

**Boston University**

**OpenBU**

**<http://open.bu.edu>**

---

Boston University Theses & Dissertations

Boston University Theses & Dissertations

---

2021

# Topics of deep learning in security and compression

---

<https://hdl.handle.net/2144/41938>

*"Downloaded from OpenBU. Boston University's institutional repository."*

BOSTON UNIVERSITY  
COLLEGE OF ENGINEERING

Dissertation

**TOPICS OF DEEP LEARNING IN SECURITY AND  
COMPRESSION**

by

**XIAO WANG**

B.S., Shandong University, 2015

M.S., Boston University, 2020

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2021

© 2021 by  
XIAO WANG  
All rights reserved

## Approved by

First Reader

---

Sang ("Peter") Chin, PhD  
Research Professor of Computer Science

Second Reader

---

Brian Kulis, PhD  
Associate Professor of Electrical and Computer Engineering  
Associate Professor of Systems Engineering  
Associate Professor of Computer Science

Third Reader

---

David A. Castañón, PhD  
Professor of Electrical and Computer Engineering  
Professor of Systems Engineering

Fourth Reader

---

Roberto Tron, PhD  
Assistant Professor of Mechanical Engineering  
Assistant Professor of Systems Engineering

## Acknowledgments

First, I would like to express my sincerest thanks to my advisor, Prof. Peter Chin for his endless support, patience, and encouragement during every stage of my PhD. I am also deeply thankful to my co-advisor, Prof. Brian Kulis for his guidance and help. I would also like to thank my committee members: Prof. David Castañón and Prof. Roberto Tron, as well as Prof. Alex Olshevsky for chairing my thesis defense.

I would like to acknowledge the following funding agencies for sponsoring my research: National Science Foundation (NSF DMS 1737897), National Institute of Health (NIH 5R21EY028381-02) and National Geospatial-Intelligence Agency (NGA HM0476-19-C-0011-P0002).

I feel extremely privileged to work across the systems engineering division and computer science department, and thanks are due to the faculty and staff of both departments. I would especially like to express my gratitude to SE associate division head Prof. Hua Wang, and graduate program manager Elizabeth Flagg, who have done amazing jobs in supporting students and accommodating their needs.

I would also like to thank my labmates: Dr. Jacob Harer, Xiao Zhou, Laura Greige, Louis Jensen, Yida Xin, Peilun Dai, Hieu Le, Allison Mann, Ryan Yu, Trung Dang, Shan Huang, and Andrew Wood. I also would like to thank my collaborators outside BU, especially Dr. Jie Zhang, Siyue Wang, and Dr. Pin-Yu Chen among others. Working with these brilliant minds made research such a joy.

I was also fortunate to have done four internships during my Ph.D which greatly enriched my experience and broadened my knowledge. I would like to thank Prof. Ralph Etienne-Cummings and Prof. Trac Tran for hosting me at Johns Hopkins University, Dr. Kyeong Jin Kim and Dr. Ye Wang for hosting me at MERL, Wei Wang and Dr. Wei Jiang for hosting me at Tencent, and Harshad Kulkarni and Jas Chhabra for hosting me at

Amazon.

Finally, my deepest gratitude goes out to my family and all my friends. Their unconditional love and support have been a pillar for me through the ups and the downs.

Xiao Wang

PhD Candidate

Division of Systems Engineering

# TOPICS OF DEEP LEARNING IN SECURITY AND COMPRESSION

XIAO WANG

Boston University, College of Engineering, 2021

Major Professors: Sang ("Peter") Chin, PhD

Research Professor of Computer Science

Brian Kulis, PhD

Associate Professor of Electrical and Computer  
Engineering

Associate Professor of Systems Engineering

Associate Professor of Computer Science

## ABSTRACT

This thesis covers topics at the intersection of deep learning (DL), security and compression. These topics include the issues of security and compression of DL models themselves, as well as their applications in the fields of cyber security and data compression.

The first part of the thesis focuses on the security problems of DL. Recent studies have revealed the vulnerability of DL under several malicious attacks such as adversarial attacks, where the output of a DL model is manipulated through an invisibly small perturbation of the model's input. We propose to defend against these threats by incorporating stochasticity into DL models. Multiple randomization schemes are introduced including Defensive Dropout (DD), Hierarchical Random Switching (HRS) and Adversarially Trained Model Switching (AdvMS).

The next part of the thesis discusses the usage of DL in security domain. In particular, we consider anomaly detection problems in an unsupervised learning setting using auto-

encoders and apply this method to both side-channel signals and proxy logs.

In the third part we discuss the interaction between DL and Compressed Sensing (CS). In CS systems, the processing time is largely limited by the computational cost of sparse reconstruction. We show that full reconstruction can be bypassed by training deep networks that extract information directly from the compressed signals. From another perspective, CS also help reducing the complexity of DL models by providing a more compact data representation.

The last topic is DL based codecs for image compression. As an extension to the current framework, we propose Substitutional Neural Image Compression (SNIC) that finds the optimal input substitute for a specific compression target. SNIC leads to both improved rate-distortion trade-off and easier bit-rate control.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Organization . . . . .	7
<b>2</b>	<b>Deep Learning Security: Adversarial Attack and Stochastic Defense</b>	<b>10</b>
2.1	Adversarial Threats . . . . .	10
2.1.1	Adversarial Attack . . . . .	10
2.1.2	Adversarial Reprogramming . . . . .	13
2.2	Defense Methods Against Adversarial Attacks . . . . .	13
2.2.1	Stochastic Defense . . . . .	13
2.2.2	Other Defense Methods . . . . .	16
2.3	Defensive Dropout . . . . .	18
2.3.1	Method . . . . .	18
2.3.2	Experiments . . . . .	22
2.3.3	Discussion . . . . .	23
2.4	Toward Better Robustness-Accuracy Trade-off: Hierarchical Random Switch- ing . . . . .	24
2.4.1	Method . . . . .	24
2.4.2	Quantifying Robustness-Accuracy Trade-off . . . . .	27
2.4.3	Experiments . . . . .	28
2.4.4	Discussion . . . . .	33
2.5	AdvMS: a Multi-source Multi-cost Defense Scheme . . . . .	33

2.5.1	Method . . . . .	33
2.5.2	Experiments . . . . .	34
2.5.3	Discussion . . . . .	39
<b>3</b>	<b>Deep Learning Applications in Cyber Security</b>	<b>41</b>
3.1	Classification in Side-channel Analysis . . . . .	41
3.1.1	Side-channel Analysis . . . . .	41
3.1.2	Method . . . . .	43
3.1.3	Experiments . . . . .	44
3.2	Anomaly Detection in Side-channel Analysis . . . . .	45
3.2.1	Method . . . . .	45
3.2.2	Experiments . . . . .	48
3.3	Anomaly Detection in Proxy Logs . . . . .	49
3.3.1	Feature Representation of Proxy Logs . . . . .	49
3.3.2	Method . . . . .	50
3.3.3	Experiments . . . . .	52
<b>4</b>	<b>Deep Learning in Spatio-Temporal Compressed Sensing Systems</b>	<b>53</b>
4.1	Spatio-temporal Compressed Sensing by Pixel-wise Coded Exposure . . . . .	53
4.2	Approach . . . . .	54
4.2.1	Extracting Temporal Information from Coded Image . . . . .	54
4.2.2	Temporal to Spatial Feature Conversion . . . . .	57
4.2.3	CNN Receptive Fields . . . . .	58
4.3	Spatial-motion Feature . . . . .	59
4.3.1	Comparing PCE and Naive Down-sampling . . . . .	59
4.3.2	Characteristics of Spatial-motion Feature . . . . .	60
4.3.3	Transferability of Spatial-motion Feature . . . . .	62
4.4	Experiments . . . . .	64

4.4.1	Motion Classification of MNIST Moving Digits . . . . .	64
4.4.2	Motion Classification of Gesture Videos . . . . .	67
<b>5</b>	<b>Deep Learning for Lossy Image Compression</b>	<b>72</b>
5.1	Neural Image Compression . . . . .	72
5.2	Substitutional Neural Image Compression . . . . .	74
5.2.1	Method . . . . .	74
5.2.2	Experiments . . . . .	77
<b>6</b>	<b>Discussions and Concluding Remarks</b>	<b>83</b>
6.1	Connections between Security and Compression . . . . .	83
6.1.1	The Usage of Input Gradient . . . . .	83
6.1.2	The Usage of Compression Models in Anomaly Detection . . . . .	84
6.1.3	The Usage of Randomness in Improving Robustness and Compact- ness of Deep Learning . . . . .	86
6.2	Summary and Future Directions . . . . .	87
	<b>References</b>	<b>90</b>
	<b>Curriculum Vitae</b>	<b>103</b>

# List of Tables

2.1	Test accuracy, attack success rate, and $L_2$ norm using SAP against C&W attack on CIFAR-10. . . . .	22
2.2	Input gradient standard deviation and mean defense rate (1 - attack success rate) under PGD attack of multiple strengths. . . . .	27
2.3	Mean DES of different defense methods . . . . .	31
3.1	Classification results on side-channel signals collected from different devices.	45
3.2	Selected features and their descriptions for proxy logs. . . . .	50
4.1	Comparison of classification performance on coded images and on videos. .	67
5.1	Average time consumption per image for generating substitutes and corresponding performance using different number of steps. Program time refers to the total time consumption including overheads such as variable initialization while generation time refers to only time for gradient updates. ICLR2017 is used as the base model. . . . .	82

## List of Figures

2.1	Distribution of adversarial examples for a group of model individuals of the same architecture. Each grid shows the relative positions of adversarial examples to the clean image (the origin) on two randomly selected input dimensions (each sub-figure refers to a different choice of 2 randomly selected dimensions). Here, the distances of adversarial examples on the given x-y plane are normalized. Adversarial examples are generated by CW-PGD attack. . . . .	15
2.2	(a) A standard neural network with 2 hidden layers. (b) A sub-network produced by applying dropout. Units in grey color are dropped from the whole network. . . . .	18
2.3	(a) At training time, a unit presents with probability $p$ . (b) At test time, the unit is always present and the outgoing weights are multiplied by $p$ . . . . .	19
2.4	Probability density of sampled gradients when generating adversarial example using CW attack on CIFAR-10. The same neural network architecture, the same original input image, and the same training dropout rate of 0.7 for the best test accuracy is used throughout (a)~(f). Histogram and the corresponding fitted probability density curve in each color denote one out of $32 \times 32 \times 3$ dimensions in the sampled gradients and include 50 data points. (a)~(e) are from our proposed defensive dropout for different test dropout rate, and (f) is from stochastic activation pruning (SAP). . . . .	21
2.5	(a) Test accuracy and (b) Attack success rate under CW attack on CIFAR-10 dataset using different training dropout rates and test dropout rates. . . . .	23

2.6	An illustration of a HRS-protected model. . . . .	24
2.7	An example of input gradient distribution of stochastic defense models on a randomly selected dimension. We sample the input gradient of each defense for 200 times at the first step of CW-PGD attack. While SAP, dropout and Gaussian noise all yield a unimodal distribution, this trend is less obvious for HRS. . . . .	28
2.8	Attack success rate on CIFAR-10 using (a) PGD, (b) PGD + EOT, (c) PGD + fixed randomness (d) CW-PGD, (e) CW-PGD + EOT, and (f) CW-PGD + fixed randomness. . . . .	30
2.9	Scatter plots of different defenses. Attacks on CIFAR-10 and MNIST are performed with $8/255$ and $64/255$ $\epsilon$ bounds respectively. . . . .	31
2.10	Adversarial reprogramming test accuracy during training a locally connected layer with different kernels as input transform. . . . .	32
2.11	Attack success rate on CIFAR-10 dataset using <i>above</i> :(White-Box) FGSM, PGD, and CW-PGD attacks, <i>below</i> : FGSM + EOT, PGD + EOT, and CW-PGD + EOT attacks . . . . .	35
2.12	Attack success rate on MNIST dataset using <i>above</i> : (White-Box) FGSM, PGD, and CW-PGD attacks, <i>below</i> : FGSM + EOT, PGD + EOT, and CW-PGD + EOT attacks. . . . .	36
2.13	Attack success rate on CIFAR-10 dataset using different (white-box or EOT) PGD attack settings with fixed number of models $M$ in AdvMS . . . .	37
2.14	CIFAR-10 with fixed $\epsilon_{train}$ PGD Attack. Attack success rate on CIFAR-10 dataset using different (white-box or EOT) PGD attack settings with fixed adversarial training strength $\epsilon_{train}$ in AdvMS . . . . .	38

2.15	Robustness-Accuracy-Memory trade-off on CIFAR-10 dataset. (a) Set $\epsilon_{attack} = 8/255$ for PGD attack. (b) Set $\epsilon_{attack} = 8/255$ for PGD attack + EOT (EOT: $n=10$ ).	40
3.1	Side-Channel Analysis gleans information from external signals.	41
3.2	Data collection for side-channel analysis.	44
3.3	Five-state signals sampled from a Sieman's PLC	46
3.4	Anomaly detection system based on auto-encoders.	47
3.5	ROC curve in detecting the anomalous machine state of a Sieman's PLC using an auto-encoder.	48
3.6	An illustration of a 1D convolutional auto-encoder used for Proxy log anomaly detection.	51
3.7	ROC curve of anomaly detection in proxy logs.	52
4.1	Four coded images output from a PCE camera sampled at 7.1 FPS [Zhang et al., 2016]; 14 images are reconstructed from each coded image resulting in 100 FPS reconstruted videos.	55
4.2	Conventional information extraction (left) vs. our approach (right).	56
4.3	PCE images with different motion speeds.	58
4.4	CNN model architecture on MNIST coded images.	60
4.5	Coded images (left) and corresponding sensitivity maps (right) in horizontal motion classification (left v.s. right).	62
4.6	Classification accuracy using local information.	63
4.7	Examples of training coded images from MNIST dataset and testing coded images from Quick, Draw! dataset. The CNN models achieve 93.1% and 94.0% accuracy of recognizing moving direction horizontally and vertically on Quick, Draw! although they were trained on MNIST moving digits.	64
4.8	An example of generated training videos and coded images of the videos.	66

4.9	(A): A comparison of CS coded images, average images and random images compressed from the same video clip. (B): A comparison of classification accuracy using coded images, average images and random images. . . . .	68
4.10	CNN model architecture on 20BN-Jester coded images. This architecture follows VGG-16 net with varied width of the last fully connected layer and output layer. . . . .	69
4.11	Visualization of activation maps of convolutional filters that are sensitive to the motion region. This indicates that some filters learned to identify temporally transformed spatial features after training. . . . .	71
5.1	<b>Left:</b> compression and reconstruction by a specific compression model can be viewed as a 2-step mapping. If there exists an $x'_0$ such that it is mapped to $\hat{x}_0$ that is closer to $x_0$ with $R(x'_0) < R(x_0)$ , then better compression can be achieved by simply substituting $x_0$ with $\hat{x}_0$ . <b>Right:</b> consider $\lambda D(\cdot, x_0) + R(\cdot)$ as a function of the input image. The best compression performance is achieved at the global minimum of this function. . . . .	75
5.2	R-D curves with and without SNIC (averaged over the Kodak dataset). . . . .	78
5.3	Achievable range of bit-rate control with different baseline models and target metrics. Points in the same color are generated with the same model instance using Eq. 5.4. The original R-D curve (blue line) is given by varying $\lambda$ at training phase. The R-D curves are averaged over the Kodak dataset. . . . .	80
5.4	Substitutional image clips that target different bit-rates and their reconstructions. Substitutional images are generated from the same original image and the same compression model instance. . . . .	81

## List of Abbreviations

AdvMS	.....	Adversarially Trained Model Switching
ASR	.....	Attack Success Rate
AUC	.....	Area Under Curve
B-LSTM	.....	Binary Long Short-Term Memory
BPP	.....	Bit Per Pixel
CNN	.....	Convolutional Neural Network
CR	.....	Compression Ratio
CS	.....	Compressed Sensing
CW	.....	Carlini-Wagner
DD	.....	Defensive Dropout
DL	.....	Deep Learning
EOT	.....	Expectation Over Transformation
FGSM	.....	Fast Gradient Sign Method
GDN	.....	Generalized Divisive Normalization
HRS	.....	Hierarchical Random Switching
LSTM	.....	Long Short-Term Memory
MS-SSIM	.....	Multi-Scale Similarity Structural Index Measure
MSE	.....	Mean Squared Error
NIC	.....	Neural Image Compression
PCA	.....	Principal Component Analysis
PCE	.....	Pixel-wise Coded Exposure
PGD	.....	Projected Gradient Descent
PSNR	.....	Peak Signal-to-Noise Ratio
R-D	.....	Rate-Distortion
RCNN	.....	Recurrent Convolutional Neural Networks
RF	.....	Random Forest
RIP	.....	Restricted Isometry Property
RNN	.....	Recurrent Neural Network
ROC	.....	Receiver Operating Characteristic
SAP	.....	Stochastic Activation Pruning
SCA	.....	Side-Channel Analysis
SNIC	.....	Substitutional Neural Image Compression
SNR	.....	Signal-to-Noise Ratio

SOM	.....	Self-Organizing Map
SVM	.....	Support Vector Machine
T-to-S	.....	Temporal-to-Spatial
VAE	.....	Variational Auto-Encoder

# Chapter 1

## Introduction

### 1.1 Background

Deep learning has revolutionized the field of machine learning and led to impressive improvements on a number of benchmarks in computer vision [Russakovsky et al., 2015, Krizhevsky et al., 2017, Zhao et al., 2019, Chahal and Dey, 2018], speech recognition [Amodei et al., 2016, Hannun et al., 2014, Noda et al., 2015, Graves et al., 2013], natural language processing [Young et al., 2018, Collobert and Weston, 2008, Goyal et al., 2018, Brownlee, 2017], and automatic game playing [Justesen et al., 2019, Guo et al., 2014, Mnih et al., 2013, Silver et al., 2016]. Besides outperforming classical machine learning methods in the aforementioned fields, deep learning also provides remarkable solutions in many advanced areas such as analysing effects of mutations in DNA [Xiong et al., 2015] and reconstruction of brain circuits [Helmstaedter et al., 2013]. Especially, with the evolution of deep learning platforms [Abadi et al., 2016, Collobert et al., 2002, Chen et al., 2015] and availability of high performance hardware [Jouppi et al., 2018, Zhang et al., 2015a], real world applications and services based on deep neural networks are growing rapidly. Typical examples includes cloud computing based AI services from commercial players like Google and Alibaba and popular voice controllable systems like Apple Siri and Amazon Alexa.

Extensive use of deep learning also exists in safety and security-critical environments, such as self-driving cars [Sallab et al., 2017, Grigorescu et al., 2020], malware detection [Yuan et al., 2014, Kolosnjaji et al., 2018] and robotics [Pierson and Gashler, 2017, Nath

and Levinson, 2014], where malfunction of a deep learning model will lead to serious consequences. Therefore, the security and integrity of deep learning have posed great concern to researchers in the past few years. Security threats to deep learning have various forms and goals. Typical attacks to deep learning models includes model extraction [Tramèr et al., 2016, Oh et al., 2019] that attempts to duplicate a model through provided interfaces, model inversion [Long et al., 2018, Liu et al., 2018b] that attempts to infer training data from trained models, poisoning attack [Liu et al., , Gu et al., 2019] that seeks to downgrade prediction accuracy by polluting training data, and adversarial attack [Szegedy et al., 2013, Carlini and Wagner, 2017b] that aims at fooling a trained model to make erroneous predictions.

Among these attacks, adversarial attack is particularly harmful to deep learning models in the security domain. First discovered by Szegedy et al. [Szegedy et al., 2013], it is shown that a well-performing deep neural network can be misled into making an arbitrary wrong prediction by adding a small perturbation of the model’s input. This adversarial perturbation can be crafted effectively while being small enough so that the perturbed input example has no perceptual difference to humans compared with the original input example. Besides the aforementioned threats, adversarial reprogramming [Elsayed et al., 2018] is another recently proposed security threats where adversaries target to steal computation of a target model by building their applications on top of it, even though the target model is trained for a different task. That is, a model trained for a particular task can be reprogrammed to perform another task, by training input and output transformation networks that bridge these two tasks. By doing so, the attacker can transfer part of computation burdens to models that are publicly available in an unnoticeable way.

Another challenge of deep learning relates its enormous parameter size. For example, Megatron-LM [Shoeybi et al., 2019], a recently developed transformer model, has 8.3 billion parameters. Even a toy neural network usually contains millions of parameters, which

can be problematic when it faces limited computation resources, transmission bandwidth or power budget. This issue is especially significant for deep learning models used for video processing. Conventionally, in order to capture the inter-frame spatio-temporal information of a video clip, a model’s architecture needs to be sophisticated enough to handle spatial and temporal patterns at the same time. For instance, 3D convolutional networks [Hou et al., 2019, Lin et al., 2019] and recurrent convolutional networks (RCNNs) [Xu et al., 2016, Zhang et al., 2017] are commonly used for video recognition, which are dramatically more computationally intensive than standard 2D convolutional networks.

In this thesis, we explore approaches that tackle the above mentioned challenges of deep learning associated with its security and compactness. Interestingly, we found randomness plays an important role in tackling both problems. Randomness can be added to a model’s architecture or parameters, which results in a stochastic model, in order to improve its adversarial robustness. Randomness is also the key component for compressed sensing, which reduces model complexity by providing a more compact data representation.

The motivation of using stochastic models to defend against adversarial threats stems from the fact that these threats are model-dependent. For example, the process of generating adversarial examples makes use of input gradients, i.e. gradients of a loss function with respect to input. Note that in the above process, a model is implicitly exploited through computation of the input gradients, so that a model-specific worse-case perturbation of the input example can be crafted. The benefits of replacing a deterministic model with a stochastic model can be interpreted in two aspects. First, the stochasticity makes the weak spots of the model randomized over time. From the attackers’ perspective, in order to ensure successful attacks, the crafted adversarial examples must work generally well for all variants of a stochastic model, which makes it more difficult than attacking a fixed model. Secondly, the variation of a stochastic model also causes stochastic input gradients that may point to a less beneficial direction for performing gradient update. Although this ef-

fect can be alleviated by sampling stochastic gradient for many times and computing the expected value, it still leads to increased query times of the target model as well as higher computational burdens.

We tackle the second challenge of model compactness with the help of compressed sensing (CS). In compressed sensing theory, a sparse signal can be faithfully recovered from a compressed representation if the sensing matrix satisfies the Restricted Isometry Property (RIP). In particular, random sensing matrices such as Gaussian and Bernoulli matrices, are shown to satisfy RIP condition with exponentially high probability [Baraniuk et al., 2008]. Given the fact that real-world signals such as videos and images are highly redundant and thus have sparse representation under certain bases, compressed sensing works well for these signals. As original signals can be reconstructed from compressed signals, it is natural to wonder if information can be extracted directly from compressed signals. We verify this hypothesis in the context of spatio-temporal compressed sensing, in which a multi-frame video clip is compressed as a single-frame coded image. We train 2D Convolutional Neural Networks (CNNs) on CS coded images and show that they achieve competitive accuracy of recognizing the motion speed and direction compared with models trained with full-frame video in some motion recognition tasks. We interpret this result by introducing a concept of spatial-motion feature, as a consequence of spatio-temporal compressed sensing that can be viewed as a temporal-to-spatial (T-to-S) feature conversion process. These findings shed light on a new way of processing video data using image models which significantly reduce the models' complexity and processing time. This approach fits particularly well to systems where resources are limited and a quick response is required.

The benefits of combining deep learning and compressed sensing are not restricted to data analysis but also to data acquisition. On the one hand, because compressed sensing can be implemented by hardware before the AD conversion, it has economical advantages

in signal processing and data transmission. On the other hand, inferring directly from the compressed data also helps bypass the need for sparse reconstruction of original signals, which is the most computational intensive step in conventional compressed sensing systems. Because sparse reconstruction cannot be implemented efficiently, adaptive sensor adjustments that requires real-time feedback based on sensed information is infeasible. However, if we can infer directly from the compressed data using deep learning models, the reconstruction of original signals can be avoided, and real-time feedback for adaptive compressed sensing becomes possible.

Besides topics regarding to the security and compression aspects of deep learning, this thesis also discusses how deep learning can be applied to conventional tasks in the security and compression domains. Actually, in the above example we have already seen that deep learning benefits compressed sensing systems by enabling fast temporal information retrieval.

There is a rich body of deep learning applications in fields of cyber security, such as malware detection [Alzaylaee et al., 2020, Pascanu et al., 2015, Kolosnjaji et al., 2016], botnet detection [Torres et al., 2016, McDermott et al., 2018, Kolias et al., 2017], spam identification [Tzortzis and Likas, 2007, Mi et al., 2015], and anomaly detection in a variety of different scenarios [Cheng et al., 2016, Maimó et al., 2018, Du et al., 2017a, Garg et al., 2019]. In this thesis, we apply deep learning in analyzing two types of data, side-channel signals and proxy logs. Deep learning applications in the security domain can either be formulated as a classification problem or an anomaly detection problem (here anomaly detection refers to a specific problem setting rather than a particular application domain). The difference between classification and anomaly detection is that both negative (benign) and positive (malicious) training examples are available for classification, but only negative training examples are available for anomaly detection.

Why anomaly detection is a widely used problem setting in security domain? There are

mainly two reasons. One reason has to do with the variety of positive (malicious) examples. For example, there may be many different types of malware, each of which has different behavior. In addition, it is hard to obtain a complete list of them as new types of malware are invented, nor it is necessary to do so as we do not care too much about distinguishing different types of malware. Another reason is that the costs of collecting positive and negative examples are different. Usually, negative (benign) examples are relatively cheap to obtain as they refer to the normal status of a machine. They can either be easily collected from a controlled environment in which anomalies are excluded, or from an uncontrolled environment if we can assume anomalies are rare. To the contrary, labeled positive examples are expensive to obtain as they usually require domain expertise and human effort for labelling. These properties make unsupervised anomaly detection task particularly suitable for cyber security applications.

There are also attempts to apply deep learning models for data compression. In particular, learning based codecs are growing rapidly in the field of image and video compression. Compression tasks can be classified in two different types, lossless compression and lossy compression, and deep learning has been used in both of them. Some typical examples of deep learning based lossless compression methods include [Kingma et al., 2019], [Tatwawadi, 2018], and [Schiopu and Munteanu, 2019]. Deep learning methods for lossy compression are more pervasive, which can be further classified into two different types. The first type of methods embrace traditional coding schemes but use trained neural networks as tools for specific tasks in a coding scheme, such as intra-picture prediction [Li et al., 2018a], inter-picture prediction [Lin et al., 2018, Zhao et al., 2018], cross-channel prediction [Li et al., 2018b] and down/up sampling [Liu et al., 2018a, Li et al., 2018c]. The second type of methods are primarily built upon deep neural networks [Toderici et al., 2015, Ballé et al., 2016, Ballé et al., 2018, Theis et al., 2017]. These schemes usually have an encoder-decoder structure, where both encoder and decoder contains trainable parameters

that are optimized during the training phase.

During the past years, Neural Image Compression (NIC), i.e. image coding based on deep neural networks, progressed rapidly and outperforms many classical engineered codecs such as JPEG [Skodras et al., 2001] and BPG [Bellard, 2015]. However, one major hurdle to pervasive use of NIC regards its inefficiency in bit-rate control where multiple model instances that have different bit-rates are needed. In this thesis, we propose Substitutional Neural Image Compression (SNIC) which finds the optimal substitutional input (denoted as substitutional image) of a specific NIC model for a particular compression target. With SNIC, bit-rate control can be simply conducted by generating substitutional images with different characteristics, without the need for multiple compression model instances.

Last but not least, it is important to note that the security and compression domains of deep learning are not isolated. Rather, there are many interesting connections which we want to identify in the thesis. This is one major reason that we bring both security-related and compression-related topics in the thesis. For example, auto-encoders used in unsupervised anomaly detection are essentially compression models which have the same encode-decoder structure as NIC models. The insight of using compression models for anomaly detection is that as they are trained with negative (benign) data points, the reconstruction error is relatively small for (unseen) negative example but is relatively large when feeding an positive example to the model, by which we can distinguish anomalous examples from normal ones. More discussions regarding to the connections between security and compression will be presented in the rest of the thesis.

## **1.2 Organization**

In Chapter 2, we focus on the security issues of deep learning models. We first introduce two typical security threats, adversarial attack and adversarial reprogramming, to deep neu-

ral networks, where the former targets at fooling a model to make erroneous predictions and the latter targets at re-purposing a model to perform a task that the model is trained for. For adversarial attacks, multiple different attacking schemes are mentioned such as Fast Gradient Sign Method (FGSM), Carlini-Wagner attack (CW) and Projected Gradient Descent attack (PGD). Different defense methods against adversarial attack are also discussed. In this thesis, we focus on a specific type of defense methods called stochastic defenses. For stochastic defenses, conventional deterministic deep learning models are transformed to stochastic models where some or all of model parameters are random variables. Besides discussing the motivation and insights of stochastic defense in general, we also propose three specific randomization schemes. We first introduce defensive dropout, which is a simple extension of the dropout scheme used for regularization. Motivated by pursuing a better security-accuracy trade-off, we next propose Hierarchical Random Switching (HRS). Finally we show how stochastic defense can be used jointly with other defense methods to form a multi-source multi-cost defense. Toward this direction, Adversarially Trained Model Switching (AdvMS) is proposed.

In Chapter 3, we present applications of deep learning in the field of cyber security. Depending on the training data, a task can either be formulated as a supervised learning task such as classification or an unsupervised learning task such as anomaly detection. For the anomaly detection problem, we train auto-encoders with (negative) data examples, and flag an example in the testing phase as anomaly if it triggers a large reconstruction error. We apply this approach to two different domains including side-channel signals and proxy logs.

In Chapter 4, we discuss the interaction between deep learning and compressed sensing. We first introduce spatio-temporal compressed sensing in Sec. 4.1, which down-samples a video scene into a single-frame coded image. Traditionally, full video needs to be reconstructed from the coded image before conducting any analysis. This step is computationally

expensive, as it usually requires iteratively solving an optimization problem. We propose to use Convolutional Neural Networks (CNNs) to extract temporal information directly from the coded images as detailed in Sec. 4.2. Further analysis of spatial-motion features as an explanation of the proposed approach is presented in Sec. 4.3. Experimental results on both synthesis and real-world datasets are presented in Sec. 4.4.

Chapter 5 focuses on deep learning methods for lossy image compression. We first give a generic introduction of neural image compression (NIC) in Sec. 5.1. Two major limitations of current NIC are (a) there is lack of adaptation to the input image as model parameters are fixed after training and (b) bit-rate control cannot be performed easily as it requires to train and store multiple model instances. Motivated by addressing the aforementioned limitations, we propose Substitutional Neural Image Compression (SNIC) in Sec. 5.2. Experimental results that shows its effectiveness in improving compression performance as well as performing bit-rate control with a single model instance are presented.

Finally, discussions and conclusions are summarized in Chapter 6. Sec. 6.1.1 and Sec. 6.1.2 identify two interesting connections between security and compression topics mentioned in this thesis. The first connection relates to the usage of input gradients (i.e. gradients of the loss function with respect to the input), which is used in both adversarial example generation in Chapter 2 and substitutional image generation in Chapter 5. The second connection is the usage of compression models in anomaly detection. Recall that auto-encoders are essentially compression models, but the reconstruction error of auto-encoders provides a good indicator for detecting anomalies. Lastly, Sec. 6.2 summarizes this thesis.

## Chapter 2

# Deep Learning Security: Adversarial Attack and Stochastic Defense

### 2.1 Adversarial Threats

#### 2.1.1 Adversarial Attack

In this section, we introduce adversarial attack in the context of image classification. Although the purpose of using image classification is to provide a concrete example for describing adversarial attacks, it is also important to note that adversarial attacks are not restricted to image classification.

The input images can be denoted as 3-dimensional tensors  $x \in \mathbb{R}^{h \times w \times c}$ , where  $h$ ,  $w$  and  $c$  denote the height, width and number of channels. For a gray-scale image (e.g. an image in the MNIST dataset),  $c = 1$ ; and for a colored RGB image (e.g. an image in the CIFAR-10 dataset),  $c = 3$ . We assume all pixel values in the images are scaled to  $[0, 1]$ , and therefore a valid input image should be inside a unit cube in a high dimensional space.

A deep neural network working as an  $m$ -class classifier can be denoted as a function  $F(x) = y$  that maps an input image  $x$  to  $y = \{y_1, \dots, y_m\}$ . Usually, the softmax function is used on top of the the last layer of a neural network classifier, and thus the components of output vector  $y$  satisfy  $0 \leq y_i \leq 1$  and  $y_1 + y_2 + \dots + y_m = 1$ , and can be treated as the confidence level of assigning a particular label to the input  $x$ . The predicted class  $C(x)$  is the output component that has the highest confidence level, that is,  $C(x) = \arg \max_i y_i$ . Some attack methods also requires the output of a model before the softmax operation, which is

usually referred to as the logits of that model. We use  $Z(x)$  to denote logits of a model and we have  $F(x) = \text{softmax}(Z(x)) = y$ .

Adversarial attacks can be both targeted or untargeted. Given a legitimate input  $x$  with its correct label  $t^*$  and a target label  $t \neq t^*$ , a targeted adversarial attack is to find an input  $x'$  such that  $C(x') = t$ , whereas an untargeted adversarial attack is to find an input  $x'$  such that  $C(x') \neq t^*$ . Targeted attacks are stronger and are of much concerns in security as they force the targeted model predict a particular label chosen beforehand by the attacker. Therefore, we primarily use the targeted attack setting in our experiments to evaluate defense methods.

The adversarial example  $x'$  should be very close to the original input  $x$  based on some measurements of distortion. Common distortion metrics include  $L_0$ ,  $L_2$ , and  $L_\infty$  norms. The usage of a particular distortion metric depends on practical requirements of generated adversarial examples. For instance, if it is requires to have a minimal number of perturbed pixels, then  $L_0$  norm should be used.

We then summarize some of the most used attacking algorithms as follows.

**FGSM.** Fast Gradient Sign Method (FGSM) [Goodfellow et al., 2015] is an one-shot attacking algorithm that generates an adversarial example  $x'$  by taking one step of gradient update. Formally, it can be formulated as

$$x' = x - \epsilon \cdot \text{sign}(\nabla(\text{Loss}_{F,t}(x))). \quad (2.1)$$

As it utilizes the sign of gradients, the maximal change to any pixel is bounded by  $\epsilon$ . FGSM is designed to be fast instead of accurate as it only performs one step of gradient descent.

**CW Attack** Carlini & Wagner (CW) attack [Carlini and Wagner, 2017b] formulates the search of an adversarial example by solving the following optimization problem:

$$\begin{aligned} & \text{minimize}_{\delta} && D(\delta) + c \cdot f(x + \delta) \\ & \text{subject to} && x + \delta \in [0, 1]^n \end{aligned} \quad (2.2)$$

where  $\delta$  denotes the perturbation to  $x$ ,  $D(\delta)$  refers to the distortion metric, and  $f$  is a designed attack objective function for misclassification, and the optimal term  $c > 0$  is obtained by binary search.

In the targeted attack setting,  $f$  takes the following form:

$$f(x + \delta) = \max(\max\{Z(x + \delta)_i : i \neq t\} - Z(x + \delta)_t, -\kappa) \quad (2.3)$$

where  $\kappa$  controls the confidence level of the attacked model predicting the target class.

In practice, the optimization problem is solved by stochastic gradient decent. CW attack is known to be a strong attack that achieves 100% attack success rate on unprotected deep learning models.

**PGD.** Projected Gradient Decent (PGD) [Kurakin et al., 2016, Madry et al., 2017] is an iterative attack that applies FGSM with a small step size  $\alpha$  repeatedly. It controls the distortion of adversarial examples through clipping on the perturbation of  $x$ . For the  $t$ -th iteration, the updating process is:

$$x'_{t+1} = \prod_{x+S}(x'_t + \alpha \cdot \text{sign}(\nabla(\text{loss}(x)))) \quad (2.4)$$

where  $\prod_{x+S}$  refers to projection to  $S$  that is the allowed perturbation in an  $\ell_\infty$  ball centered at  $x$ .

**CW-PGD.** CW-PGD [Athalye et al., 2018] uses the attacking objective function  $f$  from the CW attack and applies it to the PGD attack. Differently from the CW attack, CW-PGD directly controls the distortion of perturbed images by projecting updated images to an  $\ell_\infty$  ball of radius  $\epsilon$ .

### 2.1.2 Adversarial Reprogramming

Adversarial reprogramming [Elsayed et al., 2018] is a recently introduced adversarial threat that aims at reprogramming a deep learning model trained for task  $T_a$  to perform another task  $T_b$ . Reprogramming is accomplished by learning an input transformation  $h_f$  and an output transformation  $h_g$  that bridge the two tasks  $T_a$  and  $T_b$ . Assuming the model that performs  $T_a$  is accessible, the computational cost after applying adversarial reprogramming for performing task  $T_b$  only depends on  $h_f$  and  $h_g$ , so that the attacker transfers part of the computational cost for performing task  $T_b$  to the model for  $T_a$ . Additionally, it can also result in easier training for the model for  $T_b$ , as functionality of the model for  $T_a$  is exploited.

A reprogramming example is given in [Elsayed et al., 2018], where a classifier trained with the Image-net dataset is reprogrammed to perform MNIST digits classification. The input transform in this example is given by

$$X_{adv} = \tilde{X}_i + \tanh(W \odot M). \quad (2.5)$$

Here  $X_{adv}$  is the adversarial input of the target model;  $\tilde{X}_i$  is a MNIST image enlarged to the Image-net image size via zero padding;  $W$  contains trainable weights and  $M$  is a mask wiping out the position of  $W$  that corresponds to the MNIST image in  $X_{adv}$ . The output transform maps the first 10 categories of Image-net labels to MNIST labels. The reprogrammed classifier achieves 97.52% testing accuracy in MNIST classification after training.

## 2.2 Defense Methods Against Adversarial Attacks

### 2.2.1 Stochastic Defense

Stochastic defense, i.e. using stochastic deep learning models to defend against adversarial attacks is the main focus of this chapter. Several different randomization schemes for

deep neural networks, including Defensive Dropout, Hierarchical Random Switching and Adversarially Trained Model Switching, are proposed in the following sections. In this section, we seek to give a generic introduction to stochastic defense, the motivation and intuition behind it, and related works in this field.

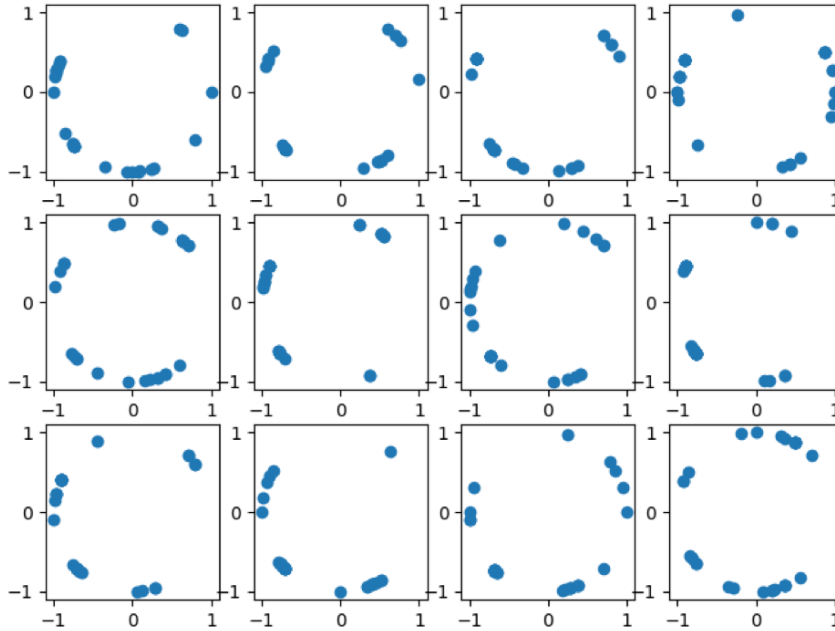
Stochastic models are deep learning models where some or all model parameters are random variables. Consequently, the outputs, as well as input gradients, also become random variables. Since stochastic models are ever-changing, they prevent the attacker from exploiting the weak spots due to a fixed model structure. We are particularly interested in stochastic defense because of the following reasons: (a) they exhibit promising robustness performance; (b) they are easily compatible with typical neural network training procedures; and (c) they do not depend on specific adversarial attacks for training robust models, such as adversarial training [Goodfellow et al., 2015, Madry et al., 2017].

### **Motivation of Using Randomized Models for Defense**

Why randomness can be useful in defending against adversarial attacks? Here we conduct two sets of experiments, and report some insightful observations.

First, we find that if we move toward a random direction from the original input example, the chance of finding an adversarial example is actually very low. The changes to the output when moving toward a random direction from the original example are very small. In contrast, when comparing to the changes of an adversarial direction made by interpolating between a pair of original example and adversarial example found by performing targeted CW-PGD attack, the logit of the target class has a notable increase. This indicates that deep neural networks are robust in general, but the lack of worst-case robustness makes them vulnerable to adversarial attacks.

Second, we find that when training models with the same network architecture but with different weight initialization, each model has its own vulnerable spots. Figure 2-1 shows the adversarial examples found by the same attack for different model instances.



**Figure 2-1:** Distribution of adversarial examples for a group of model individuals of the same architecture. Each grid shows the relative positions of adversarial examples to the clean image (the origin) on two randomly selected input dimensions (each sub-figure refers to a different choice of 2 randomly selected dimensions). Here, the distances of adversarial examples on the given x-y plane are normalized. Adversarial examples are generated by CW-PGD attack.

These models all have the same architecture, training process and performance in terms of test accuracy and defense capability. However, it is observed that adversarial examples for different model instances exist in different directions from the original example. This indicates that each model has its own characteristics in terms of robustness even though all models trained from the same procedure are identical at the macro level.

Combining these two findings, one can reach an intuitive motivation on why stochastic network defenses can be effective. As adversarial attacks are associated with worst-case model performance and the model vulnerability varies with initial weight randomization, a successful attack on a stochastic model basically requires finding a common weakness that applies to all stochastic model variants. This indicates that finding an effective adversarial example for a randomized network is strictly more difficult than that for a deterministic

model.

### **Related Works of Stochastic Defense**

In this section we briefly describe two stochastic defense approaches that are used as baseline methods in the following sections.

**Stochastic Activation Pruning (SAP).** Stochastic activation pruning (SAP), proposed by Dhillon et al. [Dhillon et al., 2018], randomizes the neural network by stochastically dropping neuron outputs with a weighted probability. Specifically, the probability  $p_j^i$  of pruning the  $j$ -th neuron output at the  $i$ -th layer (denoted as  $h_j^i$ ) in SAP is given by

$$p_j^i = \frac{|h_j^i|}{\sum_{k=1}^{a^i} |h_k^i|}. \quad (2.6)$$

After pruning, the remaining neuron outputs are properly scaled up depending on the number of pruned neurons.

**Gaussian Noise.** Liu et al. [Liu et al., 2017] introduce randomness to the network by adding Gaussian noise before each convolutional layer. Gaussian noise is added to the network in both training and testing phases. The authors suggest to use different noise deviations for the input convolutional layer and other convolutional layers, which they refer to as "init-noise" and "inner-noise" respectively.

## **2.2.2 Other Defense Methods**

### **Adversarial Training**

Adversarial training [Madry et al., 2017] uses adversarial examples to train a model. Although it is a general consensus that adversarial training improves the robustness of the network against these adversarial examples, the drawbacks of this method are also apparent. It requires seeking for the worst-case adversarial examples for each training data point at each step of training. Therefore, adversarial training is often referred as a brute-force

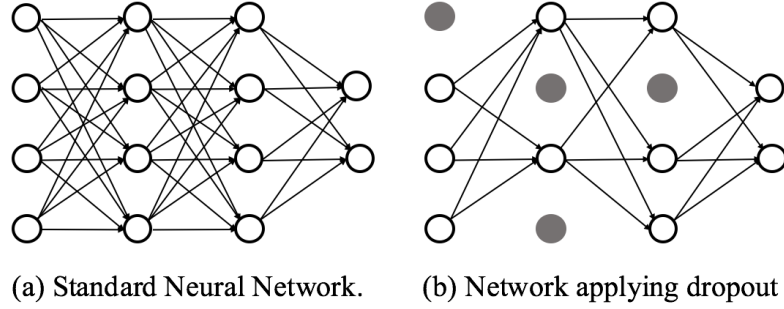
strategy.

### **Input Rectifying**

The key idea of input rectifying is to add a filter in front of the model to remove potential adversarial perturbations of the input images. A typical work of this kind is done by Das et al. [Das et al., 2017] who used JPEG compression to remove the high frequency components from input images, which are shown to be more likely to contain adversarial components. Bhagoji et al. [Bhagoji et al., 2018] proposes to compress the input data using Principal Component Analysis (PCA) for adversarial robustness. However, Xu et al. [Xu et al., 2017] noted that the compression often harms the testing accuracy, as it corrupts the structure of the input.

### **Detection**

Researchers also try to recognize adversarial examples by exploiting the artifacts introduced when adding adversarial perturbations, and then reject them from further processing of the network. Different detection strategies have been developed by making use of statistics of the input images [Meng and Chen, 2017], statistics of intermediate activations [Lu et al., 2017], or by training classifier with one additional class for adversarial examples [Grosse et al., 2017]. Although they claim high detection accuracy on adversarial examples, they are vulnerable to counter-counter measures since the attacker can generate corresponding adversarial examples designed to bypass the detection [Carlini and Wagner, 2017a].



**Figure 2-2:** (a) A standard neural network with 2 hidden layers. (b) A sub-network produced by applying dropout. Units in grey color are dropped from the whole network.

## 2.3 Defensive Dropout

### 2.3.1 Method

#### Dropout Preliminaries

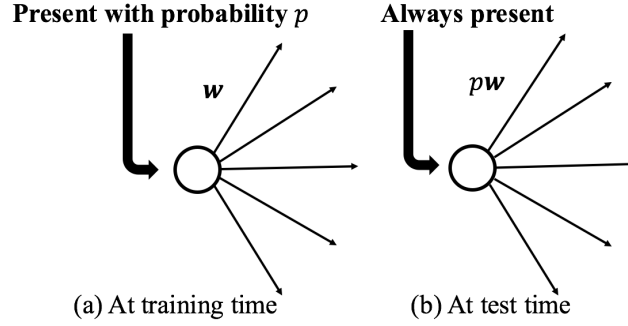
Motivated by the mixability theory in evolutionary biology [Livnat et al., 2010], dropout was proposed as a regularization method in machine learning to prevent the overfitting issue with limited training data [Srivastava et al., 2014]. The term “dropout” refers to dropping out units (hidden and visible) in a neural network. By dropping out a unit, the unit along with all its incoming and outgoing connections are temporarily removed from the network. Fig. 2-2 shows the application of dropout to a neural network amounts to sampling a sub-network from it.

The feedforward operation of a standard neural network can be described as:

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \mathbf{y}^{(l)} + b_i^{(l+1)} \quad (2.7)$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \quad (2.8)$$

where  $\mathbf{y}^{(l)}$  denotes the vector of outputs from layer  $l$ ,  $\mathbf{z}^{(l)}$  denotes the vector of inputs to layer  $l$ ,  $\mathbf{W}^{(l)}$  and  $\mathbf{b}^{(l)}$  are weight matrix and bias vector of layer  $l$ , and  $f$  is any activation



**Figure 2.3:** (a) At training time, a unit presents with probability  $p$ . (b) At test time, the unit is always present and the outgoing weights are multiplied by  $p$ .

function. With dropout, the feedforward operation becomes:

$$r_j^{(l)} \sim \text{Bernoulli}(p) \quad (2.9)$$

$$\tilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} \odot \mathbf{y}^{(l)} \quad (2.10)$$

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^{(l)} + b_i^{(l+1)} \quad (2.11)$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \quad (2.12)$$

where  $\mathbf{r}^{(l)}$  is a vector of independent Bernoulli random variables, each with probability  $p$  of being 1, and  $\odot$  denotes an element-wise product.

The purpose of applying dropout is to prevent units from co-adapting too much by combining the predictions of many sub-networks with shared weights. However, at test time, it is not feasible to explicitly average the predictions from exponentially many sub-networks. A very simple approximate averaging method is to use a single neural network at test time without dropout, the weights of which are scaled-down versions of the trained weights. If a unit presents with probability  $p$  during training with dropout, the outgoing weights of that unit are multiplied by  $p$  at test time, as shown in Fig. 2.3. In this way, the output at test time conforms to the expected output at training time.

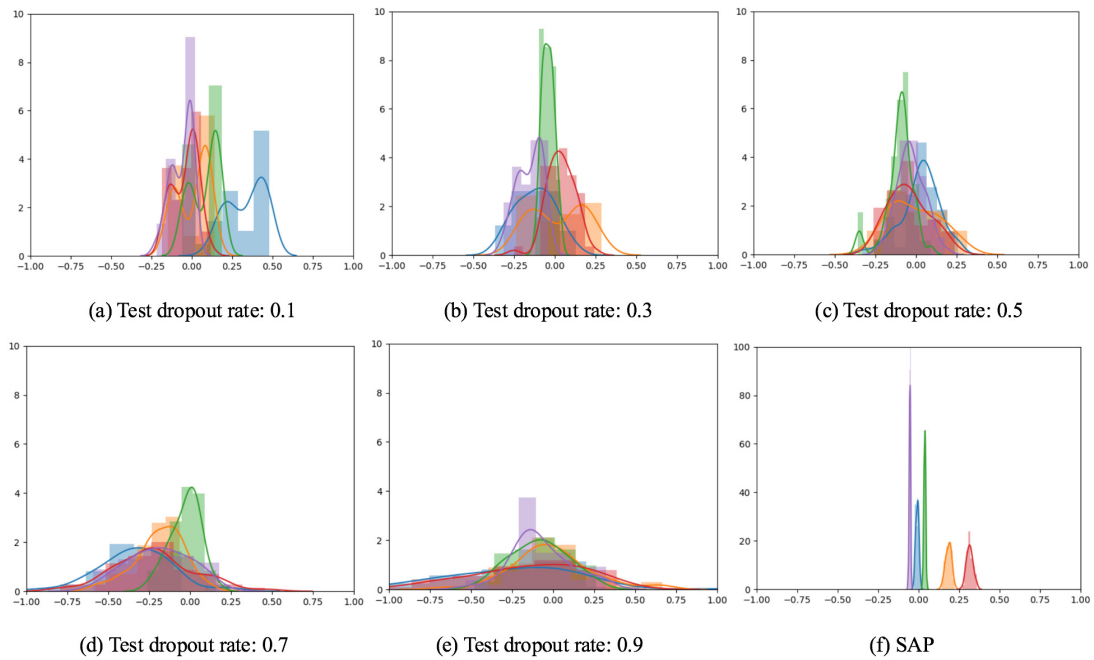
### Defensive Dropout Implementations in Training and Testing Phases

Dropout was originally used as a regularization method. As introducing randomness in the test time helps harden deep neural networks against adversarial attacks, we propose to apply dropout also at the test time as a defense method. If dropout was applied to units in a specific layer during training with dropout rate  $r$ , we are going to apply dropout to the same layer at test time with dropout rate  $r'$ . For each test case, units are dropped with rate  $r'$  and a sub-network is sampled for it. To have roughly the same expected output as the whole neural network at test time, we also need to scale-up the activation functions of the retained units in the dropout layer of the sub-network by  $\frac{1}{1-r'}$ . Please note  $r$  is the dropout rate used in training, and  $r'$  is dropout rate used in testing, which does not need to be as same as  $r$ .

Our work aims at defending against the strongest attacks, i.e., the white-box attacks, that is, the attacker has perfect information about the neural network architecture and parameters. Therefore, when we defend against adversarial attacks, we assume the attacker knows not only the complete neural network model but also the stochastics in the model (i.e., which layer applied dropout and the dropout rates  $r$  and  $r'$ ). Specifically, when the attacker generates adversarial examples by solving an optimization problem based on stochastic gradient descent, the gradients are calculated in a similar manner as training with dropout i.e., using sampled sub-networks and the activation functions scaled-up by  $\frac{1}{1-r'}$ . By doing this, we give the attacker full access to the neural network model and we are able to evaluate our defense method against the strongest white-box attacks.

### Insights of Defensive Dropout

We investigate the mechanism behind the appealing defense effects achieved by the proposed defensive dropout. Fig. 2-4 is plotted for the probability density of uniformly sampled gradients when generating adversarial example using CW attack on CIFAR-10 dataset.



**Figure 2-4:** Probability density of sampled gradients when generating adversarial example using CW attack on CIFAR-10. The same neural network architecture, the same original input image, and the same training dropout rate of 0.7 for the best test accuracy is used throughout (a)~(f). Histogram and the corresponding fitted probability density curve in each color denote one out of  $32 \times 32 \times 3$  dimensions in the sampled gradients and include 50 data points. (a)~(e) are from our proposed defensive dropout for different test dropout rate, and (f) is from stochastic activation pruning (SAP).

**Table 2.1:** Test accuracy, attack success rate, and  $L_2$  norm using SAP against C&W attack on CIFAR-10.

	train 0	train 0.1	train 0.3	train 0.5	train 0.7	train 0.9
Test acc.	72.07%	75.69%	76.39%	78.12%	78.15%	68.35%
CW ASR	54.44%	64.44%	77.78%	78.89%	85.56%	70%
$L_2$ norm	0.504	0.522	0.618	0.498	0.679	0.784

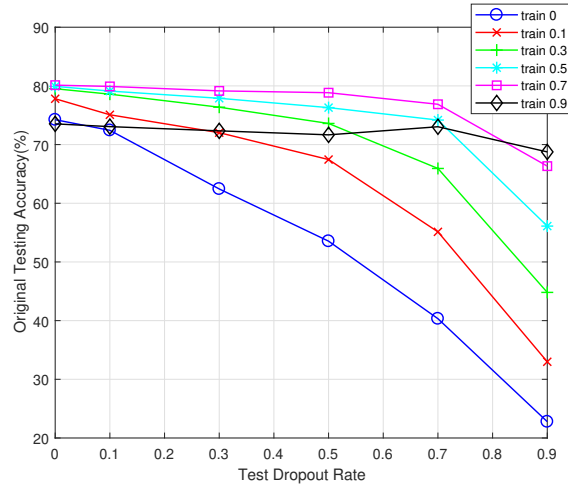
From Fig. 2.4 (a)~(e), with increasing test dropout rate, the probability densities become shorter and fatter, demonstrating increasing variances of the gradients, which is the key for the improved defense effects (decreasing attack success rate) with increasing test dropout rate. The larger variances of the gradients, the more difficult for the attacker to generate effective adversarial examples by using stochastic gradient descent when solving the optimization problem. Fig. 2.4 (f) is the probability densities of the gradients from stochastic activation pruning (SAP) [Dhillon et al., 2018], which shows very small variances of the gradients comparing with our defense method. That explains why defensive dropout outperforms SAP.

### 2.3.2 Experiments

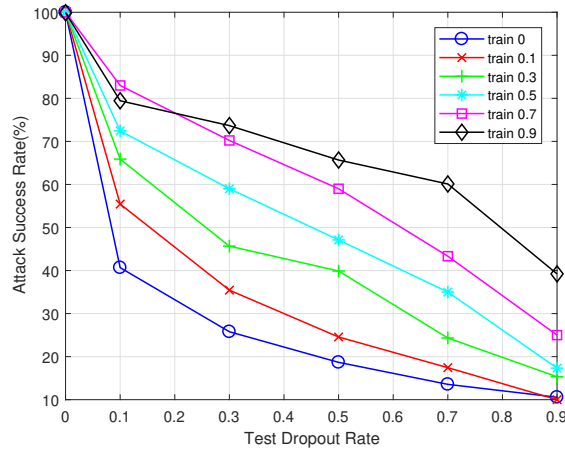
For DNN models in our experiment, we use standard convolutional neural networks with 4 convolutional layers and 2 fully-connected layers. This architecture has been used as standard model in many previous work [Carlini and Wagner, 2017b, Papernot et al., 2016].

We compare defensive dropout with SAP on the defense effects against CW attack using CIFAR-10 dataset. The results of SAP are summarized in Table 2.1. The results of our defensive dropout are summarized through Fig. 2.5. In Table 2.1, we perform SAP on neural network models using different training dropout rates (in columns). The second to forth rows report test accuracy, attack success rate (ASR), and  $L_2$  norm. If we allow test accuracy decrease within 4%, the SAP can reduce the attack success rate from 100% to 77.78% with a test accuracy of 76.39%. From Fig. 2.5, we can observe that at training dropout rate of 0.7 and test dropout rate of 0.7, our defensive dropout can reduce the attack

success rate to 43.33% with a test accuracy of 77%, demonstrating superior defense effects to SAP.



(a) Test Accuracy



(b) CW Attack Success Rate

**Figure 2-5:** (a) Test accuracy and (b) Attack success rate under CW attack on CIFAR-10 dataset using different training dropout rates and test dropout rates.

### 2.3.3 Discussion

Defensive Dropout is a simple but effective randomization method against adversarial attack. In Fig. 2-5, it is shown that the attack success rate can be reduced to around 10%, close

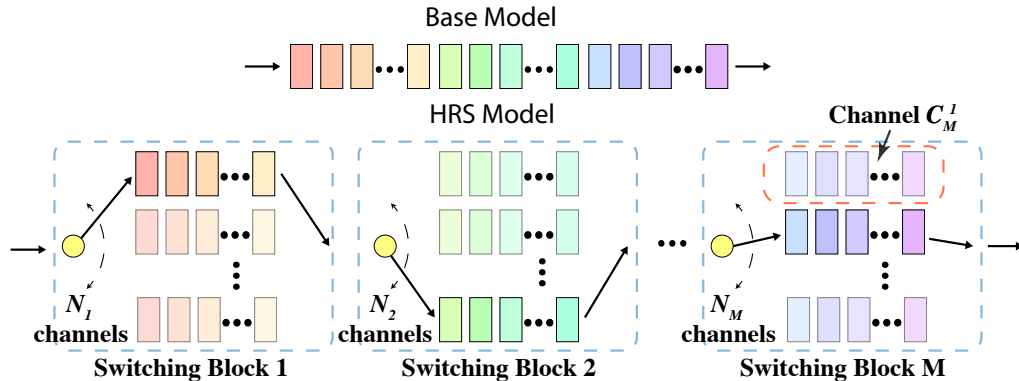
to the accuracy of random guessing. However, one important observation is that stronger defense is always associated with worse test accuracy, indicating a trade-off between adversarial robustness and test accuracy. How to design a stochastic defense method with a better robustness-accuracy trade-off? How to design an evaluation metric that comprehensively evaluate different defense methods in terms of robustness-accuracy trade-off? We will answer these two questions in the next section.

## 2.4 Toward Better Robustness-Accuracy Trade-off: Hierarchical Random Switching

### 2.4.1 Method

#### Hierarchical Random Switching

Hierarchical Random Switching (HRS) divides a base neural network into several *blocks* and replaces each block with a *switching block* which contains a bunch of parallel *channels* with different weights but the same structure as shown in Figure 2-6. HRS features a switcher that randomly assigns the input of the block to one of the parallel channels in the run time. We call the selected channel by the switcher an *active* channel, and at any given time all active channels from each block constitute an *active path* which has the same structure as the base model.



**Figure 2-6:** An illustration of a HRS-protected model.

Intuitively, a HRS-protected model can assure comparable performance to the base model if each possible path is fully functional while its random switching nature prevents the attacker from exploiting the weakness of a fixed model structure. We will introduce a training procedure to ensure full function of each path shortly.

HRS has two main advantages over current stochastic defenses. First, many defenses introduce randomness by dropping neurons or adding noise, leading to undesirable and even disruptive noisy information for model inference, deteriorating accuracy on legitimate data. In contrast, HRS introduces randomness in block switching, where each active path in HRS is trained to have a comparable performance to the base model, which greatly alleviates the issue of significant drop in test accuracy.

Second, HRS is a decentralized randomization scheme. Each variant of HRS has no privilege over others due to random switching. Therefore, it is fundamentally different from Dropout or Gaussian noise where all variations are derived from the base deterministic model, making the base model a centralized surrogate model and potentially leveraged by attackers to bypass these defenses.

### **Training for HRS**

To facilitate HRS model training and ensure the performance of every path, we propose a bottom-up training approach. The training process start with training for the first switching block by constructing  $N_1$  randomly initialized paths. These paths are trained independently to convergence, and after this round of training the weights in the first switching block will be fixed. We then train for the second switching block by constructing  $N_2$  paths with random initialization except for the first switching block. During training for the second block, the switching scheme of the first block is activated, which forces the following upper blocks to adapt the variation of the first block. The training process continues until all switching blocks are trained. We find that by doing so, the training performance is stable, and each channel in a switching block is decentralized. Details of the bottom-up training

approach are summarized in Algorithm 1.

---

**Algorithm 1** Training HRS-protected Model

---

**Require:**

HRS model architecture with  $M$  switching blocks.  $N^i$  denotes the number channels in block  $i$  and  $c_i^j$  denotes the  $j$ 's channel in block  $i$ .

- 1: **for** block  $i \in [1, M]$  **do**
  - 2:   **for** channel  $j \in [1, N_i]$  **do**
  - 3:     Construct a HRS Model with  $M$  blocks:
  - 4:     **for** block  $k \in [1, M]$  **do**
  - 5:       **if**  $k < i$  **then**
  - 6:         Construct  $N_i$  channels with trained channels  $\{c_k^l \mid l \in [1, N_k]\}$ ;
  - 7:         **Freeze** channels in block  $k$ ;
  - 8:       **else**
  - 9:         Construct 1 channel for block  $k$  with randomly initialized weights;
  - 10:        Set all channels in block  $k$  to be **trainable**;
  - 11:       **end if**
  - 12:       Train all trainable channels to convergence;
  - 13:       Save trained channel  $c_i^j$ ;
  - 14:     **end for**
  - 15:   **end for**
  - 16: **end for**
  - 17: **return** A HRS Model with trained channels  $\{c_i^j \mid i \in [1, M], j \in [1, N_i]\}$
- 

### Stochastic Gradients

One unique property of stochastic models is the consequence of stochastic input gradients when performing backpropagation from the model output to the input. By inspecting the mean standard deviation of input gradient under different attack strengths ( $\ell_\infty$  constraint) over each input dimension, we find that it is strongly correlated with the defense performance, as shown in Table 2.2. By simply mounting a white-box attack (PGD), we find that SAP becomes as vulnerable as the base (deterministic) model, and defensive dropout and Gaussian noise have little defense effects. However, defenses that have larger standard deviations of the input gradient, such as the proposed HRS method, are still quite resilient to this white-box attack. For visual comparison, an example of different stochastic models'

Model	Deviation	Defense rate (%)
Base	0	29.18
SAP	0.0343	29.40
Dropout 0.1	0.1143	29.60
Dropout 0.3	0.2295	29.63
Dropout 0.7	0.5186	32.62
Gaussian	0.6413	39.92
HRS $10 \times 10$	0.7376	61.03
HRS $20 \times 20$	0.7888	66.67
HRS $30 \times 30$	0.7983	69.70

**Table 2.2:** Input gradient standard deviation and mean defense rate (1 - attack success rate) under PGD attack of multiple strengths.

input gradient distributions under the same attack is illustrated in Figure 2.7.

## 2.4.2 Quantifying Robustness-Accuracy Trade-off

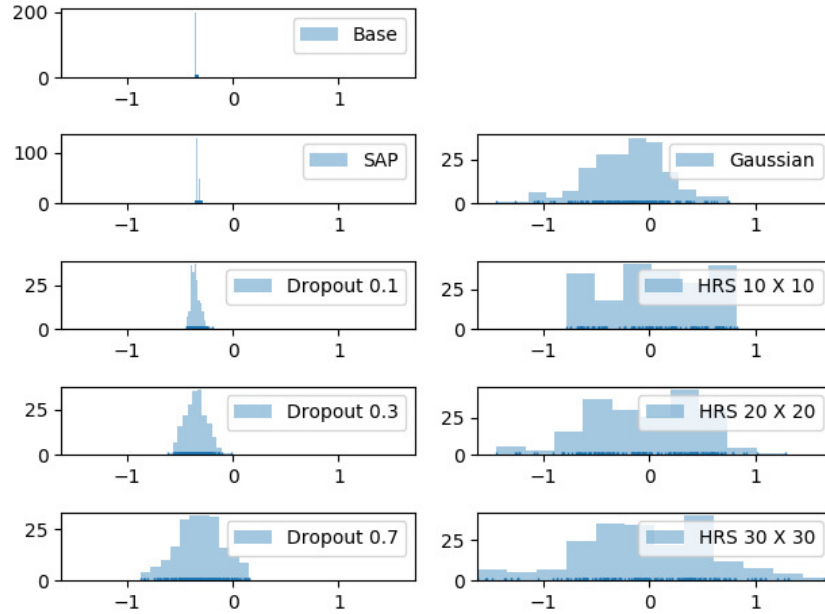
For most current defense methods, there are factors controlling the defense strength. We note that although defense effectiveness can be improved by using a stronger defense controlling factor, it is usually traded with a sacrifice of test accuracy on clean examples.

Therefore, it is worth noting that even under the same norm-ball bounded adversarial attack threat model, comparing different defense methods is not an easy task as they vary in both defense rate and test accuracy. In order to tackle this difficulty, we propose Defense Efficiency Score (DES) as

$$DES_{D,A}(\theta) = \Delta d / \Delta t \quad (2.13)$$

where  $\Delta d$  is the gain in defense rate (percentage of adversarial examples generated by attack  $A$  that fails to fool the protected model using defense scheme  $D$  with strength factor  $\theta$ ) and  $\Delta t$  is the associated test accuracy drop relative to the unprotected base model<sup>1</sup>. Intuitively, DES indicates the defense improvement per test accuracy drop.

<sup>1</sup>In practice, we use  $\Delta d / (\Delta t + \eta)$  where  $\eta$  is a small value (we set  $\eta = 0.002$ ) to offset the effect of noise (e.g. random training initialization) leading to negative  $\Delta t$  when it is close to the origin.



**Figure 2-7:** An example of input gradient distribution of stochastic defense models on a randomly selected dimension. We sample the input gradient of each defense for 200 times at the first step of CW-PGD attack. While SAP, dropout and Gaussian noise all yield a unimodal distribution, this trend is less obvious for HRS.

### 2.4.3 Experiments

#### Setup

For a fair comparison all defenses have to be applied on the same unprotected model (base model). We use two convolutional neural network (CNN) architectures as our base models for MNIST and CIFAR-10 datasets, respectively, as they were standard settings considered in previous works such as [Carlini and Wagner, 2017b, Papernot et al., 2016, Chen et al., 2017].

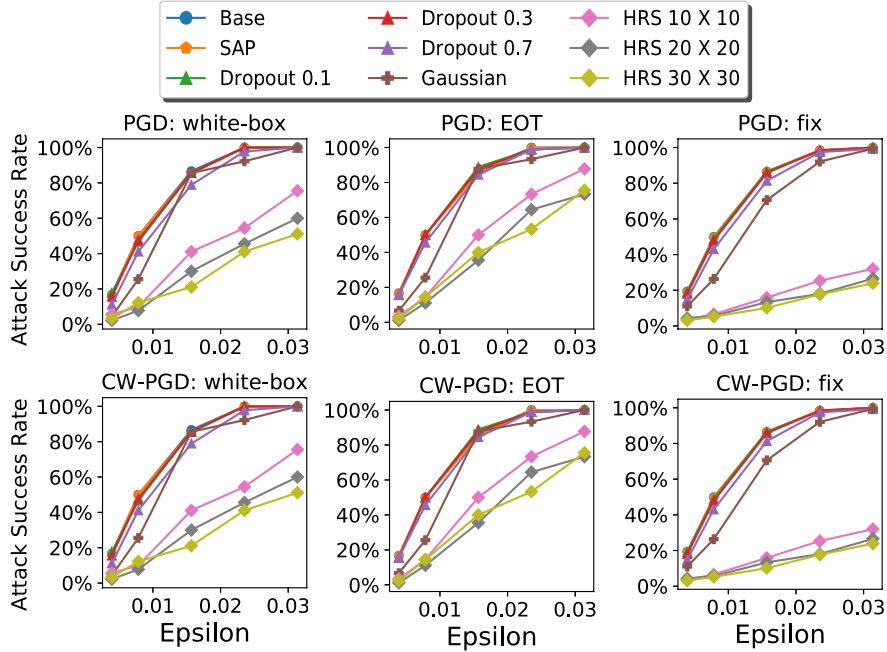
We summarize the implementation details of stochastic defense methods in comparison as follows.

- **SAP** [Dhillon et al., 2018] : Stochastic activation pruning (SAP) scheme is implemented on the base model between the first and second fully-connected layers.

- **Defensive Dropout** [Wang et al., 2018a]: dropout is used between the first and second fully-connected layers with three different dropout rates, 0.1, 0.3 and 0.7 (0.7 is omitted on MNIST as it severely degrades test accuracy).
- **Gaussian Noise**: Following the setting in [Liu et al., 2017], we add Gaussian noise to the input of each convolutional layer in both training and testing phases.
- **HRS** (proposed): We divide the base model structure into two switching blocks between the first and second fully-connected layers. We implement this 2-block HRS-protected model with  $10 \times 10$ ,  $20 \times 20$  and  $30 \times 30$  channels. Note that at any given time, the active path of HRS has the same structure as the base model.

We consider the standard white-box attack setting and two adaptive white-box attacks settings (expectation over transformation (EOT) [Athalye et al., 2017, Athalye et al., 2018] and fixed randomness) for stochastic defenses. The purpose of using EOT and fixed randomness attacks is to show the defense effectiveness is not a consequence of obfuscated gradients. The implementation details of these three attack settings are summarized as follows.

- **White-box Attack**: The adversary uses the stochastic model directly to generate adversarial examples.
- **Expectation Over Transformation (EOT)**: When computing the input gradient, the adversary samples the input gradient for  $n$  times, and uses the mean of gradients to update the perturbed example. We set  $n = 10$  in our experiments as we observe no significant gain when using  $n > 10$ .
- **Fixed Randomness**: Generating adversarial examples using a fixed model by disabling any randomness scheme. For SAP, defensive dropout and Gaussian noise, it is done by removing their randomness generation modules (e.g. dropout layers). For HRS, it is done by fixing an active path.



**Figure 2-8:** Attack success rate on CIFAR-10 using (a) PGD, (b) PGD + EOT, (c) PGD + fixed randomness (d) CW-PGD, (e) CW-PGD + EOT, and (f) CW-PGD + fixed randomness.

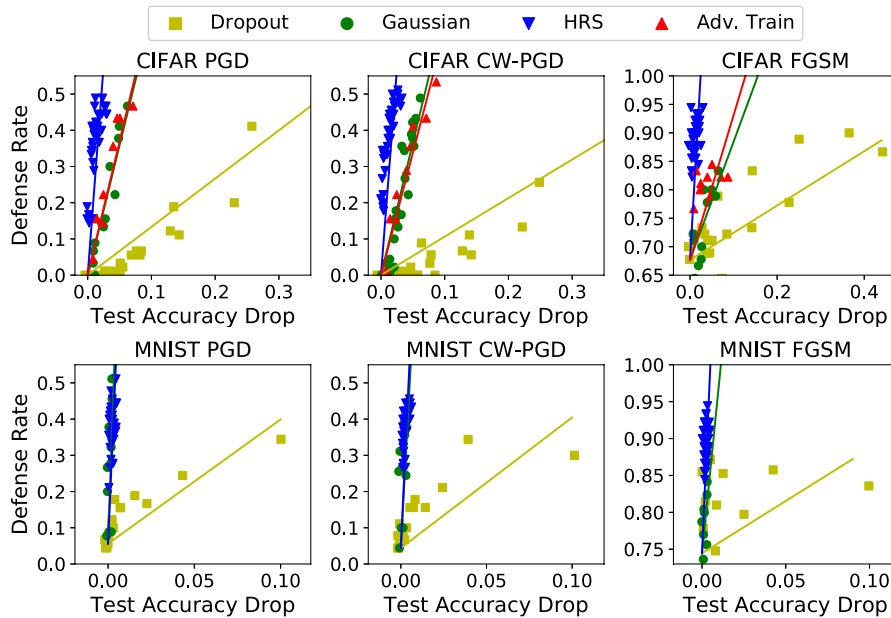
### White-box and Adaptive Attack Analysis

We compare the attack success rate (ASR) of different defenses on CIFAR-10 dataset against PGD and CW-PGD attacks with different strengths (the  $\ell_\infty$  constraint) and under three attack settings in Figure 2-8.

As observed in Fig. 2-8, HRS achieves superior defense performance under all attack settings. The advantage of HRS over other defenses becomes more apparent under stronger attacks such as PGD and CW-PGD, where SAP, defensive dropout and Gaussian noise provide little defense given the same level of test accuracy drop. For example, even with a reasonably large  $\ell_\infty$  perturbation constraint  $8/255$ , on CIFAR-10 PGD and CW-PGD attacks only attain 51.1% and 54.5% ASRs on HRS with only at most 0.48% drop in test accuracy, respectively, while all other stochastic defenses are completely broken (100% ASR), and adversarial training with the same defense rate has 7% additional test accuracy

**Table 2.3:** Mean DES of different defense methods

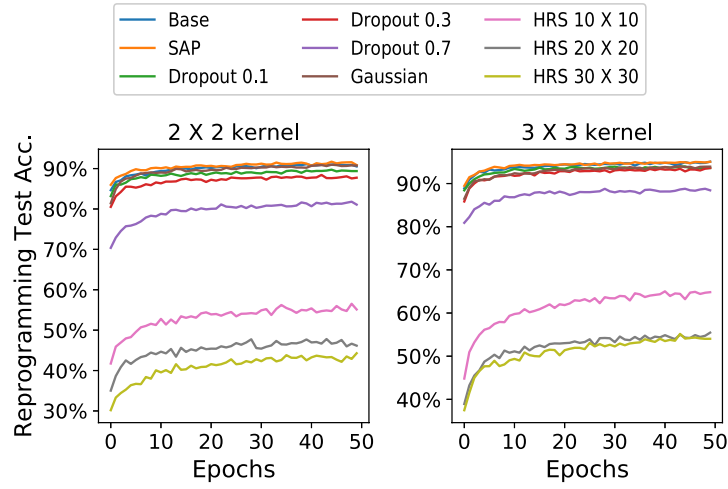
Dataset	Defense	FGSM	PGD	CW-PGD
MNIST	Dropout	11.99	17.94	19.55
	Gaussian	14.90	71.78	<b>82.76</b>
	HRS	<b>36.41</b>	<b>76.64</b>	74.90
CIFAR-10	Dropout	1.09	0.81	0.76
	Gaussian	2.73	6.17	5.47
	Adv. Train	5.05	7.37	6.57
	HRS	<b>32.23</b>	<b>35.43</b>	<b>35.55</b>

**Figure 2.9:** Scatter plots of different defenses. Attacks on CIFAR-10 and MNIST are performed with  $8/255$  and  $64/255$   $\epsilon$  bounds respectively.

drop.

### Defense Efficiency

To characterize the robustness-accuracy trade-offs of different defenses, we compare the DES of different stochastic defense methods in Table 2.3. Our HRS attains the highest DES on CIFAR-10 for all attacks and on MNIST for most attacks. Fig. 2.9 visualizes the trade-off of each defense methods in a scatter plot.



**Figure 2-10:** Adversarial reprogramming test accuracy during training a locally connected layer with different kernels as input transform.

### First Defense against Adversarial Reprogramming

In addition to defending adversarial misclassification attacks, we demonstrate the effectiveness of HRS against adversarial reprogramming [Elsayed et al., 2018]. We use the same base network on CIFAR-10 as the target model to be reprogrammed to classify MNIST images. We use a locally connected layer to perform the input transformation with different kernel sizes and use an identical mapping as the output transformation. The unprotected classifier can easily be reprogrammed to achieve up to 90.53% and 95.07% test accuracy using kernel sizes  $2 \times 2$  and  $3 \times 3$ , respectively, on classifying MNIST images after several epochs of training for the input transformation.

We compare the defenses against adversarial reprogramming of different stochastic defense methods using the reprogramming test accuracy during 50 epochs of training in Figure 2-10. We observe that HRS-protected models can significantly reduce the adversarial reprogramming test accuracy whereas all other defenses are less effective.

#### 2.4.4 Discussion

HRS is designed to provide strong defense against adversarial threat without sacrificing too much test accuracy. However, it is also observed that the marginal gain in defense tend to decrease as we increase the number of channels (see Fig. 2.8). How to prevent defense performance from plateauing? One approach is to combine different types of defensive methods to construct a complex defense method. This topic will be further discussed in the next section.

### 2.5 AdvMS: a Multi-source Multi-cost Defense Scheme

#### 2.5.1 Method

##### Motivation and Principles of Multi-source Multi-cost Defense

We first explain the terms source and cost of a defense scheme using adversarial training [Madry et al., 2017] as an example, as it is considered one of the most advanced defense method in the literature. Adversarial training improves the worst-case robustness of a neural network by incorporating adversarial examples in the training process. As a result, the adversarially trained weights learn to adapt inputs with adversarial perturbations. We thus term the weights of adversarially trained models the *source* of the defense, as they are the surface of performing defense efforts. The cost of adversarial training is the test accuracy of the model on natural examples: the defense can be made stronger by using larger adversarial perturbations for training, at the cost of worse test accuracy on clean examples.

We call defense schemes in this fashion single-source single-cost defenses, and most existing defenses fall into this category. As they are implemented alone, the drawbacks are obvious. Since there is only one scheme boosting robustness, the benefit in robustness tends to plateau with increased defense strength. Another issue is that the cost on a particular factor can easily exceed the acceptable range, making the defense strategy infeasible.

A natural solution to the above limitations is to design multi-source multi-cost defense

schemes. Although many proposed defenses claim to be compatible with others, designing a complex defense is not trivial. We anticipate the following principles of combining multiple defense components as a complex defense. First, components need to have different defense sources. This ensures that the defenses do not interfere with each other, which may cause degraded performance. Second, components need to have different defense costs. This avoids trading robustness with a single cost factor which leads to decreased marginal defense improvement, and at the same time it allows a more flexible combination of costs over multiple factors.

### **A Multi-Source Multi-Cost Defense: AdvMS**

Following the above guidelines, we propose a complex defense AdvMS by merging adversarial training and switching-based stochastic defense. The scheme of model switching [Zhou et al., 2018b] protects the model by using a group of parallel sub-models trained from different random initializations, where the active one that processes model inputs is ever-switching in the run time. It will not degrade test accuracy but increases the memory consumption at testing time. It thus has different source (stochasticity) and cost (memory consumption) compared to adversarial training.

Implementing AdvMS with  $M$  sub-models contains 3 steps: *step 1*, randomly initialize the weights of  $M$  sub-models of the same architecture and train them individually with the same training settings ( $\epsilon_{train}$ ), and training data; *step 2*, save all  $M$  adversarially trained models and construct a pool of parallel sub-models; *step 3*, add the stochastic scheme that randomly activate one model from the pool in each round of inference.

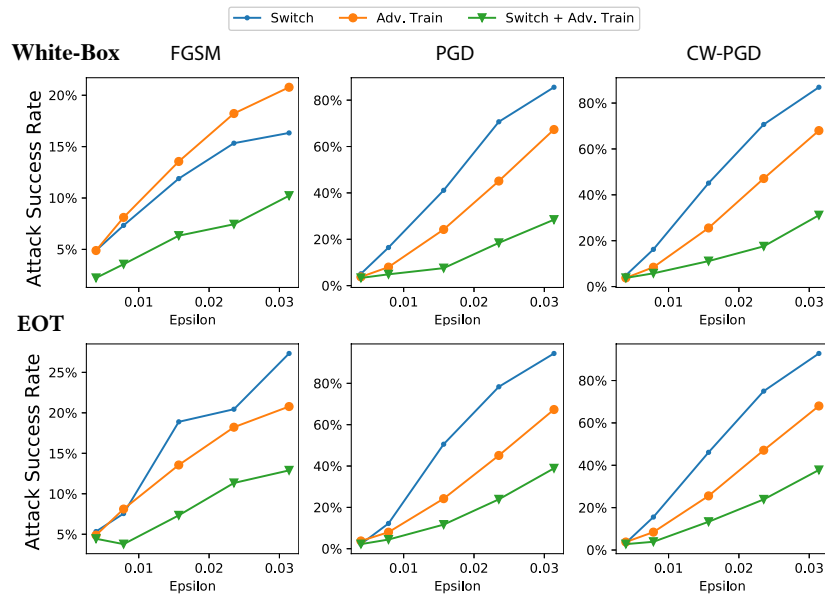
## **2.5.2 Experiments**

### **Defense Against White-box and Adaptive Attacks**

In this experiment, our proposed AdvMS is compared with its building blocks: adversarial training [Madry et al., 2017] and random model switching [Sengupta et al., 2018, Zhou

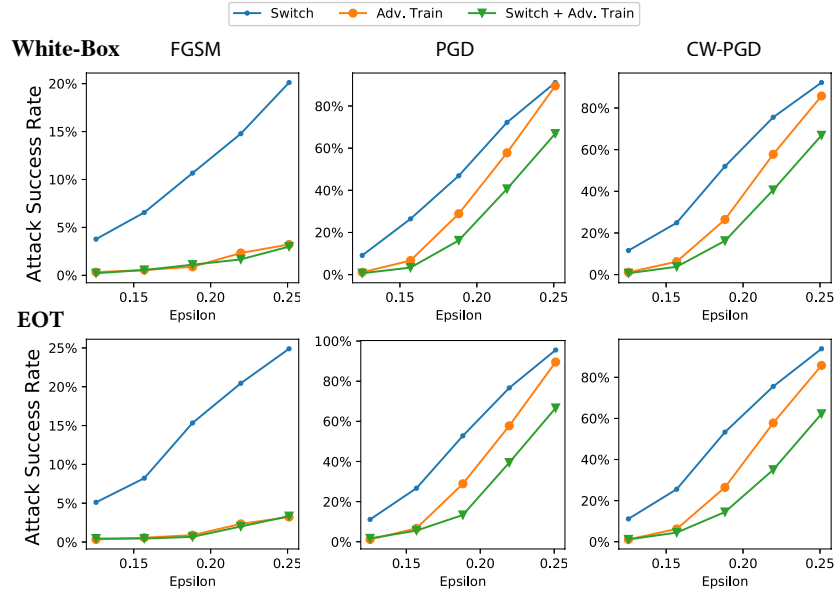
et al., 2018b]. The purpose of this experiment is to show that the proposed complex defense outperforms each of its defending component performed alone.

We perform three attacks: FGSM, PGD, and CW-PGD which are among the strongest baselines in the literature. All attacks are conducted in multiple strengths (controlled by the  $L_\infty$  norm  $\epsilon$  of the perturbation) and in both white-box setting, where we assume attackers know all information of the target model, and EOT (Expectation Over Transformation) [Athalye et al., 2018, Wang et al., 2019] which is a counter-measure of attacks against randomized defense schemes by using the expectation of stochastic gradients. Fig.2.11 and Fig.2.12 summarize the defending performance on CIFAR-10 and MNIST datasets respectively.



**Figure 2.11:** Attack success rate on CIFAR-10 dataset using *above*: (White-Box) FGSM, PGD, and CW-PGD attacks, *below*: FGSM + EOT, PGD + EOT, and CW-PGD + EOT attacks

As we can observe, the proposed AdvMS achieves superior defense performance under all attack settings and performs even better under stronger attacks such as PGD and CW-PGD, where model switching and adversarial training are less effective given the same level



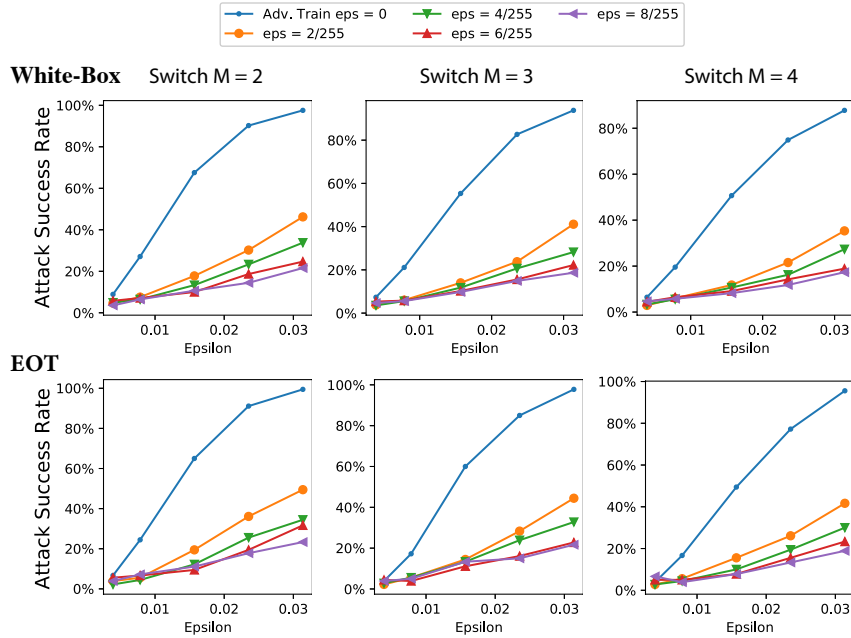
**Figure 2-12:** Attack success rate on MNIST dataset using *above*: (White-Box) FGSM, PGD, and CW-PGD attacks, *below*: FGSM + EOT, PGD + EOT, and CW-PGD + EOT attacks.

of attack strength.

### Quantitative Analysis on Effects of Adversarial Training and Model Switching in AdvMS

In this experiment, we perform a quantitative analysis by controlling each of the two strength factors of AdvMS:  $\epsilon$ , which is the  $\ell_\infty$  distortion of adversarial examples used in training, and  $M$  which is the number of models for switching. This experiment illustrates how enforcing one defending component contributes to boost the performance on the top of the other.

In Fig.2-13, we fix the number of models in AdvMS and compare the robustness performance obtained by changing the  $\epsilon_{train}$  for adversarially trained models. In all cases we can observe the trend that AdvMS trained with larger  $\epsilon_{train}$  value shows stronger resiliency to adversarial attacks in both attack settings. However, it is also observed that although the gap between no adversarial training and adversarial training with  $\epsilon_{train} = 2/255$  is large,

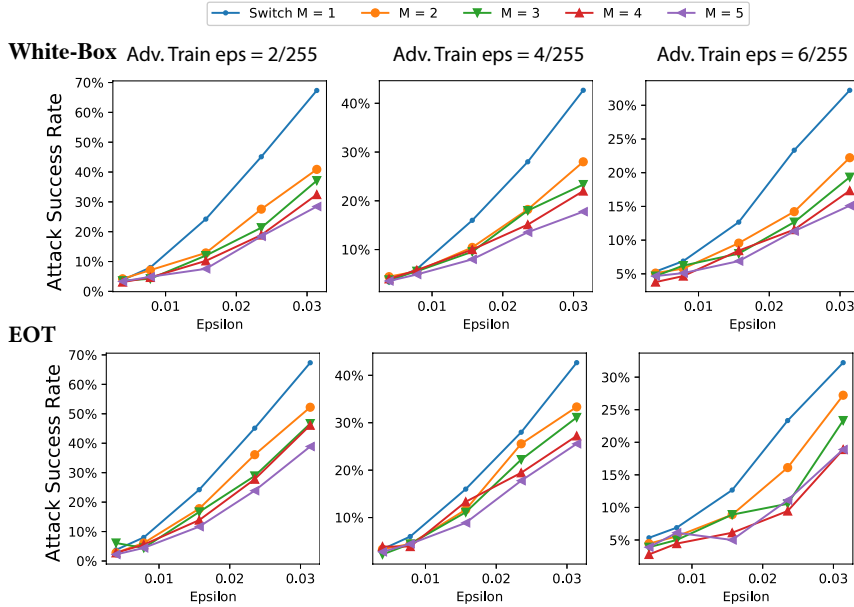


**Figure 2-13:** Attack success rate on CIFAR-10 dataset using different (white-box or EOT) PGD attack settings with fixed number of models  $M$  in AdvMS

the benefits gained by increasing  $\epsilon_{train}$  tend to saturate with large  $\epsilon_{train}$ .

We further perform quantitative analysis on changing the number of sub-models in AdvMS. Fig.2-14 exhibits our findings that AdvMS with more adversarially trained models shows more resistance against adversarial attacks. While the gain in the defense rate when changing the deterministic model ( $M = 1$ ) to a stochastic model with  $M = 2$  is sufficiently large, the marginal gain in defense rate also tends to decrease when  $M$  is further increased.

These experiments illustrate the performance plateau issue as discussed earlier, and verify the importance of using multi-source defenses to break this limitation of single-source defenses.



**Figure 2-14:** CIFAR-10 with fixed  $\epsilon_{train}$  PGD Attack. Attack success rate on CIFAR-10 dataset using different (white-box or EOT) PGD attack settings with fixed adversarial training strength  $\epsilon_{train}$  in AdvMS

**Quantitative Analysis on Robustness-Accuracy and Robustness-Memory Trade-offs of AdvMS**

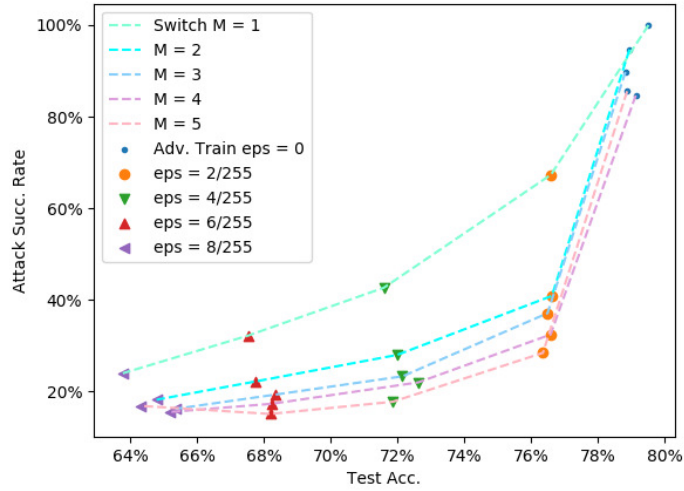
In Fig.2-15 (a) and Fig.2-15 (b), we plot defense effectiveness (quantified by ASRs) against test accuracy of defense models given by different combinations of adversarial training strength  $\epsilon$  and number of switching models  $M$  in white-box and EOT attack settings respectively. Here, points on the same curve are given by AdvMS models with the same  $M$  and have the same amount of memory consumption. Points using the same marker type are given by AdvMS models with the same adversarial training distortion  $\epsilon$  and have similar values of test accuracy.

A defense scheme with ideal robustness-accuracy trade-off will result in a curve approaching the bottom-right corner. As shown in Fig.2-15, when  $M$  increases, the curve moves toward the bottom-right corner which indicates a better robustness-accuracy trade-off. On the other hand, using a large  $\epsilon$  dramatically reduces the ASR, providing a better

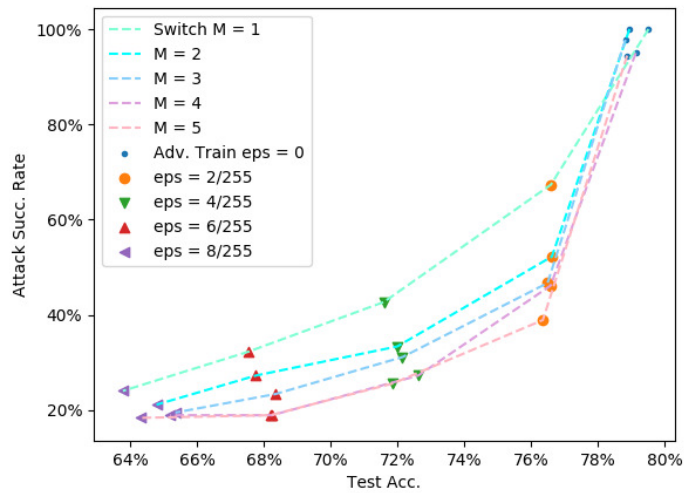
robustness-memory trade-off of the defense at the cost of a degraded test accuracy. It is also clear that the slope of curves become smaller on the left, which corresponds to the defense effectiveness plateau issue as discussed previously.

### **2.5.3 Discussion**

We proposed AdvMS motivated by the need of designing multi-source multi-cost defensive methods against adversarial attack. Comparing to conventional single-source single-cost defenses, AdvMS is advantaged in preventing performance plateau issue and providing a more flexible trade-off among robustness, test accuracy and memory consumption.



(a) PGD



(b) PGD + EOT

**Figure 2-15:** Robustness-Accuracy-Memory trade-off on CIFAR-10 dataset. (a) Set  $\epsilon_{attack} = 8/255$  for PGD attack. (b) Set  $\epsilon_{attack} = 8/255$  for PGD attack + EOT (EOT:  $n=10$ ).

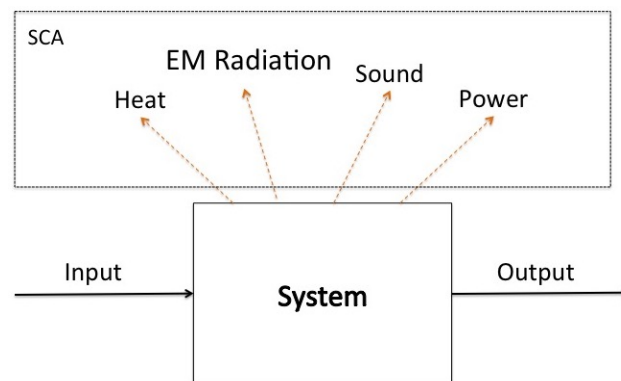
## Chapter 3

# Deep Learning Applications in Cyber Security

### 3.1 Classification in Side-channel Analysis

#### 3.1.1 Side-channel Analysis

Recent works on side-channel analysis (SCA) has shown that any physical implementation of computations inevitably leads to information leakage through physical channels, suggesting that cyber security cannot rely on cryptographic assurances alone. Information can leak through channels such as power consumption [Kocher et al., 1999], electromagnetic emanation [Quisquater and Samyde, 2001], processing time [Kocher, 1996] or even temperature. These side-channel signals, although often highly noisy, can be used to glean sensitive information, endangering the overall security of the system.



**Figure 3.1:** Side-Channel Analysis gleans information from external signals.

SCA was first introduced by Kocher in 1996 [Kocher, 1996], and was aimed at reveal-

ing cryptography keys by exploiting the information leakage obtained from side-channels. With the fast growing popularity of machine learning, more recent publications show that it is possible to apply machine learning methods to SCA. Often these methods outperform more classical attacks. Backes et al. [Backes et al., 2010] used machine learning techniques for acoustic side-channel attacks on printers. Hospodar et al. [Hospodar et al., 2011] applied a Least Squares Support Vector Machine for classifying intermediate bit values in Template Attacks. Lerman et al. [Lerman et al., 2011] performed SCA using three different algorithms: Random Forests (RF), Support Vector Machines (SVM) and Self-Organizing Maps (SOM). Later, Heuser and Zohner [Heuser and Zohner, 2012] used multi-class SVM for an attack on multi-bit values (Hamming weight model). This work was further improved by Bartkewitz and Lemke-Rust [Bartkewitz and Lemke-Rust, 2012], who proposed a new multi-class classification strategy, based on the ordering of the classes. The first neural network-based method was proposed by Martinasek and Zeman [Martinasek and Zeman, 2013], where they classified AES keys.

However, all of those works require hand-crafted features and domain-specific knowledge. As such, the performance of these works critically depends on how well the features are selected. The recent success of deep learning suggests the possibility of avoiding this problem by letting DL models learn the appropriate feature transformations themselves.

SCA has traditionally been used to exploit information and thus break a cryptosystem, but it can also be used for defending purposes. Malware and cyber attacks are dangerous threats to the security of any real system and often utilize sophisticated techniques of avoiding detection by traditional anti-malware software. However, all malware must at some point affect the functioning of the machine. SCA can then be used to monitor and detect these changes at a fundamental level that malware cannot compromise directly. Depending on the assumption of available training data, malware detection can be either formulated as a classification problem or an anomaly detection problem. We present meth-

ods and experiments regarding to classification formulation in this section, and methods and experiments regarding to anomaly detection formulation in the next section.

### 3.1.2 Method

We explored two classification methods, MLP and LSTM. Each classifier was trained by a variant of gradient descent to minimize the standard cross-entropy loss. For testing, performance of the model was evaluated with top-1 testing accuracy.

In our application we trained a 9-layer MLP network with the raw current or voltage signal as input. The Adam [Kingma and Ba, 2014] optimizer was used. We used ReLU [Maas et al., 2013] activation for the hidden layers and softmax for the output layer with a categorical cross entropy loss function. Dropout [Srivastava et al., 2014] was used as regularization.

Recurrent neural networks are networks that contain self-loops, and are useful for processing sequences of data. The output  $h_t$  at point  $t$  is used as an additional input when computing  $h_{t+1}$ . A very popular kind of recurrent neural networks is the Long Short-Term Memory unit (LSTM) [Hochreiter and Schmidhuber, 1997], which adds an additional state  $c_t$  in order to help with the vanishing gradient problem [Le and Zuidema, 2016].

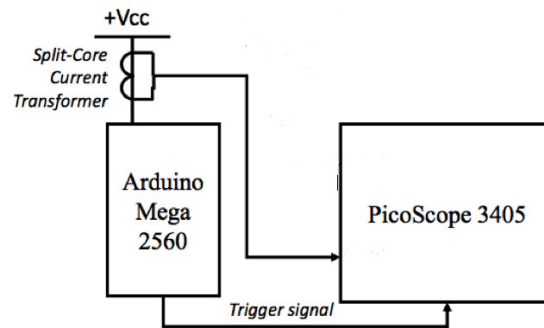
We implemented a bidirectional LSTM (B-LSTM) [Graves and Schmidhuber, 2005] to improve our sequence-based classification results. A B-LSTM contains two LSTM networks; one processes the input sequence from front to back, and the other works in reverse. The outputs of these two LSTM are then concatenated and used as the output of the B-LSTM. This also helps with the vanishing gradient problem and captures information both from the past and future of the current time frame.

Our network consists of a two-layer B-LSTM with 256 neurons in each layer, followed by a fully connected layer also with 256 neurons, and then an output softmax layer. The network was trained using the Adam optimizer and cross-entropy loss.

### 3.1.3 Experiments

We ran three sets of experiments to evaluate the effectiveness of deep learning classifiers across different tasks and devices.

**Experiment 1: Classifying Multiplication with Different Operands.** To illustrate the ability of our models to differentiate even a slight variance in computer states, we construct a binary classification task where the same Multiplication operation is performed with two different sets of operands. The experiment was run on an Arduino board, and the power-supply voltage was recorded (see Fig. 3-2). Each of the 400 traces contains 4096 data points sampled at a rate of 125 MHz. The traces are synchronized so that the recording period aligns consistently with the multiplication operation. Both the MLP and LSTM models receive the full trace as input. In this simple experiment, both models achieved 100% testing accuracy.



**Figure 3-2:** Data collection for side-channel analysis.

**Experiment 2: Detecting Machine States Under Attack.** The next experiment tests binary classification in a more realistic setting. Traces were recorded from a Raspberry Pi; one class of traces is recorded while the board operated normally, the other class is recorded while it was under a botnet attack. Each recording lasted roughly 45 minutes and was sampled at a rate of 30518Hz. The full recording was chopped into non-overlapping 10-second sections to be used as independent data points.

Since the traces are not synchronized to any particular operation, we hope to make our models invariant to shifts through time. Perhaps the simplest method to achieve this invariance is through transforming our data into the frequency domain. The MLP took as input the frequency spectrum of the time signal computed through Fast Fourier Transform (FFT), while the LSTM received a Short-term Fourier Transform (STFT) of each window of the training trace. Performance of MLP and LSTM are given in table 3.1.

**Table 3.1:** Classification results on side-channel signals collected from different devices.

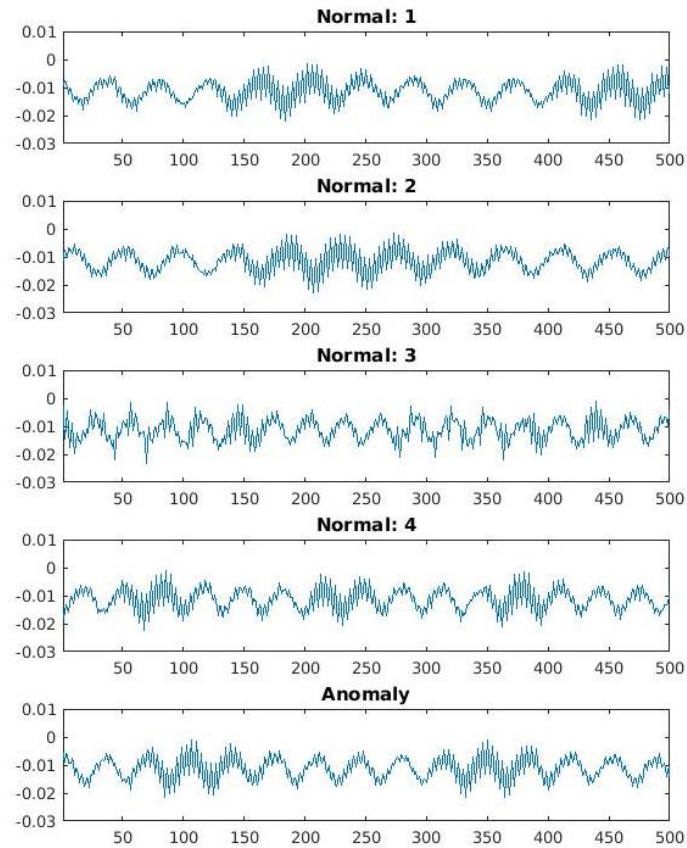
Devices	Classes	Samples	Trace Length	Sync	model	Accuracies %
Arduino	2	800	4096	Yes	MLP	100.00
					LSTM	100.00
Raspberry Pi	2	600	305180	No	MLP	98.84
					LSTM	100.00
Siemen’s PLC	5	975	305180	No	MLP	91.28
					LSTM	89.00

**Experiment 3: Multi-Class Normal and Malicious States.** In addition to distinguishing between normal and malicious states, we hope to also differentiate between several states of operation. The final classification task uses recordings from a Sieman’s PLC in 4 normal machine states and one malicious state under botnet attack (see Fig.3-3 for a visualization of waveforms of all classes). Similarly to before, 30 minute recordings were made at a 30518Hz sampling rate and split into 10-second sections. The same transformations into the frequency domain were performed.

## 3.2 Anomaly Detection in Side-channel Analysis

### 3.2.1 Method

In supervised classification, we assume the training data contains examples of all classes. In contrast, an anomaly detection model only needs training data of the benign class and hopes to identify data points that from a different, arbitrary distribution. In other words, we



**Figure 3.3:** Five-state signals sampled from a Sieman's PLC

hope to learn the distribution of benign data and identify test examples that deviate from this distribution.

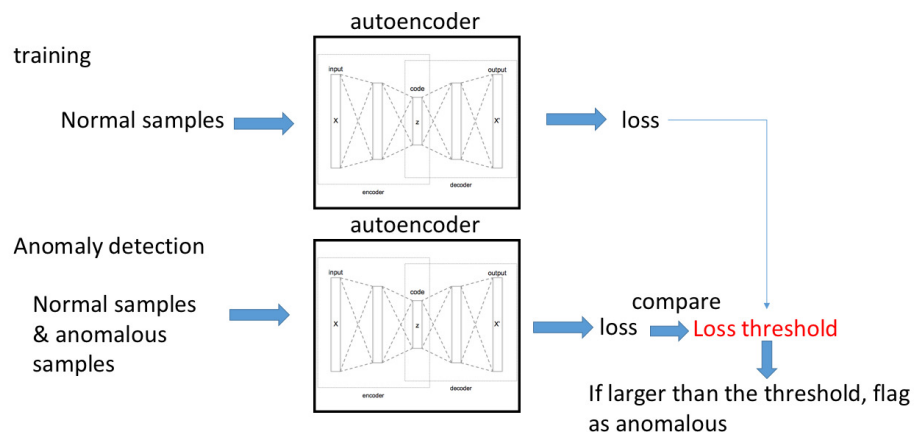
Unsupervised anomaly detection problem is a suitable problem formulation in the field of side-channel analysis. Because of the increasing amount of different attacking methods and advances of anti-detection methods, it is very hard to include data of all malicious attacks and train a classifier in a supervised learning manner.

We thus turn to auto-encoders, a specific type of neural networks used for learning compact representations of data. These models have been used for anomaly detection in many related fields of security including [Sakurada and Yairi, 2014], [Dau et al., 2014], and [Sabokrou et al., 2016], and provide a effect way to detect suspicious data points with

different patterns.

An auto-encoder in its simplest form is similar to an MLP with its output dimension equal to its input dimension. During training auto-encoders try to recover the output as the input, i.e. learn a identity mapping. In the middle of the network, there is a bottleneck of the size of nodes which is significantly smaller than the input and output size. This forces the model to learn a low-dimensional representation of high dimensional data.

We train auto-encoders using benign data, so that it learns to reconstruct a benign data point from a compressed representation. In the testing phase, test data points, including both benign and anomalous data are fed to auto-encoders, and we examine the loss of auto-encoders with input data of both classes. Since the network is only trained on non-anomalous data, the reconstruction error with anomalous data is higher, which provides an effective indicator for anomaly detection. A system diagram describing auto-encoder based anomaly detection systems is given in Fig. 3-4.



**Figure 3-4:** Anomaly detection system based on auto-encoders.

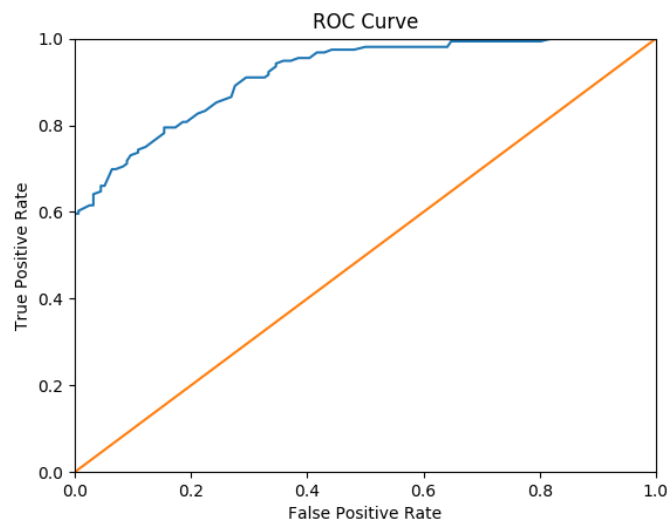
There is one key difference between the conventional usage of auto-encoders and their use for anomaly detection. Since our goal is to distinguish between normal and anomalous data points, we are only interested in the relative difference between the reconstruction errors between these two classes, rather than the reconstruction error itself. Therefore,

sometime we might prefer simple models if they amplify the difference between reconstruction errors of the two classes even though they have higher reconstruction compared with more complicated models.

### 3.2.2 Experiments

We use the Sieman's PLC dataset in the previous section which has four normal machine states and one anomalous machine state for the anomaly detection task. At training time, only data points from the four normal machine states are used for training the auto-encoder. At testing time, the  $\ell_2$  reconstruction error is computed for signals from both normal and anomalous machine states. A threshold of reconstruction error is set, and any example with reconstruction error above the threshold is categorized as an anomalous examples.

We use various values of the threshold and show the Receiver Operating Characteristic (ROC) curve in Fig. 3-5. The Area Under Curve (AUC) of the ROC curve is 0.9166, indicating the proposed method is an effective anomaly detection approach.



**Figure 3-5:** ROC curve in detecting the anomalous machine state of a Sieman's PLC using an auto-encoder.

### 3.3 Anomaly Detection in Proxy Logs

#### 3.3.1 Feature Representation of Proxy Logs

Proxy logs are generated by proxy servers which contain requests made by users and applications on the network. Proxy logs provide an important source for debugging and supervision. However, monitoring proxy logs manually would be an expensive solution, which motivates attempts for automated anomaly detection using deep learning.

Unfortunately, unlike side-channel signals that can be naturally represented as time series as presented in Sec. 3.2, log data are series of log entries, each of which is essentially a character string. Hence, a key challenge in proxy log analysis is to convert proxy logs into some mathematical formats before they can be processed.

One conventional way of extracting information from proxy logs is to design statistical features [Machlica et al., 2017, Gržinic et al., 2013]. Typical examples of statistical features of proxy logs include occurrence of special characters, maximum occurrence-ratio of a character etc. However, these features only describe proxy logs on a very rough level. E.g. information associated with components that have semantic meaning can not be represented in these features. Motivated by modern natural language processing techniques, another way of features representation is treating some tokens in proxy logs as words. However, unlike natural languages that have a finite set of vocabulary, tokens in proxy logs have infinite variations. We handle this by using a fixed size of vocabulary for each token by selecting the most common words from the training data and grouping all other words into a special "other" category.

In this work, we use a combination of the above two methods for feature representation. On the one hand, we use exact values in proxy logs such as HTTP method, file extension, etc. On the other hand, we also use statistical values in a proxy log entry, such as number of upper case letters, number of levels, etc. These selected features are either represented as categorical features containing discrete values or numerical features containing continuous

**Table 3.2:** Selected features and their descriptions for proxy logs.

Feature	Exact/Statistical	Categorical/Numerical	Description
method	exact	categorical	HTTP method
sub_domain_word	exact	categorical	Sub-domain words in URL
domain_word	exact	categorical	Domain words in URL
g_TLD	exact	categorical	Generic top-level domain
c_TLD	exact	categorical	Country code top-level domain
file_extension	exact	categorical	File extension in URL
status	exact	categorical	Status code of HTTP request
nb_level	statistical	numerical	Number of "/" in URL
nb_lower	statistical	numerical	Number of lower case letters in URL
nb_upper	statistical	numerical	Number of upper case letters in URL
nb_numerical	statistical	numerical	Number of digits in URL
nb_special	statistical	numerical	Number of special characters in URL
nb_parameters	statistical	numerical	Number of parameters in URL
nb_bytes	exact	numerical	Number of types in response

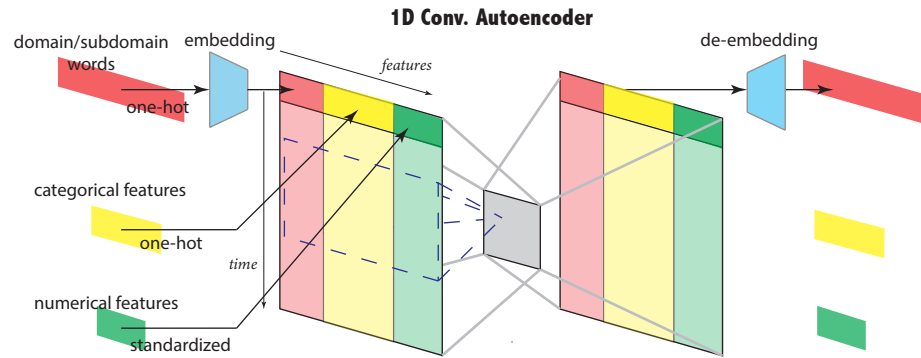
values, depending on the nature of each feature. We summarize the details of the selected features in the Tab. 3.2.

### 3.3.2 Method

#### Domain Words and Sub-domain Words Embedding

Words embedding is a commonly technique to represent words in an efficient way where words with similar meaning have similar vector representation. In proxy logs, domain words and sub-domain words contains natural language words whose semantic meaning may provide important information in anomaly detection. Also, the vocabulary sizes of domain and sub-domain words are very large, where using naive one-hot representation is inefficient.

Therefore, we embed domain words and sub-domain words into a more compact space using a embedding network before they are fed into the auto-encoder. The embedding network contains trainable weights and are optimized together with the auto-encoder during training. Considering domain words and sub-domain words have different vocabularies, there are two separated embedding network for domain words and sub-domain words respectively. Every embedding network is also paired with a de-embedding network. Similarly, de-embedding network also contains trainable weights and are also optimized together with other networks during training.



**Figure 3-6:** An illustration of a 1D convolutional auto-encoder used for Proxy log anomaly detection.

### Pre-processing of Categorical and Numerical Features

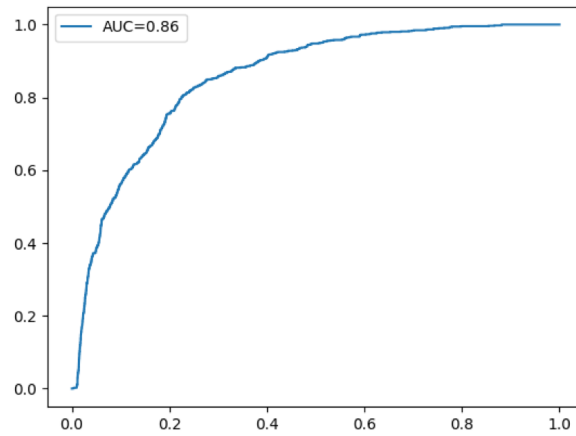
Except domain words and sub-domain words, all other categorical features are represented as one-hot vectors. Numerical features are standardized according to their distributions in the training data.

### 1D Convolutional Auto-encoder

Every log entry is eventually represented as a vector by concatenating all processed features of the log entry. However, certain types of anomalies may require multiple log entries to identify them. For instance, some malwares will periodically access a certain URL. In order to capture temporal correlation among log entries, we use a 1D convolutional auto-encoder for anomaly detection. The 1D convolutional auto-encoder together with feature processing is illustrated in Fig. 3-6.

### Training Loss

Cross-entropy loss is used for categorical features and mean squared error (MSE) is used for numerical features. The overall loss is a weighted sum of all loss terms where the



**Figure 3-7:** ROC curve of anomaly detection in proxy logs.

weight factor associated with loss of each feature is a hyper-parameter that is tuned against the validation set.

### 3.3.3 Experiments

Our proposed approach is evaluated using a collection of real-world proxy logs. The ROC curve is plotted in Fig. 3-7 with an AUC of 0.86.

## Chapter 4

# Deep Learning in Spatio-Temporal Compressed Sensing Systems

### 4.1 Spatio-temporal Compressed Sensing by Pixel-wise Coded Exposure

Pixel-wise Coded Exposure (PCE) [Zhang et al., 2016] is a spatio-temporal CS technique used to downsample a video scene into a single image. To illustrate its principles, let  $\mathbf{X} \in \mathbb{R}^{H \times W \times T}$  represent the pixels value of a video, where  $H$ ,  $W$  and  $T$  indicate the height, width and the number of frames respectively. And let  $\mathbf{S} \in \mathbb{R}^{H \times W \times T}$  represent the on/off state of each pixels' exposures at a given location and time:

$$\mathbf{S}(h, w, t) = \begin{cases} 1 & t \in [t_1^{h,w}, t_2^{h,w}] \\ 0 & otherwise \end{cases} \quad (4.1)$$

Here a 1 indicates that the exposure is turned on and, a 0 indicates that the exposure is turned off.  $t_2^{h,w} - t_1^{h,w}$  denotes the exposure duration of the pixel at  $(h, w)$ . In conventional CMOS image sensors without in-pixel charge storage, a pixel's value must be sampled at the end of its exposure. Thus, within  $T$  frames, a pixel's exposure can only be turned on once.

**Sampling:** The PCE sampling operation can be written as a pixel-wise multiplication of the scene  $\mathbf{X}(h, w, t)$  with the coded exposure pattern  $\mathbf{S}(h, w, t)$ , followed by integration over all frames:

$$\mathbf{Y}(h, w) = \sum_{t=1}^T \mathbf{S}(h, w, t) \cdot \mathbf{X}(h, w, t) \quad \forall w, h. \quad (4.2)$$

Here,  $\mathbf{Y} \in \mathbb{R}^{H \times W}$  is the coded image with pixel dimension  $H$  and  $W$ .

There are various advantages with PCE sampling, including reduction of analog-to-digital conversion speed and power, compression and prolonged pixel exposure that leads to improved image signal-to-noise ratio (SNR).

**Reconstruction:** From the coded image  $\mathbf{Y}$ , we can recover the original scene  $\hat{\mathbf{X}} \in \mathbb{R}^{H \times W \times T}$  using sparse reconstruction by solving the following optimization problem:

$$\min_{\mathbf{a}} \frac{1}{2} \|\mathbf{Y} - \mathbf{S}\mathbf{D}\mathbf{a}\|_2^2 + \lambda \|\mathbf{a}\|_1, \quad (4.3)$$

where  $\mathbf{D} \in \mathbb{R}^{H \times W \times T \times L}$  is an over-complete dictionary learned from training samples over which the video  $X \in \mathbb{R}^{H \times W \times T}$  has a sparse representation  $\mathbf{a} \in \mathbb{R}^L$ .  $\lambda$  is a coefficient controlling the trade-off between sparsity and residual error in the recovered signals.

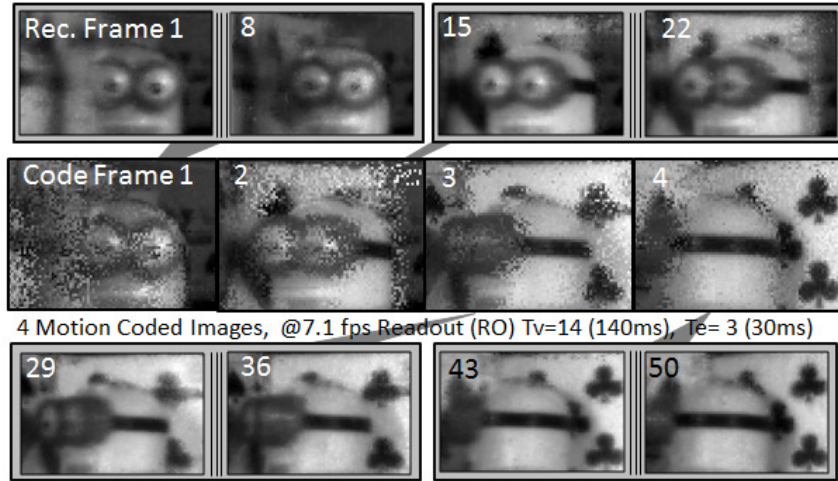
For the reconstruction in equation (4.3) to be guaranteed, the sensing cube  $\mathbf{S}$  must satisfy the Restricted Isometry Principle (RIP) for sparse vectors. RIP characterizes nearly orthonormal matrix when operating on sparse signals.

Typically, for low power CS sensor applications, the reconstruction is executed not at the sensor node but at a receiving base station, as implementation of Eq. 4.3 requires more power and computational complexity. Fig. 4-1 shows an example of the coded image with reconstructed frames acquired using our CMOS PCE camera.

## 4.2 Approach

### 4.2.1 Extracting Temporal Information from Coded Image

Although many compressed sensing inspired sensors have shown superior sensing performance while offering the advantage of reduced power consumption [Hitomi et al., 2011, Zhang et al., 2015b, Zhang et al., 2016, Pareschi et al., 2016], one practical disadvantage of them is that they do not function well in real time closed-loop systems because the sparse reconstruction step is computationally and time intensive. A system would need to recon-

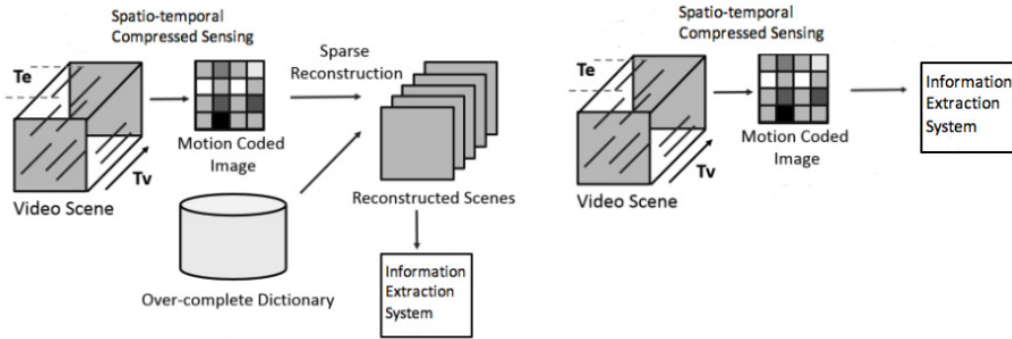


**Figure 4-1:** Four coded images output from a PCE camera sampled at 7.1 FPS [Zhang et al., 2016]; 14 images are reconstructed from each coded image resulting in 100 FPS reconstructed videos.

struct the signal from compressed measurement first before an algorithm can be used to extract relevant information and provide feedback to the sensors.

The computationally expensive reconstruction step may also negatively impacts the ability for adaptive sensing. We found that in most CS based systems, the scene reconstruction quality heavily depends on the scene statistics [Wu et al., 2012] [Zelnik-Manor et al., 2011] [Zhang et al., 2015b]. It can be affected by motion, scene complexity (or sparsity) and noise level as well as compression rate. Therefore, to guarantee performance, we seek to build a smarter adaptive system which can alter the sensing matrix and compression rate to enhance the overall reconstruction quality. But to do so, we would need to find an efficient method to measure these factors that is fast enough to drive the close-loop feedback to adjust the sensing mechanism.

Here we explore the possibility of using deep learning methods to extract signal information directly from the compressed samples such as PCE coded images instead from the reconstructed scenes. Specifically, we choose to use convolutional neural networks (CNNs) as classifiers of the coded images. The choice of CNN is inspired by the



**Figure 4.2:** Conventional information extraction (left) vs. our approach (right).

recent emergence of computer vision algorithms based on the deep neural networks. However, it is not so clear if CNN can also be applied to compressed measurements yet. Theoretically it is a more challenging task since the CNN is forced to learn useful patterns in the coded samples at high compression ratios. A high-level comparison of our proposed approach that combines DL and CS and information extraction in conventional CS system is shown in Fig. 4.2.

The intuition of this approach is derived from the Restricted Isometry Principle (RIP). Random sensing matrix are shown to have small RIP constants, which indicates that they are orthonormal for sparse vectors [Candes and Tao, 2005]. Then it follows that the features of these sparse vectors are also persevered within the compressed samples. In the case of PCE imaging where a video clip is compressed into a single image, the temporal motion are transformed into spatial features of the coded images, which we refer to as temporal to spatial feature conversion (T-to-S). With a deep CNN containing convolutional layers tuned to extract pixel patch characteristics, we expect to extract video labels and motion information by exploiting only spatial patterns of coded images.

We further demonstrate that the benefits of combining deep learning and compressed sensing are two-fold:

*From the sensing perspective*, this method replaces computationally intensive recon-

struction and scene analysis steps with a single CNN. Unlike sparse reconstruction which needs to solve an optimization problem with iterative computation, inference with CNNs only needs a one-shot forward propagation which can be implemented efficiently in real time.

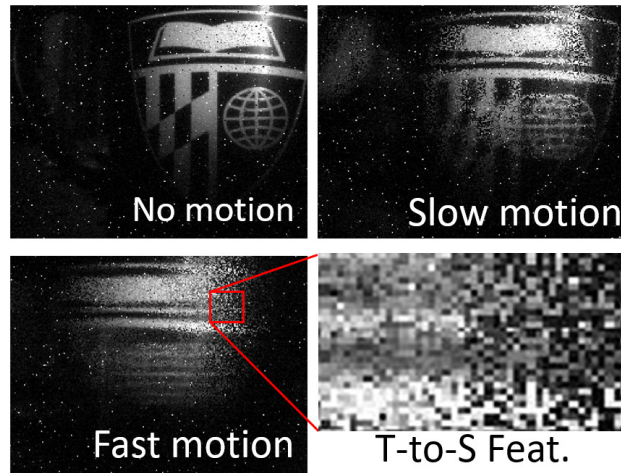
*From the analysis perspective*, PCE provides an efficient mechanism to compress highly redundant video frames into a more compact representation, which supports a more effective approach in preserving temporal information than naive down-sampling methods. Conventionally, deep learning models for video recognition are either based on recurrent neural networks (RNNs) [Donahue et al., 2015, Ebrahimi Kahou et al., 2015, Yue-Hei Ng et al., 2015, Du et al., 2017b] or 3-D CNNs [Ji et al., 2013, Yang et al., 2019, Huang et al., 2015, Li et al., 2020]. Due to an additional dimension handling temporal features, these video models are substantial more difficult to train and expensive to implement than standard image models (i.e. 2D convolutional models). PCE shows a novel way of solving video recognition problems with 2D image models.

#### **4.2.2 Temporal to Spatial Feature Conversion**

Sparse reconstruction is able to faithfully reconstruct the video because motion features are encoded in the compressed images. These features are visible within the coded images: For example, Fig. 4-3 illustrates an example of PCE coded images taken at different object movement speed. For a motionless scene, pixels experience no changes in intensity. Thus the coded image is no different than a single frame from the video. With motion, pixels experience changes in intensity, which are then integrated by pixel exposures activating at different times. These introduce new mosaic-like features along the edges in the coded image. These patterns are essentially the consequences of motion undergoing a transformation into spatial features (T-to-S feature conversion). We call these spatial-motion features.

A quick visual inspection of Fig. 4-3 shows that T-to-S conversion occurs mostly at the edges of the moving object, and is parameterized by the motion speed and direction.

Faster motion yields more and prolonged spatial-motion features; the trace of these features also exist along the direction of motion. We shall quantify these characteristics in the next section.



**Figure 4-3:** PCE images with different motion speeds.

### 4.2.3 CNN Receptive Fields

It is well-known that CNN classifies input images as a hierarchical spatial feature extraction process [Zhou et al., 2014, Olah et al., 2017, Mordvintsev et al., 2015]. The learned feature extractors (filters) are sensitive to specific patterns and are activated when the patterns are discovered in the inputs. The activation map of lower-level filters are passed to the next layer as the input for higher-level feature extraction. In this way, a CNN predicts the label of a given input image by hierarchically exploiting the spatial structure of the image.

Since PCE coded image's spatial-motion features are introduced by the T-to-S feature conversion, it is reasonable to expect that temporal information can also be extracted from PCE coded images where spatial-motion features are interpreted hierarchically as the above-mentioned process.

Moreover, in coded images the motion-less spatial structure is maintained, making it possible to extract spatial and temporal information at the same time by training only one

single classifier. This illuminates a new approach of efficiently extracting scene information from a sequence of frames which utilizes the advantages of both compressed sensing and machine learning. On the one hand, it helps bypass the need for burdensome sparse reconstruction step in compressed sensing. On the other hand, it makes it feasible to extract information from video frames using standard 2D CNNs instead of 3D CNNs or recurrent CNNs (RCNNs) which are substantially more expensive to train and implement.

### 4.3 Spatial-motion Feature

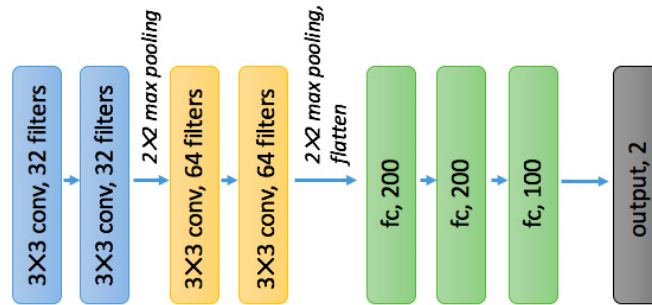
#### 4.3.1 Comparing PCE and Naive Down-sampling

In this section, we demonstrate that CNNs can learn spatial-motion features and can robustly use them as representations to classify motion information through a controlled experiment. We first created synthetic videos using the MNIST dataset. Each synthetic video consists of 10 frames. We generate them by shifting an image ( $28 \times 28$  pixels) towards a fixed direction (left, right, top or down) by 1 pixel position for 10 frames. Thus each frame's size would be  $28 \times 37$  for left or right movement, and  $37 \times 28$  for top and down movement. The videos are then compressed to single-frame coded images using pixel-wise pseudo-random exposure patterns described in Sec. 4.1. This corresponds to compression ratio (CR) of 10.

We consider a binary classification task of distinguishing two opposite motion directions (left v.s. right and up v.s. down). We compare classification on PCE coded images with other naive downsampling techniques such as averaging or random sampling frames, both of which transform a video clip to a single image frame and has the same CR as PCE. We show that the resulting spatial-motion features that contains key motion direction information is a unique consequence of PCE.

We use a standard 6-layer CNN (4 convolutional layers and 2 fully-connected layers) to learn the direction of moving digits. For the 2 classification cases (left vs. right and up vs.

down) the input size of the model are different due to the different spatial size of the videos and coded images ( $28$  and  $37 \times 28$ ). The model architecture is illustrated in Fig. 4-4, and stochastic gradient descent is used for training; the mini-batch size is 128.



**Figure 4-4:** CNN model architecture on MNIST coded images.

With the synthetic dataset, we can achieve classification accuracy of 100% using coded images, while classification using average and random images achieve no better than random guessing. This result is expected as there is a lack of motion related features to distinguish digits' movement directions when they are down sampled through averaging or taking a random frame. This indicates that the spatial patterns introduced by PCE indeed contains important temporal information, which can be extracted by 2D convolutional image models.

### 4.3.2 Characteristics of Spatial-motion Feature

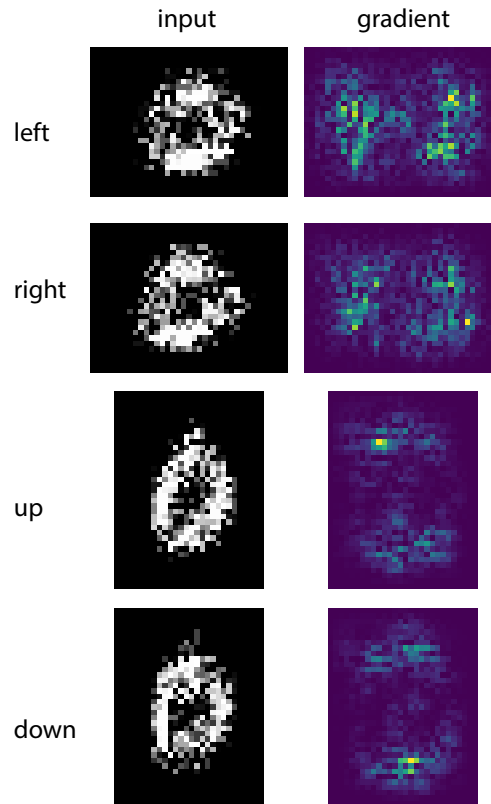
We explore the characters of spatial-motion features in this section. As they are introduced through randomized exposure and compression, it is hard to identify one single feature and quantify its contribution to the classification result. But by designing proper experiments, we show that collectively they have two main properties: (a) they are richer along the movement direction; (b) they are local features.

It is intuitive to expect that spatial-motion features are introduced along the motion direction due to randomized exposure. If that is true, temporal information is better pre-

served when the moving edge is perpendicular to the moving direction because of less feature overlaps. To explain this further, we use the sensitivity maps of the input coded images to the CNN classifier. Sensitivity maps are the absolute value of gradients of classification loss w.r.t to input image pixels which often indicates the saliency of input regions to the classification results [Simonyan et al., 2013]. We show an example in Fig. 4-5. The highlighted regions of the sensitivity map suggests these pixels are more important for correct motion classification. We see that for left and right motion, the spatial motion features along the vertical edges of the image are more salient compared to the horizontal edges. This suggests that spatial motion features on the vertical edges can be better exploited than the spatial motion features on the horizontal edges for left and right motion classification. This compares to up and down motion classification, where the spatial motion features along the horizontal edges are more salient.

We next show that the dependency of motion classification result is based on local features, and that motion classification accuracy is maintained where even the coded image is only partially observed. To examine this, we first trained the network using the full coded images. But in the testing step, we provide the network with only local information by masking pixel rows or columns of the coded images. We then check if the network can still recognize the motion direction labels given only local information.

In Fig. 4-6, we plotted the motion classification accuracy against the number of rows or column that are observed to the network at testing time. For videos with left or right digit motion, we see that as we remove pixel columns in the coded image, the motion classification accuracy drops significantly. But as we remove the pixel rows, CNN maintains solid accuracy of near 100% when only 4 rows of pixels remain of the coded image. It still maintains 85% accuracy when only 1 pixel rows remains. This result is opposite for up and down movement, where CNN maintains solid accuracy with columns removed. This result further verifies our projection that (a) CNN learns to classify motion based on local

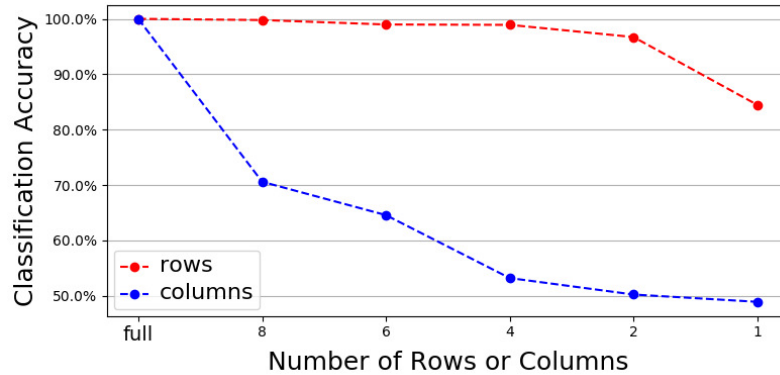


**Figure 4-5:** Coded images (left) and corresponding sensitivity maps (right) in horizontal motion classification (left v.s. right).

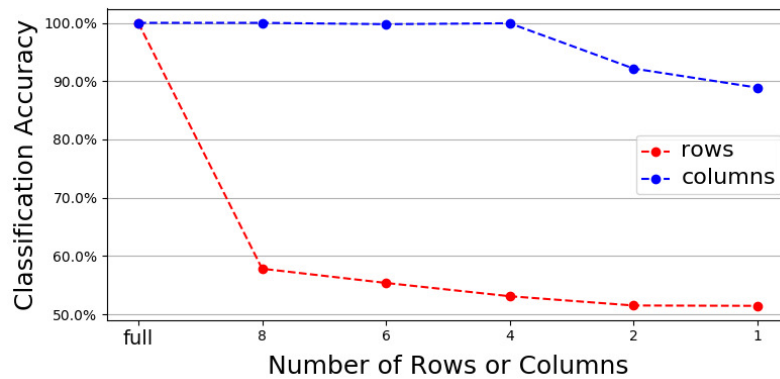
spatial-motion features and (b) PCE spatial-motion features are presented along the motion direction.

### 4.3.3 Transferability of Spatial-motion Feature

After demonstrating that CNNs can successfully learn to use local spatial-motion features for motion classification, we next explore their learning transferability. That is, if we train a network using a set of images for motion classification, can the network learn the local spatial motion features and use them to classify motion of a different set of images? We expect the learned features to be transferable because any scene can be decomposed into a combination of basic elements such as surfaces, curves or edges, which are common across all images.



(a) Classification: Left v.s. Right

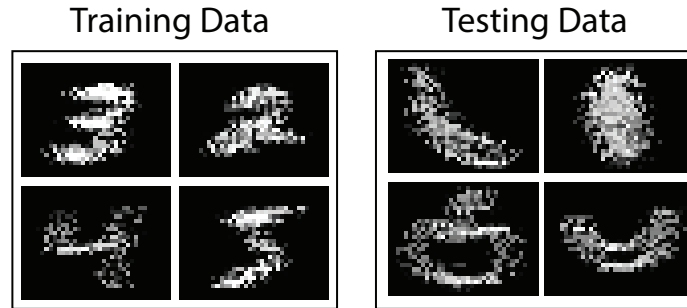


(b) Classification: Up v.s. Down

**Figure 4-6:** Classification accuracy using local information.

We examine this by first training a CNN to recognize motion direction in the case of left v.s. right on MNIST moving digits, but then testing motion classification accuracy on coded images generated from another dataset. We use the *Quick, Draw!* dataset launched by Google. It contains  $28 \times 28$  gray-scale images of common objects. Also like MNIST, these are sketch drawings which may share common pen strokes compared to hand draw digits of MNIST.

To generate a testing set from *Quick, Draw!*, we pick 1000 drawings for each object from apple, banana and carrot, and thereby form a 3000-sample subset. We then generate synthetic video clip of 10 frames by moving the object toward left or right, as what we did



**Figure 4.7:** Examples of training coded images from MNIST dataset and testing coded images from Quick, Draw! dataset. The CNN models achieve 93.1% and 94.0% accuracy of recognizing moving direction horizontally and vertically on Quick, Draw! although they were trained on MNIST moving digits.

for the MNIST moving digits in the previous subsection. We then compressed them to CS coded images using the same sensing matrix. Examples of *Quick, Draw!* coded images are shown in showed in Fig. 4.7.

Even though these sketch drawings do not appear in the training data, we still get 93.1% and 94.0% test accuracy in detecting the motion direction horizontally and vertically. This result illustrates that the spatial-motion features learned by the CNN model are general across different domains to some extent. For applications where a large amount of original training data is too expensive to obtain, the transferability property suggests that the training process can be done with data obtained from other domains or synthetic data.

## 4.4 Experiments

### 4.4.1 Motion Classification of MNIST Moving Digits

In this simulation experiment we intend to demonstrate and verify the following: Given a coded image that encodes a moving scene, can we extract useful information regarding the scene such as: (a) the direction of the motion; (b) velocity of the movement; (c) object detection and classification.

## Data Generation

We choose to use the MNIST dataset in this preliminary experiment. For each image in the training dataset, we generate a video of 10 frames by keeping the digit still or moving the digit to one of the four directions (up, down, left and right) with two speeds (one pixel or two pixel offset from the previous frame). We then apply a random exposure pattern (shown in Fig. 2) to encode this video into a single coded image. This corresponds to a compression rate of 10. Note that the random exposure pattern is fixed and used for all the training and testing datasets. An example of the data is shown in Fig. 4-8.

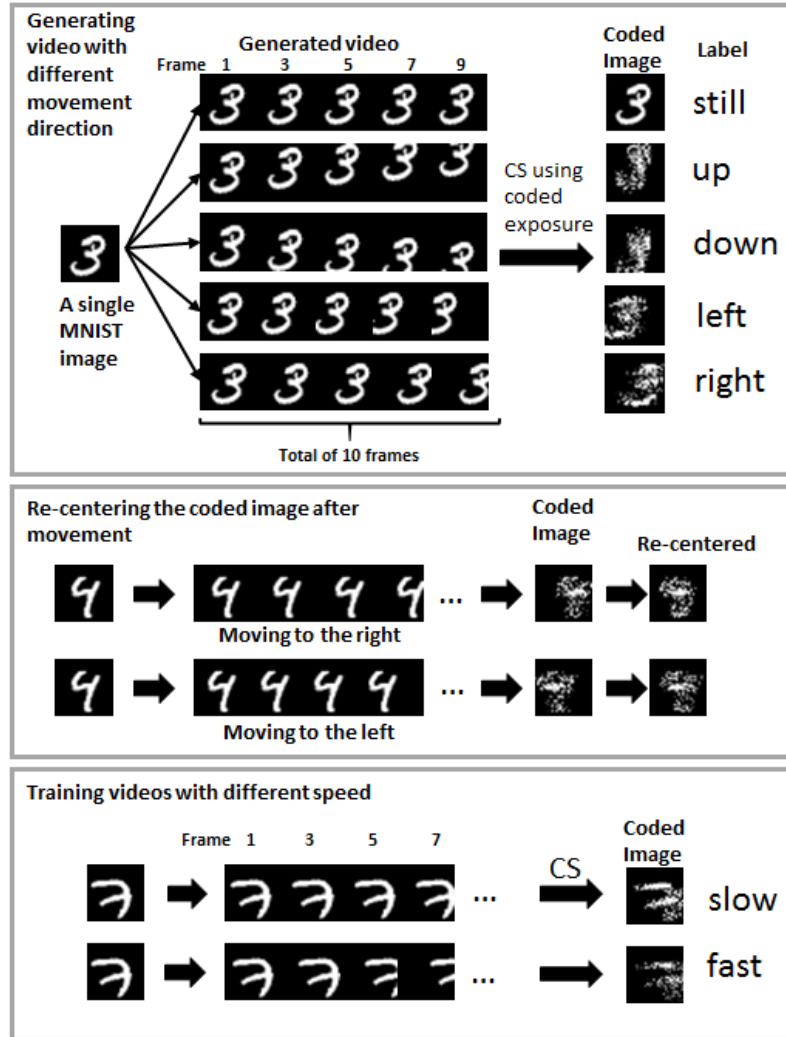
One observation on the training dataset is that when a digit moves to a direction, its coded image would be biased towards that particular direction, shown in the 2nd box in Fig. 4-8. This makes the classification task trivial because the algorithm can simply classify direction based on counting the number of non-zero pixels at a particular quadrant. To avoid this bias, we re-center the coded image and use this coded image in both training and testing sets. This would ensure the CNN learns correct features of a motion instead of doing simple non-zero pixel counting.

## Classification

We trained three different CNN networks for three classification tasks respectively. During training, the network is presented with coded images and label pairs.

The classification results are shown in Tab. 4.1. We see that the CNN does an excellent job with speed classification from coded images compared to raw videos. It however does a slightly less decent job with direction classification and object classification.

Even with some performance drop, using coded image for classification directly could be the method of choice due to its computational efficiency. It is two times faster than same classification done on a raw video. More importantly, using the coded image for these classification tasks also bypasses the need for sparse reconstruction which involves solving



**Figure 4·8:** An example of generated training videos and coded images of the videos.

**Table 4.1:** Comparison of classification performance on coded images and on videos.

Classification	CNN + video		
	Acc.	Train time (s)	Test time (s/1000 samples)
Digit	98.60%	5566.72	1.01
Direction	100%	4861.49	1.25
Velocity	100%	2199.52	1.21
Classification	CNN + coded image		
	Acc.	Train time (s)	Test time (s/1000 samples)
Digit	96.22%	3079.57	0.65
Direction	90.04%	3812.42	0.56
Velocity	100%	1483.64	0.53

computationally intensive convex optimization.

#### 4.4.2 Motion Classification of Gesture Videos

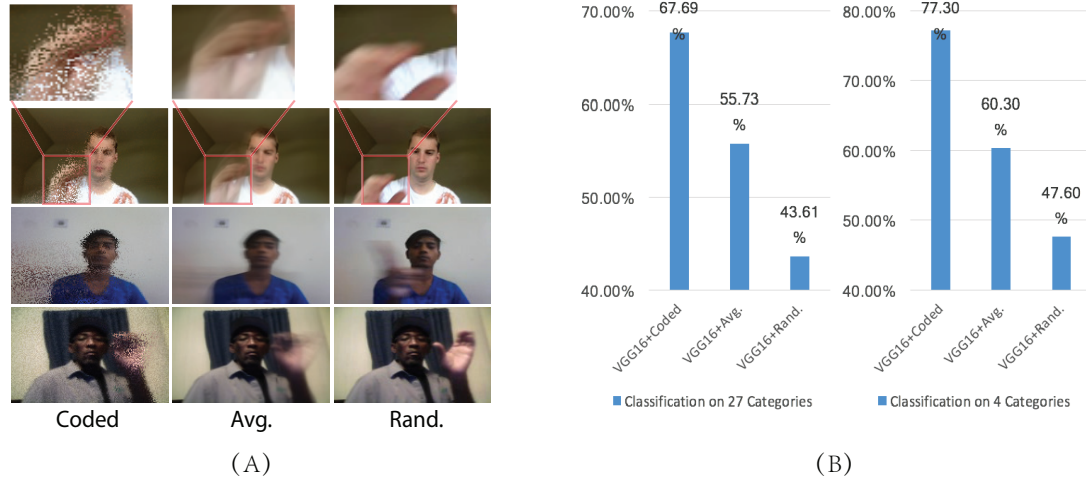
##### Data

The 20BN-Jester dataset contains labeled video clips of human performing pre-defined hand gestures, such as rolling hand and sliding fingers. It has 27 different gesture categories. When the same gesture is performed in different directions (e.g. sliding fingers from left to right and right to left), they are labeled as different gesture categories. The dataset has 118562 samples for training and 14787 samples for testing.

Because video clips in the dataset are of slightly different resolution and duration, we first rescale all the videos to the same spatial resolution of  $100 \times 170$ . We then ensure each clip contains 30 frames by padding the last frame (for clips with fewer frames) or removing frames from beginning and end equally (for clips with more frames).

The training and testing datasets are then generated by compressing the video into coded images using a coded exposure pattern as shown in equation 4.2. The exposure duration for each pixel is 5 frames while the pixel exposure starting time is randomly chosen. By doing so, a 30-frame video clip is transformed to a single-frame coded image equivalent to CR of 30.

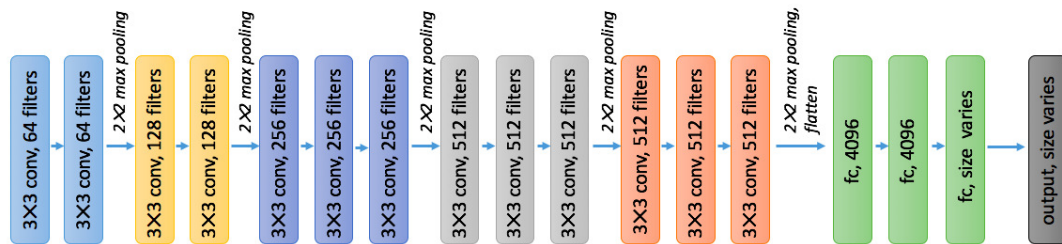
For comparison, we also generated two sets of naively downsampled frames of the same CR by taking a random frame or temporally averaging all frames. This process is the same as what we did previously on the MNIST dataset. Compared to coded images, the random frame contains no temporally transformed features since it is just a snapshot of the video. Note that there still exists some spatial information that may contribute to gesture classification, e.g. the pose of the hand provides sufficient information to differentiate between sliding and zooming. Thus the base-line accuracy of random sampling will be better than random guessing. The average image is equivalent to an coded image with all exposure turned on, corresponding to a sensing matrix with rank of 1. This operation eliminates most of the high frequency spatial details but converts some temporal features into spatial as motion blur. Fig. 4-9 illustrates some examples of these videos after being compressed as single-frame images.



**Figure 4-9:** (A): A comparison of CS coded images, average images and random images compressed from the same video clip. (B): A comparison of classification accuracy using coded images, average images and random images.

## CNN Model Architecture

We use a VGG-16 architecture [Simonyan and Zisserman, 2014] as the classifier on the images. We change the width of the last fully-connected layer and the output layer in order to fit our task. The VGG-16 model contains 16 layers including 13 convolutional layers and 3 fully-connected layers. We use the pre-trained weights of VGG-16 on ImageNet [Krizhevsky et al., 2012] to initialize the filters in convolutional layers but re-train the entire network on the (coded/ average/ random) images of 20BN-Jester. The architecture of our model is visualized in Fig. 4-10, and the training is performed by stochastic gradient descent with a mini-batch size of 128. Three separate networks are trained to classify coded images, average images, and random images respectively.



**Figure 4-10:** CNN model architecture on 20BN-Jester coded images. This architecture follows VGG-16 net with varied width of the last fully connected layer and output layer.

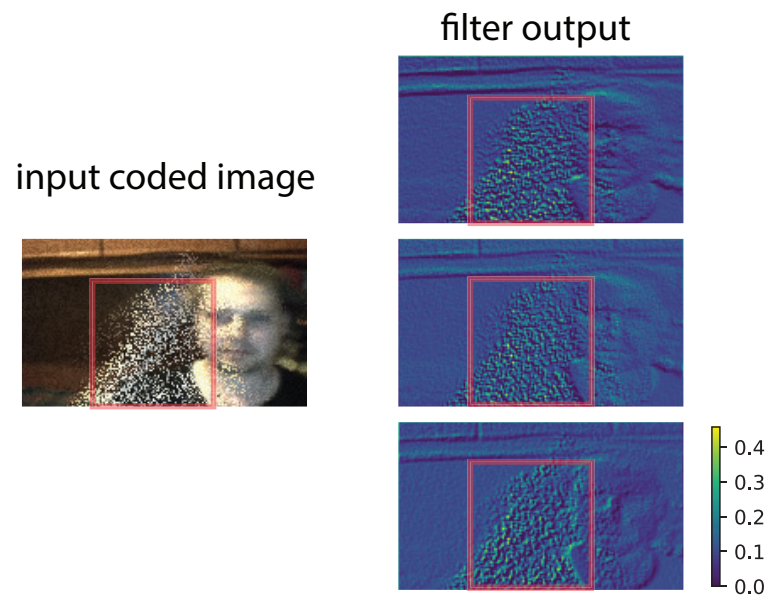
## Classification Tasks and Results

The goal of the classification task is to demonstrate the difference in classification accuracy of coded images and naively down-sampled frames. We first train image models to recognize hand gestures (27 categories) using coded, averaged, or random sampled images respectively and show that using different type of images indeed result in different classification accuracy (see Fig. 4-9 B left). We see that using coded images achieve the highest accuracy (67.69%) while random images have the lowest accuracy (43.61%). The gain of accuracy in gesture recognition indicate more temporal information is preserved. Note

that in all cases the input of classifier is only one single frame, the temporal information that increases classification accuracy refers only to the temporal information that is coded into spatial features, verifying the superiority of PCE coded images in converting temporal features to spatial features.

However, as discussed above, even a random frame of a video clip contains spatial information that is useful for gesture recognition. As our goal is to identify the accuracy gain caused by temporal-to-spatial feature conversion in the PCE coding process, we also compare the classification accuracy of using different types of images in a more restricted setting, i.e. recognizing the motion direction of the same gesture type. Here we choose to classify the direction of finger swipe out of four directions (top, down, left and right). This task would decrease the amount of pure spatial features' contribution to the classification results. The result of this classification task is shown on the right side of Fig. 4-9 B. In this more restricted classification setting, we see that the difference in accuracy using different types of images is even more significant where the accuracy enhancement by using coded images is nearly 30% from random frames and 17% from averaged frames respectively. This further verifies our hypothesis that CS coded images preserve temporal information that would be lost when using naive frame down-sampling.

To show that the trained network using coded images indeed tends to be sensitive in exploiting the mosaic-like spatial motion features, we visualize the activation map of some convolutional filters in the first layer as shown in Fig. 4-11. The highlighted region of the activation map is where a particular feature is densely present. We can see that the highlighted region in the activation maps is exactly where the motion occurs. This further demonstrates that spatial-motion features, as a special type of spatial feature, can also be learned by convolutional filters as we discussed in Sec. 4.2.3.



**Figure 4-11:** Visualization of activation maps of convolutional filters that are sensitive to the motion region. This indicates that some filters learned to identify temporally transformed spatial features after training.

## Chapter 5

# Deep Learning for Lossy Image Compression

### 5.1 Neural Image Compression

Neural image compression (NIC) has recently emerged as a promising direction for advancing the state-of-the-art of lossy image compression [Toderici et al., 2015, Ballé et al., 2016, Shen et al., 2018, Liu et al., 2019, Ballé et al., 2018, Toderici et al., 2017, Rippel and Bourdev, 2017, Mentzer et al., 2018]. Besides outperforming modern engineered codecs such as JPEG [Skodras et al., 2001], JPEG 2000 [Rabbani, 2002] and BPG [Bellard, 2015] in rigorous distortion-based evaluation (e.g. PSNR), NIC also enabled many unique applications which largely broadens the scope of conventional image codecs, e.g. reconstructing realistic images from extremely low bit-rates by adding artificially synthesized image details [Agustsson et al., 2019, Santurkar et al., 2018].

NIC models feature an encoder-decoder style structure. An input image  $x$  is first encoded into a compact latent representation  $y$  by the encoder network, followed by a quantization step (we denote the quantized  $y$  as  $\hat{y}$ ). The discrete  $\hat{y}$  can be further compressed into a bit-stream with length  $R(x)$  for efficient storage and transmission, using lossless entropy coding methods such as arithmetic coding. We can then reconstruct a restored image  $\hat{x}$  from  $\hat{y}$  through a decoder network. The reconstruction quality is usually measured by distortion metrics such as PSNR and MS-SSIM [Wang et al., 2003] (we denote it as  $D(\hat{x}, x)$ ).

Both encoder and decoder networks contain trainable parameters. However, applying end-to-end training directly is difficult due to the non-differentiable operations in quantization and entropy coding. To facilitate back-propagation during training, non-differentiable

operations need to be replaced by differentiable ones. For instance, it is common to replace quantization with noise injection, and entropy coding with a parameterized entropy estimator. For clarity, we use  $\tilde{x}, \tilde{y}, R_e(x)$  to denote the fully differentiable versions of  $\hat{x}, \hat{y}, R(x)$  in training.

Note that in lossy image compression, one needs to deal with two competing desires: better reconstruction quality measured by  $D(\hat{x}, x)$  versus less bits consumption measured by  $R_e(x)$ . Therefore, a trade-off factor  $\lambda$  is usually used in the optimization loss:

$$\lambda D(\tilde{x}, x) + R_e(x). \quad (5.1)$$

Training with a large  $\lambda$  results in compression models with smaller distortion but more bits consumption, and vice versa. This is a key trade-off in lossy compression known as the rate-distortion (R-D) trade-off which characterizes the performance of a compression method. One important requirement for lossy compression methods is bit-rate control, i.e the capability of providing compressed images with varying quality based on practical needs. A desired compression algorithm should be able to perform bit-rate control smoothly, accurately and efficiently.

Bit-rate control, unfortunately, has been a challenging task for NIC methods. Conventionally, it requires training multiple model instances using different  $\lambda$  values and storing all of them for both senders and receivers. Even so, only limited control at a very coarse level can be achieved. It is not only because of the cost for training and storing a large number of model instances needed to achieve a better control precision, but also because it usually takes several rounds of trial-and-error before finding the right model instance that has the desired R-D trade-off, as the relation between  $\lambda$  and real bit-rate measured by Bit Per Pixel (BPP) is unknown and varying per image.

Another limitation of NIC methods comparing with modern engineered codecs is the lack of adaptivity in the compression process. For instance, BPG has a mechanism for

dividing a block into sub-blocks based on comparing the compression results of the root block and the sum of its children. Contrarily, NIC models pass the input image only once and their parameters are fixed once trained. It would be possible to achieve a better compression performance by making NIC adaptive to individual images.

Motivated by addressing the above mentioned limitations, we propose Substitutionary Neural Image Compression (SNIC). This approach is built on top of an arbitrary pre-trained NIC model, enforced by the key idea of searching for a suitable substitutional image for the original image to be compressed, where using the substitute as the model input leads to a desired compression performance such as controlling bits consumption and/or improving reconstruction quality (measured against the original image). We will further describe SNIC in details in the next section.

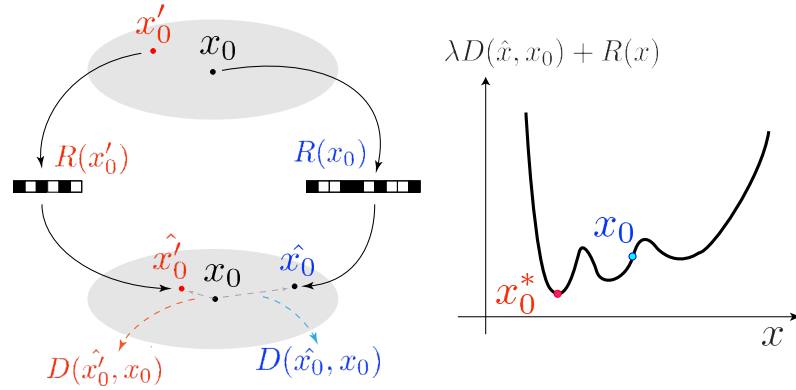
## 5.2 Substitutional Neural Image Compression

### 5.2.1 Method

#### The Optimal Substitutional Image

Neural image compression can be viewed as a two-step mapping process (see Fig. 5.1 left). The original image  $x_0$  in a high dimensional space is mapped to a bit-stream with length  $R(x_0)$  (the encoding mapping), which is then mapped back to the original space as  $\hat{x}_0$  (the decoding mapping). We denote the 2-step mapping as a function  $T(\cdot)$ , i.e.  $\hat{x}_0 = T(x_0)$ . The reconstruction quality is evaluated by a distortion metric  $D(\hat{x}_0, x_0)$ .

The question is: When compressing an image, does applying the compression algorithm to the original image necessarily yield the best compression result? For example, if there exists a substitute image  $x'_0$ , such that after applying the same two-step mapping, it is mapped to  $\hat{x}'_0$  with  $D(\hat{x}'_0, x_0) < D(\hat{x}_0, x_0)$ , and at the same time  $R(x'_0)$  is smaller than  $R(x)$ , it means we can improve compression performance by simply substituting  $x_0$  with  $x'_0$  as the input of the compression model without any other change.



**Figure 5-1: Left:** compression and reconstruction by a specific compression model can be viewed as a 2-step mapping. If there exists an  $x'_0$  such that it is mapped to  $\hat{x}'_0$  that is closer to  $x_0$  with  $R(x'_0) < R(x_0)$ , then better compression can be achieved by simply substituting  $x_0$  with  $\hat{x}'_0$ . **Right:** consider  $\lambda D(\cdot, x_0) + R(\cdot)$  as a function of the input image. The best compression performance is achieved at the global minimum of this function.

From another perspective, if we view our minimization target  $\lambda D(\hat{x}, x_0) + R(x)$  as a function of the input image  $x$  (see Fig. 5-1 right), the original input  $x_0$  does not necessarily lie on the global or even a local minimum. In the high dimensional space of  $x$ , a better substitute could potentially be found by effective searching.

Substitutional Neural Image Compression (SNIC) aims at finding the optimal substitute  $x_0^*$  when compressing  $x_0$  using a particular NIC model, which can be formally expressed as

$$x_0^* := \arg \min_x \lambda D(\hat{x}, x_0) + R(x) \quad \text{s.t.} \quad \hat{x} = T(x). \quad (5.2)$$

It is worth noting that the reconstruction quality is still measured against the original image, i.e. we seek to minimize  $D(\hat{x}, x_0)$  over  $x$  rather than minimizing  $D(\hat{x}, x)$ .

Unfortunately, the above problem is intrinsically difficult to solve, as searching in such high dimensional input space without any heuristic is hopeless. However, for NIC, we do have a solvable alternative of the above optimization problem as follows.

$$x_0^* := \arg \min_x \lambda_s D_s(\tilde{x}, x_0) + R_e(x) \quad \text{s.t.} \quad \tilde{x} = T_e(x) \quad (5.3)$$

Specifically, as what we did in the training phase, we replace any non-differentiable operations with differentiable alternatives, and we use  $\tilde{x} = T_e(x)$  and  $R_e(x)$  to indicate this change. As such, an approximation of the optimal solution of optimization 5.3 can be efficiently found by starting from  $x_0$ , and updating it iteratively toward minimizing the loss through a gradient descent process.

Once  $x_0^*$  is found, it does not require any change to neither the encoding network on the sender’s end, nor to the decoding network on the receiver’s end. This approach is compatible with any pre-trained compression model, as long as it has a differentiable version. This precondition should be met by most neural compression models as it is also required for training.

### **Bit-rate and Distortion Metric Control**

One appealing property of SNIC is that the loss for generating substitutional images could be different from the loss used for training the underlying NIC model. In equation 5.3, we use  $\lambda_s$  to differentiate the trade-off factor in substitute generation from  $\lambda$  for training. Bit-rate control is achieved by changing  $\lambda_s$  and using different substitutes that have different characteristics. Since  $\lambda_s$  is a continuous variable, bit-rate can be smoothly adjusted, at least in theory. This is a significant advantage over conventional methods that switch among different model instances. Nonetheless, it remains to investigate the limits of the reachable BPP range and what an R-D trade-off curve it produces. It would be appealing if the curve is above the original R-D curve over a non-trivial BPP range. If so, in this range our proposed bit-rate control completely dominates over conventional methods as it does not only make bit-rate control easier but also improves compression performance. We show that this is indeed the case with empirical results in the experiments section.

Similarly, the target distortion metric in Eq. 5.3 can also be specified as desired, regardless of the target distortion metric used for training the underlying NIC model. When changing the target metric, we are interested in seeing how it influences the distortion against the original metric. This part will also be further discussed in the experimental results.

### Direct Bit-rate or Distortion Control

Controlling the R-D trade-off by specifying  $\lambda_s$  can still be inconvenient, as the correspondence between  $\lambda_s$  and the resulting bit-rate is nonlinear and unknown to system designers. Many application scenarios require a fixed bit-rate with a certain precision level given by bandwidth of transmission conditions. Therefore, it is desired to control the bit-rate directly instead of searching for a suitable  $\lambda_s$ , which may take several rounds of trial-and-error. This can be achieved by applying SNIC using a slightly modified formula:

$$\underset{x}{\text{minimize}} \quad D_s(T_e(x), x_0) + \kappa \max\{R_e(x) - R_t, \tau\}. \quad (5.4)$$

In the above formula,  $\kappa$  is a large number putting a heavy penalty when  $R_e(x)$  exceeds the target bit-rate  $R_t$ . Since gradient updates with a particular step-size could lead to  $R_e(x)$  being slightly smaller than  $R_t$ , we use a small value of  $\tau$  to calibrate this offset.

Similarly, compression distortion can also be directly controlled if we switch  $D$  and  $R$  in the above formula, i.e.

$$\underset{x}{\text{minimize}} \quad R_e(x) + \kappa \max\{D_s(T_e(x), x_0) - D_t, \tau\}. \quad (5.5)$$

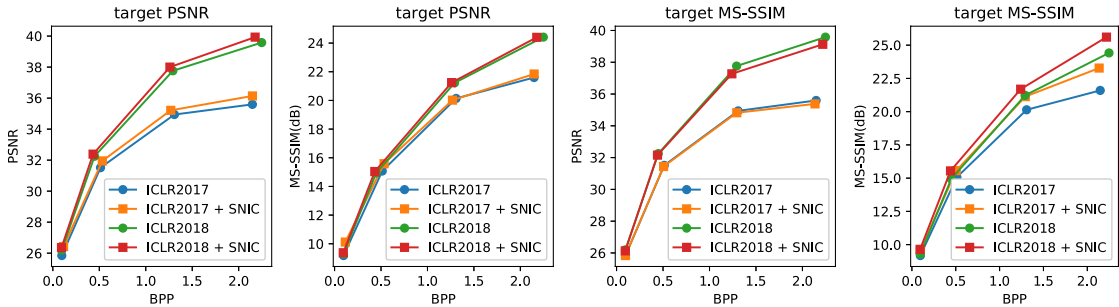
### 5.2.2 Experiments

In this section, we demonstrate the effectiveness of SNIC in enhancing compression performance and bit-rate control. It is important to note that the bit-rates in the experiments are

real bit-rates using arithmetic coding instead of estimations. We also conduct experiments on the computational overhead of the proposed method.

The testing dataset is Kodak, the mostly used benchmark dataset for image compression. We use two base NIC models from [Ballé et al., 2016] (indicated below as ICLR2017) and [Ballé et al., 2018] (indicated below as ICLR2018), where we obtain four model instances for each of them though training with different  $\lambda$  values (0.001, 0.01, 0.1, 1 for ICLR2017 and 0.001, 0.01, 0.1, 0.5 for ICLR2018), in order to characterize their R-D trade-off curves as baselines. The training data is a subset of ImageNet [Deng et al., 2009] that contains images larger than  $256 \times 256$  pixels, and the training distortion metric is mean squared error (MSE).

### Enhancing Compression Performance



**Figure 5.2:** R-D curves with and without SNIC (averaged over the Kodak dataset).

For both ICLR2017 and ICLR2018, we generate substitutional images for all trained model instances, so that a rigorous comparison between SNIC and baseline models can be measured by their R-D curves over a wide range of bit-rate. There are two sets of experiments that target improving PSNR and MS-SSIM respectively. When targeting PSNR, formula 5.3 is used to generate substitutes, where  $\lambda_s$  is set to be the same as  $\lambda$  of the underlying base model (i.e same loss used for both training and substitute generation). When targeting MS-SSIM, in order to align the resulting bit-rates with original bit-rates, for a

better comparison, we use formula 5.4 with target BPPs being the same as using original images and model instances.

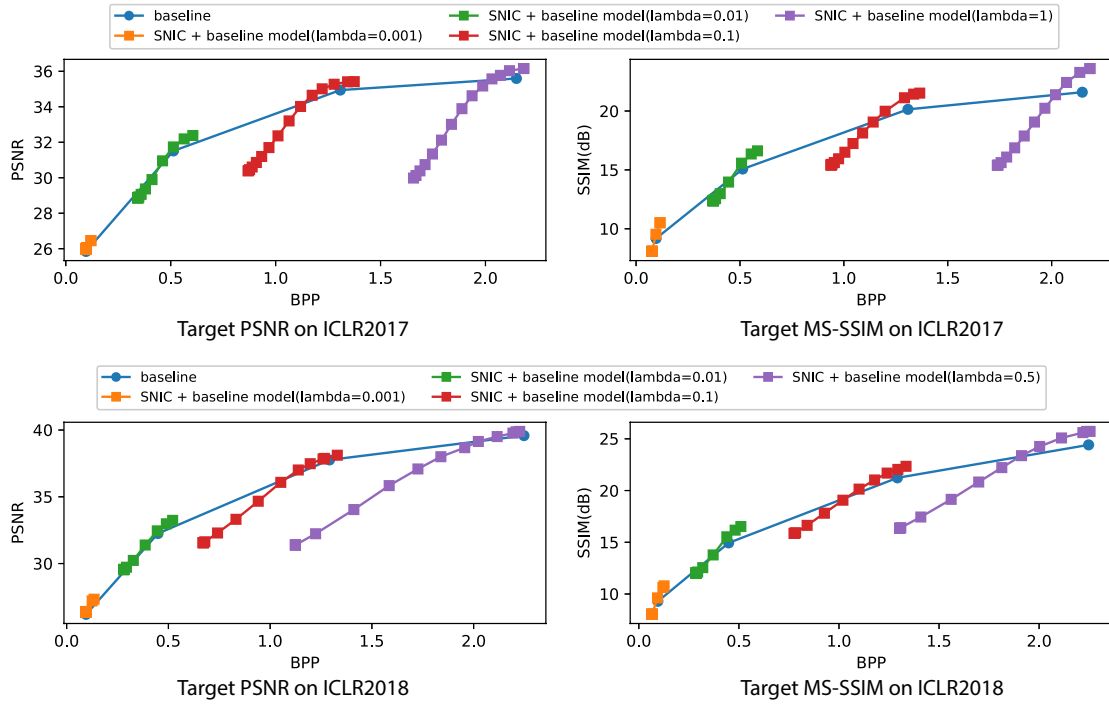
All substitute images are generated from original images through a 100-step gradient descent process, and following each update step there is a clipping step assuring that pixel values are in the allowed perceptual range (i.e.  $[0, 1]$ ).

Fig. 5-2 shows compression performance measured by R-D trade-offs using original images and substitutional images respectively. Key observations are as follows. When targeting PSNR, SNIC improves the PSNR-BPP trade-off on both baseline models. While it also leads to slightly improved MS-SSIM records. When targeting MS-SSIM, the improvement is more significant (e.g. ICLR2017 with SNIC almost achieves the same R-D curve as ICLR2018 does). This indicates SNIC is effective in enhancing compression even if the distortion metric is different from its base model. However, there is a small drop of PSNR on high bit-rates though. This drop may be avoided by using a smaller step size at the cost of less significant improvements on MS-SSIM.

### **Bit-rate Control**

Bit-rate control can be performed by either controlling the trade-off factor  $\lambda_s$  in Eq. 5.3 or by controlling the target BPP  $R_t$  directly in Eq. 5.4. We found that results of using these two formula are similar, but using Eq. 5.4 is easier to have a equal spacing on BPP range for a better demonstration. Thus we show experimental results using Eq. 5.4 in Fig. 5-3.

Increases and decreases in bit-rate using SNIC have different effects. When increasing bit-rate, both reconstruction quality and BPP increase, with a R-D curve above the baseline curve, before it reaches to the upper limit, where a larger  $R_t$  or  $\lambda$  will not further increase neither BPP or reconstruction quality. The upper limit is usually very close to the original BPP. A larger controllable range is achieved for bit-rates below the original rate. However, as bit-rate decreases, the R-D trade-off tends to be worse, which eventually makes the R-D curve below the baseline curve. In Fig. 5-3, we show the entire achievable ranges of bit-

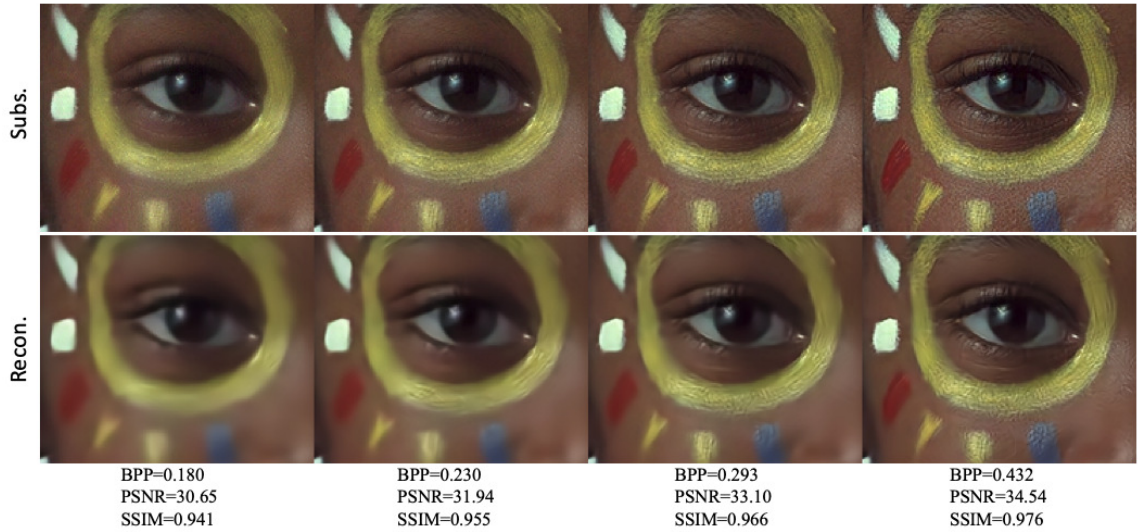


i

**Figure 5-3:** Achievable range of bit-rate control with different baseline models and target metrics. Points in the same color are generated with the same model instance using Eq. 5.4. The original R-D curve (blue line) is given by varying  $\lambda$  at training phase. The R-D curves are averaged over the Kodak dataset.

rate control by SNIC with a single model instance. There is a non-trivial range of around 0.3 where SNIC curve is above the baseline. It indicates that facilitated by SNIC we are able to use a few model instances to cover the entire interested BPP range, while achieving superior compression performance compared with conventional switch-based control with infinite many model instances.

An example of substitutional images and corresponding recovered images with different target bit-rates is shown in Fig. 5-4. There is a visible difference between these images where the substitutes targeting for high bit-rates are more sharper than substitutes targeting for low bit-rates.



**Figure 5-4:** Substitutional image clips that target different bit-rates and their reconstructions. Substitutional images are generated from the same original image and the same compression model instance.

### Speed of SNIC

As substitutional images are generated in an iterative manner, the speed of SNIC mainly depend on the number of gradient update steps and the base compression model being used. Theoretically, using more steps should be no worse than using less in terms of compression performance, so there exist a trade-off between performance and speed. In this experiment, we study the speed of substitutional image generation by comparing various numbers of steps and their effects on compression performance (see Tab. 5.1). This experiment is conducted using TensorFlow (V 1.15) and a single Nvidia GeForce RTX 2080Ti graphics card.

It is observed that even a 5-step process provides a noticeable improvement, which takes less than 0.1 second generation time. There is no much improvement after 100 steps. In practice, the number of generation steps could be adjusted in real time easily based on practical needs as it requires no other change of the system.

**Table 5.1:** Average time consumption per image for generating substitutes and corresponding performance using different number of steps. Program time refers to the total time consumption including overheads such as variable initialization while generation time refers to only time for gradient updates. ICLR2017 is used as the base model.

Steps	0	5	10	50	100	500	1000
Program Time(s)	-	0.159	0.223	0.778	1.488	7.129	14.141
Generation Time(s)	-	0.068	0.132	0.685	1.396	7.037	14.050
Performance(PSNR) on base models with different <b>training</b> $\lambda$							
$\lambda=0.001$	25.84	25.90	25.96	26.23	26.32	26.42	26.44
$\lambda=0.01$	31.51	31.65	31.77	32.15	32.22	32.28	32.29
$\lambda=0.1$	34.94	35.14	35.24	35.39	35.40	35.40	35.40
$\lambda=1$	35.59	35.93	36.02	36.14	36.15	36.16	36.16

## **Chapter 6**

# **Discussions and Concluding Remarks**

## **6.1 Connections between Security and Compression**

### **6.1.1 The Usage of Input Gradient**

When training deep neural networks, we take gradients of the loss function with respect to weights parameters by using the so-called backpropagation algorithm which makes the foundation of modern deep learning. In this thesis, we also described two contexts in which gradients (of a loss function) is taken with respect to the model's input instead. The first use case of input gradients regards adversarial attacks, where they are used to iteratively construct an adversarial perturbation on the input image, which misleads the target model to make erroneous classification. The second use case of input gradient is in Substitutional Neural Image Compression, where the input image of a compression model is optimized so that it leads to a desired rate-distortion trade-off. Computing input gradient is a key step in the above context as the optimization of the input image is conducted iteratively via a gradient descent process.

The usage of input gradient in these context stems from the most important property of deep learning models: they are designed to be differentiable. The differentiability of deep learning models make sure that gradient can be backpropagated efficiently so that parameters of a DL model can be updated toward minimizing a loss function. And as the output of a DL model is a function of both model parameters and the input example, gradients of any loss respect to the input example is also available by backpropagation.

We show that the differentiable nature of deep learning unfortunately leaves holes for its security. In adversarial attack, input gradient of the adversarial loss actually contains sensitive information about the weak spots of a specific model. More specifically, the input gradient in the context illuminates the direction where the decision boundary is close to the original example, i.e. the direction where the model has the worst robustness. Usually we want to hide the weak spots of our deep learning models and prevent them from being exploited. That is why many defensive methods against adversarial attack focus on "obfuscating" the input gradient of a model by e.g. adding non-differentiable operations to the network. In a sense, stochastic defense also lies in this category where input gradient is randomized for obfuscating the adversary. However, many methods can be bypassed with counter-measures of attackers, interested readers can refer to [Athalye et al., 2018] for more details.

Nonetheless, the differentiable nature of deep learning also provides unique advantages in many tasks, and Substitutional Neural Image Compression is a typical example of them. The differentiability of neural image compression models enables us to find a model-specific and target-specific optimized input image easily using input gradients, which is difficult for conventional engineered codecs. As demonstrated in this section, the same technique of computing input gradients can be used for either benign and malicious purposes. Finally, it is also worth mentioning that the technique of input gradients appears in many other fields of deep learning, such as feature visualization [Olah et al., 2017, Mordvintsev et al., 2015].

### **6.1.2 The Usage of Compression Models in Anomaly Detection**

Auto-encoders are essentially compression models: they learn a compressed representation of the input examples from which the original uncompressed input examples can be reconstructed. In the context of anomaly detection tasks, auto-encoders are trained with only benign data examples. The philosophy is that since there are only benign examples used

for training, the reconstruction error using the trained auto-encoder is relatively small for benign examples, but relatively large for anomalous examples. Therefore, we can use the trained auto-encoder to detection anomalies in the inference phase by setting a threshold of the reconstruction error.

Neural image compression models, and neural compression models in general, have no essential difference from auto-encoders, except that the latent variable of neural image compression models are further compressed into a bit-stream by quantization and arithmetic coding. Since quantization and arithmetic coding used in all NIC models are the same, the compression performance is mostly depend on the quality of latent representation of NIC models which are directly related to the learned encoder and decoder networks. Besides training data, the architecture of the encoder and decoder has a crucial impact on the compression quality. Not surprisingly, compression models with greater complexity (e.g. parameter size) usually result in better compression, indicating a trade-of between model complexity and compression performance. Also, there are activation functions that are commonly used for NIC, but rarely used in other fields of deep learning, e.g. generalized divisive normalization (GDN) used in [Ballé et al., 2016].

However, the objective of training anomaly detection models and neural compression models are different. For NIC models, we seek to minimize the reconstruction error. But for anomaly detection models, we seek to maximize the difference of reconstruction error between normal data points and anomalous data points. As a consequence, a good anomaly detection model does not need to have high complexity and minimized the reconstruction error to a certain level, as long as the error distributions of normal and anomalous data are separative enough. This is why anomaly detection models are usually much lighter compressed with neural compression models. As we need the difference of reconstruction error between normal and anomalous data as an evaluation metric, sometimes we need a validation set of anomalous examples for efficient architecture search of auto-encoders.

It is important to note that not all types of anomalies are detectable by auto-encoders. Auto-encoders detect examples as anomalies that do not compress well. There might be examples whose values deviate from the normal distribution but still fit well into the trained auto-encoder with small reconstruction errors. This type of anomalies is called point anomalies. In order to capture this type of anomalies, models solely trained with reconstruction error is not enough. Therefore, sometimes it requires auto-encoders with regularized latent distributions, such as Variational Auto-Encoders (VAEs) [Xu et al., 2018, An and Cho, 2015].

### **6.1.3 The Usage of Randomness in Improving Robustness and Compactness of Deep Learning**

Randomness plays an important role in tackling both security and compression challenges of deep learning. However, for different problems, randomness is Incorporated in different parts of deep learning models.

In Sec. 2, we showed that the key idea of stochastic defense is to randomize conventional a deterministic deep learning model so that its parameters and/or structure vary in the inference time. The benefits that randomness brings to a network can be interpreted in different perspectives. One intuitive way to understand it is the variants of a stochastic models tend to have different weak spots (see Fig. 2-1), and these weak spots neutralize with each other so that the robustness of the stochastic model improves as a whole. Another appealing property of stochastic defense is that it is also shown to be effective against other security threats such as adversarial reprogramming, as we discussed in Sec. 2.4.

In Sec. 4, we showed that by training deep learning models directly on CS coded signals, we could reduce model complexity as a consequence of a more compact input representation. Although randomness is not directly applied in this approach, it is exploited implicitly through the sensing matrices which are usually generated randomly from a Gaussian or Bernoulli distribution. A randomly generated sensing matrix ensures the RIP con-

dition so that information is not lost in the compression process, which is a fundamental precondition of our proposed method of extracting information from CS coded signals using CNNs.

## 6.2 Summary and Future Directions

In this thesis, we discussed topics in the middle field of deep learning, security and compression. These topics include both security-related and compression-related issues of deep learning models themselves, as well as their applications for conventional tasks in the security and compression domains.

In the field of security, we showed that deep learning models are vulnerable to adversarial attack and adversarial reprogramming. To defense against these malicious attacks, we proposed multiple stochastic defense methods, including Defensive Dropout, Hierarchical Random Switching, and Adversarially Trained Model Switching. We also applied deep learning models for conventional security tasks such as cyber-attack detection. We show these tasks can be either formulated as classification problems or anomaly detection problems, depending on the assumption of available training data.

In the field of compression, we showed that deep learning model such as CNNs can be used for information retrieval from coded signals in compressed sensing systems. This method benefits mutually to both compressed sensing systems and deep learning analyzing models. On the one hand, it bypasses sparse reconstruction, which is computational intensive, and thus enables fast feedback to sensors for adaptive adjustments. On the other hand, a more compact representation of signals using compressed sensing also helps reduce the complexity and parameter size of analyzing models. Besides, deep learning based lossy image codecs are also discussed in the thesis. Our contribution in this field is Substitutional Neural Image Compression, which finds the optimal substitutional image input for a particular compression target (e.g. bit-rate) given a pre-trained neural compression model.

There are many potential future extensions of the works of this thesis. In the field of stochastic defense, we are curious about if there are randomization schemes that surpass Hierarchical Random Switching (HRS) in terms of security-accuracy trade-off. In fact, for defense methods that result in increased parameter size such as HRS, a comprehensive evaluation metric should also take the memory consumption into account, leading to a more generalized three-factor trade-off among adversarial robustness, accuracy and memory consumption. How to fairly evaluate and compare different defense method in the presence of this three-factor trade-off is another interesting problem. In the field of anomaly detection using auto-encoders, an issue that we encountered in our previous works is the lack of theoretical guidance of model architecture design. As we discussed, the anomaly detection performance may not be improved by adding up complexity of the detection model as a good detection models needs to amplify the difference between normal and anomalous data instead of reducing reconstruction error for all data points. In our experiment, the model architectures are usually decided empirically by trying and comparing multiple architectures. But they are still lack of either theoretical performance guarantee or effective hyper-parameter search methods.

Our proposed approach of classifying coded signals illuminates a promising direction of information retrieval without sparse reconstruction in compressed sensing systems. However, it remains to demonstrate how feedback information is used to build a close-loop control system. One example of sensor parameter in compressed sensing systems is the compression ratio. Previous works in this field indicate that the reconstruction quality depends on both CR and velocity of the moving object. Therefore, using our propose approach, the CR of sensing which is decided by the exposure pattern, can be dynamically adjusted for a better sensing quality. Moreover, information extraction in practice could be far more complicated than a simple classification task. This is something worthy of exploring in the future work. In addition, the key approach used in SNIC can also be applied

to other compression tasks easily, such as compression of audio and video. Moreover, for certain applications, the strict distortion-based reconstruction quality metric can be relaxed. For instance, if the compressed images are known to be used for a certain task, we could replace the distortion loss with task-specific loss for task-aware compression.

## References

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283.
- Agustsson, E., Tschannen, M., Mentzer, F., Timofte, R., and Gool, L. V. (2019). Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 221–231.
- Alzaylaee, M. K., Yerima, S. Y., and Sezer, S. (2020). D1-droid: Deep learning based android malware detection using real devices. *Computers & Security*, 89:101663.
- Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182.
- An, J. and Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18.
- Athalye, A., Carlini, N., and Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*.
- Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. (2017). Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*.
- Backes, M., Dürmuth, M., Gerling, S., Pinkal, M., and Sporleder, C. (2010). Acoustic side-channel attacks on printers. In *USENIX Security symposium*, pages 307–322.
- Ballé, J., Laparra, V., and Simoncelli, E. P. (2016). End-to-end optimization of nonlinear transform codes for perceptual quality. In *2016 Picture Coding Symposium (PCS)*, pages 1–5. IEEE.
- Ballé, J., Minnen, D., Singh, S., Hwang, S. J., and Johnston, N. (2018). Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*.

- Baraniuk, R., Davenport, M., DeVore, R., and Wakin, M. (2008). A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 28(3):253–263.
- Bartkewitz, T. and Lemke-Rust, K. (2012). Efficient template attacks based on probabilistic multi-class support vector machines. In *International Conference on Smart Card Research and Advanced Applications*, pages 263–276. Springer.
- Bellard, F. (2015). Bpg image format. URL <https://bellard.org/bpg>.
- Bhagoji, A. N., Cullina, D., Sitawarin, C., and Mittal, P. (2018). Enhancing robustness of machine learning systems via data transformations. In *Information Sciences and Systems (CISS), 2018 52nd Annual Conference on*, pages 1–5. IEEE.
- Brownlee, J. (2017). *Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems*. Machine Learning Mastery.
- Candes, E. J. and Tao, T. (2005). Decoding by linear programming. *IEEE transactions on information theory*, 51(12):4203–4215.
- Carlini, N. and Wagner, D. (2017a). Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM.
- Carlini, N. and Wagner, D. (2017b). Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE.
- Chahal, K. S. and Dey, K. (2018). A survey of modern object detection literature using deep learning. *arXiv preprint arXiv:1808.07256*.
- Chen, P.-Y., Sharma, Y., Zhang, H., Yi, J., and Hsieh, C.-J. (2017). Ead: elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114*.
- Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.
- Cheng, M., Xu, Q., Lv, J., Liu, W., Li, Q., and Wang, J. (2016). Ms-lstm: A multi-scale lstm model for bgp anomaly detection. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE.
- Collobert, R., Bengio, S., and Mariéthoz, J. (2002). Torch: a modular machine learning software library. Technical report, Idiap. [https://ronan.collobert.com/pub/matos/2002\\_torch\\_rr.pdf](https://ronan.collobert.com/pub/matos/2002_torch_rr.pdf).

- Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.
- Das, N., Shanbhogue, M., Chen, S.-T., Hohman, F., Chen, L., Kounavis, M. E., and Chau, D. H. (2017). Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression. *arXiv preprint arXiv:1705.02900*.
- Dau, H. A., Ciesielski, V., and Song, A. (2014). Anomaly detection using replicator neural networks trained on examples of one class. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 311–322. Springer.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.
- Dhillon, G. S., Azzadenesheli, K., Bernstein, J. D., Kossafi, J., Khanna, A., Lipton, Z. C., and Anandkumar, A. (2018). Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*.
- Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634.
- Du, M., Li, F., Zheng, G., and Srikumar, V. (2017a). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298.
- Du, W., Wang, Y., and Qiao, Y. (2017b). Recurrent spatial-temporal attention network for action recognition in videos. *IEEE Transactions on Image Processing*, 27(3):1347–1360.
- Ebrahimi Kahou, S., Michalski, V., Konda, K., Memisevic, R., and Pal, C. (2015). Recurrent neural networks for emotion recognition in video. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 467–474.
- Elsayed, G. F., Goodfellow, I., and Sohl-Dickstein, J. (2018). Adversarial reprogramming of neural networks. *arXiv preprint arXiv:1806.11146*.
- Garg, S., Kaur, K., Kumar, N., Kaddoum, G., Zomaya, A. Y., and Ranjan, R. (2019). A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. *IEEE Transactions on Network and Service Management*, 16(3):924–935.
- Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. *2015 ICLR*, arXiv preprint arXiv:1412.6572.

- Goyal, P., Pandey, S., and Jain, K. (2018). Deep learning for natural language processing. *Deep Learning for Natural Language Processing: Creating Neural Networks with Python [Berkeley, CA]: Apress*, pages 138–143.
- Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE.
- Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610.
- Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386.
- Grosse, K., Manoharan, P., Papernot, N., Backes, M., and McDaniel, P. (2017). On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*.
- Gržinic, T., Kišasondi, T., Šaban, J., and AG, H. A.-A.-B. I. (2013). Detecting anomalous web server usage through mining access logs. In *Central European Conference on Information and Intelligent Systems*, pages 228–296.
- Gu, T., Liu, K., Dolan-Gavitt, B., and Garg, S. (2019). Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244.
- Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. (2014). Deep learning for real-time atari game play using offline monte-carlo tree search planning. *Advances in neural information processing systems*, 27:3338–3346.
- Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.
- Helmstaedter, M., Briggman, K. L., Turaga, S. C., Jain, V., Seung, H. S., and Denk, W. (2013). Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500(7461):168–174.
- Heuser, A. and Zohner, M. (2012). Intelligent machine homicide. In *International Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE 2012. Lecture Notes in Computer Science, vol 7275*, pages 249–264. Springer.
- Hitomi, Y., Gu, J., Gupta, M., Mitsunaga, T., and Nayar, S. K. (2011). Video from a single coded exposure photograph using a learned over-complete dictionary. In *2011 IEEE International Conference on Computer Vision (ICCV)*, pages 287–294. IEEE.
- Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

- Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., and Vandewalle, J. (2011). Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302.
- Hou, R., Chen, C., Sukthankar, R., and Shah, M. (2019). An efficient 3d cnn for action/object segmentation in video. *arXiv preprint arXiv:1907.08895*.
- Huang, J., Zhou, W., Li, H., and Li, W. (2015). Sign language recognition using 3d convolutional neural networks. In *2015 IEEE international conference on multimedia and expo (ICME)*, pages 1–6. IEEE.
- Jensen, L., Brown, G., Wang, X., Harer, J., and Chin, S. (2019). Deep learning for minimal-context block tracking through side-channel analysis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3207–3211. IEEE.
- Ji, S., Xu, W., Yang, M., and Yu, K. (2013). 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231.
- Jouppi, N., Young, C., Patil, N., and Patterson, D. (2018). Motivation for and evaluation of the first tensor processing unit. *IEEE Micro*, 38(3):10–19.
- Justesen, N., Bontrager, P., Togelius, J., and Risi, S. (2019). Deep learning for video game playing. *IEEE Transactions on Games*, 12(1):1–20.
- Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kingma, F. H., Abbeel, P., and Ho, J. (2019). Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. *arXiv preprint arXiv:1905.06845*.
- Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *Advances in cryptology—CRYPTO’99*, pages 789–789. Springer.
- Kocher, P. C. (1996). Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer.
- Kolias, C., Kambourakis, G., Stavrou, A., and Voas, J. (2017). Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84.
- Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., and Roli, F. (2018). Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 533–537. IEEE.

- Kolosnjaji, B., Zarras, A., Webster, G., and Eckert, C. (2016). Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence*, pages 137–149. Springer.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.
- Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*.
- Le, P. and Zuidema, W. (2016). Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs. *CoRR*.
- Lerman, L., Bontempi, G., and Markowitch, O. (2011). Side channel attack: an approach based on machine learning. *Center for Advanced Security Research Darmstadt*, pages 29–41.
- Li, J., Li, B., Xu, J., Xiong, R., and Gao, W. (2018a). Fully connected network-based intra prediction for image coding. *IEEE Transactions on Image Processing*, 27(7):3236–3247.
- Li, J., Liu, X., Zhang, M., and Wang, D. (2020). Spatio-temporal deformable 3d convnets with attention for action recognition. *Pattern Recognition*, 98:107037.
- Li, Y., Li, L., Li, Z., Yang, J., Xu, N., Liu, D., and Li, H. (2018b). A hybrid neural network for chroma intra prediction. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 1797–1801. IEEE.
- Li, Y., Liu, D., Li, H., Li, L., Wu, F., Zhang, H., and Yang, H. (2018c). Convolutional neural network-based block up-sampling for intra frame coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(9):2316–2330.
- Lin, G., Zhang, Y., Xu, G., and Zhang, Q. (2019). Smoke detection on video sequences using 3d convolutional neural networks. *Fire Technology*, 55(5):1827–1847.
- Lin, J., Liu, D., Li, H., and Wu, F. (2018). Generative adversarial network-based frame extrapolation for video coding. In *2018 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE.
- Liu, H., Chen, T., Guo, P., Shen, Q., Cao, X., Wang, Y., and Ma, Z. (2019). Non-local attention optimized deep image compression. *arXiv preprint arXiv:1904.09757*.

- Liu, K., Liu, D., Li, H., and Wu, F. (2018a). Convolutional neural network-based residue super-resolution for video coding. In *2018 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE.
- Liu, K. S., Li, B., and Gao, J. (2018b). Generative model: Membership attack, generalization and diversity. *CoRR*, *abs/1805.09898*.
- Liu, X., Cheng, M., Zhang, H., and Hsieh, C.-J. (2017). Towards robust neural networks via random self-ensemble. *arXiv preprint arXiv:1712.00673*.
- Liu, X., Si, S., Zhu, X., Li, Y., and Hsieh, C.-J. A unified framework for data poisoning attack to graph-based semi-supervised learning. *arXiv preprint arXiv:1910.14147*, *year=2019*.
- Livnat, A., Papadimitriou, C., Pippenger, N., and Feldman, M. W. (2010). Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences*, 107(4):1452–1457.
- Long, Y., Bindschaedler, V., Wang, L., Bu, D., Wang, X., Tang, H., Gunter, C. A., and Chen, K. (2018). Understanding membership inferences on well-generalized learning models. *arXiv preprint arXiv:1802.04889*.
- Lu, J., Issaranon, T., and Forsyth, D. (2017). Safetynet: Detecting and rejecting adversarial examples robustly. *CoRR*, *abs/1704.00103*.
- Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. International Conference on Machine Learning (ICML)*, volume 30.
- Machlica, L., Bartos, K., and Sofka, M. (2017). Learning detectors of malicious web requests for intrusion detection in network traffic. *arXiv preprint arXiv:1702.02530*.
- Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.
- Maimó, L. F., Gómez, Á. L. P., Clemente, F. J. G., Pérez, M. G., and Pérez, G. M. (2018). A self-adaptive deep learning-based system for anomaly detection in 5g networks. *IEEE Access*, 6:7700–7712.
- Martinasek, Z. and Zeman, V. (2013). Innovative method of the power analysis. *Radio-engineering*, 22(2):586–594.
- McDermott, C. D., Majdani, F., and Petrovski, A. V. (2018). Botnet detection in the internet of things using deep learning approaches. In *2018 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE.

- Meng, D. and Chen, H. (2017). Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147. ACM.
- Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., and Van Gool, L. (2018). Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4394–4402.
- Mi, G., Gao, Y., and Tan, Y. (2015). Apply stacked auto-encoder to spam detection. In *International Conference in Swarm Intelligence*, pages 3–15. Springer.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Mordvintsev, A., Olah, C., and Tyka, M. (2015). Inceptionism: Going deeper into neural networks. <https://research.google/pubs/pub45507/>.
- Nath, V. and Levinson, S. E. (2014). *Autonomous robotics and deep learning*. Springer.
- Noda, K., Yamaguchi, Y., Nakadai, K., Okuno, H. G., and Ogata, T. (2015). Audio-visual speech recognition using deep learning. *Applied Intelligence*, 42(4):722–737.
- Oh, S. J., Schiele, B., and Fritz, M. (2019). Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144. Springer.
- Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*, 2(11):e7.
- Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE.
- Pareschi, F., Albertini, P., Frattini, G., Mangia, M., Rovatti, R., and Setti, G. (2016). Hardware-algorithms co-design and implementation of an analog-to-information converter for biosignals based on compressed sensing. *IEEE transactions on biomedical circuits and systems*, 10(1):149–162.
- Pascanu, R., Stokes, J. W., Sanossian, H., Marinescu, M., and Thomas, A. (2015). Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1916–1920. IEEE.
- Pierson, H. A. and Gashler, M. S. (2017). Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31(16):821–835.

- Quisquater, J.-J. and Samyde, D. (2001). Electromagnetic analysis (ema): Measures and counter-measures for smart cards. *Smart Card Programming and Security*, pages 200–210.
- Rabbani, M. (2002). Jpeg2000: Image compression fundamentals, standards and practice. *Journal of Electronic Imaging*, 11(2):286.
- Rippel, O. and Bourdev, L. (2017). Real-time adaptive image compression. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2922–2930. JMLR. org.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.
- Sabokrou, M., Fathy, M., and Hoseini, M. (2016). Video anomaly detection and localisation based on the sparsity and reconstruction error of auto-encoder. *Electronics Letters*, 52(13):1122–1124.
- Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4. ACM.
- Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76.
- Santurkar, S., Budden, D., and Shavit, N. (2018). Generative compression. In *2018 Picture Coding Symposium (PCS)*, pages 258–262. IEEE.
- Schiopu, I. and Munteanu, A. (2019). Deep-learning based lossless image coding. *IEEE Transactions on Circuits and Systems for Video Technology*.
- Sengupta, S., Chakraborti, T., and Kambhampati, S. (2018). Mtdeep: boosting the security of deep neural nets against adversarial attacks with moving target defense. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.
- Shen, Q., Cai, J., Liu, L., Liu, H., Chen, T., Ye, L., and Ma, Z. (2018). Codedvision: Towards joint image understanding and compression via end-to-end learning. In *Pacific Rim Conference on Multimedia*, pages 3–14. Springer.
- Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

- Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.
- Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Skodras, A., Christopoulos, C., and Ebrahimi, T. (2001). The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.
- Tatwawadi, K. (2018). Deepzip: Lossless compression using recurrent networks. *URL <https://web.stanford.edu/class/cs224n/reports/2761006.pdf>*.
- Theis, L., Shi, W., Cunningham, A., and Huszár, F. (2017). Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*.
- Toderici, G., O’Malley, S. M., Hwang, S. J., Vincent, D., Minnen, D., Baluja, S., Covell, M., and Sukthankar, R. (2015). Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085*.
- Toderici, G., Vincent, D., Johnston, N., Jin Hwang, S., Minnen, D., Shor, J., and Covell, M. (2017). Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5306–5314.
- Torres, P., Catania, C., Garcia, S., and Garino, C. G. (2016). An analysis of recurrent neural networks for botnet detection behavior. In *2016 IEEE biennial congress of Argentina (ARGENCON)*, pages 1–6. IEEE.
- Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. (2016). Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618.
- Tzortzis, G. and Likas, A. (2007). Deep belief networks for spam filtering. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, volume 2, pages 306–309. IEEE.
- Wang, S., Wang, X., Ye, S., Zhao, P., and Lin, X. Defending dnn adversarial attacks with pruning and logits augmentation. *Accepted by IEEE Global Conference on Signal and Information Processing (GlobalSIP) 2018*.

- Wang, S., Wang, X., Zhao, P., Wen, W., Kaeli, D., Chin, P., and Lin, X. (2018a). Defensive dropout for hardening deep neural networks under adversarial attacks. In *Proceedings of the International Conference on Computer-Aided Design*, page 71. ACM.
- Wang, X., Wang, S., Chen, P.-Y., Lin, X., and Chin, P. Block switching: A stochastic approach for deep learning security. *Workshop on Adversarial Learning Methods for Machine Learning and Data Mining at KDD 2019*.
- Wang, X., Wang, S., Chen, P.-Y., Lin, X., and Chin, P. (2020). Advms: A multi-source multi-cost defense against adversarial attacks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2902–2906. IEEE.
- Wang, X., Wang, S., Chen, P.-Y., Wang, Y., Kulis, B., Lin, X., and Chin, P. (2019). Protecting neural networks with hierarchical random switching: towards better robustness-accuracy trade-off for stochastic defenses. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 6013–6019. AAAI Press.
- Wang, X., Zhang, J., Xiong, T., Tran, T. D., Chin, S. P., and Etienne-Cummings, R. (2018b). Using deep learning to extract scenery information in real time spatiotemporal compressed sensing. In *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pages 1–4. IEEE.
- Wang, X., Zhou, Q., Harer, J., Brown, G., Qiu, S., Dou, Z., Wang, J., Hinton, A., Gonzalez, C. A., and Chin, P. (2018c). Deep learning-based classification and anomaly detection of side-channel signals. In *Cyber Sensing 2018*, volume 10630, page 1063006. International Society for Optics and Photonics.
- Wang, Z., Simoncelli, E. P., and Bovik, A. C. (2003). Multiscale structural similarity for image quality assessment. In *The Thirty-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee.
- Wu, X., Dong, W., Zhang, X., and Shi, G. (2012). Model-assisted adaptive recovery of compressed sensing with imaging applications. *IEEE Transactions on Image Processing*, 21(2):451–458.
- Xiong, H. Y., Alipanahi, B., Lee, L. J., Bretschneider, H., Merico, D., Yuen, R. K., Hua, Y., Gueroussov, S., Najafabadi, H. S., Hughes, T. R., et al. (2015). The human splicing code reveals new insights into the genetic determinants of disease. *Science*, 347(6218).
- Xu, H., Chen, W., Zhao, N., Li, Z., Bu, J., Li, Z., Liu, Y., Zhao, Y., Pei, D., Feng, Y., et al. (2018). Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, pages 187–196.

- Xu, W., Evans, D., and Qi, Y. (2017). Feature squeezing mitigates and detects carlini/wagner adversarial examples. *arXiv preprint arXiv:1705.10686*.
- Xu, Z., Hu, J., and Deng, W. (2016). Recurrent convolutional neural network for video classification. In *2016 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE.
- Yang, H., Yuan, C., Li, B., Du, Y., Xing, J., Hu, W., and Maybank, S. J. (2019). Asymmetric 3d convolutional neural networks for action recognition. *Pattern recognition*, 85:1–12.
- Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75.
- Yuan, Z., Lu, Y., Wang, Z., and Xue, Y. (2014). Droid-sec: deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 371–372.
- Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. (2015). Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702.
- Zelnik-Manor, L., Rosenblum, K., and Eldar, Y. C. (2011). Sensing matrix optimization for block-sparse decoding. *IEEE Transactions on Signal Processing*, 59(9):4300–4312.
- Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., and Cong, J. (2015a). Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170.
- Zhang, J., Mitra, S., Suo, Y., Cheng, A., Xiong, T., Michon, F., Welkenhuysen, M., Kloosterman, F., Chin, P. S., Hsiao, S., et al. (2015b). A closed-loop compressive-sensing-based neural recording system. *Journal of neural engineering*, 12(3):036005.
- Zhang, J., Newman, J. P., Wang, X., Thakur, C. S., Rattray, J., Etienne-Cummings, R., and Wilson, M. A. (2020). A closed-loop, all-electronic pixel-wise adaptive imaging system for high dynamic range videography. *IEEE Transactions on Circuits and Systems I: Regular Papers*.
- Zhang, J., Xiong, T., Tran, T., Chin, S., and Etienne-Cummings, R. (2016). Compact all-cmos spatiotemporal compressive sensing video camera with pixel-wise coded exposure. *Optics express*, 24(8):9013–9024.

- Zhang, Y., Wang, J., and Yang, X. (2017). Real-time vehicle detection and tracking in video based on faster r-cnn. In *Journal of Physics: Conference Series*, volume 887, page 012068.
- Zhao, Z., Wang, S., Wang, S., Zhang, X., Ma, S., and Yang, J. (2018). Enhanced bi-prediction with convolutional neural network for high-efficiency video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(11):3291–3301.
- Zhao, Z.-Q., Zheng, P., Xu, S.-t., and Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232.
- Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., and Oliva, A. (2014). Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495.
- Zhou, X., Wang, C., Xu, Y., Wang, X., and Chin, P. (2017). Domain specific inpainting with concurrently pretrained generative adversarial networks. In *Signal and Information Processing (GlobalSIP), 2017 IEEE Global Conference on*. IEEE.
- Zhou, X., Wang, X., and Chin, P. (2018a). Learning in parrondo’s paradox. In *The 29th International Conference on Game Theory*. Stony Brook University.
- Zhou, Y., Kantarcioglu, M., and Xi, B. (2018b). Breaking transferability of adversarial samples with randomness. *arXiv preprint arXiv:1805.04613*.

# CURRICULUM VITAE

