12-12-2022

# Towards Robust and Interpretable Deep Learning

Xiang Li
*Georgia State University*

# Towards Robust and Interpretable Deep Learning

by

Xiang Li

Under the Direction of Shihao Ji, Ph.D.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2022

## ABSTRACT

Recent progress in deep learning has enabled applications in many areas, such as business, security, and science, that could impact our lives. Despite recent advances in these domains, deep neural network models have been shown to be vulnerable to adversarial attacks and lack of interpretability of their predictions. Therefore, it is crucial to investigate robust and interpretable deep learning models and algorithms to solve the above issues.

In this dissertation, we proposed a series of algorithms for delivering robust and interpretable deep learning methods. To begin with, we study the problem of how to defend against adversarial attacks with a purification-based algorithm called defense-VAE. Secondly, we proposed GDPA, a patch attack algorithm that can be readily used in adversarial training. With this algorithm, we can train deep learning models that are robust to patch attacks. Thirdly, we proposed an interpretation algorithm NICE, which learns sparse masks on input images. We also showed how to use this interpretation algorithm for semantic compression on images. Fourthly, we applied NICE on brain MRI data for the schizophrenia discrimination task, in which we detected the important regions of the brain for schizophrenia discrimination. Lastly, we proposed the PSP algorithm, which applied parameter-wise smooth policy in the PPO algorithm to improve the performance and robustness of reinforcement learning (RL) agents.

INDEX WORDS: Deep learning, Reinforcement learning, Robustness, Interpretability, Brain MRI image, Adversarial defense

# Towards Robust and Interpretable Deep Learning

by

Xiang Li

## ACKNOWLEDGMENTS

My enjoyable and rewarding Ph.D. journey has come to an end. I owe great gratitude to the many incredible individuals I met along this journey.

First, I sincerely appreciate my Ph.D. advisor, Professor Shihao Ji, for supporting and advising me throughout these five years. I want to thank him for his tremendous support and guidance during these years. His words of guidance and research insights have been crucial in keeping me focused on and achieving my research goals. His intellect and dedication to research have always been inspirational to me and will keep inspiring me in the future.

I'd also like to thank my thesis committee members: Dr. Rajshekhar Sunderraman, Dr. Jingyu Liu, and Dr. Xiaojing Ye, for their valuable advice. They have been leaders in their respective research areas, and their advice has influenced many aspects of this thesis.

I thank my colleagues in our lab: Xiulong Yang, Yang Li, Yang Ye, Qing Su, Hui Ye, Songyang Li, and Krishanu Sarker. I will always miss the time I spent with you, having lunch or discussing our research.

I also thank my colleagues and friends outside our lab: Dr. Jiayu Chen, Dr. Yingshu Li, Ms. Tammie Dudley, Ms. Tyra Robinson, Mr. Paul Bryan, Xucan Chen, Fengpan Zhao, Xueli Xiao, Xueting Liao, Jun Yi, Mu Ge, Dhara P Shah, William Ashbee, etc.

Finally, I dedicate this thesis to my family. I want to thank my wife, Yihan Wang, for giving me love at difficult times. I want to thank my father for teaching me how to be a responsible man and my mother for teaching me how to love people unconditionally.

## TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1
# Introduction

Deep neural networks (DNNs) have demonstrated remarkable success in solving complex prediction tasks. However, recent studies show that they are particularly vulnerable to adversarial attacks Biggio et al. (2013); Papernot et al. (2016a); Szegedy et al. (2013) in the form of small perturbations to inputs that lead DNNs to predict incorrect outputs. While various defense algorithms Papernot et al. (2016c); Moosavi-Dezfooli et al. (2016b) have been developed, they are later defeated by stronger attack methods Biggio et al. (2013); Papernot et al. (2016a); Szegedy et al. (2013). Such unexpected behavior of deep neural network models and potential security issue highlights the need to interpret the models themselves. A new line of research in interpretable deep learning emerged and aimed to interpret the behavior of such black-box models. This chapter presents a general introduction to robust and interpretable deep learning areas.

## 1.1 Robust Deep Learning

Security and integrity of the applications posed great concern recently as the deep neural networks make their way from research to the real world. For example, adversaries transform input features craftily to make the transformed images imperceptible to humans but can fool a trained deep neural network model into outputting incorrect predictions.

Since Szegedy et al. (2013) first discover that neural networks are vulnerable to adversarial attack, the area of adversarial learning on deep neural networks has become a hotspot. The adversarial attack methods can be divided into three categories according to different stages of the target model: training stage attack, testing stage attack, and deployment stage attack. We only describe the attack during the training stage and testing stage in this paper because the testing stage attack and deployment stage attack are very similar.

The training stage attack, also called poison attack, refers to carrying out attacks by modifying the training dataset, including input features and labels, during the training process of the target model.

The testing stage attack refers to constructing adversarial examples by utilizing the knowledge of parameters, structures, and algorithms of the target model. If adversaries possess this knowledge when conducting the attack, it is called a white-box attack. On the other hand, if the adversaries cannot obtain the target model's information, they can train a substitute model by querying the target model, which is called a black-box attack.

### 1.2 Interpretable Deep Learning

Due to that deep learning algorithms with the black-box property start to substitute the previously entrusted decision-making process by humans, it is necessary for these deep learning models to interpret the decision by themselves. The mistrust between human and deep learning models exists despite the success in a broad range of tasks: including computer vision, natural language processing, graph data analysis, etc. Moreover, deep neural networks are found to be unreliable, biased, and not robust. This unexpected behavior highlights the need to explain and interpret deep learning algorithms.

The purpose of the interpretability of deep learning models is to describe the internal process of a model such that humans can understand the decision-making process. In other words, we want to produce descriptions from the mechanism of the deep learning model that is understandable by a human.

## 1.3 Relationship between Robustness and Interpretability

The robustness and interpretability have close connections in several aspects. Firstly, interpreting a deep learning model helps improve the model's robustness. On the one hand, if adversaries understand how the target model works, they can use it to discover model weaknesses and conduct attacks accordingly. On the other hand, if model developers understand how the model operates, they may identify the vulnerability and find remediation in advance. Some recent work involves interpretability in the analysis of adversarial robustness. Secondly, many existing adversary learning works can be analyzed from another angle as the extension of model interpretation. The methods of robust deep learning and interpretable deep learning are overlapped significantly Liu et al. (2021).

## 1.4 Parameter-wise Robustness of Deep neural network

Most study on deep neural network robustness is about the vulnerability of DNNs against input data corruption. However, the vulnerability of DNNs does not only exhibit in input data. As functions of input data and model parameters, the parameters of neural networks are a source of vulnerability. For neural networks deployed on electronic computers, parameter attacks can be conducted as training data poisoning, bit flipping, compression, or quantization. Suppose the neural network is deployed in physical devices. In this case, advances in hardware neural networks also call for a study on parameter robustness because of hardware damage and environmental noise, which is parameter corruption. Furthermore, a study on parameter robustness can improve our understanding of different mechanisms in neural networks, inspiring innovation in architecture design and the training process.

## 1.5 Robust Reinforcement Learning

Combining the power of classic reinforcement learning algorithms and modern deep neural network techniques, deep reinforcement learning (DRL) is capable of training agents for complex tasks with inputs of high dimensional observations, such as pixels Mnih et al. (2013). It has proven that DRL is successful across a wide range of problems, including game playing Mnih et al. (2013); Silver et al. (2016, 2017); Vinyals et al. (2019), robot control Tai et al. (2018, 2017); Zhelo et al. (2018); Hwangbo et al. (2019), natural language processing Hudson & Manning (2018); Wang et al. (2018); Wu et al. (2018), autonomous driving Talpaert et al. (2019); Milz et al. (2018); Li et al. (2020), and recommendation systems Zheng et al. (2018); Chen et al. (2019c), etc.

Despite achieving excellent performance with DRL on various tasks, the presence of adversarial examples in DNNs and many successful attacks on DRL encourages us to study robust DRL algorithms. When an RL agent obtains its current observations, it may contain uncertainty that originates naturally from unavoidable sensor errors or equipment inaccuracy. A policy not robust to such uncertainty can lead to destructive failures. For this reason, training a robust policy is crucial to the RL algorithms.

## 1.6 Dissertation Organization

The overall structure of this dissertation is organized as below. To begin with, we briefly introduce robust deep learning and interpretable deep learning in Chapter 1. From Chapter 2 to Chapter 6, we present five algorithms on robust and interpretable deep learning.

Chapter 2 proposes a simple yet effective defense algorithm *Defense-VAE* that uses variational autoencoder (VAE) to purge adversarial perturbations from contaminated images. The proposed method is generic and can defend white-box and black-box attacks without retraining the original CNN classifiers. It can further strengthen the defense by retraining CNN or end-to-end finetuning the whole pipeline. In addition, the proposed method is very efficient compared to the optimization-based alternatives, such as Defense-GAN, since no iterative optimization is needed for online prediction.

Chapter 3 proposes an end-to-end patch attack algorithm, Generative Dynamic Patch Attack (GDPA), which generates both patch pattern and patch location adversarially for each input image. First, we show that GDPA is a generic attack framework that can produce dynamic/static and visible/invisible patches with a few configuration changes. Secondly, GDPA can be readily integrated into adversarial training to improve model robustness to various adversarial attacks.

Chapter 4 proposes a novel end-to-end Neural Image Compression and Explanation (NICE) framework that learns to (1) explain the predictions of convolutional neural networks (CNNs) and (2) subsequently compress the input images for efficient storage or transmission. Specifically, NICE generates a sparse mask over an input image by attaching a stochastic binary gate to each pixel of the image, whose parameters are learned through the interaction with the CNN classifier to be explained. The generated mask is able to capture the saliency of each pixel measured by its influence on the final prediction of CNN; it can also be used to produce a mixed-resolution image, where important pixels maintain their original high resolution and insignificant background pixels are subsampled to a low resolution.

Chapter 5 introduces a sparse deep neural network (DNN) approach to identify sparse and interpretable features for schizophrenia (SZ) discrimination. An $L_0$ regularization is implemented on the network's input layer for sparse feature selection, which can later be interpreted based on importance weights. We applied the proposed approach on a large multi-study cohort (N = 1,684) with gray matter volume (GMV) and single nucleotide polymorphism (SNP) data for SZ discrimination.

Chapter 6 proposes to train a parameter-wise smooth policy network in PPO to tackle these challenges. Specifically, we introduce a Parameter-wise Smooth Policy (PSP) regularization to enforce the outputs of a policy or loss values not to change much when injecting small perturbations to the parameters of policy networks. With the PSP regularization, we improve the stability of PPO and promote sample efficiency by learning policies with high entropy outputs for exploration, thus improving the cumulative rewards. Furthermore, models trained with our PSP regularization are more robust against random and adversarial parameter corruption.

## 1.7 List of Publications

### *1.7.1 Refereed Publications*

[*denotes equal contribution]

1. **Xiang Li**, Shihao Ji, “Generative Dynamic Patch Attack”, BMVC, 2021.[link]
2. J. Chen*, **Xiang Li***, V. D. Calhoun, J. A. Turner, T. G. M. van Erp, L. Wang, O. A. Andreassen, I. Agartz, L. T. Westlye, E. Jonsson, J. M. Ford, D. H. Mathalon, F. Macciardi, D. S. O’Leary, J. Liu, Shihao Ji, “Sparse Deep Neural Networks on Imaging Genetics for Schizophrenia Case-Control Classification,” Human Brain Mapping (IF: 4.55), March 2021. [Link]
3. **Xiang Li**, Shihao Ji, “Neural Image Compression and Explanation,” IEEE Access (IF: 3.75), Vol. 8, Nov. 30, 2020. [link]
4. **Xiang Li**, Shihao Ji, “Defense-VAE: A Fast and Accurate Defense against Adversarial Attacks,” Machine Learning for Cybersecurity (ECML workshop on MLCS), Würzburg, Germany, Sept. 2019. [link]
5. Yinying Wang, Sing Hui Lee, Briana Keith, Yasmine Bey, Xiulong Yang, **Xiang Li**, Shihao Ji, “A Convenient Rhetoric or Substantial Change of Teacher Diversity? A Text Mining Approach and Systemic Review.” 2021 AERA Annual Meeting, Nov, 2020.
6. J. Chen*, **Xiang Li***, V. Calhoun, J. Turner, T.G.M. Erp, L. Wang, O. Andreassen, I. Agartz, L. Westlye, J. Liu, and Shihao Ji, “Sparse Deep Neural Networks on Imag-

ing Genetics for Schizophrenia Discrimination," The Organization for Human Brain Mapping (OHBM), Montreal, June 2020. [link]

7. Xiulong Yang, Hui Ye, Yang Ye, **Xiang Li**, Shihao Ji, "Generative Max-Mahalanobis Classifiers for Image Classification, Generation and More", ECML, 2021. [link]

8. Yinying Wang, Sing Hui Lee, Briana Keith, Yasmine Bey, Xiulong Yang, **Xiang Li**, Shihao Ji, "A Convenient Rhetoric or Substantial Change of Teacher Racial Diversity? A Text Mining Analysis of Federal, State, and District Documents", Education Policy Analysis Archives. 30, (Jun. 2022), (78). [link]

#### *1.7.2 Under Review*

1. **Xiang Li**, Shihao Ji, "Proximal Policy Optimization with Parameter-wise Smooth Policy", Under review.

# CHAPTER 2
# Defense-VAE: A Fast and Accurate Defense against Adversarial Attacks

## 2.1 Introduction

Deep neural networks (DNNs) have demonstrated remarkable success in solving complex prediction tasks. However, recent studies show that they are particularly vulnerable to adversarial attacks Biggio et al. (2013); Papernot et al. (2016a); Szegedy et al. (2013) in the form of small perturbations to inputs that lead DNNs to predict incorrect outputs. For images, such perturbations are often almost imperceptible to human vision system, while being very effective at fooling DNN-based systems. Both white-box attacks Papernot et al. (2016b) and black-box attacks Papernot et al. (2017) have been proposed to attack DNNs, and they can often fool the network with high probabilities. These attacks pose a serious threat to the applications of DNNs in security-sensitive systems, e.g., identity authentication surveillance, self-driving cars, malware detection, and voice command recognition. As a result, it is critical to develop effective and efficient defense mechanisms to counter adversarial attacks.

In this paper, we propose a simple yet effective defense algorithm called *Defense-VAE* which uses Variational AutoEncoder (VAE) Kingma & Welling (2013); Rezende et al. (2014) to purge the adversarial perturbations from contaminated images before feeding the images to the downstream CNN classifiers. To illustrate the idea, we generate some adversarial images based on the FGSM attack Goodfellow et al. (2014b) with $\epsilon = 0.05$ and $\epsilon = 0.1$ on four popular image classification benchmarks: MNIST Lecun et al. (1998), Fashion-

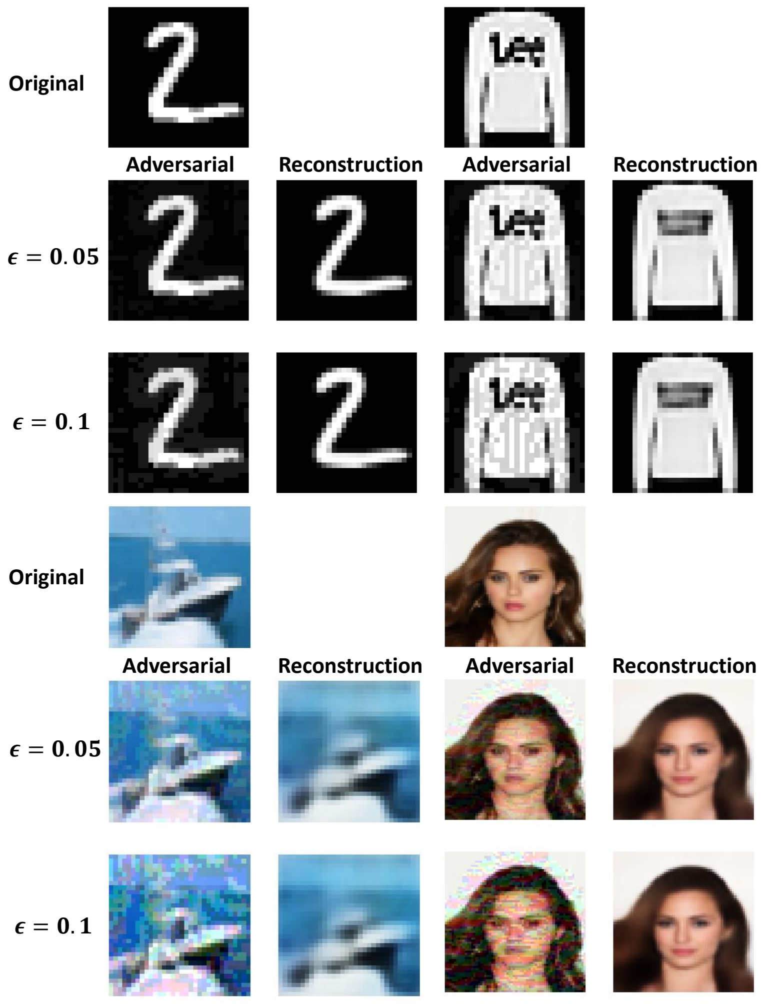


Figure 2.1 Defense-VAE purges adversarial perturbations from contaminated images. Example images are from (left top) MNIST, (right top) Fashion MNIST, (left bottom) CIFAR-10, and (right bottom) CelebA. FGSM Goodfellow et al. (2014b) with $\epsilon = 0.05$ and $\epsilon = 0.1$ are used to generate the adversarial attacks.

MNIST Xiao et al. (2017), CIFAR-10 Krizhevsky (2009) and CelebA Liu et al. (2015). These adversarial images are then fed into Defense-VAE for reconstruction. Figure 2.1 illustrates some of the typical examples from Defense-VAE. As we can see, the Defense-VAE generated images are the faithful reconstructions from the underlying clean images, with the majority of adversarial perturbations removed. As we will demonstrate later, such reconstructed images can recover almost all the accuracy losses due to adversarial attacks, without introducing much computation overhead compared to Defense-GAN Samangouei et al. (2018), a closely related state-of-the-art defense algorithm that is based on Generative Adversarial Networks (GAN) Goodfellow et al. (2014a).

Compared with the state-of-the-art defense algorithms, our method has the following properties:

- Defense-VAE is very generic and can defend both white-box attacks and black-box attacks without the need of retraining the original CNN classifiers, and can further strengthen the defense by retraining or end-to-end finetuning;

- Defense-VAE achieves much higher accuracy than the state-of-the-art defense algorithms on white-box and black-box attacks. Especially, it outperforms Defense-GAN by about 30% in defending black-box attacks on Fashion-MNIST;

- Defense-VAE is very efficient compared to the optimization-based alternatives, such as Defense-GAN, as no iterative optimization is needed for online prediction. From our experiments, it shows that Defense-VAE is about 50x faster than Defense-GAN. This makes our method widely deployable in real-time security-sensitive applications.

A preliminary ECML workshop version of this work was presented earlier Li & Ji (2019). The present work adds to the initial version in significant ways. First, we give an in-depth introduction to Defense-VAE to set up the context to readers. Second, we extend Defense-VAE for adversarial image detection, which allows a flexible integration of Defense-VAE with other defense algorithms to form a defense pipeline, where Defense-VAE can be a component for adversarial detection or adversarial defense or both. Third, we examine the robustness of our model under all sorts of untrained attacks, demonstrating the robustness of Defense-VAE can be solidified by incorporating more adversarial examples from diverse attacking algorithms. Experimentally, we provide more results on black-box defense, adversarial detection and robustness validation. We also provide more details on network architectures and experimental configurations for reproducible research.

## 2.2 Defense-VAE: The Proposed Algorithm

At a high level, Defense-VAE is a defense algorithm that is based on deep generative models for image reconstruction. That is, given an adversarial image as input, the generative model attempts to produce a denoised image that is closely related to the underlying clean image, with the adversarial perturbations removed. As the name suggested, Defense-VAE is built upon Variational AutoEncoder (VAE) Kingma & Welling (2013); Rezende et al. (2014). Therefore, we first give a brief introduction to VAE.

### *2.2.1 Variational Auto-Encoder*

Variational Autoencoder (VAE) Kingma & Welling (2013); Rezende et al. (2014) is one of the most powerful deep generative models that is based on latent variable models. It consists of an encoder network to encode an input image to the latent variable $\boldsymbol{z}$ and a decoder network to decode the latent variable $\boldsymbol{z}$ back to the image domain:

$$\boldsymbol{z} \sim \mathrm{Enc}(\boldsymbol{x}) = q(\boldsymbol{z}|\boldsymbol{x}), \quad \boldsymbol{x} \sim \mathrm{Dec}(\boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z}). \tag{2.1}$$

Since the maximum likelihood (ML) estimate of this latent variable model is intractable, a variational lower bound (ELBO) is optimized instead:

$$\begin{aligned}\mathcal{L}_{\mathrm{VAE}} &= -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})}\right] \\ &= -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{x}|\boldsymbol{z})] + D_{\mathrm{KL}}(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z}))\end{aligned} \tag{2.2}$$

where the first term is the reconstruction error and the second term is a regularization that prefers the posterior to be close to the prior. Typically, a simple unit Gaussian prior is assumed in VAE. To facilitate efficient computation, a diagonal covariance Gaussian posterior is further assumed, which enables the use of the reparameterization trick to reduce the variance of Monte-Carlo sampling Kingma & Welling (2013).

As a generative model, VAE can generate high quality images that follow the similar distribution of the training images.

### *2.2.2 Defense-VAE*

VAE is typically trained to reproduce the same image from an input image. As for adversarial defense, reproducing the same adversarial images is an undesirable task as the adversarial perturbations may be preserved during the image reconstruction. Instead, in Defense-VAE,

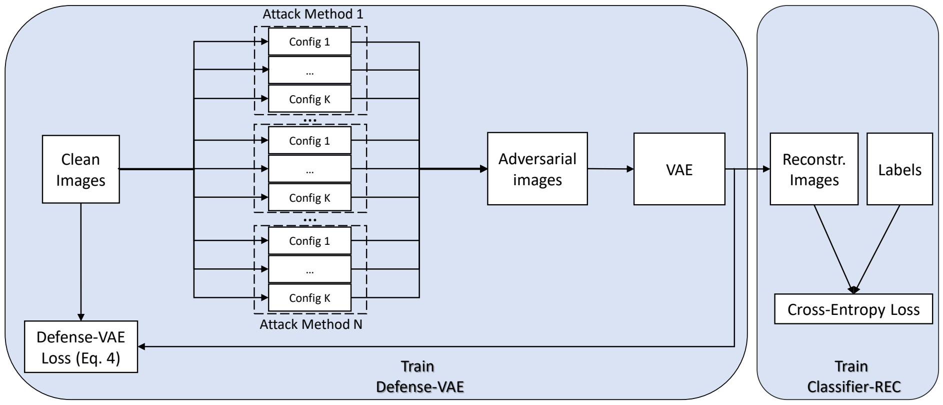


Figure 2.2 Training pipeline of Defense-VAE. Defense-VAE (left) and Classifier-REC (right) can be trained separately, or jointly end-to-end (from scratch or by fine-tuning). See text for more details.

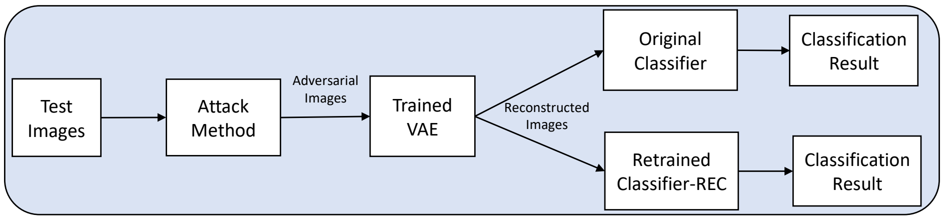


Figure 2.3 Test pipeline of Defense-VAE.

we modify the encoder and the decoder of the latent variable model as follows:

$$\boldsymbol{z} \sim \mathrm{Enc}(\hat{\boldsymbol{x}}) = q(\boldsymbol{z}|\hat{\boldsymbol{x}}), \quad \boldsymbol{x} \sim \mathrm{Dec}(\boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z}), \tag{2.3}$$

where $\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\delta}$ is an adversarial image with the perturbation $\boldsymbol{\delta}$ added on top of a clean image $\boldsymbol{x}$. This adversarial image is encoded to a latent variable $\boldsymbol{z}$, which is decoded to the underlying clean image $\boldsymbol{x}$. Accordingly, the training loss of Defense-VAE is updated as follows:

$$\begin{aligned}\mathcal{L}_{\text{Defense}-\text{VAE}} &= -\mathbb{E}_{q(\boldsymbol{z}|\hat{\boldsymbol{x}})}\left[\log \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z}|\hat{\boldsymbol{x}})}\right] \\ &= -\mathbb{E}_{q(\boldsymbol{z}|\hat{\boldsymbol{x}})}[\log p(\boldsymbol{x}|\boldsymbol{z})] + D_{\text{KL}}(q(\boldsymbol{z}|\hat{\boldsymbol{x}})\|p(\boldsymbol{z})),\end{aligned} \tag{2.4}$$

where the input to Defense-VAE is an adversarial image $\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\delta}$, and the expected output is the underlying clean image $\boldsymbol{x}$. The compatibility between input and output pair is measured by the loss function 2.4.

To train the Defense-VAE model, we can generate adversarial images given any clean image from a training set. Since there are many different adversarial attack algorithms and for each attack algorithm we can generate multiple adversarial images with different configurations, we can in principle generate an unlimited amount of training pairs for Defense-VAE, i.e., multiple adversarial images can be mapped to one clean image. The detailed training pipeline is demonstrated in Figure 2.2 (left). Being an effective approach of generating sufficient training pairs for Defense-VAE, using multiple attack algorithms to produce adversarial training examples will also boost the capability of Defense-VAE to counter an ensemble of adversarial attacks and make Defense-VAE a generic defense algorithm that is robust to a wide range of attacks. As we will discuss later, this ensemble training strategy entails Defense-VAE superior defense capability over Defense-GAN.

Once the Defense-VAE model is trained, we can also use the reconstructed images from Defense-VAE to retrain the downstream CNN classifiers Figure 2.2 (right). As we will see later, the retrained CNN classifier can further boost the defense accuracy over the original CNN classifier.

We can also train the whole pipeline end to end from scratch or finetuning from pre-trained VAE and CNN classifier by optimizing the joint loss function:

$$\mathcal{L}_{\text{End-to-End}} = \mathcal{L}_{\text{Defense-VAE}} + \lambda \mathcal{L}_{\text{Cross-Entropy}}. \tag{2.5}$$

As we will see from the experiments, this end-to-end training can boost the defense accuracy even further.

After training the Defense-VAE model and potentially retraining CNN classifiers or end-to-end finetuning the whole pipeline, we can use the trained Defense-VAE to purge the adversarial perturbations from any contaminated images, and the reconstructed images are then fed to the original CNN classifier or retrained CNN classifier for the final image classification. This test pipeline is shown in Figure 2.3.

## 2.3 Related Work

Adversarial attacks and defenses is one of the active research areas in deep learning, with tens of different attack and defense algorithms developed in the past few years. For a general introduction to this exciting research area and the related terminologies, we refer the readers to Vorobeychik & Kantarcioglu (2018); Yuan et al. (2017); Samangouei et al. (2018) for more details. Here we will focus on the defense algorithms that are most closely related to Defense-VAE.

Defending against adversarial attacks is a challenging task. Different types of defense algorithms Papernot et al. (2016c); Moosavi-Dezfooli et al. (2016b) have been proposed in the past few years. The first type of defense algorithms Dziugaite et al. (2016); Guo et al. (2017a); Luo et al. (2015a) augments the training data to make the DNN model resillient to the trained adversarial attacks. The second type of defense algorithms Gu & Rigazio (2014a); Ross & Doshi-Velez (2017); Lyu et al. (2015a); Nguyen et al. (2017); Nayebi & Ganguli (2017); Gao et al. (2017) modifies the training process by introducing regularization to the objective functions. The third type of defense algorithms Akhtar et al. (2017); Xu et al. (2018); Guo et al. (2017b) attempts to remove the adversarial perturbations via input transformations before feeding the image to the classifier. According to this categorization, our Defense-VAE belongs to the input transformation based defense approach. In the following, we will therefore review the defense algorithms that are closely related to our work.

Adversarial training Goodfellow et al. (2014b); Kurakin et al. (2016) is a popular and well investigated defense approach against adversarial attacks. It attempts to use adversarial images as data augmentation to train a robust classifier. It shows that this method can improve the defense accuracy effectively and sometimes it can even improve the accuracy upon the model trained only on the original clean training set. However, this defense mechanism is more effective in white-box attacks than in black-box attacks due to the gradient masking problem. In Defense-VAE, we also use adversarial examples to improve the robustness of the defense model. However, instead of improving the targeted CNN classifiers directly, adversarial training is used to train a Defense-VAE model to purge adversarial perturbations for the downstream CNN classifiers.

Magnet propsed by Meng and Chen Meng & Chen (2017) is another effective strategy to defend adversarial attacks. Magnet has two phases for defense: detector network and reformer network. Detector network learns the manifold of the normal clean images so that it can detect if an input image is an adversarial. If an image is detected as an adversarial, it will be forwarded to the reformer network, which will modify the adversarial image to the manifold of normal images. In Magnet, the reformer network is trained only on clean images with the goal of reconstructing the same clean input images, while Defense-VAE is trained on adversarial and clean image pairs with the goal of removing the adversarial perturbations from the contaminated images.

Another closely related work is Defense-GAN that is proposed by Samangouei et. al. in Samangouei et al. (2018), where a Generative Adversarial Network (GAN) Goodfellow et al. (2014a) is used to reconstruct a clean image from an adversarial image. Defense-GAN firstly trains a GAN model purely on a training set of clean images, and as such it learns the distribution of the normal images. Then given an adversarial image, multiple iterations of back-propagations are used to identify a proper $\boldsymbol{z}$ from the clean image latent space, such that after decoded through the GAN generator, the reconstructed image is expected to be as close as possible to the adversarial image. Given the non-convex loss function of the GAN generator model, multiple random $\boldsymbol{z}$'s are used to initialize the back-propagation image search. Typically, given an adversarial image, Defense-GAN needs to perform $L$ iterations of back-propagation for each of $R$ random initializations, with the typical values of $L = 200$

and $R = 10$. As a comparison, to reconstruction a clean image, Defense-VAE can directly identify a proper $\boldsymbol{z}$ by forward-propagating an adversarial image through the VAE encoder network, and the $\boldsymbol{z}$ is subsequently used to reconstruct a clean image through the VAE-decoder network. No expensive iterative online optimization is needed in Defense-VAE. As we will discuss later, such reconstructed images are not only more accurate, but the whole process is much faster than Defense-GAN.

### 2.4 Experiments

The details of network architectures used in our experiments are described below. Table 2.1 shows the architectures of the CNN classifiers and their substitute models, which are identical to those used in the Defense-GAN paper Samangouei et al. (2018). This is for the purpose of fair comparison since we compare our method mainly with Defense-GAN Samangouei et al. (2018). The meanings of the notations used in the tables are described below:

- Conv$(C_1, C_2, K, S, P)$ refers to a convolutional layer with input channel $C_1$, output channel $C_2$, filter size $K$, stride $S$ and padding $P$. If $C_1$ is *, it equals to 1 for gray images and 3 for RGB images.

- ConvT$(C_1, C_2, K, S, P)$ refers to a transposed convolutional layer with input channel $C_1$, output channel $C_2$, filter size $K$, stride $S$ and padding $P$.

- FC$(M, N)$ refers to a fully-connected layer with $M$ inputs and $N$ outputs.

- Dropout$(P)$ refers to a dropout layer with dropout probability $P$.

- ReLU refers to the Rectified Linear Unit activation.

- BN refers to a Batch Normalization layer.

Table 2.2 shows the architecture of the Defense-VAE model used in the experiments on MNIST and Fashon-MNIST. The Defense-VAE architecture used for CIFAR-10 Krizhevsky (2009) is largely the same except that it has one additional convolutional layer of Conv(64, 64, 4, 2, 1) in the encoder and one additional transposed convolutional layer of ConvT(64, 64, 4, 2, 1) in the decoder due to the different size of input images compared with MNIST. Similarly, the VAE architecture used for CelebA Liu et al. (2015) has two additional convolutional layers of Conv(64, 64, 4, 2, 1) in the encoder and two additional transposed convolutional layers of ConvT(64, 64, 4, 2, 1) in the decoder. Additionally, we don't use Batch Normalization in the experiments on CIFAR-10 and CelebA.

| A | B | C | D | E |
|---|---|---|---|---|
| Conv(*, 64, 5, 1, 2) | Dropout(0.2) | Conv(*, 128, 3, 1, 1) | FC(200) | FC(200) |
| ReLU | Conv(*, 64, 8, 2, 5) | ReLU | ReLU | ReLU |
| Conv(64, 64, 5, 2, 0) | ReLU | Conv(128, 64, 5, 2, 0) | Dropout(0.5) | FC(200) |
| ReLU | Conv(64, 128, 6, 2, 0) | ReLU | FC(200) | ReLU |
| Dropout(0.25) | ReLU | Dropout(0.25) | ReLU | FC(10) + Softmax |
| FC(128) | Conv(128, 128, 5, 1, 0) | FC(128) | Dropout(0.25) | |
| ReLU | ReLU | ReLU | FC(10) + Softmax | |
| Dropout(0.5) | Dropout(0.5) | Dropout(0.5) | | |
| FC(10) + Softmax | FC(10) + Softmax | FC(10) + Softmax | | |

Table 2.1 The architectures of the classifiers and the substitute models used in the white-box and black-box attacks.

We validate our algorithm on four popular image classification benchmarks: MNIST Lecun et al. (1998), Fashion-MNIST Xiao et al. (2017), CelebA Liu et al. (2015) and CIFAR-10 Krizhevsky (2009). MNIST and Fashion-MNIST are two gray-level image datasets, each containing 60,000 training images and 10,000 test images with the size of $28 \times 28$. While

| **Encoder** | **Decoder** |
|---|---|
| Conv(*, 64, 5, 1, 2) + BN + ReLU | FC(128, 4096) + ReLU |
| Conv(64, 64, 4, 2, 3) + BN + ReLU | ConvT(256, 128, 4, 2, 1) + BN + ReLU |
| Conv(64, 128, 4, 2, 1) + BN + ReLU | ConvT(128, 64, 4, 2, 1) + BN + ReLU |
| Conv(128, 256, 4, 2, 1) + BN + ReLU | ConvT(64, 64, 4, 2, 3) + BN + ReLU |
| FC1(4096, 128), FC2(4096, 128) | ConvT(64, 64, 5, 1, 2) + BN + ReLU |

Table 2.2 The encoder and decoder of Defense-VAE used in the experiments on MNIST and Fashion-MNIST.

MNIST consists of 10 hand-written digits, Fashion-MNIST contains 10 different articles, e.g., shoes, shirts, etc. CelebA contains 202,599 RGB images of human faces, split into training and test sets. We use this dataset for binary classification to distinguish if a face image is from a male or a female. CIFAR-10 contains 10 classes of RGB images of the size of $32 \times 32$, in which 50,000 images are for training and 10,000 images are for test.

We consider both the white-box attacks and the black-box attacks to test the defense performance of our algorithm. For the white-box attacks, FGSM Goodfellow et al. (2014b), Randomized FGSM Kurakin et al. (2016), and CW Carlini & Wagner (2017) attacks are used. For the black-box attacks, we train a substitute model to generate adversarial images to attack the targeted CNN classifiers. For a fair comparison, our experimental setups closely follow those of Defense-GAN [1].

To demonstrate the generalization of our algorithm, we test our algorithms with the targeted CNN classifiers of different architectures: different number of convoluational or full-connected layers, different convolution parameters, and with/without dropout or batch normalization. For the black-box attacks, different architectures are also considered for the substitute models. When we present results, we denote the targeted model as A, B, C, D

[1]https://github.com/kabkabm/defensegan

and the substitute model as B, E.

For the defense algorithms, we compare our algorithm with Adversarial Training Goodfellow et al. (2014b); Kurakin et al. (2016), MagNet Meng & Chen (2017) and Defense-GAN Samangouei et al. (2018). All of our experiments are performed on NVIDIA Titan-Xp GPUs.

### *2.4.1 Results on White-box Attacks*

| Attack | Classifier Model | No Attack | No Defense | Defense VAE | Defense VAE-REC | Defense VAE-E2E | Defense GAN | MagNet | Adv. Tr. $\epsilon = 0.3$ |
|---|---|---|---|---|---|---|---|---|---|
| FGSM $\epsilon = 0.3$ | A | 90.85 | 9.18 | 86.9 | 89.03 | 91.02 | 87.9 | 8.9 | 79.7 |
| | B | 71.62 | 15.89 | 70.88 | 74.41 | 77.86 | 62.9 | 16.8 | 13.6 |
| | C | 90.78 | 8.68 | 85.8 | 89.72 | 90.85 | 89.6 | 11.0 | 80.4 |
| | D | 86.94 | 8.51 | 85.36 | 87.09 | 89.26 | 87.5 | 9.9 | 69.8 |
| RAND FGSM $\epsilon = 0.3$ $\alpha = 0.05$ | A | 90.85 | 7.91 | 86.42 | 88.91 | 90.57 | 88.8 | 9.6 | 44.7 |
| | B | 71.62 | 13.14 | 71.12 | 73.91 | 77.09 | 66.1 | 16.1 | 11.9 |
| | C | 90.78 | 5.48 | 86.42 | 89.38 | 90.28 | 89.3 | 11.2 | 69.9 |
| | D | 86.94 | 7.79 | 85.77 | 87.18 | 88.97 | 86.2 | 10.4 | 62.6 |
| CW $l_2$ norm | A | 90.85 | 11.67 | 81.81 | 86.99 | 88.54 | 89.6 | 6.0 | 15.7 |
| | B | 71.62 | 18.74 | 67.43 | 73.69 | 74.72 | 65.6 | 13.1 | 11.8 |
| | C | 90.78 | 7.70 | 78.64 | 87.47 | 88.69 | 89.6 | 8.4 | 10.7 |
| | D | 86.94 | 9.35 | 64.38 | 86.21 | 87.83 | 87.5 | 6.9 | 14.9 |
| Average | | 84.05 | 10.34 | 79.24 | 84.50 | **86.31** | 82.55 | 10.69 | 40.48 |

Table 2.3 Classification accuracies of different defense methods under FGSM, RAND-FGSM and CW white-box attacks on the F-MNIST image classification benchmarks. The defense accuracies of Defense-GAN, MagNet, and Adversarial Training are imported from the Defense-GAN paper Samangouei et al. (2018). Results on MNIST, CelebA and CIFAR-10 have the same pattern as above. Details can be found in section 2.5.

First, we test our algorithm on three types of white-box attacks: FGSM, RAND-FGSM and CW attacks. The targeted CNN models are trained on the original training dataset for 10 epochs until convergence. Then for each clean training image we generate 12 different adversarial images by using 3 different white-box attack algorithms, each with 4 different configurations. For FGSM and RAND-FGSM, 4 different $\epsilon = 0.25, 0.3, 0.35$ and $0.4$ are used. For the CW attack, 4 different learning rates $lr = 6, 8, 10$ and $12$ are used. We combine these adversarial images and the original clean images to form the input and output pairs to train the Defense-VAE model. We initialize the weights of VAE with the normal distribution of $\mathcal{N}(0, 0.02)$ for the convolutional layers and $\mathcal{N}(1, 0.02)$ for the batch normalization layers. We note that usually 5 epochs are required for the Defense-VAE model to converge.

Additionally, we use the reconstructed images of Defense-VAE to retrain the CNN classifier to improve the classification accuracy. Although the original CNN classifiers have already yielded very competitive performance compared with Defense-GAN, we note that retraining CNN classifiers for Defense-VAE can further strengthen the defense accuracy notably. Interestingly, the authors of Defense-GAN reported that for Defense-GAN retraining of CNN classifiers has negligible impact to the defense accuracy, while this is not true for Defense-VAE.

As discussed in Sec. 2.2.2, we can also train the whole pipeline end to end by optimizing the joint loss function 2.5 directly. This can be done through two approaches: (1) randomly initialize the VAE and CNN classifier model parameters and train the whole pipeline from scratch, and (2) pretrain VAE and CNN classifier separately and finetune the whole pipeline. Our experiments show that both approaches are almost equally effective, with the finetuning yielding slightly better results. We therefore only report the finetuning results in the following.

To demonstrate the effectiveness of this end-to-end finetuning approach, we provide one typical learning curve of the finetuning process in Figure 2.4, where the adversarial attacks are generated by FGSM with $\epsilon = 0.3$. Starting from separately pretrained VAE model and

CNN classifier (a.k.a., Defense-VAE-REC), we finetune the whole pipeline by optimizing the joint loss function 2.5. As we can see, the end-to-end finetuning boosts the defense accuracy by about 4% over the Defense-VAE model.

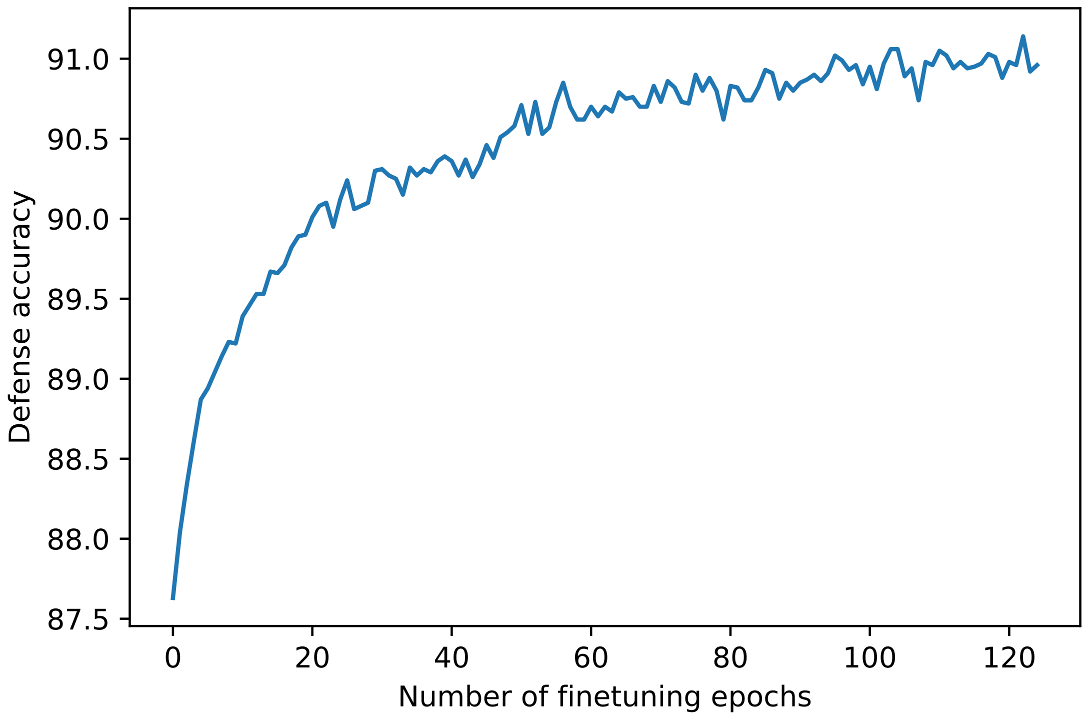


Figure 2.4 The end-to-end finetuning can boost the defense accuracy even further, and yields a stronger defense model.

Table 2.3 reports the defense accuracies of Defense-VAE on three different white-box attacks: FGSM, RAND-FGSM and CW attacks. As a comparison, we also include the results of Defense-GAN, MagNet and Adversarial Training under the same experimental setups; for those results, we import them directly from the Defense-GAN paper Samangouei et al. (2018). As we can see, Defense-VAE and Defense-GAN are very competitive to each other, and outperform all the other defense algorithms by significant margins on all four benchmarks. Defense-VAE achieves superior performance over Defense-GAN, and can recover almost all the accuracy losses due to the adversarial attacks. We also note that retraining CNN classifiers (Defense-VAE-REC) and finetuning (Defense-VAE-E2E) can further improve the defense accuracies beyond the original CNN classifiers (Defense-VAE) by a notable margin, with the finetuning yielding the strongest defense against adversarial attacks.

### *2.4.2 Robustness under Untrained Attacks*

In principle we can train Defense-VAE on all known adversarial attacks to best counter possible attacks in test. However, in reality new attacks are constantly invented; it's almost certain that after the deployment of Defense-VAE, some new adversarial attacks will emerge and Defense-VAE has never been trained on those attacks. To investigate the robustness of Defense-VAE in this circumstance, in this part of the experiments we train Defense-VAE on two attacks and test its defense capability against the third untrained attack. Again, three adversarial attacks are considered: FGSM, RAND-FGSM and CW, which gives three possible combinations that are shown in Table 2.4. As we can see, Defense-VAE is very robust for the first two attacks: FGSM and RAND-FGSM as the defense accuracies largely remain the same as it's trained on all three attacks. But for the CW attack, Defense-VAE is less robust, manifested by the significant accuracy loss compared to the Defense-VAE trained on all three attacks. Indeed, the CW attack is considered a much stronger attack and could have a very distinct attack pattern to that of FGSM and RAND-FGSM. We therefore incorporate Deepfool Moosavi-Dezfooli et al. (2016a) to the training of Defense-

VAE to counter the untrained CW attack since DeepFool and CW have very similar attack patterns. The results in parentheses show that this is indeed the case and Defense-VAE again can recover the most accuracy losses under untrained CW attack.

| Attack | Classifier | Trained on other 2 | Trained on 3 |
|---|---|---|---|
| FGSM | A | 87.34 | 89.03 |
| | B | 73.38 | 74.41 |
| | C | 88.03 | 89.72 |
| | D | 86.49 | 87.09 |
| RAND FGSM | A | 87.30 | 88.91 |
| | B | 73.59 | 73.91 |
| | C | 88.19 | 89.38 |
| | D | 86.73 | 87.18 |
| CW | A | 43.48 (85.06) | 86.99 |
| | B | 34.52 (71.64) | 73.69 |
| | C | 44.45 (85.22) | 87.47 |
| | D | 30.77 (84.69) | 86.21 |

Table 2.4 Defense accuracy of Defense-VAE when it's trained on two attacks but is used to defend another attack. The results in parentheses are the accuracies after incorporating DeepFool Moosavi-Dezfooli et al. (2016a) as additional adversarial training examples for Defense-VAE.

| $\epsilon$ | MNIST | F-MNIST |
|---|---|---|
| 0.10 | 98.95 | 86.04 |
| 0.15 | 98.67 | 86.32 |
| 0.20 | 98.58 | 86.39 |
| 0.25 | 98.44 | 86.51 |
| 0.30 | 98.29 | 86.36 |

Table 2.5 Defense accuracy of Defense-VAE with Model A under the FGSM attack with various noise level $\epsilon$ when VAE is trained only on $\epsilon = 0.3$.

Another interesting defense scenario is: what if Defense-VAE were tested on the same type of attacks but with different attack configurations? To investigate this, we train Defense-VAE on the FGSM attack with $\epsilon = 0.3$ and test its defense accuracies with different $\epsilon$. We validate this on MNIST and Fashion-MNIST, with the results shown in Table 2.5. It shows that that Defense-VAE is very robust to the untrained FGSM configurations as the defense accuracies largely remain the same under the trained attacks, e.g., with $\epsilon = 0.3$.

### *2.4.3 Results on Black-box Attacks*

| Classifier/ Substitute | No Attack | No Defense | Defense-VAE | Defense-VAE-REC | Defense-VAE-E2E | Defense-GAN | MagNet | Adv. Tr. $\epsilon = 0.3$ |
|---|---|---|---|---|---|---|---|---|
| A/B | 90.85 | 37.92 | 83.69 | 86.64 | 86.39 | 58.60 | 54.04 | 73.93 |
| A/E | 90.85 | 24.94 | 76.97 | 83.02 | 83.61 | 47.90 | 33.11 | 69.45 |
| B/B | 71.62 | 17.61 | 73.66 | 72.42 | 75.22 | 49.40 | 38.12 | 31.77 |
| B/E | 71.62 | 13.44 | 69.29 | 69.36 | 71.78 | 37.20 | 31.19 | 26.17 |
| C/B | 90.78 | 39.14 | 83.64 | 86.88 | 87.67 | 52.89 | 46.64 | 77.91 |
| C/E | 90.78 | 22.89 | 76.27 | 80.16 | 80.32 | 48.71 | 30.16 | 75.04 |
| D/B | 86.94 | 32.87 | 80.31 | 85.80 | 84.78 | 57.79 | 54.78 | 61.72 |
| D/E | 86.94 | 23.51 | 70.66 | 79.48 | 77.53 | 40.07 | 33.96 | 50.93 |
| Average | 85.05 | 26.54 | 76.81 | 80.47 | **80.91** | 49.07 | 40.25 | 58.37 |

| Classifier/ Substitute | No Attack | No Defense | Defense-VAE | Defense-VAE-REC | Defense-VAE-E2E | Defense-GAN | MagNet | Adv. Tr. $\epsilon = 0.3$ |
|---|---|---|---|---|---|---|---|---|
| A/B | 99.15 | 65.89 | 98.68 | 98.71 | 99.16 | 93.12 | 69.37 | 96.54 |
| A/E | 99.15 | 76.32 | 98.64 | 98.92 | 99.19 | 91.39 | 67.10 | 96.68 |
| B/B | 96.10 | 14.40 | 95.89 | 95.95 | 96.71 | 90.57 | 56.87 | 20.92 |
| B/E | 96.10 | 26.48 | 96.26 | 95.81 | 97.09 | 88.41 | 46.27 | 11.20 |
| C/B | 99.08 | 60.74 | 97.91 | 98.02 | 99.15 | 93.57 | 75.71 | 98.34 |
| C/E | 99.08 | 72.73 | 98.30 | 98.59 | 99.28 | 92.23 | 67.60 | 98.43 |
| D/B | 97.87 | 33.36 | 97.68 | 98.22 | 97.85 | 92.72 | 68.17 | 76.67 |
| D/E | 97.87 | 39.95 | 97.72 | 98.22 | 97.69 | 91.64 | 60.73 | 76.76 |
| Average | 98.05 | 48.73 | 97.63 | 97.81 | **98.27** | 91.71 | 63.98 | 71.92 |

Table 2.6 Classification accuracies of different defense methods under FGSM black-box attacks on different image classification benchmarks: (top) F-MNIST, and (bottom) MNIST. The defense accuracies of Defense-GAN, MagNet, and Adversarial Training are imported from the Defense-GAN paper Samangouei et al. (2018).

Next, we test the defense capability of Defense-VAE under black-box attacks on the MNIST and Fashion-MNIST datasets. We train the targeted CNN model on the training set for 10 epochs with the batch size of 100 and the learning rate of $10^{-3}$ until convergence. Then the substitute model is trained with 150 images from the test set with the labels predicted by the targeted CNN classifier.

In the black-box attacks, Defense-VAE, as a defender, has no prior knowledge of the trained substitute model. Thus, we can only train Defense-VAE on the white-box attacks.

Therefore, the same Defense-VAE model trained from the experiments of white-box attacks is used to defend the black-box attacks. [2] In this experiment, 4 targeted CNN classifiers: A, B, C, and D, and 2 substitute models: B and E are considered, and this produces 8 possible Classifier/Substitute combinations. Given a large number of experiments, in this part of experiments, only the black-box FGSM attack is considered, with the results on MNIST and Fashion-MNIST reported in Table 2.6. As a comparison, we also include the results of Defense-GAN, MagNet and Adverarial Training under the same experimental setups; again, for this set of results, we import them directly from the Defense-GAN paper Samangouei et al. (2018). As we can see, on both datasets Defense-VAE outperforms Defense-GAN and all other defense algorithms by significant margins. In particular, on Fashion-MNIST, Defense-VAE improves the accuracy over Defense-GAN by about 30%. Also, as in the white-box attack experiments, retrained CNN classifiers (Defense-VAE-REC) and finetuning (Defense-VAE-E2E) can further boost the defense accuracies over the original CNN classifiers (Defense-VAE) by a notable margin, with the end-to-end finetuning yielding the best defense accuracies among all the methods.

## 2.5 Experiments on MNIST, CelebA and CIFAR-10

We perform the white-box and black-box attacks on MNIST Lecun et al. (1998), CelebA Liu et al. (2015) and CIFAR-10 Krizhevsky (2009) datasets, with the results provided in Tables 2.8, 2.5 and 2.10, respectively. Again, we compare our method mainly with Defense-GAN Samangouei et al. (2018). Since Defense-GAN didn't provide results on CIFAR-10, we

[2]In other words, we just need to train one Defense-VAE to defend both white-box and black-box attacks.

run their code on CIFAR-10 and make sure the experimental settings for both algorithms are the same. We didn't provide the results related to the classifier model B due to its unreasonable performance on CIFAR-10 probably due to the improper parameter configuration of model B for CIFAR-10, e.g., model B has much more parameters due to the large convolutional kernel size (e.g., 8×8) and 3 input channels.

Similar to the results on F-MNIST provided in the main text, Defense-VAE outperforms Defense-GAN consistently, and retraining CNN classifiers on the reconstructions of Defense-VAE boosts the accuracy significantly. We also notice that the defense accuracies on CIFAR-10 are not as good as on CelebA. This is because CIFAR-10 is a much more challenging task than CelebA: the former is a 10-way classification task, while the latter is only a binary classification on human faces.

| Attack | Classifier Model | No Attack | No Defense | Defense VAE | Defense VAE-REC | Defense VAE-E2E | Defense GAN | MagNet | Adv. Tr. $\epsilon = 0.3$ |
|---|---|---|---|---|---|---|---|---|---|
| FGSM $\epsilon = 0.3$ | A | 99.15 | 14.65 | 98.29 | 98.98 | 99.28 | 98.8 | 19.1 | 65.1 |
| | B | 96.10 | 1.81 | 95.92 | 95.97 | 96.91 | 95.6 | 8.2 | 6.0 |
| | C | 99.08 | 29.53 | 98.41 | 98.91 | 99.24 | 98.9 | 16.3 | 78.6 |
| | D | 97.87 | 4.33 | 97.56 | 98.16 | 98.05 | 98.0 | 9.4 | 73.2 |
| RAND FGSM $\epsilon = 0.3$ $\alpha = 0.05$ | A | 99.15 | 8.65 | 98.40 | 99.08 | 99.34 | 98.8 | 17.1 | 77.4 |
| | B | 96.10 | 1.65 | 95.83 | 96.04 | 96.87 | 94.4 | 9.1 | 13.8 |
| | C | 99.08 | 5.99 | 98.33 | 98.87 | 99.35 | 98.5 | 15.1 | 90.7 |
| | D | 97.87 | 3.25 | 97.81 | 98.3 | 98.05 | 98.0 | 11.5 | 53.9 |
| CW $l_2$ norm | A | 99.15 | 8.45 | 92.69 | 95.12 | 96.95 | 98.9 | 3.8 | 7.7 |
| | B | 96.10 | 3.00 | 87.66 | 88.56 | 95.08 | 91.6 | 3.4 | 28.0 |
| | C | 99.08 | 5.53 | 94.46 | 96.05 | 96.44 | 98.9 | 2.5 | 3.1 |
| | D | 97.87 | 3.92 | 83.42 | 89.46 | 95.71 | 98.3 | 2.1 | 1.0 |
| Average | | 98.05 | 7.56 | 94.90 | 96.13 | **97.61** | 97.39 | 9.80 | 27.38 |

Table 2.7 Classification accuracies of different defense methods under FGSM, RAND-FGSM and CW white-box attacks on MNIST.

| Attack | Classifier Model | No Attack | No Defense | Defense VAE | Defense VAE-REC | Defense VAE-E2E | Defense GAN | MagNet | Adv. Tr. $\epsilon = 0.3$ |
|---|---|---|---|---|---|---|---|---|---|
| FGSM $\epsilon = 0.3$ | A | 96.55 | 3.94 | 92.40 | 94.89 | 95.10 | 92.55 | 9.85 | 12.25 |
| | B | 93.69 | 5.20 | 90.05 | 92.45 | 92.85 | 91.40 | 9.20 | 23.45 |
| | C | 95.62 | 4.45 | 92.47 | 94.46 | 95.25 | 92.55 | 10.85 | 11.30 |
| | D | 94.89 | 5.92 | 90.05 | 93.66 | 93.91 | 92.05 | 9.75 | 77.55 |
| RAND FGSM $\epsilon = 0.3$ $\alpha = 0.05$ | A | 96.55 | 4.04 | 92.11 | 94.56 | 95.34 | 92.80 | 11.05 | 7.00 |
| | B | 93.69 | 4.76 | 90.55 | 92.57 | 93.07 | 90.30 | 10.15 | 45.15 |
| | C | 95.62 | 5.12 | 91.70 | 93.76 | 94.15 | 92.00 | 10.45 | 10.55 |
| | D | 94.89 | 6.15 | 91.42 | 93.53 | 93.87 | 91.65 | 11.05 | 6.96 |
| CW $l_2$ norm | A | 96.55 | 4.94 | 93.70 | 95.07 | 95.90 | 82.10 | 9.85 | 56.90 |
| | B | 93.69 | 4.90 | 90.65 | 92.40 | 93.55 | 74.65 | 9.55 | 7.25 |
| | C | 95.62 | 8.00 | 93.28 | 94.57 | 95.92 | 79.85 | 9.85 | 26.35 |
| | D | 94.89 | 6.47 | 91.15 | 93.12 | 93.39 | 77.40 | 10.40 | 50.10 |
| Average | | 95.19 | 5.32 | 91.63 | 93.75 | **94.36** | 87.44 | 10.17 | 27.90 |

Table 2.8 Classification accuracies of different defense methods under FGSM, RAND-FGSM and CW white-box attacks on CelebA.

| Attack | Classifier Model | No Attack | No Defense | Defense VAE | Defense VAE-REC | Defense VAE-E2E | Defense GAN |
|---|---|---|---|---|---|---|---|
| FGSM $\epsilon = 0.3$ | A | 86.52 | 2.44 | 44.86 | 48.52 | 50.72 | 51.92 |
| | C | 87.62 | 5.05 | 43.92 | 47.29 | 47.39 | 47.84 |
| | D | 61.76 | 8.24 | 47.75 | 50.69 | 53.36 | 33.80 |
| RAND FGSM $\epsilon = 0.3$ | A | 86.52 | 3.71 | 39.84 | 47.80 | 50.51 | 50.36 |
| | C | 87.62 | 3.87 | 41.28 | 46.16 | 47.91 | 48.52 |
| | D | 61.76 | 7.94 | 47.88 | 50.67 | 51.18 | 26.78 |
| CW $l_2$ norm | A | 86.52 | 2.34 | 38.41 | 45.91 | 49.44 | 45.62 |
| | C | 87.62 | 7.13 | 41.21 | 46.26 | 46.19 | 43.87 |
| | D | 61.76 | 7.78 | 53.32 | 55.81 | 57.21 | 20.35 |
| Average | | 78.63 | 5.39 | 44.27 | 48.79 | **50.43** | 41.01 |

Table 2.9 Classification accuracies of different defense methods under FGSM, RAND-FGSM and CW white-box attacks on CIFAR-10. Since the Defense-GAN paper didn't provide the white-box attack results on CIFAR-10, we run their original code and provide the results in the table.

### *2.5.1 Why is Defense-VAE so effective?*

The results above demonstrated superior performance of Defense-VAE over Defense-GAN. For the black-box FGSM attack, the former even outperforms the latter by about 30%. To understand why Defense-VAE can have such a large leap, we investigate the reconstructed images by Defense-VAE and Defense-GAN in this experimental setup, i.e., the black-box

FGSM attack on Fashion-MNIST. Figure 2.5 shows some typical examples from this experiment. As can be seen, the reconstructed images from Defense-VAE often preserve the correct class information of their underlying clean images, while Defense-GAN has a harder time to identify a correct reconstruction even though it searches for the right $\boldsymbol{z}$ from $R$ random initializations and optimizes in $L$ back-propagations, with typical $R = 10$ and $L = 200$. As we discussed in Sec. 2.3, Defense-VAE identifies a proper $z$ directly by forward-propagating the input adversarial image through the VAE-encoder, and reconstructs a high quality denoised image through the VAE-decoder, and no online iterative optimization is involved.

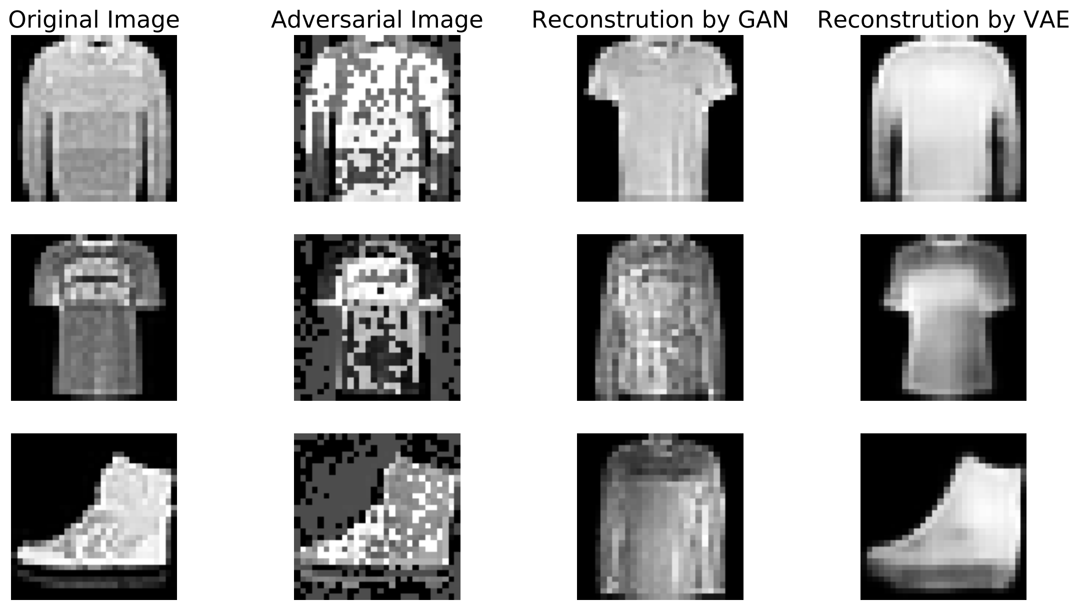


Figure 2.5 The example reconstructions by Defense-VAE and Defense-GAN from the black-box FGSM attacks on Fashion-MNIST: (first column) original images; (second column) adversarial images; (third column) reconstruction by Defense-GAN; (fourth column) reconstruction by Defense-VAE.

| Classifier/ Substitute | No Attack | No Defense | Defense-VAE | Defense-VAE-REC | Defense-VAE-E2E | Defense-GAN |
|---|---|---|---|---|---|---|
| C/E | 87.62 | 14.13 | 37.22 | 42.68 | 45.72 | 20.24 |
| D/E | 61.76 | 10.39 | 32.60 | 38.10 | 37.18 | 11.68 |
| Average | 74.69 | 12.16 | 34.91 | 40.39 | **41.45** | 16.32 |

Table 2.10 Classification accuracies of different defense methods under FGSM black-box attacks on CIFAR-10. Since the Defense-GAN paper didn't provide the black-box attack results on CIFAR-10, we run their original code and provide the results in the table.

### *2.5.2 Defense Speed*

Besides the superior defense accuracy of Defense-VAE, another advantage of Defense-VAE is its superior defense speed over Defense-GAN. As discussed above, to identify a high quality reconstruction, Defense-VAE doesn't need expensive online iterative optimizations, while Defense-GAN requires $L$ iterative back-propagations with $R$ random restarts. To have a quantitative speed comparison between Defense-VAE and Defense-GAN, we calculate their reconstruction times on 1000 adversarial images from Fashion-MNIST, with the results reported in Table 2.11, where different $R$ and $L$ configurations are considered.

As we can see, compared to the default Defense-GAN configuration, i.e., $L = 200$ and $R = 10$, Defense-VAE is about 50x faster than Defense-GAN. Moreover, as $L$ and $R$ increase, Defense-GAN generally has a slightly better defense accuracy, but the run time also increases linearly as $\mathcal{O}(L \times R)$. The constant run-time complexity of Defense-VAE makes it widely deployable in real-time security-sensitive systems.

### *2.5.3 Adversarial Detection*

The reconstruction property of Defense-VAE also entails a simple detection mechanism for the model to detect if an input image contains adversarial perturbations or not. Such a

| Defense Method | | Run Time on 1000 Images (s) |
|---|---|---|
| Defense-VAE | | 9.03 |
| Defense-GAN | L* = 200, R* = 10 | 441.81 |
| | L = 400, R = 10 | 875.48 |
| | L = 200, R = 20 | 876.10 |
| | L = 400, R = 20 | 1720.13 |

Table 2.11 Run-time comparison between Defense-VAE and Defense-GAN, where * denotes Defense-GAN recommended configuration.

functionality allows a flexibility of integrating Defense-VAE with other defense algorithms to form a defense pipeline, where Defense-VAE can be a component for adversarial detection or adversarial defense or both.

Since Defense-VAE is trained on the pair of $(\hat{\boldsymbol{x}}, \boldsymbol{x})$, where $\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\delta}$. If the input $\hat{\boldsymbol{x}}$ contains adversarial perturbations $\boldsymbol{\delta}$, the reconstructed output will be close to a clean image $\boldsymbol{x}$. Therefore, the MSE between input and output of Defense-VAE will be relatively large. On the other hand, if the input $\hat{\boldsymbol{x}}$ contains no adversarial perturbations, then the MSE will be very small and potentially close to zero. Thus, we can use the MSE as a signal to determine if an input image contains adversarial perturbations or not by simply thresholding the MSE.

To verify this idea, we generate a set of white-box FGSM attacks on Fashion-MNIST with different $\epsilon$. We then use the Defense-VAE model trained from the white-box attack experiment for adversarial detection. For each FGSM attack with a different $\epsilon$, we compute the MSE between the input and output images of Defense-VAE. By sweeping the threshold on the MSE values, we can produce a Receiver Operating Characteristic (ROC) curve for each $\epsilon$. The ROC curves for 5 different $\epsilon$ are reported in Figure 2.6. To measure the detection performance, the Area Under Curve (AUC) for each $\epsilon$ is also included. As we can see, for large magnitude FGSM attack, e.g., $\epsilon = 0.3$ or $0.25$, the AUC is almost 100%. When the attack magnitude $\epsilon$ becomes smaller, the adversarial images become harder to detect. However, Defense-VAE still achieves 97% AUC when $\epsilon = 0.15$ and an AUC of 87% when $\epsilon = 0.1$.

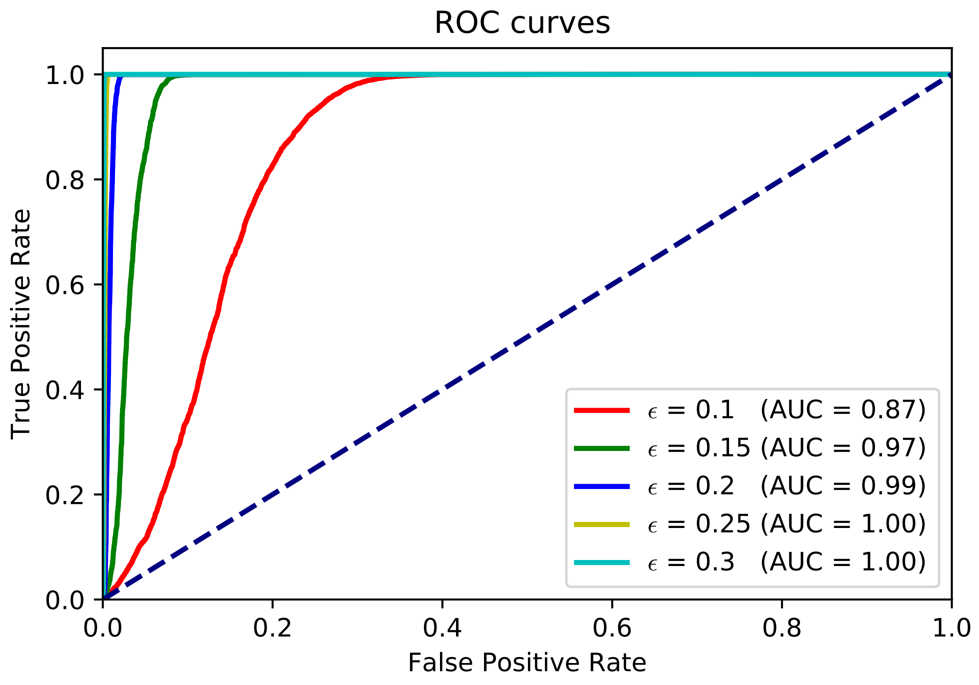


Figure 2.6 The ROC curves when using Defense-VAE to detect the adversarial attacks generated by FGSM with different $\epsilon$.

## 2.6 Conclusion

In this paper, we propose Defense-VAE, a fast and accurate defense algorithm against adversarial attacks. The algorithm is generic and can defense both white-box and black-box attacks without the need of retraining the original CNN classifier, and can further boost the defense strength by retraining or end-to-end finetuning. Compared with the state-of-the-art algorithms, in particular, Defense-GAN, our algorithm outperforms them in almost all white-box and black-box defense benchmarks. In addition, Defense-VAE is very efficient as compared to the optimization-based defense alternatives, such as Defense-GAN, as no expensive iterative online optimization is needed. Speed test shows that Defense-VAE is about 50x faster than Defense-VAE. Given the superior defense accuracy and speed, we believe Defense-VAE is widely deployable in real-time security-sensitive systems.

# CHAPTER 3
# Generative Dynamic Patch Attack

## 3.1 Introduction

Deep neural networks (DNNs) have demonstrated remarkable success in solving complex prediction tasks in a variety of fields: computer vision Deng et al. (2009), natural language processing Sutskever et al. (2014) and speech recognition Senior et al. (2012). However, recent studies show that they are particularly vulnerable to adversarial examples Goodfellow et al. (2015) in the form of small perturbations to inputs that lead DNNs to predict incorrect outputs.

Recent works Brown et al. (2017); Karmon et al. (2018); Evtimov et al. (2017); Sharif et al. (2016); Yang et al. (2020, 2019), show that perturbing part of an image with perceivable noise is another effective method to attack neural network models. Typically, attackers can craft perceivable patches to replace part of images for adversarial attack. The advantage of this perceivable patch attack is that it is more practical than the imperceptible adversarial attacks in the real world: adversaries can paste a sticker on a traffic sign to attack the autopilot system of autonomous vehicles. There are several situations where patch attack is significant concerning due to its security threats: 1) an attacker uses adversarially designed eyeglass frames Sharif et al. (2016) to fool face recognition (Fig. 3.1a), 2) an attacker pastes adversarially crafted stickers Evtimov et al. (2017) on stop signs to fool traffic sign classification (Fig. 3.1b), and 3) a universal adversarial patch Brown et al. (2017) causes targeted misclassification of any object (Fig. 3.1c).

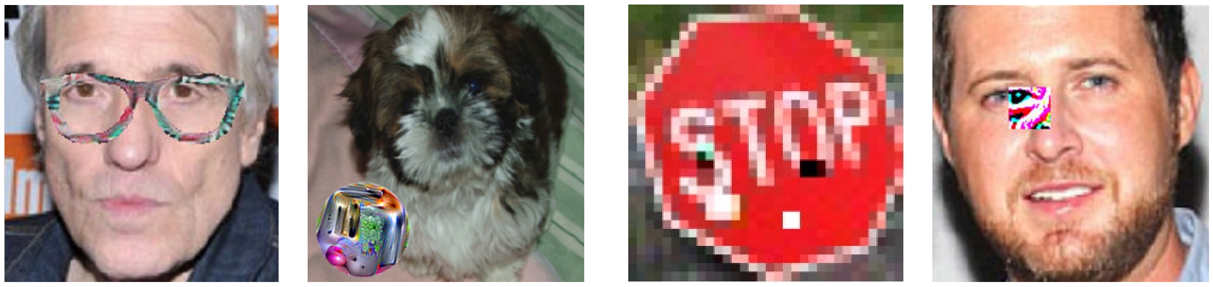


Figure 3.1 Different types of patch attacks: (a) Eyeglasses Attack Sharif et al. (2016), (b) Sticker Attack Evtimov et al. (2017), (c) Adversarial Patch Brown et al. (2017), and (d) GDPA (ours).

However, it is a significant limitation that most patch attack algorithms do not consider the problem of finding the best location in an image to inject the patch. Existing patch attack algorithms either use a fixed position as patch location Sharif et al. (2016); Evtimov et al. (2017); Yang et al. (2019) or learn patches that are universal across different locations Brown et al. (2017); Karmon et al. (2018); Yang et al. (2020). The fixed location methods show high attack success rates but are poorly performed at other locations, while the random location patches do not have competitive attack success rates compared to the fixed location methods. To address this issue, in this paper we propose a Generative Dynamic Patch Attack (GDPA), which learns image-dependent patch pattern and patch location altogether. GDPA is inspired by the idea that different images have different sets of weak pixels since DNN classifiers typically focus on different image regions when queried by different images Simonyan et al. (2013). Therefore, an image-dependent dynamic patch attack would be more effective than a fixed location or random location patch attack.

On the other hand, due to the security threats of adversarial attacks, a variety of adversarial defense algorithms have been developed recently Goodfellow et al. (2015); Lyu et al.

(2015b); Shaham et al. (2018), among which adversarial training (AT) Goodfellow et al. (2015) has been proved the most effective one for hardening neural networks against adversarial attacks. Although AT with the PGD attack Madry et al. (2018) is the most scalable and effective method for learning robust models, a recent work of Wu et al. Wu et al. (2019) shows that AT exhibits limited effectiveness against three high-profile physically realizable patch attacks: eyeglasses attack Sharif et al. (2016), sticker attack Evtimov et al. (2017) and adversarial patch Brown et al. (2017). To overcome this limitation, Wu et al. Wu et al. (2019) propose a Rectangular Occlusion Attack (ROA) for adversarial training, which yields models highly robust to patch attacks. ROA is a two-stage patch attack algorithm, which first uses a *gray pattern* to find the location in image that maximizes the cross-entropy loss via grid search, and then optimizes the patch pattern at the identified position. However, this two-stage patch attack method is suboptimal and has quite a few limitations (see a discussion in Sec. 3.2), which motivates us to propose GDPA that learns patch pattern and patch location simultaneously. Moreover, to improve the inference efficiency, GDPA employs a generator to generate patch pattern and location with one forward propagation, without expensive iterative optimizations that are usually employed by other attack algorithms, such as PGD Madry et al. (2018) and ROA Wu et al. (2019). Concretely, we make the following contributions:

- We introduce a generic patch attack method GDPA that can generate dynamic/static and visible/invisible patch attacks with a few configuration changes.

- GDPA employs a generator to generate patch pattern and patch location altogether per image, and reduces the inference time substantially (e.g., 40-50x faster).

- GDPA is end-to-end differentiable and can be readily integrated for adversarial training to defend high-profile patch attacks.

- Experiments show that GDPA has superior attack success rates over strong patch attack baselines, and the adversarially trained model with GDPA is more robust to various adversarial attacks than state-of-the-art methods.

## 3.2 Related Works

Adversarial Attack

Most adversarial attack methods focus on adding imperceptible perturbation covering the entire image Goodfellow et al. (2015); Szegedy et al. (2013). Recently, researchers have shown that perturbing a part of image with perceptible noise is another practical method to attack DNN models Brown et al. (2017); Karmon et al. (2018); Wu et al. (2019); Sharif et al. (2016); Evtimov et al. (2017); Yang et al. (2020); Liu et al. (2020); Yang et al. (2019). Sharif et al. Sharif et al. (2016) propose to add eyeglasses with a specially constructed frame texture to attack face recognition. Eykholt et al. Evtimov et al. (2017) show that adding specific rectangular solid-colored patches on traffic signs can fool traffic sign classification. LAVAN Karmon et al. (2018) learns visible and localized patches that are transferable across images and locations by training the pattern at a random location with a randomly picked image in each iteration. Recently, Wu et al. Wu et al. (2019) propose a Rectangle Occlusion Attack (ROA) to generate adversarial patches for adversarial training. ROA uses an exhaustive search (ROA-Exh) or a gradient guided search (ROA-Grad) to find the location that maximizes the cross-entropy (CE) loss and optimizes the patch pattern afterwards. Specifically, ROA-Exh exhaustively searches on images with a stride, and ROA-Grad uses the magnitude of gradient of the CE loss as the sensitivity of regions to identify the top candidate regions to accelerate the location search. However, ROA has some considerable limitations. Firstly, it employs a two-stage attack generation, which separates the process of finding the patch location and patch pattern into two steps: it first finds the position using a *gray pattern* and then optimizes the patch pattern at that position. Hence, the location identified by a *gray pattern* may not be the best patch location for the optimized pattern. Secondly, the two-stage optimization of ROA is computationally expensive and slows down the patch generation process during inference. Different from these algorithms, our GDPA trains a generator to generate the patch pattern and location altogether for each input image. Moreover, GDPA is end to end differentiable, which entails an efficient optimization and easy integration for adversarial training.

Before GDPA, several works Poursaeed et al. (2018); Baluja & Fischer (2017); Reddy Mopuri et al. (2018); Xiao et al. (2018) have proposed to train generators to generate perturbation to improve the fooling rate and inference speed. Poursaeed et al. Poursaeed et al. (2018) present a trainable network to transform input images to adversarial perturbations. Baluja and Fischer Baluja & Fischer (2017) train feed-forward neural networks in a self-supervised manner to generate adversarial examples against a target network. Different to these generator-based attack methods, our GDPA generates both patch pattern and patch location altogether, and employ an affine transform to synthesize adversarial patch examples.

## Adversarial Defense

Defending against adversarial attacks is a challenging task. Different types of defense algorithms have been proposed in the past few years Dziugaite et al. (2016); Guo et al. (2017a); Luo et al. (2015b); Gao et al. (2017); Gu & Rigazio (2014b); Lyu et al. (2015b); Akhtar et al. (2018); Guo et al. (2017a); Xu et al. (2018); Wu et al. (2019); Naseer et al. (2019); Chiang et al. (2020); Hayes (2018), among which adversarial training (AT) Madry et al. (2018) has been proved the most effective one against adversarial attacks. AT employs adversarial examples as data augmentation to train a robust model. It has been shown that this method can improve the defense accuracy effectively and sometimes can even improve the accuracy upon the model trained only on the original clean dataset Wang et al. (2020). However, a recent work of Wu et al. Wu et al. (2019) shows that robust models trained by AT exhibit limited effectiveness against high-profile patch attacks Sharif et al. (2016); Evtimov et al. (2017); Brown et al. (2017). As the first work attempting to defend patch attacks, Wu et al. Wu et al. (2019) propose DOA, which performs a standard adversarial training with Rectangle Occlusion Attack (ROA). As we discussed earlier in this section, ROA has some considerable limitations, which limit its performance on adversarial defense. Our GDPA does not suffer from those limitations of ROA, and is end-to-end differentiable and more amenable for adversarial training.

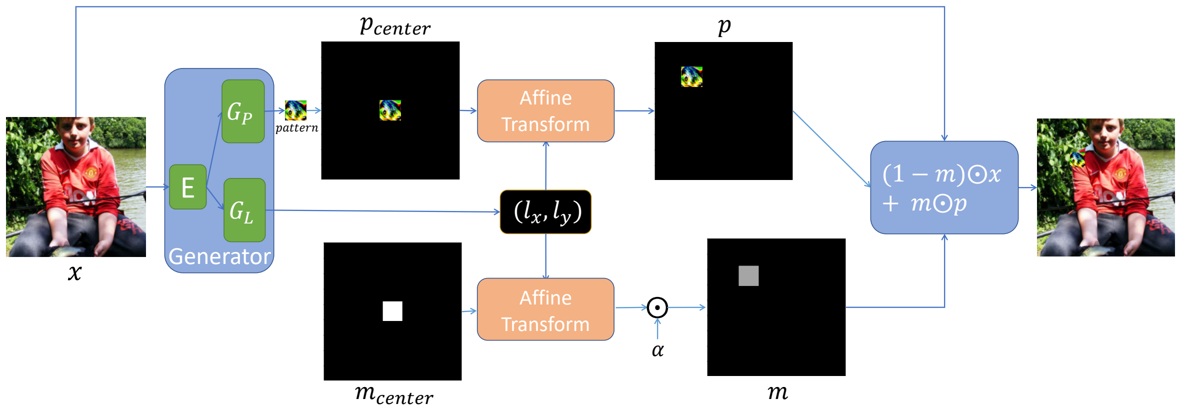


Figure 3.2 The GDPA generation pipeline. Given an image $x$, GDPA generates a patch pattern and a patch location for weighted adversarial patch injection. $\alpha \in [0, 1]$ controls the visibility of the patch attack. The pipeline is fully differentiable.

## 3.3 The GDPA Framework

GDPA is a framework that aims to conduct dynamic patch attack by generating adversarial patch pattern and patch location altogether for each input image. It has a generic formulation that can generate dynamic/static and visible/invisible patch attacks. As an overview, Figure 3.2 illustrates the GDPA generation pipeline, while Figure 3.3 demonstrates how GDPA can be utilized to train an adversarially robust model.

### *3.3.1 Problem Formulation*

We start with the definition of dynamic patch attack. Let $\mathbb{D} = \{\mathcal{X}, \mathcal{Y}\}$ denote a training dataset, where $\mathcal{X}$ is a set of images of size $w \times h$, and $\mathcal{Y}$ are their corresponding labels. Let $T : \mathcal{X} \to \mathcal{Y}$ denote a target model that we attempt to attack. Given an image $x \in \mathcal{X}$ and a target model $T$, our *dynamic patch attack* aims to find a pattern of size $w' \times h'$ and a position in image that once placed on image $x$ it can mislead the target model.

### *3.3.2 Localized Pattern Generation*

One crucial component of GDPA is the generator that generates patch pattern and patch location for a given image. Since patch pattern and patch location are coupled to a given image, we design a generator $G$ with two heads that share the same latent features extracted by an encoder. Specifically, our generator includes an encoder $G_E$ to extract the feature representation of image $x$, followed by a location decoder $G_L$ and a pattern decoder $G_P$ to generate location and pattern of the adversarial patch:

$$l_x, l_y = \tanh\left(G_L(G_E(x))/\beta\right), \tag{3.1}$$

$$\textit{pattern} = 0.5 \times \tanh(G_P(G_E(x))) + 0.5, \tag{3.2}$$

where $l_x$ and $l_y$ are the location (2D coordinates) of a patch in image $x$ with the origin at the center of image, and *pattern* is the patch pattern of size $w' \times h'$. To keep the patch location $l_x$ and $l_y$ within the boundary of image, we use a tanh function to constrain $l_x$ and $l_y$ in the range of $[-1, 1]$, where $\beta$ is a hyperparameter that controls the slope of tanh. All experiments in this paper use $\beta = 3000$, which we found to work well across a variety of architectures and datasets. Similarly, we use another tanh to impose the pattern values in the range of $[0, 1]$[1].

Specially, we use a convolutional neural network as our encoder network $G_E$, with an architecture adapted from the work of image-to-image translation Zhu et al. (2017). On top of $G_E$, we use two fully-connected networks as our decoders $G_P$ and $G_L$, respectively. Due

[1]As a preprocessing step, all images are normalized to have pixel values in the range of $[0, 1]$.

to page limit, details of the network architectures are provided in the Appendix.

### *3.3.3 Weighted Adversarial Patch Injection*

With the generated patch location and patch pattern, we then define a function to inject the patch into image $x$. Standard adversarial attacks Madry et al. (2018) employ an additive function to inject noise: $x' = x + p$, where $p$ is an imperceptible adversarial perturbation. Recently, other forms of perturbations, such as multiplicative ones $x' = x \odot m$ Yang & Ji (2020), have been explored to inject perturbations. In addition, LAVAN Karmon et al. (2018) employs $(1 - m) \odot x + m \odot p$ with a binary mask $m \in \{0, 1\}^{w' \times h'}$ to generate patch attack adversarial examples. Inspired by LAVAN, we extend this function by relaxing the binary mask to a continuous mask $m \in [0, 1]^{w' \times h'}$ for adversarial patch injection. Specifically, we employ the following weighted adversarial patch injection

$$\begin{aligned} x^{adv} &= (1 - m) \odot x + m \odot p, \\ \text{with} \quad & m \in [0, 1]^{w' \times h'} \end{aligned} \tag{3.3}$$

which is a convex combination of original image $x$ and patch pattern $p$ with the weight defined by $m$. We find this relaxed version is more flexible and easier to optimize than the one LAVAN explored. Next, we discuss how to use the generated $(l_x, l_y)$ and *pattern* to inject an adversarial patch to image $x$.

### *3.3.4 Differentiable Affine Transformation*

We employ an affine transformation in GDPA to inject adversarial patches into images. To make the whole pipeline differentiable w.r.t. $l_x$ and $l_y$, a bilinear interpolation is used to estimate the pixel values that are not on the pixel grids after transformation. By doing this, the whole pipeline is fully differentiable and the gradient can be back-propagated end-to-end to update parameters of generator $G$. Specifically, we adopt the affine transformation and image sampling method of Spatial Transformer Networks Jaderberg et al. (2015) to define a differentiable translate operator, which can translate a source image to a target image by a displacement of $(l_x, l_y)$.

We first use an affine transform to compute the pixel index relationship between source image and target image:

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}, \tag{3.4}$$

where $(x_i^t, y_i^t)$ is the pixel index of target image, and $(x_i^s, y_i^s)$ is the corresponding pixel index in source image. We set $\theta_{11} = 1, \theta_{21} = 0, \theta_{12} = 0, \theta_{22} = 1, \theta_{13} = w'/2 \cdot l_x$ and $\theta_{23} = h'/2 \cdot l_y$ for translation purpose[2]. Thus, we have $x_i^s = x_i^t + w'/2 \cdot l_x$ and $y_i^s = y_i^t + h'/2 \cdot l_y$, where

[2]Note that we can also learn $\theta_{11}, \theta_{12}, \theta_{21}, \theta_{22}$ to rotate, dilate or shear an adversarial patch to further improve GDPA's performance. For simplicity and also because we can already achieve state-of-the-art attack success rate (ASR) with translation, in this paper we only consider translation and will leave advanced affine transform to future work.

$l_x, l_y \in [-1, 1]$. Since $(x_i^s, y_i^s)$ are continuous variables, we can use a bilinear interpolation to sample the pixel values from source image:

$$v_i = \sum_{j=0}^{w-1} \sum_{k=0}^{h-1} u_{jk} \max(0, 1 - |x_i^s - j|) \max(0, 1 - |y_i^s - k|), \tag{3.5}$$

where $u_{jk}$ is the pixel value at index $(j, k)$ of the source image, and $v_i$ is the output value of pixel $i$ at index $(x_i^t, y_i^t)$ of the translated image. With the affine transform and bilinear sampler described above, we have a differentiable translate operator, which we denote as $Translate()$ in the rest part of the paper.

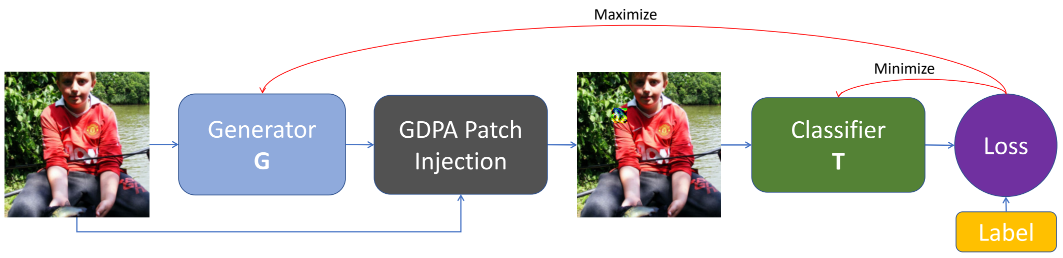


Figure 3.3 The GDPA-AT pipeline. Given an image, GDPA generates an adversarial patch to maximize the loss of classifier $T$, while classifier $T$ learns from the patch attack to minimize its loss.

### *3.3.5 Generative Dynamic Patch Attack*

Figure 3.2 illustrates the GDPA generation pipeline, which includes the three components we described above: patch pattern and location generator, differentiable affine transform, and the weighted adversarial patch injection to produce an image-dependent dynamic patch attack.

As shown in Figure 3.2, we introduce an initial mask $m_{center}$ of the same size of input image, and the center part of the mask has value 1 and rest of 0. Then we use the affine transform $Translate()$ to translate $m_{center}$ by a displacement of $(l_x, l_y)$:

$$m = \alpha \cdot Translate(m_{center}, l_x, l_y), \tag{3.6}$$

where $\alpha \in [0, 1]$ is a hyperparameter that controls the visibility of adversarial patches. When $\alpha = 1$, the patch would be completely visible and replace the original image pixel values; otherwise, the visibility of the adversarial patch will be lower. In practice, we can use a small value of $\alpha$ to generate human imperceptible adversarial patches.

Similarly, we can generate a translated patch pattern. As shown in Figure 3.2, once *pattern* is generated, we zero pad it to create a pattern $p_{center}$ of the same size of input image with *pattern* at the center. We then translate $p_{center}$ by $(l_x, l_y)$ via the affine transform:

$$p = Translate(p_{center}, l_x, l_y). \tag{3.7}$$

Finally, we can generate a GDPA adversarial example for image $x$ by

$$x^{adv} = (1 - m) \odot x + m \odot p. \tag{3.8}$$

As we can see, all the components in Figure 3.2 are differentiable. Therefore, the whole GPDA generation pipeline is fully differentiable and can be optimized efficiently with gradient-based methods.

To train the generator for non-targeted patch attack, we can optimize $G$ to maximize the cross-entropy loss of target model $T$ with class labels

$$\arg\min_{G} -\mathcal{L}_{\mathtt{CE}}(T, x^{adv}, y). \tag{3.9}$$

We can also launch a targeted patch attack to fool the target model $T$ to misclassify an input $x$ as target class $y_{target}$ by

$$\arg\min_{G} \mathcal{L}_{\mathtt{CE}}(T, x^{adv}, y_{target}) \tag{3.10}$$

Details of the GDPA training algorithm are summarized in Algorithm 1.

**Algorithm 1:** GDPA generator training

```
Input: training set $\mathbb{D}$; target model $T$; visibility $\alpha$
Output: generator $G$
initialize generator $G$;
for number of training epochs do
    for each $(x, y) \in \mathbb{D}$ do
        $l_x, l_y = G_L((G_E(x))$;
        $pattern = G_P(G_E(x))$;
        $m = \alpha \cdot Translate(m_{center}, l_x, l_y)$;
        $p = Translate(p_{center}, l_x, l_y)$;
        $x^{adv} = (1 - m) \odot x + m \odot p$;
        if targeted attack then
            $loss = \mathcal{L}_{\mathtt{CE}}(T, x^{adv}, y_{target})$;
        else
            $loss = -\mathcal{L}_{\mathtt{CE}}(T, x^{adv}, y)$;
        end
        $\theta_G = \theta_G - lr * \partial loss / \partial \theta_G$
    end
end
```

### *3.3.6 Adversarial Training with GDPA*

Adversarial training with the PGD attack exhibits limited effectiveness against high-profile patch attacks Wu et al. (2019). In this section, we discuss how to utilize GDPA for adversarial training to improve model robustness against high-profile patch attacks.

Figure 3.3 illustrates the GDPA adversarial training (GDPA-AT) pipeline to train a robust model against patch attacks. Similar to Generative Adversarial Networks Goodfellow et al. (2014a), GDPA-AT trains generator $G$ and target classifier $T$ iteratively to optimize the following minimax objective:

$$\min_{T} \max_{G} \mathbb{E}_{(x,y)\sim\mathbb{D}}[\mathcal{L}_{\texttt{CE}}(T, x^{adv}, y)], \tag{3.11}$$

where the inner maximization step optimizes generator $G$ to maximize the classification loss of $T$, while the outer minimization step optimizes target classifier $T$ to minimize the classification loss. Unlike the traditional adversarial training, in which the inner maximization step usually optimizes an adversarial example $x^{adv}$ directly, our GDPA-AT optimizes a generator $G$ to generate patch attack with one forward propagation. As the iterative training proceeds, the generator $G$ searches for the weakest image region to attack classifier $T$ at each iteration, while $T$ learns from the current patch attacks and becomes more resilient to these attacks over time. Details of our GDPA-AT algorithm are described in Algorithm 2.

**Algorithm 2:** GDPA-AT

```
Input: training set $\mathbb{D}$
Output: target classifier $T$; generator $G$
initialize classifier $T$ and generator $G$;
for number of training epochs do
    for each $(x, y) \in \mathbb{D}$ do
        $x^{adv} = GDPA(G, x)$ ;
        $loss = -\mathcal{L}_{\texttt{CE}}(T, x^{adv}, y)$ ;
        $\theta_G = \theta_G - lr_G * \partial loss / \partial \theta_G$ ;
        $x^{adv} = GDPA(G, x)$ ;
        $loss = \mathcal{L}_{\texttt{CE}}(T, x^{adv}, y)$ ;
        $\theta_T = \theta_T - lr_T * \partial loss / \partial \theta_T$
    end
end
```

## 3.4 Experimental Results

We now validate GDPA on benchmark datasets for adversarial patch attack and adversarial defense. Specifically, we evaluate the performance of GDPA on patch attack in Section 3.4.1 and GDPA-AT on improving model robustness in Section 3.4.2. To evaluate the inference efficiency, we also compare the run-times of GPDA and state-of-the-art attack algorithms in Section 3.4.8. All our experiments are performed with PyTorch on Nvidia RTX GPUs. Our source code is provided as a part of supplementary materials.

Experimental Setup

We evaluate GDPA and GDPA-AT on three benchmark datasets: VGGFace Sharif et al. (2016), Traffic Sign Evtimov et al. (2017) and ImageNet Deng et al. (2009). To evaluate GDPA's attack performance, we compare GDPA with LAVAN Karmon et al. (2018) and ROA Wu et al. (2019), two state-of-the-art patch attack algorithms that generate patches based on iterative optimizations. Following their experimental settings, we run LAVAN and ROA for 50 optimization iterations with a learning rate of 4. For adversarial defense experiments, we compare GDPA-AT with DOA Wu et al. (2019) and PGD-AT Madry et al. (2018). The former is a state-of-the-art defense algorithm for patch attacks, while the latter is a well-established defense algorithm for adversarial attacks. We evaluate the robustness of the models under eyeglasses attack Sharif et al. (2016), sticker attack Evtimov et al. (2017) and standard PGD attack Madry et al. (2018). Following the settings in DOA Wu et al. (2019), we use $70 \times 70$ patches with stride 5 for VGGFace and $7 \times 7$ patches with stride 2 for Traffic Sign to generate ROA attacks. We set $\epsilon = 16$ for PGD-AT since this yields the best results of PGD-AT. We use attack success rate (ASR) Dong et al. (2020) as the metric to evaluate the effectiveness of an attack, and use classification accuracy to evaluate the robustness of a model when under adversarial attacks. Details of benchmark datasets, high-profile patch attacks, network architectures and training procedures can be found in the Appendix.

#### *3.4.0.1 Experimental Details*

We first describe the three benchmark datasets and target models used in our experiments. These datasets are used to train our GDPA generator, robust models with adversarial training, and evaluate the performance of patch attacks.

#### *3.4.0.2 VGGFace*

Dataset

The VGGFace dataset Parkhi et al. (2015) is a benchmark for face recognition, containing 2,622 subjects and 2.6 million images in total. Same with DOA Wu et al. (2019), we choose 10 subjects and sample face images only containing those individuals. We process the data to the size of $224 \times 224$ by standard crop-and-resize, and perform class-balanced split to generate training, validation, and test datasets with ratio 7:2:1. As a result, we obtain 3178, 922 and 470 images for training, validation and test, respectively. The training set is used to train the target model, the GDPA generator and robust models with adversarial training. Likewise, the test set is used to evaluate the target model, the performance of patch attack and adversarial defense.

Target Model

We use the VGGFace CNN model Parkhi et al. (2015) as the target classifier in our experiments. We use standard transfer learning on our processed dataset, keeping the convolutional layers in the VGGFace CNN model, but adjusting the number of output neurons of the last fully connected layer to 10. In order to use the pre-trained weights from the convolutional layers of VGGFace CNN model, we convert the images from RGB to BGR and subtract the mean value $[129.2, 104.8, 93.6]$. We set the batch size to 64 and use the Adam Optimizer with an initial learning rate of $10^{-4}$. We drop the learning rate by 0.1 every 10 epochs. For hyperparameter tuning and model selection, we track the accuracy on validation set to avoid overfitting. We train the model on training set for 30 epochs and obtain an accuracy of 98.94% on test data.

#### *3.4.0.3 Traffic Sign*

Dataset

To have a fair comparison with DOA Wu et al. (2019), we pick the same 16 traffic signs from the dataset LISA Mogelmose et al. (2012) with 3,509 training and 1,148 validation images. Following the prior works Evtimov et al. (2017); Wu et al. (2019), we further sample 40 stop signs from the validation set as the test data to evaluate performance of the stop sign classification. Similarly, all the data are processed by standard crop-and-resize to $32 \times 32$ pixels. Same with VGGFace, we use the training set to train the target model, the GDPA generator and robust models with adversarial training. We use the test set to evaluate the performance of the target model, patch attack and adversarial defense.

Target Model

We use the LISA-CNN Evtimov et al. (2017) as the target model, which contains three convolutional layers and one fully-connected layer. We use the Adam Optimizer with initial learning rate 0.1 and drop the learning rate by 0.1 every 10 epochs. We set the batch size to 128. After 30 epochs, we achieve an accuracy of 98.69% on the validation set, and 100% accuracy on the test data.

#### *3.4.0.4 ImageNet*

Dataset

ImageNet Deng et al. (2009) is a well-known large scale object recognition benchmark. To develop the training and validation sets to train and evaluate the GDPA generator and robust models with adversarial training, we follow Moosavi-Dezfooli et al. Moosavi-Dezfooli et al. (2017) to select a subset of $10,000$ images from ImageNet training set (randomly choose ten images for each class) as our training set, and use the whole ImageNet validation set ($50,000$ images) as our validation set.

Target Model

Following Poursaeed at el. Poursaeed et al. (2018), we use a pre-trained VGG19 model Simonyan & Zisserman (2014) from PyTorch library as the target model. This model achieves an accuracy of 72.4% on the validation set.

#### *3.4.0.5 Patch Attacks*

Eyeglasses Attack

This is an effective physically realizable patch attack developed by Sharif et al. Sharif et al. (2016). It first initializes the eyeglass frames with 5 different colors, and chooses the color with the highest cross-entropy loss as starting color. For each update step, it divides the gradient value by its maximum and multiplies the results with the learning rate. Then it only keeps the gradient value in the eyeglass frame area. Finally, it clips and rounds the pixel values to keep them in the valid range. We evaluate the eyeglasses attack on the test set of VGGFace.

Sticker Attack

Proposed by Evtimov et al. Evtimov et al. (2017), this is another physically realizable patch attack. It initializes the stickers on the stop signs with random noise at fixed locations. For each update step, it uses the Adam optimizer with the learning rate 0.1 (and default parameters) to maximize the classification loss of the target model. Just as the other patch attacks, adversarial perturbations are restricted to the mask area; in our experiments, we use the same collection of small rectangles as in Evtimov et al. (2017). We evaluate the sticker attack on the test set of Traffic Sign.

#### *3.4.0.6 GDPA Network Architecture and Training Details*

Network Architecture

For VGGFace and ImageNet, both having images of size $224 \times 224$, we adopt the encoder network structure $G_E$ from the work of image-to-image translation Zhu et al. (2017). For the Traffic Sign dataset, which has images of size $32 \times 32$, we adopt a CNN of 3 convolutional layers with kernel size 4 and stride 2 as the encoder network $G_E$. We then use a neural network of one fully-connected layer with output size $3 \times w' \times h'$ as the pattern decoder $G_P$, and a neural network of one fully-connected layer with output size 2 as the location decoder $G_L$.

GDPA Training Details

Following Algorithm 1 in the main text, we train the GDPA generator $G$ by using the Adam optimizer with an initial learning rate of 0.1 for VGGFace and ImageNet, and 0.01 for Traffic Sign. We drop the learning rate by 0.2 every 10 epochs and train the generator for 30 epochs. We set the batch size to 32 and $\beta$ to 3000, which we find works well across various architectures and datasets in our experiments.

GDPA-AT Training Details

Following Algorithm 2 in the main text, we train the GDPA generator $G$ and target model $T$ iteratively. We initialize the generator with a pre-trained GDPA generator and the target model with a cross-entropy trained model. We set the $w'$ and $h'$ to 70 for VGGFace and 7 for Traffic Sign during the adversarial training. We use the Adam optimizer to train the generator and the target model, with a learning rate of 0.0001 for both VGGFace and Traffic Sign, and drop the learning rate by 0.2 every 50 epochs. We use batch size 32 and train for 1000 epochs for VGGFace and 5000 epochs for Traffic Sign.

#### ***3.4.1 Dynamic Patch Attack***

We first evaluate the performance of GDPA on non-targeted and targeted patch attacks and compare it with the state-of-the-arts: LAVAN Karmon et al. (2018) and ROA Wu et al. (2019). We provide results of two versions of ROA: ROA-Exh and ROA-Grad, where the former exhaustively searches for a patch location in images with a fixed stride, and the latter uses the magnitude of gradient as the sensitivity of regions to identify top regions to

| Dataset | Algorithm | Percentage of Attacked Pixels | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Non-Targeted Attack | | | | Targeted Attack | | | |
| | | 1% | 2% | 5% | 10% | 1% | 2% | 5% | 10% |
| Traffic Sign | LAVAN Karmon et al. (2018) | 33.4 | 58.7 | 85.1 | 93.9 | 32.2 | 48.1 | **80.9** | 89.9 |
| | ROA-Grad Wu et al. (2019) | 36.2 | 61.8 | 87.3 | 93.6 | 29.8 | 44.6 | 74.5 | 90.5 |
| | ROA-Exh Wu et al. (2019) | 37.1 | 63.0 | 89.4 | 93.8 | 31.3 | 45.9 | 76.2 | 91.7 |
| | **GDPA** | **39.6** | **64.1** | **91.3** | **94.3** | **33.9** | **50.4** | 77.5 | **92.8** |
| VGGFace | LAVAN Karmon et al. (2018) | 31.9 | 42.7 | 56.3 | 92.0 | 37.8 | 57.9 | 67.2 | 94.6 |
| | ROA-Grad Wu et al. (2019) | 37.5 | 62.3 | 84.2 | **99.6** | 46.3 | 75.6 | 89.0 | 99.2 |
| | ROA-Exh Wu et al. (2019) | 38.3 | 64.5 | 86.0 | **99.6** | 48.2 | 76.7 | 91.1 | 99.3 |
| | **GDPA** | **46.3** | **76.4** | **88.4** | 99.5 | **50.5** | **83.4** | **95.5** | **99.8** |
| ImageNet | LAVAN Karmon et al. (2018) | 89.2 | 92.8 | 97.8 | **99.9** | 86.3 | 93.8 | **99.7** | 99.8 |
| | ROA-Grad Wu et al. (2019) | 93.5 | 94.6 | 98.7 | 99.7 | 79.6 | 88.3 | 97.5 | 99.8 |
| | ROA-Exh Wu et al. (2019) | 94.8 | 95.3 | 99.2 | 99.7 | 81.1 | 89.6 | 98.4 | 99.8 |
| | **GDPA** | **96.3** | **96.9** | **99.7** | 99.8 | **89.3** | **94.4** | 99.6 | **99.9** |

Table 3.1 The ASRs of different patch attack algorithms on datasets Traffic Sign, VGGFace and ImageNet. Both non-targeted attack and targeted attack are considered. The performances are evaluated with patches of different sizes.

accelerate the location search. We evaluate the effectiveness of the attack algorithms when perturbing different percentages of pixels. To interpret the results, we also visualize the perturbed images generated by GDPA.

### Non-targeted Patch Attack

Table 3.1 (left part) reports the ASRs of GDPA and the other competing algorithms for non-targeted patch attacks. The ASRs of an attack algorithm are evaluated on a model trained with cross-entropy (CE) loss when attacked with patches of different sizes (1%, 2%, 5% or 10% of pixels). Specifically, We use square patches of width 3, 5, 7, 10 for Traffic Sign and 23, 32, 50, 71 for VGGFace and ImageNet. As expected, the larger patch size is, the higher ASR is achieved for all patch attack algorithms. In most of the cases, GDPA achieves higher ASRs than the competing algorithms.

Figure 3.4 visualizes the perturbed images of different patch sizes generated by GDPA for

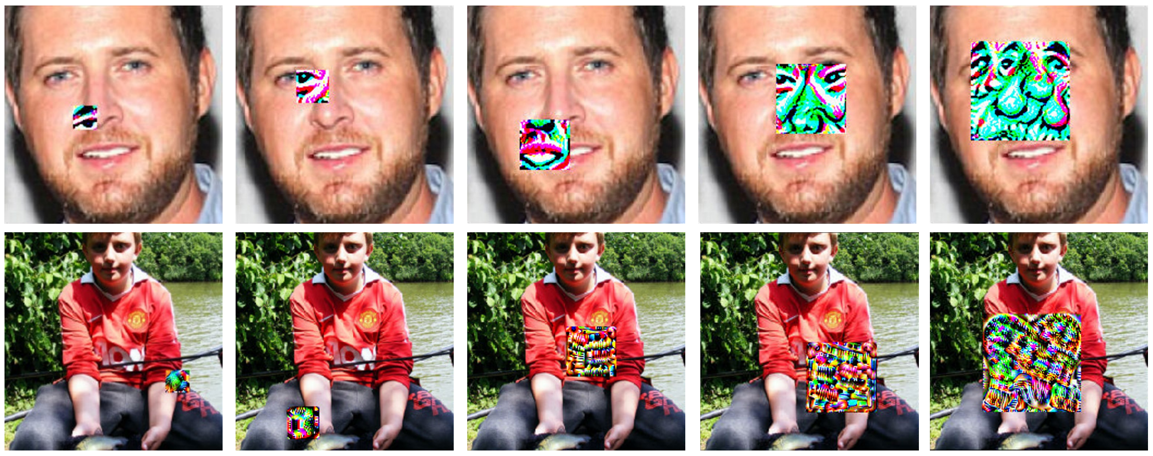


Figure 3.4 Non-targeted Attack: Perturbed images of VGGFace and ImageNet generated by GDPA with different patch sizes. The last column of targeted attack are example images of target classes.

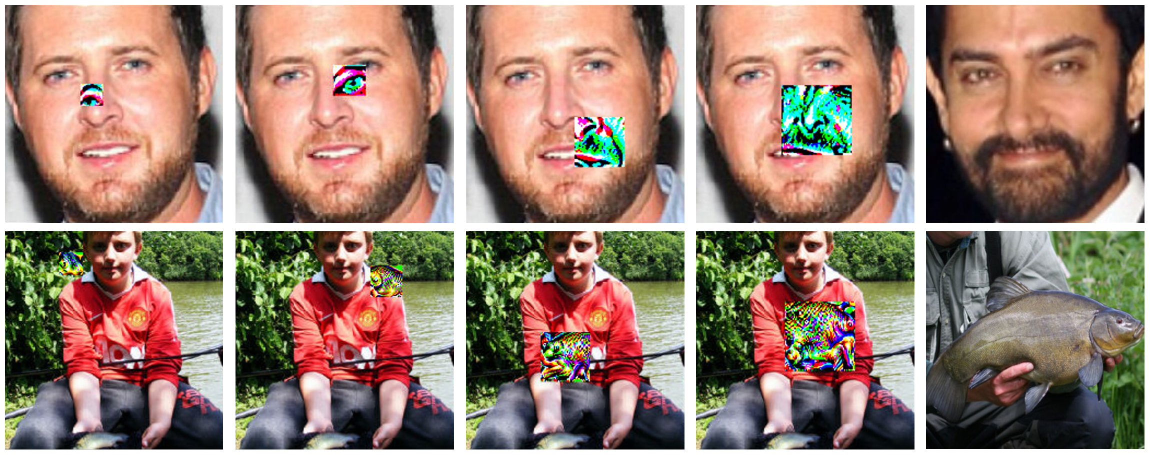


Figure 3.5 Targeted Attack:Perturbed images of VGGFace and ImageNet generated by GDPA with different patch sizes. The last column of targeted attack are example images of target classes.

non-targeted attack on VGGFace and ImageNet. As we can see, the patches generated on VGGFace (top row) demonstrates clear semantic meanings, resembling human eyes, nose or mouth. Moreover, the positions chosen by GDPA on face images are in a close proximity of the original face features. On the other hand, the patches generated on ImageNet (bottom row) do not demonstrate a strong semantic meaning that is comprehensible by human. These patch patterns look similar to the adversarial patches found by Brown et al. (2017). We conjecture that this may be because ImageNet is a much more complex dataset than VGGFace with 1,000 object categories; to attack an ImageNet model, more complex patterns (e.g., superimposition of multiple object categories) might be required, and thus are harder for human to comprehend. However, this hypothesis may be worthy of further investigation, which we will leave to our future work.

### Targeted Patch Attack

Table 3.1 (right part) reports the ASRs of different algorithms for targeted patch attack with different patch sizes. For each of the three datasets, we choose the first class as the target class, i.e., “AddedLine”, “Aamir Khan” and “tench, Tinca tinca”, respectively. Similar to the results of untargeted attack, patches of larger sizes have higher ASRs than smaller ones. GDPA achieves higher ASRs than other competing methods in most of cases.

Figure 3.5 visualizes the perturbed images of different patch sizes generated by GDPA for targeted attack on VGGFace and ImageNet. It can be observed that the patches generated for both datasets have clear semantic meanings. For example, the patches generated on a VGGFace image look like eyes, mouth or beard of the target identity, while the patches generated for an ImageNet image look like a fish, which is the similar to the target label “tench”.

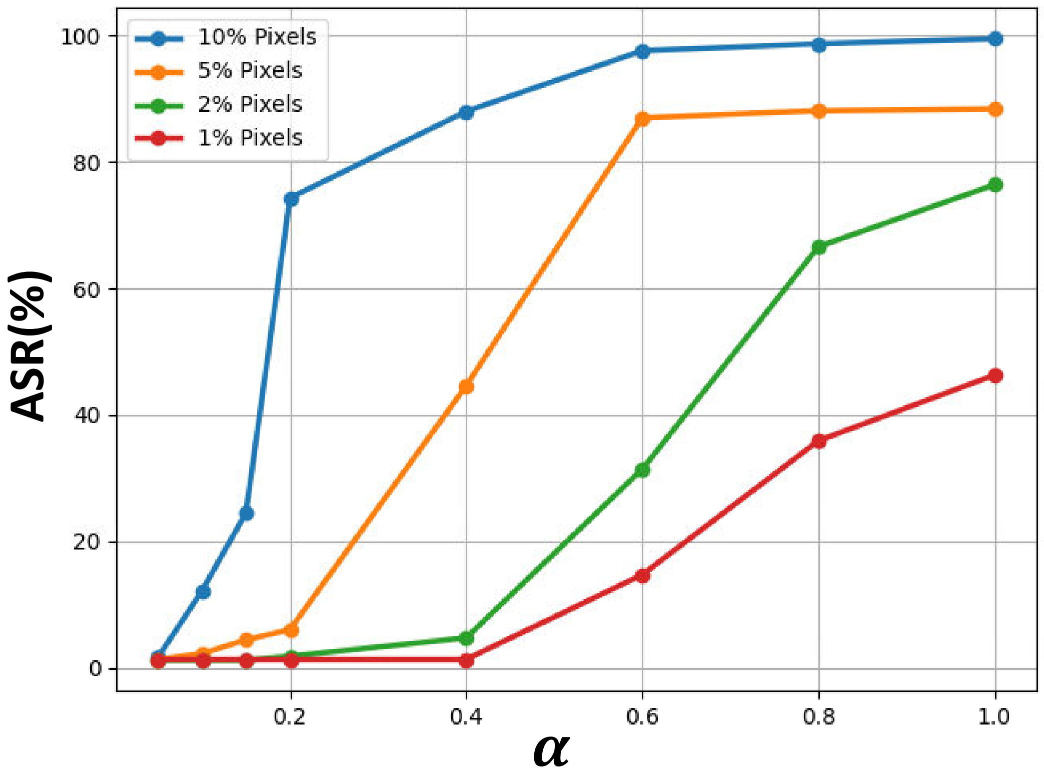


Figure 3.6 The impact of $\alpha$ to the ASR of GDPA on VGGFace.

Visibility $\alpha$ vs. ASR

We further investigate the impact of visibility parameter $\alpha$ of Eq. 3.6 to GDPA's ASR. The results on VGGFace are shown in Figure 3.6, where we consider different patch sizes. As expected, when $\alpha$ increases, the attack strength of GDPA gets stronger for all different patch sizes. Notably, when the patch size is 5% or 10% of pixels, GDPA can reach almost the highest ASRs when $\alpha \geq 0.6$, indicating that when patches are sufficiently large, the attack can be more invisible to attack a model successfully. Example perturbed images generated by GDPA with different $\alpha$ are provided.

### *3.4.2 Dynamic Patch Adversarial Training*

Next we validate the robustness of models trained by GDPA-AT against various adversarial attacks. Specifically, we report the results of GDPA-AT trained models against patch attacks and conventional adversarial attacks, and compare them with state-of-the-art defense methods.

#### GDPA-AT against Patch Attacks

Table 3.2 reports the accuracies of robust models trained by different defense algorithms against two types of patch attacks: 1) eyeglasses attack on VGGFace, and 2) sticker attack on Traffic Sign. As can be seen, PGD-AT, a well-established defense method for conventional adversarial attacks, is not robust to either eyeglasses attack or traffic sign attack, which is consistent with the results reported in Wu et al. (2019). While both DOA and GDPA-AT improve the robustness over PGD-AT significantly, GDPA-AT achieves substantially higher accuracies than the two variants of DOA.

Figure 3.7 and Figure 3.8 shows example results when using eyeglasses attack to evade a standard CE-trained model (3.7) and the GDPA-AT trained model (3.8). As we can see, the eyeglasses attack fails to attack the GDPA-AT trained model because it is not able to generate effective adversarial patterns on the eyeglass frames in 5 out of 6 cases, while being very successful on standard CE-trained model.

| | Attack Iterations | | | |
|---|---|---|---|---|
| | 0 | 100 | 200 | 300 |
| CE Training | 98.9 | 0.2 | 0.1 | 0.0 |
| PGD-AT Madry et al. (2018) | 97.3 | 37.7 | 36.9 | 36.6 |
| DOA-Grad Wu et al. (2019) | 99.2 | 85.3 | 83.7 | 81.9 |
| DOA-Exh Wu et al. (2019) | 99.0 | 89.8 | 87.8 | 85.9 |
| **GDPA-AT** | **99.6** | **96.5** | **94.7** | **94.9** |
| | 0 | 10 | 100 | 1000 |
| CE Training | 98.7 | 42.9 | 32.5 | 24.3 |
| PGD-AT Madry et al. (2018) | 97.5 | 57.6 | 45.1 | 42.5 |
| DOA-Grad Wu et al. (2019) | 95.6 | 85.2 | 84.8 | 82.8 |
| DOA-Exh Wu et al. (2019) | 92.9 | 92.2 | 91.8 | 90.8 |
| **GDPA-AT** | **98.5** | **96.2** | **95.8** | **94.6** |

Table 3.2 The accuracies of different robust models under eyeglasses attack, and sticker attack.

GDPA-AT against Adversarial Attack

Alternatively, we also evaluate the robustness of models under conventional adversarial attacks, such as the PGD attack Madry et al. (2018). The results are reported in Table 3.3, where different PGD attack strengths $\epsilon$ have been considered. It can be observed that GDPA-AT achieves significantly higher robustness than DOA against the PGD attack. More interestingly, the accuacies that GDPA-AT achieve are almost on par with PGD-AT even though GDPA is a patch attack algorithm. We believe this is because during the adversarial training process, GPDA generates the adversarial patches to attack the classifier iteratively; even though each patch attack is localized, the combination of all patch attacks generated during the iterative process resembles a whole image attack that PGD usually produces. For this reason, the model trained by GDPA-AT can defend conventional adversarial attacks.

These results demonstrate that GDPA-AT is a generic defense algorithm that can defend

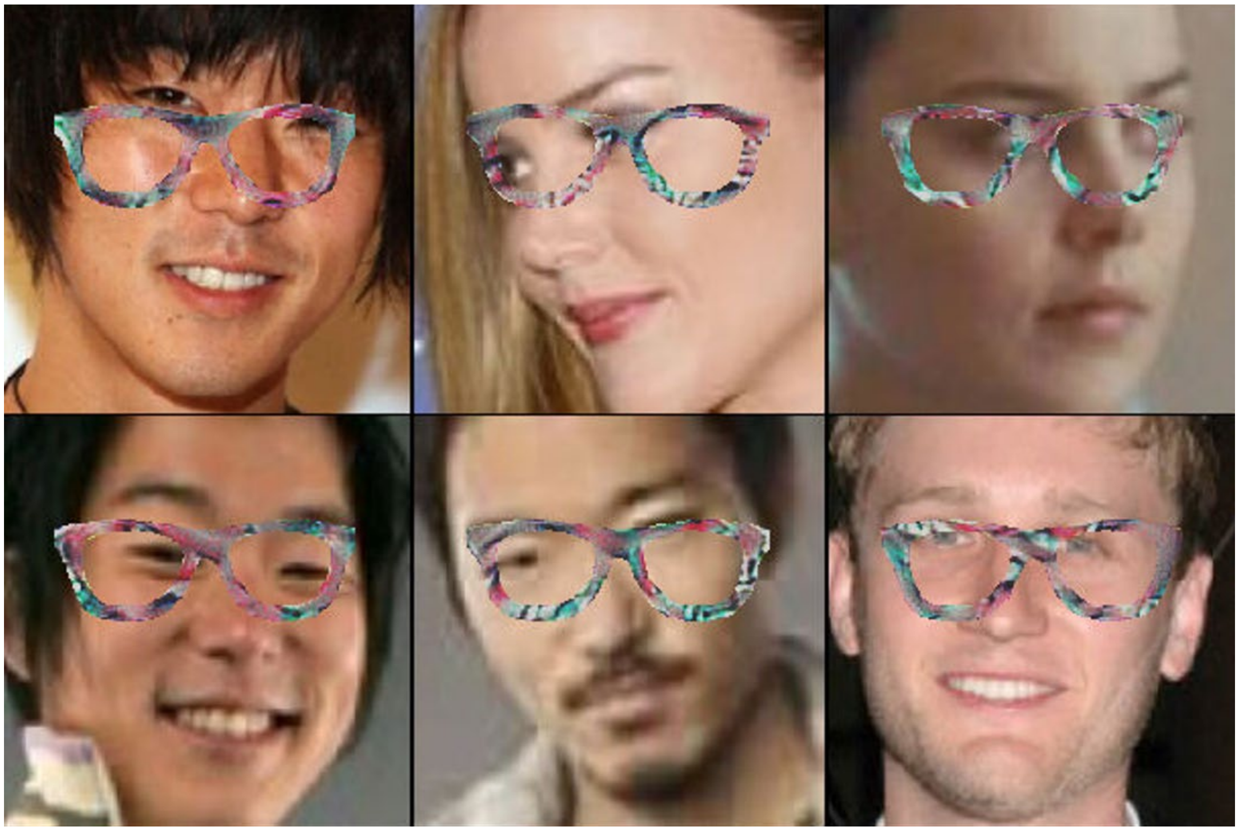


Figure 3.7 Perturbed images generated by eyeglasses attack on standard CE-trained model

both patch attacks and conventional adversarial attacks, while PGD-AT and DOA fail on one of them.

### *3.4.3 Ablation Study*

#### *3.4.3.1 Generate pattern vs p*

Instead of generating *pattern* from the GDPA generator, we can generate $p$ directly by adjusting the output size of pattern decoder $G_P$ to $3 \times w \times h$. Directly generating $p$ can simplify the pipeline of GDPA as we do not need to translate *pattern* to generate $p$ in two

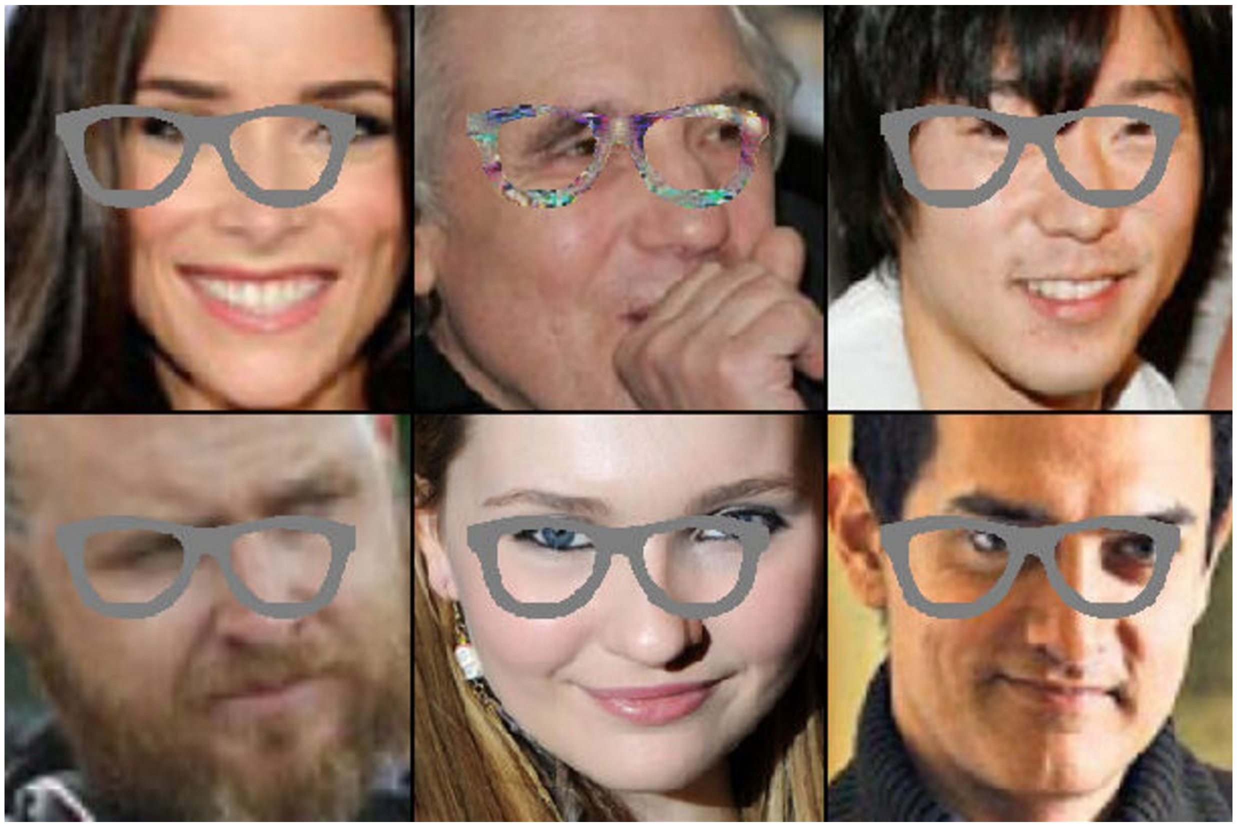


Figure 3.8 Perturbed images generated by eyeglasses attack on GDPA-AT trained model.

steps. Thus, it's worth investigating which design choice works better. Table 3.4 shows the results comparing these two design choices. As we can see, generating *pattern* achieves significantly higher ASRs than generating $p$ directly. We conjecture that this is because $p$ has a larger space to optimize than *pattern*, and thus is more difficult to optimize. Hence, in our GDPA pipeline we generate *pattern* first and then translate *pattern* to generate $p$.

### *3.4.3.2 Visibility $\alpha$ vs. ASR*

In Section 4.1, we investigate the impact of visibility parameter $\alpha$ of Eq. 6 on GDPA's ASR. Figure 3.9 visualizes some example perturbed images generated by GDPA with different $\alpha$'s

| | Attack Strength ($\epsilon$) | | | | |
|---|---|---|---|---|---|
| | 0 | 2 | 4 | 8 | 16 |
| CE training | 98.9 | 44.4 | 1.7 | 0 | 0 |
| PGD-AT Madry et al. (2018) | 97.3 | 96.9 | 96.6 | 96.1 | 95.8 |
| DOA-Grad Wu et al. (2019) | 97.5 | 33.4 | 0.4 | 0 | 0 |
| DOA-Exh Wu et al. (2019) | 98.5 | 35.7 | 0.4 | 0 | 0 |
| **GDPA-AT** | **98.9** | **95.1** | **94.9** | **94.6** | **94.5** |
| | 0 | 2 | 4 | 8 | 16 |
| CE training | 98.7 | 89.5 | 61.6 | 24.6 | 5.1 |
| PGD-AT Madry et al. (2018) | 97.5 | 95.8 | 94.6 | 92.9 | 91.0 |
| DOA-Grad Wu et al. (2019) | 95.6 | 91.2 | 79.5 | 46.9 | 6.7 |
| DOA-Exh Wu et al. (2019) | 92.9 | 89.5 | 77.1 | 42.8 | 5.8 |
| **GDPA-AT** | **98.5** | **94.7** | **93.5** | **92.2** | **90.3** |

Table 3.3 The accuracies of different robust models on VGGFace, and Traffic Sign when under the PGD attack.

| | Generate *pattern* | Generate $p$ |
|---|---|---|
| Traffic Sign | **87.9%** | 69.7% |
| VGGFace | **46.3%** | 24.9% |
| ImageNet | **96.3%** | 63.8% |

Table 3.4 ASRs of GDPA when generating *pattern* vs. $p$.

and patch sizes. As we can see, by using different $\alpha$'s, we can control the visibility of GDPA attack.

#### *3.4.3.3 Effect of $\beta$*

The $\beta$ in Eq. 1 controls the slope of tanh that constrains $l_x$ and $l_y$ in the range of $[-1, 1]$. It is critical to find an appropriate value of $\beta$ to train the GDPA generator. Intuitively, a too large or too small $\beta$ value can cause different training difficulties. If $\beta$'s value is too small, the tanh activation function saturates quickly and pushes $l_x$ and $l_y$ to the saturated value of -1 or 1, which corresponds to corners of an image. On the other hand, if $\beta$ is too large,

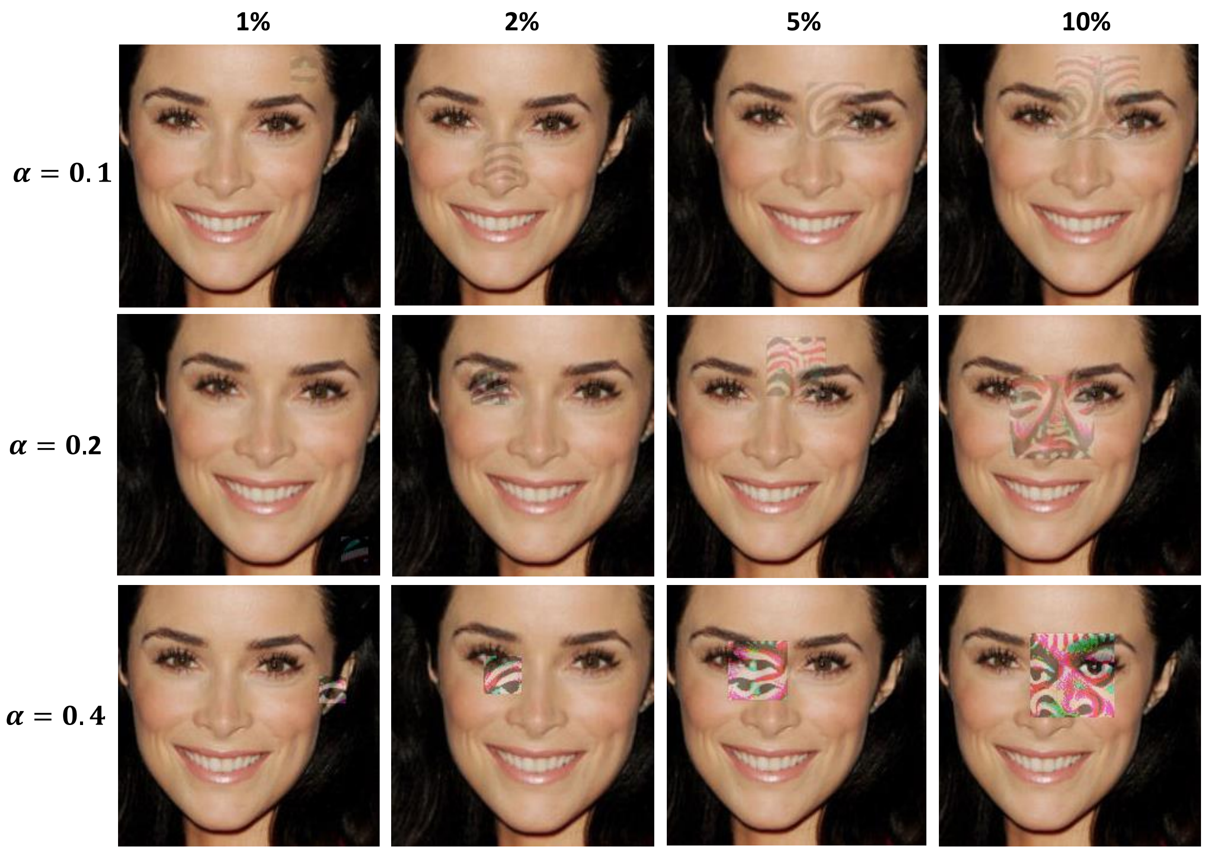


Figure 3.9 Perturbed images generated by GDPA with different $\alpha$'s and patch sizes (1%, 2%, 5% or 10% pixels).

the tanh activation function has a slow transition from -1 to 1, which may not be able to push $l_x$ and $l_y$ away from the origin $[0, 0]$ of an image, and likely causes ineffective training as well. Therefore, we treat $\beta$ as a hyperparameter and tune it on the validation set. The results with different values of $\beta$ on VGGFace are shown in Table 3.5 and Figure 3.10. It can be observed that we get the highest ASR with $\beta = 3000$. With small $\beta$s like 100 or 500, the patch location saturates at the corners of images; With large $\beta$s such as 5000 or 7000, the learned patch locations are close to the origin for most of the images. We find $\beta = 3000$ works well across a variety of architectures and datasets, and thus set it as the default value.

| $\beta$ | 100 | 500 | 1000 | 3000 | 5000 | 7000 |
|---|---|---|---|---|---|---|
| ASR | 15.6 | 20.8 | 88.2 | **88.4** | 88.1 | 87.9 |

Table 3.5 ASRs of GDPA with different values of $\beta$. We use 5% of pixels as the patch size.

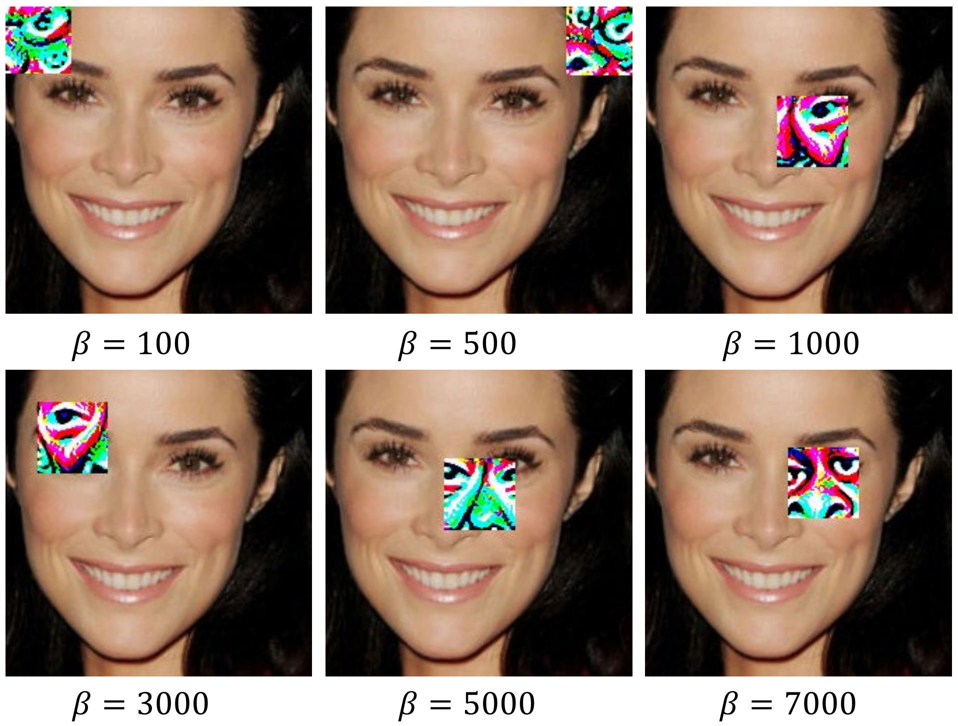


Figure 3.10 Perturbed images by GDPA with different values of $\beta$. The patch size is 5% of pixels.

### *3.4.4 Generating Static Patch Attack with GDPA*

Contrary to dynamic patch attack, static patch attack uses a fixed patch location for all the images. To conduct static patch attack with GDPA, we set $l_x$ and $l_y$ to fix values instead of generating them from $G_L$. To compare the performance between dynamic and static patch attacks, we conduct static patch attacks on VGGFace at 25 fixed locations ($l_x, l_y \in [-0.8, -0.4, 0, 0.4, 0.8]$). We use patch size $32 \times 32$ (2% of pixels) in the experiment.

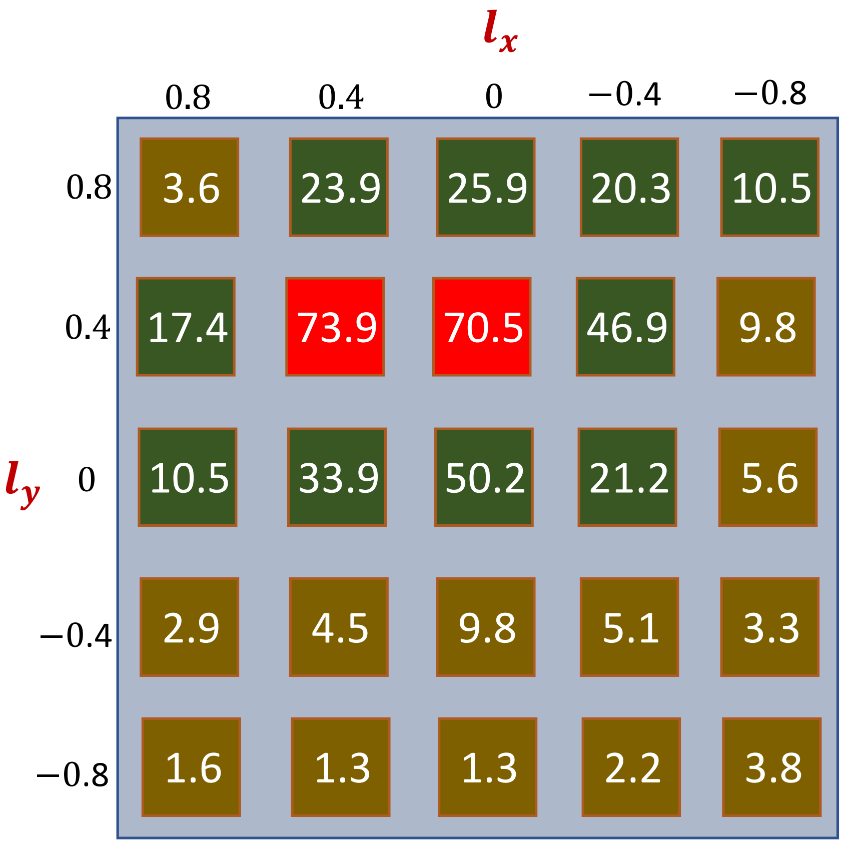


Figure 3.11 ASRs of static patch attack on different locations. We use different colors to denote ASRs in different ranges. Red: above 70%; Green: 10% - 70%; brown: below 10%. Dynamic GDPA achieves 76.4% ASR in this experiment.

Figure 3.11 shows the ASRs of static patch attacks at the 25 locations. As we can see, patch location is an important factor in the performance of static patch attack. Notably, patch locations around the area of eyes have the best ASRs. The highest ASR we obtain from static patch attack is 73.9%, while dynamic GDPA achieves 76.4%, demonstrating the effectiveness of dynamic GDPA.

| | | $\epsilon = 6$ | $\epsilon = 8$ | $\epsilon = 10$ |
|---|---|---|---|---|
| VGGFace | PGD | 81.5 | 90.7 | 97.8 |
| | GDPA-ADV | **82.6** | **91.9** | **98.2** |
| ImageNet | PGD | 58.3 | 65.2 | 71.3 |
| | GDPA-ADV | **60.5** | **68.6** | **87.8** |

Table 3.6 ASRs of the adversarial attacks generated by PGD and GDPA-ADV.

### *3.4.5 Generating Adversarial Attack with GDPA*

Thanks to its generic formulation, we can also generate conventional adversarial attacks with GDPA by adjusting its pipeline slightly. To do this, we use a fixed mask of value 0.5 for all image pixels, and update the generator to produce $p$ of the same size of image directly. To make sure the adversarial noise is within a small $L_\infty$-norm bound, we multiple $p$ by $\epsilon/255$ such that the adversarial noise is bounded by $\epsilon/255$. Finally, we scale the perturbed image by 2 and clip its pixel values to $[0, 1]$ to create an adversarial example. We call this GDPA version of adversarial examples as GDPA-ADV.

We then compare the attack performances of GDPA-ADV and PGD on VGGFace and ImageNet. The results with different $\epsilon$'s are provided in Table 3.6, where the PGD attack is generated with learning rate 10 for 20 iterations. It can be observed that GDPA-ADV achieves slightly higher ASRs than PGD in all the cases considered. Some adversarial examples generated by GDPA-ADV on VGGFace are visualized in Figure 3.12. These adversarial examples look very similar to the conventional adversarial examples.

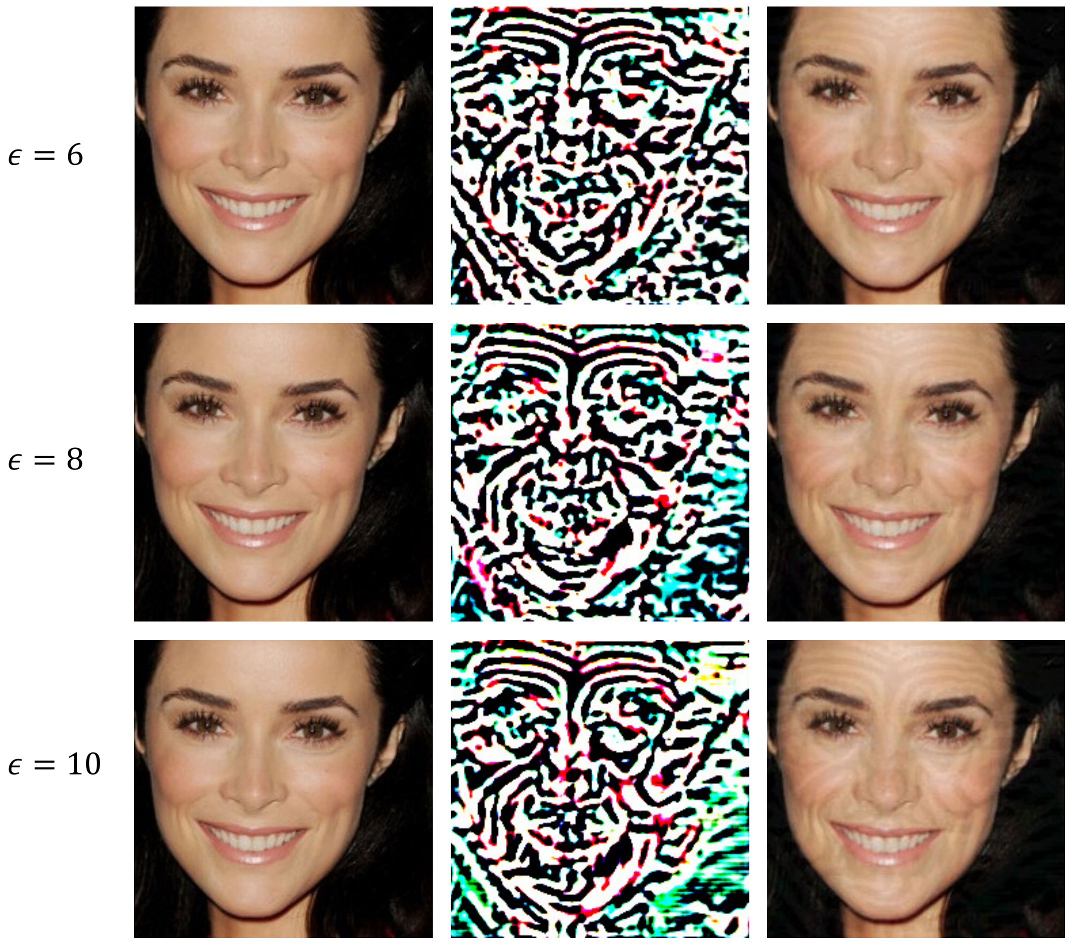


Figure 3.12 Adversarial examples generated by GDPA-ADV with different $\epsilon$'s. Left: original images; Middle: adversarial noise scaled to $[0, 1]$ for visualization; Right: adversarial examples.

### *3.4.6 Cross Attacks and Defenses*

In this section, we compare the defense performances of PGD-AT, DOA and GDPA-AT when they are attacked by their corresponding attack algorithms. In this experiment, the PGD attack uses $\epsilon = 8$, and ROA and GDPA use 10% pixels as patch size. The results on VGGFace are shown in Table 3.7. As we can see, PGD-AT achieves the highest robustness under the PGD attack, but is not very robust under the ROA and GDPA attacks. On the other hand, DOA achieves decent robustness under the ROA and GDPA attacks, but fails completely under the PGD attack. Notably, GDPA-AT is the only defense algorithm that achieves almost the highest robustness under all three attacks. It's expected that GPDA-AT would be robust under the ROA and GDPA attacks since both are patch attacks. An explanation of the robustness of GDPA-AT under the PGD attack is provided in Section 4.2.

| AT \ Attack | PGD | ROA | GDPA |
|---|---|---|---|
| PGD-AT | **96.1** | 32.8 | 30.5 |
| DOA | 0 | 88.1 | 86.9 |
| GDPA-AT | 94.6 | **90.4** | **88.2** |

Table 3.7 Accuracies of adversarially trained models under PGD, ROA and GDPA attacks.

### *3.4.7 Additional Results on Targeted Attack*

Figure 3.13 provides additional perturbed images generated by targeted GDPA attack on VGGFace. The top row shows the target subjects, while the bottom two rows show the perturbed images with different patch sizes. As we can see, the patches generated by GDPA attempt to replace the corresponding face features with the ones from the target subjects.

### *3.4.8 Inference Speed*

Besides the improved attack and defense performance of GDPA, another advantage of GDPA is its superior inference speed to generate attacks over the optimization-based methods, such as PGD Madry et al. (2018) and ROA Wu et al. (2019). To have a quantitative comparison in terms of inference time, we evaluate the run-time of GDPA, PGD and ROA on the VGGFace

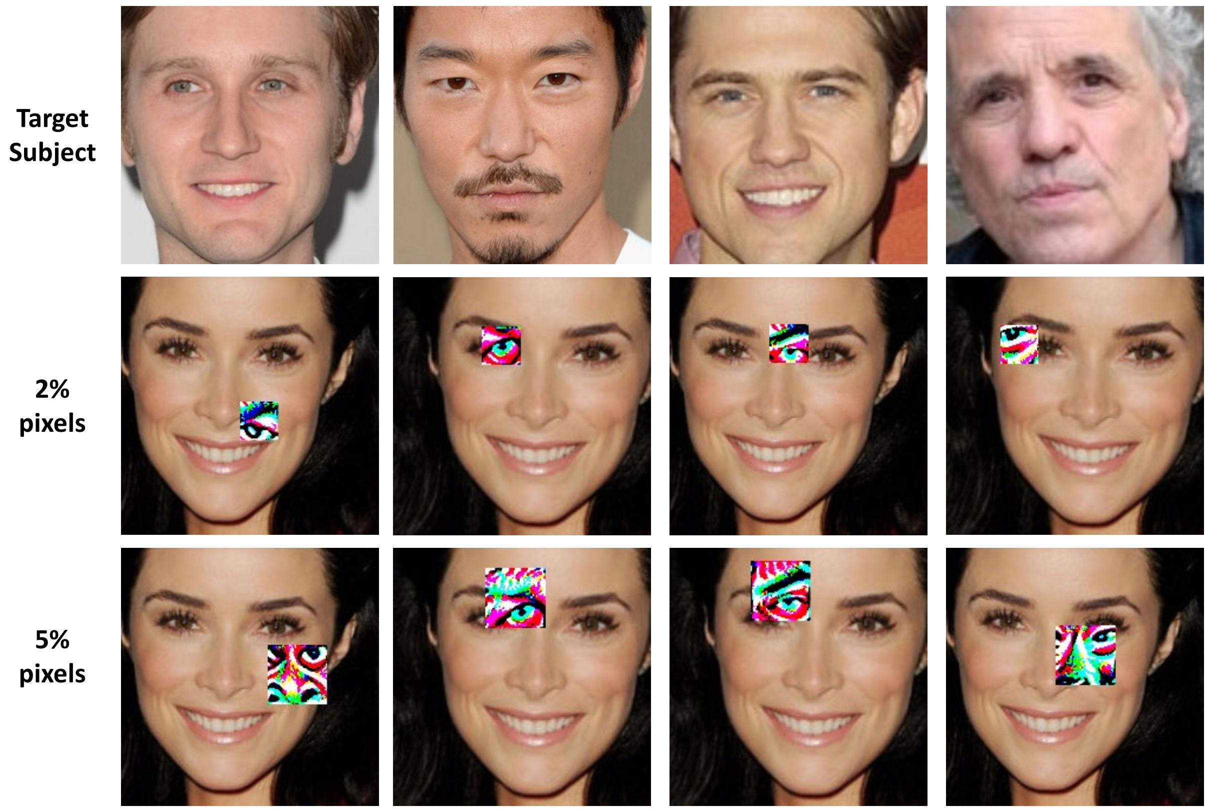


Figure 3.13 Perturbed images generated by GDPA with targeted attack on VGGFace. Each column corresponds to one targeted attack with a different target subject.

test dataset (470 images). GDPA needs one forward propagation to generate a patch attack, while we follow the settings of ROA and PGD and run 50 iterative optimizations to generate their attacks. As shown in Table 3.8, GDPA is about 40x faster than PGD and 47x faster than ROA.

## 3.5 Conclusion

This paper introduces GDPA, a novel *dynamic* patch attack algorithm, that generates patch pattern and patch location altogether for each input image. Due to its generic formulation,

|  | Inference-time (s) |
|---|---|
| PGD Madry et al. (2018) | 108.31 |
| ROA-Grad Wu et al. (2019) | 129.42 |
| ROA-Ex Wu et al. (2019) | 248.82 |
| **GDPA** | **2.76** |

Table 3.8 Inference-time comparison of different attack algorithms on the VGGFace test dataset (470 images).

GDPA can generate dynamic/static and visible/invisible patch attacks. GDPA is end-to-end differentiable, which entails an efficient optimization and easy integration for adversarial training. We validated our method on multiple benchmarks with different model architectures. GDPA demonstrates superior ASR over strong patch attack methods, and the adversarially trained model with GDPA is more robust to both patch attacks and conventional adversarial attacks. Moreover, GDPA is 40-50x faster than competing attack algorithms, making it a highly effective attack and defense algorithm.

# CHAPTER 4
# Neural Image Compression and Explanation

## 4.1 Introduction

Deep neural networks (DNNs) have become the de-facto performing technique in the field of computer vision He et al. (2016), natural language processing Devlin et al. (2018), and speech recognition Battenberg et al. (2017). Given sufficient data and computation, they require only limited domain knowledge to reach state-of-the-art performance. However, the current DNNs are largely black-boxes with many layers of convolution, non-linearities, and gates, optimized solely for competitive performance, and our understanding of the reasoning of DNNs is rather limited. DNNs' predictions may be backed up by a claimed high accuracy on benchmarks. However, it is human's nature not to trust them unless human experts are able to verify, interpret, and understand the reasoning of the system. Therefore, the usage of DNNs in real world decision-critical applications, such as surveillance cameras, drones, autonomous driving, medicine and legal, still must overcome a trust barrier. To address this problem, researchers have developed many different approaches to explain the reasonings of DNNs Simonyan et al. (2013); Zeiler & Fergus (2014); Bach et al. (2015); Gan et al. (2015); Ribeiro et al. (2016); Dabkowski & Gal (2017); Fong & Vedaldi (2017); Li et al. (2016); Lundberg & Lee (2017); Shrikumar et al. (2017); Yang et al. (2018); Ish-Horowicz et al. (2019). Intuitively, interpretable explanations should be concise and coherent such that they are easier for human to comprehend. However, most existing approaches do not take these requirements into account as manifested by the opaqueness and redundancies in their explanations Simonyan et al. (2013); Bach et al. (2015); Dabkowski & Gal (2017); Fong & Vedaldi (2017).

On the other hand, over 70% of internet traffic today is the streaming of digital media, and this percentage keeps rising over years some string reflecting how you wish the entry alphabetized (2019). It has been challenging for classic compression algorithms, such as JPEG and PNG, to adapt to the growing demand. Recently, there is an increasing interest of using machine learning (ML) based approaches to improve the compression of images and videos Ballé et al. (2017); Prakash et al. (2017); Johnston et al. (2018); Nakanishi et al. (2018). Rather than using manually engineered basis functions for compression, these ML-based techniques learn semantic structures and basis functions directly from training images and achieve impressive performance compared to the classic compression algorithms.

Usually, neural explanation and semantic image compression are addressed independently by two different groups of researchers. In light of the similarity between sparse explanation to image classification and sparse representation for image compression, in this paper we propose a deep learning based framework that integrates neural explanation and semantic image compression into an end-to-end training pipeline. With this framework, we can train a sparse mask generator to generate a concise and coherent mask to explain the prediction of CNN; subsequently, this sparse mask can be used to generate a mixed-resolution image with a very high compression rate, superior to the existing semantic compression algorithms. This Neural Image Compression and Explanation (NICE) framework is critical to many real world decision-critical systems, such as surveillance cameras, drones and self-driving cars, that heavily rely on the deep learning techniques today. For these applications, the outputs of NICE: prediction, sparse mask / explanation, and the compressed mixed-resolution image can be stored or transmitted efficiently for decision making, decision interpretation and system diagnosis.

The main contributions of the paper are:

- We propose a deep learning based framework that unifies neural explanation and semantic image compression into an end-to-end trainable pipeline, which produces prediction, sparse explanation and compressed images at the same time;

- The proposed $L_0$-regularized sparse mask generator is trained in a weakly supervised manner without resorting to expensive dense pixel-wise annotations, and outperforms many existing explanation algorithms that heavily rely on backpropagation;

- The proposed mixed-resolution image compression achieves a higher compression rate compared to the existing semantic compression algorithms, while retaining a similar classification accuracy with the original images.

- The proposed method is very efficient compared to the backpropagation-based alternatives, such as Saliency Map Simonyan et al. (2013) and CAM Zhou et al. (2016), as our method only requires forward propagation of the generator network. Experiments show that NICE is about 23x faster than Saliency Map, 16.5x faster than CAM and 2.8x faster than RTIS Dabkowski & Gal (2017). This makes our method widely deployable in real-time applications.

## 4.2 Related Work

Our work is related to two active research areas of deep learning: neural explanation and semantic image compression. We therefore review them next.

### *4.2.1 Neural Explanation*

In order to interpret DNN's prediction and gain insights of their operations, a variety of neural explanation methods have been proposed in recent years Simonyan et al. (2013); Zeiler & Fergus (2014); Bach et al. (2015); Gan et al. (2015); Ribeiro et al. (2016); Zhou et al. (2016); Dabkowski & Gal (2017); Fong & Vedaldi (2017); Li et al. (2016); Lundberg & Lee (2017); Selvaraju et al. (2017); Shrikumar et al. (2017); Yang et al. (2018); Ish-Horowicz et al. (2019); Chen et al. (2018a); Bang et al. (2019). These methods can be categorized based on whether it is designed to explain the entire model behavior (global interpretability) or a single prediction (local interpretability) Carvalho et al. (2019). The goal of global interpretability is to identify predictor variables that best explain the overall performance of a trained model. This class of methods are crucial to inform population level decision for rule extraction or knowledge discovery Yang et al. (2018); Ish-Horowicz et al. (2019). Local interpretability aims to produce interpretable explanations for each individual prediction and the interpretability occurs locally. Local interpretability is by far the most explored area of explainable AI Simonyan et al. (2013); Ribeiro et al. (2016); Zhou et al. (2016); Lundberg & Lee (2017); Chen et al. (2018a); Bang et al. (2019). The primary idea is to measure a change of the final prediction with respect to changes of input or getting feature attribution for the final prediction. Different local explanation methods implement this idea in different ways. For example, occlusion-based explanation methods remove or alter a fraction of input data and evaluate its impact to the final prediction Dabkowski & Gal (2017); Fong & Vedaldi (2017); Li et al. (2016); Zeiler & Fergus (2014). Gradient-based methods compute the gradient of an output with respect to an input sample by using backpropagation to locate salient features that are responsible to the prediction Bach et al. (2015); Gan et al. (2015); Selvaraju et al. (2017); Shrikumar et al. (2017). Other local interpretability methods explain data instances by approximating the decision boundary of a DNN with an inherently interpretable model around the predictions. For example, LIME Ribeiro et al. (2016) and SHAP Lundberg & Lee (2017) sample perturbed instances around a single data sample and fit a linear model to perform local explanations. RTIS Dabkowski & Gal (2017) extracts features from a DNN classifier and feeds extracted features and target label to an U-Net like generator to generate saliency maps for local explanations. L2X Chen et al. (2018a) learns a stochastic map based on mutual information that selects instance-wise informative features. Built on top of L2X, VIBI Bang et al. (2019) selects instance-wise key features that are maximally compressed about an input and informative about a decision based on an information-bottleneck principle.

NICE falls in the category of local interpretability and aims to produce concise and coherent local explanations similar to Saliency Map Simonyan et al. (2013), RTIS Dabkowski & Gal (2017) and VIBI Bang et al. (2019). But our method achieves briefness and comprehensiveness explicitly through an $L_0$-norm regularization and a smoothness constraint, optimized via stochastic binary optimization.

The sparse mask generator of NICE is also related to a large body of research on semantic segmentation Long et al. (2015); He et al. (2017); Chen et al. (2018b); Papandreou et al. (2015); Pinheiro & Collobert (2015); Bearman et al. (2016); Chen et al. (2019b). In particular, our sparse mask generator is trained to maximize the final classification accuracy of the mixed-resolution images without resorting to expensive dense pixel-wise annotations. Therefore, it can be considered as a weakly supervised *binary* segmenation algorithm that detects salient regions of an image. This is different to the existing semantic segmentation algorithms that employ different levels of supervision, such as full pixel-wise annotation Long et al. (2015); He et al. (2017), image-level labels Papandreou et al. (2015), bounding boxes Dai et al. (2015), scribbles Lin et al. (2016), points Bearman et al. (2016), or adversarial loss Chen et al. (2019b). Since the main goal of NICE is to provide a competitive or improved neural explanation, in our experiments we mainly compare NICE with deep explanation methods instead of segmentation algorithms.

#### *4.2.2 Semantic Image Compression*

Classic image compression algorithms, such as JPEG Wallace (1992) and PNG Sayood (2002), have hard-coded procedures / components to compress images. For example, the JPEG compression first employs a discrete cosine transform (DCT) over each $8 \times 8$ image block, followed by quantization to represent the frequency coefficients as a sequence of binaries. The DCT can be seen as a generic feature extractor with a fixed set of basis functions that are irrespective of the distribution of the input images. Compared to standard image compression algorithms, the ML-based approaches Ballé et al. (2016); Toderici et al. (2016, 2017); Theis et al. (2017); Ballé et al. (2017); Johnston et al. (2018); Li et al. (2018b); Nakanishi et al. (2018) can automatically discover semantic structures and learn basis functions from training images to achieve even higher compression rate. All of these ML-based approaches follow a similar structure of autoencoder, where an encoder is used to extract feature representation from images and a decoder is responsible to reconstruct images from the quantized representations. The main differences among these ML-based approaches are the architectures of encoder and decoder. While the majority of these algorithms Ballé et al. (2016); Theis et al. (2017); Ballé et al. (2017); Johnston et al. (2018); Li et al. (2018b); Nakanishi et al. (2018) employ CNNs as the encoder and decoder, some others explore recurrent networks such as LSTM and GRU Toderici et al. (2016, 2017).

To the best of our knowledge, all of these methods are not sufficiently content-aware, except the work Prakash et al. (2017) from Prakash *et al.* which is probably the most relevant work to ours. While Prakash *et al.* adopt CAM Zhou et al. (2016) as the semantic region detector, we develop a principled $L_0$-regularized sparse mask generator to detect the semantic regions and further compress images with mixed resolutions. We will compare NICE with Prakash et al. (2017) when we present results of semantic image compression.

### 4.3 The NICE Framework

Given a training set $D = \{(\boldsymbol{x}_i, y_i), i = 1, 2, \cdots, N\}$, where $\boldsymbol{x}_i$ denotes the $i$-th input image and $y_i$ denotes the corresponding target, a neural network is a function $h(\boldsymbol{x}; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$ that fits to the training data $D$ with the goal of achieving good generalization to unseen test data. To optimize $\boldsymbol{\theta}$, typically the following empirical risk minimization (ERM) is adopted:

$$\mathcal{R}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\left(h\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right), y_i\right), \tag{4.1}$$

where $\mathcal{L}(\cdot)$ denotes the loss over training data $D$, such as the cross-entropy loss for classification or the mean squared error (MSE) for regression. The goal of this paper is to develop an approach that can explain the prediction of a neural network $h(\boldsymbol{x};\theta)$ in response to an input image $\boldsymbol{x}$; meanwhile, to reduce storage or network transmission cost of the image, we'd like to compress the image $\boldsymbol{x}$ based on the above derived explanation such that the compressed image $\tilde{\boldsymbol{x}}$ has the minimal file size while retaining a similar classification accuracy as the original image $\boldsymbol{x}$.

To meet these interdependent goals, we develop a Neural Image Compression and Explanation (NICE) framework that integrates explanation and compression into an end-to-end trainable pipeline as illustrated in Fig. 4.1. In this framework, given an input image, a mask generator under the $L_0$-norm and smoothness constraints generates a sparse mask that indicates salient regions of the image. The generated mask is then used to transform the original input image to a mixed-resolution image that has a high resolution in the salient regions and a low resolution in the background. To evaluate the quality of sparse mask generator and the compressed image, at the end of the pipeline a discriminator network (e.g., CNN) classifies the generated image for prediction. Finally, the prediction, sparse mask and compressed image can be stored or transmitted efficiently for decision making, interpretation and system diagnosis. The whole pipeline is fully differentiable and can be trained end-to-end by backpropagation. We will introduce each of these components next.

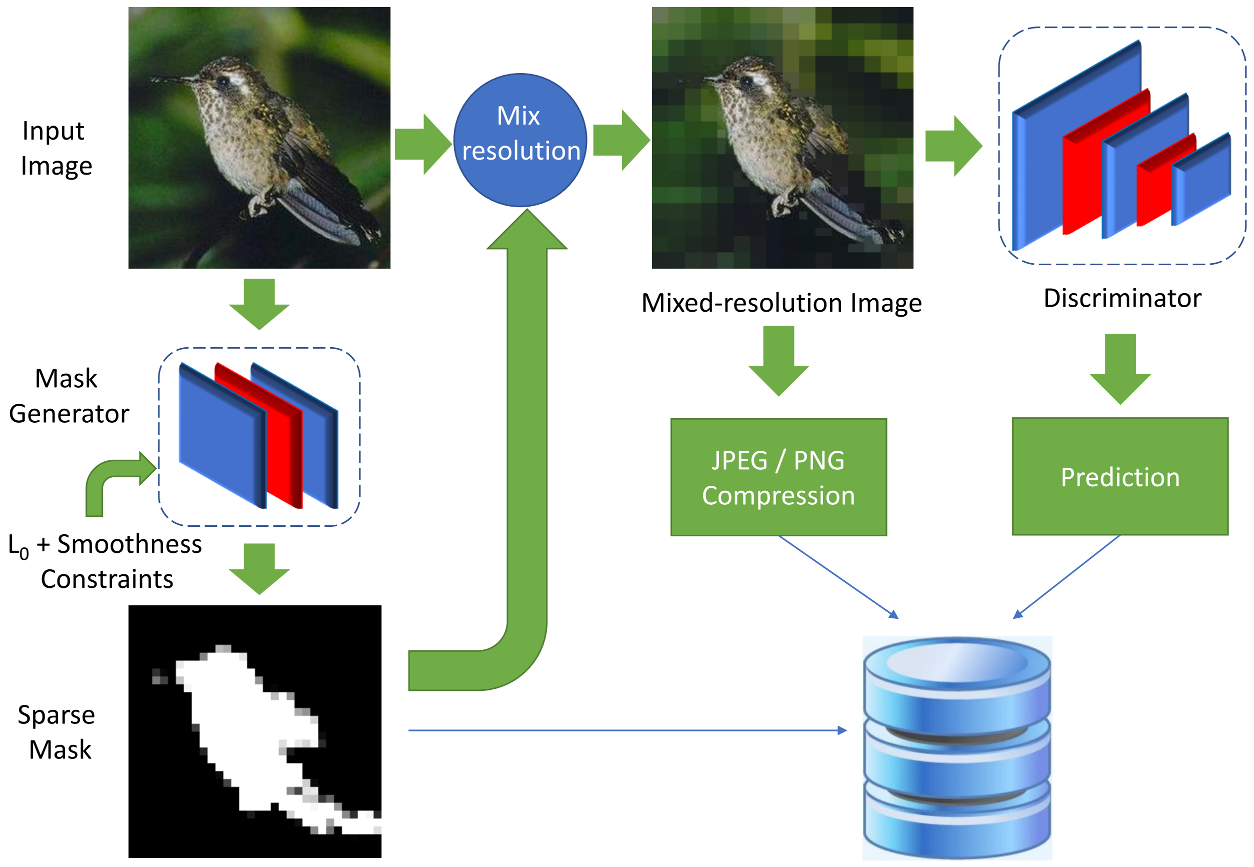


Figure 4.1 Overall architecture of NICE.

## *4.3.1 Sparse Neural Explanation*

To correctly classify an image, a state-of-the-art CNN classifier does not need to analyze all the pixels in an image. Partially, this is because not all the pixels in an image are equally important for image recognition. For example, although the background pixels may provide some useful clues to recognize an object, it is the pixels on the object that play a decisive role for recognition. Based on this understanding, we'd like to learn a set of random variables (one for each pixel of an image) such that the variables on object pixels receive high values while the variables on background pixels receive low values. In other words, we want to learn a binary segmentation model that can partition pixels into object pixels and background pixels. To make our segmentation discriminative, we require the output of our model to be sparse/concise such that only the most important or influential pixels receive high values, and the remaining pixels receive low values. Furthermore, we expect the segmentation to be smooth/coherent within a small continuous region since most of natural objects usually have smooth appearances. We therefore request our neural explanation model to produce explanations that are concise and coherent. We will materialize these two requirements mathematically.

We model our neural explanation by attaching a binary random variable $z \in \{0, 1\}$ to each pixel of an image:

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i \odot \boldsymbol{z}_i, \quad \boldsymbol{z}_i \in \{0, 1\}^P, \tag{4.2}$$

where $\boldsymbol{z}_i$ denotes a binary mask for image $\boldsymbol{x}_i$, and $\odot$ is an element-wise product. Furthermore, we define $z_i^j$ the binary variable for pixel $j$ of image $\boldsymbol{x}_i$. We assume both image $\boldsymbol{x}_i$ and its mask $\boldsymbol{z}_i$ have the same spatial dimension of $m \times n$ or $P$ pixels. After training, we wish $z_i^j$ takes value 1 if pixel $j$ is on object and 0 otherwise.

We regard $\boldsymbol{z}_i$ as our explanation to the prediction of $h(\boldsymbol{x}_i; \boldsymbol{\theta})$ and learn $\boldsymbol{z}_i$ by minimizing the following $L_0$-norm regularized loss function:

$$\begin{aligned}\mathcal{R}(\boldsymbol{\theta}, \boldsymbol{z}) &= \frac{1}{N}\sum_{i=1}^{N}\Big(\mathcal{L}\left(h\left(\boldsymbol{x}_i \odot \boldsymbol{z}_i; \boldsymbol{\theta}\right), y_i\right) + \lambda ||\boldsymbol{z}_i||_0\Big) \\ &= \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{L}\left(h\left(\boldsymbol{x}_i \odot \boldsymbol{z}_i; \boldsymbol{\theta}\right), y_i\right) + \lambda \sum_{j=1}^{P} 1_{[z_i^j \neq 0]}\right),\end{aligned} \tag{4.3}$$

where $\mathbf{1}_{[c]}$ is an indicator function that is 1 if the condition $c$ is satisfied, and 0 otherwise. Here, we insert (4.2) into (4.1) and add an $L_0$-norm on the elements of $\boldsymbol{z_i}$, which *explicitly* measures number of non-zeros in $\boldsymbol{z}_i$ or the sparsity of $\boldsymbol{z}_i$. By doing so, we'd like the masked image achieves the similar classification accuracy as the original image, while using as fewer pixels as possible. In other words, the sparse mask $\boldsymbol{z}_i$ can produce a concise explanation to the prediction of the classifier (i.e., the first requirement). To optimize (4.3), however, we note that both the first term and the second term of (4.3) are not differentiable w.r.t. $\boldsymbol{z}$. Therefore, further approximations need to be considered.

We can approximate this optimization problem via an inequality from stochastic variational optimization Bird et al. (2018a). Specifically, given any function $\mathcal{F}(\boldsymbol{z})$ and any distribution $q(\boldsymbol{z})$, the following inequality holds

$$\min_{\boldsymbol{z}} \mathcal{F}(\boldsymbol{z}) \leq \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z})}[\mathcal{F}(\boldsymbol{z})], \tag{4.4}$$

i.e., the minimum of a function is upper bounded by the expectation of the function. With this result, we can derive an upper bound of (4.3) as follows.

Since $z_i^j, \forall j \in \{1, \cdots, P\}$ is a binary random variable, we assume $z_i^j$ is subject to a Bernoulli distribution with parameter $\pi_i^j \in [0, 1]$, i.e. $z_i^j \sim \text{Ber}(z; \pi_i^j)$. Thus, we can upper bound $\min_{\boldsymbol{z}} \mathcal{R}(\boldsymbol{\theta}, \boldsymbol{z})$ by the expectation

$$\tilde{\mathcal{R}}(\boldsymbol{\theta}, \boldsymbol{\pi}) = \frac{1}{N}\sum_{i=1}^{N} \Big( \mathbb{E}_{q(\boldsymbol{z}_i|\boldsymbol{\pi}_i)} \Big[ \mathcal{L}\left( h\left( \boldsymbol{x}_i \odot \boldsymbol{z}_i; \boldsymbol{\theta} \right), y_i \right) \Big] + \lambda \sum_{j=1}^{P} \pi_i^j \Big). \tag{4.5}$$

Now the second term of (4.5) is differentiable w.r.t. the new model parameters $\boldsymbol{\pi}$. However, the first term is still problematic since the expectation over a large number of binary random variables $\boldsymbol{z}_i \in \{0, 1\}^P$ is intractable, so is its gradient.

#### *4.3.1.1 The Hard Concrete Gradient Estimator*

Fortunately, this kind of binary latent variable models has been investigated extensively in the literature. There exist a numerous of gradient estimators to this problem, including REINFORCE Williams (1992), Gumble-Softmax Jang et al. (2017); Maddison et al. (2017), REBAR Tucker et al. (2017), RELAX Grathwohl et al. (2018) and the hard concrete estimator Louizos et al. (2018), among which the hard concrete estimator is the one that is easy to implement and demonstrates superior performance in our experiments. We therefore resort to this gradient estimator to optimize (4.5). Specifically, the hard concrete gradient estimator employs a reparameterization trick to approximate the original optimization problem of (4.5) by a close surrogate loss function

$$\begin{aligned}\hat{\mathcal{R}}(\boldsymbol{\theta},\log\boldsymbol{\alpha}) &= \frac{1}{N}\sum_{i=1}^{N}\Bigg(\mathbb{E}_{\boldsymbol{u}_i\sim\mathcal{U}(0,1)}\Big[\mathcal{L}(h(\boldsymbol{x}_i\odot g(f(\log\boldsymbol{\alpha}_i,\boldsymbol{u}_i));\boldsymbol{\theta}),y_i)\Big] \\ &\quad + \lambda\sum_{j=1}^{P}\sigma\left(\log\alpha_i^j - \beta\log\frac{-\gamma}{\zeta}\right)\Bigg) \\ &= \mathcal{L}_D(\boldsymbol{\theta},\log\boldsymbol{\alpha}) + \lambda\mathcal{L}_C(\log\boldsymbol{\alpha}), \end{aligned} \tag{4.6}$$

with

$$\begin{aligned} f(\log\boldsymbol{\alpha}_i,\boldsymbol{u}_i) = \sigma\left((\log\boldsymbol{u}_i - \log(1-\boldsymbol{u}_i)\right. \\ \left. + \log\boldsymbol{\alpha}_i)/\beta\right)(\zeta-\gamma)+\gamma, \end{aligned} \tag{4.7}$$

and

$$g(\cdot) = \min(1,\max(0,\cdot)), \tag{4.8}$$

where $\sigma(t) = 1/(1+\exp(-t))$ is the sigmoid function, $\mathcal{L}_D$ measures how well the classifier fits to training data $D$, $\mathcal{L}_C$ measures the expected number of non-zeros in $\boldsymbol{z}$, and $\beta = 2/3$, $\gamma = -0.1$ and $\zeta = 1.1$ are the typical parameters of the hard concrete distribution. Function $g(\cdot)$ is a hard-sigmoid function that bounds the stretched concrete distribution between 0 and 1. For more details on the hard concrete gradient estimator, we refer the readers to Louizos et al. (2018). With this reparameterization, the surrogate loss function (4.6) is differentiable w.r.t. its parameters.

#### *4.3.1.2 Smoothness Regularization*

The $L_0$-regularized objective function developed above enforces the sparsity/conciseness of an explanation. To improve the coherence of an explanation, we introduce an additional smoothness constraint on the mask:

$$\begin{aligned}\mathcal{L}_S(\log \boldsymbol{\alpha}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{q(\boldsymbol{z}_i|\log \boldsymbol{\alpha}_i)}\Bigg[\sum_{m,n=1}^{w,h}\Big(\big|z_i^{m,n}- z_i^{m-1,n}\big| \\ +\big|z_i^{m,n}-z_i^{m,n-1}\big|+\big|z_i^{m,n}-z_i^{m-1,n-1}\big|+\big|z_i^{m,n}- z_i^{m-1,n+1}\big|\Big)\Bigg] \\ \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{m,n=1}^{w,h}\big(\big|y_i^{m,n}- y_i^{m-1,n}\big|+\big|y_i^{m,n}- y_i^{m,n-1}\big| \\ +\big|y_i^{m,n}- y_i^{m-1,n-1}\big|+\big|y_i^{m,n}- y_i^{m-1,n+1}\big|\big)\,, \end{aligned} \tag{4.9}$$

where $y_i^{m,n}$ is the expectation of random variable $z_i^{m,n}$ under the hard concrete distribution $q(\boldsymbol{z}_i|\log \boldsymbol{\alpha}_i)$, which can be calculated as:

$$y = \mathbb{E}_{q(z|\log \alpha)}[z] = \sigma\left(\log \alpha - \beta \log \frac{-\gamma}{\zeta}\right). \tag{4.10}$$

Note that this smoothness constraint penalizes the discrepancy of $z$ among its four neighborhoods, and thus a coherence explanation is preferred (i.e., the second requirement). To avoid notational clutter, in (4.9) some of the boundary conditions are not rigorously checked, but we hope they will be apparent given the context. With this additional regularization, our final objective is then a composition of three terms

$$\mathcal{L}(\boldsymbol{\theta}_d, \log \boldsymbol{\alpha}) = \mathcal{L}_D + \lambda_1 \mathcal{L}_C + \lambda_2 \mathcal{L}_S, \tag{4.11}$$

where $\lambda_1$ and $\lambda_2$ are the regularization hyperparameters that balance the data loss $\mathcal{L}_D$, the capacity loss $\mathcal{L}_C$ and the smoothness loss $\mathcal{L}_S$. It is worthy noting that from now on we denote the parameters of classifier (discriminator) $\boldsymbol{\theta}_d$ to distinguish it from the parameters of generator $\boldsymbol{\theta}_g$ that will be introduced next.

After training, we get $\log \boldsymbol{\alpha}$ for each input image $\boldsymbol{x}$. At testing time, we employ the following estimator to generate a sparse mask:

$$\hat{\boldsymbol{z}} = \min(\mathbf{1}, \max(\mathbf{0}, \sigma\left((\log \boldsymbol{\alpha})/\beta\right)(\zeta - \gamma) + \gamma)), \tag{4.12}$$

which is the sample mean of $\boldsymbol{z}$ under the hard concrete distribution $q(\boldsymbol{z}|\log \boldsymbol{\alpha})$.

### *4.3.2 Semantic Image Compression*

Upon receiving the sparse mask $\hat{\boldsymbol{z}}$ from above, we can use it to generate a mixed-resolution image for semantic image compression, as shown in Fig. 4.1. Suppose that we have an input image $\boldsymbol{x}$ and a sparse mask $\hat{\boldsymbol{z}} \in [0,1]^P$, a mixed-resolution image can be generated by

$$\tilde{\boldsymbol{x}} = M(\boldsymbol{x}, \hat{\boldsymbol{z}}) = \boldsymbol{x} \odot \hat{\boldsymbol{z}} + \boldsymbol{x}_b \odot (1 - \hat{\boldsymbol{z}}), \tag{4.13}$$

where $\boldsymbol{x}_b$ is a low resolution image that can be generated by subsampling original image $\boldsymbol{x}$ with a block size of $b \times b$, which can be efficiently implemented by average pooling with a $b \times b$ filter and a stride of $b$. Here $b$ is a tunable hyperparameter that trades off between the image compression rate and the classification accuracy of the classifier. In other words, the larger $b$ is, the lower resolution images will be generated and thus a lower classification accuracy, and vice-versa. As we can see, when $b = 1$ the mixed-resolution image $\tilde{\boldsymbol{x}}$ is equal to the original image $\boldsymbol{x}$; when we use the image size as $b$, the mixed-resolution image $\tilde{\boldsymbol{x}}$ becomes a masked image with a constant value as background. When $b$ is a value between these two extremes, we can generate mixed-resolution images of different levels of quality.

### *4.3.3 Sparse Mask Generator*

The learning of sparse mask $\boldsymbol{z}$ discussed above is transductive, by which we can learn a mask for each image in training set $D$. However, this approach cannot generate masks for new images that are not in the training set $D$. A more desirable approach is inductive, which can be implemented through a generator $G(\boldsymbol{x}; \boldsymbol{\theta}_g)$ such that it can produce a sparse mask given any image $\boldsymbol{x}$ as input. We model this generator as a neural network parameterized by $\boldsymbol{\theta}_g$.

To integrate this generator into an end-to-end training pipeline, we model this generator to output $\log \boldsymbol{\alpha}$ given an input image $\boldsymbol{x}$; we can then sample a sparse mask $\boldsymbol{z}$ from the hard concrete distribution $q(\boldsymbol{z} | \log \boldsymbol{\alpha})$, i.e., $\boldsymbol{x} \xrightarrow{G(\cdot;\boldsymbol{\theta}_g)} \log \boldsymbol{\alpha} \xrightarrow{\text{sample}} \boldsymbol{z}$. With this reparameterization, the overall loss function (4.11) becomes $\mathcal{L}(\boldsymbol{\theta}_d, \boldsymbol{\theta}_g)$, which can be minimized by optimizing the generator network $\boldsymbol{\theta}_g$ and the discriminator network $\boldsymbol{\theta}_d$ jointly with backpropagation. In the experiments, we employ a CNN as our sparse mask generator as CNN is the de-facto technique today for image related analysis.

## 4.4 Experiments

To evaluate the performance of NICE, we conduct extensive experiments on three image classification benchmarks: MNIST LeCun et al. (1998), CIFAR10 Krizhevsky (2009) and Caltech256 Griffin et al. (2007) [1]. Since NICE is a neural explanation and semantic compression algorithm, we compare NICE with the state-of-the-art algorithms in neural explanation and semantic compression. For neural explanation, we compare NICE with Saliency Map Simonyan et al. (2013), RTIS Dabkowski & Gal (2017) and CAM Zhou et al. (2016) via visualization and the post-hoc classification. For semantic image compression, we compare NICE with the CAM-based method proposed in Prakash et al. (2017), a state-of-the-art semantic compression algorithm that is the most relevant to ours.

### *4.4.1 Implementation Details*

#### *4.4.1.1 Image Classification Benchmarks*

MNIST LeCun et al. (1998) is a gray-level image dataset containing 60,000 training images and 10,000 test images of the size $28 \times 28$ for handwritten digits classification. CIFAR10 Krizhevsky (2009) contains 10 classes of RGB images of the size $32 \times 32$, in which 50,000 images are for training and 10,000 images are for test. Caltech256 Griffin et al. (2007) is a high-resolution RGB image dataset containing 22,100 images from 256 classes of man-made and natural objects, such as plants, animals and buildings, etc. Since MNIST and CIFAR10 are low-resolution images, we use them mainly to demonstrate NICE's perfor-

[1]`http://www.vision.caltech.edu/Image_Datasets/Caltech256/`

s[th!]

Table 4.1 Network architectures of the generators and discriminators used in the experiments. Layer abbreviations used in the table: [C: Convolution; R: Relu; M: MaxPooling; Up: UpSample].

| Dataset | Generator | Discriminator |
|---|---|---|
| MNIST | C(1,1,3,1,1) | LeNet5-Caffe |
| CIFAR10 | C(1,1,5,1,0) + M(2) + Up(2) | VGG11 + FC(512, 10) |
| Caltech256 | C(3,1,3,1,1) + R + M(2)<br>C(1,1,3,1,1) + R + M(2)<br>C(1,1,3,1,1) + M(2) + Up(8) | ResNet18<br>FC(512, 256) |

mance on neural explanation. For the high-resolution images of Caltech256, we demonstrate NICE's performance on neural explanation and semantic image compression.

#### *4.4.1.2 Network Architectures and Training Details*

The network architectures of the sparse mask generators and CNN classifiers (discriminators) used in the MNIST, CIFAR10 and Caltech256 experiments are provided in Table 4.1.

We pretrain three CNN classifiers (discriminators) on the three image classification benchmarks: MNIST, CIFAR10 and Caltech256 and achieve the classification accuracies of 99%, 90.8% and 78.3%, respectively. These classifiers are the target CNNs we aim to explain. The architectures of the generators are tuned by us through extensive architecture search. The hyperparameters $\lambda_1$ and $\lambda_2$ in the overall loss (4.11) are tuned on validation set to balance the classification accuracy and sparsity/smoothness of the masks.

In the MNIST experiments, different $\lambda_1$s are used to generate sparse masks with different percentages of non-zeros (sparse explanations). $\lambda_2$ is set to 0 for all the MNIST experiments as the algorithm can generate coherent explanations without the smoothness constraint. The block size of the low resolution image $\boldsymbol{x}_b$ is set to 28, which means a constant background is used to generate the mixed-resolution images. We use the Adam optimizer Kingma & Ba (2015) with a learning rate of 0.001 and a decay rate of 0.1 at every 5 epochs.

In the CIFAR10 experiments, the block size of the low resolution image $\boldsymbol{x}_b$ is set to 32, thus a constant background image is used to generate the mixed-resolution images. We set $\lambda_1 = 3$ and $\lambda_2 = 0.01$ and train the pipeline by using the Adam optimizer with a learning rate of 0.001 and a decay rate 0.1 at every 5 epochs.

In the Caltech256 experiments, we split the dataset into a training set of 16,980 images and a test set of 5,120 images[2], where 5,120 images in training set is first used as validation set for architecture search and hyperparameter tuning and later the full 16,980 training images are used to train the final pipeline. The images are resized to $256 \times 256$ as inputs. We set $b = 256$ to generate the lowest resolution images $\boldsymbol{x}_b$, and set $\lambda_1 = 5$ and $\lambda_2 = 0.01$ and train the pipeline by using the SGD optimizer with a learning rate of 0.001 and a cosine decay function.

On different datasets, we experiment with different optimizers. The best performing one is selected based on its performance on validation set. It turns out that SGD works better on Caltech256, while Adam works better on MNIST and CIFAR10.

### *4.4.2 Explaining CNN's Predictions*

We first demonstrate NICE on explaining the predictions of the target CNNs we pretrained above. To do so, we incorporate the target CNN as discriminator into the pipeline (Fig. 4.1), and freeze its parameters $\boldsymbol{\theta}_d$ and only update the parameters of generator $\boldsymbol{\theta}_g$ by optimizing

[2]20 images per class are included in the test set.

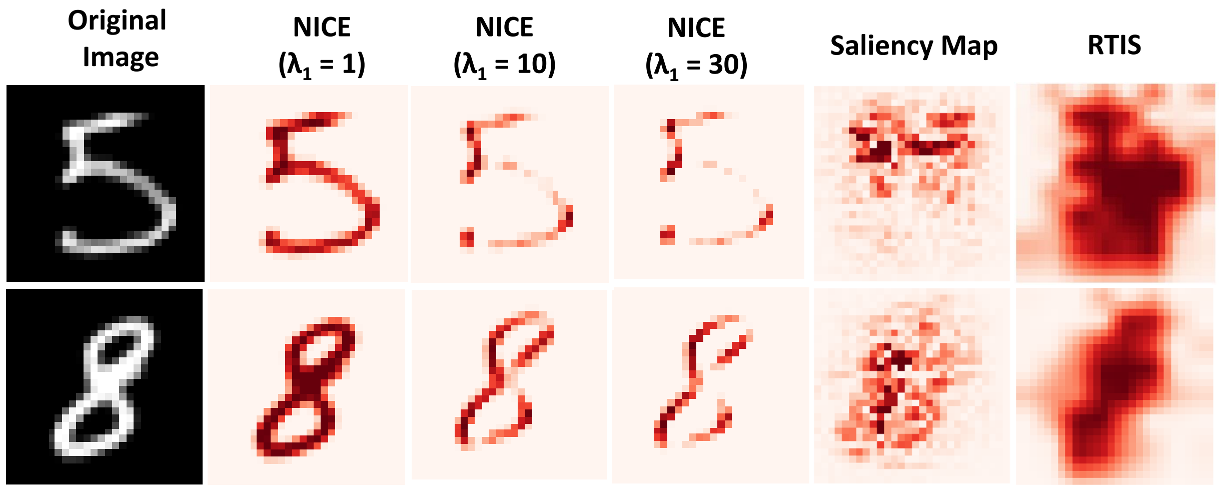


Figure 4.2 The sparse masks generated by NICE, Saliency Map Simonyan et al. (2013) and RTIS Dabkowski & Gal (2017) on the MNIST dataset. The dark red color represents high values (close to 1), indicating strong influence to the final decisions. By adjusting $\lambda_1$ of NICE, we can control the sparsity of the explanations.

the overall loss (4.11). The sparse mask $\boldsymbol{z}$ generated by the generator serves as the explanation to CNN's prediction since the mask indicates the salient region that has strong influence to the final prediction.

#### *4.4.2.1 MNIST*

We train the NICE pipeline on the MINST dataset to explain the prediction of the target LeNet5 classifier we pretrained above. Fig. 4.2 illustrates example sparse explanations generated by NICE with different $\lambda_1$s (when $\lambda_2 = 0$). As we can see, when $\lambda_1$ increases, the amount of non-zeros in the mask $\boldsymbol{z}$ decreases and NICE can produce sparser explanations to the final predictions. When $\lambda_1 = 1$, the explanations are almost identical to the input images, and when $\lambda_1 = 30$, the masks identify sparser but more influential regions for the final predictions. As a comparison, we also include the explanation results produced by Saliency Map Simonyan et al. (2013) and RTIS Dabkowski & Gal (2017) [3]. While NICE highlights coherent regions over digits as explanations, Saliency Map, a backpropagation-based approach, identifies discontinued regions as explanations, which are quite blurry and difficult to understand. RTIS can yield coherent regions as explanations but the regions identified are overly smooth. Apparently, the explanations produced by NICE are more concise, coherent and match well with how humans explain their own predictions.

#### *4.4.2.2 CIFAR10*

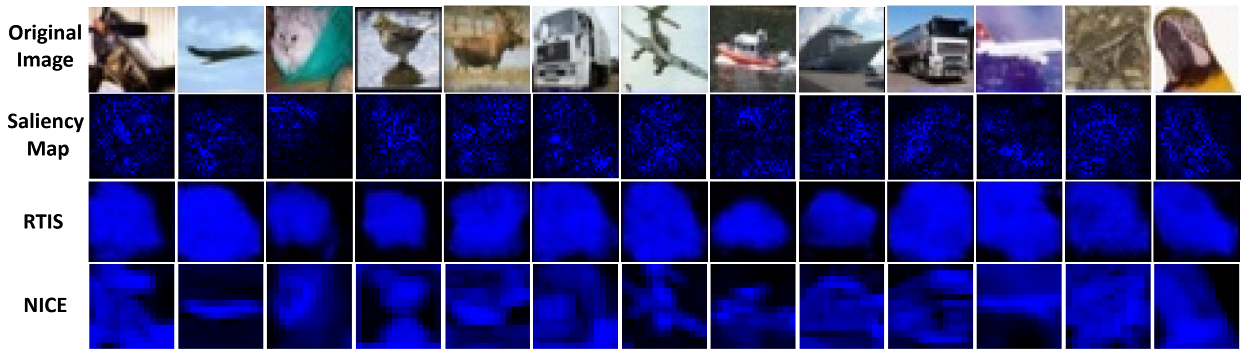


Figure 4.3 Comparison of explanations generated by Saliency Map Simonyan et al. (2013), RTIS Dabkowski & Gal (2017) and NICE on some CIFAR10 images. The RTIS results are from the RTIS paper. Compared to Saliency Map and RTIS, the explanations generated by NICE are more concise and the boundaries of salient regions are much sharper.

We also train the NICE pipeline on the CIFAR10 dataset to explain the target VGG11 classifier we pretrained above. Fig. 4.3 compares the explanations produced by Saliency Map Simonyan et al. (2013), RTIS Dabkowski & Gal (2017) and NICE on some CIFAR10

[3]CAM Zhou et al. (2016) does not perform well on small images, and we observe no published work provides CAM's results on MNIST and CIFAR10. We therefore ignore its results here as well.

images. The RTIS results are directly cited from the RTIS paper, and we apply Saliency Map and NICE on the same set of CIFAR10 images selected by the RTIS paper. Due to the low resolution of the images, it's very challenging to generate reliable explanations. As we can see, the explanations generated by NICE are more concise and the boundaries of salient regions are much sharper than those of Saliency Map and RTIS. The superior performance of NICE is most likely due to the $L_0$-norm regularization that *explicitly* promotes the sparsity of an explanation.

#### *4.4.2.3 Caltech256*

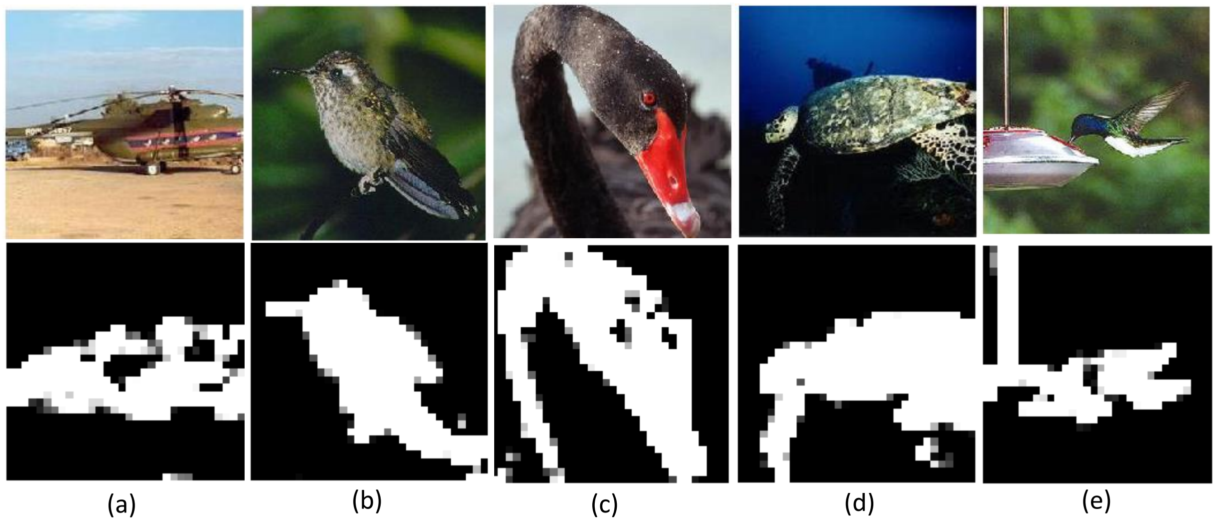


Figure 4.4 The sparse masks generated by NICE on Caltech256 images. The predictions are correct to (a,b,c,d) and incorrect to (e). Even though the prediction is incorrect, the sparse mask (e) provides an intuitive explanation why the discriminator predicts an image of "humming bird" as "bread maker".

Similarly, we train the NICE pipeline on the Caltech256 dataset to explain the predictions of the ResNet18 classifier we pretrained above. Fig. 4.4 demonstrates the sparse masks produced by NICE for different images in Caltech256. As we can see, the generated explanations are very concise and coherent, i.e., the sparse masks are mainly concentrated on the object regions, which align very well with our reasoning on these images. Additionally, the generated sparse masks also provide intuitive explanations when the classifier makes mistakes. For example, as shown in Fig. 4.4(e), the classifier incorrectly predicts an image of "humming bird" as "bread maker". The corresponding sparse explanation highlights the influential regions contributing the most to the classifier's prediction. Clearly, the classifier utilizes both the regions of the humming bird and the bird-feeder for the prediction, and the combination of the two regions confuses the classifier and leads to the incorrect classification. Such an explanation is very useful for system diagnosis: it uncovers the vulnerabilities and flaws of the classifier, and can help to improve the performance of the system.

Fig. 4.5 illustrates the comparison of NICE with Saliency Map Simonyan et al. (2013), RTIS Dabkowski & Gal (2017) and CAM Zhou et al. (2016) on the Caltech256 images. As we can see, our algorithm highlights the whole body of object as the explanation while Saliency Map typically identify edges or scattered pixels as the explanation. RTIS and CAM can identify coherent salient regions of an image, however, those regions are overly smooth and cover large background regions. Moreover, the saliency maps generated by RTIS usually have some black grids in the highlighted parts, which are caused by the upsampling step in the mask generator. Apparently, our explanations are more concise and coherent than those of the competing methods, and can preserve semantic contents of the images with a high accuracy. The superior performance of NICE on identifying semantic regions plays a critical role in semantic image compression as we will demonstrate later.

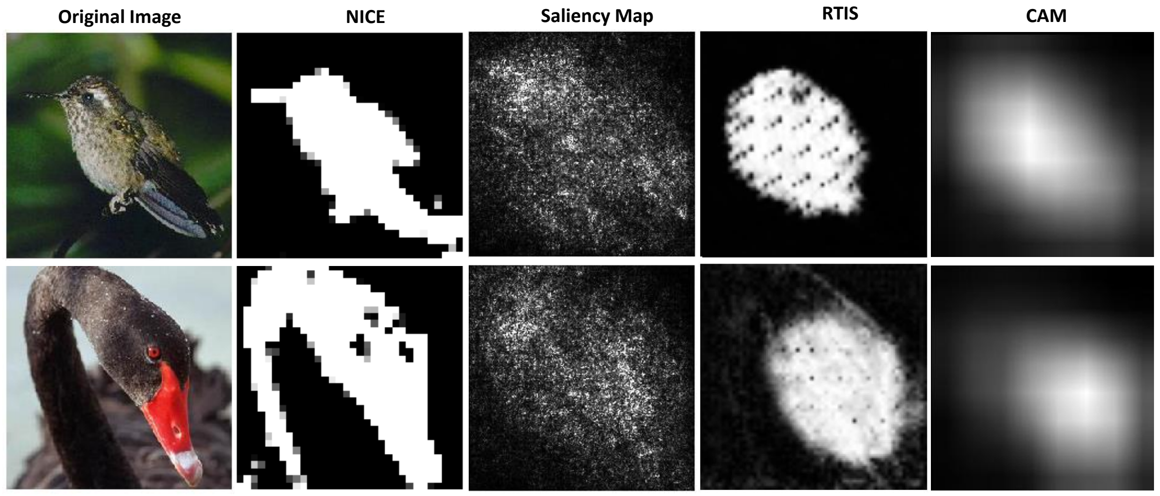


Figure 4.5 The sparse masks generated by NICE, Saliency Map Simonyan et al. (2013), RTIS Dabkowski & Gal (2017) and CAM Zhou et al. (2016) on the Caltech256 dataset. NICE highlights the whole body of object as the explanation instead of edges or scattered pixels as identified by Saliency Map, or overly-smooth regions as identified by RTIS and CAM.

To evaluate NICE's performance of identifying important pixels from an image, Fig. 4.6 demonstrates the evolution of classification accuracies on the Caltech256 test dataset when different percentages of pixels are filled with random values sampled uniformly from $[0, 255]$ (a.k.a. post-hoc classification evaluation). We compare three different strategies of selecting pixels for random value imputation: (1) Top-K% pixels sorted descending by $\log \alpha_j, \forall j \in \{1, 2, \cdots, P\}$, (2) Bottom-K% pixels sorted descending by $\log \alpha_j$, and (3) uniformly random K% pixels. Similarly, the same post-hoc classification evaluation is performed with Saliency Map, RTIS and CAM. As we can see from Fig. 4.6, NICE identifies important pixels from images as randomizing their Top-K% values incurs a dramatic accuracy loss compared to random pixel selection or Bottom-K% pixel selection. The results of Saliency Map, RTIS and

CAM show insignificant accuracy loss when randomizing their Top-K% pixels, demonstrating the superior performance of NICE on identifying salient regions.

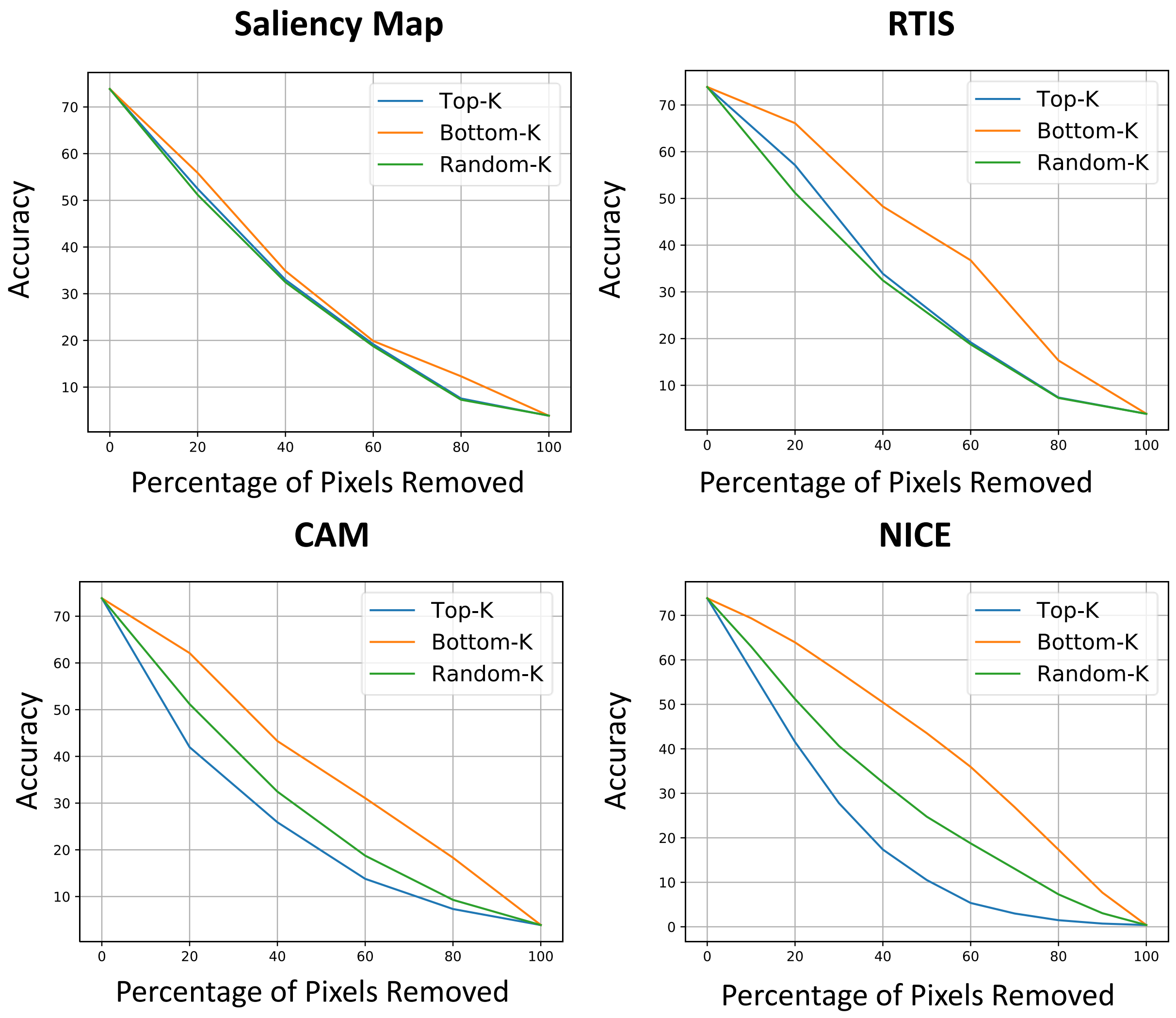


Figure 4.6 The evolution of classification accuracies on the Caltech256 test dataset when different percentages of pixels are filled with random values.

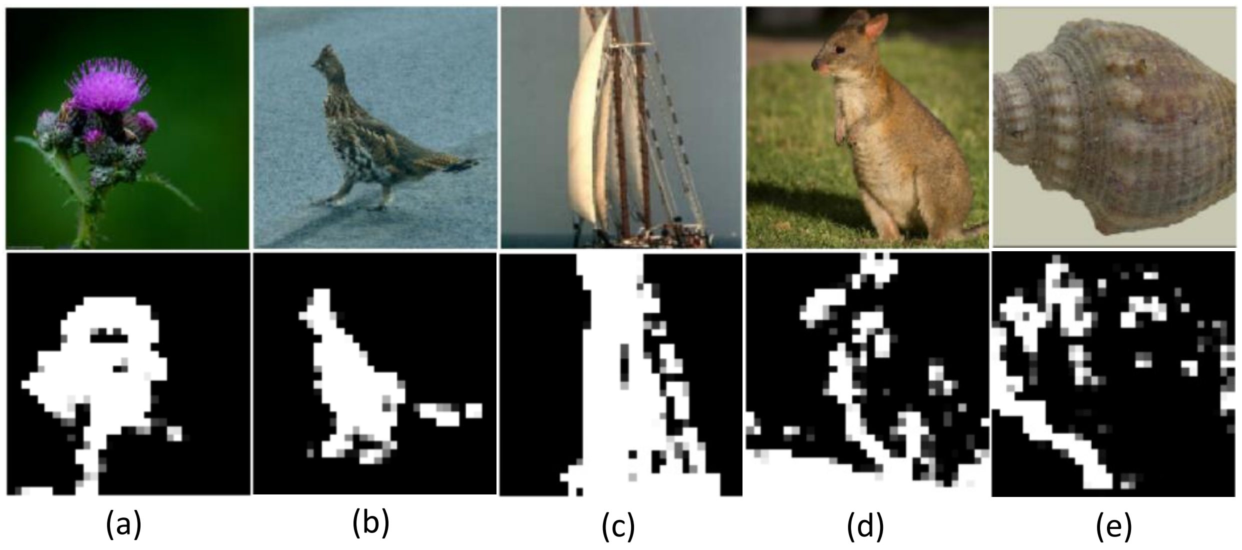


Figure 4.7 Sample ImageNet images and their sparse masks generated by the generator trained on Caltech256. While the ground truth labels of (a, b, c) are included in Caltech256, the ground truth labels of (d, e) are not in Caltech256. NICE is able to generate accurate sparse masks for images in (a, b, c). But when the classes are not in Caltach256 the masks are not very accurate as shown in (d, e).

#### *4.4.2.4 Transferability of Sparse Mask Generator*

The experiments above demonstrate the superiority of the sparse mask generator in generating concise and coherent explanations to target classifier's predictions. This has been verified in the case when the generator is applied to the test images from the same dataset. Since our generator is inductive, it would be interesting to test if a generator trained from one dataset could be applied to images from other datasets, which have similar statistics but yet have some mismatch, i.e., the transferability of generator.

To measure the transferability of generator, we apply the generator trained on Caltech256 to the ImageNet images Deng et al. (2009). Although both Caltech256 and ImageNet contain high-resolution RGB images, ImageNet contains 1000 classes which is 4 times of Caltech256's and the ImageNet images tend to be more complex than those in Caltech256. To feed the ImageNet images to the generator trained on Caltech256, we resize the ImageNet images to $256 \times 256$. Fig. 4.7 illustrates some sample ImageNet images and their sparse masks. Images (a, b, c) are from the classes that included in Caltech256, while images (d, e) are from the classes that are not in Caltech256. It shows that for the classes that overlap with Caltech256, the generator can generate sparse masks that align well with the object in the images, while for the images that are from non-overlapping classes the masks are not very accurate, indicating the transferability of generator is class dependent.

### *4.4.3 Semantic Image compression*

Finally, we evaluate the semantic image compression performance of NICE on the Caltech256 images. As a comparison, we also use the salient regions generated by Saliency Map, RTIS and CAM for semantic image compression. In this task, two approaches can be used to train the NICE pipeline: (1) **Discriminator-fixed**: given a pretrained discriminator, we freeze its parameters $\boldsymbol{\theta}_d$ in the pipeline and only update the parameters of generator $\boldsymbol{\theta}_g$ by optimizing the overall loss (4.11). In this case, the mask generator is trained to generate sparse explanation to the original discriminator. (2) **Discriminator-finetuned**: similar to discriminator-fixed except that the top few layers of the discriminator $\boldsymbol{\theta}_d$ are finetuned. In this case, the discriminator can adjust its parameters to improve its predictions on the mixed-resolution images, and thus higher accuracy and compression rate are expected. Note that due to their specific training methodologies, Saliency Map, RTIS and CAM do not have the flexibility of finetuning their discriminators, limiting their applications to semantic

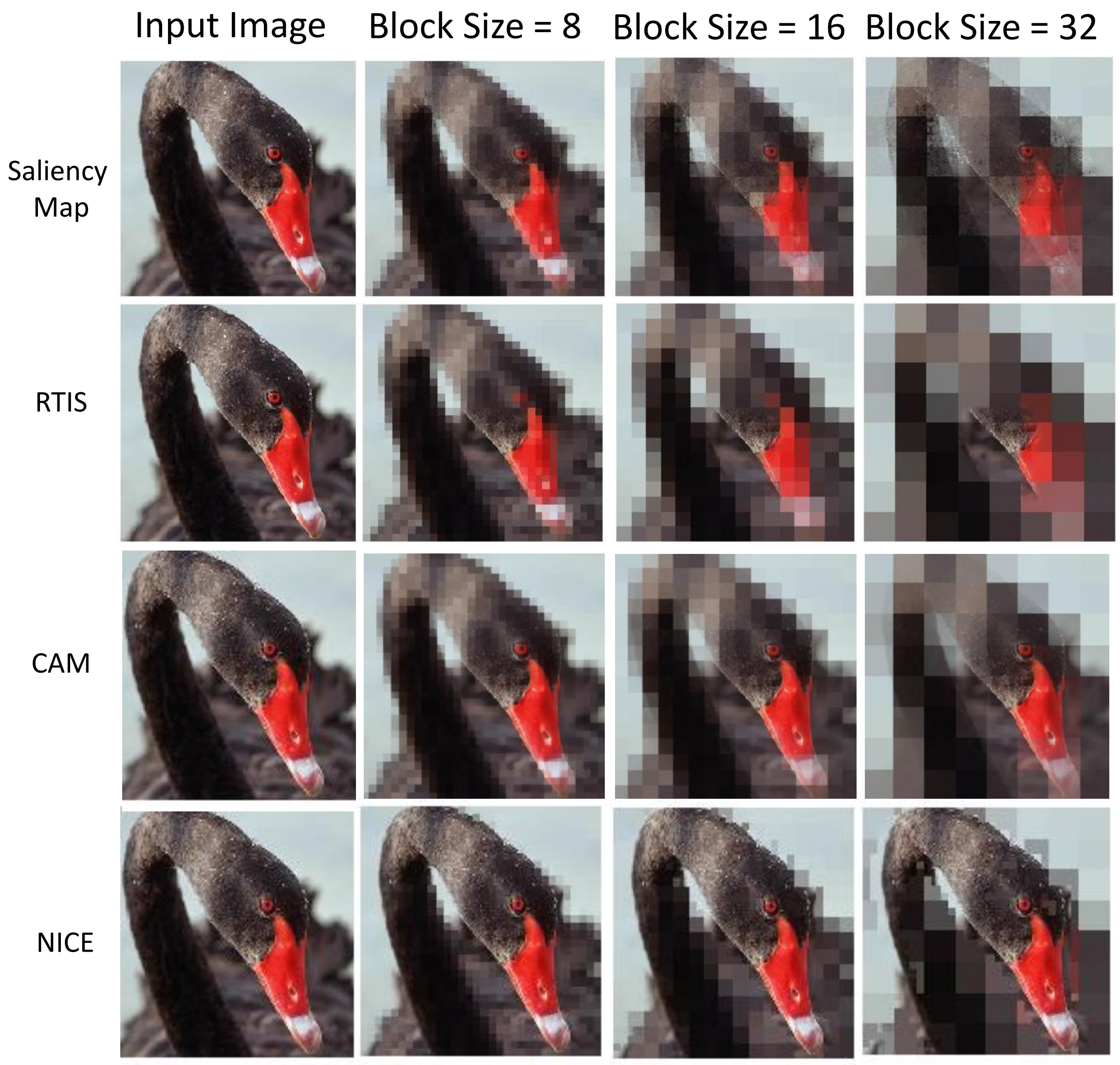


Figure 4.8 The mixed-resolution images generated by NICE, Saliency Map, RTIS and CAM with different block size $b$s.

compression tasks.

As a start, we use the sparse masks generated by NICE, Saliency Map, RTIS and CAM to produce a set of mixed-resolution images via (4.13) for visualization. Fig. 4.8 illustrates some example mixed-resolution images generated with different algorithms and block size

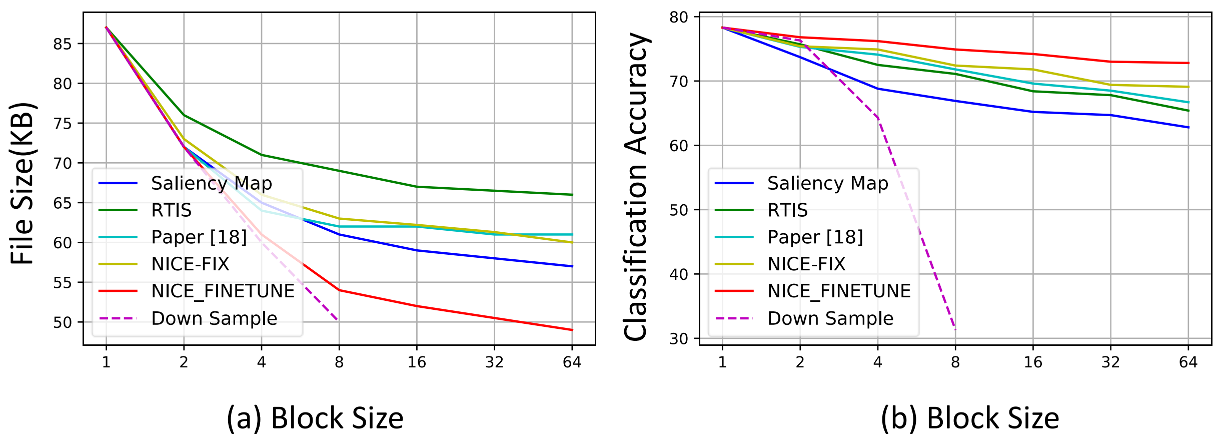


Figure 4.9 The evolution of (a) average file size of the PNG compressed image and (b) classification accuracy as a function of block size $b$ of NICE-fixed, NICE-finetuned, Saliency Map, RTIS, CAM (paper Prakash et al. (2017)) and down sampling.

$b$s. As we can see, NICE generated mixed-resolution images are clearly better than those from other algorithms. Thanks to the high accuracy of NICE on identifying salient regions, even when the background regions are subsampled with a block size of 32, the discriminator can still successfully classify these images. As a result, high compression rate and high classification accuracy can be achieved simultaneously.

To quantitatively evaluate the trade-off between semantic compression rate and classification accuracy, we train the NICE pipeline with **Discriminator-fixed** and **Discriminator-finetuned** [4] with $b = 16$ to generate sparse masks for each Caltech256 test image. After training, we generate mixed-resolution images with a different $b$ in $\{1, 2, 4, 8, 16, 32, 64\}$. We then use PNG Sayood (2002), a standard image compression algorithm, to store the gener-

[4]For discriminator-finetuned, we set the parameters of Conv-4, Conv-5 and the FC layers of ResNet18 to be trainable and freeze all the other layers.

ated mixed-resolution images and report the file sizes [5]. We also classify the mixed-resolution images with the discriminators to report classification accuracies. As a comparison, the same procedure is applied to Saliency Map, RTIS and CAM (paper Prakash et al. (2017)) for semantic image compression. We also include a baseline that uses down sampling in our compression pipeline to demonstrate the importance of salient region detection for semantic compression. Specifically, we use down sampling to generate low resolution images regardless of salient regions of the images. When testing the accuracy on the down sampled images, we upsample them to original resolution by bilinear interpolation.

Fig. 4.9 shows the average file size of compressed images and the corresponding classification accuracy as a function of block size *b*. As we can see, when the block size increases, the file size of the compressed images decreases (higher compression rate) and the classification accuracy also decreases (lower classification accuracy), and vis-versa. The classification accuracies of NICE-finetuned are significantly higher than the other four baseline methods, meanwhile it achieves the best compression rate. When the block-size is 8, NICE-finetuned achieves a 1.6x compression rate (87KB vs. 54KB) with a small (3.35%) accuracy drop (78.30% vs. 74.95%), demonstrating the superior performance of NICE on semantic image compression.

Comparing NICE-finetuned with down sampling, the results show that down sampling hurts classification accuracy significantly because it uniformly drops pixels regardless of

[5] The reason that we choose PNG Sayood (2002) instead of JPEG Wallace (1992) for compression is because PNG is a lossless compression. Thus, the file size reduction of the mixed-resolution images can be 100% attributed to NICE, and the possible artifacts introduced by JPEG, a lossy compression, can be avoided.

| Explanation Algorithm | Saliency Map Simonyan et al. (2013) | CAM Zhou et al. (2016) | RTIS Dabkowski & Gal (2017) | NICE |
|---|---|---|---|---|
| Run Time (sec) on 1000 Images mean(std) | 13.94(0.55) | 9.87(0.48) | 1.73(0.02) | 0.59(0.01) |

Table 4.2 Inference time comparison between NICE and the baseline algorithms. The results are averaged over 100 runs.

their saliency. For example, when we down sample images with a factor of 8, it reduces the average file size significantly (slightly better than NICE), but the corresponding classification accuracy is only 30.83%, while NICE still achieves an accuracy of 74.95%. Therefore, the mixed-resolution images produced by NICE can achieve a much better balance between compression rates and final classification accuracies than down sampling.

Note that semantic image compression rate depends on the size of salient regions of an image. Given the large objects in Caltech256, the 1.6x compression rate means NICE uses 60% pixels to achieve a similar classification accuracy. To achieve even higher compression rates, other image compression algorithms Rippel & Bourdev (2017); Nakanishi et al. (2018) can be used to compress NICE generated images further since our algorithm is complementary to these compression techniques.

### *4.4.4 Inference Time Comparison*

Besides the improved explanation performance of NICE, another advantage of NICE is its superior inference speed over the competing methods. As discussed above, to generate the sparse mask to explain the decision of a target CNN, NICE only needs one forward propagation of the generator network, while backpropagation-based algorithms, like Saliency Map and CAM, requires heavy computation of backpropagation to generate the salient regions. Similar to NICE, RTIS does not need backpropagation at inference time, but it requires to compute the feature maps of intermediate layers of the target CNN as the input of the generator, which is also time consuming. To have a quantitative speed comparison, we calculate the inference times of different explanation algorithms on 1000 images from Caltech256. We run each experiment 100 times on a NVIDIA Tesla V100 GPU and report the average run-times in Table 4.2. As we can see, NICE is about 23x times faster than Saliency Map and 16.5x faster than CAM, while being 2.8x faster than non-backpropagation based RTIS.

## 4.5 Conclusion

We propose NICE, a unified end-to-end trainable framework, for neural explanation and semantic image compression. Compared to many existing explanation algorithms that heavily rely on backpropagation, the sparse masks generated by NICE are much more concise and coherent and align well with human intuitions. With the sparse masks, the proposed mixed-resolution image compression further achieves higher compression rates compared to the existing semantic compression algorithms, while retaining similar classification accuracies with the original images. We conduct a series of experiments on multiple image classification benchmarks with multiple CNN architectures and demonstrate its improved explanation quality and semantic image compression rate.

As for future work, we plan to extend the technique developed here to other domains, such as text and bioinformatics for neural explanation and summarization, where interpretable decisions are also critical for the deployment of DNNs.

# CHAPTER 5

# Sparse Deep Neural Networks on Imaging Genetics for Schizophrenia Discrimination (Joint work with TReNDS Center)

## 5.1 Introduction

Schizophrenia (SZ), as one of the most disabling psychiatric disorders with a lifetime prevalence  0.5%, casts a serious socioeconomic burden worldwide J. McGrath (2008). More than a century after Kraepelin's dichotomy was formulated, precise treatment is still not available for SZ T. Insel (2010); Insel (2014). Current diagnostic and treatment practice are largely based on symptoms whose relationships to underlying biological processes await delineation T. Insel (2010); Cuthbert & Insel (2013). This gap underlies many issues faced by the psychiatric community, including vague boundaries between defined clinical entities, and heterogeneity within individual clinical entities. As a result, symptom presentations often do not neatly fit the categorical diagnostic system, and one diagnostic label covers biologically diverse conditions. These issues challenge treatment planning, which turns out to be largely empirical Insel & Cuthbert (2015); Chen et al. (2019a). It has now been widely acknowledged that objective biological markers are needed to quantify abnormalities underlying phenotypic manifestation, which allows characterizing disorders based on a multitude of dimensions and along a spectrum of functioning, so as to improve patient stratification and inform treatment planning B. J. Casey (2013); Cuthbert (2014).

Hopes have been invested in machine learning approaches as a solution to this challenge, given the complexity of SZ that has been well established in previous studies. Patients with SZ present widespread structural and functional abnormalities across the brain, including gray matter loss in the frontal, temporal and parietal cortices and subcortical structures E. I. Ivleva (2013), reduced fractional anisotropy in 20 major white matter fasciculi S. Kelly (2018), as well as abnormal resting state functional connectivity in default mode, executive control and attention networks A. G. Garrity (2007); Woodward et al. (2011). In parallel, genome wide association studies (GWASs) of SZ lend support for a polygenic architecture, where the disease risk is attributable to many genetic variants with modest effect sizes S. Ripke (2014). These findings have boosted the efforts to model SZ in a multivariate framework, which is expected to not only delineate the relationships between individual biomarkers and disease, but also to provide a generalizable mathematical model that can be used to predict risk.

One straightforward approach is to feed voxelwise neurobiological features (e.g. gray matter density) into support vector machine (SVM). With this strategy, Nieuwenhuis et al. obtained a classification accuracy of 70% which was confirmed in independent data with a sample size of a few hundred M. Nieuwenhuis (2012). It has also been explored whether more sophisticated feature selection can be combined with classifiers to yield improved discrimination. For instance, resting state connectivity between networks extracted by independent component analysis (ICA), followed by K nearest neighbors, yielded an accuracy of 96% in a data set consisting of 28 controls and 28 patients, which were randomly partitioned to serve as training and testing samples M. R. Arbabshirani (2013). In addition, fusion of multiple modalities that may carry complementary information of the brain holds promise for further improvement. In a work by Liang et al., combining gray and white matter features resulted in an average classification accuracy of 76% in 48 controls and 54 patients with first episode SZ, in a 10-fold cross validation set up S. G. Liang (2019). In contrast to neurobiological features, genetic variables, such as single nucleotide polymorphisms (SNPs), in general suffer modest effect sizes S. Ripke (2014) and could hardly be directly trained for classification. A more commonly used feature for risk discrimination is polygenic risk score (PGRS), which reflects the cumulative risk of multiple variants, and proves to be a generalizable and promising biomarker for disease discrimination and patient stratification J. Frank (2015); E. Vassos (2017).

More recently, the advancement of deep learning methods has opened a new perspective on elucidating biological underpinnings of SZ. Deep Neural Networks (DNNs) are known to excel in handling high-dimensional data and automatically identifying high-level latent features, which promotes them as promising tools for better understanding of complex traits such as SZ. In one pioneer work, Plis et al. demonstrated the application of restricted Boltzmann machine-based deep belief network to sMRI data. An classification accuracy 90% was obtained with a 10-fold cross validation in 181 controls and 198 patients with SZ S. M. Plis (2014a). Deep discriminant autoencoder network has been proposed and applied to functional connectivity features, and yielded a leave-site-out classification accuracy of 81% in 377 controls and 357 patients of SZ L. L. Zeng (2018). A comparable leave-site-out accuracy of 80% was observed in 542 controls and 558 patients with SZ, when a multi-scale recurrent neural network was applied to time courses of fMRI data W. Z. Yan (2019).

However, these approaches do not provide importance weights of original biological features regarding their contribution to classification, making interpretation less straightforward.

As commonly implemented, DNNs are black-boxes with hundreds of layers of convolution, non-linearities, and gates, optimized solely for competitive performance. While the value of DNN may be backed up with a claimed high accuracy on benchmarks, it would be desired to be able to verify, interpret, and understand the reasoning of the system. This is particularly essential for the psychiatric community, for the purpose of deconstructing complex disorders and facilitating improved treatment. In this work we introduce a sparse DNN model which allows identifying sparse and interpretable features for SZ discrimination. The sparsity is achieved with an L0-norm regularization on the input layer of the network for feature selection. Under the L0-norm sparsity constraint, the model is trained to select the most important features while retaining the high SZ classification accuracy. We applied the sparse DNN approach on a large multi-site gray matter volume (GMV) and SNP data set for SZ discrimination. In brief, a total of 634 individuals (346 controls and 288 patients with SZ) served as the training set, which was internally partitioned for hyperparameter tuning. The resulting classification model was then evaluated for generalizability on three independent data sets (n = 635, 255 and 160, respectively). We examined the classification power of pure GMV features, as well as whether combining GMV with SNP features would benefit classification. And the performance of the proposed approach was compared with that yielded by ICA+linear SVM. Empirical experiments demonstrate that the selected voxel regions from sparse DNNs are interpretable and echo many previous neuroscience studies.

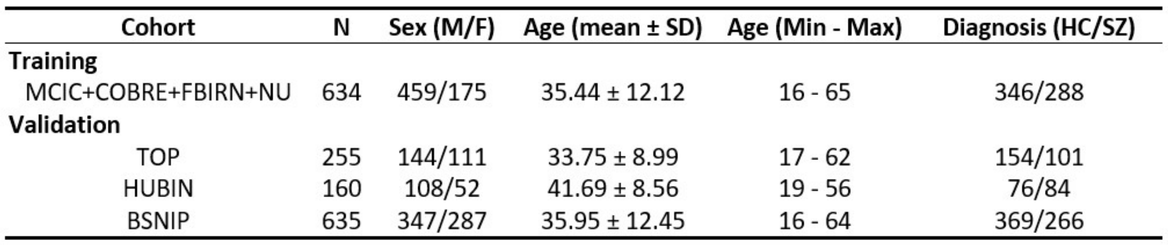
## 5.2 Materials and Methods

### *5.2.1 Participants*

A total of 1,684 individuals aggregated from multiple studies, including MCIC, COBRE, FBIRN, NU, BSNIP, TOP and HUBIN, were employed for the current study. The institutional review board at each site approved the study and all participants provided written informed consents. Each data set was shared by the individual research group according to their protocol. Diagnosis of SZ was confirmed using the Structured Clinical Interview for Diagnosis for DSM-IV or DSM-IV-TR. Table 5.1 provides the primary demographic information of individual study. The training sample consisted of 288 cases and 346 controls from MCIC, COBRE, FBIRN and NU. Meanwhile, three independent data sets, BSNIP (n = 635), TOP (n = 255) and HUBIN (n = 160) were used for validation.

### *5.2.2 Structural MRI data*

Whole-brain T1-weighted images were collected with 1.5T and 3T scanners of various models, as summarized in Table 5.1. The images of the training set were preprocessed using a standard Statistical Parametric Mapping 12 (SPM12, http://www.fil.ion.ucl.ac.uk/spm) voxel based morphometry pipeline Ashburner & Friston (2005); J. M. Segall (2009); C. N. Gupta (2015); D. Lin (2017), a unified model where image registration, bias correction and tissue classification are integrated. The resulting modulated images were resliced to 1.5mm×1.5mm×1.5mm and smoothed by 6mm full width at half-maximum Gaussian kernel. A mask (average GMV 0.2) was applied to include 429,655 voxels. We further investigated correlations between individual images and the average GMV image across all the subjects. Subjects with correlations $< 3$ were considered as outliers and excluded from subsequent analyses J. Chen (2017). Finally, voxelwise regression was conducted to eliminate the effects from age, sex, and dummy-coded site covariates C. N. Gupta (2015). While all the scanning parameters would yield 93 dummy variables in the training data, we chose to correct scanning effects by 'site' before association analysis to avoid eliminating too much information due to unknown collinearity. The validation images were preprocessed separately, using the same pipeline.



| Cohort | N | Sex (M/F) | Age (mean ± SD) | Age (Min - Max) | Diagnosis (HC/SZ) |
|---|---|---|---|---|---|
| **Training** | | | | | |
| MCIC+COBRE+FBIRN+NU | 634 | 459/175 | 35.44 ± 12.12 | 16 - 65 | 346/288 |
| **Validation** | | | | | |
| TOP | 255 | 144/111 | 33.75 ± 8.99 | 17 - 62 | 154/101 |
| HUBIN | 160 | 108/52 | 41.69 ± 8.56 | 19 - 56 | 76/84 |
| BSNIP | 635 | 347/287 | 35.95 ± 12.45 | 16 - 64 | 369/266 |

Table 1: Subject demographic information.

### *5.2.3 SNP data*

The SNP data were collected and processed as described in our previous work J. Chen (2017). DNA samples drawn from blood or saliva were genotyped with different platforms. No significant difference was observed in genotyping call rates between blood and saliva samples. A standard pre-imputation quality control (QC) J. Chen (2013) was performed using PLINK S. Purcell (2007). In the imputation, SHAPEIT was used for pre-phasin Delaneau et al. (2012), IMPUTE2 for imputation Marchini & Howie (2010), and the 1000 Genomes data as the reference panel D. M. Altshuler (2012). Only markers with INFO score ¿ 0.3

were retained. Polygenic risk scores (PGRS) for SZ were then computed using PRSice, which was a sum of genetic profiles weighted by the odds ratios reported in the PGC SZ GWAS, reflecting the cumulative risk for SZ of a set of SNPs. Specifically, the genotype data were pruned at r2 ¡ 0.1 J. Chen (2017). Then a full model PGRS was computed on 61,253 SNPs retained after pruning.

### *5.2.4 Sparse DNN*

Figure 5.1 shows the overall architecture of our method, which contains three stages. First, the GMV voxels are partitioned into a set of groups (or brain regions) with a pre-defined radius. Then a sparse DNN model is deployed for feature (brain region) selection, followed by augmenting the selected sparse regions of GMV with the SNP data for classifier retraining. In the sequel, we will introduce each of these steps in more details.

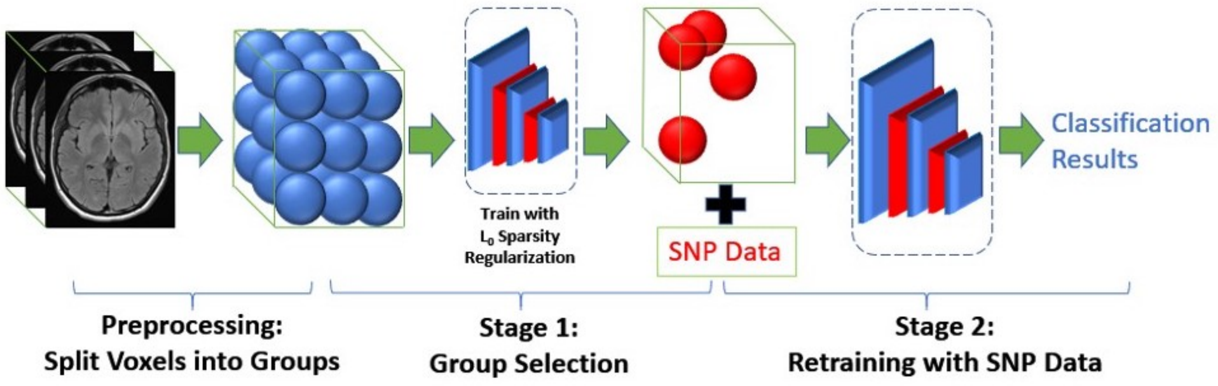


Figure 5.1 Overall architecture of our method.

Given a GMV dataset $D = \{(\boldsymbol{x}_i, y_i), i = 1, 2, \cdots, N\}$, where $\boldsymbol{x}_i$ denotes the $i$-th subject's GMV image and $y_i$ denotes the corresponding label: case or control, we train a neural network $h(\boldsymbol{x};\ \boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta}$, to fit to the dataset $D$ with the goal of achieving good generalization to unseen test data. For a GMV image $\boldsymbol{x} \in R^{M\times 1}$, we use $x^j$ to represent the $j$-th voxel of image $\boldsymbol{x}$, where $j = 1, 2, \cdots, M$ and $M = 429{,}655$ in our study.

As the number of voxels $M$ is much larger than the number of functional regions of human brain (e.g., typically around 100 as defined by various brain atlases), we first partition the brain voxels into a set of small regions, each of which is represented by a ball of a pre-defined radius $R$. We enumerate all $M$ voxels one by one: if a voxel hasn't been assigned to any region, we assign that voxel as a root to start a new region. After selecting a root voxel, we compute the Euclidean distance between the root voxel and all the unassigned voxels. All the unassigned voxels with distance smaller than $R$ are then assigned into this region. We then iterate this process over the remaining voxels to form next region until all the voxels are assigned to one of the regions. We denote the $k$-th region $G_k$. After this preprocessing step, we identify $K$ regions, from which we aim to identify important regions for SZ discrimination.

Stage 1 of our algorithm is to prune insignificant regions from $K$ pre-defined regions. We formulate our region selection algorithm by considering a regularized empirical risk minimization procedure with an $L_0$-norm regularization. Specifically, we attach a binary random variable $z^k \in \{0, 1\}$ to all the voxels in region $G_k$:

$$\widetilde{\boldsymbol{x}} = \boldsymbol{x} \odot \boldsymbol{A}\boldsymbol{z}, \qquad \boldsymbol{z} \in \{0,1\}^K, \tag{5.1}$$

where $\boldsymbol{z} \in R^{K\times 1}$ denotes a binary mask for brain image $\boldsymbol{x} \in R^{M\times 1}$, $\odot$ is an element-wise product, and $A \in R^{M\times K}$ is an affiliation matrix we construct from the preprocessing step above, with element $A_{j,k} = 1$ if voxel $x^j$ is in region $G_k$, and 0 otherwise. For all the voxels in a region $G_k$, they share the same binary mask $z^k$, and $k \in \{1, 2, \cdots, K\}$. This means if $z^k$ is 0, all the voxels in region $G_k$ will have a value of 0, otherwise the value of $x^j$ is retained. In the sequel, we will discuss our method that can learn $\boldsymbol{z}$ from training set $D$, and we wish $z^k$ takes value of 1 if $G_k$ is an important region and 0 otherwise. In other words, $\mathbf{z}$ is a measure of feature (region) importance that we wish to learn from data.

We regard $\mathbf{z}$ as the feature importance weight for the prediction of DNN model $h(x^i;\ \boldsymbol{\theta})$ and learn $\mathbf{z}$ by minimizing the following $L_0$-norm regularized loss function:

$$\begin{aligned} R(\ \boldsymbol{\theta},\ \boldsymbol{z}) &= \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\left(h\left(\boldsymbol{x}_i \odot A\boldsymbol{z};\boldsymbol{\theta}\right), y_i\right) + \lambda\|\boldsymbol{z}\|_0 \\ &= \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(h(\boldsymbol{x}_i \odot A\boldsymbol{z};\boldsymbol{\theta}), y_i) + \lambda\sum_{k=1}^{K}1_{[z^k \neq 0]}, \end{aligned} \tag{5.2}$$

where $\mathcal{L}(\bullet)$ denotes the data loss over training data $D$, such as the cross-entropy loss for classification, $\|\boldsymbol{z}\|_0$ is the $L_0$-norm that measures number of nonzero elements in $\boldsymbol{z}$, $\lambda$ is a regularization hyperparameter that balances between data loss and feature sparsity, and $1_{[c]}$is an indicator function that is 1 if the condition c is satisfied, and 0 otherwise. To optimize Eq. 5.2, however, we note that both the first term and the second term of Eq. 5.2 are not differentiable w.r.t. $\mathbf{z}$. Therefore, further approximations need to be considered.

We can approximate this optimization problem via an inequality from stochastic variational optimization Bird et al. (2018b). Specifically, given any function $\mathcal{F}(\boldsymbol{z})$ and any distribution $q(\boldsymbol{z})$, the following inequality holds

$$\underset{\boldsymbol{z}}{min}\, \mathcal{F}\left(\boldsymbol{z}\right) \leq \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z})}\left[\mathcal{F}\left(\boldsymbol{z}\right)\right], \tag{5.3}$$

i.e., the minimum of a function is upper bounded by the expectation of the function. With this result, we can derive an upper bound of Eq. 5.2 as follows.

Since $z^k, \forall k \in \{1, \cdots, K\}$ is a binary random variable, we assume $z^k$ is subject to a Bernoulli distribution with parameter $\pi^k \in [0,1]$, i.e. $z^k \sim \text{Ber}(z; \pi^k)$. Thus, we can upper bound $\min_z R(\boldsymbol{\theta}, \boldsymbol{z})$ by the expectation

$$\tilde{R}(\boldsymbol{\theta}, \boldsymbol{\pi}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\pi})}[\mathcal{L}(h(\boldsymbol{x}_i \odot A\boldsymbol{z}; \boldsymbol{\theta}), y_i)] + \lambda \sum_{k=1}^{K} \pi^k. \tag{5.4}$$

Now the second term of the Eq. 5.4 is differentiable w.r.t. the new model parameters $\boldsymbol{\pi}$. However, the first term is still problematic since the expectation over a large number of binary random variables $\mathbf{z} \in \{0,1\}^K$ is intractable, so is its gradient. To solve this problem, we adopt the hard-concrete estimator Louizos et al. (2017). Specifically, the hard-concrete gradient estimator employs a reparameterization trick to approximate the original optimization problem of Eq. 5.4 by a close surrogate loss function

$$\begin{aligned}\hat{R}(\boldsymbol{\theta}, \log\boldsymbol{\alpha}) &= \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\mathbf{u}\sim\mathcal{U}(0,1)}\left[\mathcal{L}\left(h\left(\boldsymbol{x}_i \odot g\left(Af\left(\log\boldsymbol{\alpha}, \mathbf{u}\right)\right); \boldsymbol{\theta}\right), y_i\right)\right] + \lambda \sum_{k=1}^{K} \sigma\left(\log\alpha_k - \beta\log\frac{-\gamma}{\zeta}\right) \\ &= \mathcal{L}_D\left(\boldsymbol{\theta}, \log\boldsymbol{\alpha}\right) + \lambda\mathcal{L}_C\left(\log\boldsymbol{\alpha}\right),\end{aligned} \tag{5.5}$$

with

$$f\left(\log\alpha_k\ , u_k\right) = \sigma\left(\frac{\log u_k\ - \log\left(1-u_k\right)\ + \log\alpha_k}{\beta}\right)\left(\zeta-\gamma\right) + \gamma, \tag{5.6}$$

and

$$\mathrm{g}(\cdot)\ = min(1, max(0,\ \cdot)) \tag{5.7}$$

where $\sigma(t) = 1/(1+\exp(-t))$ is the sigmoid function, $\mathcal{L}_D$ measures how well the classifier fits to training data $D$, $\mathcal{L}_C$ measures the expected number of non-zeros in $\boldsymbol{z}$, and $\beta{=}\frac{2}{3}$ , $\gamma{=}\ -0.1$ and $\zeta{=}\ 1.1$ are the typical parameters of the hard-concrete distribution. Function $g(\cdot)$ is a hard-sigmoid function that bounds the stretched concrete distribution between 0 and 1. With this reparameterization, the surrogate loss function Eq. 5.5 is differentiable w.r.t. its parameters.

After training, we learn $\log\boldsymbol{\alpha}$ from the dataset $D$. At test time, we employ the following estimator to generate a sparse mask or feature importance weight:

$$\widehat{\boldsymbol{z}} = min\left(\mathbf{1}, max\left(\mathbf{0}, \sigma\left(\frac{\log\boldsymbol{\alpha}}{\beta}\right)(\zeta-\gamma)+\gamma\right)\right), \tag{5.8}$$

which is the sample mean of $\boldsymbol{z}$ under the hard-concrete distribution $q(\boldsymbol{z}|\log\boldsymbol{\alpha})$.

After we train the sparse DNN with the $L_0$-norm regularization, we get the trained neural network parameters $\boldsymbol{\theta}$ and sparse mask $\widehat{\boldsymbol{z}} \in [0,1]^{\boldsymbol{K}}$ over all $K$ regions, with element $\hat{z}^k$ a continuous variable that represents the importance of region $G_k$. Because of the sparsity inducing property of the $L_0$-norm, many elements of learned $\widehat{\boldsymbol{z}}$ are pushed to zero, which are considered as unimportant regions and thus pruned from the model. The level of sparsity can be modulated by hyperparameter $\lambda$: the larger $\lambda$ is, the sparser regions is identified, and *vis-a-versa.*

In Stage 2 of our algorithm, we can further improve the accuracy of the classifier by finetuning the DNN with the selected $L$ regions from Stage 1 but without the $L_0$-norm regularization. To examine whether incorporating genetic features can improve the classification accuracy, we also concatenate the PGRS feature to the selected voxels as the input of the DNN classifier to finetune the classifier.

In our study, the training data consists of 634 individuals (346 controls and 288 cases), which were equally partitioned into three subsets (each containing 33% of the samples). A nested 3-fold cross validation was then implemented to identify the discriminating genetic and neurobiological features and construct a classification model for SZ. The region radius $R$ we used was 12mm and each brain image was partitioned into 1111 regions as we described above. In Stage 1 group selection and Stage 2 retraining, we used a DNN classifier with 2 fully connected layers of 200 and 16 neurons, respectively, and the Rectified Linear Unit (ReLU) activation function. We performed grid search to find the best hyperparameters for our sparse DNN model. In Stage 1 group selection, we used the SGD optimizer with learning rates of 0.005 and 1 for model parameter $\boldsymbol{\theta}$and $\log\boldsymbol{\alpha}$, respectively. In Stage 2 retraining classifier, we used the Adam optimizer with learning rate of 0.005 for $\boldsymbol{\theta}$ and a weight decay of 1e-5. After the sparse DNN was trained on the GMV features, the regions with nonzero $\hat{z}$s were considered as important regions for the SZ classification. The selected regions across 3-fold cross validation were highlighted for model interpretation. In particular, we tuned hyperparameter $\lambda$ to compare the classification performances with different levels of sparsity, i.e. with 5 or 20 regions as predictors. In Stage 2 retraining, the selected voxel regions were fed into the classifier and may concatenate the PGRS feature to improve the classification accuracy. The model established in the training data was further evaluated on three external data sets: BSNIP, TOP and HUBIN.

### *5.2.5 ICA+linear SVM*

To compare with sparse DNN, we also conducted classification using linear SVM with components extracted by ICA as input. ICA decomposes data into a linear combination of underlying components among which independence is maximized Bell & Sejnowski (1995); Amari (1998). When applied to sMRI data, ICA essentially identifies voxels with covarying gray matter patterns across samples, and cluster these voxels into one component L. Xu (2008). It has been well established that ICA is a trustable model in the neuroimaging field, yielding meaningful and generalizable brain networks which may not be captured by anatomical atlas V. D. Calhoun (2001); C. F. Beckman (2005); C. N. Gupta (2015). In the current work, following the training and testing of the sparse DNN, we applied ICA on the GMV data of 67% of the training samples. The resulting components were then fed into linear SVM to obtain a classification model. This model was then assessed on the remaining 33% of the training samples for accuracy. While the number of ICA components was a hyperparameter to be tuned, we repeated the above process with different component numbers. The optimal model was then determined to be the one yielding the highest accuracy, which awaited further validation in the three independent data sets. Echoing the sparse DNN experiments, we also investigated whether having more GMV components as predictors would affect the performance of classification. When genetic feature was further incorporated, PGRS was treated as an additional predictor, which was sent into linear SVM along with the GMV components. Note that genetic data were available only for TOP and HUBIN, such that only these two data sets were examined for imaging genetic based classification.

## 5.3 Results

The performance was summarized in Table 5.2. When only GMV features were used for classification, the ICA+SVM approach achieved the highest accuracy with 20 components in the training samples. In parallel, the performance of DNN also started to saturate around a sparsity level of 20 regions. It can be seen that for both ICA and DNN approaches, lower error rates were achieved when 20 rather than only 5 brain regions/components served as predictors. When less brain regions were used to train the model, the mean error rate across three independent data sets was  35% for both ICA and DNN, though in specific data sets discrepancies could be noted. When the classification model was allowed to incorporate more brain regions/components, the mean error rate across three data sets decreased to 31.03% for DNN models and 31.86% for ICA models. Specifically, the error rates were comparable between ICA and DNN in HUBIN and BSNIP, while the error rate improved by 3.66% in

TOP when DNN was used. When PGRS was further incorporated for classification, the DNN approach yielded consistent improvement in accuracy across all the data sets, either with 5 or 20 regions as predictors, where the decrease in error rate ranged from 1.41% to 3.94%. In contrast, with ICA components were combined with PGRS for classification, the error rate did not always decrease. The lowest error rate (27.75%) was observed in HUBIN, when the DNN classification model used 20 brain regions plus the PGRS. The brain regions identified by DNN are summarized in Tables 5.2 (5 regions) and 5.3(20 regions), and Figures 5.2 and 5.3 show the spatial maps of individual regions. Note that only the regions identified in all three folds are listed. When 5 regions were to be selected as predictors, the three folds consistently identified the same 5 regions, spanning inferior, middle and superior frontal gyrus, superior temporal gyrus, as well as cerebellum. When 20 regions were to be selected, variations were noted across folds, such that 13 brain regions were consistently identified. Compared to those covered by 5 regions, cuneus, precuneus, medial frontal gyrus, and paracentral lobule were also determined to be informative and included for classification. The importance weights yielded by the interpretable DNN model were overall highly consistent with those inferred from the original features, such that a positive/negative DNN weight indicated that the region showed higher/lower values in controls compared to patients with SZ. The only exception was region 27 which was identified in the 20-region model.

| | sMRI | | | sMRI + PGRS | |
|---|---|---|---|---|---|
| | TOP (225) | HUBIN (160) | BSNIP (635) | TOP (255) | HUBIN (160) |
| DNN (5 regions) | | | | | |
| EER1 | 35.69 | 33.08 | 34.49 | 32.94 | 28.13 |
| EER2 | 34.90 | 36.25 | 33.60 | 33.33 | 33.13 |
| EER3 | 34.90 | 36.25 | 36.80 | 32.55 | 32.50 |
| EER mean | 35.16 | 35.19 | 34.96 | 32.94 | 31.25 |
| ICA + SVM (5 ICs) | | | | | |
| EER1 | 36.85 | 31.88 | 37.17 | 30.20 | 35.00 |
| EER2 | 37.25 | 34.38 | 37.32 | 30.59 | 35.63 |
| EER3 | 34.90 | 32.50 | 36.85 | 29.80 | 35.63 |
| EER mean | 36.34 | 32.92 | 37.11 | **30.20** | 35.42 |
| DNN (20 regions) | | | | | |
| EER1 | 30.59 | 28.13 | 31.16 | 30.65 | 26.27 |
| EER2 | 30.98 | 32.50 | 32.91 | 27.75 | 27.25 |
| EER3 | 33.33 | 28.75 | 31.02 | 32.26 | 28.24 |
| EER mean | 31.63 | 27.79 | 31.69 | **30.22** | **27.75** |
| ICA + SVM (20 ICs) | | | | | |
| EER1 | 33.33 | 27.50 | 30.87 | 32.94 | 29.38 |
| EER2 | 39.22 | 31.25 | 31.02 | 35.29 | 33.75 |
| EER3 | 33.33 | 28.75 | 31.50 | 3.98 | 30.00 |
| EER mean | 35.29 | 29.17 | 31.13 | 33.07 | 31.04 |

Table 5.1 Summary of classification error rates.

| Region | Area | Brodmann Area | Volume (cc) | MNI (x,y,z) |
|---|---|---|---|---|
| DL87 | Uvula | * | 0.7/0.0 | (-18, -81, -33)/(0, 0, 0) |
| DL382 | Inferior Frontal Gyrus | 47 | 1.9/0.0 | (-54, 30, 0)/(0, 0, 0) |
| DL493 | Superior Frontal Gyrus | 10 | 0.0/1/2 | (0, 0, 0)/(27, 60, 9) |
| | Middle Frontal Gyrus | 10 | 0.0/0.9 | (0, 0, 0)/(34.5, 57, 9) |
| DL555 | Superior Temporal Gyrus | 13, 22, 41 | 1.0/0.0 | (-45, -30, 15)/(0, 0, 0,) |
| DL775 | Inferior Frontal Gyrus | 9 | 0.0/1.0 | (0, 0, 0)/(57, 12, 36) |

Table 5.2 Summary of the 5 important brain regions identified by DNN.

## 5.4 Discussion

An interpretable sparse DNN approach was proposed for application to medical data analysis and its capability was examined on a large and heterogeneous SZ data set. The results confirmed that the proposed approach yielded reasonable classification accuracies, could identify meaningful brain regions, and the interpretation of these brain regions was consistent with that directly inferred from original features. Particularly, the proposed model appeared to

| Region | Area | Brodmann Area | Volume (cc) | MNI (x,y,z) |
|---|---|---|---|---|
| DL2 | Inferior Semi-Lunar Lobule | * | 0.1/0.0 | (-7.5, -60, -54)/(0, 0, 0) |
| DL27 | Cerebellar Tonsil | * | 1.4/0.0 | (-15, -55.5, -43.5)/(0, 0, 0) |
| DL45 | Cerebellar Tonsil | * | 0.7/0.0 | (-12, -55.5, -40.5)/(0, 0, 0) |
| DL172 | Superior Temporal Gyrus | 38 | 0.0/1.0 | (0, 0, 0)/(48, 22.5, -19.5) |
| DL260 | Middle Frontal Gyrus | 11 | 0.9/0.0 | (-37.5, 40.5, -10.5)/(0, 0, 0) |
| DL509 | Inferior Frontal Gyrus | 13, 47 | 1.3/0.0 | (-42, 25.5, 10.5)/(0, 0, 0) |
| DL599 | Cuneus | 18, 19 | 0.0/1.0 | (0, 0, 0)/(18, -88.5, 19.5) |
| DL691 | Middle Frontal Gyrus | 10, 46 | 1.2/0.0 | (-34.5, 46.5, 27)/(0, 0, 0) |
| DL805 | Middle Frontal Gyrus | 9 | 2.0/0.0 | (-45, 28.5, 39)/(0, 0, 0) |
| DL846 | Precuneus | 7, 19 | 0.0/1.0 | (0, 0, 0)/(30, -66, 42) |
| DL1008 | Medial Frontal Gyrus | 6 | 0.0/1.3 | (0, 0, 0)/(7.5, -4.5, 63) |
| DL1017 | Paracentral Lobule | 4, 5, 6 | 0.2/1.7 | (-1.5, -40.5, 61.5)/(4.5, -37.5, 64.5) |
| DL1039 | Middle Frontal Gyrus | 6 | 1.3/0.0 | (-21, 9, 67.5)/(0, 0, 0) |

Table 5.3 Summary of the 13 important brain regions identified by DNN.

more effectively fuse imaging and genetic features for classification compared to ICA+SVM, holding potential for data fusion.

The DNN models reliably generalized to data collected at different sites, with reasonable classification accuracies compared to ICA+SVM. The generalizability indicates that the classification models are not vulnerable to scanning protocol, recruiting criteria, ethnicity influence, medication history, etc. Regarding performance, both DNN and ICA+SVM approaches presented higher accuracies when more brain regions/components served as predictors, with error rates being 31.03% and 31.86%, respectively. The ICA+SVM performance was comparable to those reported by Cai et al., where the authors conducted a comprehensive study on generalizability of machine learning for SZ classification using ICA-extracted resting-state fMRI features, and achieved an external accuracy of 70% with transfer learning procedures X. L. Cai (2020). Notably, Cai et al. emphasized the importance of assessing models across sites and studies, while results based on a single study need to be interpreted cautiously. This might explain why our classification accuracy based on a large and multi-study cohort is lower than some previous studies with smaller sample sizes or single-study cohort S. M. Plis (2014b), indicating complex heterogeneity of patients with SZ. Increasing sample size of the training data and incorporating other data modalities promise further improvement.

The proposed approach highlights a sparsity constraint, which allows trade-off between explained variance and interpretability of identified features. In general, a low level of sparsity allows more features to be admitted into the classification model, which however results in more variance across samples. As shown in the current work, when a higher level of sparsity was enforced, the same 5 regions were identified across 3 folds. In contrast, with a lower sparsity, 13 out of 20 regions were consistently identified, although the latter explained more variance and yielded higher classification accuracies. It should be pointed out that, increasing the predictors from 5 to 20 regions resulted in a decrease of  4% in error rate, which was indeed not profound. In other words, although GMV abnormalities are widely present in SZ, the identified five regions comprise primary and unique SZ-related disruptions in brain structure. The samples missed in the classification, or missing variance, likely call for a larger training data set to allow better capturing heterogeneity, as well as for information from other data modalities, rather than simply adding more features from the sMRI modality.

SZ is a complex disorder, where genetic and environmental factors interact with each other to affect brain structure and function which ultimately manifest into clinical symptoms. With so many factors involved in the pathology of SZ, it is expected that multiple data modalities need to be integrated to fully characterize the disorder. This also applies to classification, which should capitalize on data fusion to extract complementary information from different modalities. The proposed model holds promise for this purpose. In all the tested scenarios, the DNN approach effectively fused GMV and PGRS features to yield improved classification accuracies, indicating that the model reliably extracted SZ-related variance in PGRS that was not captured by GMV. In contrast, no consistent improvement was noted for ICA+SVM, where PGRS and brain components were directly fed into linear SVM for classification training. The results appeared to lend support that nonlinear models excel in delineating the relationships across different modalities in hidden layers and robustly capturing complementary variance that is related to the trait of interest.

The brain regions identified by DNN are overall well documented in SZ studies. With high sparsity, 5 brain regions were consistently identified across 3 folds, as listed in Table 5.2, highlighting frontal gyrus, superior temporal gyrus, and cerebellum. All the five regions presented positive weights, indicating higher GMV in controls compared to patients, which was consistent with the results of two-sample t-tests on original GMV features. SZ-related gray matter reduction has been widely observed in temporal and frontal regions. A longitudinal study by Thompson et al. revealed accelerated gray matter loss in early-onset SZ, with earliest deficits found in parietal regions and progressing anteriorly into temporal and prefrontal regions over 5 years, the latter related to frontal executive impairments P. M. Thompson (2001). The identified frontal and temporal brain regions have also been identified for SZ-related reduction in a comprehensive study on gray matter volume in psychosis using the BSNIP cohort E. I. Ivleva (2013). The role of cerebellum in SZ has been revised in recent years, where accumulating evidence suggests that cerebellum is also involved in higher cognitive functions and cerebellar abnormalities are noted in SZ Andreasen & Pierson (2008). Gray matter loss around the identified cerebellar region has also been reported previously T. F. D. Farrow (2005).

With low sparsity, 13 brain regions were consistently identified by DNN across 3 folds, as listed in Table 5.3. In addition to frontal, temporal and cerebellar regions discussed above, parietal regions including cuneus, precuneus and paracentral lobule were highlighted. As implicated in Thompson et al, while temporal and prefrontal gray matter loss were characteristic of adult SZ, parietal regions were noted for earliest gray matter loss which was faster in younger patients with SZ P. M. Thompson (2001). The identified parietal regions also echoed the BSNIP findings to show higher GMV in controls compared to patients E. I. Ivleva (2013). Overall, it is reasonable that DNN prioritized to select temporal and frontal regions for classification when high sparsity was enforced, which aligns with the notion that gray matter loss in these regions characterizes adult SZ. In the meantime, when a lower sparsity was enforced, parietal abnormalities were the first priority to be added as additional predictors which offered complementary variance. Among the 13 regions, region 27 was the only feature whose DNN weights did not coincide with the inference drawn from original GMV features. It was noted that the voxels in region 27 showed modest case-control differences compared to voxels in other identified brain regions. We suspect the selection of region 27 by DNN might be driven by some hidden properties rather than group differences, which explains the inconsistency in interpretation between DNN and two-sample t-tests. One limitation of our algorithm is that we assume the brain regions to be spherical, which we obtained by measuring the Euclidean distance. This may not align with the optimal partition. And we did not extensively investigate how the radius of brain regions would affect the performance. In the future, we plan to test whether defining regions based on a brain atlas (such as Yeo atlas B. T. T. Yeo (2011)) would benefit the model training. Besides, likely due to the limited sample size, the DNN performance saturated at 2 hidden layers. It remains a question how the performance would scale with increasing sample size. This awaits investigation when more data become available. Furthermore, while the DNN approach holds promise for data fusion, its capability of integrating multiple high-dimensional imaging modalities was not examined in the current work, given that incorporating another modality would further reduce the sample size. This will also be part of our future work. In summary, to the best of our knowledge, this is the first study of DNN application to sMRI and genetic features for SZ discrimination with generalizability assessed in a large and multi-study cohort. An interpretable sparse DNN approach was first proposed to allow identifying, refining and interpreting features used in classification. The results indicate that the new approach yielded reasonable classification performances, highly interpretable classification features, as well as potential for data fusion. Collectively, the current work validates the application of the proposed approach to SZ classification, and promises extended utility on other data modalities (e.g. functional and diffusion images) and traits (e.g. continuous scores).

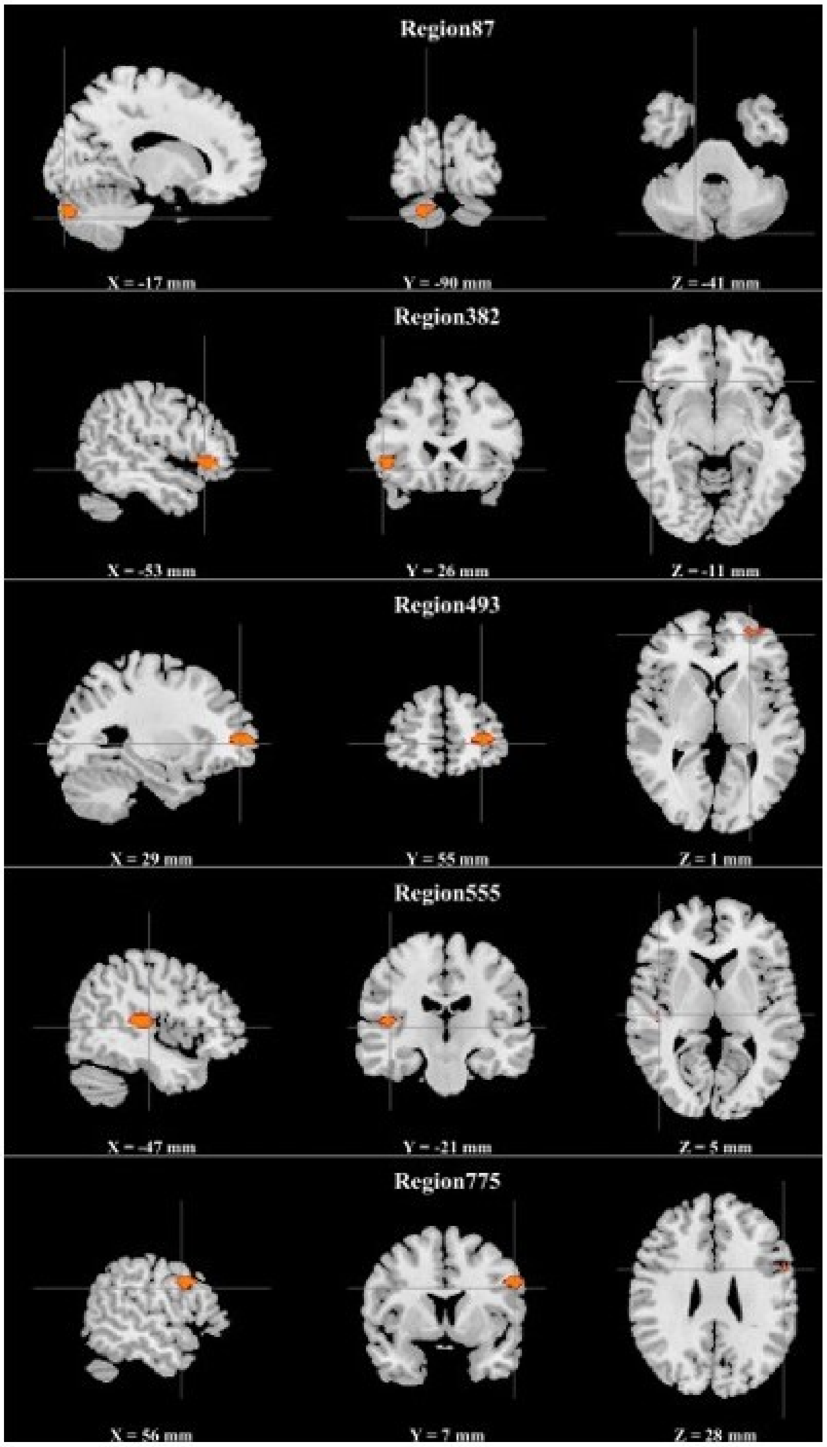


Figure 5.2 Spatial maps of the five schizophrenia-discriminating regions identified by sparse DNN.

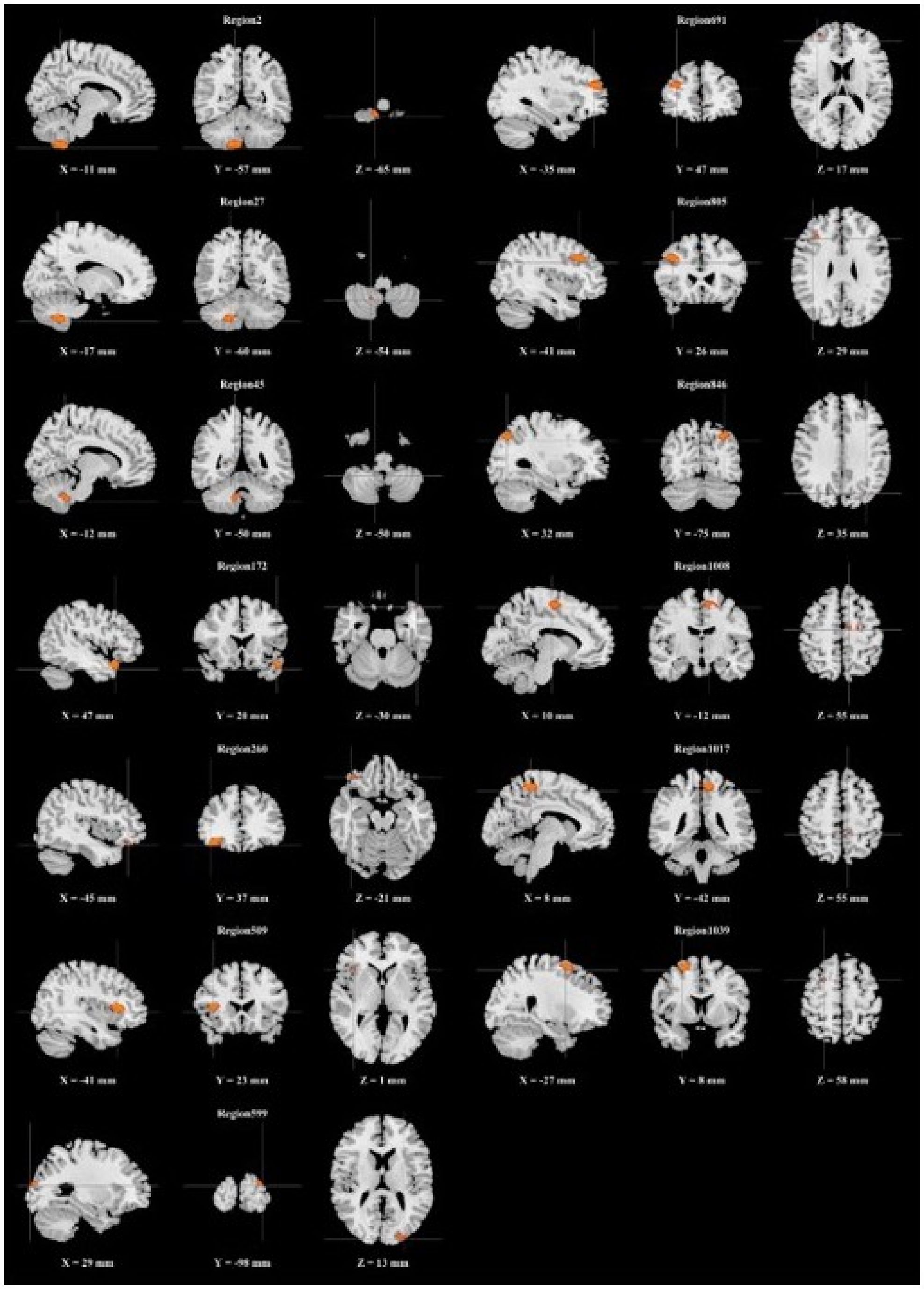


Figure 5.3 Spatial maps of the 13 schizophrenia-discriminating regions identified by sparse DNN.

# CHAPTER 6

# Proximal Policy Optimization with Parameter-wise Smooth Policy

## 6.1 Introduction

Combining the power of classic reinforcement learning algorithms and modern deep neural network techniques, deep reinforcement learning (DRL) is capable of training agents for complex tasks with inputs of high dimensional observations, such as pixels Mnih et al. (2013). It has proven that DRL is successful across a wide range of problems, including game playing Mnih et al. (2013); Silver et al. (2016, 2017); Vinyals et al. (2019), robot control Tai et al. (2018, 2017); Zhelo et al. (2018); Hwangbo et al. (2019), natural language processing Hudson & Manning (2018); Wang et al. (2018); Wu et al. (2018), autonomous driving Talpaert et al. (2019); Milz et al. (2018); Li et al. (2020), and recommendation systems Zheng et al. (2018); Chen et al. (2019c), etc.

In recent years, there has been a considerable advance in the development of DRL, which is verified in multiple simulation environments Bellemare et al. (2013); Todorov et al. (2012). However, a large gap still exists for DRL to succeed in the real world because of several drawbacks of DRL algorithms. For example, deep neural networks often suffer from overfitting due to the large parameter space fitted with a small number of sampled observations and actions; training algorithms are often unstable with unpredictable behaviors by the learning policy due to lack of robustness and smoothness.

Among DRL algorithms, policy gradient (PG) Mnih et al. (2016) is a popular and effective model-free on-policy algorithm. However, the first-order optimizer used by PG tends to be

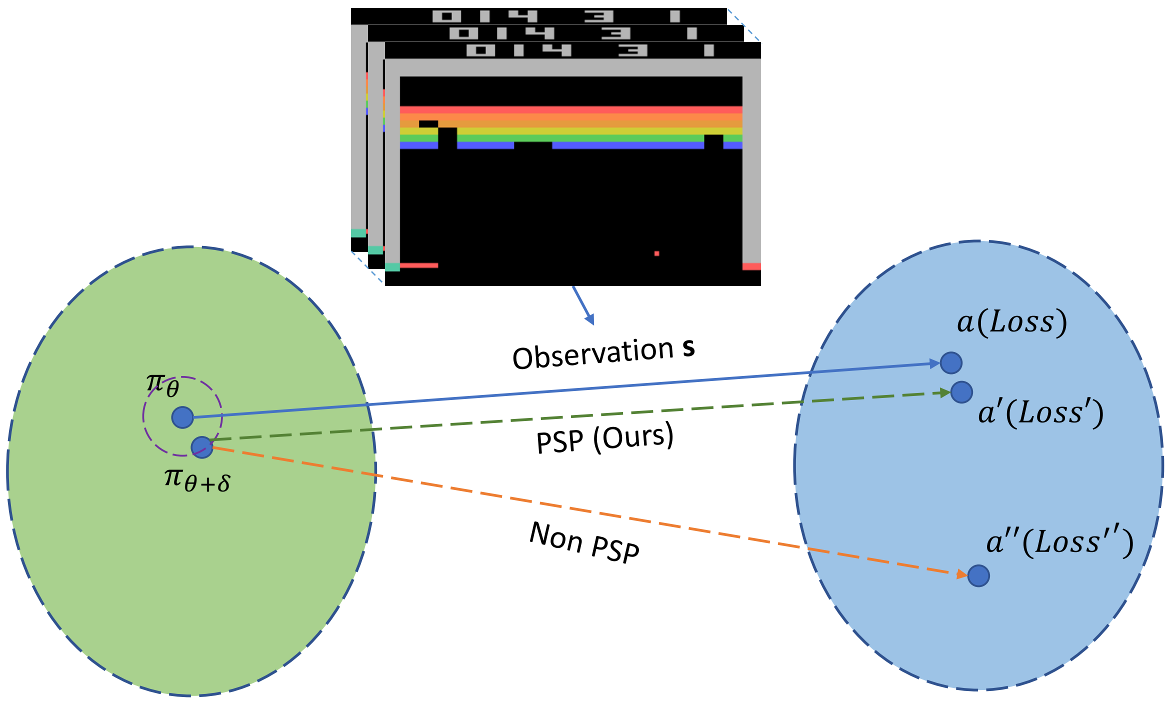


Figure 6.1 Our proposed method learns a parameter-wise smooth policy, which can improve the stability of and promote sample efficiency of the PPO algorithm. Specifically, the policy trained with our proposed PSP-O or PSP-L method gives a more robust decision output when the policy parameters are perturbed by random or adversarial noise. As we can see from the figure, the policy decision $a'$ (loss value $Loss'$) from PSP-O (PSP-L) trained policy under perturbation is close to the $a$ ($Loss$) from the original policy, while $a''$ (loss value $Loss''$) from Non-PSP trained policy under perturbation is far away from $a$ ($Loss$). PSP has two variants: PSP-O puts the regularization on policy decision space, and PSP-L regularizes the loss space.

overfitting and makes terrible updates to the model during training. As a result, it is difficult for PG to estimate the correct step size to update the policy and balance the trade-off between learning stability and learning speed. To mitigate this issue, TRPO Schulman et al. (2015) and PPO Schulman et al. (2017) placed a constraint on the distance between the new and old policies. Specifically, PPO applies a surrogate objective to regularize large policy updates, such that in each iteration, the new policy is limited to a close neighborhood around the old policy.

Instead of imposing a constraint on the distance between old and new policies during optimization like PPO, can we regularize the policy smoothness (distance between the current policy and its neighbors) to improve the optimization? Recently, researchers have worked on improving the smoothness of deep neural networks to solve the above issues of RL algorithms Raffin et al. (2022); Shen et al. (2020). However, existing works Raffin et al. (2022); Shen et al. (2020) applying smoothness constraints to RL algorithms mainly focus on observation-wise policy smoothness, which encourages the output of the policy to be close when perturbing the observations. Notably, recent researches Foret et al. (2020); Chen et al. (2021); Bahri et al. (2021) show that improving parameter-wise smoothness of neural networks can improve the performance of deep learning models for CV Foret et al. (2020); Chen et al. (2021) and NLP Bahri et al. (2021) tasks. Therefore, it is a significant limitation that no current works apply the parameter-wise smooth policies to RL tasks.

To address the above issues, we propose parameter-wise smooth policy regularization in this paper to constrain the distance between the current policy and its neighbors. With this regularization during training, the distance between the new policy in the worst case and the old policy is small in each iteration. For this reason, the proposed regularization can improve the training stability of the PPO algorithm. Concretely, we make the following contributions:

- We propose to improve the PPO algorithm by improving the policy smoothness (parameter-wise robustness). This is the first time parameter-wise smooth policies are applied to

RL tasks.

- We introduce two variants of the parameter-wise smooth policy (PSP) algorithm: PSP-O and PSP-L. PSP-O places constrain on the policy output, and PSP-L imposes regularization on the loss value. Both methods encourage the policy to not change much when injecting small perturbations into the parameters.

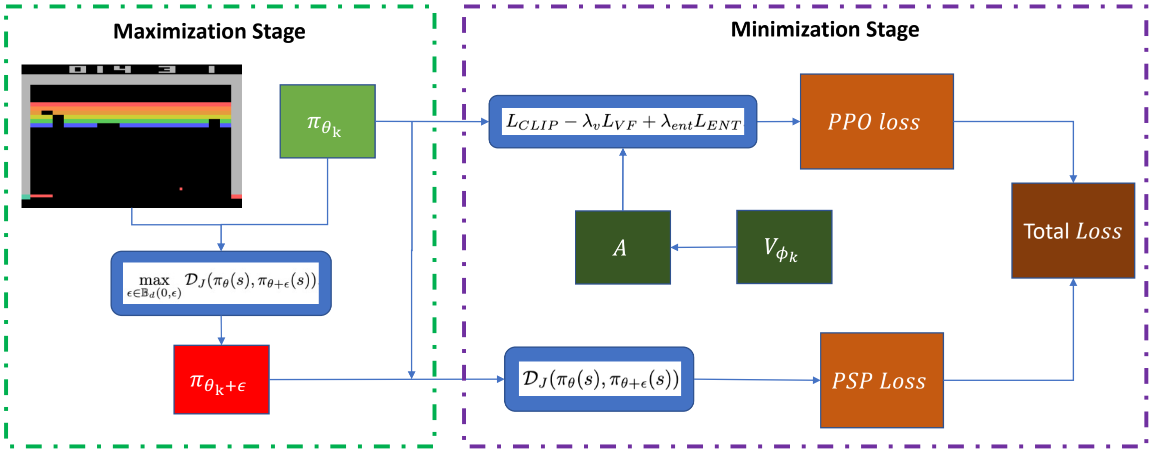
- We apply the proposed algorithms to RL tasks with discrete (Atari games) and continuous action space (OpenAI control). Extensive results show that the proposed algorithm can significantly improve the cumulative reward result on these benchmarks.

- As additional benefits of the PSP algorithm, we also show that the PSP method improves the exploration by increasing the entropy value of output distribution. At the same time, PSP improves the parameter-wise robustness against random or adversarial parameter corruption.

## 6.2 Background and Related Work

### *6.2.1 Reinforcement Learning Framework*

Markov decision framework (MDP) is a classic framework to solve decision making problem. MDP is based on two assumptions: 1, the environment is Markovian; and 2, the environment is fully observable. Formally, MDP can be formulated as an agent interacting with an environment in discrete time sequences in a quintuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}, r, p_0, \gamma)$, in which $\mathcal{S} \subseteq \mathbb{R}^S$ is a set of all observable states, $\mathcal{A} \subseteq \mathbb{R}^A$ is a set of actions, $\mathbb{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the state transition



Maximization Stage
Minimization Stage
$\pi_{\theta_k}$
$\max_{\epsilon \in \mathbb{B}_d(0,\epsilon)} \mathcal{D}_J(\pi_\theta(s), \pi_{\theta+\epsilon}(s))$
$\pi_{\theta_k+\epsilon}$
$L_{CLIP} - \lambda_v L_{VF} + \lambda_{ent} L_{ENT}$
PPO loss
$A$
$V_{\phi_k}$
Total Loss
$\mathcal{D}_J(\pi_\theta(s), \pi_{\theta+\epsilon}(s))$
PSP Loss

Figure 6.2 The overall pipeline of the proposed PSP-O algorithm. There are two stages of the algorithm: the maximization stage and the minimization stage. During the maximization state, the algorithm searches for a $\epsilon$ to maximize the distance between the current and adversarial policies. In the minimization stage, the algorithm minimizes the original PPO loss and the PSP-O loss to update the policy parameters.

function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $p_0$ is the initial distribution and $\gamma$ is a discount factor to reduce the impact of future reward.

The goal of a RL algorithm under MDP framework is to train an agent that can produce the trajectories that maximize the cumulative reward, which can be expressed as the summation of expected discounted rewards:

$$\max_\pi V(\pi) = \mathbb{E}_{s_0,a_0,\ldots}\left[\sum_{t\geq 0} \gamma^t r(s_t, a_t)\right], \tag{6.1}$$

where $s_0 \sim p_0, a_t \sim \pi(s_t), s_{t+1} \sim \mathbb{P}(s_{t+1}|s_t, a_t)$.

### *6.2.2 Proximal Policy Optimization (PPO)*

The PPO algorithm Schulman et al. (2017) is a on-policy RL algorithm that is based on PG Mnih et al. (2016) method. The PPO algorithm is designed based on TRPO algorithm by replacing the trust region method with the surrogate objective regularization. We briefly introduce the TRPO algorithm before describing the PPO algorithm. The objective function of TRPO algorithm shows below:

$$\begin{aligned} \theta_{k+1} = \arg\max_{\theta} \quad & \mathbb{E}_{\substack{s\sim\rho^{\pi_{\theta_k}}, \\ a\sim\pi_{\theta_k}}} \left[ \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a) \right], \\ \text{subject to} \quad & \mathbb{E}_{s\sim\rho^{\pi_{\theta_k}}} \left[ \mathcal{D}_{\mathrm{KL}}(\pi_{\theta_k}(\cdot|s) \| \pi_\theta(\cdot|s_)) \right] \leq \delta, \end{aligned} \tag{6.2}$$

where $A^\pi(s,a)$ is the advantage function and $\rho^\pi(s)$ is the state visitation distribution. They are defined as:

$$\begin{aligned} A^\pi(s,a) &= Q^\pi(s,a) - V^\pi(s), \\ \rho^\pi(s) &= \sum_{i\geq 0} \gamma^i \mathbb{P}(s_i = s), \end{aligned} \tag{6.3}$$

where $Q^{\pi}(s,a)$ is the the state-action value function and $V^{\pi}(s)$ is the value function. They can be expressed as:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{s_0=s,a_0,\ldots}\left[\sum_{t\geq 0}\gamma^t r(s_t,a_t)\right], \\ Q^{\pi}(s,a) &= \mathbb{E}_{s_1,a_1,\ldots}\left[\sum_{t\geq 0}\gamma^t r(s_t,a_t)|s_0=s,a_0=a\right]. \end{aligned} \tag{6.4}$$

PPO simplifies TRPO using a truncated objective function to reduce the complexity and allow it to use a first-order optimizer. The objective function to maximize in PPO is defined as:

$$L_{PPO} = L_{CLIP} - \lambda_v L_{VF} + \lambda_{ent} L_{ENT}. \tag{6.5}$$

The objective contains three parts: the surrogate loss $L_{CLIP}$, value function loss $L_{VF}$, and the entropy loss $L_{ENT}$. The $L_{VF}$ and $L_{ENT}$ is defined same with in the policy gradient algorithm. The $L_{CLIP}$ is defined as:

$$\begin{aligned} L_{CLIP} = \mathbb{E}_{\substack{s\sim\rho^{\pi_{\theta_k}},\\ a\sim\pi_{\theta_k}}}[min(r(\theta)A^{\pi_{\theta_k}}(s,a), \\ clip(r(\theta),1-\epsilon_{clip},1+\epsilon_{clip})A^{\pi_{\theta_k}}(s,a))], \end{aligned} \tag{6.6}$$

where $clip()$ clips the value if it is out of the bounds provided. By applying this function, it removes the incentive for moving outside of the interval. The $L_{VF}$ is defined as:

$$L_{VF} = \mathop{\mathbb{E}}_{s \sim \rho^{\pi_{\theta_t}}} \Big( V_\phi(s) - r(s,a) \Big)^2 \tag{6.7}$$

### *6.2.3 Related Works*

Parameter-wise smooth neural network is closely related to the concept of sharpness and weight space robustness of neural networks. The earliest research works on weight-space robustness is back to 1990s Murray & Edwards (1992); Hochreiter & Schmidhuber (1994). Adding random perturbations on the neural network parameter is proved to be an effective method to improve the training Jim et al. (1996); Graves et al. (2013); Srivastava et al. (2014). More recently, researchers observed that the sharpness has strong correlation with the generalization error on a large set of models Jiang et al. (2019). Followed the observation of paper Jiang et al. (2019), Sharpness-aware Minimization (SAM) Foret et al. (2020) is proposed to search flat local minima to improve the generalization performance. SAM uses the gradient at a worse-case parameter point in the vicinity to update the neural network at each iteration. In addition, SAM is essentially similar to the parameter-wise adversarial training that can improve the robustness of neural network to parameter corruption Sun et al. (2021). The difference between the PSP regularization proposed in this paper and the above methods are: 1, We apply parameter-wise smooth network on RL tasks for the first time; 2, We not only improve the smoothness in loss space, but also in output space for RL tasks.

Even though there is no research on parameter-wise smoothness in RL field before our work, smooth exploration and observation-wise smoothness has applied to RL in several research works. Paper Raffin et al. (2022) adapts state-dependent exploration (SDE) to Deep RL algorithms to imporve the smoothness of exploration. $SR^2L$ is closely related to this work; $SR^2L$ Shen et al. (2020) applies a smoothness-inducing regularization to encourage the output of the policy (decision) to not change much when injecting small perturbation to the input of the policy (observed state). Results show that , our $SR^2L$ effectively reduces the size of the search space when learning the policy network and achieves state-of-the-art sample efficiency. Compared to $SR^2L$, our work applies parameter-wise smooth policy, instead of observation-wise, to RL algorithms.

## 6.3 Method

In this section, we describe two variants of parameter-wise smooth policy (PSP) algorithms that improve the parameter-wise smoothness of the policy network in PPO algorithm. The first algorithm, PSP-O, improves the parameter-wise smoothness with respect to the policy outputs (decision). On the other hand, the second algorithm, PSP-L, improves the parameter-wise smoothness with respect to the loss value space.

### *6.3.1 Motivation*

While it is appealing to perform multiple optimization steps on the policy gradient using the same trajectory, empirically, it often leads to destructively large policy updates. To mitigate this issue, TRPO proposed the trust region method, while PPO proposed the CLIP Loss method. Essentially, TRPO and PPO are searching for a new policy that is not destructively updated, or in other words, smoothly updated. Instead of constraining each iteration's optimization, can we regularize the smoothness of the policy network to enforce the smoothness of policy updates?

With this idea, we introduce a novel regularization with two variants, PSP-O and PSP-L, to improve the parameter-wise policy smoothness. The framework of algorithm PSP-O is shown in Figure 6.2. As seen in the figure, we aim to train a parameter-wise smooth policy in two stages in PSP-O: the maximization and the minimization stages. In the maximization stage, it searches for an adversarial parameter corruption to disturb the current policy as strongly as possible. In the minimization stage, it aims to update the current policy to minimize the disturbance of the adversarial parameter noise. We propose to use two criteria to measure the distance (disturbance) between the original and the adversarial policies in this Min-Max process: distance in the output (decision) space and distance in the loss value space. We call the methods with these two criteria as PSP-O and PSP-L accordingly.

### *6.3.2 Parameter-Wise Smooth Policy Regularization*

Parameter-Wise Smooth Policy (PSP) regularization encourages the policy $\pi_\theta$ and adversarial policy $\pi_{\theta+\epsilon}$ to be close to each other, where $\epsilon$ is the gradient-based corruption on the parameter $\theta$ that maximize the distance $\mathcal{D}$ between the policy and the adversarial policy. We apply a metric function $\mathcal{D}$ to measure the distance between the outputs of the original policy and the corrupted policy. The parameter-wise policy smoothness can be computed by searching for the adversarial parameter noise to maximize the distance: $\max_{\epsilon\in\mathbb{B}_d(0,\epsilon)} \mathcal{D}(\pi_\theta, \pi_{\theta+\epsilon})$. By minimizing this PSP regularization, we can improve the smoothness of the trained policy. With this definition, our parameter-wise smooth policy regularization is defined as:

$$\mathcal{R}^\pi(\theta) = \max_{\epsilon\in\mathbb{B}_d(0,\epsilon)} \mathcal{D}(\pi_\theta, \pi_{\theta+\epsilon}). \tag{6.8}$$

In practice, $\mathcal{D}(\pi_\theta, \pi_{\theta+\epsilon})$ can be measured with different approximation methods. We consider two variants in this paper: the first one is Jeffrey's divergence of the output of the policy network, and the second one is the difference between the loss values of the two policies. These two variants will be described in Section 6.3.3 (policy output space) and Section 6.3.4 (loss value space).

### *6.3.3 Parameter-wise Smooth Policy Regularization in Output Space*

The first method we propose to approximate $\mathcal{D}(\pi_\theta, \pi_{\theta+\epsilon})$ is to measure the distance between the outputs (decision) of the original policy and adversarial policy. We call this method PSP-O (PSP-Output). Specifically, we compute Jeffrey's Divergence of original and adversarial

**Algorithm 3:** PPO with PSP-O

```
Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$, coefficient
       for value loss $\lambda_{VF}$, coefficient for parameter-wise smooth policy
       regularization $\lambda_{PSP}$
for $k = 0, 1, 2, ...$ do
    Initialize $\epsilon$
    Collect set of trajectories $s \sim \rho^{\pi_{\theta_t}}$ by running policy $\pi(\theta_k)$ in the environment.
    Compute $D(\pi_{\theta_k}, \pi_{\theta_k}) \approx \mathbb{E}_{s\sim\rho^{\pi_{\theta_t}}} \mathcal{D}_J(\pi_{\theta_k}(s), \pi_{\theta_k+\epsilon}(s))$
    for $j = 0, 1, 2, ...$ do
        Update $\epsilon$ by: $\epsilon_{j+1} = \mathbb{B}_d(\epsilon_j + \bigtriangledown_\epsilon(D(\pi_{\theta_k}, \pi_{\theta_k})))$
    Compute $L_{PSRO}$ with Equation 6.10
    Compute advantage estimates, $\hat{A}_t$ based on the current value function $V_{\phi_k}$
    Compute the PPO-Clip loss with Equation 6.6
    Compute the value function loss with Equation 6.7
    Compute the gradient:
        $$g = \bigtriangledown_{(\theta,\phi)}(L_{ppo} - \lambda_{VF}L_{val} - \lambda_{PSP}L_{PSP})$$
    Update policy network and value function:
        $$(\theta_{k+1}, \phi_{k+1}) = (\theta, \phi) + \eta g$$
```

policies as the distance measurement criterion. The approximation is shown below:

$$\mathcal{D}(\pi_\theta, \pi_{\theta+\epsilon}) \approx \mathcal{D}_J(\pi_\theta(s), \pi_{\theta+\epsilon}(s)). \tag{6.9}$$

During training, we compute the state transition distribution $\rho^\pi$ by the policy $\pi_\theta$, and take expectation with respect to it. Thus PSP-O regularization can be expressed as:

$$L_{PSP-O}(\theta) = \mathop{\mathbb{E}}_{s\sim\rho^{\pi_{\theta_t}}} \max_{\epsilon\in\mathbb{B}_d(0,\epsilon)} \mathcal{D}_J(\pi_\theta(s), \pi_{\theta+\epsilon}(s)), \tag{6.10}$$

in which the Jeffrey's divergence is shown as:

$$\mathcal{D}_{\mathrm{J}}(P||Q) = \frac{1}{2}\mathcal{D}_{\mathrm{KL}}(P||Q) + \frac{1}{2}\mathcal{D}_{\mathrm{KL}}(Q||P). \tag{6.11}$$

PSP-O regularization encourages the output (decision) of $\pi_\theta(s)$ to not change much to the output of $\pi_{\theta+\epsilon}(s)$ when adding a small perturbation on the policy parameters. In other words, it encourages $\pi_\theta(s)$ to be smooth on all possible updating to $\pi_{\theta+\epsilon}(s)$ within the neighborhoods. For this reason, PSP-O can effectively avoid destructive large policy updates which hurt the training.

With the proposed PSP-O regularization, we can train smooth policy by incorporating it into the PPO algorithm. To apply the PSP-O regularization to the PPO algorithm, we keep the CLIP loss and the value function loss of PPO and discard the entropy loss. Formally, the loss function of PSP-O is shown as follows:

$$L_{PPO} = L_{CLIP} - \lambda_{VF}L_{VF} + \lambda_{PSP}L_{PSP-O}, \tag{6.12}$$

in which $\lambda_{PSP}$ is the ratio of PSP-O regularization. Notably, with the PSP-O regularization, we discard the entropy loss in the PPO algorithm. Algorithm details can be found in the Algorithm 3.

### *6.3.4 Parameter-wise Smooth Policy Regularization in Loss Space*

The second method we propose to approximate $\mathcal{D}(\pi_\theta, \pi_{\theta+\epsilon})$ is to measure the difference between loss values of the original policy and adversarial policy, which is called PSP-L (PSP-Loss). Policy output (decision) smoothness is directly related to the exploration of PPO algorithms, while loss value closely reflects the advantage value and is optimized during the agent training process. For this reason, the loss value difference is a meaningful criterion to measure the policy distance. With this alternative definition, the PSP-L regularization can be represented as the below form:

$$\mathcal{R}_{\mathrm{s}}^{\pi}(\theta) \approx \mathop{\mathbb{E}}_{s\sim\rho^{\pi_{\theta_t}}} [\max_{\epsilon\in\mathbb{B}_d(0,\epsilon)} (L(\theta+\epsilon, s) - L(\theta, s))]. \tag{6.13}$$

With the above form of PSP regularization, we can derive the new loss function as follows:

$$\begin{aligned} L_{PSP-L} &= \mathop{\mathbb{E}}_{s\sim\rho^{\pi_{\theta_t}}} L(\theta, s) + \mathcal{R}_{\mathrm{s}}^{\pi}(\theta) \\ &= \mathop{\mathbb{E}}_{s\sim\rho^{\pi_{\theta_t}}} L(\theta, s) + [\max_{\epsilon\in\mathbb{B}_d(0,\epsilon)} L(\theta+\epsilon, s) - L(\theta, s)] \\ &= \max_{\epsilon\in\mathbb{B}_d(0,\epsilon)} L(\theta+\epsilon, s). \end{aligned} \tag{6.14}$$

We can see that this new loss function is essentially similar to the SAM algorithm Bahri et al. (2021). The current policy can be updated by the gradient of the adversarial policy parameters in the vicinity at each iteration.

To apply the PSP-L loss in the PPO algorithm, we can use the gradient of the CLIP loss

**Algorithm 4:** PPO with PSP-L

```
Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$, coefficient
       for value loss $\lambda_{VF}$
for $k = 0, 1, 2, ...$ do
    Collect set of trajectories $D_k = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$ in the
     environment.
    Compute rewards-to-go $\hat{R}_t$
    Compute advantage estimates, $\hat{A}_t$ based on the current value function $V_{\phi_k}$
    Compute the PPO-Clip loss with Equation 6.6
    Compute the value function loss with Equation 6.7
    Compute the gradient of the current weight:
        $$g_{cur} = \bigtriangledown_{(\theta,\phi)}(L_{CLIP} - \lambda_{VF}L_{VF})$$
    Compute the gradient of the adversarial weight:
        $$g_{adv} = \bigtriangledown_{(\theta,\phi)}(L_{CLIP} - \lambda_{VF}L_{VF}) \mid_{(\theta,\phi)+g_{cur}}$$
    Update policy network and value function:
        $$(\theta_{k+1}, \phi_{k+1}) = (\theta, \phi) + \eta g_{adv}$$
```

with respect to the adversarial parameters to update the current parameter in every iteration. Training Agent with this novel loss function $L_{PSP-L}$ can increase the smoothness of the policy network in the loss value space. Algorithm details can be found in the Algorithm 4.

## 6.4 Experiments

We now validate PSP-O and PSP-L on benchmark datasets. Specifically, we evaluate the performance of PSP-O and PSP-L on Atari 2600 games Bellemare et al. (2013) in Section 6.4.1 and OpenAI gym control environments Todorov et al. (2012) in Section 6.4.2. We also evaluate the parameter-wise robustness in Section 6.4.3 and conduct ablation studies in other sections to show the insights and advantages of the proposed algorithms. Benchmark datasets, network structures, and training procedure details can be found in the Appendix.

Our source code is provided as a part of supplementary materials.

| | PPO | PSP-O | PSP-L |
|---|---|---|---|
| Alien | 1307 | **4697** | 4314 |
| Amidar | 1280 | **1297** | 1188 |
| Assault | 10596 | 11622 | **12832** |
| Asterix | 13203 | **29185** | 21827 |
| Asteroids | 2393 | **2412** | 2278 |
| Atlantis | 2866108 | 3238262 | **3571303** |
| BankHeist | **1285** | 1243 | 1248 |
| BattleZone | 29475 | 30376 | **38245** |
| BeamRider | 6235 | **8325** | 8214 |
| Bowling | 56 | 42 | **68** |
| Boxing | 96 | **98** | **98** |
| Breakout | 425 | 417 | **465** |
| Centipede | 3895 | **6372** | 5655 |
| ChopperCommand | 1047 | 1007 | **1111** |
| CrazyClimber | 121540 | 123490 | **129824** |
| DemonAttack | 89663 | 114291 | **184582** |
| DoubleDunk | -2 | **-1** | **-1** |
| Endura | 1163 | 1167 | **1259** |
| FishingDerby | 35 | **48** | 28 |
| Freeway | 32 | **33** | **33** |
| Frostbite | 319 | **9228** | 3968 |
| Gopher | 12050 | 12939 | **17550** |
| Gravitar | 1156 | **2259** | 1184 |
| IceHockey | **-3** | -4 | **-3** |
| Jamesbond | 1225 | **8925** | 4401 |
| Kangaroo | 13607 | 1797 | **14383** |
| Krull | 8610 | 8828 | **9236** |
| KungFuMaster | 33078 | 38326 | **53388** |
| MontezumaRevenge | **44** | 2 | 2 |
| MsPacman | 4235 | **6407** | 5348 |
| NameThisGame | **8800** | 6235 | 6094 |
| Pitfall | **0** | -11 | **0** |
| Pong | 20 | 20 | 20 |
| PrivateEye | 108 | **109** | 99 |
| Qbert | 21697 | **26960** | 26725 |
| Riverraid | 12340 | 13252 | **13435** |
| RoadRunner | **55412** | 52219 | 36946 |
| Robotank | 21 | **26** | 0 |
| Seaquest | 1868 | **1916** | 1836 |
| SpaceInvaders | 2511 | **2824** | 2730 |
| StarGunner | 62789 | **84777** | 77881 |
| Tennis | -2 | -3 | **-1** |
| TimePilot | 11650 | **13190** | 12916 |
| Tutankham | **198** | 177 | 167 |
| UpNDown | 532388 | **684425** | 399486 |
| Venture | 7 | **12** | 2 |
| VideoPinball | **186256** | 168582 | 177764 |
| WizardOfWor | 9710 | **10137** | 9612 |
| Zaxxon | 13953 | **19028** | 15691 |
| Winning Games Number | 8 | 25 | 20 |

Table 6.1 Maximum cumulative reward score of Atari games. The last row reports the number of games each algorithm achieves the highest score.

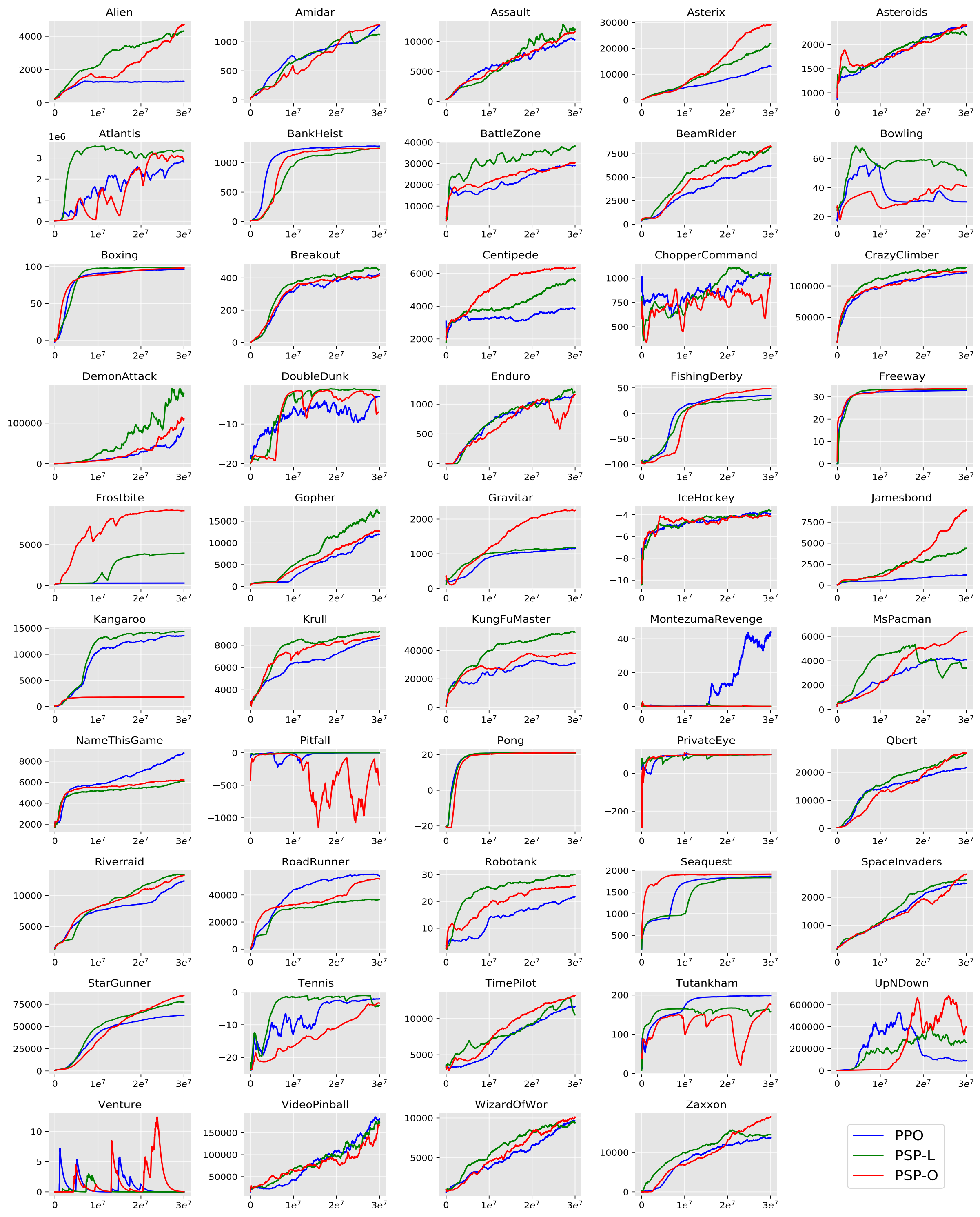


Figure 6.3 Training curves of PPO, PSP-O and PSP-L on 49 Atari games.

### *6.4.1 Results on the Atari Games*

We first evaluate the performance of PSP on Atari games, which have pixels as high dimensional observations. We train policies on each game with three different seeds and report the average cumulative rewards in this section.

In Figure 6.3, we show the cumulative rewards during the training process with PPO, PSP-O, and PSP-L. We report the maximum testing cumulative reward of the three algorithms during the training process in Table 6.1. It can be observed that PSP-O and PSP-L can achieve higher cumulative rewards than PPO in most Atari games. In total, there are only 8 out of 49 games in which PPO gets the best results, while PSP-O "won" in 25 games and PSP-L "won" in 20 games.

### *6.4.2 Results in Continuous Domain: OpenAI Gym Control Tasks*

We further evaluate the performance of PSP-O and PSP-L on tasks in the continuous domain. We choose four tasks from the OpenAI gym control environments with the MuJoCo physics simulator. Same as in Atari experiments, we train policies with three different seeds and report the average cumulative rewards. A detailed description of these four environments can be found in the Appendix.

Figure 6.4 shows the cumulative reward curves during the training process of PPO, PSP-O, and PSP-L in environments Ant-v3, Waker2d-v3, Halfcheetah-v3, and Swimmer-v3. As we can see, PSP-O and PSP-L learn better policies with significantly higher cumulative rewards. Specifically, PSP-O or PSP-L achieves better maximum reward than PPO on all four tasks. Furthermore, PSP-O and PSP-L have faster training speed and need fewer number of iterations to achieve the maximum reward thanks to the improved sample efficiency enabled by high entropy output exploration.

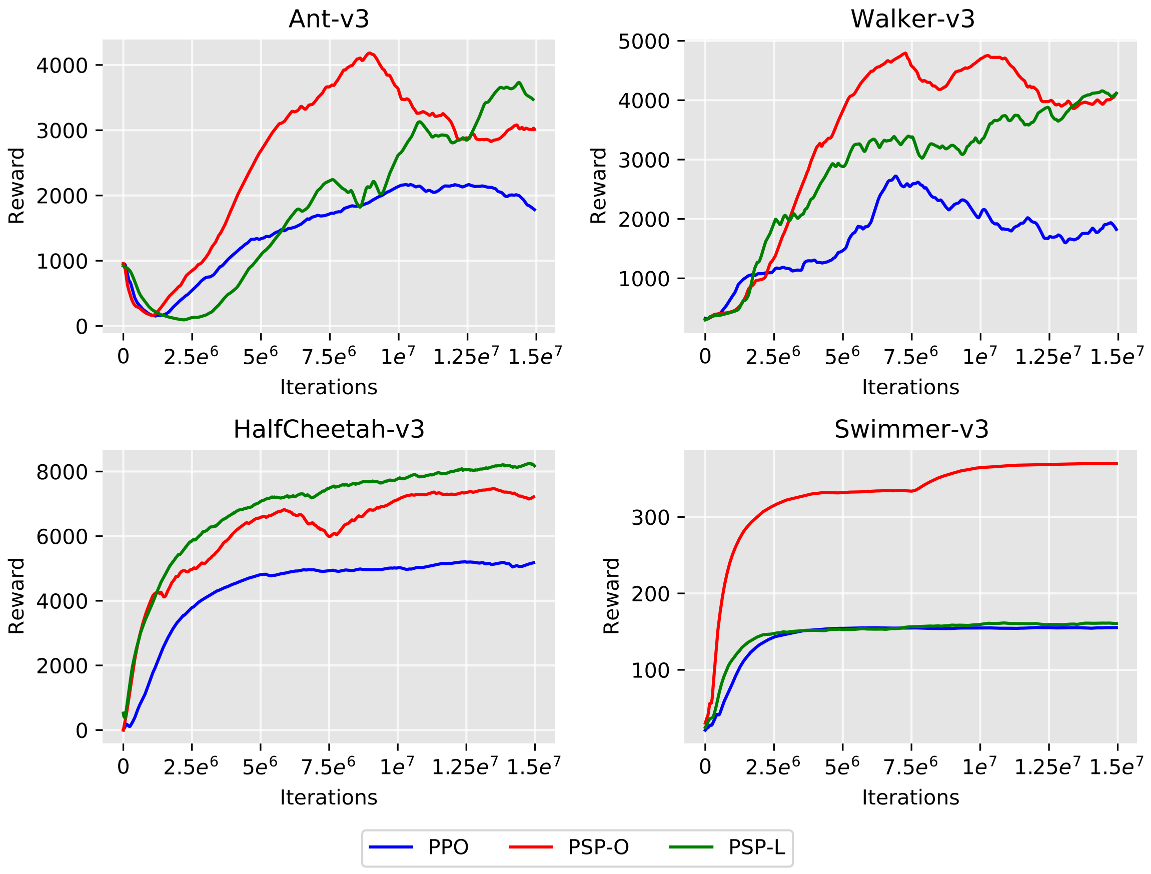


Figure 6.4 Training curves of PPO, PSP-O, and PSP-L. PSP-O and PSP-L learn better policies than PPO with significant improvements in terms of accumulated rewards.

### *6.4.3 Parameter-wise Policy Robustness*

Parameter-wise smoothness is closely related to the model robustness against parameter corruption Foret et al. (2020); Sun et al. (2021). Although PSP is motivated by improving the parameter-wise smoothness of policies, we demonstrate that PSP also improves the robustness of the policy against both random and adversarial parameter corruption attacks.

To evaluate the parameter-wise robustness of policies, we apply two types of policy parameter corruption: random attack and adversarial attack. For random parameter corruption, we add noise uniformly sampled from $\mathbb{B}_d(0, \epsilon) = \{\delta : \|\delta\|_\infty \leq \epsilon\}$ to the parameters; for adversarial parameter corruption, we apply projected gradient descent on the original policy on every iteration during the evaluation, by solving $\tilde{\delta} = \arg\max_{\delta \in \mathbb{B}_d(0,\epsilon)} \mathcal{D}(\pi_\theta(s), \pi_{\theta+\epsilon}(s))$.

Figure 6.5 shows the evaluation result. As we can see, the reward decreases as we increase the attack strength for both random and adversarial attacks on PPO, PSP-O, and PSP-L. However, the cumulative rewards of policies trained by PSP-O and PSP-L decrease much slower than policies trained by PPO against both attacks. This observation demonstrates the improved parameter-wise robustness of PSP-O and PSP-L.

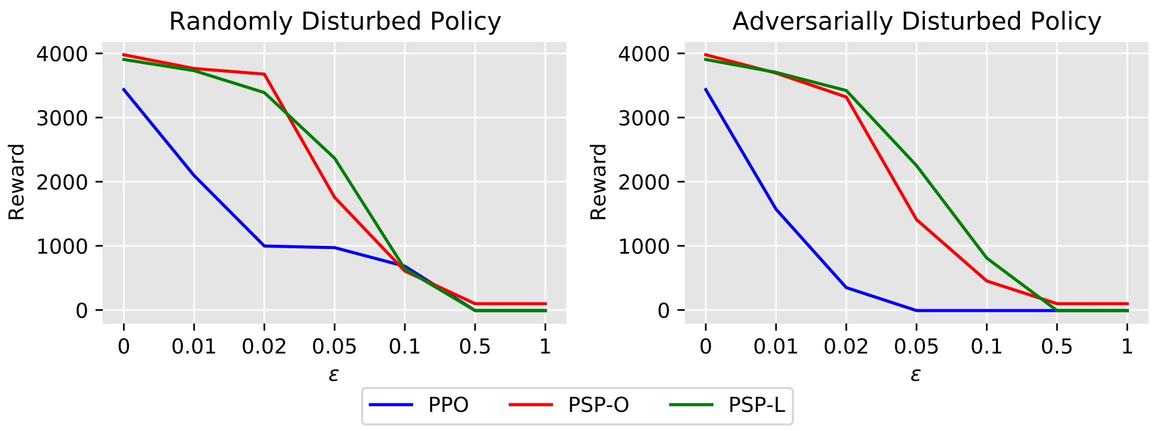


Figure 6.5 Comparison of the cumulative rewards of PPO, PSP-O, and PSP-L under random (left) and adversarial (right) parameter corruption attack in Walker2d environment. The result shows that policies trained with PSP-O and PSP-L are more robust than policies trained with PPO under both parameter corruption attacks.

### *6.4.4 Entropy of Policy Outputs*

In this section, we show that policy trained with PSP has higher output entropy than policy trained with PPO, even though no policy output entropy loss is applied in PSP. In other words, PSP regularization can promote higher entropy by improving the parameter-wise policy smoothness. With high output entropy, PSP can enhance the exploration by avoiding situations where the agents fall into local optimum Haarnoja et al. (2018).

In Figure 6.6, we show the comparison of the entropy values of policy output distribution between PPO and PSP on two Atari games (Alien and Asterix) during the training process. As we can see, PSP-O and PSP-L learn policies with large output entropy, which is significantly higher than that of the policies trained with PPO with an explicit entropy loss.

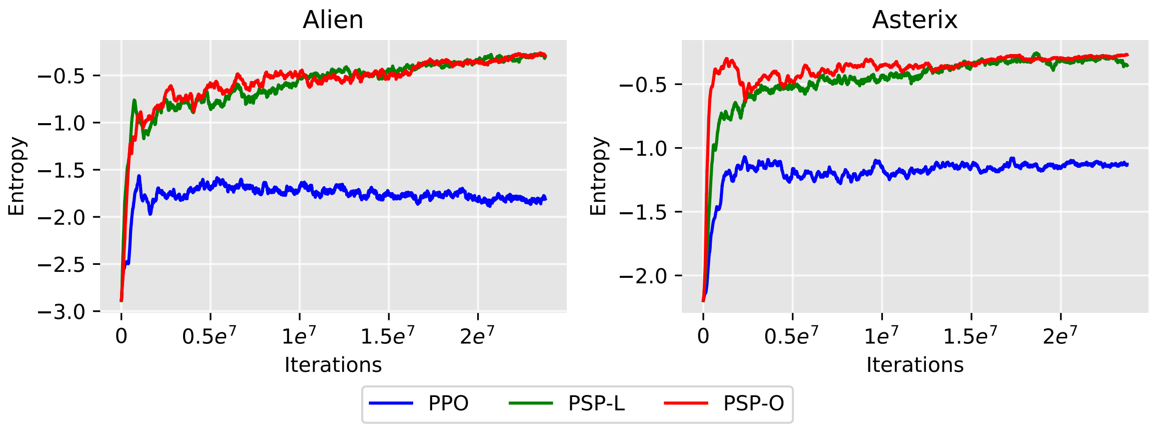


Figure 6.6 Entropy value of output distribution from the policies trained by PPO, PSP-O, and PSP-L. We show the results of two Atari games (Alien and Asterix) along the training process. It can be observed that PSP-O and PSP-L can train policies with higher entropy, even though PSP-O and PSP-L do not use the entropy loss during training.

### *6.4.5 Parameter-wise Loss Landscape Visualization*

Following the paper Li et al. (2018a), we visualize the loss landscape of the learned agents of PPO and PSP-L algorithms. Policy networks have a large number of parameters, and thus the loss function is in a high-dimensional space. For visualization, we attempt to use 2D plots as the landscape of policy network loss.

Similar to paper Li et al. (2018a), to plot 2D visualization of loss landscape around the current policy, we use the parameters of current policy network $\theta^*$ as the center point, and randomly choose two directions: $\delta$ and $\eta$; we then plot a function of the form: $f(\alpha, \beta) = L(\theta^* + \alpha\delta + \beta\eta)$, where $L$ denotes the loss function.

In Figure 6.7, we plot the landscape of models trained with PPO and PSP-L on Atari game Asteroids. As we can see, the model trained with PPO has a sharper local minimum than the model trained with PSP-L. In other words, PSP-L can train parameter-wise smoother policy networks than PPO.

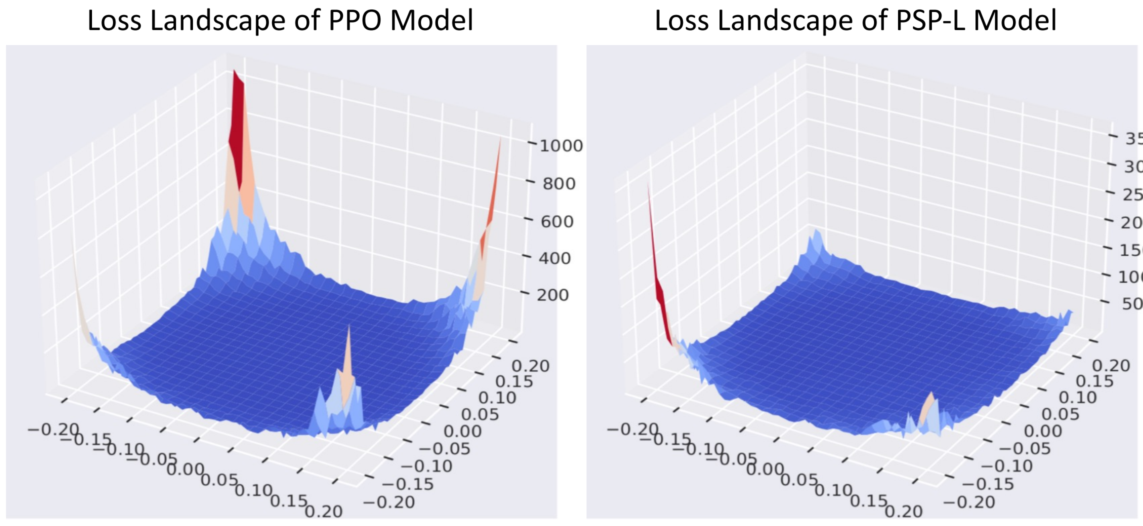


Figure 6.7 Visualizing the loss landscapes Li et al. (2018a) of different policy networks trained by PPO (left) and PSP-L(right). Note the significant scale differences of the y-axis, indicating PSP-L learns a policy that is parameter-wise smoother in the loss space.

## 6.5 Conclusion

This paper introduces parameter-wise smooth policy (PSP) regularization for PPO to promote the training stability and sample efficiency (thanks to improved exploration). Two types of parameter-wise policy smoothness are introduced. PSP-O encourages small changes in the output decision space when inserting bounded perturbations to the parameters of policy networks. On the other hand, PSP-L encourages small changes in the loss values when perturbing the policy parameters. To investigate the effectiveness of PSP-O and PSP-L, we examine them on well-established RL benchmarks with discrete and continuous action spaces. Empirical results show that PSP significantly improve the cumulative rewards and parameter-wise robustness.

# CHAPTER 7
# Future Works

## 7.1 Observation-wise and Parameter-wise Smooth (Robust) Policy

In PSP, we proposed to use the parameter-wise smooth policy to improve the PPO algorithms. On the other hand, recent researches show that observation-wise smooth policy can also improve the performance of RL algorithms. A promising direction for the following work to PSP is to incorporate observation-wise smoothness into the PSP framework. In other words, we can apply policy networks that are smooth to both observation and parameter perturbation to RL algorithms. Ideally, this can produce an agent model that is robust to both observation and parameter perturbation.

## 7.2 Combining the Output Space and Loss Space Parameter-wise Smoothness (Robustness)

In the PSP paper, we proposed two effective parameter-wise smoothness algorithms, PSP-O and PSP-L. PSP-O encourages small changes in the output decision space when inserting bounded perturbations into the parameters of policy networks. On the other hand, PSP-L encourages small changes in the loss values when perturbing the policy parameters. Can we combine these two methods to make the policy to be smooth in both output and loss space? This should be a promising direction to improve the current PSP algorithm.

## 7.3 Parameter-wise Smoothness (Robustness) Training in Output Space

y

# CHAPTER 8
# Conclusion

In this dissertation, we comprehensively study and propose a series of works on robust and interpretable deep learning algorithms in both computer vision and reinforcement learning areas.

The first algorithm proposed in this dissertation is a purification-based adversarial defense algorithm: Defense-VAE. Defense-VAE is a simple yet effective defense algorithm that uses a variational autoencoder (VAE) to purge adversarial perturbations from contaminated images. The proposed method is generic and can defend white-box and black-box attacks without retraining the original CNN classifiers. It can further strengthen the defense by retraining CNN or end-to-end finetuning the whole pipeline.

The second algorithm proposed in this dissertation, GDPA, is a novel patch attack and adversarial training algorithm. GDPA is an end-to-end patch attack algorithm, Generative Dynamic Patch Attack (GDPA), which generates both patch pattern and patch location adversarially for each input image. We show that GDPA is a generic attack framework that can produce dynamic/static and visible/invisible patches with a few configuration changes.

The third algorithm, NICE, is a generative interpretation algorithm on deep neural networks. Compared to many existing explanation algorithms that heavily rely on backpropagation, the sparse masks generated by NICE are much more concise and coherent and align well with human intuitions. With the sparse masks, the proposed mixed-resolution image compression achieves higher compression rates compared to the existing semantic compression algorithms while retaining similar classification accuracies.

The fourth work SDNN in this dissertation is an interpretable sparse DNN algorithm on MRI and Gene data. SDNN is a sparse deep neural network approach to identify sparse and interpretable features for schizophrenia (SZ) discrimination. An L0-norm regularization is implemented on the network's input layer for sparse feature selection, which can later be interpreted based on importance weights. We applied the proposed approach on a large multi-study cohort (N = 1,684) with gray matter volume (GMV) and single nucleotide polymorphism (SNP) data for SZ discrimination.

The last work, PSP, proposed in this dissertation is parameter-wise adversarial training in the reinforcement learning area. PSP trains a parameter-wise smooth policy network in PPO to tackle these challenges. Specifically, we introduce a Parameter-wise Smooth Policy (PSP) regularization to enforce the outputs of a policy or loss values not to change much when injecting small perturbations to the parameters of policy networks. With the PSP regularization, we improve the stability of PPO and promote sample efficiency by learning policies with high entropy outputs for exploration, thus improving the cumulative rewards and robustness against random and adversarial parameter corruption.

As for future works, I believe the following three directions are worthy of further investigation: 1) How to apply policy networks that are smooth (robust) to both observation and parameter perturbation to RL algorithms; 2) How to combine PSP-O and PSP-L to make the policy to be smooth (robust) in both output and loss space; 3) How to extend output space smoothness (robustness) to tasks other than RL areas.

## REFERENCES

A. G. Garrity, e. a. 2007, Am J Psychiat

Akhtar, N., Liu, J., & Mian, A. 2017, Defense against Universal Adversarial Perturbations

Akhtar, N., Liu, J., & Mian, A. 2018, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition

Amari, S. 1998, Neural Comput

Andreasen, N. C., & Pierson, R. 2008, Biol Psychiat

Ashburner, J., & Friston, K. J. 2005, Neuroimage

B. J. Casey, e. a. 2013, Nat Rev Neurosci

B. T. T. Yeo, e. a. 2011, J Neurophysiol

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., & Samek, W. 2015, PloS one, 10

Bahri, D., Mobahi, H., & Tay, Y. 2021, arXiv preprint arXiv:2110.08529

Ballé, J., Laparra, V., & Simoncelli, E. P. 2016, arXiv preprint arXiv:1611.01704

Ballé, J., Laparra, V., & Simoncelli, E. P. 2017, in ICLR

Baluja, S., & Fischer, I. 2017, arXiv preprint arXiv:1703.09387

Bang, S., Xie, P., Wu, W., & Xing, E. 2019, arXiv preprint arXiv:1902.06918

Battenberg, E. et al. 2017, arXiv preprint arXiv:1707.07413

Bearman, A., Russakovsky, O., Ferrari, V., & Fei-Fei, L. 2016, in European conference on computer vision

Bell, A. J., & Sejnowski, T. J. 1995, Neural Comput

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. 2013, Journal of Artificial Intelligence Research

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., & Roli, F. 2013, in Joint European conference on machine learning and knowledge discovery in databases, Springer, 387–402

Bird, T., Kunze, J., & Barber, D. 2018a, arXiv preprint arXiv:1809.04855

——. 2018b, arXiv preprint

Brown, T. B., Mané, D., Roy, A., Abadi, M., & Gilmer, J. 2017, arXiv preprint arXiv:1712.09665

C. F. Beckman, e. a. 2005, Philos Trans R Soc Lond B Biol Sci

C. N. Gupta, e. a. 2015, Schizophr Bull

Carlini, N., & Wagner, D. 2017, in 2017 IEEE Symposium on Security and Privacy (SP), 39–57

Carvalho, D. V., Pereira, E. M., & Cardoso, J. S. 2019, Electronics, 8

Chen, J., Liu, J., & Calhoun, V. D. 2019a, P Ieee

Chen, J., Song, L., Wainwright, M. J., & Jordan, M. I. 2018a, arXiv preprint arXiv:1802.07814

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. 2018b, in Proceedings of the European conference on computer vision (ECCV)

Chen, M., Artières, T., & Denoyer, L. 2019b, in Advances in Neural Information Processing

Systems (NIPS)

Chen, M., Beutel, A., Covington, P., Jain, S., Belletti, F., & Chi, E. H. 2019c, in Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining

Chen, X., Hsieh, C.-J., & Gong, B. 2021, arXiv preprint arXiv:2106.01548

Chiang, P.-y., Ni, R., Abdelkader, A., Zhu, C., Studer, C., & Goldstein, T. 2020, in ICLR 2020

Cuthbert, B. N. 2014, World Psychiatry

Cuthbert, B. N., & Insel, T. R. 2013, BMC medicine

D. Lin, e. a. 2017, Schizophr Bull

D. M. Altshuler, e. a. 2012, Nature

Dabkowski, P., & Gal, Y. 2017, in NIPS

Dai, J., He, K., & Sun, J. 2015, in Proceedings of the IEEE international conference on computer vision

Delaneau, O., Marchini, J., & Zagury, J. F. 2012, Nat Methods

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. 2009, in IEEE conference on computer vision and pattern recognition (CVPR), 248–255

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. 2018, arXiv preprint arXiv:1810.04805

Dong, Y., Fu, Q.-A., Yang, X., Pang, T., Su, H., Xiao, Z., & Zhu, J. 2020, in Conference on Computer Vision and Pattern Recognition

Dziugaite, G. K., Ghahramani, Z., & Roy, D. M. 2016, A study of the effect of JPG compression on adversarial images

E. I. Ivleva, e. a. 2013, Am J Psychiatry

E. Vassos, e. a. 2017, Biol Psychiat

Evtimov, I., Eykholt, K., Fernandes, E., Kohno, T., Li, B., Prakash, A., Rahmati, A., & Song, D. 2017, in CoRR

Fong, R. C., & Vedaldi, A. 2017, in IEEE International Conference on Computer Vision (CVPR), 3429–3437

Foret, P., Kleiner, A., Mobahi, H., & Neyshabur, B. 2020, arXiv preprint arXiv:2010.01412

Gan, C., Wang, N., Yang, Y., Yeung, D.-Y., & Hauptmann, A. G. 2015, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2568–2577

Gao, J., Wang, B., Lin, Z., Xu, W., & Qi, Y. 2017, DeepCloak: Masking Deep Neural Network Models for Robustness Against Adversarial Samples

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. 2014a, in NIPS

Goodfellow, I. J., Shlens, J., & Szegedy, C. 2014b, arXiv preprint arXiv:1412.6572

Goodfellow, I. J., Shlens, J., & Szegedy, C. 2015, in International Conference on Learning Representations

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., & Duvenaud, D. 2018, in International Conference on Learning Representations (ICLR)

Graves, A., Mohamed, A.-r., & Hinton, G. 2013, in 2013 IEEE international conference on acoustics, speech and signal processing

Griffin, G., Holub, A., & Perona, P. 2007

Gu, S., & Rigazio, L. 2014a, Towards Deep Neural Network Architectures Robust to Adversarial Examples

——. 2014b, arXiv preprint arXiv:1412.5068

Guo, C., Rana, M., Cisse, M., & van der Maaten, L. 2017a, Countering Adversarial Images using Input Transformations

Guo, C., Rana, M., Cissé, M., & van der Maaten, L. 2017b, CoRR, abs/1711.00117

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. 2018, in International conference on machine learning

Hayes, J. 2018, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops

He, K., Gkioxari, G., Dollár, P., & Girshick, R. 2017, in Proceedings of the IEEE international conference on computer vision

He, K., Zhang, X., Ren, S., & Sun, J. 2016, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778

Hochreiter, S., & Schmidhuber, J. 1994, Advances in neural information processing systems

Hudson, D. A., & Manning, C. D. 2018, arXiv preprint arXiv:1803.03067

Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., & Hutter, M. 2019, Science Robotics

Insel, T. R. 2014, Am J Psychiat

Insel, T. R., & Cuthbert, B. N. 2015, Science

Ish-Horowicz, J., Udwin, D., Flaxman, S., Filippi, S., & Crawford, L. 2019, arXiv preprint arXiv:1901.09839

J. Chen, e. a. 2013, Neuroimage

——. 2017, Biol Psychiat

J. Frank, e. a. 2015, Mol Psychiatr

J. M. Segall, e. a. 2009, Schizophrenia Bull

J. McGrath, e. a. 2008, Epidemiologic reviews

Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. 2015, in International Conference on Neural Information Processing Systems

Jang, E., Gu, S., & Poole, B. 2017, in International Conference on Learning Representations (ICLR)

Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., & Bengio, S. 2019, arXiv preprint arXiv:1912.02178

Jim, K.-C., Giles, C. L., & Horne, B. G. 1996, IEEE Transactions on neural networks

Johnston, N. et al. 2018, in CVPR, 4385–4393

Karmon, D., Zoran, D., & Goldberg, Y. 2018, in International Conference on Machine Learning

Kingma, D. P., & Ba, J. 2015, in International Conference on Learning Representations (ICLR)

Kingma, D. P., & Welling, M. 2013, arXiv preprint arXiv:1312.6114

Krizhevsky, A. 2009, Learning multiple layers of features from tiny images, Tech. rep.

Kurakin, A., Goodfellow, I., & Bengio, S. 2016, arXiv preprint arXiv:1611.01236

L. L. Zeng, e. a. 2018, Ebiomedicine

L. Xu, e. a. 2008, International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. 1998, in Proceedings of the IEEE

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. 1998, Proceedings of the IEEE, 86, 2278

Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. 2018a, Advances in neural information processing systems

Li, J., Monroe, W., & Jurafsky, D. 2016, arXiv preprint arXiv:1612.08220

Li, J., Yao, L., Xu, X., Cheng, B., & Ren, J. 2020, Information Sciences, 532, 110

Li, M., Zuo, W., Gu, S., Zhao, D., & Zhang, D. 2018b, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

Li, X., & Ji, S. 2019, in The ECML Workshop on Machine Learning for Cybersecurity (MLCS)

Lin, D., Dai, J., Jia, J., He, K., & Sun, J. 2016, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition

Liu, A., Wang, J., Liu, X., Cao, B., Zhang, C., & Yu, H. 2020, in Proc. Eur. Conf. Comput. Vis.

Liu, N., Du, M., Guo, R., Liu, H., & Hu, X. 2021, ACM SIGKDD Explorations Newsletter

Liu, Z., Luo, P., Wang, X., & Tang, X. 2015, in Proceedings of International Conference on Computer Vision (ICCV)

Long, J., Shelhamer, E., & Darrell, T. 2015, in Proceedings of the IEEE conference on computer vision and pattern recognition

Louizos, C., Welling, M., & Kingma, D. P. 2017, arXiv preprint

Louizos, C., Welling, M., & Kingma, D. P. 2018, in International Conference on Learning Representations (ICLR)

Lundberg, S. M., & Lee, S.-I. 2017, in NIPS

Luo, Y., Boix, X., Roig, G., Poggio, T., & Zhao, Q. 2015a, Foveation-based Mechanisms Alleviate Adversarial Examples

——. 2015b, arXiv preprint arXiv:1511.06292

Lyu, C., Huang, K., & Liang, H.-N. 2015a, 2015 IEEE International Conference on Data Mining

Lyu, C., Huang, K., & Liang, H.-N. 2015b, in 2015 IEEE international conference on data mining

M. Nieuwenhuis, e. a. 2012, Neuroimage

M. R. Arbabshirani, e. a. 2013, Front Neurosci-Switz

Maddison, C. J., Mnih, A., & Teh, Y. W. 2017, in International Conference on Learning Representations (ICLR)

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. 2018, in International Conference on Machine Learning

Marchini, J., & Howie, B. 2010, Nat Rev Genet

Meng, D., & Chen, H. 2017, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security

Milz, S., Arbeiter, G., Witt, C., Abdallah, B., & Yogamani, S. 2018, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 247–257

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. 2016, in International conference on machine learning

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. 2013, arXiv preprint arXiv:1312.5602

Mogelmose, A., Trivedi, M. M., & Moeslund, T. B. 2012, IEEE Transactions on Intelligent Transportation Systems

Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., & Frossard, P. 2017, in Proceedings of the IEEE conference on computer vision and pattern recognition

Moosavi-Dezfooli, S.-M., Fawzi, A., & Frossard, P. 2016a, in CVPR

Moosavi-Dezfooli, S.-M., Fawzi, A., & Frossard, P. 2016b, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2574–2582

Murray, A., & Edwards, P. 1992, Advances in neural information processing systems

Nakanishi, K., ichi Maeda, S., Miyato, T., & Okanohara, D. 2018, in Asian Conference on Computer Vision (ACCV)

Naseer, M., Khan, S., & Porikli, F. 2019, in 2019 IEEE Winter Conference on Applications of Computer Vision (WACV)

Nayebi, A., & Ganguli, S. 2017, arXiv preprint arXiv:1703.09202

Nguyen, L., Wang, S., & Sinha, A. 2017, A Learning and Masking Approach to Secure

Learning

P. M. Thompson, e. a. 2001, P Natl Acad Sci USA

Papandreou, G., Chen, L.-C., Murphy, K. P., & Yuille, A. L. 2015, in Proceedings of the IEEE international conference on computer vision

Papernot, N., McDaniel, P., & Goodfellow, I. 2016a, arXiv preprint arXiv:1605.07277

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., & Swami, A. 2017, Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security - ASIA CCS '17

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., & Swami, A. 2016b, 2016 IEEE European Symposium on Security and Privacy (EuroSP)

Papernot, N., McDaniel, P., Wu, X., Jha, S., & Swami, A. 2016c, in 2016 IEEE Symposium on Security and Privacy (SP), IEEE, 582–597

Parkhi, O. M., Vedaldi, A., & Zisserman, A. 2015, BMVC

Pinheiro, P. O., & Collobert, R. 2015, in Proceedings of the IEEE conference on computer vision and pattern recognition

Poursaeed, O., Katsman, I., Gao, B., & Belongie, S. 2018, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition

Prakash, A., Moran, N., Garber, S., Dilillo, A., & Storer, J. 2017, 2017 Data Compression Conference (DCC)

Raffin, A., Kober, J., & Stulp, F. 2022, in Conference on Robot Learning

Reddy Mopuri, K., Ojha, U., Garg, U., & Venkatesh Babu, R. 2018, in Proceedings of the

IEEE Conference on Computer Vision and Pattern Recognition

Rezende, D. J., Mohamed, S., & Wierstra, D. 2014, in ICML

Ribeiro, M. T., Singh, S., & Guestrin, C. 2016, in Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 1135–1144

Rippel, O., & Bourdev, L. 2017, in International Conference on Machine Learning (ICML)

Ross, A. S., & Doshi-Velez, F. 2017, Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing their Input Gradients

S. G. Liang, e. a. 2019, Schizophrenia Bull

S. Kelly, e. a. 2018, Mol Psychiatr

S. M. Plis, e. a. 2014a, Front Neurosci-Switz

——. 2014b, Front Neurosci-Switz

S. Purcell, e. a. 2007, Am J Hum Genet

S. Ripke, e. a. 2014, Nature

Samangouei, P., Kabkab, M., & Chellappa, R. 2018, in ICLR

Sayood, K. 2002, Lossless compression handbook (Elsevier)

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. 2015, in International conference on machine learning

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. 2017, arXiv preprint arXiv:1707.06347

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. 2017, in IEEE International Conference on Computer Vision (CVPR), 618–626

Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., et al. 2012, IEEE Signal processing magazine

Shaham, U., Yamada, Y., & Negahban, S. 2018, Neurocomputing

Sharif, M., Bhagavatula, S., Bauer, L., & Reiter, M. K. 2016, in Proceedings of the 2016 acm sigsac conference on computer and communications security

Shen, Q., Li, Y., Jiang, H., Wang, Z., & Zhao, T. 2020, in International Conference on Machine Learning

Shrikumar, A., Greenside, P., & Kundaje, A. 2017, in ICML

Silver, D. et al. 2016, nature

——. 2017, nature

Simonyan, K., Vedaldi, A., & Zisserman, A. 2013, arXiv preprint arXiv:1312.6034

Simonyan, K., & Zisserman, A. 2014, arXiv preprint arXiv:1409.1556

some string reflecting how you wish the entry alphabetized. 2019, White Paper: Cisco Visual Networking Index: Forecast and Trends, 2017–2022, Tech. rep.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014, The journal of machine learning research

Sun, X., Zhang, Z., Ren, X., Luo, R., & Li, L. 2021, in Proceedings of the AAAI Conference on Artificial Intelligence

Sutskever, I., Vinyals, O., & Le, Q. V. 2014, in NIPS

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. 2013, arXiv preprint arXiv:1312.6199

T. F. D. Farrow, e. a. 2005, Biol Psychiat

T. Insel, e. a. 2010, Am J Psychiat

Tai, L., Paolo, G., & Liu, M. 2017, in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Tai, L., Zhang, J., Liu, M., & Burgard, W. 2018, in 2018 IEEE International Conference on Robotics and Automation (ICRA)

Talpaert, V., Sobh, I., Kiran, B. R., Mannion, P., Yogamani, S., El-Sallab, A., & Perez, P. 2019, arXiv preprint arXiv:1901.01536

Theis, L., Shi, W., Cunningham, A., & Huszár, F. 2017, arXiv preprint arXiv:1703.00395

Toderici, G., O'Malley, S. M., Hwang, S. J., Vincent, D., Minnen, D., Baluja, S., Covell, M., & Sukthankar, R. 2016, in International Conference on Learning Representations (ICLR)

Toderici, G., Vincent, D., Johnston, N., Jin Hwang, S., Minnen, D., Shor, J., & Covell, M. 2017, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 5306–5314

Todorov, E., Erez, T., & Tassa, Y. 2012, in 2012 IEEE/RSJ international conference on intelligent robots and systems

Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., & Sohl-Dickstein, J. 2017, in NIPS

V. D. Calhoun, e. a. 2001, Hum Brain Mapp

Vinyals, O. et al. 2019, Nature

Vorobeychik, Y., & Kantarcioglu, M. 2018, Adversarial Machine Learning (Morgan & Claypool)

W. Z. Yan, e. a. 2019, Ebiomedicine

Wallace, G. K. 1992, IEEE transactions on consumer electronics, 38

Wang, H., Chen, T., Gui, S., Hu, T.-K., Liu, J., & Wang, Z. 2020, Advances in neural information processing systems

Wang, X., Chen, W., Wu, J., Wang, Y.-F., & Wang, W. Y. 2018, in Proceedings of the IEEE conference on computer vision and pattern recognition

Williams, R. J. 1992, Machine Learning, 8, 229

Woodward, N. D., Rogers, B., & Heckers, S. 2011, Schizophr Res

Wu, L., Tian, F., Qin, T., Lai, J., & Liu, T.-Y. 2018, arXiv preprint arXiv:1808.08866

Wu, T., Tong, L., & Vorobeychik, Y. 2019, in International Conference on Learning Representations

X. L. Cai, e. a. 2020, Hum Brain Mapp

Xiao, C., Li, B., Zhu, J.-Y., He, W., Liu, M., & Song, D. 2018, in IJCAI

Xiao, H., Rasul, K., & Vollgraf, R. 2017

Xu, W., Evans, D., & Qi, Y. 2018, Proceedings 2018 Network and Distributed System Security Symposium

Yang, C., Kortylewski, A., Xie, C., Cao, Y., & Yuille, A. 2020, in European Conference on Computer Vision, Springer, 681–698

Yang, C., Rangarajan, A., & Ranka, S. 2018, in IEEE International Conference on Data Science and Systems (DSS)

Yang, X., & Ji, S. 2020, in International Conference on Pattern Recognition (ICPR)

Yang, X., Wei, F., & Zhang, H. 2019, ECCV

Yuan, X., He, P., Zhu, Q., & Li, X. 2017, arXiv preprint arXiv:1712.07107

Zeiler, M. D., & Fergus, R. 2014, in European conference on computer vision (ECCV)

Zhelo, O., Zhang, J., Tai, L., Liu, M., & Burgard, W. 2018, arXiv preprint arXiv:1804.00456

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N. J., Xie, X., & Li, Z. 2018, in Proceedings of the 2018 world wide web conference

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. 2016, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. 2017, in Computer Vision (ICCV), 2017 IEEE International Conference on