuracy, precision, recall, false alarm rate, false negative rate, F1 score. These per-class metrics are available in the results repository <sup>3</sup>.
- Overall accuracy.
- Weighted macro averages of precision, false alarm rate, false negative rate, F1 score.
- Training time: the number of training epochs and the time taken to reach the average accuracy achieved by a model (epochs to convergence).
- Testing or inference time: time taken to classify a fixed size batch of 1 million instances from the dataset.

## B.4.2 Details of the experiments

In conducting the experiments of this empirical analysis, we have placed a high importance on the reproducibility of results. To this end, we have closely followed the Reproducibility Checklist of the NeurIPS 2019 conference [57]. The relevant details of the experiments are outlined in the following subsections.

### Data pre-processing

The following pre-processing steps were performed on the datasets.

- Removing invalid flow records: the IDS 2017 and IDS 2018 datasets contain flow records with invalid values in the fields (eg: missing values, strings in numeric fields). Such records were removed from both datasets (2,867 in the IDS 2017 dataset, and 95,819 in the IDS 2018 dataset).
- Changing class labels to formulate the 5-class problem: the KDD 99 and NSL-KDD datasets contain 38 types of attacks (class labels), which fall into 4 attack classes (DOS,

---

<sup>3</sup>[https://github.com/sgamage2/dl\\_ids\\_survey/paper\\_results](https://github.com/sgamage2/dl_ids_survey/paper_results)

R2L, U2R and probing). Since we are developing models for the 5-class version of the problem, the 38 class labels are mapped to the 4 attack classes. With the "Normal" (benign) label, this yields 5 class labels for classification.

- One-hot encoding of categorical features: the KDD 99 and NSL-KDD datasets contain 3 non-numeric (categorical) features: *protocol\_type*, *service* and *flag*. These 3 features were one-hot encoded, which yielded 84 encoded features.
- Scaling features: All features of all 4 datasets were scaled by standard normalization.
- Preparing sequences for the LSTM model: inputs to recurrent neural networks are sequences of instances. For the LSTM model, sequences were formed by grouping a number of consecutive flow records (adjacent in time). The length of a sequence then becomes a hyper-parameter of the model. The class label of each flow record was attached to the individual records in a sequence (i.e. input and output is synchronized). All sequences have a fixed length.

### Sample allocation and hyper-parameter tuning

The 3-way holdout method was used for allocating the data for training, hyper-parameter tuning and model evaluation.

- In the KDD 99 and NSL-KDD datasets, separate test sets are available (in the NSL-KDD dataset, the test is named KDDTest<sup>+</sup>). The rest of the dataset was split into two sets: 80% for training and 20% for validation. The split was made in a stratified fashion in order to preserve the original class proportions.
- In the IDS 2017 and IDS 2018 datasets, a separate test set is not available. The dataset was split into three sets: 60% for training, 20% for validation and 20% for testing. The split was made in a stratified fashion in order to preserve the original class proportions.

*Hyper-parameter tuning.* The final sets of hyper-parameters for each model on each dataset were found by a random search of the parameter space. We used hyper-parameter values reported in the intrusion detection literature as a guide to define the value ranges for the random search. Table B.9 shows the range of values used in the random search of hyper-parameters.

The configurations with the highest weighted macro-average F1 score on the validation set were selected as the final hyper-parameter configurations (Table B.10).

In order to minimize overfitting of neural network models, dropout, batch normalization and early stopping of training was performed. In each model-dataset combination, the hyper-parameter configurations that gave the highest weighted macro-average F1 score were selected as the winning models.

Table B.9: Hyper-parameter configurations/ ranges used in the tuning process

Model	Range of hyper-parameters
ANN	no. of layers = 1 - 5 no. of nodes in layers = 64, 128, 256, 512, 1024, 2048 layer activations = ReLU output layer activation = softmax loss function = cross-entropy weight initialization = He [28] layer dropout rates = 0.05, 0.20, 0.50 batch normalization for each layer batch size = 32, 64, 256, 1024 optimizer = Adam [41]
AE + ANN	Similar configurations to ANN, except: unsupervised training data percentage (for AE) = 25%, 50%, 80% no. of nodes in AE encoder layers = 64, 128, 256, 512, 1024, 2048 symmetric encoder and decoder output layer activation of AE = sigmoid, ReLU loss function of AE = binary cross-entropy, mean squared error
DBN + ANN	Similar configurations to ANN, except: unsupervised training data percentage (for DBN) = 25%, 50%, 80% DBN optimizer = Persistent Contrastive Divergence (PCD) [75] DBN learning rate = 0.0001, 0.0005, 0.001, 0.1 DBN pre-train epochs (each layer) = 20, 50
LSTM	Similar configurations to ANN, except: recurrence relation to self in all hidden nodes cell type = LSTM sequence length (time steps) = 16, 32, 50
RF	no. of trees = 100, 500, 1000 minimum no. of samples required to split a node = 2, 10, 100 (this parameter controls tree depth)

### Experiments with the best hyper-parameter configurations

The best-performing (highest weighted macro-average F1 score on validation set) hyper-parameter configurations selected from the hyper-parameter tuning process are listed in Table B.10. The

following experiments were conducted with these hyper-parameter configurations of the 8 neural networks (shallow and deep ANN, AE + ANN, DBN + ANN, LSTM) and the random forest model.

- *Validating the benchmark implementation and comparing model performance.* The models were trained (with the best hyper-parameter configurations) on the training sets and evaluated on the test sets of the four datasets (KDD 99, NSL-KDD, IDS 2017, IDS 2018). This training and testing process was repeated 5 times for each model-dataset combination to observe any variance of performance caused by random initialization of model weights and to verify the stability of the selected hyper-parameter configurations. Performance metrics were recorded for the 36 model-dataset combinations shown in B.10 and the weighted macro average of metrics are reported (mean and standard deviation over 5 runs). As a form of validating our benchmark implementation, the metrics were compared to the results reported in the literature. A general comparison of the performance of different models along different metrics was also conducted.
- *Measuring model training and testing (inference) time.* For each model-dataset combination, the average test set accuracy from the previous experiment was set as the out-of-sample goal metric. The models were trained until they achieved the goal accuracy, and the number of training epochs and the time taken were recorded. This process was repeated 3 times for each model-dataset combination to obtain a distribution of results (mean result is reported). In the case of semi-supervised models, the autoencoders and deep belief networks were pre-trained for a fixed number of epochs (10), before fine-tuning until convergence to the goal accuracy on the test set. Since the random forest is not trained by an iterative algorithm, its training time was simply measured in minutes. The testing (inference) time was measured as the time taken (in minutes) to classify a fixed size batch of 1 million instances from the dataset (repeated 5 times for a distribution; mean and standard deviation are reported).
- *Comparing semi-supervised methods (AE + ANN and DBN + ANN models) with the supervised feed-forward neural network (ANN).* The training set of the NSL-KDD and IDS 2017 datasets were split 75%-25% in a stratified fashion. The first 75% (labels removed) was used during the unsupervised training phase (autoencoder and deep belief network pre-training), and the second 25% was used in the supervised training phase (fine-tuning). These models were then compared with an ANN model that was trained (supervised) *only* on the second 25% of the split. Thus, the semi-supervised models have a 75% data advantage in the form of unsupervised pre-training over the ANN model. A

second experiment followed the same process with a 50% unlabeled, 50% labeled split of the datasets. A shallow network architecture with 64 nodes in the hidden layer was used for training all the models.

- *Neural network architecture search.* The initial result set showed that feed-forward neural networks (ANN) yielded better evaluation metrics on all four datasets. Therefore, a set of experiments was designed to identify how different ANN architectures impact performance in the intrusion detection task. Three types of architectures were tested. 1) Shallow but wide architectures (3 hidden layers with up to 3000 nodes in the first hidden layer). 2) Deep architectures with the same number of nodes in each layer (up to 10 hidden layers with 64 nodes in each layer). 3) Narrowing deep architectures (up to 10 hidden layers with 800 nodes in the first hidden layer narrowing down to 16 nodes in the last hidden layer). The exact network architectures and the number of parameters in the models are given in Tables C.6, C.7 and C.8. These models were trained and evaluated on three datasets: NSL-KDD and IDS 2017 and IDS 2018.
- *Measuring the effect of training set size on ANN performance.* Network intrusion detection is a domain where large quantities of data can be acquired, both by simulation and recording packet flows in real networks. In order to understand the ability of an ANN to improve performance as more data is made available to it for training, the following experiment was conducted. The training sets of NSL-KDD, IDS 2017 and IDS 2018 datasets were split in a stratified fashion (preserving class proportions) into 10%, 20%, ... 100% subsets. ANN models (shallow: 64 nodes in hidden layer) were trained on these subsets (10% to 100%) of the original training set and evaluated on the test sets.

## Compute environment

The experiments were carried out on several PCs with and without using GPU acceleration. The time measurements (training and testing time) were obtained on a machine with the following specifications.

- Processors: AMD Ryzen 9 3900X CPU @ 3.80GHz, 12-cores (24 threads)
- Memory (RAM): 64 GB
- GPU: 2 x Nvidia GeForce RTX 2080 Ti (11 GB) GPUs

- Operating system: Ubuntu 18.04

Table B.10: Hyper-parameter configurations selected from the tuning process (configurations that give the highest validation weighted macro-average F1 score). mse = mean squared error, bce = binary cross-entropy

Model hyper-parameters	KDD 99		NSL-KDD		IDS 2017		IDS 2018	
	Shal- low	Deep	Shal- low	Deep	Shal- low	Deep	Shal- low	Deep
<b>ANN</b>								
no. of layers	1	3	1	3	1	4	1	4
no. of nodes in layers	256	64,32,16	256	64,32,16	64	256,128, 64,32	512	128,64, 32,16
layer dropout rates	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
batch size	256	256	256	256	256	256	256	256
<b>AE + ANN</b>								
no. of layers	1	3	1	3	1	3	1	3
no. of nodes in encoder layers	32	128,64,32	32	128,64,32	32	128,64,32	32	128,64,32
layer dropout rates	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
batch size	256	256	256	256	256	256	256	256
unsupervised training data percentage	50%	25%	50%	25%	50%	25%	50%	25%
output layer of AE	relu	sigmoid	sig- moid	sigmoid	sig- moid	sigmoid	sig- moid	sigmoid
loss function of AE	mse	bce	bce	bce	bce	bce	bce	bce
AE pre-train epochs	200	200	200	200	200	200	40	40
<b>DBN + ANN</b>								
no. of layers	1	3	1	3	1	3	1	3
no. of nodes in layers	32	128,64,32	32	128,64,32	32	128,64,32	32	128,64,32
layer dropout rates	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
batch size	256	256	256	256	256	256	256	256
unsupervised training data percentage	50%	25%	50%	25%	50%	50%	50%	50%
DBN learning rate	0.0001	0.001	0.0001	0.001	0.0001	0.0001	0.0001	0.0001
DBN pre-train epochs	50	50	50	50	50	50	12	12
<b>LSTM</b>								
no. of layers	1	2	1	2	1	2	1	2
no. of nodes in layers	32	32,16	32	64,32	32	64,32	32	64,32
layer dropout rates	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
batch size	256	256	256	256	256	256	256	256
sequence length	32	32	32	16	32	32	32	32
<b>Random forest</b>								
no. of trees	100		100		100		100	
minimum no. of samples required to split a node	2		2		2		2	

### B.4.3 Results and analysis

#### Validating the benchmark implementation

The evaluation results of the 36 model-dataset combinations (Table B.10) closely approximate the best results reported in the recent literature. The full set of results is given in Tables C.1, C.2, C.3 and C.4 and a comparison of results with prior works is given in Table B.11. The deep feed-forward neural networks proposed in [77] give similar results to our work ( $\pm 3\%$  difference in some metrics) on KDD 99, NSL-KDD and IDS 2017 datasets. The deep neural network suit (ANN, AE, DBN) in [18] on the IDS 2018 dataset also yield similar results ( $\pm 2.5\%$  difference). In other model-dataset combinations, evaluation metrics from our experiments are in the same range as the values reported in the literature. This comparison validates the correctness of our benchmark implementation and verifies that the machine learning models are well-trained on the datasets.

It must be noted that direct comparison is often difficult as many papers do not provide adequate details on the pre-processing steps performed on datasets (eg: removing new attack types in the test set that are not present in the training set, which gives more optimistic results), as well as the exact type of evaluation metric (eg: micro-averaged vs. weighted-macro-averaged precision, F1-score etc). Since the pre-processing steps and evaluation metrics in our study are uniform across all model evaluations in our study, the compilation of results allows for fair comparison of the selected deep learning models at the intrusion detection task. Models trained on IDS 2017 and IDS 2018 datasets give accuracy values in the range of 98.5% - 99.8%, while those trained on NSL-KDD give accuracy figures in the 75.5% - 77.5% range. This large difference is due to the fact that the test set of NSL-KDD is created with samples that were consistently misclassified by a set of 21 machine learning models (samples with high difficulty level [73]).

#### Comparing model performance

The accuracy, precision and F1-score of the models on each dataset are illustrated in Figure B.3, and the false positive and false negative rates are illustrated in Figure B.4. The result set indicates that all the trained models are generally capable of the intrusion detection classification task on these datasets. On each dataset, there is minimal variation of evaluation metrics between the models. In order to check the statistical significance of the variations in the met-



rics, a Friedman's test followed by a Nemenyi test was conducted on the accuracy figures of all models on the four datasets. These are two recommended statistical tests for comparing the performance multiple classifiers on multiple datasets [13].

Friedman's test yielded a p-value of 0.0120, which results in a rejection of the null hypothesis that all models have equal accuracy (with a significance threshold of 0.05). Therefore, a post-hoc pairwise multiple comparison Nemenyi test was done, which has a null hypothesis that performance of each pair of models is equal (p-values are given in Table C.5). The null hypothesis is rejected (p-value below 0.05 significance threshold) in only one case, when the deep ANN model is compared with the deep AE model. This implies that the deep ANN model performs better than the deep AE model in a statistically significant sense.

In all other model pair comparisons, the null hypothesis cannot be rejected, which suggests that their performance differences cannot be asserted with statistical significance. Similar results were observed when the statistical tests were performed for false negative rate and F1-score. It is worth noting that on any of the datasets, the semi-supervised models do not outperform other models along any of the metrics (a separate experiment was carried out to compare ANNs with semi-supervised models and its results are discussed later).

The random forest (RF) gives high accuracy and F1-score on IDS 2017 and IDS 2018 datasets. RFs have been a popular classical machine learning model in the intrusion detection literature. The results from our study suggest that RFs perform well on recent intrusion detection datasets, and it is a suitable model to consider along with other machine learning models in future evaluations.

### **Performance of LSTM models**

The purpose of evaluating an LSTM network for the intrusion detection task was to study the ability of sequence models to capture temporal patterns in the flows that may improve the classification. The experiment results (Figures B.3, B.4 and significance test results in Table C.5) indicate that the LSTM models trained on these datasets do not outperform the other models. They yield high false negative rates for attack classes that are time-correlated by their nature, such as denial of service and brute force style attacks.

Increasing the sequence length (number of look-back time steps) of an LSTM did not improve the evaluation metrics. A reason for this could be that a single flow, which is a window of

related packets, captures the significant packet-level time dependencies in flow features such as arrival rate, no. of packets or bytes in a flow etc. Inter-flow relationships may not contain useful dependencies learn-able by a sequence model like an LSTM network. The capability of sequence models at the intrusion detection task needs to be further investigated.

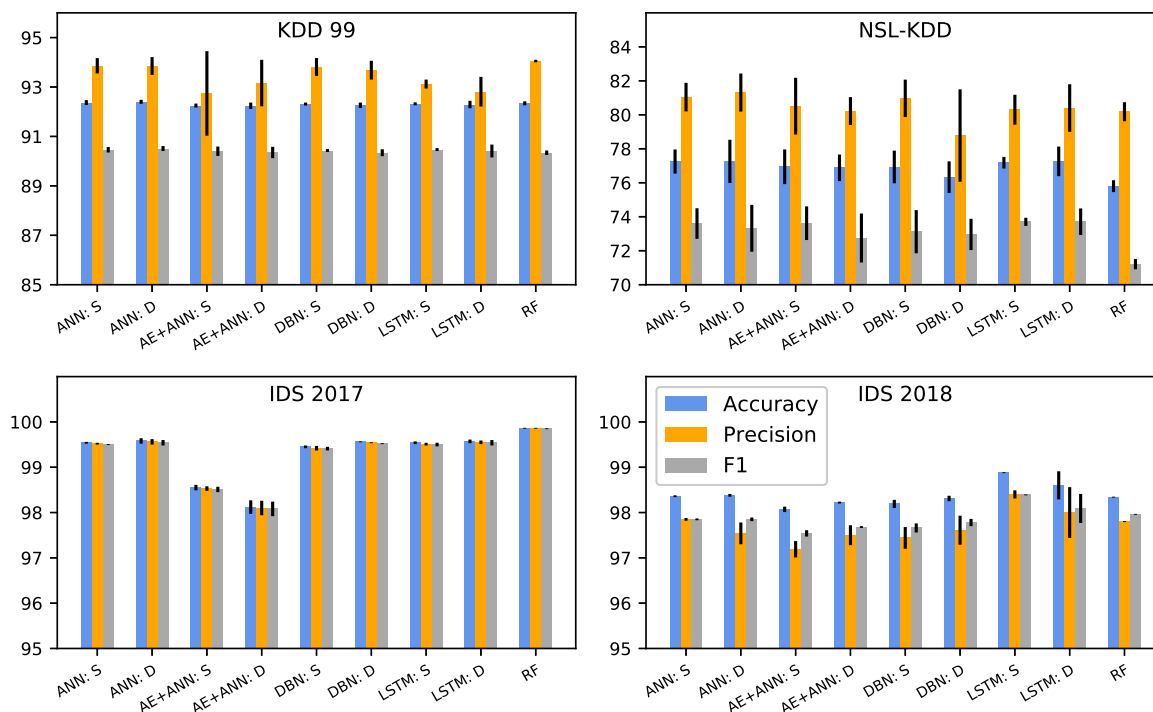


Figure B.3: Accuracy, precision and F1-score of the models on each dataset. ‘S’ and ‘D’ suffixes stand for ‘Shallow’ and ‘Deep’ respectively. The error bars indicate +/- one standard deviation of the error distribution obtained from 5 repetitions of training and evaluation.

### Computational cost of training and inference

The number of training epochs and time in minutes taken to achieve the average goal accuracy of a model, and test (inference) time to classify a batch of 1 million instances are given in the last three columns of Tables C.1, C.2, C.3 and C.4, and illustrated in Figure B.5 for the NSL-KDD and IDS 2017 datasets.

Feed-forward neural networks (ANN) have the fastest training times across the four datasets. The semi-supervised models (AE + ANN and DBN + ANN) are generally slow to train (1.5x to 10x slower than ANNs) due to the computationally expensive unsupervised pre-training phase (10 epochs on the IDS 2018 dataset and 50 epochs on others). In comparison, the ANNs

Table B.11: Comparison of model performance with prior works. (Acc = accuracy, Prec = precision, Rec = recall, FAR = false alarm rate, F1 = F1-score)

Model	NSL-KDD	KDD 99	IDS 2017	IDS2018
ANN	Vinayakumar at al. Acc = 78.5%, Prec = 81.0%, F1 = 76.5%	Vinayakumar at al. Acc = 92.5%, Prec = 93.4%, F1 = 92.1%	Vinayakumar at al. Acc = 95.6%, Prec = 96.2%, F1 = 95.7%	Ferrag at al. Acc = 97.28%
	Ours: Acc = 77.26%, Prec = 81.31%, F1 = 73.32%	Ours: Acc = 92.40%, Prec = 93.85%, F1 = 90.51%	Ours: Acc = 99.58%, Prec = 99.56%, F1 = 99.54%	Ours: Acc = 98.38%
AE	Javaid et al. Acc = 75%, Prec = 86%, F1 = 72%	Shone et al. Acc = 97.85%, Prec = 99.99%, F1 = 98.15%	Ours: Acc = 98.55%, Prec = 98.53%, F1 = 98.51%	Ferrag et al. Acc = 97.37%
	Ours: Acc = 76.94%, Prec = 80.51%, F1 = 73.62%	Ours: Acc = 92.25%, Prec = 92.74%, F1 = 90.40%		Ours: Acc = 98.22%
DBN	Alom et al. (40% subset of NSL-KDD) Acc = 97.45%	Gao et al. Acc = 93.49%, FAR = 0.76%	Ours: Acc = 99.56%, Prec = 99.54%, F1 = 99.52%	Ferrag et al. Acc = 97.3%
	Ours: Acc = 76.93%, Prec = 80.97%, F1 = 73.12%	Ours: Acc = 92.31%, Prec = 93.81%, FAR = 1.99%		Ours: Acc = 98.31%
LSTM	Yin et al. Acc = 81.29%	Le at al. Acc = 97.54%, Prec = 97.69%, FAR = 9.98%	Ours: Acc = 99.57%, Prec = 99.55%, F1 = 99.54%	Ferrag et al. Acc = 96%
	Ours: Acc = 77.26%, Prec = 80.40%, F1 = 73.71%	Ours: Acc = 92.32%, Prec = 93.12%, FAR = 2.17%		Ours: Acc = 98.88%
RF	Vinayakumar at al. Acc = 75.3%, Prec = 81.4%, F1 = 71.5%	Vinayakumar at al. Acc = 92.5%, Prec = 94.4%, F1 = 91.8%	Vinayakumar at al. Acc = 94.4%, Prec = 97.0%, F1 = 95.3%	Ferrag et al. Acc = 92%
	Ours: Acc = 75.80%, Prec = 80.18%, F1 = 71.21%	Ours: Acc = 92.34%, Prec = 94.05%, F1 = 90.34%	Ours: Acc = 99.86%, Prec = 99.86%, F1 = 99.85%	Ours: Acc = 98.34%

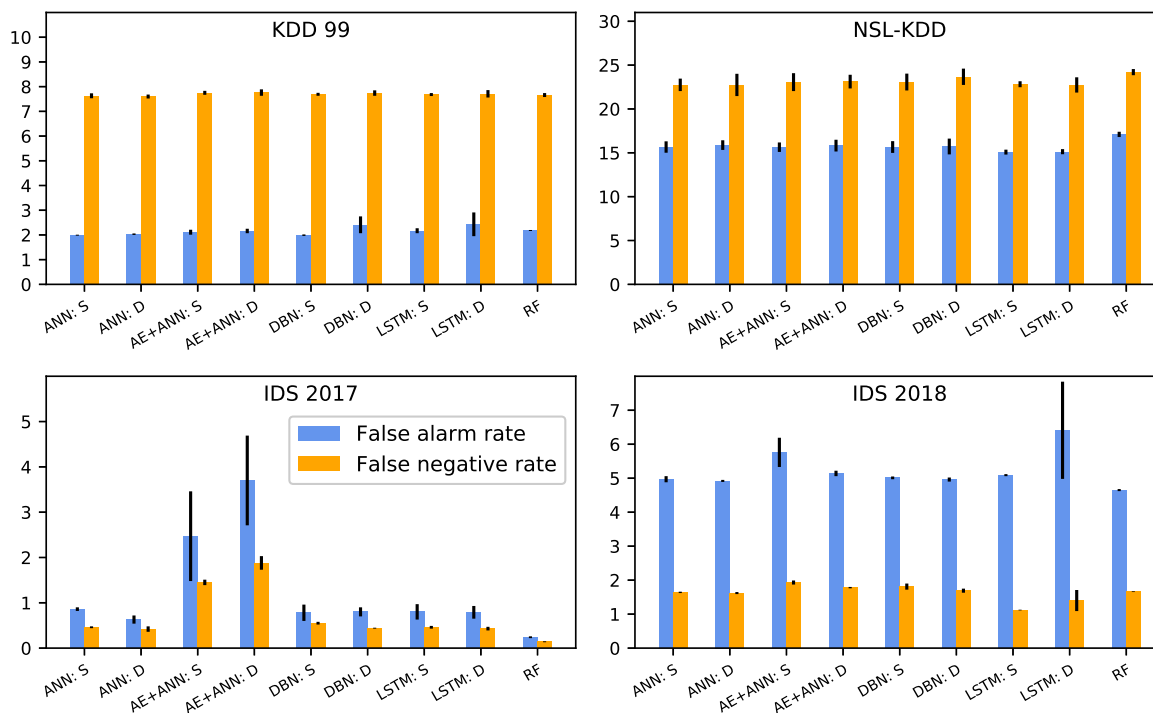


Figure B.4: False positive and false negative rates of the models on each dataset. 'S' and 'D' suffixes stand for 'Shallow' and 'Deep' respectively. The error bars indicate +/- one standard deviation of the error distribution obtained from 5 repetitions of training and evaluation.

converge to their average accuracy with a small number of training epochs compared to the other neural network models.

For inference (classifying instances), ANNs are fast across all datasets, and AE + ANN, LSTM and RF models are the slowest. This is likely due to the fact that features must be transformed by the encoder in the the AE + ANN model, and model complexity of unrolled LSTM networks is high. Random forests have slow inference time (5x to 10x slower than ANNs), which make them undesirable for deployment in a real-time intrusion detection system.

This general comparison of the five models indicate that feed-forward neural networks (ANNs: shallow and deep variants) yield desirable evaluation metrics (accuracy, F1-score, training and inference time etc.) across all four intrusion detection datasets. Random forests show comparable performance and since they are easy to implement and fast to train, they are a suitable model to consider in the intrusion detection model building process (for initial results on new datasets and as a potential baseline for more complex models). LSTM networks and semi-supervised models (AE + ANN, DBN + ANN) do not show improvement over the ANNs.

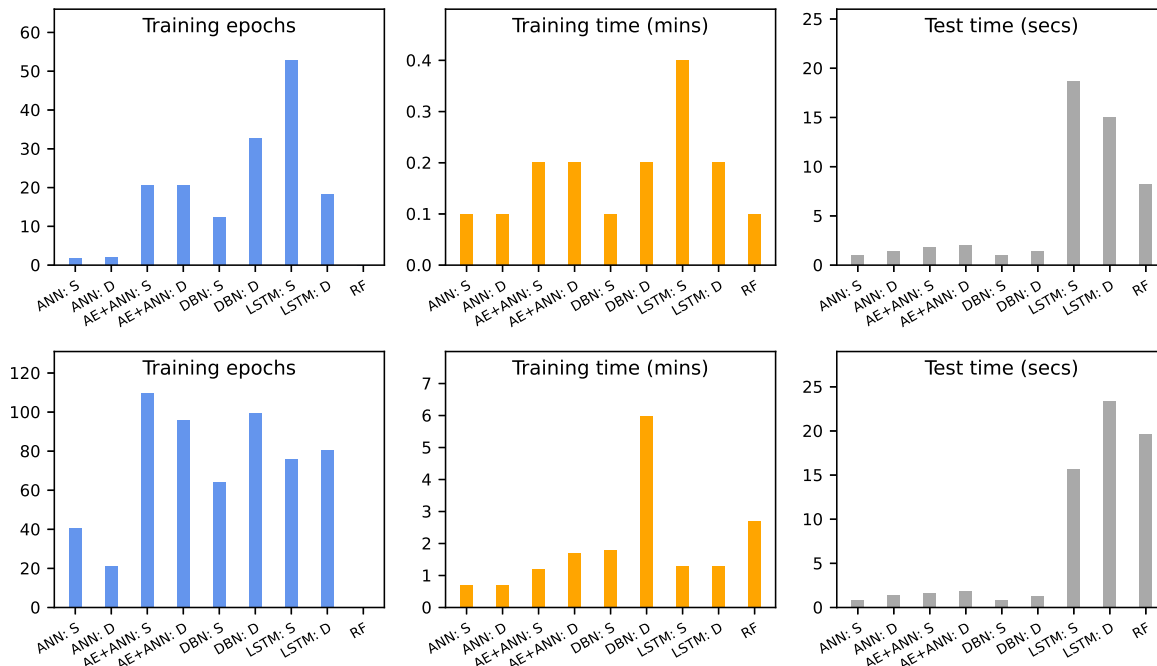


Figure B.5: Training epochs (left) and training time in minutes (middle) to achieve the average goal accuracy of a model, and test (inference) time to classify a batch of 1 million instances (right). Top row: NSL-KDD dataset, bottom row: IDS 2017 dataset

### Comparing semi-supervised methods with the supervised feed-forward neural network (ANN)

To verify the finding from the initial evaluations that semi-supervised models (AE + ANN and DBN + ANN) do not show improvement over the ANNs, an additional set of experiments was carried out where the semi-supervised models were pre-trained with a data advantage (unlabeled) of 75% the size of the original dataset (see section B.4.2). The results of the experiments are given in Table B.12 and illustrated in Figure B.6.

In all cases, ANNs achieve higher or similar performance to the semi-supervised models. 10-fold cross validation  $t$ -tests were performed to verify any statistical significance in differences of accuracy (null hypothesis: two compared models have equal accuracy) and the results are given in Table B.13. The results show that either the ANN model performs better (null hypothesis is rejected in favor of ANNs) or the accuracy differences are not significant. Another consideration is that the semi-supervised models are slow to train due to the expensive pre-training on a large chunk of the dataset (1.5x to 3.5x slower than ANN).

Figure B.7 shows the validation error during the fine-tuning stage of the three models (after the pre-training stage). The DBN + ANN model error at the start of the fine-tuning process is very high in comparison to the other two methods, and it has a slower rate of convergence (in the IDS 2017 case). This indicates that taking DBN-pre-trained weights as the initial point for the fine-tuning stage does not help it achieve better performance or converge faster. In contrast, the random-initialized ANN model starts with a lower validation loss, and converges faster to its minimum error. Similar results were obtained with a 50% unlabeled, 50% labeled split of the datasets. The observation that semi-supervised models bring no improvement over feed-forward neural networks aligns with research trends in the broader machine learning community. The following two paragraphs give a summary of relevant works.

The greedy layer-wise pre-training of a deep belief network on unlabeled data is expected to learn weights for all layers that will provide a good initial point in the parameter space for the supervised fine-tuning step [16]. An autoencoder learns a non-linear feature transformation from unlabeled data, which can then be input to a supervised learner [5]. These models were intended to address issues in the difficult optimization problem of training deep architectures, utilizing abundantly available unlabeled data.

However, techniques including activation functions such as the rectified linear unit (ReLU [25]) and scaled exponential linear units (SELU [42]), better random weight initialization schemes [28, 24], batch normalization [34], regularization methods like dropout [67], and efficient GPU implementations that allow longer training have successfully addressed the difficulties in training deep neural networks. Researchers report unsupervised pre-training brings no improvement over pure supervised training of deep feed-forward neural networks (DNN) [25, 6]. Therefore, computationally expensive unsupervised pre-training methods like deep belief networks and the autoencoders have largely been replaced by supervised training of deep feed forward neural networks with labeled datasets, helped by transfer learning. The results from our experiments on the intrusion detection datasets agree with the above observations made by the broader machine learning community on other tasks.

### Neural network architecture search

The results of the network architecture search experiments are shown in Tables C.6, C.7 and C.8. In deep neural networks with the same number of nodes (64) in each layer (Table C.7), the depth of the network showed no correlation with the F1-score or other metrics on any of the

Table B.12: Comparing the performance of semi-supervised models (AE + ANN, DBN + ANN) with supervised ANN. The results are the mean of weighted macro average of metrics over 3 runs.

Dataset	Model	Precision	Detection rate	False alarm rate	False negative rate	F1	Train time (min)
NSL-KDD	ANN	0.8110	0.7814	0.1502	0.2186	0.7438	0.2
	AE + ANN	0.7970	0.7693	0.1510	0.2307	0.7339	0.5
	DBN + ANN	0.7894	0.7686	0.1577	0.2314	0.7336	1.2
IDS 2017	ANN	0.9943	0.9945	0.0107	0.0055	0.9941	3.0
	AE + ANN	0.9941	0.9943	0.0105	0.0057	0.9939	5.8
	DBN + ANN	0.9914	0.9920	0.0198	0.0080	0.9915	14.9
IDS 2018	ANN	0.9779	0.9827	0.0495	0.0173	0.9776	8.9
	AE + ANN	0.9734	0.9815	0.0520	0.0185	0.9763	7.6
	DBN + ANN	0.9782	0.9833	0.0502	0.0167	0.9782	10.5



Figure B.6: Results of training semi-supervised models AE + ANN and DBN + ANN (unsupervised pre-training on 75% of original training set followed by supervised fine-tuning on the other 25%) and ANN (supervised training on 25% of original training set).

three datasets. However, in deep networks with a narrowing architecture (Table C.8), accuracy and F1-scores improved with depth on the IDS 2017 and IDS 2018 datasets, as illustrated in Figure B.8. For example, the F1-score on the IDS 2017 dataset improves from 99.15% with a single-hidden-layer network to 99.61% with a 6-hidden-layer-network.

Table B.13: p-values of the 10-fold cross validation  $t$ -tests comparing ANNs with semi-supervised models.

	NSL-KDD	IDS 2017	IDS 2018
ANN vs. AE + ANN	$2.7 \times 10^{-5}$	0.45	0.8405
ANN vs. DBN + ANN	$1.8 \times 10^{-5}$	$1.8 \times 10^{-17}$	0.9911

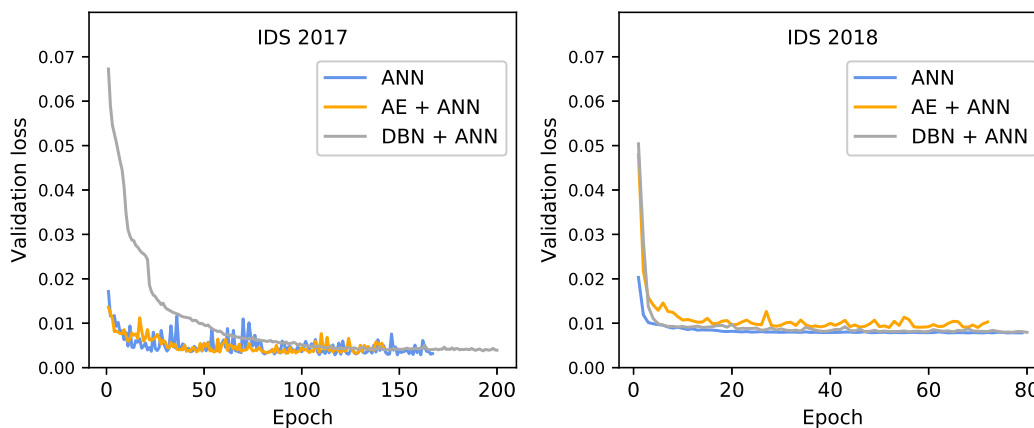


Figure B.7: Validation loss history during the supervised training stage of feed-forward neural network (ANN), autoencoder (AE + ANN) and deep belief network (DBN + ANN) models.

The statistical significance of the performance improvement was verified with a paired  $t$ -test of a 10-fold cross validation (rejection of the null hypothesis that the two models have equal accuracy with  $p$ -value =  $6.5 \times 10^{-20}$ ). This improvement corresponds to a drop in false negatives from 2248 to 820 (a 63% reduction), which can be significant in a critical task like intrusion detection. Shallow and wide networks (Table C.6) also produced improved evaluation metrics very similar to the deep and narrow architectures. The accuracy and F1-score of shallow and wide models are shown as the last 3 data points on the graphs in Figure B.8.

This improvement is likely due to the large number of parameters in these networks, which increases their learning capacity. However, it also means that training these large networks is computationally very expensive. For example, training the 10-layer network in Table C.8 takes 15x the time it takes to train the single-layer network on the IDS 2017 dataset. Adding one layer in the narrowing architectures increases training time by 16% on average. Similarly, training shallow but wide architectures take long training times due to the large number of parameters. Therefore, researchers and practitioners must consider the trade-offs between accuracy, model complexity and training and inference time when selecting the models that go into intrusion detection systems.



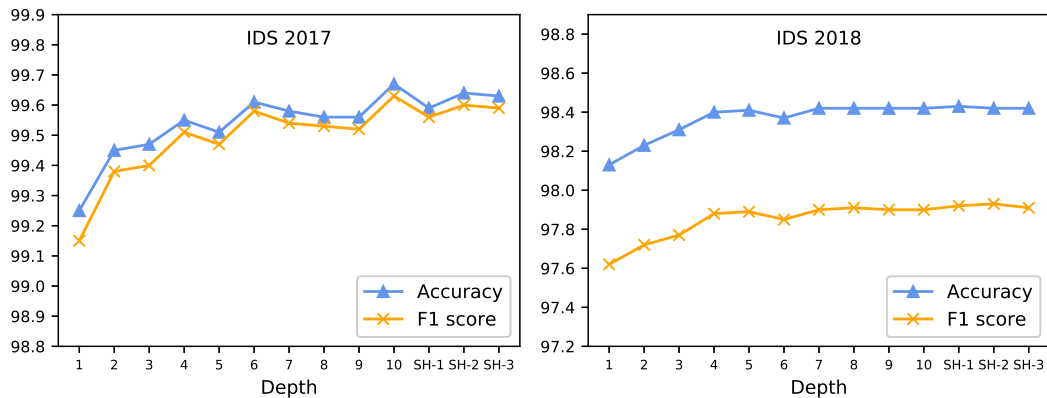


Figure B.8: Accuracy and F1-score on IDS 2017 and IDS 2018 datasets for ANNs with deep and narrowing architectures (first 10 data points in the graphs), and shallow and wide architectures (last 3 data points in graphs labeled SH-1, SH-2 and SH-3). The architectures are given in Tables C.6 and C.8.

### Measuring the effect of training set size on ANN performance

The evaluation results of training a shallow ANN model on increasingly large datasets are given in Table C.9. Evaluation metrics on the NSL-KDD and IDS 2018 test set do not show a correlation with the training set size. This suggests that smaller, stratified subsets of the NSL-KDD and IDS 2018 training sets contain the information required for training models that can reach the accuracy of models trained with the full dataset. It also indicates that practitioners can use stratified small subsets of large intrusion detection datasets for fast experimentation during the model building process.

Results on the IDS 2017 test set show that increasing the training set size improves the evaluation metrics of the ANN models (Figure B.9). The number of false negatives drop by 59.98% when the ANN model is trained with the full IDS 2017 training set as opposed to when it is trained with 10% of the training set. The statistical significance of the performance improvement was verified with a paired  $t$ -test of a 10-fold cross validation (rejection of the null hypothesis that the two models have equal accuracy with  $p$ -value =  $8.9 \times 10^{-10}$ ). Assuming that the IDS 2017 dataset is representative of normal and attack traffic in a modern network, this result suggests that it is possible to build more accurate neural network models for the intrusion detection task by collecting large amounts of representative network data.

## Results on the IDS 2018 dataset

All models trained on the IDS 2018 dataset yield very similar evaluation metrics (over 98.3% accuracy and 97.8% F1-score). Along with results reported by [18], these evaluations can be used as a basis for further study of this large and realistic dataset.

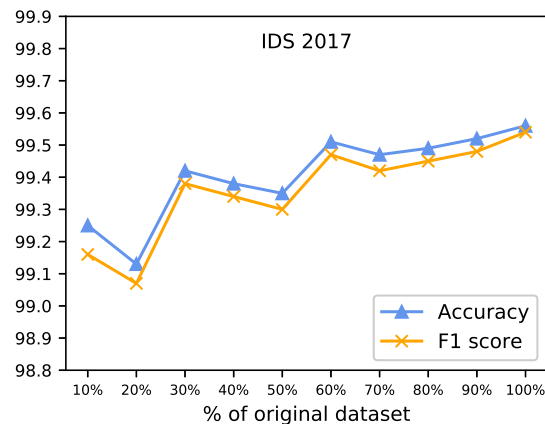


Figure B.9: Evaluation metrics from shallow ANN models trained on increasingly large subsets of the IDS 2017 dataset. Models trained with larger training sets show high accuracy and F1-score.

A summary of the conclusions that can be drawn from the results of our empirical study is listed below.

- All five evaluated models perform generally well on all four intrusion detection datasets. Considering performance (accuracy, F1-score, false negatives), training time and inference time, deep feed-forward neural networks (ANN) are the more capable model in the benchmark.
- Semi-supervised models (AE + ANN, DBN + ANN) do not perform better than other purely supervised ANNs even when they have a substantial data advantage (unlabeled). They are also slow to train due to unsupervised pre-training required.
- LSTM networks operating on flow sequences do not show improved performance for highly time-correlated attacks like denial of service or brute force. Additionally, they have high inference times due to the network complexity.
- Random forests perform well, especially on the IDS 2017 and IDS 2018 datasets. However, they are slow during inference. Since they are easy to implement and fast to train,

random forests can be used as a potential baseline model when building machine learning models for intrusion detection.

- Narrowing deep neural network architectures (large number of nodes in the first hidden layer, narrowing down to a small number of nodes in the final hidden layer) show improved performance than small networks (shallow and narrow) on IDS 2017 and IDS 2018 datasets.
- ANN performance can be improved by training it with more network flow data, which suggests that efforts to collect more data for network intrusion detection will be worthwhile.

## B.5 Issues in current research and future directions

The extensive literature survey that we conducted revealed several common weaknesses in research papers. This section discusses these issues, and highlights a number of potential future research avenues based on our observations.

- *Issues in evaluation of novel methods.* Many papers on the topic of machine learning methods for intrusion detection use only older datasets (mostly, KDD 99 and NSL-KDD) for training and evaluating the models. These datasets are not representative of modern network traffic and intrusions, and only using these datasets for validating novel methods is not adequate. This issue has been discussed in [66]. We recommend that researchers build and validate their models on recent intrusion detection datasets (see [61] for a survey of intrusion detection datasets). Comprehensive analyses of datasets, their differences and understanding how machine learning models may generalize across datasets will provide useful information to make better decisions on new research. The literature also lacks reports on evaluating intrusion detection methods in real-life attack scenarios (eg: advanced persistent threats and data exfiltration), and comparisons of traditional signature-based intrusion detection and machine learning methods in these cases. New results in this area can be valuable to the intrusion detection and machine learning research communities.
- *Issue of reproducibility.* The inability to reproduce results of published research hinders building upon those ideas. In our survey, we found that many papers on deep learning

for intrusion detection do not report adequate information on their methodology to fully understand their work (model details, operations on data etc). The machine learning research community has identified this problem, and now places a high importance on reproducibility (eg: a guide on reproducibility of machine learning research [57]). In our empirical analysis, we found that adhering to such a guide helps with conducting and reporting reproducible experiments. We have also open-sourced our code, which promotes transparency and allows researchers to use it as a reference point for their work. We encourage researchers in intrusion detection to consider the aspect of reproducibility when designing and reporting their systems and methods.

- *Semi-supervised machine learning methods for intrusion detection.* Since large quantities of unlabeled network flow data can be acquired from operating networks with relatively little effort, semi-supervised machine learning models that make use of this unlabeled data for the intrusion detection problem have been actively researched. The most frequent models in the literature are autoencoders and deep belief networks. In our empirical study, we show that these models do not perform better than feed-forward neural networks trained in a supervised fashion on labeled data. This observation aligns with the general trends in the machine learning community. Therefore, researchers in the intrusion detection domain may find it more fruitful to explore new methods in unsupervised and semi-supervised machine learning.
- *Exploring the use of novel machine learning paradigms and methods.* The field of machine learning has seen many advances in the recent years, some of which may be used to address problems in the intrusion detection domain. For example, transfer learning methods will allow practitioners to download pre-trained neural network intrusion detectors and adapt them to a new network environment with minimal supervised training on new data. Techniques such as the attention mechanism may be effective in detecting long-term attacks. Neural Architecture Search methods can be employed to find neural networks that are optimal under a set of metrics (eg: minimum false negatives, low model complexity and high inference throughput) for different intrusion detection settings. Research on interpretable machine learning could be used to explain the classifications of an intrusion detection system. For example, identifying which features in a flow record or set of packets were significant [54] for an attack detection may provide useful information to a security analyst for investigating a threat incident (eg: root cause analysis and triage). In the case of misclassifications or overfitting, such explanations can be useful to the system developer to identify issues or weaknesses of machine learning models (eg: reliance on spurious feature correlations [47]) and improve them.

- *Open-source machine learning based intrusion detection systems (IDS)*. There are several popular rule-based open-source IDS projects [56, 62] that have contributed to research and development in the field of rule-based intrusion detection. However, no such projects exist for machine learning based IDS. An open-source IDS with machine learning detectors will allow researchers to quickly deploy an IDS in a network environment and benchmark their models against a repository of machine learning based intrusion detectors. It may also help in fostering collaboration and attracting new researchers to the field.

## B.6 Conclusion

Attacks against computer networks are a growing threat and intrusion detection systems perform the critical task of detecting them and alerting the security teams. Machine learning algorithms have become a viable option for network intrusion detection, as they are capable of learning intrusion patterns from large datasets. In this study, we present the results of a broad literature survey on the topic of deep learning models for network intrusion detection, and a set of experiments comparing the performance of four key deep learning models on four intrusion detection datasets. We have made the implementation and the complete set of results publicly available, so that researchers may build upon it.

The results show that supervised deep feed-forward neural networks (ANN) perform well on all four datasets across all metrics (accuracy, F1-score, false negatives, training and inference time). Two popular semi-supervised learning models, autoencoders and deep belief networks do not perform better than supervised feed-forward neural networks. The accuracy of ANNs increase as they are trained on larger datasets, which suggests that efforts to build large datasets with labeled flows of benign and attack traffic will be a worthwhile investment.

Our survey revealed several weaknesses in current research such as model evaluation performed only on legacy datasets and inadequate details of models leading to experiments and results that cannot be replicated. Several potential research avenues in the field of machine learning for intrusion detection have been suggested. This survey aims to provide a bird's-eye view of the field of deep learning methods for intrusion detection, along with reproducible empirical results that researchers can rely on.

The set of experiments in this study were designed and conducted to identify the capability

of different machine learning models in the intrusion detection task and to give researchers a bird's-eye view of the field. However, some limitations exist in the study due to resource constraints. We have evaluated the models on only four datasets, which lack variety in terms of network traffic patterns and the attacks included in them. Furthermore, all models in the benchmark classify flow records, and no analysis is performed at the packet-level. For future work, we plan to extend the benchmark by training and evaluating these models on more large intrusion detection datasets. Other varieties of recurrent neural networks will also be added to the model repository.

## References

- [1] A. Abeshu and N. Chilamkurti. “Deep Learning: The Frontier for Distributed Attack Detection in Fog-to-Things Computing”. In: *IEEE Communications Magazine* 56.2 (Feb. 2018), pp. 169–175. DOI: 10.1109/MCOM.2018.1700332.
- [2] M. Z. Alom, V. Bontupalli, and T. M. Taha. “Intrusion Detection Using Deep Belief Networks”. In: *2015 National Aerospace and Electronics Conference (NAECON)*. June 2015, pp. 339–344. DOI: 10.1109/NAECON.2015.7443094.
- [3] K. Alrawashdeh and C. Purdy. “Toward an Online Anomaly Intrusion Detection System Based on Deep Learning”. In: *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. Dec. 2016, pp. 195–200. DOI: 10.1109/ICMLA.2016.0040.
- [4] Dario Amodei et al. “Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin”. In: *International Conference on Machine Learning*. June 2016, pp. 173–182.
- [5] Pierre Baldi. “Autoencoders, Unsupervised Learning, and Deep Architectures”. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. June 2012, pp. 37–49.
- [6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation Learning: A Review and New Perspectives”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50.
- [7] Daniel S. Berman et al. “A Survey of Deep Learning Methods for Cyber Security”. In: *Information* 10.4 (Apr. 2019), p. 122. DOI: 10.3390/info10040122.
- [8] Loic Bontemps et al. “Collective Anomaly Detection Based on Long Short Term Memory Recurrent Neural Network”. In: *arXiv* (Mar. 2017). DOI: 10.48550/arXiv.1703.09752.

- [9] A. L. Buczak and E. Guven. “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection”. In: *IEEE Communications Surveys Tutorials* 18.2 (2016), pp. 1153–1176. doi: 10.1109/COMST.2015.2494502.
- [10] Nadia Chaabouni et al. “Network Intrusion Detection for IoT Security Based on Learning Techniques”. In: *IEEE Communications Surveys Tutorials* 21.3 (2019), pp. 2671–2701. doi: 10.1109/COMST.2019.2896380.
- [11] Cisco. *Cisco Security Analytics Whitepaper*. Tech. rep. 2018. URL: <https://www.cisco.com/c/dam/en/us/products/collateral/security/stealthwatch/white-paper-c11-740605.pdf> (visited on 05/24/2019).
- [12] CyberEdge Group. *2019 Cyberthreat Defense Report*. Tech. rep. 2019. URL: <https://www.imperva.com/resources/reports/CyberEdge-2019-CDR-Report-v1.1.pdf> (visited on 03/16/2024).
- [13] Janez Demšar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *Journal of Machine Learning Research* 7.Jan (2006), pp. 1–30.
- [14] A. Diro and N. Chilamkurti. “Leveraging LSTM Networks for Attack Detection in Fog-to-Things Communications”. In: *IEEE Communications Magazine* 56.9 (Sept. 2018), pp. 124–130. doi: 10.1109/MCOM.2018.1701270.
- [15] Abebe Abeshu Diro and Naveen Chilamkurti. “Distributed Attack Detection Scheme Using Deep Learning Approach for Internet of Things”. In: *Future Generation Computer Systems* 82 (May 2018), pp. 761–768. doi: 10.1016/j.future.2017.08.043.
- [16] Dumitru Erhan et al. “Why Does Unsupervised Pre-Training Help Deep Learning?” In: *Journal of Machine Learning Research* 11.Feb (2010), pp. 625–660.
- [17] Nutan Haq Farah et al. “Application of Machine Learning Approaches in Intrusion Detection System: A Survey”. In: *International Journal of Advanced Research in Artificial Intelligence* 4.3 (2015). doi: 10.14569/IJARAI.2015.040302.
- [18] Mohamed Amine Ferrag et al. “Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study”. In: *Journal of Information Security and Applications* 50 (Feb. 2020), p. 102419. doi: 10.1016/j.jisa.2019.102419.
- [19] Ugo Fiore et al. “Network Anomaly Detection with the Restricted Boltzmann Machine”. In: *Neurocomputing*. Advances in Cognitive and Ubiquitous Computing 122 (Dec. 2013), pp. 13–23. doi: 10.1016/j.neucom.2012.11.050.
- [20] FireEye. *M-Trends 2019 Cyber Security Report*. Tech. rep. 2019. URL: <https://www.fireeye.com/current-threats/annual-threat-report/mtrends.html> (visited on 08/20/2019).

- [21] Sunanda Gamage and Jagath Samarabandu. “Deep Learning Methods in Network Intrusion Detection: A Survey and an Objective Comparison”. In: *Journal of Network and Computer Applications* 169 (Nov. 2020), p. 102767. doi: 10.1016/j.jnca.2020.102767.
- [22] N. Gao et al. “An Intrusion Detection Model Based on Deep Belief Networks”. In: *2014 Second International Conference on Advanced Cloud and Big Data*. Nov. 2014, pp. 247–252. doi: 10.1109/CBD.2014.41.
- [23] Mohammed Ali Al-Garadi et al. “A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security”. In: *IEEE Communications Surveys & Tutorials* 22.3 (2020), pp. 1646–1685. doi: 10.1109/COMST.2020.2988293.
- [24] Xavier Glorot and Yoshua Bengio. “Understanding the Difficulty of Training Deep Feed-forward Neural Networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Mar. 2010, pp. 249–256.
- [25] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. June 2011, pp. 315–323.
- [26] CyberEdge Group. *2019 Cyberthreat Defense Report*. Tech. rep. CyberEdge Group, 2019. URL: <https://go.illusivenetworks.com/2019-cyberthreat-defense-report> (visited on 08/19/2019).
- [27] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.
- [28] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV ’15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 1026–1034. doi: 10.1109/ICCV.2015.123.
- [29] Youbiao He, Gihan J. Mendis, and Jin Wei. “Real-Time Detection of False Data Injection Attacks in Smart Grid: A Deep Learning-Based Intelligent Mechanism”. In: *IEEE Transactions on Smart Grid* 8.5 (Sept. 2017), pp. 2505–2516. doi: 10.1109/TSG.2017.2703842.
- [30] Geoffrey Hinton et al. “Deep Neural Networks for Acoustic Modeling in Speech Recognition”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. doi: 10.1109/MSP.2012.2205597.
- [31] E. Hodo et al. “Threat Analysis of IoT Networks Using Artificial Neural Network Intrusion Detection System”. In: *2016 International Symposium on Networks, Computers and Communications (ISNCC)*. May 2016, pp. 1–6. doi: 10.1109/ISNCC.2016.7746067.



- [32] Elike Hodo et al. “Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey”. In: *arXiv* (Jan. 2017). doi: 10.48550/arXiv.1701.02145.
- [33] Bhupendra Ingre and Anamika Yadav. “Performance Analysis of NSL-KDD Dataset Using ANN”. In: *2015 International Conference on Signal Processing and Communication Engineering Systems*. Jan. 2015, pp. 92–96. doi: 10.1109/SPACES.2015.7058223.
- [34] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. France: JMLR, July 2015, pp. 448–456.
- [35] Ahmad Javaid et al. “A Deep Learning Approach for Network Intrusion Detection System”. In: *9th EAI International Conference on Bio-Inspired Information and Communications Technologies*. BICT’15. New York City, United States: ICST, 2016, pp. 21–26. doi: 10.4108/eai.3-12-2015.2262516.
- [36] F. Jiang et al. “Deep Learning Based Multi-Channel Intelligent Attack Detection for Data Security”. In: *IEEE Transactions on Sustainable Computing* (2019), pp. 1–1. doi: 10.1109/TSUSC.2018.2793284.
- [37] Jin Kim et al. “Method of Intrusion Detection Using Deep Neural Network”. In: *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*. Feb. 2017, pp. 313–316. doi: 10.1109/BIGCOMP.2017.7881684.
- [38] Min-Joo Kang and Je-Won Kang. “Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security”. In: *PLOS ONE* 11.6 (June 2016), e0155781. doi: 10.1371/journal.pone.0155781.
- [39] KDD. *KDD Cup 1999 Data*. 1999. URL: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> (visited on 08/20/2019).
- [40] Jihyun Kim et al. “Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection”. In: *2016 International Conference on Platform Technology and Service (PlatCon)*. Feb. 2016, pp. 1–5. doi: 10.1109/PlatCon.2016.7456805.
- [41] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations*. San Diego, USA, 2015. doi: 10.48550/arXiv.1412.6980.
- [42] Günter Klambauer et al. “Self-Normalizing Neural Networks”. In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 971–980.
- [43] Casey Kneale and Kolia Sadeghi. “Semisupervised Adversarial Neural Networks for Cyber Security Transfer Learning”. In: *arXiv* (July 2019). doi: 10.48550/arXiv.1907.11129.

- [44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105.
- [45] D. Kwon et al. “An Empirical Study on Network Anomaly Detection Using Convolutional Neural Networks”. In: *38th International Conference on Distributed Computing Systems (ICDCS)*. July 2018, pp. 1595–1598. doi: 10.1109/ICDCS.2018.00178.
- [46] Donghwoon Kwon et al. “A Survey of Deep Learning-Based Network Anomaly Detection”. In: *Cluster Computing (2017)*. doi: 10.1007/s10586-017-1117-8.
- [47] Sebastian Lapuschkin et al. “Unmasking Clever Hans Predictors and Assessing What Machines Really Learn”. In: *Nature Communications* 10.1 (Mar. 2019), p. 1096. doi: 10.1038/s41467-019-08987-4.
- [48] T. Le, J. Kim, and H. Kim. “An Effective Intrusion Detection Classifier Using Long Short-Term Memory with Gradient Descent Optimization”. In: *2017 International Conference on Platform Technology and Service (PlatCon)*. Feb. 2017, pp. 1–6. doi: 10.1109/PlatCon.2017.7883684.
- [49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. doi: 10.1038/nature14539.
- [50] Yuancheng Li, Rong Ma, and Runhai Jiao. “A Hybrid Malicious Code Detection Method Based on Deep Learning”. In: *International Journal of Software Engineering and Its Applications* 9 (May 2015), pp. 205–216. doi: 10.14257/ijseia.2015.9.5.21.
- [51] G. Loukas et al. “Cloud-Based Cyber-Physical Intrusion Detection for Vehicles Using Deep Learning”. In: *IEEE Access* 6 (2018), pp. 3491–3508. doi: 10.1109/ACCESS.2017.2782159.
- [52] Tao Ma et al. “A Hybrid Spectral Clustering and Deep Neural Network Ensemble Algorithm for Intrusion Detection in Sensor Networks”. In: *Sensors (Basel, Switzerland)* 16.10 (Oct. 2016). doi: 10.3390/s16101701.
- [53] Preeti Mishra et al. “A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection”. In: *IEEE Communications Surveys Tutorials* 21.1 (2019), pp. 686–728. doi: 10.1109/COMST.2018.2847722.
- [54] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. “Methods for Interpreting and Understanding Deep Neural Networks”. In: *Digital Signal Processing* 73 (Feb. 2018), pp. 1–15. doi: 10.1016/j.dsp.2017.10.011.

- [55] S. Mukkamala, G. Janoski, and A. Sung. “Intrusion Detection Using Neural Networks and Support Vector Machines”. In: *Proceedings of the 2002 International Joint Conference on Neural Networks*. Vol. 2. May 2002, pp. 1702–1707. doi: 10.1109/IJCNN.2002.1007774.
- [56] Vern Paxson. “Bro: A System for Detecting Network Intruders in Real-Time”. In: *Computer Networks: The International Journal of Computer and Telecommunications Networking* 31.23-24 (Dec. 1999), pp. 2435–2463. doi: 10.1016/S1389-1286(99)00112-7.
- [57] Joelle Pineau. *ReproducibilityChecklist-v1.2*. Tech. rep. 2019, p. 1. URL: <https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf> (visited on 01/05/2020).
- [58] Sasanka Potluri and Christian Diedrich. “Accelerated Deep Neural Networks for Enhanced Intrusion Detection System”. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2016, pp. 1–8. doi: 10.1109/ETFA.2016.7733515.
- [59] Benjamin J. Radford et al. “Network Traffic Anomaly Detection Using Recurrent Neural Networks”. In: *arXiv* (Mar. 2018). doi: 10.48550/arXiv.1803.10769.
- [60] Radiflow. *Detection of a Crypto-Mining Malware Attack at a Water Utility*. Tech. rep. 2018. URL: <https://www.radiflow.com/wp-content/uploads/CS-Crypto-Mining-Water-Utility-2018.pdf> (visited on 08/19/2019).
- [61] Markus Ring et al. “A Survey of Network-Based Intrusion Detection Data Sets”. In: *Computers & Security* 86 (Sept. 2019), pp. 147–167. doi: 10.1016/j.cose.2019.06.005.
- [62] Martin Roesch. “Snort - Lightweight Intrusion Detection for Networks”. In: *Proceedings of the 13th USENIX Conference on System Administration*. LISA '99. USA: USENIX Association, Nov. 1999, pp. 229–238.
- [63] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”. In: *4th International Conference on Information Systems Security and Privacy*. Funchal, Portugal, 2018, pp. 108–116. doi: 10.5220/0006639801080116.
- [64] Timothy J. Shimeall and Jonathan M. Spring. “Chapter 12 - Recognition Strategies: Intrusion Detection and Prevention”. In: *Introduction to Information Security*. Syngress, Jan. 2014, pp. 253–274.
- [65] N. Shone et al. “A Deep Learning Approach to Network Intrusion Detection”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.1 (Feb. 2018), pp. 41–50. doi: 10.1109/TETCI.2017.2772792.

- [66] Robin Sommer and Vern Paxson. “Outside the Closed World: On Using Machine Learning for Network Intrusion Detection”. In: *2010 IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE, 2010, pp. 305–316. doi: 10.1109/SP.2010.25.
- [67] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.
- [68] Ralf C. Staudemeyer. “Applying Long Short-Term Memory Recurrent Neural Networks to Intrusion Detection”. In: *South African Computer Journal* 56.1 (July 2015). doi: 10.18489/sacj.v56i1.248.
- [69] Nasrin Sultana et al. “Survey on SDN Based Network Intrusion Detection System Using Machine Learning Approaches”. In: *Peer-to-Peer Networking and Applications* 12.2 (Mar. 2019), pp. 493–501. doi: 10.1007/s12083-017-0630-0.
- [70] Dzmitry Sutskever, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *3rd International Conference on Learning Representations*. San Diego, USA, 2015. doi: 10.48550/arXiv.1409.0473.
- [71] Zahra Taghiyarrenani et al. “Transfer Learning Based Intrusion Detection”. In: *2018 8th International Conference on Computer and Knowledge Engineering (ICCCKE)*. Oct. 2018, pp. 92–97. doi: 10.1109/ICCCKE.2018.8566601.
- [72] T. A. Tang et al. “Deep Learning Approach for Network Intrusion Detection in Software Defined Networking”. In: *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. Oct. 2016, pp. 258–263. doi: 10.1109/WINCOM.2016.7777224.
- [73] M. Tavallaee et al. “A Detailed Analysis of the KDD CUP 99 Data Set”. In: *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. July 2009, pp. 1–6. doi: 10.1109/CISDA.2009.5356528.
- [74] A. Taylor, S. Leblanc, and N. Japkowicz. “Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks”. In: *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Oct. 2016, pp. 130–139. doi: 10.1109/DSAA.2016.20.
- [75] Tijmen Tieleman. “Training Restricted Boltzmann Machines Using Approximations to the Likelihood Gradient”. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML ’08. Helsinki, Finland: ACM, 2008, pp. 1064–1071. doi: 10.1145/1390156.1390290.

- [76] Chih-Fong Tsai et al. “Intrusion Detection by Machine Learning: A Review”. In: *Expert Systems with Applications* 36.10 (2009), pp. 11994–12000. doi: 10.1016/j.eswa.2009.05.029.
- [77] R. Vinayakumar et al. “Deep Learning Approach for Intelligent Intrusion Detection System”. In: *IEEE Access* 7 (2019), pp. 41525–41550. doi: 10.1109/ACCESS.2019.2895334.
- [78] Z. Wang. “Deep Learning-Based Intrusion Detection With Adversaries”. In: *IEEE Access* 6 (2018), pp. 38367–38384. doi: 10.1109/ACCESS.2018.2854599.
- [79] Wei Wang et al. “Malware Traffic Classification Using Convolutional Neural Network for Representation Learning”. In: *2017 International Conference on Information Networking (ICOIN)*. Jan. 2017, pp. 712–717. doi: 10.1109/ICOIN.2017.7899588.
- [80] Y. Xin et al. “Machine Learning and Deep Learning Methods for Cybersecurity”. In: *IEEE Access* 6 (2018), pp. 35365–35381. doi: 10.1109/ACCESS.2018.2836950.
- [81] C. Xu et al. “An Intrusion Detection System Using a Deep Neural Network With Gated Recurrent Units”. In: *IEEE Access* 6 (2018), pp. 48697–48707. doi: 10.1109/ACCESS.2018.2867564.
- [82] J. Yan et al. “A Comparative Study of Off-Line Deep Learning Based Network Intrusion Detection”. In: *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. July 2018, pp. 299–304. doi: 10.1109/ICUFN.2018.8436774.
- [83] C. Yin et al. “A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks”. In: *IEEE Access* 5 (2017), pp. 21954–21961. doi: 10.1109/ACCESS.2017.2762418.
- [84] Y. Zeng et al. “Deep-Full-Range : A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework”. In: *IEEE Access* 7 (2019), pp. 45182–45190. doi: 10.1109/ACCESS.2019.2908225.
- [85] Juan Zhao et al. “Transfer Learning for Detecting Unknown Network Attacks”. In: *EURASIP Journal on Information Security* 2019.1 (Feb. 2019), p. 1. doi: 10.1186/s13635-019-0084-4.

# Appendix C

## Network Intrusion Detection Results

### C.1 Evaluation metrics

The following evaluation metrics are calculated based on the values in the confusion matrix (Table B.3).

- Overall accuracy: the proportion of correctly classified instances out of a given set of instances. When the classes are heavily imbalanced, this metric is less useful.

$$accuracy = \frac{TP + TN}{all\ instances}$$

- Precision or positive predictive value (PPV): the proportion of correct classifications out of a set of predictions classified as attacks.

$$precision = \frac{TP}{TP + FP}$$

- Recall, sensitivity, true positive rate or detection rate: the proportion of correct attack classifications out of a given set of attack instances. This metric represents the attack detection capability of the model.

$$recall = \frac{TP}{TP + FN}$$

- Specificity or true negative rate: the proportion of correct benign classifications out of a given set of benign instances.

$$specificity = \frac{TN}{TN + FP}$$

- False positive rate (FPR) or Fall out: the proportion of instances classified incorrectly as attacks out of a given set of benign instances. An intrusion detection system with a high false positive rate is practically less useful, as it generates a flood of false alarms, overwhelming security operators.

$$FPR = \frac{FP}{TN + FP} = 1 - specificity$$

- False negative rate (FNR) or miss rate: the proportion instances classified incorrectly as benign out of a given set of attack instances. Note that in an IDS, a false negative corresponds to an attack that was not detected by the system; an event that can have severe consequences.

$$FNR = \frac{FN}{TP + FN} = 1 - recall$$

- F1 score: the harmonic mean of precision and recall. For class-imbalanced problems, which is usually the case with intrusion datasets, this is a useful metric to evaluate models.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

## C.2 Results of experiments

Table C.1: Overall results (weighted macro average of metrics) of the models on the KDD 99 test set in the format: mean (std\_dev) from 5 repetitions.

Model	Accuracy	Precision	False alarm rate	False negative rate	F1	Train epochs	Train time (min)	Test time (sec)
ANN: shallow	0.9237 (0.0010)	0.9386 (0.0031)	0.0199 (0.0002)	0.0763 (0.0010)	0.9046 (0.0011)	2.0 (1.00)	0.1 (0.05)	1.0 (0.01)
ANN: deep	0.9240 (0.0008)	0.9385 (0.0036)	0.0203 (0.0003)	0.0760 (0.0008)	0.9051 (0.0010)	1.7 (0.58)	0.1 (0.03)	1.4 (0.04)
AE + ANN: shallow	0.9225 (0.0008)	0.9274 (0.0171)	0.0211 (0.0010)	0.0775 (0.0008)	0.9040 (0.0019)	11.0 (0.00)	1.0 (0.00)	1.7 (0.06)
AE + ANN: deep	0.9224 (0.0013)	0.9316 (0.0094)	0.0216 (0.0009)	0.0776 (0.0013)	0.9035 (0.0023)	15.7 (0.00)	0.7 (0.00)	1.7 (0.91)
DBN + ANN: shallow	0.9231 (0.0006)	0.9381 (0.0036)	0.0199 (0.0003)	0.0769 (0.0006)	0.9043 (0.0006)	12.7 (0.58)	2.6 (0.02)	1.0 (0.04)
DBN + ANN: deep	0.9226 (0.0011)	0.9368 (0.0038)	0.0241 (0.0034)	0.0774 (0.0011)	0.9034 (0.0014)	39.0 (4.58)	6.2 (0.38)	1.5 (0.04)
LSTM: shallow	0.9232 (0.0006)	0.9312 (0.0018)	0.0217 (0.0010)	0.0768 (0.0006)	0.9047 (0.0006)	7.0 (4.36)	0.4 (0.22)	18.8 (0.00)
LSTM: deep	0.9229 (0.0015)	0.9281 (0.0060)	0.0243 (0.0048)	0.0771 (0.0015)	0.9041 (0.0026)	17.3 (16.65)	0.9 (0.83)	26.0 (0.72)
RF	0.9234 (0.0008)	0.9405 (0.0005)	0.0218 (0.0002)	0.0766 (0.0008)	0.9034 (0.0009)		2.0 (0.00)	7.7 (0.00)

Table C.2: Overall results (weighted macro average of metrics) of the models on the NSL KDD test set in the format: mean (std\_dev) from 5 repetitions.

Model	Accuracy	Precision	False alarm rate	False negative rate	F1	Train epochs	Train time (minutes)	Test time (secs)
ANN: shallow	0.7725 (0.0071)	0.8104 (0.0084)	0.1566 (0.0065)	0.2275 (0.0071)	0.7360 (0.0090)	1.7 (0.58)	0.1 (0.00)	1.0 (0.02)
ANN: deep	0.7726 (0.0127)	0.8131 (0.0112)	0.1588 (0.0055)	0.2274 (0.0127)	0.7332 (0.0138)	2.0 (0.00)	0.1 (0.00)	1.4 (0.03)
AE + ANN: shallow	0.7694 (0.0102)	0.8051 (0.0167)	0.1564 (0.0054)	0.2306 (0.0102)	0.7362 (0.0099)	20.7 (4.62)	0.2 (0.04)	1.8 (0.03)
AE + ANN: deep	0.7688 (0.0078)	0.8022 (0.0082)	0.1582 (0.0067)	0.2312 (0.0078)	0.7275 (0.0144)	20.6 (2.65)	0.2 (0.02)	2.0 (0.04)
DBN + ANN: shallow	0.7693 (0.0096)	0.8097 (0.0110)	0.1566 (0.0067)	0.2307 (0.0096)	0.7312 (0.0127)	12.3 (0.58)	0.1 (0.00)	1.0 (0.01)
DBN + ANN: deep	0.7633 (0.0093)	0.7878 (0.0272)	0.1572 (0.0090)	0.2367 (0.0093)	0.7296 (0.0092)	32.7 (0.58)	0.2 (0.00)	1.4 (0.04)
LSTM: shallow	0.7718 (0.0034)	0.8030 (0.0088)	0.1508 (0.0028)	0.2282 (0.0034)	0.7370 (0.0024)	52.7 (22.85)	0.4 (0.19)	18.7 (0.93)
LSTM: deep	0.7726 (0.0087)	0.8040 (0.0140)	0.1513 (0.0029)	0.2274 (0.0087)	0.7371 (0.0078)	18.3 (6.81)	0.2 (0.06)	15.0 (0.12)
RF	0.7580 (0.0035)	0.8018 (0.0056)	0.1710 (0.0030)	0.2420 (0.0035)	0.7121 (0.0030)		0.1 (0.00)	8.2 (0.00)



Table C.3: Overall results (weighted macro average of metrics) of the models on the IDS 2017 test set in the format: mean (std\_dev) from 5 repetitions.

Model	Accuracy	Precision	False alarm rate	False negative rate	F1	Train epochs	Train time (minutes)	Test time (secs)
ANN: shallow	0.9954 (0.0002)	0.9952 (0.0002)	0.0086 (0.0004)	0.0046 (0.0002)	0.9950 (0.0001)	40.7 (17.04)	0.7 (0.28)	0.9 (0.01)
ANN: deep	0.9958 (0.0006)	0.9956 (0.0006)	0.0063 (0.0009)	0.0042 (0.0006)	0.9954 (0.0006)	21.3 (5.13)	0.7 (0.17)	1.4 (0.04)
AE + ANN: shallow	0.9855 (0.0006)	0.9853 (0.0005)	0.0247 (0.0099)	0.0145 (0.0006)	0.9851 (0.0006)	110.0 (0.00)	1.2 (0.00)	1.6 (0.02)
AE + ANN: deep	0.9812 (0.0015)	0.9810 (0.0016)	0.0370 (0.0099)	0.0188 (0.0015)	0.9808 (0.0016)	96.0 (28.29)	1.7 (0.47)	1.9 (0.05)
DBN + ANN: shallow	0.9945 (0.0003)	0.9942 (0.0005)	0.0078 (0.0018)	0.0055 (0.0003)	0.9941 (0.0004)	64.3 (17.04)	1.8 (0.28)	0.9 (0.02)
DBN + ANN: deep	0.9956 (0.0001)	0.9954 (0.0001)	0.0080 (0.0010)	0.0044 (0.0001)	0.9952 (0.0001)	99.3 (17.24)	6.0 (0.29)	1.3 (0.03)
LSTM: shallow	0.9954 (0.0003)	0.9951 (0.0003)	0.0080 (0.0017)	0.0046 (0.0003)	0.9950 (0.0004)	76.0 (20.78)	1.3 (0.35)	15.7 (0.21)
LSTM: deep	0.9957 (0.0004)	0.9955 (0.0004)	0.0079 (0.0014)	0.0043 (0.0004)	0.9954 (0.0006)	80.7 (30.09)	1.3 (0.50)	23.4 (0.19)
RF	0.9986 (0.0001)	0.9986 (0.0001)	0.0024 (0.0002)	0.0014 (0.0001)	0.9985 (0.0001)		2.7 (0.00)	19.6 (0.08)

Table C.4: Overall results (weighted macro average of metrics) of the models on the IDS 2018 test set in the format: mean (std\_dev) from 5 repetitions.

Model	Accuracy	Precision	False alarm rate	False negative rate	F1	Train epochs	Train time (minutes)	Test time (secs)
ANN: shallow	0.9836 (0.0002)	0.9785 (0.0003)	0.0497 (0.0009)	0.0164 (0.0002)	0.9785 (0.0002)	43.0 (5.20)	5.0 (0.61)	0.9 (0.02)
ANN: deep	0.9838 (0.0003)	0.9754 (0.0024)	0.0492 (0.0003)	0.0162 (0.0003)	0.9785 (0.0004)	19.7 (7.64)	3.0 (1.15)	1.4 (0.11)
AE + ANN: shallow	0.9807 (0.0006)	0.9719 (0.0018)	0.0576 (0.0043)	0.0193 (0.0006)	0.9754 (0.0007)	40.3 (27.23)	7.7 (2.27)	1.6 (0.04)
AE + ANN: deep	0.9822 (0.0002)	0.9750 (0.0022)	0.0514 (0.0008)	0.0178 (0.0002)	0.9768 (0.0002)	29.8 (2.31)	6.5 (0.27)	1.9 (0.10)
DBN + ANN: shallow	0.9819 (0.0009)	0.9744 (0.0024)	0.0501 (0.0004)	0.0181 (0.0009)	0.9766 (0.0010)	19.0 (5.00)	5.1 (0.50)	0.9 (0.02)
DBN + ANN: deep	0.9831 (0.0006)	0.9761 (0.0032)	0.0496 (0.0006)	0.0169 (0.0006)	0.9778 (0.0008)	44.0 (8.89)	25.1 (0.59)	1.3 (0.05)
LSTM: shallow	0.9888 (0.0001)	0.9840 (0.0009)	0.0509 (0.0003)	0.0112 (0.0001)	0.9839 (0.0001)	66.0 (15.72)	5.5 (1.31)	16.1 (0.20)
LSTM: deep	0.9860 (0.0031)	0.9800 (0.0056)	0.0641 (0.0143)	0.0140 (0.0031)	0.9809 (0.0032)	46.0 (31.10)	3.8 (2.59)	24.0 (0.14)
RF	0.9834 (0.0001)	0.9780 (0.0001)	0.0465 (0.0003)	0.0166 (0.0001)	0.9796 (0.0001)		5.6 (0.01)	25.5 (0.10)

Table C.5: p-values of the Nemenyi pairwise multiple comparison test on the accuracy figures of models with the null hypotheses: performance of each pair of models in the matrix is equal. p-values below a 0.05 significance threshold (null hypothesis is rejected) are shown in bold.

	ANN: shal- low	ANN: deep	AE + ANN: shal- low	AE + ANN: deep	DBN + ANN: shal- low	DBN + ANN: deep	LSTM: shal- low	LSTM: deep	RF
ANN: shallow	-1	0.9	0.4971	0.3274	0.8106	0.8889	0.9	0.9	0.9
ANN: deep	0.9	-1	0.0736	<b>0.0341</b>	0.2553	0.3274	0.9	0.9	0.8889
AE + ANN: shallow	0.4971	0.0736	-1	0.9	0.9	0.9	0.5756	0.4118	0.8106
AE + ANN: deep	0.3274	<b>0.0341</b>	0.9	-1	0.9	0.9	0.4118	0.2553	0.6539
DBN + ANN: shallow	0.8106	0.2553	0.9	0.9	-1	0.9	0.8889	0.7323	0.9
DBN + ANN: deep	0.8889	0.3274	0.9	0.9	0.9	-1	0.9	0.8106	0.9
LSTM: shallow	0.9	0.9	0.5756	0.4118	0.8889	0.9	-1	0.9	0.9
LSTM: deep	0.9	0.9	0.4118	0.2553	0.7323	0.8106	0.9	-1	0.9
RF	0.9	0.8889	0.8106	0.6539	0.9	0.9	0.9	0.9	-1

Table C.6: Performance of shallow and wide neural network architectures. No. of parameters are calculated with 78 input units and 12 output units. (Acc = accuracy, Prec = precision, FNR = false negative rate, F1 = F1-score, T.T. = training time in minutes)

Name	No. of nodes in hidden layers	No. of params	IDS 2017				IDS 2018			
			Acc.	FNR	F1	T.T.	Acc.	FNR	F1	T.T.
SH-1	1000-500-200	680.4 K	0.9959	0.0041	0.9956	10.4	0.9843	0.0157	0.9792	20.9
SH-2	2000-1000-500	2.66 M	0.9964	0.0036	0.996	15.2	0.9842	0.0158	0.9793	32.1
SH-3	3000-1500-500	5.49 M	0.9963	0.0037	0.9959	30.4	0.9842	0.0158	0.9791	68.1

Table C.7: Performance of deep neural network architectures with the same no. of nodes in each layer. No. of parameters are calculated with 78 input units and 12 output units. (Acc = accuracy, Prec = precision, FNR = false negative rate, F1 = F1-score, T.T. = training time in minutes)

Depth	No. of nodes in hidden layers	No. of params	IDS 2017				IDS 2018			
			Acc.	FNR	F1	T.T.	Acc.	FNR	F1	T.T.
2	64-64	9.9 K	0.9954	0.0046	0.9951	4.8	0.9840	0.0160	0.9789	9.8
3	64-64-64	14 K	0.9961	0.0039	0.9957	5.6	0.9840	0.0160	0.9788	8.5
4	64-64-64-64	18 K	0.9946	0.0054	0.9942	6.0	0.9836	0.0164	0.9782	8.9
5	64-64-64-64-64	22.1 K	0.9947	0.0053	0.9943	6.8	0.9832	0.0168	0.9778	11.5
6	64-64-64-64-64-64	26.2 K	0.9958	0.0042	0.9954	7.6	0.9841	0.0159	0.9789	17.1
7	64-64-64-64-64-64-64	30.3 K	0.9955	0.0045	0.9951	8.4	0.9829	0.0171	0.9775	7.1
8	64-64-64-64-64-64-64-64	34.4 K	0.9952	0.0048	0.9948	9.2	0.9837	0.0163	0.9783	19.7
9	64-64-64-64-64-64-64-64-64	38.6 K	0.9955	0.0045	0.9951	9.6	0.9832	0.0168	0.9778	18.9
10	64-64-64-64-64-64-64-64-64-64	42.7 K	0.9952	0.0048	0.9948	10.4	0.9837	0.0163	0.9785	23.1

Table C.8: Performance of narrowing deep neural network architectures. No. of parameters are calculated with 78 input units and 12 output units. (Acc = accuracy, Prec = precision, FNR = false negative rate, F1 = F1-score, T.T. = training time in minutes)

Depth	No. of nodes in hidden layers	No. of params	IDS 2017				IDS 2018			
			Acc.	FNR	F1	T.T.	Acc.	FNR	F1	T.T.
1	16	1.4 K	0.9925	0.0075	0.9915	4.0	0.9813	0.0187	0.9762	3.7
2	32-16	2.7 K	0.9945	0.0055	0.9938	4.8	0.9823	0.0177	0.9772	5.0
3	64-32-16	7.7 K	0.9947	0.0053	0.9940	5.6	0.9831	0.0169	0.9777	9.5
4	128-64-32-16	20.1 K	0.9955	0.0045	0.9951	6.0	0.9840	0.0160	0.9788	10.7
5	256-128-64-32-16	63.7 K	0.9951	0.0049	0.9947	7.2	0.9841	0.0159	0.9789	15.8
6	400-256-128-64-32-16	177.3 K	0.9961	0.0039	0.9958	8.4	0.9837	0.0163	0.9785	10.8
7	500-400-256-128-64-32-16	385.1 K	0.9958	0.0042	0.9954	10.4	0.9842	0.0158	0.9790	24.1
8	600-500-400-256-128-64-32-16	693 K	0.9956	0.0044	0.9953	12.8	0.9842	0.0158	0.9791	24.5
9	700-600-500-400-256-128-64-32-16	1.12 M	0.9956	0.0044	0.9952	15.6	0.9842	0.0158	0.9790	33.1
10	800-700-600-500-400-256-128-64-32-16	1.69 M	0.9967	0.0033	0.9963	19.6	0.9842	0.0158	0.9790	40.2

Table C.9: Evaluation metrics from shallow ANN models trained on increasingly large subsets of the IDS 2017 and IDS 2018 datasets.

Subset size	IDS 2017				IDS 2018			
	Acc.	Prec.	FNR	F1	Acc.	Prec.	FNR	F1
10%	0.9925	0.9910	0.0075	0.9916	0.9832	0.9781	0.0168	0.9779
20%	0.9913	0.9905	0.0087	0.9907	0.9830	0.9780	0.0170	0.9779
30%	0.9942	0.9940	0.0058	0.9938	0.9834	0.9779	0.0166	0.9782
40%	0.9938	0.9936	0.0062	0.9934	0.9831	0.9757	0.0169	0.9779
50%	0.9935	0.9932	0.0065	0.9930	0.9830	0.9776	0.0170	0.9777
60%	0.9951	0.9949	0.0049	0.9947	0.9834	0.9791	0.0166	0.9782
70%	0.9947	0.9941	0.0053	0.9942	0.9831	0.9784	0.0169	0.9779
80%	0.9949	0.9947	0.0051	0.9945	0.9835	0.9792	0.0166	0.9783
90%	0.9952	0.9950	0.0048	0.9948	0.9831	0.9784	0.0169	0.9779
100%	0.9956	0.9955	0.0044	0.9954	0.9825	0.9775	0.0175	0.9771

# Appendix D

## Structural Damage Detection and Localization using Neural Networks on Vibration Signals

### Synopsis

This chapter is based on a technical paper [21] co-authored by the thesis author in the civil engineering journal Structures.

This chapter proposes a novel damage detection and localization method of civil structures using a windowed Long Short-Term Memory (LSTM) network. Sequences of windowed samples are extracted from acceleration responses in a novel data pre-processing pipeline, and an LSTM network is developed to classify the signals into multiple classes. The LSTM network's classification of a signal into a damage level indicates damage presence. Furthermore, multiple structural responses obtained from the vibration sensors placed on a structure are provided as input to the LSTM model, and the resulting predicted class probabilities are used to identify the locations with a high probability of damage. The proposed method is validated on the experimental benchmark data of the Qatar University Grandstand Simulator (QUGS) for binary classification, as well as the Z24 bridge benchmark data for multiclass damage classification associated with different levels of pier settlement and the numbers of ruptured tendons. The results show that the proposed LSTM-based method performs on par with the one dimensional

convolutional neural network (1D CNN) on the QUGS dataset and outperforms 1D CNN on the Z24 bridge dataset. The novelty of this work lies in the use of recurrent neural network-based windowed LSTM for multiclass damage identification and localization using vibration response of the structure.

## D.1 Introduction

### D.1.1 Structural Health Monitoring (SHM)

Next-generation civil infrastructure is designed and constructed using emerging technologies and advanced construction materials to achieve resiliency against natural disasters and operational conditions. However, integrity and condition monitoring of existing aging structures are still lacking in advancements. Over time, structures deteriorate and lose their load-carrying capacity due to various factors such as environmental, heavy traffic, and human-induced damages, requiring cutting-edge technology for continuous structural inspection. Recently, the American Society of Civil Engineers (ASCE) reported the current condition of infrastructure in the United States with a low grade of D+. Canadian infrastructure has been ranked along the same lines; it has been reported that it would take 1.1 trillion dollars to replace the crumbling infrastructure that is in deplorable condition.

The development of efficient strategies for continuous structural monitoring is of paramount importance for aging structures. The conventional approach is to employ a well-trained structural inspector to investigate and inspect the structure, identify defects and implement appropriate maintenance strategies. However, manual condition assessment of the structures is subjective, error-prone and laborious, incurring a significant portion of the annualized maintenance budget. Structural Health Monitoring (SHM) provides a sensor-driven real-time inspection technology to address these challenges of manual visual inspection [15, 5].

SHM employs suitable diagnostic algorithms and assists infrastructure owners and decision-makers in maximizing the safety, serviceability, and functionality of critical structures. Vibration-based SHM has emerged as one of the promising fields for condition monitoring and damage diagnosis of civil infrastructure [3, 7] and offers a viable option for tracking time-varying damages in the structures based on the measured data. Existing damage identification techniques involve analysis in time-domain, frequency-domain, and time–frequency domain [20, 9, 19] along with a combination of machine learning techniques [22, 16].



## D.1.2 Machine learning methods in structural health monitoring

Machine learning techniques provide promising opportunities for detection and localization of damages in civil infrastructure by analyzing various sensor measurements with minimal user intervention, thereby reducing cost and increasing accuracy and reliability of infrastructure management. The focus of this chapter is on structural health monitoring using temporal sensor data.

Authors of [10] proposed sparse coding to extract features from unlabeled acceleration measurements. The damage classification was carried out using CNN, and the results were compared with the traditional machine learning methods, such as logistic regression and decision trees. In [8], the authors conducted a simulation study on a steel gusset plate connection by varying the size and location of the damage. A CNN was used to classify damage, achieving an error of 2% and robustness against environmental noise. Out of various convolutional architectures, one dimensional CNNs (1D CNN) have shown promising results in capturing the temporal information and undertaking damage detection using vibration data.

In the recent years, [1, 2] introduced 1D CNN for nonparametric vibration-based damage detection. The authors trained the neural network on a vibration signal dataset obtained on a 30-joint steel truss structure, named Qatar Grandstand, by damaging each joint and keeping the other joints undamaged. The proposed model was trained individually on each joint, therefore a total of 31 measurement setups were conducted to develop the training database for binary classification. The work of [14] showed the applicability of 1D CNN with autoencoders for anomaly detection under data compression.

The authors of [18] showed the applicability of 1D CNN for damage detection in structural steel frames. Experimental validation was performed on a 2D-steel frame with different damage locations and severity of the damage. In another study, [12] used transmissibility function-based 1D CNN to effectively identify damage in the ASCE SHM benchmark structure and compared the performance against time-domain and frequency-domain methods. 1D CNN primarily exhibited superior performance over multilayer perceptron (MLPs) networks in the context of computation efficiency and noise in temporal data.

While 1D CNN captures relevant information in a neighborhood of samples, it lacks the ability to learn the long-term dependencies of the sequential datasets, which is relevant for structural damage identification over a long period of data. To address this challenge, a Long Short-Term Memory (LSTM) network was recently explored for damage detection in numerical benchmark

data [11], where the performance of LSTM was shown to be better than classical machine learning techniques (random forest, support vector machines). Moreover, the authors of [23] extended the applicability of Long Short-Term Memory (LSTM) networks to detect changes in physical parameters of the structures, such as stiffness and mass. Various structural components such as a beam and steel girder bridge were used for validating the proposed algorithm.

### D.1.3 Contributions and organization

In this work, a multi-window Long Short-Term Memory (LSTM) network is proposed for multiclass structural damage detection and localization. The proposed method captures long term patterns in an acceleration signal by feeding a sequence of windows extracted from the signal as input to the LSTM model, allowing it to make predictions on the acceleration data. This makes the LSTM architecture a valuable technique in vibration-based SHM. This study makes four novel contributions. First, it introduces a standalone LSTM-based approach for damage localization using acceleration measurements. Second, the limited dataset is augmented by windowing the acceleration measurements and a novel approach of voting on the prediction class for windowed-dataset is presented to increase the prediction accuracy. Third, a thorough hyperparameter tuning analysis is conducted and performance is compared with an implementation of the 1D CNN proposed in [2]. Fourth, the proposed method is demonstrated for multiclass and multi-level damage identification in a full-scale bridge study.

The chapter is structured as follows. A brief introduction of the structural damage identification using deep learning techniques is presented in section D.2. Next, section D.3 presents the proposed methodology based on LSTM networks along with the data pipeline and performance metrics to identify and localize damage. The results of both binary and multiclass damage localization are illustrated in section D.4. This section also highlights the importance of hyperparameter tuning and evaluation of optimal window size. Concluding remarks are given in section D.5.

## D.2 Preliminaries of LSTM networks

Given an input acceleration signal,  $x=(x_1, \dots, x_T)$ , a standard recurrent neural network computes hidden vector sequence as  $h=(h_1, \dots, h_T)$  and the output vector sequence as

$y=(y_1, \dots, y_T)$  by iterating Equations D.1 - D.2 from time  $t = 1$  to  $T$ .

$$h_t = \mathcal{H}(W_{ih}x_t + W_{hh}h_{t-1} + b_h) \tag{D.1}$$

where  $W$  denotes weight matrices,  $W_{ih}$  is the input-hidden weight matrix,  $W_{hh}$  is the hidden-hidden weight matrix,  $b$  denotes bias vectors,  $b_h$  is hidden bias vector, and  $\mathcal{H}$  is the hidden layer activation function.

$$y_t = W_{ho}h_t + b_o \tag{D.2}$$

where  $y_t$  denotes output at time  $t$ ,  $W_{ho}$  is the hidden-output weight matrix, and  $b_o$  is the output bias vector.  $\mathcal{H}$  is usually an element-wise application of a sigmoid function. Figure D.1 illustrates a typical single LSTM memory cell (adapted from [6]). The transformations applied to the input  $x_t$  and hidden state from the previous time step  $h_{t-1}$  in the LSTM cell are described by Equations D.3 - D.8 [6].

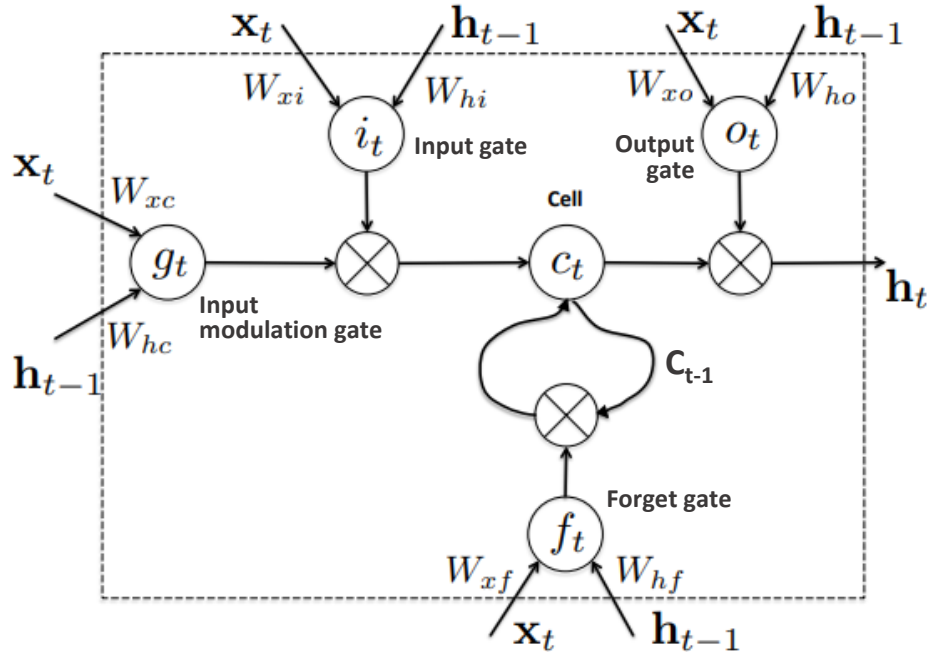


Figure D.1: The typical internal structure of an LSTM cell [6].

$$i_t = \tanh(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{D.3}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (\text{D.4})$$

$$g_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (\text{D.5})$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (\text{D.6})$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes g_t \quad (\text{D.7})$$

$$h_t = \tanh(c_t) \otimes o_t \quad (\text{D.8})$$

where  $\sigma$  is the logistic sigmoid function, and  $i$ ,  $f$ ,  $g$ ,  $o$  and  $c$  are respectively the input gate, forget gate, input modulation gate, output gate and cell activation vectors, all of which are the same size as the hidden vector  $h$ . The weight matrix subscripts have the same meaning, for example  $W_{hi}$  is the hidden-input gate matrix,  $W_{xo}$  is the input-output gate matrix. The bias terms (which are added to  $i$ ,  $f$ ,  $g$ , and  $o$ ) have been omitted in Figure D.1 for clarity. The weight matrices represent the learnable parameters of the model, and they are learned by the gradient decent algorithm to minimize prediction error on a training set.

The cell state  $c_t$  encodes the information of the sequence observed up to that time step. The input gate controls the information added to the cell state from the current time step, and the forget gate controls what information needs to be forgotten from the current cell state. For example, if the output vector of the forget gate  $f_t$  has a near-zero value in the first dimension, it indicates that the first dimension of the cell state  $c_t$  needs to be “forgotten”. The forgetting occurs in the element-wise multiplication; i.e. multiplying an element by a near-zero value results in a near-zero element in the output vector. By maintaining cell state in this manner, the LSTM cell is able to capture both long and short term relationships between the input time-series values and the predicted variable (eg: damage classification).

During training, the truncated back propagation through time (TBPTT) algorithm makes the process computationally feasible. During prediction, the forward pass can be applied to arbi-

trarily long sequences (the LSTM cell can be repeatedly applied to any number of input time steps). More details of the internal structure and training mechanism of LSTMs can be found in [6].

## D.3 Damage detection using LSTM model

### D.3.1 Data pipeline

Figure D.2 illustrates the proposed data pipeline, which consists of a series of pre-processing and post-processing steps with an LSTM network as the classification model. It should be noted that the solid black lines represent datasets, blue dotted lines represent operation on the datasets, and multiple arrows represent multiple data instances.

A single acceleration time-series consists of a large number of samples (a lengthy record of vibration data). This time signal is first normalized (scaled) with respect to its mean and standard deviation. This improves the convergence rate of models trained on the datasets and prevents outliers from dominating the input. Second, windows of the scaled time series (window size  $w$ ) are created, and a sequence of such windows (length  $L$ ) is arranged to form one input instance to the LSTM network. Thus, the input of the network is a  $w$ -dimensional sequence of length  $L$ . Many such sequences can be extracted from a single original accelerometer time series, and each sequence is assigned to a label (damage level) of the original series. The process of extracting sequences of windows from a time series is illustrated in Figure D.3.

This technique of transforming the original series into sequences of windows effectively reduces the data dimension, and additionally, it increases the training set size (multiple sequences per time-series), which in turn allows training deep learning models with less over-fitting. Imbalanced data can cause problems in model training. To alleviate this problem, a balanced training set is created by randomly selecting a number of undamaged sequences equal to the number of damaged sequences.

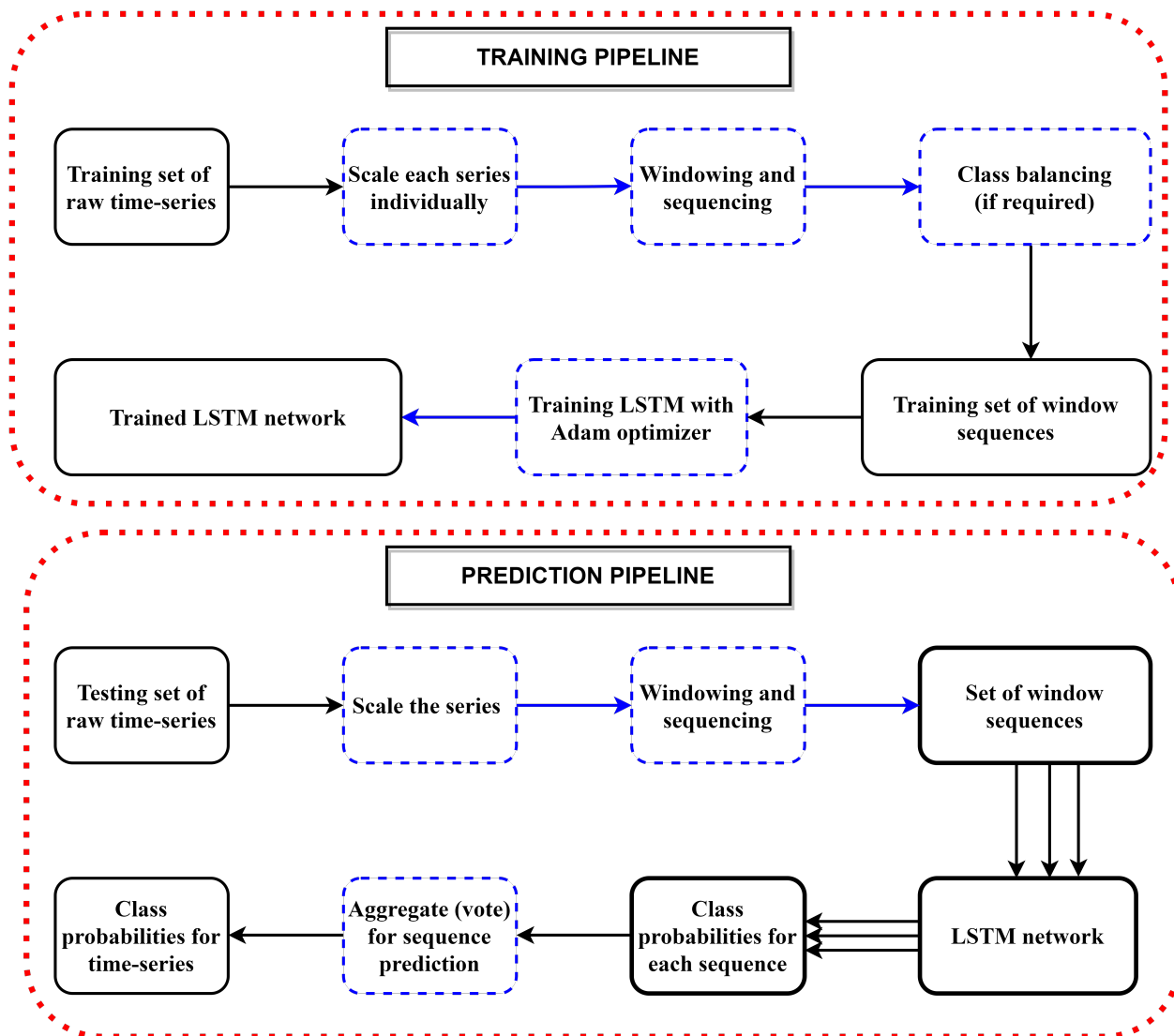


Figure D.2: Data pipelines for training the LSTM network and obtaining predictions for a given acceleration response.

### D.3.2 The model, training and inference

The proposed machine learning model for the damage time series level classification is a multi-layer LSTM network architecture, as shown in Figure D.3. A pre-processed sequence of windows are given as input to the model, and the softmax output of the final LSTM time step is considered as the prediction of the sequence (the set of classification probabilities  $P(y = c_t)$  for each class  $c_t$ ). The classification problem is formulated as binary (undamaged vs. damaged) or multiclass (undamaged, and damage of more than two levels) depending on the dataset.

During training, forward and backward passes are performed on the input sequences, and the

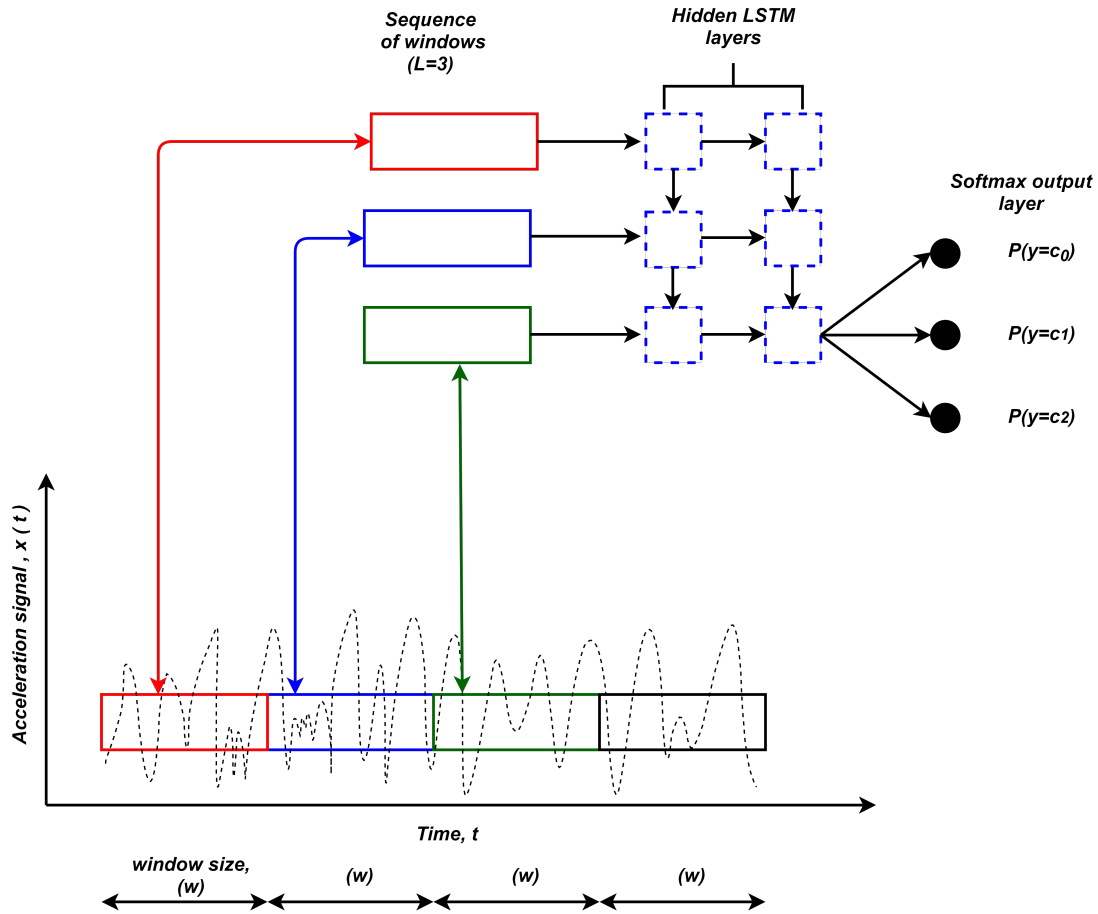


Figure D.3: Extracting sequences of windows from the vibration signals and the LSTM network architecture.

weight updates are made to minimize the cross-entropy loss on a batch of sequences. The predicted set of classification probabilities  $P_p(y_c)$  for a full acceleration measurement is obtained by summing the class probabilities of all the window sequences in one vibration signal. The class with the maximum probability is the predicted damage level classification of the series. Note that this is equivalent to voting on the classification probabilities of individual window sequences to arrive at the prediction of the full series. It is observed that the voting process improves the prediction accuracy and other evaluation metrics on the time-series test set.

### D.3.3 Hyperparameter tuning

The window size  $w$  and the sequence length  $L$  (number of windows in a sequence) are hyperparameters that are tuned to improve the accuracy of the neural network. The hyperparameters of

LSTM network include number of layers and number of nodes in each layer (network architecture), activation function, and batch size (performing weight updates during training). Optimal parameters are found using a random search on a predefined hyperparameter space [4]. In each iteration, the search algorithm randomly selects a configuration of values for hyperparameters from a specified set of possible values, and trains three models with those parameters on three splits of training and validation sets (3-fold validation) with early stopping used to minimize overfitting.

### D.3.4 Damage localization using LSTM model

The LSTM network described in the previous section produces a set of probabilities  $P_p(y_c)$  for a given acceleration signal. These indicate the probability that the signal belongs to each class  $c$ . Multiple signals are obtained from accelerometers placed in the critical locations of a structure, and model prediction probabilities are computed for each signal. This method estimates a damage probability distribution across the structure and identifies high-probability damage locations. The damage probability distribution over the structure can be visualized (for example, heatmaps of probability values over a 2D structure) to aid an engineer in quickly localizing damages. Algorithm 4 summarizes the proposed approach for damage detection and localization.

### D.3.5 Performance evaluation criteria

Several standard metrics can be used to evaluate the performance of a classification model. These metrics measure different aspects of the obtained results. A brief description of the selected metrics is provided below and explained in the context of structural health monitoring of civil infrastructure.

**Confusion matrix.** The primary form of prediction results is given by the confusion matrix, which is a tabulation of classifications made by a model. It shows the “classification distribution” of a model, and helps identify properties of the model, such as when it is consistently mis-classifying one class as another. The confusion matrix is obtained for both binary and multi-class classifications. Table D.1 shows the confusion matrix for the case of binary classification. True Positives are denoted as TP, True Negatives as TN, False Positives as FP, and False Negatives as FN.



**Algorithm 4** Damage detection and localization**Input:**

A set of acceleration signals from a structure

**Outputs:**

Predicted damage class (damage detection)

Probabilities of damage over the structure (damage localization)

- 1: The acceleration signal data is pre-processed, as shown in the training pipeline in Figure D.2.
- 2: A single LSTM model is trained for damage classification.
- 3: The predicted class label (undamaged or damage level) for a signal is computed from the model (damage detection for the signal).
- 4: Damage probabilities for all signals form a distribution of damage probabilities over the structure. High probabilities correspond to damaged locations (damage localization).
- 5: Visualize (heatmaps) the distribution of damage probabilities for visual inspection and automated decision making.

Table D.1: Confusion matrix for a binary damage classification problem.

		Predicted class	
		Damage	Healthy
Output class	Damage	TP	FN
	Healthy	FP	TN

**Evaluation metrics.** Multiple metrics can be derived from the confusion matrix as shown in Table D.2. In this study, two key metrics are used as the performance metrics to evaluate the proposed method, namely, accuracy, and false negative rate (FNR). Accuracy is the primary evaluation metric to understand the ability of the model to correctly classify the inputs. A false negative (type II error) represents a truly damaged signal that is classified as undamaged by the model, which could lead to catastrophic consequences in a critical structure. Therefore, the FNR is measured and compared in the model evaluation experiments in this work.

**Performance curves.** Two plots highlight the trade-off between metrics as the decision threshold of the classifier changes: the Receiver Operating Characteristic (ROC) curve shows the trade-off between false positive rate (FPR) and true positive rate (TPR), and the precision-recall (PR) curve shows the trade-off between precision and recall. Analyzing these curves

Table D.2: Description of various performance metrics.

Metric	Formula	Remarks
Accuracy	$\frac{TP+TN}{TP+FN+FP+TN}$	Less useful for heavily imbalanced data
Precision	$\frac{TP}{TP+FP}$	Positive predicted value
Recall	$\frac{TP}{TP+FN}$	True positive rate or sensitivity
False Positive Rate (FPR)	$\frac{FP}{TN+FP}$	False alarm when there is no damage
False Negative Rate (FNR)	$\frac{FN}{TP+FN}$	No alarm for actual damage
F1 Score	$2 * \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$	The harmonic mean of precision and recall
ROC-AUC	Recall vs. FPR	Degree of separability between classes

from model predictions enables engineers to make informed decisions on the balance that is needed between the metrics and make a decision on thresholds suitable for the task. The PR curve, in particular, is suitable for the evaluating the cases where the datasets are highly imbalanced. In this study, ROC and PR curves are plotted for visual comparison. The corresponding area-under-the-curve (AUC) of these graphs (ROC-AUC and PR-AUC) represent an aggregate measure of the model's ability in terms of the relevant metrics of the curves.

**Evaluating damage localization capability.** The damage localization results are evaluated by inspecting the visualization (heatmaps) of damage probabilities over the structure, and verifying that high damage probabilities have been assigned to the damaged location, while the other locations are assigned with low damage probabilities.

## D.4 Experiments and results

The proposed method is evaluated in two setups: an experimental setup (physical simulation of a grandstand structure), and a full-scale study (real-world data from a bridge). For comparison, the 1D CNN model from [2] is also implemented and evaluated. For performance evaluation, accuracy, FNR, ROC-AUC and PR-AUC metrics are used. Additionally, several experiments are conducted to study the performance change with window size in the model input, and the

effect of voting on individual window predictions to obtain the final prediction.

## D.4.1 Experimental study

### A brief description of the data

In this study, the experimental benchmark data of Qatar University Grandstand Simulator (QUGS) [2] is used to evaluate the performance of the proposed method. The QUGS was constructed to evaluate and develop effective structural damage detection techniques suitable for monitoring of modern stadia, as shown in Figure D.4. The frame was designed to carry a total of 30 spectators with area dimensions of 4.2 m x 4.2 m. The design considerations used for the experimental test structure was to guarantee its safety and compatibility with the specifications of modern grandstands.

The structure was utilized in its current form (steel frame only) to generate vibration data under several structural damage scenarios. The grandstand was excited using a shaker and two 16-channel data acquisition systems were used to acquire the acceleration responses. A total of 31 scenarios are considered: the first scenario corresponds to the reference (undamaged) case, while in scenarios 2 to 31, damage was introduced by loosening the bolts at the joints 1 to 30, respectively. A total of 30 accelerometers were mounted at the joint of each girder and the filler beam, and a magnetic mounting plate was used to attach the sensors on the frame. The acceleration data was sampled at 1024 Hz and collected for 256 s under a band-limited white noise excitation from the shaker with a frequency range of 0–512 Hz.

### Hyperparameter tuning

A range of values for hyperparameters is used for tuning the LSTM model using a random search, as shown in Table D.3. The window size as an external parameter is varied between 64 and 512 samples. Various values for other hyperparameters such as the number of hidden layers (range of 1-6), and nodes in hidden layers (range of 32-1024) are considered to achieve optimal performance of the proposed model. Consequently, the random search algorithm explores shallow, wide and deep LSTM architectures. All models are trained using the Adam optimizer with the ‘Rectified Linear Unit (ReLU)’ activation function in hidden layer nodes, ‘softmax’ activation in the output layer, and the ‘cross-entropy’ loss function.

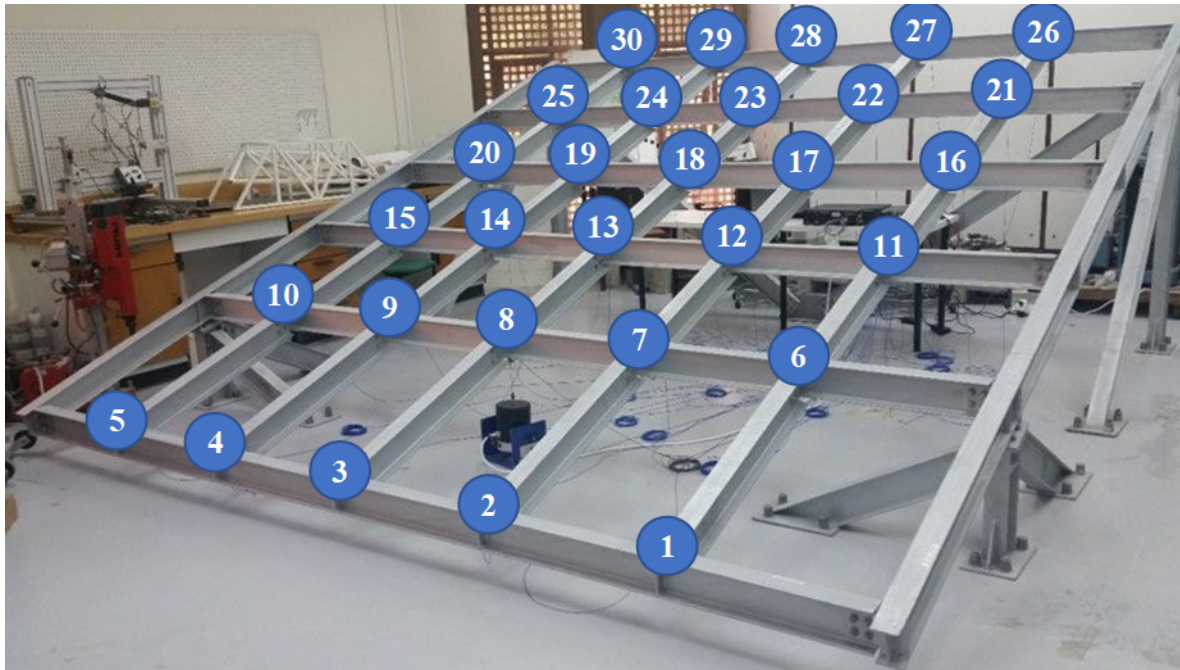


Figure D.4: QUGS testbed [2]. Joints are numbered from 1 to 30

The optimal hyperparameter configuration obtained using random search algorithm for QUGS experiment is presented in Table D.4. It can be observed that the highest performing LSTM architecture has three hidden layers (not very wide), with a window size of 64 (input dimensionality).

The change in damage detection performance with window-size ( $w$ ) is shown in Figure D.5. Three different metrics are used, namely, ROC-AUC, accuracy, and FNR as these three metrics broadly cover the efficacy and any shortcomings of the classification model. It can be observed that  $w=64$  and  $w=128$  yield the best performance metric values (high accuracy and ROC-AUC with low FNR), and performance metrics consistently degrade with increasing window size. This behavior is attributed to a reduction in the data samples (sequences of windows) with increasing  $w$ , causing the LSTM network to overfit when trained on smaller datasets.

### Performance comparison

To compare the proposed LSTM approach with the 1D CNN method introduced by [2], a 1D CNN is trained with the same hyperparameter tuning process as outlined before. The optimal 1D CNN network has 4 hidden layers with nodes 256, 128, 64, 32 in each layer and a kernel size of 64. After acquiring the optimal tuned parameters, a parametric study is conducted to

Table D.3: Hyperparameters used in LSTM for tuning by random search algorithm.

Parameter	Values
Window size	64, 128, 160, 256, 512
No. of windows in a sequence	2, 4, 8, 16
No. of hidden layers	1 - 6
No. of hidden nodes	1024, 512, 256, 128, 64, 32
Dropout rates	0.2 and 0.5 for each hidden layer
Learning rate	0.0003, 0.001, 0.01
Batch size	64, 256, 512
Cell type	LSTM with tanh activation

Table D.4: Optimal configuration of LSTM hyperparameters for QUGS experiment.

Parameter	Values
Window size	64
No. of windows in a sequence	8
No. of hidden layers	3
Architecture	[64, 128, 64, 32, 1]
Dropout rates	0.2
Learning rate	0.001
Batch size	256
Training epochs	100 with early stopping

understand variance in the metrics by training the LSTM and 1D CNN models with random initialization of network weights of 5 times. The result is shown in Figure D.6. It is found that both models perform well consistently with accuracy at 1.0 and FNR at 0.0 with negligible

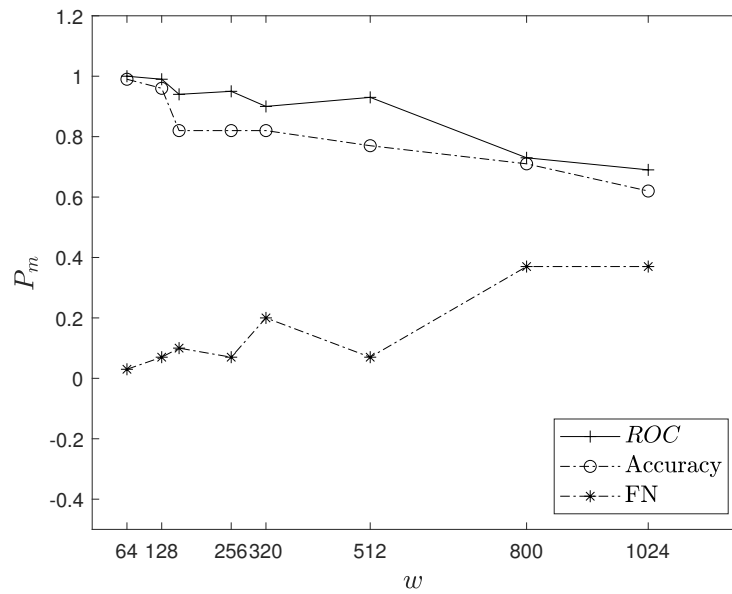


Figure D.5: Performance evaluation of LSTM based on window size for QUGS.

variability.

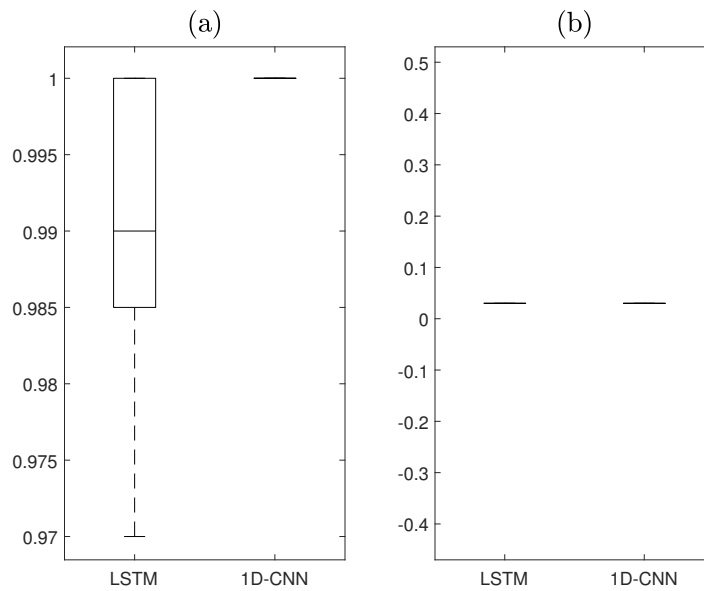


Figure D.6: Performance evaluation of 5 random weight initializations for QUGS (a) Accuracy, and (b) FNR.

One possible explanation for the perfect performance behavior of the machine learning models is based on the nature of the acceleration signals. The damages in the joints of the QUGS are highly localized, and the sensors are placed at the exact locations of damage. Furthermore,

the controlled experimental laboratory setup makes the acquired signals of high quality. These properties lead to acceleration signals that have distinct, discriminating patterns between the damaged versus undamaged cases. The 1D CNN's capability to learn local structure of the time signals, and the LSTM's capability to learn long-term irregular dependencies lead to excellent performance of both models.

### Effect of majority voting on classification performance

In the proposed method, the damage classification of a time-series is obtained by voting on the classification of individual windows that constitute the entire time-series. It is observed that this voting process increases performance metrics on time-series predictions considerably in contrast to the predictions on individual windows, as illustrated by the ROC and precision-recall (PR) curves in Figure D.7.

The ROC curve is closer to the upper left corner in the voted series predictions (Figure D.7 (b)) than on the individual window predictions (Figure D.7 (a)), which represent an increase in ROC-AUC from 0.95 to 1.0. The PR curve also shifts to the upper right corner as shown in Figure D.7 (d), with an increase in PR-AUC from 0.52 for voted series predictions to 0.99 for individual window predictions. It can be concluded that voting on windows decreases the probability of error significantly to obtain the final prediction.

### Damage localization

Damage localization is performed for the QUGS setup using Algorithm 4. In each damage scenario in the QUGS dataset, a single joint out of the 30 joints in the grandstand structure is damaged by loosening of the bolts. Acceleration signals are acquired from all joints, and the proposed localization method gives damage probabilities for each joint location.

The distributions of damage probabilities for multiple damage scenarios are presented as heatmaps in Figure D.8. Note that the color blue indicates  $P(\text{Damage}) \approx 0$  and color red  $P(\text{Damage}) \approx 1$ . For example, Figure D.8 (a) shows the scenario where joint-1 of the structure is damaged (see Figure D.4 for an illustration for numbered joints). The damage appears to be heavily localized at the damaged joint (correct localization). Figure D.8 (e) shows the scenario where joint-23 [location: (3, 5)] is damaged, and it is clear that the damage is spread out as it is surrounded by

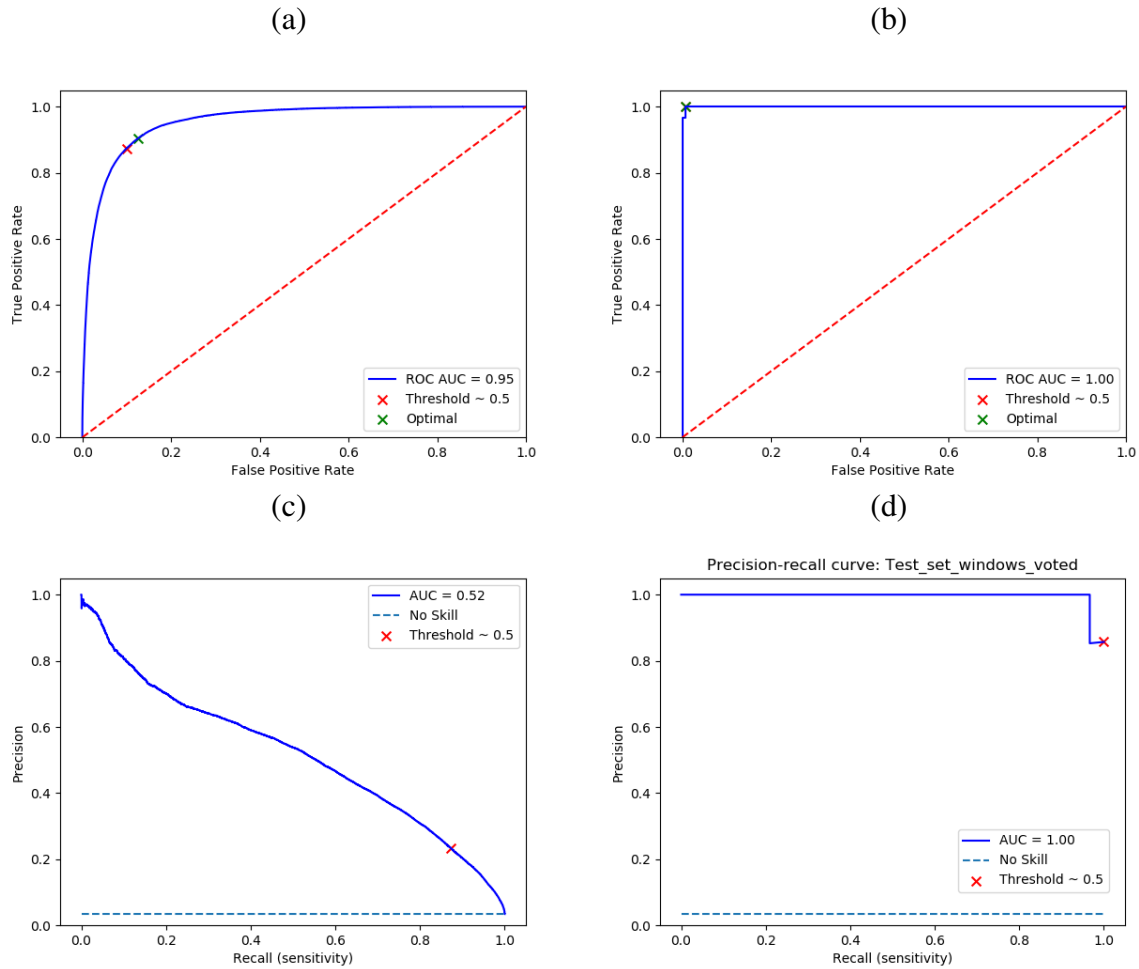


Figure D.7: Performance of the LSTM model on the QUGS data: (a) ROC on individual windows, (b) ROC with voting, (c) PR on individual windows, (d) PR with voting.

4 connected branches signifying more effect of damage after loosening of bolts.

## D.4.2 Full-scale study

### A brief description of the data

Z24 bridge benchmark data [13] is used to evaluate the performance of the proposed method for multiclass damage detection. In this study, two classes of damage are used, namely, pier settlement and rupture of tendons. Both damage classes further have multiple damage levels, hence this is a multiclass classification problem. The bridge was a classical post-tensioned concrete box-girder bridge with a main span of 30 m and two side spans of 14 m, as shown in



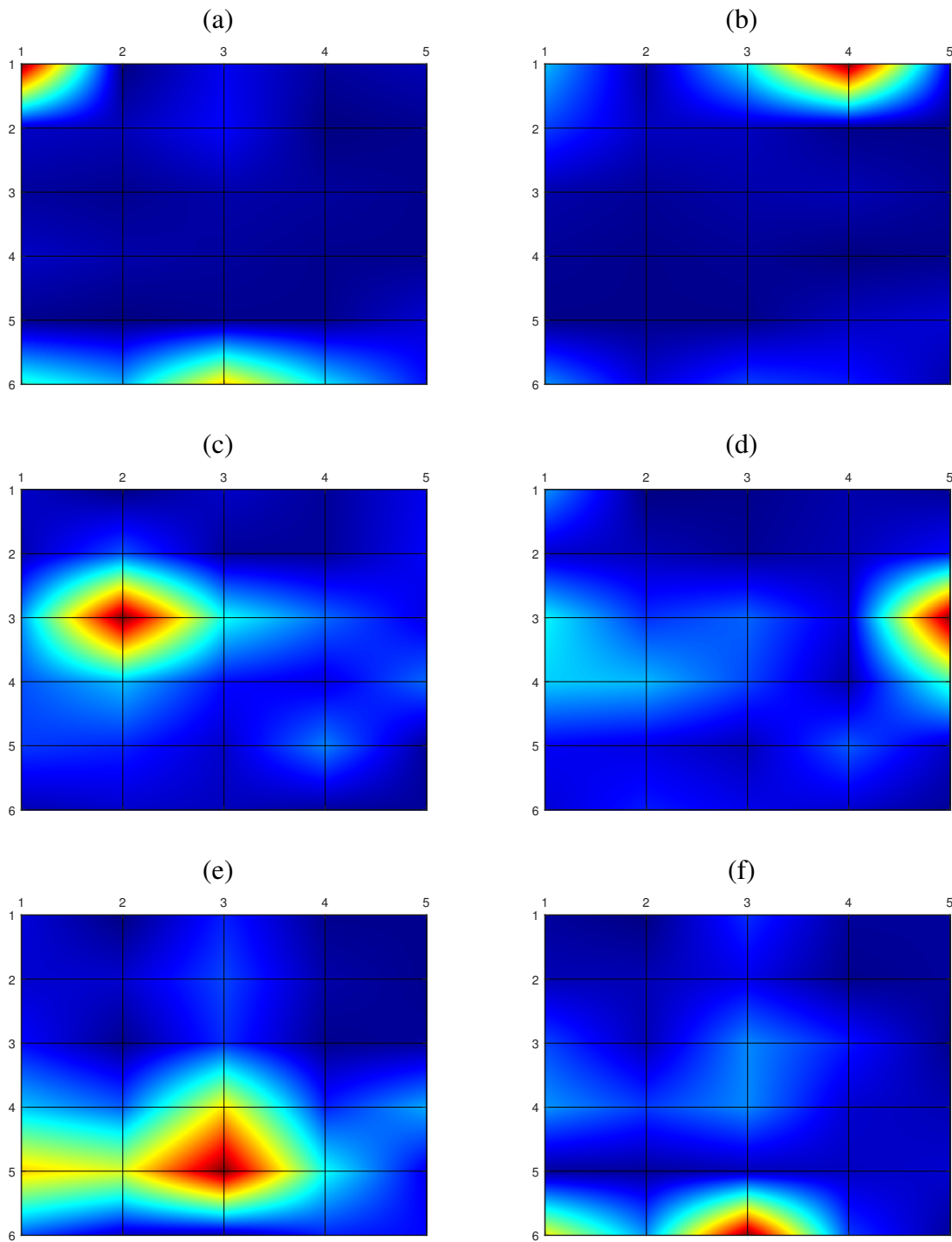


Figure D.8: Damage localization probabilities in QUGS for scenarios where the damaged joint is: (a) Joint-1 [1, 1] (b) Joint-4 [4, 1], (c) Joint-12 [2,3], (d) Joint-15 [5,3], (e) Joint-23 [3,5], and (f) Joint-28 [3, 6].

Figure D.9. The bridge was demolished at the end of 1998 because a new railway adjacent to

the highway required a new bridge with a larger side span.

The SHM data was acquired using 16 accelerometers placed at different spans of the bridge, as shown in Figure D.10. The bridge was excited by two shakers, one at the mid-span of the bridge and another at a side-span. Because of the size of the bridge, response was measured in nine setups with each setup containing up to 15 sensors each. In all setups, three accelerometers and two force sensors were used. The data was sampled at 100 Hz, and a total of 65,536 samples were acquired. This data was made publicly available by researchers at the Katholieke Universiteit Leuven and is available at: <https://bwk.kuleuven.be/bwm/z24>.

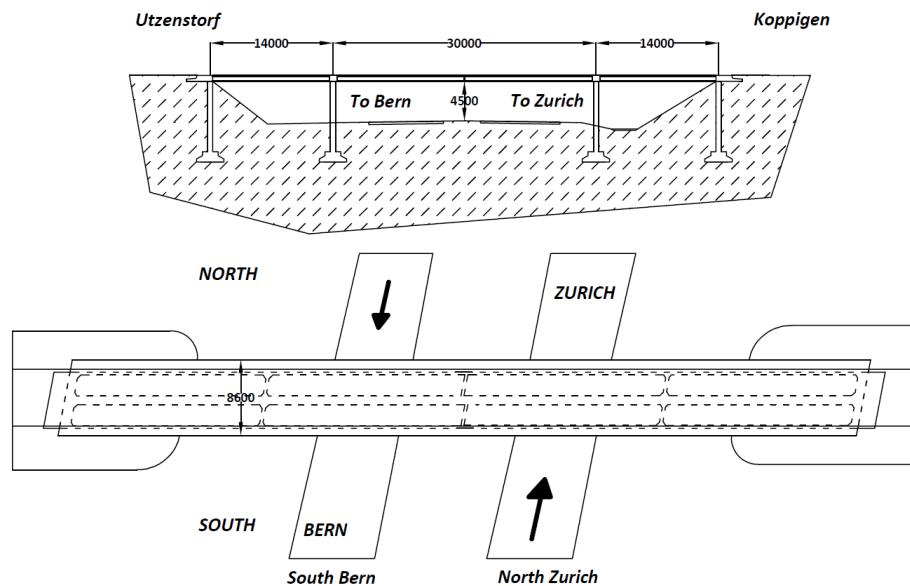


Figure D.9: Schematic of the Z24 bridge.

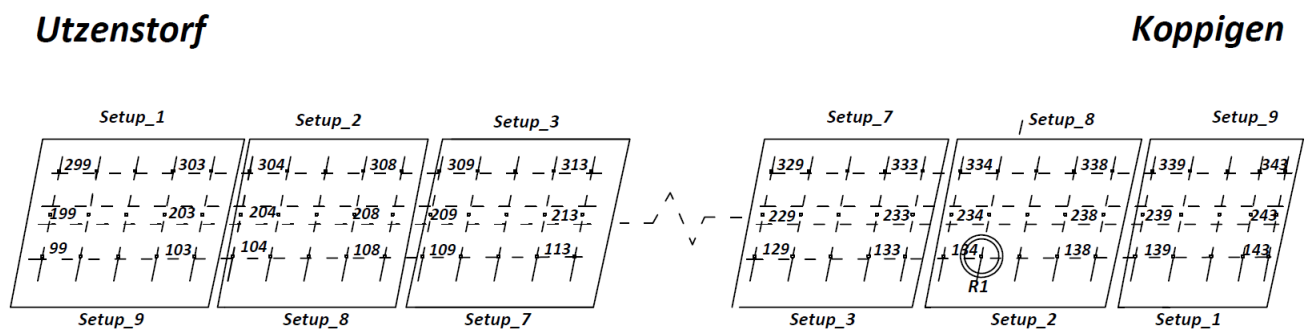


Figure D.10: Sensor placement for the data acquisition.

Several progressive damage scenarios were considered for vibration-based damage identification. Each damage scenario contains multiple levels of damage. All these damage scenarios are compared with the baseline undamaged state. For example, the rupture of tendons scenario has three damage levels and the lowering of pier scenario has four damage levels. For detailed

explanation of how the damages were induced to the bridge, readers are referred to [17]. The reference undamaged condition is considered as class-zero in all scenarios and the other damages were assigned classes starting from 1 to n depending upon the level of damage, as shown in Table D.5.

Table D.5: Multiclass classification problem description for two damage scenarios.

<b>Problem</b>	<b>Damage scenario</b>	<b>Class label</b>
1	Undamaged	0
	Rupture of 2 tendons	1
	Rupture of 4 tendons	2
	Rupture of 6 tendons	3
2	Undamaged	0
	Lowering of pier, 20 mm	1
	Lowering of pier, 40 mm	2
	Lowering of pier, 80 mm	3
	Lowering of pier, 95 mm	4

### Hyperparameter tuning

Optimal hyperparameters of the LSTM model on the Z24 bridge dataset are obtained by performing a random search. Table D.6 shows the hyperparameter configuration with highest accuracy. It can be observed that LSTM performed well with  $w = 128$ . An analysis is performed to understand the effect of window size  $w$  on performance. The results are illustrated in Figure D.11, which shows that optimal performance is achieved at  $w = 128$ , with highest ROC-AUC and accuracy, and lowest FNR. In the case of Figure D.11 (a), the ROC-AUC and accuracy reduce while FNR increases. Similarly, in the case of Figure D.11 (b), the FNR remains consistent after  $w = 512$  and other metrics are at their peak. Due to larger  $w$ , the data size reduces per damage class and it leads to overfitting of the model on the data.

A random search of hyperparameters is also conducted to find optimal parameters for the 1D

CNN model.

Table D.6: Optimal configuration of LSTM hyperparameters for the Z24 bridge dataset.

Parameter	Values
Window size	128
No. of windows in a sequence	16
No. of hidden layers	3
Architecture	[128, 64, 32, 5]
Learning rate	0.001
Batch size	512
Training epochs	100 with early stopping

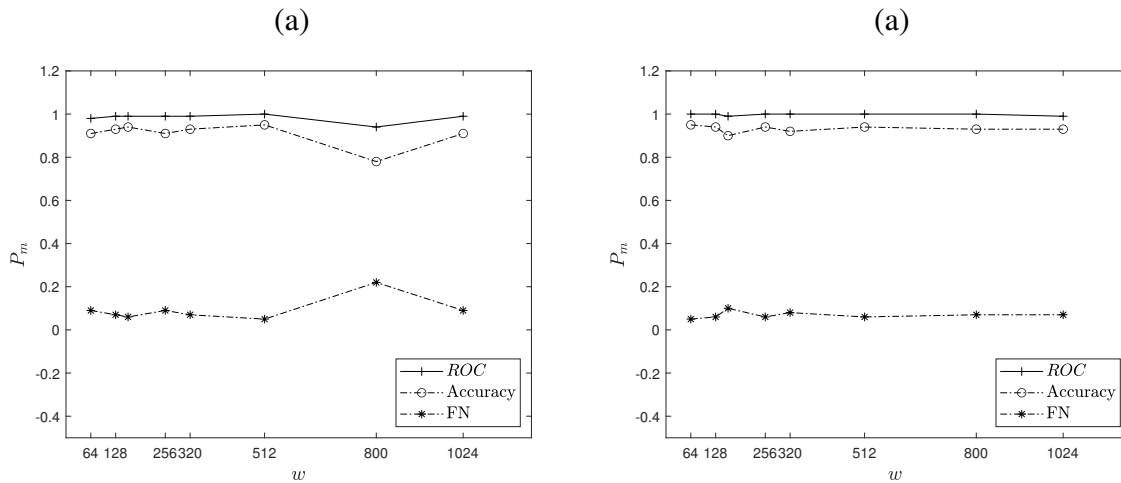


Figure D.11: Performance evaluation of LSTM based on window size for (a) rupture of tendons, and (b) pier settlement.

### Performance comparison

Five repetitions of training and testing sessions are performed to verify the robustness of the models against random initialization of network weights. Accuracy and FNR are computed for each session, and the results are shown in Figure D.12.

It can be observed that for pier settlement which was measured using 4 sensors for data acquisition, and associated with 4 damage levels, the damage is considered as localized around these sensors. The LSTM performs well with highest accuracy and lowest FNR in comparison to 1D CNN as seen in Figure D.12 (a), and (b). In case of rupture of tendons, where the damage is fairly distributed throughout the bridge deck, the damaged signals are not highly distinguishable in comparison to undamaged signal and other severe damage measurements such as lowering of piers. The accuracy of LSTM drops to 0.9 and FNR increases to 0.2. However, LSTM still performs better than 1D CNN as shown in Figure D.12 (c), and (d). The superior performance of LSTM can be attributed to its capability to learn long-term irregular dependencies of complex time signals whereas 1D CNN learns prominently the local neighborhood structure of the signals.

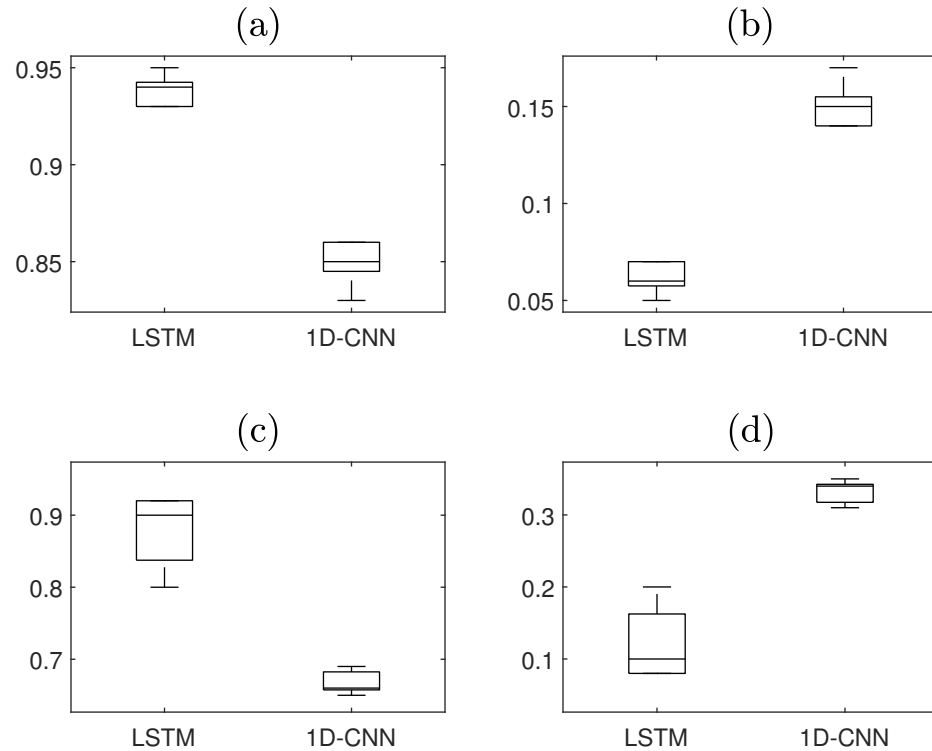


Figure D.12: Performance evaluation of LSTM for random weight initialization for various damage cases in the Z24 bridge where, (a) accuracy for pier settlement, (b) FNR for pier settlement, (c) accuracy for rupture of tendons, (d) FNR for rupture of tendons.

Figure D.13 shows the confusion matrices for the pier settlement damage classification problem with five classes, starting from no damage (class-0) to 95 mm damage (class 4), using LSTM and 1D CNN, respectively. It can be seen that the LSTM model accurately classifies a large

percentage of signals in each class (i.e., the diagonal of the confusion matrix), demonstrating high precision by the LSTM model. In comparison, the 1D CNN fails to differentiate between the undamaged case and damage level 1, as well as, between levels 2 and 3, which can lead to significant inaccuracy in the bridge maintenance.

		<b>LSTM</b>				
		Predicted Class				
		Pred-0	Pred-1	Pred-2	Pred-3	Pred-4
True Class	True-0	92	1	0	0	3
	True-1	3	87	2	2	3
	True-2	0	4	92	0	1
	True-3	0	0	3	93	1
	True-4	0	0	0	0	97

		<b>1D-CNN</b>				
		Predicted Class				
		Pred-0	Pred-1	Pred-2	Pred-3	Pred-4
True Class	True-0	76	11	5	0	4
	True-1	19	66	7	0	5
	True-2	0	0	96	0	1
	True-3	0	0	22	70	5
	True-4	0	0	2	1	94

Figure D.13: Confusion matrices for the pier settlement damage problem using LSTM (top) and 1D CNN (bottom)

Figure D.14 shows the confusion matrices for the rupture of tendons damage classification problem with four classes, starting from no rupture (class-0) to rupture of 6 tendons (class 3), using LSTM and 1D CNN, respectively. The LSTM model classifies a large percentage of signals in each class correctly (e.g., the diagonal of the matrix). This results in high precision by the LSTM model. In comparison, the 1D CNN model fails to differentiate between damage levels 1, 2 and 3. A considerable percentage of signals in damage level 2 are classified into levels 1 and 3.

**LSTM**

		Predicted Class			
		Pred-0	Pred-1	Pred-2	Pred-3
True Class	True-0	96	0	0	0
	True-1	1	95	0	1
	True-2	2	5	84	6
	True-3	7	2	6	82

**1D-CNN**

		Predicted Class			
		Pred-0	Pred-1	Pred-2	Pred-3
True Class	True-0	93	0	0	3
	True-1	0	83	11	3
	True-2	1	19	14	63
	True-3	1	9	20	67

Figure D.14: Confusion matrices for the rupture of tendon damage problem using 1D CNN (top) and 1D CNN (bottom)

**Effect of majority voting on classification performance**

Finally, the optimal parameters are used to evaluate the performance of proposed method on full-series (as one input instance) versus voted-windowed samples. It is observed that voting on windowed dataset increases accuracy considerably and it is evident in ROC and PR curves, as presented in Figure D.15 and Figure D.16 respectively. However, due to the localized measurement acquisition and severity of damage in pier settlement, the difference in AUCs of various cases was comparatively similar to the QUGS damage scenario. Moreover, as observed in Figure D.16, where the damage was considerably distributed in case of rupture of tendons, voting on windows increased the ROC-AUC by 5% and PR-AUC by 13%.

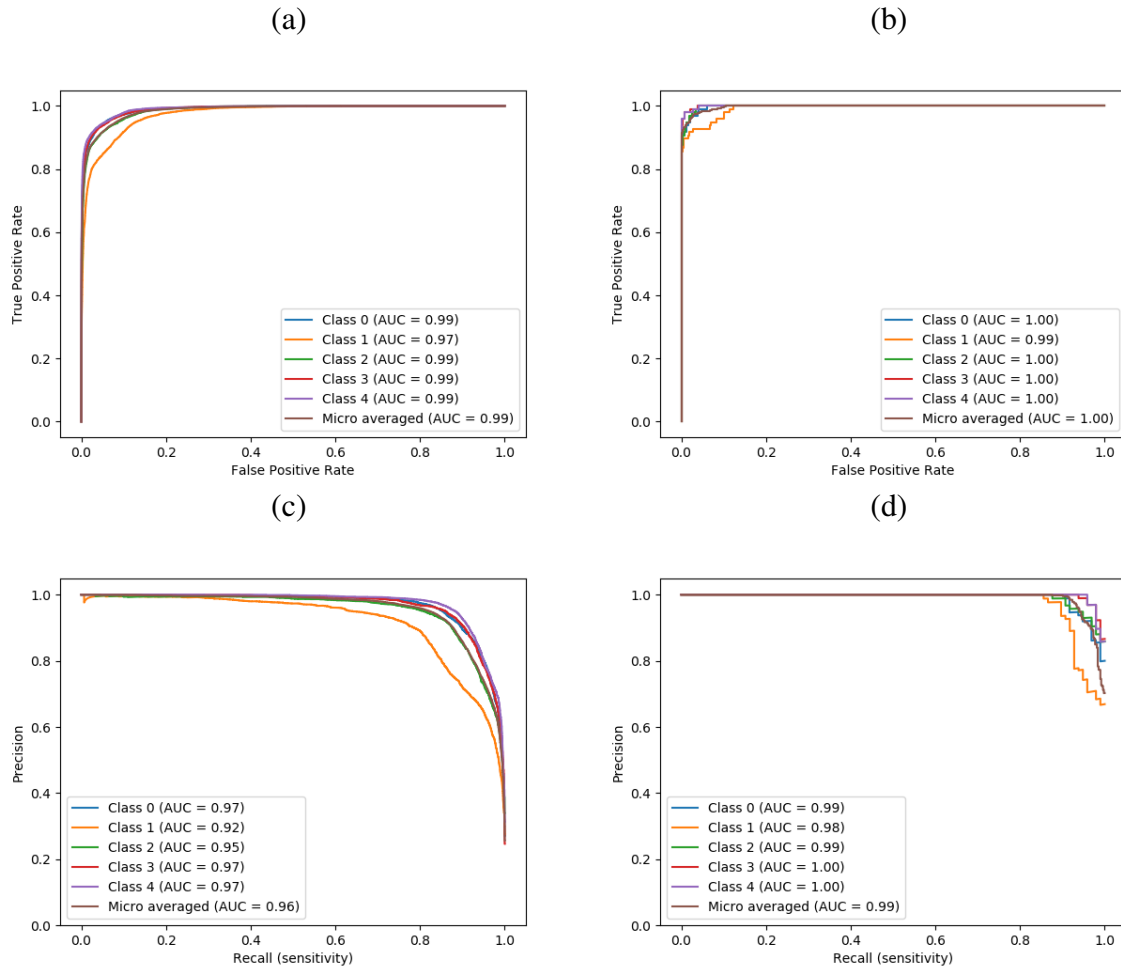


Figure D.15: Performance of the proposed LSTM model in the Z24 bridge for pier settlement: (a) ROC in individual windows, (b) ROC in voted series, (c) PR in individual windows, (d) PR in voted series.

## Damage localization

Damage localization is performed using Algorithm 1 for multiclass damage scenarios of lowering of pier and rupture of tendons. In these damage scenarios, the damage is not highly localized as in the experimental study of subsection D.4.1. Three different structural components of the bridge are used to localize damage and understand the effect of pier settlement. An undamaged pier in Utzenstorf, bridge deck, and damaged pier in Koppigen are used to represent the predicted probability ( $P_p$ ) and infer damages in three components. The Koppigen pier is used for inducing the damage by lowering it in several increments starting with 20 mm, 40 mm, 80 mm, and moving to 95 mm at the last stage. The predicted probability  $P_p$  is plotted against the sensor number and a dash-dotted average of  $P_p$  of structural component is



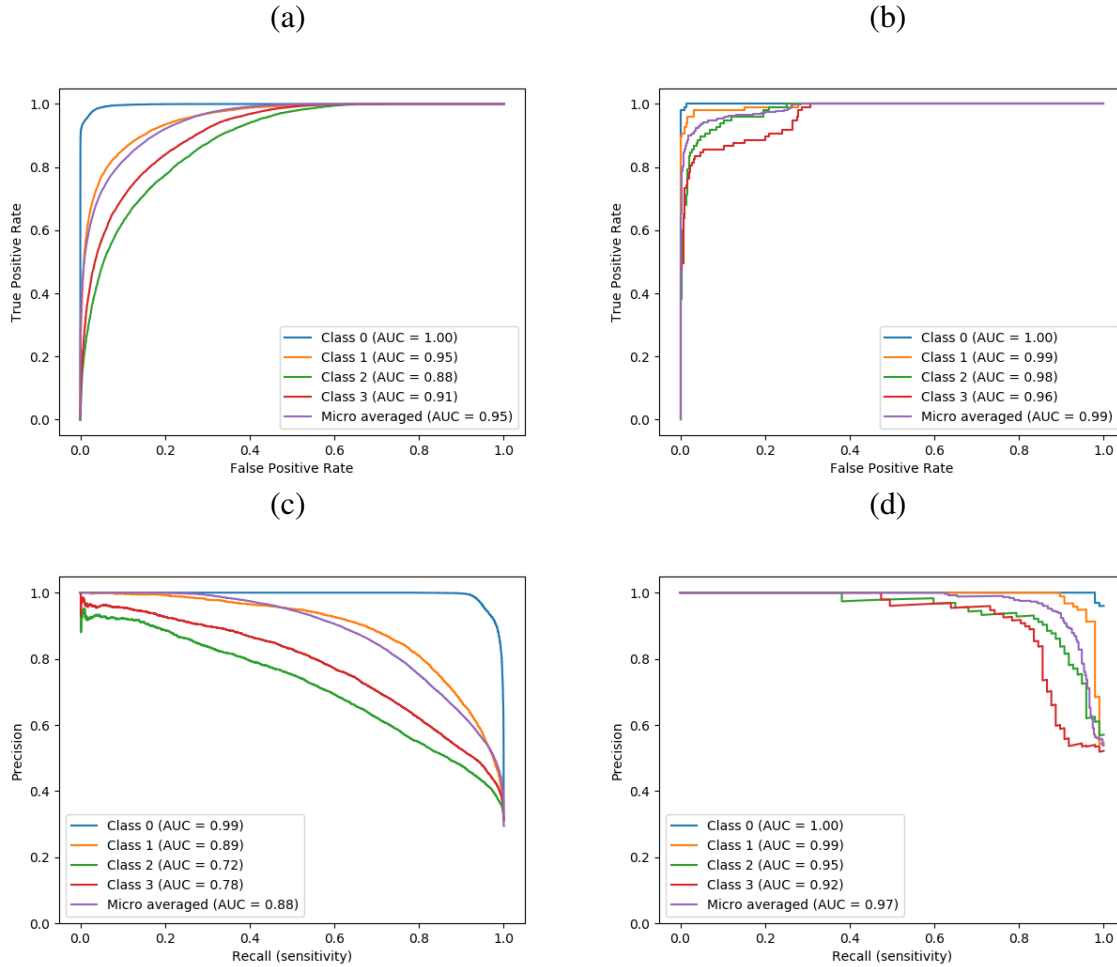


Figure D.16: Performance of the proposed LSTM model by windowing the data of the Z24 bridge in case of rupture of tendons: (a) ROC in individual windows, (b) ROC in voted series, (c) PR in individual windows, (d) PR in voted series.

shown as a representation of combined  $P_p$  for corresponding structural component, as shown in Figure D.17. For example, Figure D.17 (a, b, c) represents  $P_p$  for undamaged pier (UDP), bridge deck (BD), and damaged pier (DP) for 20 mm lowering of piers, respectively. Similarly, Figure D.17 (d, e, f) and (g, h, i) show  $P_p$  for 40 mm and 95 mm lowering of piers, respectively.

Due to non-localization of measurement acquisition, it is difficult to infer damage location while considering each sensor separately. However, it is possible to compare average of each structural component for various damage cases. The results considering average  $P_p$  for each structural component and various damage levels are shown in Figure D.18. Despite the lack of correlation between  $P_p$  and damage severity, more severe damages yield more distinguishable signals, enhancing the LSTM model's classification effectiveness. It can be observed from Figure D.18 that UDP shows lowest predicted probability due to its similarity to undamaged

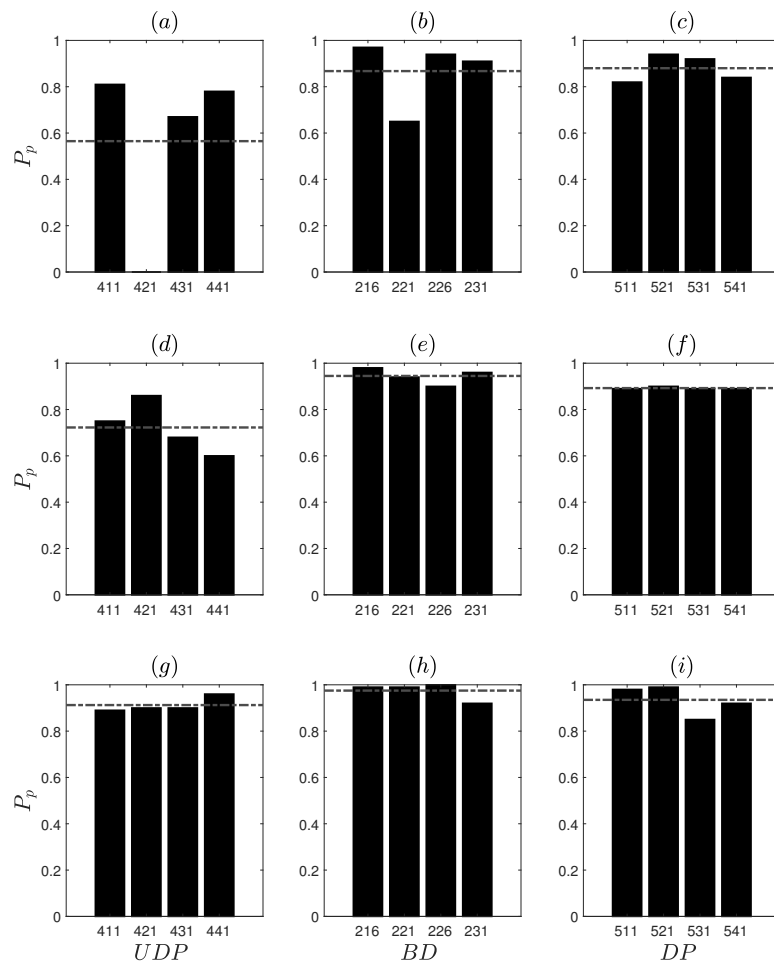


Figure D.17: Damage localization for lowering of pier for three damage levels, where, (a, b, c) are for 20 mm lowering of piers, (d, e, f) are for 40 mm lowering of piers, (g, h, i) are for 95 mm lowering of piers.

baseline signal, however, both BD and DP show higher predicted probability. The higher probability of damage on the bridge deck is attributed to the surface area and larger effect of differential pier-settlement on the entire bridge system. The bridge suffers higher changes in structural responses than at the damaged pier itself, as the bridge deck acts as a support.

Similarly, for rupture of tendons, the most affected area would be the bridge deck and the damage induced due to rupture of tendons will create a non-localized and distributed damage throughout the bridge deck in comparison to bridge piers. The damage localization per sensors is avoided due to non-conclusive inference and a comparison between structural components of the bridge is provided in Figure D.19. It can be observed that rupture of 6 tendons prove to

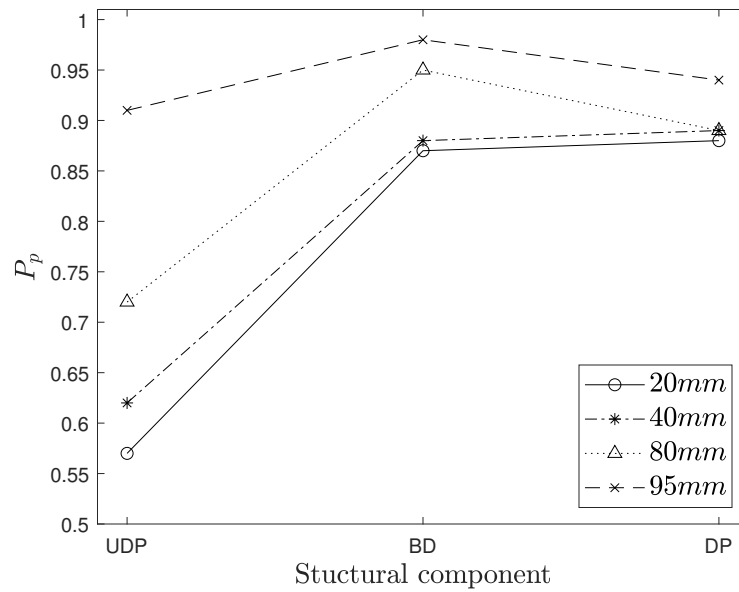


Figure D.18: Damage localization for lowering of pier, where legend shows the amount of pier-settlement.

be worse damage level scenario in comparison to the ruptures with 2 and 4 tendons.

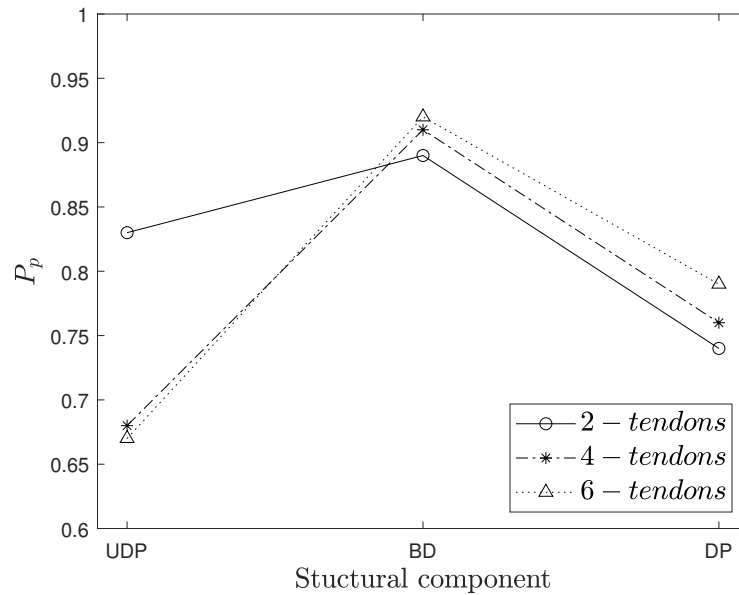


Figure D.19: Damage localization for rupture of tendons, where, the legend shows number of tendon ruptures.

## D.5 Conclusion

This chapter introduced a novel approach for multi-level structural damage detection and localization using LSTM networks on vibration signals. Limited datasets were augmented by windowing of the time-series measurements and the prediction accuracy is improved by a novel voting approach on the predictions of individual windows. Performance of the proposed approach were validated through comprehensive experiments on the Qatar University Grandstand Simulator (QUGS) and the Z24 bridge datasets. It is observed that the proposed algorithm performs well with non-localized and irregular sample sizes, and learns the long-term dependencies in vibration signals. The proposed algorithm is analyzed with sensitivity analysis on window-size as the external parameter to the model. A parametric study is also presented for random initialization of weights. It is demonstrated that a simple LSTM architecture is capable of classifying the time series signals into multiple damage levels with high accuracy and low false negative rate. This work contributes a novel tool for the early detection and localization of damages in civil structures and opens new avenues for future research in the application of deep learning techniques for structural health monitoring, such as evaluating the remaining useful life of structures.

## References

- [1] Osama Abdeljaber et al. “1-D CNNs for Structural Damage Detection: Verification on a Structural Health Monitoring Benchmark Data”. In: *Neurocomputing* 275 (Jan. 2018), pp. 1308–1317. doi: 10.1016/j.neucom.2017.09.069.
- [2] Osama Abdeljaber et al. “Real-Time Vibration-Based Structural Damage Detection Using One-Dimensional Convolutional Neural Networks”. In: *Journal of Sound and Vibration* 388 (Feb. 2017), pp. 154–170. doi: 10.1016/j.jsv.2016.10.043.
- [3] Nawaf Almasri, Ayan Sadhu, and Samit Ray Chaudhuri. “Toward Compressed Sensing of Structural Monitoring Data Using Discrete Cosine Transform”. In: *Journal of Computing in Civil Engineering* 34.1 (Jan. 2020), p. 04019041. doi: 10.1061/(ASCE)CP.1943-5487.0000855.
- [4] James Bergstra and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization”. In: *Journal of Machine Learning Research* 13.10 (2012), pp. 281–305.

- [5] Peter Cawley. “Structural Health Monitoring: Closing the Gap between Research and Industrial Deployment”. In: *Structural Health Monitoring* 17.5 (Sept. 2018), pp. 1225–1244. doi: 10.1177/1475921717750047.
- [6] Gang Chen. “A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation”. In: *arXiv* (Jan. 2018). doi: 10.48550/arXiv.1610.02583.
- [7] Marco Gatti. “Structural Health Monitoring of an Operational Bridge: A Case Study”. In: *Engineering Structures* 195 (Sept. 2019), pp. 200–209. doi: 10.1016/j.engstruct.2019.05.102.
- [8] Nur Sila Gulgec, Martin Takáč, and Shamim N. Pakzad. “Structural Damage Detection Using Convolutional Neural Networks”. In: *Society for Experimental Mechanics Series*. Vol. 3. 2017, pp. 331–337. doi: 10.1007/978-3-319-54858-6\_33.
- [9] B. Hazra et al. “Hybrid Time-Frequency Blind Source Separation Towards Ambient System Identification of Structures”. In: *Computer-Aided Civil and Infrastructure Engineering* 27.5 (2012), pp. 314–332. doi: 10.1111/j.1467-8667.2011.00732.x.
- [10] Guo Junqi et al. “Structural Health Monitoring by Using a Sparse Coding-Based Deep Learning Algorithm with Wireless Sensor Networks”. In: *Personal and Ubiquitous Computing* 18 (Dec. 2014), pp. 1977–1987. doi: 10.1007/s00779-014-0800-5.
- [11] Zhiwei Lin, Yonggui Liu, and Linren Zhou. “Damage Detection in a Benchmark Structure Using Long Short-term Memory Networks”. In: *Chinese Automation Congress (CAC) 2019*. Nov. 2019, pp. 2300–2305. doi: 10.1109/CAC48633.2019.8996864.
- [12] Tongwei Liu et al. “A Data-Driven Damage Identification Framework Based on Transmissibility Function Datasets and One-Dimensional Convolutional Neural Networks: Verification on a Structural Health Monitoring Benchmark Structure”. In: *Sensors* 20.4 (Jan. 2020), p. 1059. doi: 10.3390/s20041059.
- [13] J. Maeck and G. De roeck. “Description of Z24 Benchmark”. In: *Mechanical Systems and Signal Processing* 17.1 (Jan. 2003), pp. 127–131. doi: 10.1006/mssp.2002.1548.
- [14] FuTao Ni, Jian Zhang, and Mohammad N. Noori. “Deep Learning for Data Anomaly Detection and Data Compression of a Long-Span Suspension Bridge”. In: *Computer-Aided Civil and Infrastructure Engineering* 35.7 (2020), pp. 685–700. doi: 10.1111/mice.12528.
- [15] Hossein Qarib and Hojjat Adeli. “Recent Advances in Health Monitoring of Civil Structures”. In: *Scientia Iranica* 21.6 (Dec. 2014), pp. 1733–1742.

- [16] Mohammad Hossein Rafiei and Hojjat Adeli. “A Novel Unsupervised Deep Learning Model for Global and Local Health Condition Assessment of Structures”. In: *Engineering Structures* 156 (Feb. 2018), pp. 598–607. doi: 10.1016/j.engstruct.2017.10.070.
- [17] A. Sadhu, S. Narasimhan, and J. Antoni. “A Review of Output-Only Structural Mode Identification Literature Employing Blind Source Separation Methods”. In: *Mechanical Systems and Signal Processing* 94 (Sept. 2017), pp. 415–431. doi: 10.1016/j.ymsp.2017.03.001.
- [18] Smriti Sharma and Subhamoy Sen. “One-Dimensional Convolutional Neural Network-Based Damage Detection in Structural Joints”. In: *Journal of Civil Structural Health Monitoring* 10.5 (Nov. 2020), pp. 1057–1072. doi: 10.1007/s13349-020-00434-z.
- [19] Premjeet Singh, Majid Keyvanlou, and Ayan Sadhu. “An Improved Time-Varying Empirical Mode Decomposition for Structural Condition Assessment Using Limited Sensors”. In: *Engineering Structures* 232 (Apr. 2021), p. 111882. doi: 10.1016/j.engstruct.2021.111882.
- [20] G. F. Sirca and H. Adeli. “System Identification in Structural Engineering”. In: *Scientia Iranica* 19.6 (Dec. 2012), pp. 1355–1364. doi: 10.1016/j.scient.2012.09.002.
- [21] Sandeep Sony et al. “Vibration-Based Multiclass Damage Detection and Localization Using Long Short-Term Memory Networks”. In: *Structures* 35 (Jan. 2022), pp. 436–451. doi: 10.1016/j.istruc.2021.10.088.
- [22] Yujie Ying et al. “Toward Data-Driven Structural Health Monitoring: Application of Machine Learning and Signal Processing to Damage Detection”. In: *Journal of Computing in Civil Engineering* 27.6 (Nov. 2013), pp. 667–680. doi: 10.1061/(ASCE)CP.1943-5487.0000258.
- [23] Jun Zhang et al. “An Improved Long Short-Term Memory Model for Dam Displacement Prediction”. In: *Mathematical Problems in Engineering* 2019 (Apr. 2019), e6792189. doi: 10.1155/2019/6792189.

# Appendix E

## Experiences with Contrastive Predictive Coding in Industrial Time-Series Classification

### Synopsis

This chapter is based on a technical paper [4] co-authored by the thesis author in the ACM SIGKDD Explorations Newsletter.

Multivariate time-series classification problems are found in many industrial settings; for example, fault detection in a manufacturing process by monitoring sensors signals. It is difficult to obtain large labeled datasets in these settings, for reasons such as limitations in the automatic recording, the need for expert root-cause analysis, and the very limited access to human experts. Therefore, methods that perform classification in a label efficient manner are useful for building and deploying machine learning models in the industrial setting. In this work, we apply a self-supervised learning method called Contrastive Predictive Coding (CPC) to classification tasks on three industrial multivariate time-series datasets. First, the CPC neural network (CPC base) is trained with a large number of unlabeled time-series data instances. Then, a standard supervised classifier such as a multi-layer perception (MLP) is trained on available labeled data using the output embeddings from the pre-trained CPC base. On all three classification datasets, we see increased label efficiency (ability to reach a goal accuracy level with less

labeled examples). In the low data regime (10's or few 100's of labeled examples), the CPC pre-trained model achieves high accuracy with up to 15x less labels than a model trained only on labeled data. We also conduct experiments to evaluate the usefulness of CPC pre-trained classifiers as base models to start an active learning loop, and find that uncertainty sampling does not perform significantly better than random sampling during the initial queries.



## E.1 Introduction

Modern manufacturing plants and process systems are equipped with a large number of sensors that monitor various physical variables on the operation of the systems. These sensors generate vast amounts of time-series data, which can be used for tasks such as quality monitoring, fault detection, and process optimization. Some of these tasks require classification of the generated multi-variate time-series data. Machine learning methods, and especially deep learning methods have shown state of the art performance in these applications [12, 8].

However, deep learning classification methods are typically trained in a fully supervised manner, and they require a large amount of labeled data to achieve high accuracy [15]. In industrial applications, it can be difficult to obtain labeled time-series data due to several reasons [5]. For manual labeling, human experts need to carefully inspect and identify the different classes, which is time-consuming, expensive and also error-prone. Sensor data and labels can be highly process-specific or plant-specific, which makes it impossible to reuse a set of labeled data to train models for multiple plants. Furthermore, some tasks naturally give rise to high class imbalance, and obtaining and labeling data instances from the rare classes is difficult (e.g. faulty class in plant fault detection, because plants do not break down often).

Due to the difficulty in obtaining large labeled datasets in industrial settings, developing time-series classification methods that can be trained in a label-efficient manner to achieve high accuracy is an important research problem. Various approaches have been proposed to address the problem of unavailability of labeled data when training classification models. One solution is to use active learning [20, 6], which interactively queries an expert to label unlabeled instances that are maximally informative to model training. Several forms of semi-supervised learning have also been developed [29, 18], which generally involves the use of both labeled and unlabeled data together to train a model.

A more recent method designed to improve label efficiency of deep neural network classifiers is self-supervised learning. In self-supervised learning (SSL), a neural network is first pre-trained with a proxy task on unlabeled data, followed by a supervised fine-tuning step on labeled data for the actual classification task. Self-supervised learning has achieved state-of-the art label efficiency (high accuracy with limited labeled data) in many domains such as image [9] and text [26] classification.

In this work, we adapt Contrastive Predictive Coding, a self-supervised learning method intro-

duced in [19] to the problem of label-efficient multivariate time-series classification in industrial applications. The major contributions of this work are as follows:

- We propose a self-supervised learning pipeline (Figure E.1) that uses Contrastive Predictive Coding (CPC) [19] to train a neural network classifier for multivariate industrial time-series problems in a highly label efficient manner.
- We provide a thorough analysis of the effect of CPC pre-training on label efficiency in three industrial time-series classification datasets, including the very low data regime (10's or few 100's of labeled examples). The three datasets are the Tennessee Eastman Process dataset [24], a proprietary real-world batch process dataset, and a simulated batch process dataset [28]<sup>1</sup>.
- Since CPC models achieve high accuracy with very few training examples, we conduct further experiments to explore the possibility of using the CPC pre-trained method to improve the performance of an active learning loop. Our finding is that with CPC pre-trained models, uncertainty sampling does not have a clear advantage over random sampling in the initial queries of an active learning loop.

The rest of the chapter is organized as follows. Section E.2 briefly discusses the relevant research in self-supervised learning and time-series classification. Section E.3 provides an overview of the Contrastive Predictive Coding method, and section E.4 describes the industrial time-series datasets and the classification problems. Section E.5 explains the proposed self-supervised learning based classification pipeline. Section E.6 presents the details of the experiments and the results analysis. Section E.7 provides concluding remarks.

## E.2 Related Work

The need to learn efficiently from a limited number of labels or to obtain label information in an efficient fashion is not limited to industrial applications and several approaches to address this problem exist.

---

<sup>1</sup>The simulation dataset is currently in the process of being published.

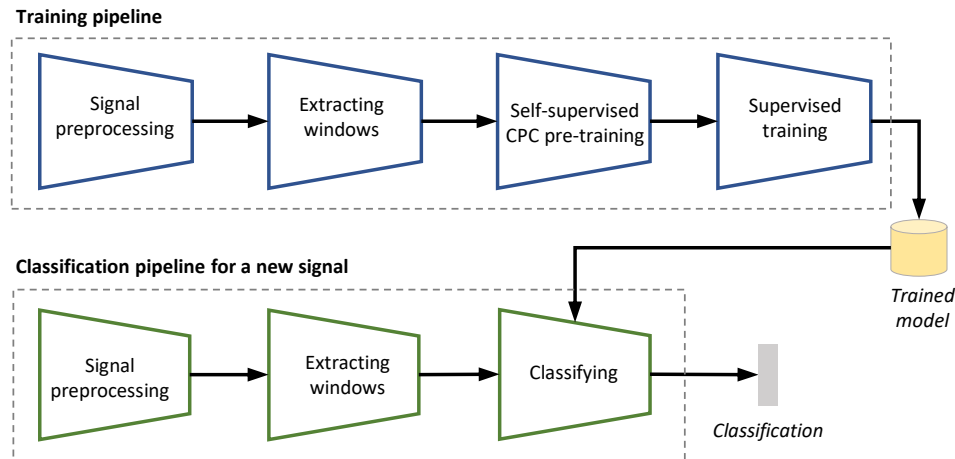


Figure E.1: Training and testing pipeline for time-series classification with self-supervised learning.

### E.2.1 Semi-supervised learning

Semi-supervised learning tries to leverage unlabeled data in the training process based on two assumptions: cluster assumption and manifold assumptions [31]. The first assumption assumes that the data has cluster structure and instances falling into the same clusters have the same class label. The latter assumes that the data lies on a manifold and nearby data points have similar predictions. Unlabeled data gives insights into which data is similar. Semi-supervised learning has been applied to time-series data as well [29, 18]. One challenge with semi-supervised learning is the possible degradation of performance when using unlabeled data, which leads to the fact that semi-supervised learning does not consistently outperform supervised learning methods [27].

### E.2.2 Self-supervised learning (SSL)

Self-supervised learning is a machine learning process using deep neural networks that learns from data using labels obtained in a "semi-automatic" process, or that tries to predict parts of the data from other parts [14]. The network trained in a self-supervised fashion is then adapted in a supervised training process to solve a more relevant task. Many studies show that self-supervised pre-training leads to higher label-efficiency. SSL has also been successfully applied to time-series classification problems, such as phoneme and speaker identification in audio signals [19], and biomedical signal classification [17]. Recent parallel work [21, 22]

proposes a self-supervised learning method for time-series classification with a benchmark on an industrial time-series dataset. However, the literature on self-supervised learning methods for industrial multivariate time-series classification problems is lacking.

### E.2.3 Active learning

Active learning tries to learn with better performance with fewer labeled samples by enabling the machine learning algorithm to decide from which data it learns. For this purpose, an active learning loop queries for labels of previously unlabeled samples [25]. The problem of *pool-based active learning* [25], where the active learner chooses samples from a large pool of unlabeled data matches the scenario described in this chapter best. Pool-based active learning has previously successfully applied on process data in the chemical industry [23, 6] and other time-series datasets [20]. The performance of active learning depends strongly on the selection of the initial labeled training set [10] and the selection of good hyper-parameters during the active learning process [7]. The combination of active learning and self-supervised learning should help to overcome both issues by making the active learning process more robust to the selection of the initial dataset and by using a relatively simple model for the actual classification task.

### E.2.4 Combining SSL and Active Learning

The idea to combine self-supervised learning and active learning is not new. For instance in one work [13], SSL is used to rank images w.r.t to *image quality* as a proxy task. Self-supervised learning can also be an approach to overcome the *cold-start* problem of active learning, the issue that the model is not stable when trained with little data at the beginning of the process [30]. In [30], the authors applied SSL to pre-train NLP models and used the loss of the pre-trained model as a measure of informativeness in early iterations of the active learning process. More recent results show that the possible gain of combining SSL and active learning can be limited. In [2], a combination of SSL and random sampling performs better in the low data regime and active learning based sampling is only beneficial in higher-data regimes. The results of [3] even imply that active learning offers only marginal improvements and SSL alone is sufficient to achieve labeling efficiency.

## E.3 Contrastive Predictive Coding

The self-supervised learning approach to building a neural network classifier involves two steps. First, a neural network is trained on a suitable pre-training task on a large number of unlabeled training examples. This is the representation learning step. Typical pre-training tasks include predicting a masked part of a signal given the remaining parts (e.g. predicting the future given past segments of the signal). This step does not require ground truth labels, which enables the use of any amount of unlabeled data instances available. In the second step (fine-tuning), a classifier (e.g. a fully connected neural network) is trained in a supervised manner with available labeled training examples. In this step, one can choose to either freeze the representation layers of the neural network (no weight updates to the base), or allow weight updates to the base. Designing self-supervised learning methods involves identifying a neural network architecture and a pre-training objective that learns good representations of the data modality. For the multivariate industrial time-series datasets we use in this work, we adapt Contrastive Predictive Coding or CPC [19], a generic self-supervised learning method designed to work with many types of data (images, audio etc.)

### E.3.1 CPC architecture and self-supervision task

The CPC neural network architecture is illustrated in Figure E.2. We call it the *CPC\_Full\_SSL\_Model*. In the self-supervised pre-training stage, the CPC model is given a sequence of past windows of the time-series, and a second set of future windows (or samples) equal to the number of future steps. However, only one of these samples correspond to the true window of that future step (positive sample), while the others are taken from elsewhere in the dataset (negative samples). The pre-training training objective is to identify the positive sample in the given set of samples, which is formulated as a classification task.

The components of the *CPC\_Full\_SSL\_Model* are described below.

- *A sequence of encoder neural networks.* The encoders have shared weights, and adjacent windows of the time-series ( $x_t$ ) are given as input to each encoder. Each encoder produces a latent vector representation  $z_t$ . The architecture of the encoder is chosen based on the data modality. For multivariate time-series data, we use a strided 1-D convolutional network.

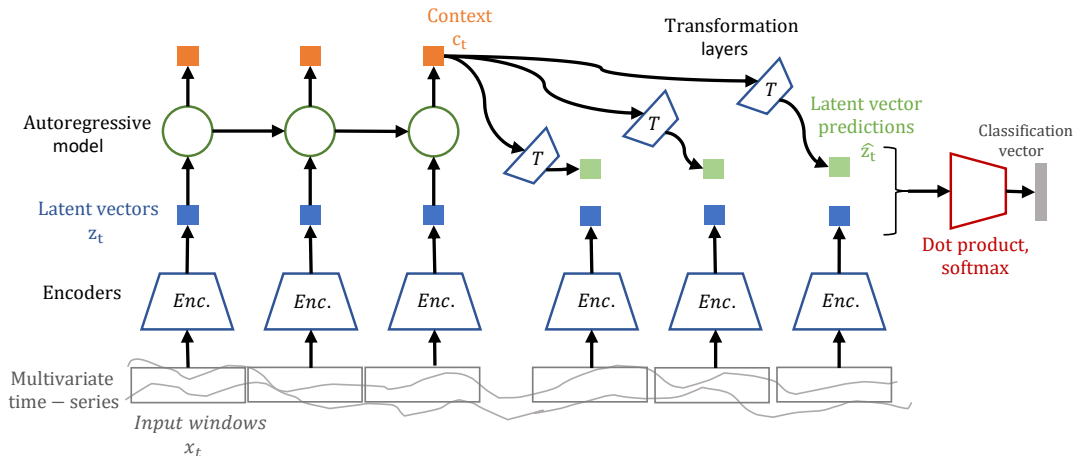


Figure E.2: Contrastive predictive coding neural architecture of the *CPC\_Full\_SSL\_Model* (adapted from [19])

- *An autoregressive (AR) model.* Typically, this is a recurrent neural network (e.g. GRU or LSTM units). The sequence of encoder outputs is given as input to the AR model. The final output state of the AR model is considered as a single vector representation of the window sequence of the original time-series. It is named the context  $c_t$ .
- *A set of transformation layers (or networks).* These layers transform the context  $c_t$  into the encoder output latent space. The output  $\hat{z}_t$  of each layer can be thought of as a latent space prediction of the future window corresponding to the time step. Note that the layers of different time steps have different weights (a unique layer for each time step). We use fully connected layers with linear activations in this work.
- *Dot product and softmax.* These operations output the classification probabilities softmax vector. The element with the highest value is treated as the correct future window (positive sample) among the given set of future windows. There are no trainable weights in these operations.

### E.3.2 CPC classifier

Once the CPC model architecture is trained for the self-supervised task of correct future window identification on unlabeled data, we can use the trained weights of part of the model for the supervised classification task. The output of this part of the model is called an embedding, and it is used as input to a classifier for the supervised classification task. We have two options for the choice of embedding.

- Take the encoder and autoregressive (AR) model together and use the final context of the AR model as the embedding (input to train a classifier). This requires a longer context of the input signal (a sequence of windows).
- Use the encoder output (encoded vector) of a single window as input to train a classifier.

In this work, we use the first method, as industrial time-series typically come from long running processes, and our experiments gave better results in this setting. The classifier that is trained on the embeddings is a small fully connected neural network with two hidden layers. We call the model that is trained for the classification the *CPC\_Classifier* (encoders + AR model + small fully connected neural network). During the supervised training stage, weight updates are allowed on the entire *CPC\_Classifier* network, which includes the base part (encoder and AR model), so that a fine-tuning of the weights occurs.

## E.4 Industrial Datasets and Classification Problems

To test the suitability of CPC for solving industrial time-series classification tasks with increased label-efficiency, we applied it to three datasets with associated classification problems. Table E.1 shows the important characteristics of these datasets.

TEP refers to the Tennessee Eastman Process [1], a process simulator that replicates a complex industrial process operated by the Eastman Chemical Company containing five units: reactor, condenser, compressor, separator, and stripper. The simulator and generated datasets are widely used in control research. TEP is a representative of continuous production process, where a continuous flow of input material is transformed into the output material with a fixed process behaviour over a long period of time (in some cases, years).

For our experiment we used a publicly available dataset created for the purpose of fault and anomaly detection [24]. The dataset contains 52 time-series where 41 variables represent sensor measurements and 11 are manipulated variables. The variables capture quantities like flow rates, pressures, temperatures, levels, and compressor power output. The public dataset contains 20 different fault scenarios by introducing different types of disturbances. This results in 21 different classes including the absence of disturbances. Besides the presence or absence of disturbances, the different simulation runs differ in the measurement noise. From a practitioners perspective, TEP is an highly idealized benchmark dataset due to its size, well balanced

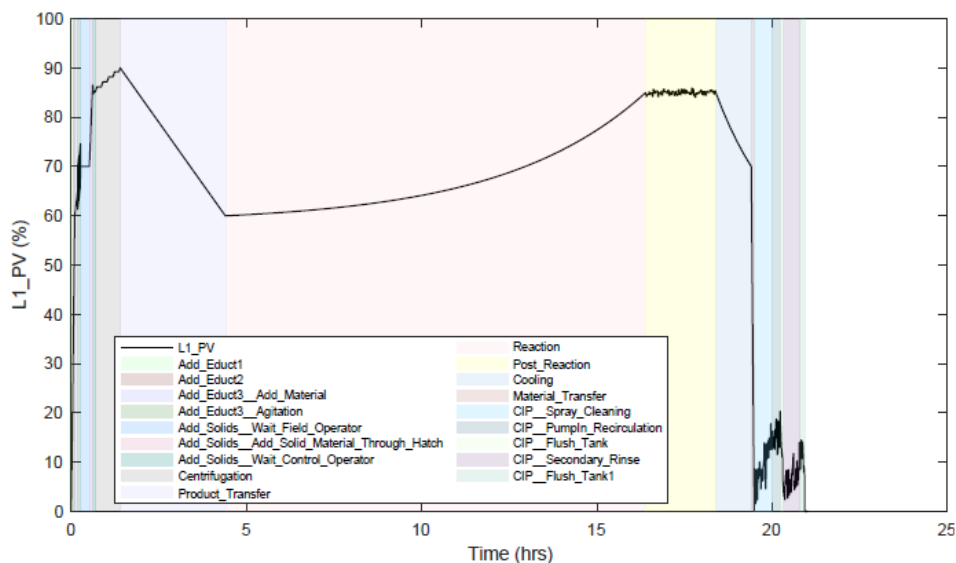


Figure E.3: Nominal behaviour of a batch in the simulated dataset. Each highlighted segment belongs to a different batch phase (a class in the classification problem) [28]

classes and very homogeneous production conditions.

The other two datasets are extracted from batch production processes, which consist of a sequence of steps such as heating, cooling, chemical reactions, or stirring. Due to their event-driven character, production runs (batches) for the same product can vary considerably. For instance, differences of several hours of batch duration are common [11]. The analysis of batch processes relies on reliable information about the start and end of the production steps and transitions are often triggered manually by operators and not consistently documented.

Figure E.3 shows the behavior of the level of the unit reactor from the simulated dataset. The colored areas indicate the duration of the different steps or batch phases in the production process. These batch phases are the classes of the time-series classification problem. A related classification problem, to recognize the presence of certain production steps in the data is described in [16]. The different sequential steps of the production take place in so-called units and the focus of both batch datasets is a single unit resulting in a much smaller number of signals compared to the TEP dataset. The first dataset ("Batch Real") was provided by a partner of a collaborative research project<sup>2</sup>, and it contains data from a real-world industrial process. The second dataset ("Batch Simulated") was produced using a process simulator [28]<sup>1</sup>.

<sup>2</sup><http://keen-plattform.de/keen/en/>



Table E.1: Dataset characteristics

	<b>TEP</b>	<b>Batch Real</b>	<b>Batch Simulated</b>
Data source	Simulation	Real-world batch process	Simulation of batch process
Availability	Public [24]	Proprietary <sup>2</sup>	Proprietary <sup>1</sup>
Classification task	Fault detection	Batch phase classification	Batch phase classification
No. of signals	52	8	5
No. of classes	21 (all taken)	15 (only 4 taken)	15 (only 3 taken)
Training set size	21k examples	6k examples	4k examples

## E.5 Proposed Classification Pipeline

In order to leverage contrastive predictive coding to train classifiers for multivariate industrial time-series datasets in a label efficient manner, we propose the classification pipeline in Figure E.1. The steps involved are described below.

- *Train-test split.* The individual time-series in the two proprietary batch process datasets (Batch Real and Batch Simulated) are split into train and test sets (80%-20% split). The splitting is done in the very first step to avoid any data leaks in later stages. TEP dataset contains a separate test set, therefore splitting is not necessary. From the training sets, a 20% validation set is separated to measure out-of-sample loss during model training and perform early stopping to avoid overfitting.
- *Data preprocessing.* Standard scaling is performed on the individual time-series of the TEP dataset. The time-series in the two batch process datasets were first mean-resampled to obtain mean values per minute. This is necessary, as the different variables in the multivariate time-series come from sensors that are sampled at different sampling intervals. Then, min-max scaling is performed on the individual time-series followed by a data imputing step (linear interpolation) to fill any missing values.
- *Data instance preparation.* The input to the *CPC\_Full\_SSL\_Model* is a sequence of past windows and a set of windows for the future steps with only one correct (positive) window. A sliding window along the time-series is used to prepare these instances. The incorrect (negative) windows for the future steps are taken randomly from anywhere in the dataset. Window size for the TEP dataset is 20, and on the batch process datasets the window size is 10. To increase the number of training examples, we set a 80% overlap

of the time-series between two input sequences. The input to the *CPC\_Classifier* is a set of past windows prepared in the same manner.

- *Self-supervised pre-training.* The *CPC\_Full\_SSL\_Model* has a strided 1-D convolutional network (no max-pool layers) as the encoder with 5 layers. Each layer has 256 nodes with ReLU activations, kernel sizes: [10, 8, 4, 4, 4] and strides: [5, 4, 2, 2, 2]. This produces a latent space encoded vector of size 256. The autoregressive model is a single-layer recurrent neural network with 128 GRU cells. The AR model sequence length is 10 for the TEP dataset, and 5 for the batch process datasets. The transformation layers are single-layer fully connected networks with 256 linear nodes (to match the encoder output size). The number of future steps is 20 for the TEP dataset, and 5 for the batch process datasets. The pre-training task is the correct future window classification task as described in section E.3 on the entire unlabeled training set with cross entropy loss function. We train the network for 80 epochs using the Adam optimizer with learning rate 0.001 and batch size 64.
- *Supervised classifier training.* Once the *CPC\_Full\_SSL\_Model* is pre-trained with unlabeled data, the *CPC\_Classifier* model is formed by taking the encoder and AR model and setting a two-layer fully connected neural network on top of it. The two layers have 128 and 64 nodes with ReLU activations with a dropout rate of 0.2. The *CPC\_Classifier* is trained for the downstream classification tasks of the datasets (fault detection for TEP dataset and batch phase classification for the batch process datasets) with only labeled data in a supervised manner with cross entropy loss function. Weight updates occur in the full network, which means that the encoder and AR model weights are fine-tuned for the downstream task. We train the network for 100 epochs using the Adam optimizer with learning rate 0.001 and batch size 64.

The hyperparameters of the CPC network architecture are based on the parameters of the original CPC paper [19]. We find that these parameters work well for all three datasets, indicating a degree of architectural robustness that makes the CPC network suitable for different time-series tasks. Parameters such as encoder input window size and autoregressive model sequence length were determined by trial-and-error (trying out a range from small to large).

## E.6 Experimental Results

### E.6.1 Initial experiments

In order to establish the classification accuracy for each task, we train multiple models with the full labeled datasets. The *CPC\_Full\_SSL\_Model* is first trained for the self-supervision task (correct future window classification) with unlabeled data. For this task, the models give 63% accuracy on the TEP dataset and 86% on the batch process datasets. Figure E.4 shows UMAP visualizations of the CPC embeddings on the test datasets. The TEP test set embedding UMAP shows certain classes forming separate clusters while some classes being scattered in the space. The batch real test set embeddings have well-separated clusters. This observation shows that the CPC embeddings are discriminative for the downstream classification task and provides an indication that the CPC model was well trained.

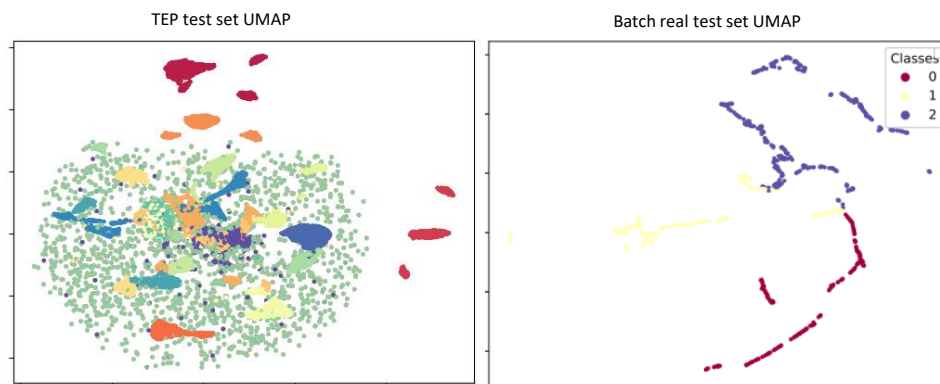


Figure E.4: UMAP visualization of CPC embeddings on the test sets of TEP (left) and batch process real (right) datasets. Different colors represent classes of each dataset.

Next, the *CPC\_Classifier* model was trained for the downstream classification tasks in a supervised manner (cross-entropy loss function) with all labeled data in the training sets. Training of the *CPC\_Classifier* was done under three settings, and an LSTM model was also trained. Following is a summary of the training settings. Note that the CPC base refers to the set of convolutional encoders and the autoregressive (GRU) network.

- *LSTM model*. The LSTM network has two hidden layers (128 and 64 nodes) with tanh activations. The length of an input sequence to the LSTM model is equal to the number of samples in an input sequence of the CPC model (200 in the TEP dataset, 50 in the batch process datasets).

- *CPC: no pre-training.* All *CPC\_Classifier* weights (1-D CNN encoder, GRU network, fully connected neural network classifier) are randomly initialized and weight updates occur in all components.
- *CPC: pre-trained, base frozen.* Weights of the 1-D CNN encoder and GRU network are pre-trained and frozen, weights of the fully connected neural network classifier are randomly initialized, and weight updates occur only in the fully connected neural network classifier.
- *CPC: pre-trained, base fine-tuned.* Weights of the 1-D CNN encoder and GRU network are pre-trained, fully connected neural network classifier weights are randomly initialized, and weight updates occur in all components.

The results are summarized in Table E.2. On the TEP multiclass classification problem (with all 21 classes), the training settings where all network weights are updated yield accuracy upward of 90%. The classifier trained on CPC embeddings (*CPC: pre-trained, base frozen* setting) yields a relatively lower accuracy of 82%. This is likely due to the fact that the representations learned by the CPC base from the self-supervision task needs fine-tuning in the downstream task of fault detection for higher performance. The LSTM network also achieves a comparable 81.5% accuracy. The accuracy figures on the TEP dataset are comparable with the 81.43% accuracy reported in [21]. However, a direct comparison is difficult due to differences in how the data instances are prepared (different windowing techniques, subset of classes used etc).

On the batch process datasets, all CPC model training settings achieve near perfect classification. In addition to establishing a comparison of the possible training methods, this experiment also verifies that the neural network architecture of the *CPC\_Classifier* is indeed capable of learning a classification function on the three datasets given all available labeled data. On all three datasets, the CPC models perform significantly better than the LSTM network.

Table E.2: Classification accuracy on the three datasets (trained on all labeled data).

<b>Training method</b>	<b>TEP</b>	<b>Batch real</b>	<b>Batch simulated</b>
LSTM model	81.5%	95%	85.6%
CPC with no pre-training	90.12%	99.33%	98.6%
Pre-trained + frozen CPC base	82%	99.7%	99.3%
Pre-trained + fine-tuned CPC base	92.3%	99.4%	100%

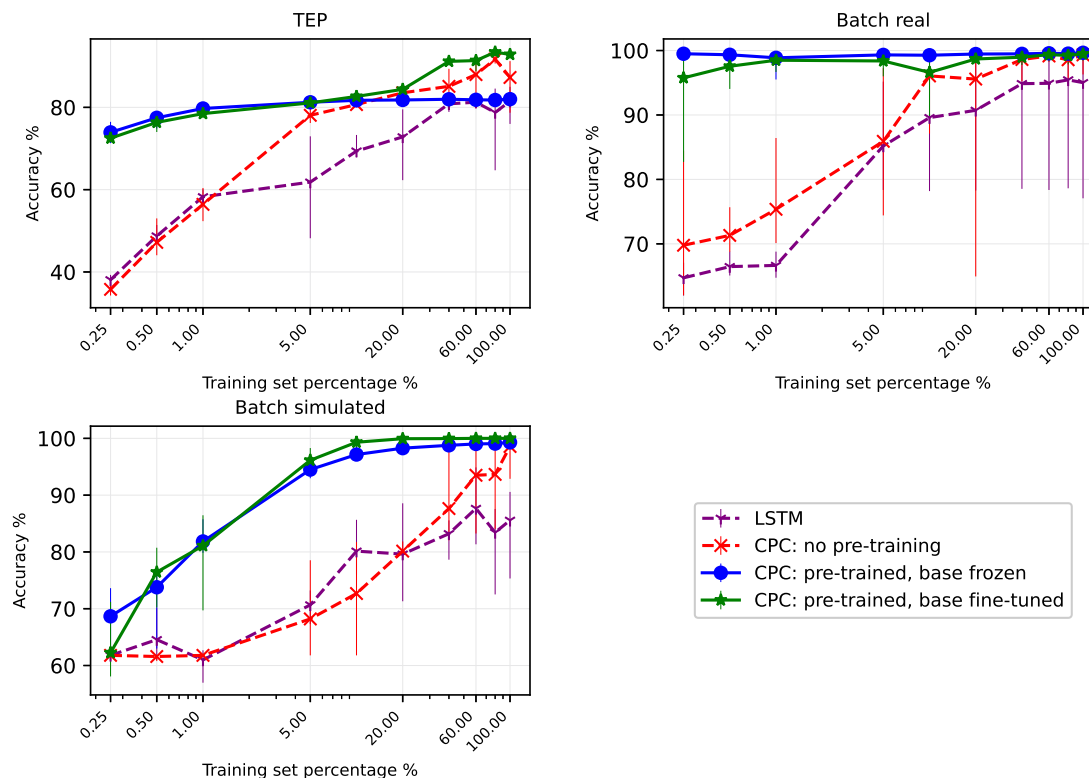


Figure E.5: Label efficiency curves of different types of *CPC\_Classifier* training on the three datasets. Top left: TEP dataset, top right: batch process real dataset, bottom: batch process simulated dataset. Vertical error bars indicate minimum and maximum values in experiment repetitions.

## E.6.2 Label efficiency experiments

To benchmark the label efficiency of the contrastive predictive coding pipeline, we conducted the four types of model training described in the previous section (Table E.2). In the downstream task supervised training stage, the models were trained with subsets of the full labeled training sets. In order to observe the effects of CPC pre-training on the very-low-data regime, we trained models with as little as 0.25% of the full training set, which corresponds to 10 - 40 examples in the three datasets. Such minimal availability of labeled data is possible in certain industrial applications. The resulting label efficiency curves for the three datasets are shown in Figure E.5.

On all three datasets, the self-supervised pre-trained *CPC\_Classifier* gives high accuracy compared to the randomly initialized *CPC\_Classifier* and baseline LSTM model when trained with as little as 0.25% of the full training set (e.g. 72.8% accuracy vs. 35.8% accuracy on the TEP

dataset with 0.25% labeled training examples). Note that this very-low-data regime consists of 42 labeled training examples in the TEP dataset and 10 labeled training examples in the batch process real dataset.

The large label efficiency gain of the pre-trained CPC models persists further in to the low-to-mid-data regime (up until 20% or more of labeled training data is made available). On the TEP dataset, the pre-trained CPC models achieve 78% accuracy with about 10x fewer labels, and 82% with about 2.5x fewer labels than the randomly initialized CPC model or the LSTM network. On the batch process real dataset, the pre-trained CPC models achieve 98% accuracy with 100x fewer labels than the randomly initialized model or the LSTM network. A similar large efficiency gain is seen on the batch process simulated dataset (100% accuracy with 10x fewer labels). All models converge to roughly the same accuracy values when they are trained with more than 20% of labeled training examples.

Small labeled training sets can be the realistic case in certain industrial time-series classification problems, and our results indicate that a pre-trained CPC model is beneficial in these applications.

### E.6.3 Active learning experiments

Active learning algorithms attempt to reduce human labeling effort by choosing most informative or important unlabeled examples and prompting a human expert to label them. Typically, a model is first trained on available labeled data, and its output is then used to select unlabeled examples to be labeled by a human expert. In the case of uncertainty sampling, examples where the model is most uncertain (e.g. largest softmax value is small) are selected. When the initial model is not well-trained, its output is unreliable, which makes it difficult for an active learning algorithm to select good examples.

Since the self-supervised pre-trained *CPC\_Classifier* gives high accuracy even when trained with a small number of labeled examples, we use it as the initial model in an active learning loop instead of a randomly initialized model. The model is trained for the classification task on an initial labeled subset (labeling budget) of 0.1% of the full training set. Then, batches of unlabeled examples are selected via uncertainty sampling and random sampling (batch size: 20 on TEP dataset, 10 on the batch process real dataset). The labels of these examples are then revealed (as if they were labeled by a human expert), and the model is trained with the set of

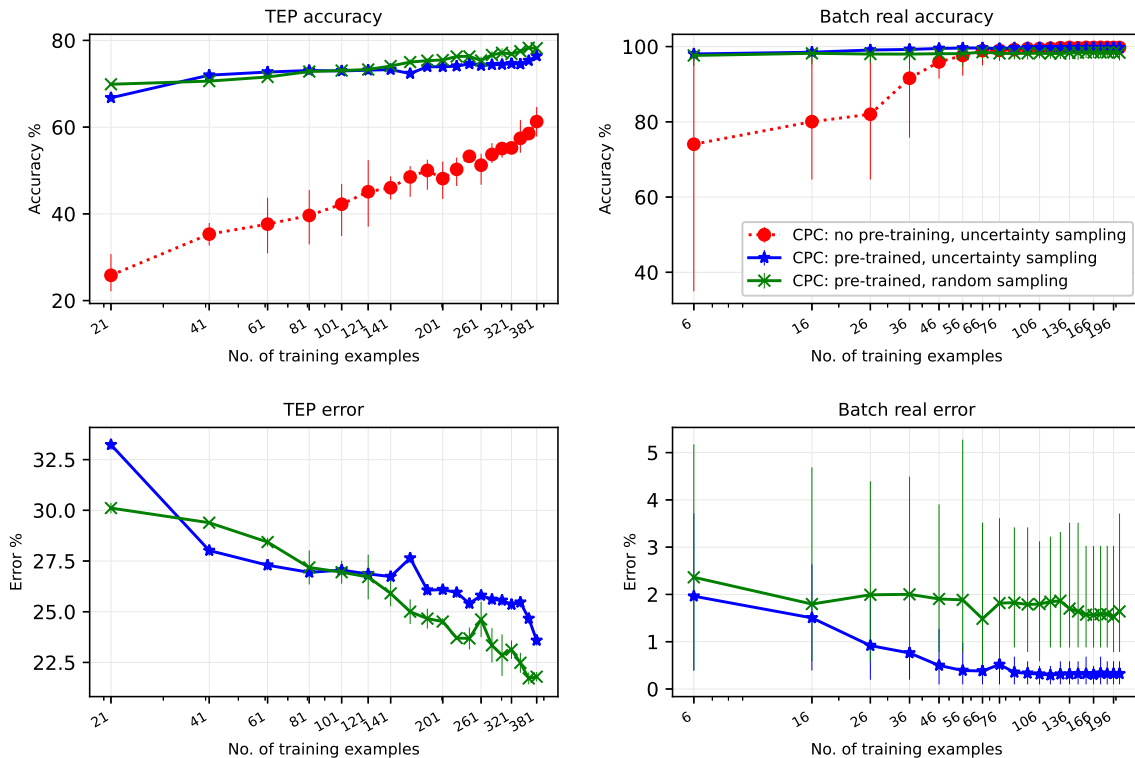


Figure E.6: Active learning curves (accuracy and error) of on two datasets. Left: TEP dataset, right: batch process real dataset. Vertical error bars indicate minimum and maximum values in experiment repetitions.

all available labeled examples (initial labeled subset and the set of examples whose labels were revealed via the active learning loop).

The resulting active learning performance curves are shown in Figure E.6. On the TEP dataset, the self-supervised pre-trained *CPC\_Classifier* does not provide a clear advantage to uncertainty sampling over random sampling. On the Batch real dataset, uncertainty sampling of the pre-trained model produces a lower error than random sampling as more queries are made. However, the maximum difference of average error between the two sampling methods is around 1.57%, which may not be considered as a significant gain by uncertainty sampling, especially during the initial few queries. On both datasets, the self-supervised pre-trained *CPC\_Classifier* models start out with a much higher accuracy than the CPC models with no pre-training (randomly initialized), and the gap continues through the low-data regime. This observation is consistent with the results of the label efficiency experiments (section E.6.2, Figure E.5).

While there is no prior work that combines self-supervised and active learning on the TEP dataset for direct comparison, such techniques applied to image data have produced similar results. In [2], authors report that random sampling with a self-supervised pre-trained CNN model on multiple image datasets performs better than active learning sampling methods based on entropy and embeddings distance, especially in the low-data regime. Similarly, authors of [3] find that active learning algorithms provide no additional benefit for image classification when combined with self-supervised and semi-supervised methods.

These results indicate that active learning with uncertainty sampling may not be a worthwhile time investment when building neural networks for industrial time-series classification problems with small labeled datasets. However, if a practitioner plans to use active learning to minimize labeling effort, self-supervised pre-training of the models is a good first consideration as the pre-trained models perform better in the low-data regime.

## E.7 Conclusion

In this chapter, we presented a self-supervised learning pipeline based on contrastive predictive coding or CPC for multivariate industrial time-series classification in a label efficient manner. This method leverages large amounts of available unlabeled time-series data for pre-training a CPC neural network to learn valid representations for downstream classification tasks. We examined the label efficiency gains of the proposed pipeline on three time-series classification datasets. In all cases, the CPC pre-trained models gave high accuracy with a very small number of labeled examples.

We also explored the possibility of using a CPC-pre-trained model to cold-start an active learning loop. We found that a loop with uncertainty sampling does not perform better than random sampling during the initial queries on the time-series datasets.

In many industrial applications with classification problems, vast amounts of unlabeled time-series data is available, yet the human expertise to label them is expensive or unavailable. In comparison, the computational cost of self-supervised pre-training is negligible. Therefore, the proposed CPC pipeline is an excellent candidate solution for such industrial classification problems.

In future work, we will extend this project to a benchmark to evaluate more self-supervised



methods in multivariate industrial time-series applications. It is important to evaluate these methods in the presence of typical data related issues such as heavy class imbalance (e.g. anomaly detection; much fewer failure cases than normal operation), varying noise levels due to uncalibrated sensors, missing values, and out-of-distribution test-time settings (e.g. due to varying raw material qualities or differing operational conditions).

## References

- [1] Andreas Bathelt, N. Lawrence Ricker, and Mohieddine Jelali. “Revision of the Tennessee Eastman Process Model”. In: *9th IFAC Symposium on Advanced Control of Chemical Processes*. Vol. 48. Jan. 2015, pp. 309–314. doi: 10.1016/j.ifacol.2015.08.199.
- [2] Javad Zolfaghari Bengar et al. “Reducing Label Effort: Self-supervised Meets Active Learning”. In: *IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1631–1639. doi: 10.1109/ICCVW54120.2021.00188.
- [3] Yao-Chun Chan, Mingchen Li, and Samet Oymak. “On the Marginal Benefit of Active Learning: Does Self-Supervision Eat Its Cake?” In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 3455–3459. doi: 10.1109/ICASSP39728.2021.9414665.
- [4] Sunanda Gamage, Benjamin Klopfer, and Jagath Samarabandu. “Experiences with Contrastive Predictive Coding in Industrial Time-Series Classification”. In: *ACM SIGKDD Explorations Newsletter* 24.2 (Dec. 2022), pp. 114–123. doi: 10.1145/3575637.3575654.
- [5] Marco Gärtler et al. “The Machine Learning Life Cycle in Chemical Operations—Status and Open Challenges”. In: *Chemie Ingenieur Technik* 93.12 (2021), pp. 2063–2080. doi: 10.1002/cite.202100134.
- [6] Zhiqiang Ge. “Active Learning Strategy for Smart Soft Sensor Development under a Small Number of Labeled Data Samples”. In: *Journal of Process Control* 24.9 (2014), pp. 1454–1461. doi: 10.1016/j.jprocont.2014.06.015.
- [7] Yonatan Geifman and Ran El-Yaniv. “Deep Active Learning over the Long Tail”. In: *arXiv* (2017). doi: 10.48550/arXiv.1711.00941.
- [8] Ashish Gupta et al. “Approaches and Applications of Early Classification of Time Series: A Review”. In: *IEEE Transactions on Artificial Intelligence* 1.1 (2020), pp. 47–61. doi: 10.1109/TAI.2020.3027279.

- [9] Olivier Henaff. “Data-Efficient Image Recognition with Contrastive Predictive Coding”. In: *International Conference on Machine Learning*. Vol. 119. PMLR, 2020, pp. 4182–4192.
- [10] Rong Hu, Brian Mac Namee, and Sarah Jane Delany. “Off to a Good Start: Using Clustering to Select the Initial Training Set in Active Learning”. In: *23rd International Florida Artificial Intelligence Research Society Conference*. 2010. doi: 10.21427/D7Q89W.
- [11] Sau Lee et al. “Modernizing Pharmaceutical Manufacturing: From Batch to Continuous Production”. In: *Journal of Pharmaceutical Innovation* 10 (Mar. 2015). doi: 10.1007/s12247-015-9215-8.
- [12] Yaguo Lei et al. “Applications of Machine Learning to Machine Fault Diagnosis: A Review and Roadmap”. In: *Mechanical Systems and Signal Processing* 138 (2020), p. 106587. doi: 10.1016/j.ymsp.2019.106587.
- [13] Xialei Liu, Joost van de Weijer, and Andrew D. Bagdanov. “Exploiting Unlabeled Data in CNNs by Self-Supervised Learning to Rank”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.8 (Aug. 2019), pp. 1862–1878. doi: 10.1109/TPAMI.2019.2899857.
- [14] Xiao Liu et al. “Self-Supervised Learning: Generative or Contrastive”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.1 (Jan. 2023), pp. 857–876. doi: 10.1109/TKDE.2021.3090866.
- [15] Gary Marcus. “Deep Learning: A Critical Appraisal”. In: *arXiv* (2018). doi: 10.48550/arXiv.1801.00631.
- [16] Silke Merkelbach et al. “Automatic Detection of Start and End Times of Batch Phases for the Process Industry”. In: *VDI-Kongress Automation*. Baden-Baden, DE, 2021.
- [17] Mostafa Neo Mohsenvand, Mohammad Rasool Izadi, and Pattie Maes. “Contrastive Representation Learning for Electroencephalogram Classification”. In: *Machine Learning for Health NeurIPS Workshop*. Vol. 136. PMLR, 2020, pp. 238–253.
- [18] Minh Nhut Nguyen, Xiao-Li Li, and See-Kiong Ng. “Positive Unlabeled Learning for Time Series Classification”. In: *2nd International Joint Conference on Artificial Intelligence*. Vol. 2. AAAI Press, 2011, pp. 1421–1426.
- [19] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation Learning with Contrastive Predictive Coding”. In: *arXiv* (2018). doi: 10.48550/arXiv.1807.03748.

- [20] Fengchao Peng, Qiong Luo, and Lionel M Ni. “ACTS: An Active Learning Method for Time Series Classification”. In: *33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 175–178. doi: 10.1109/ICDE.2017.68.
- [21] Johannes Pöppelbaum, Gavneet Singh Chadha, and Andreas Schwung. “Contrastive Learning Based Self-Supervised Time-Series Analysis”. In: *Applied Soft Computing* 117 (2022), p. 108397. doi: 10.1016/j.asoc.2021.108397.
- [22] Theivendiram Pranavan et al. “Contrastive Predictive Coding for Anomaly Detection in Multi-variate Time Series Data”. In: *arXiv* (2022). doi: 10.48550/arXiv.2202.03639.
- [23] Gokaraju K Raju and Charles L Cooney. “Active Learning from Process Data”. In: *AIChE journal* 44.10 (1998), pp. 2199–2211.
- [24] Cory A. Rieth et al. *Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*. 2017. doi: 10.7910/DVN/6C3JR1. (Visited on 12/15/2023).
- [25] Burr Settles. *Active Learning Literature Survey*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences, 2009. URL: <https://minds.wisconsin.edu/handle/1793/60660> (visited on 12/23/2023).
- [26] Chi Sun et al. “How to Fine-Tune BERT for Text Classification?” In: *China National Conference on Chinese Computational Linguistics*. Springer, 2019, pp. 194–206. doi: 10.1007/978-3-030-32381-3\_16.
- [27] Jesper E Van Engelen and Holger H Hoos. “A Survey on Semi-Supervised Learning”. In: *Machine Learning* 109.2 (2020), pp. 373–440. doi: 10.1007/s10994-019-05855-6.
- [28] Margarida L. C. Vicente et al. “A Benchmark Model to Generate Batch Process Data for Machine Learning Testing and Comparison”. In: *Computer Aided Chemical Engineering*. Vol. 51. Elsevier, Jan. 2022, pp. 217–222. doi: 10.1016/B978-0-323-95879-0.50037-0.
- [29] Li Wei and Eamonn Keogh. “Semi-Supervised Time Series Classification”. In: *12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, USA, 2006. doi: 10.1145/1150402.1150498.
- [30] Michelle Yuan, Hsuan-Tien Lin, and Jordan Boyd-Graber. “Cold-Start Active Learning through Self-supervised Language Modeling”. In: *2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, Nov. 2020, pp. 7935–7948. doi: 10.18653/v1/2020.emnlp-main.637.
- [31] Zhi-Hua Zhou. “A Brief Introduction to Weakly Supervised Learning”. In: *National science review* 5.1 (2018), pp. 44–53. doi: 10.1093/nsr/nwx106.

# Curriculum Vitae

<b>Name</b>	Sunanda Gamage
<b>Education</b>	Ph.D. candidate in Electrical and Computer Engineering University of Western Ontario (Sep. 2018 – Present)  B.Sc. Eng. (Hons) in Electronic and Telecommunication Engineering University of Moratuwa, Sri Lanka. First class Hons. (2010 – 2015)
<b>Honours &amp; Awards</b>	Ontario Graduate Scholarship (OGS) 2021-2022 Ontario Graduate Scholarship (OGS) 2022-2023 German Academic Exchange Service (DAAD) research internship scholarship
<b>Related Work Experience</b>	Research and Teaching Assistant University of Western Ontario (Sep. 2018 – Present)  Research Intern ABB Corporation, Germany (Sep. 2021 - Dec 2021)  Software Engineer London Stock Exchange Technology, Sri Lanka (2015 - 2018)

## Publications

- Gamage, S., Klopper, B., & Samarabandu, J. (2022). Experiences with contrastive predictive coding in industrial time-series classification. *ACM SIGKDD Explorations Newsletter*, 24(2), 114-123.
- Sony, S., Gamage, S., Sadhu, A., & Samarabandu, J. (2022). Multiclass damage identification in a full-scale bridge using optimally tuned one-dimensional convolutional neural network. *Journal of Computing in Civil Engineering*, 36(2), 04021035.
- Sony, S., Gamage, S., Sadhu, A., & Samarabandu, J. (2022, January). Vibration-based multiclass damage detection and localization using long short-term memory networks. In *Structures* (Vol. 35, pp. 436-451). Elsevier.
- Gamage, S., & Samarabandu, J. (2020). Deep learning methods in network intrusion detection: A survey and an objective comparison. *Journal of Network and Computer Applications*, 169, 102767.