Domain Knowledge Guided Testing and Training of Neural Networks

by

Vineel Nagisetty

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2021

Abstract

The extensive impact of Deep Neural Networks (DNNs) on various industrial applications and research areas within the last decade can not be overstated. However, they are also subject to notable limitations, namely their vulnerability to various forms of security attacks and their need for excessive data - especially for particular types of DNNs such as generative adversarial networks (GANs). Tackling the former challenge, researchers have proposed several testing, analysis and verification (TAV) methods for DNNs. However, current state-of-the-art DNN TAV methods are either not scalable to industrial sized DNNs or are not expressible (i.e. can not test DNNs for a rich set of properties). On the other hand, making GANs more data-efficient is an open area of research, and can potentially lead to improvements in training time and costs.

In this work, I address these issues by leveraging domain knowledge - task specific knowledge provided as an additional source of information - in order to better test and train DNNs. In particular, I present Constrained Gradient Descent (CGD), a novel algorithm (and a resultant tool called CGDTest) that leverages domain knowledge (in the form of logical constraints) to create a DNN TAV method that is both scalable and expressible. Further, I introduce a novel gradient descent method (and a resultant GAN referred to as xAI-GAN) that leverages domain knowledge (provided in the form of neuron importance) to train GANs to be more data-efficient. Through empirical evaluation, I show that both tools improve over current state-of-the-art methods in their respective applications. This thesis highlights the potential of leveraging domain knowledge to mitigate DNN weaknesses and paves the way for further research in this area.

Acknowledgements

I would like to acknowledge the following mentors and friends, without whom this thesis would not have been possible:

1. First, I would like to thank my supervisor, Dr. Vijay Ganesh for his mentorship and guidance. His enthusiasm for research is infectious and his focus on clarity of thought to understand a topic at a very fundamental level is inspiring. Thank you so much Vijay for giving me the opportunity to pursue my degree under your supervision, and for inspiring me to achieve more than what I thought possible.

2. Laura Graves, for helping and collaborating with me these last five years, both during undergraduate and graduate studies. Her work ethic, ability to identify interesting research ideas and dedication to stay focused on the research despite any setbacks have inspired me over the years.

3. Dr. Christopher Srinivasa, for giving me the opportunity to do an internship at Borealis AI, and for his mentorship and guidance throughout it. Your ability to succinctly summarize complex ideas is extraordinary. I would also like to thank his team at Borealis AI for providing research guidance.

4. Dr. Antonina Kolokolova, for inspiring me to pursue research, and for giving me the opportunity to attend various conferences and start on this path.

5. Dr. Florian Kerschbaum, for his guidance throughout my degree. His amazing course on the security of AI helped me learn the ropes and love this field of research. I highly value your insightful feedback on my papers over these years, and appreciate you agreeing to be a reader of this thesis.

6. Dr. Mark Crowley, thank you for agreeing to be a reader on my thesis. I really appreciate your time and knowledgeable feedback.

7. Dhananjay Ashok and Joseph Scott for collaborating with me on research over the years and for motivating me to continue this work.

8. Past and present members of Dr. Vijay Ganesh's research group (Laura Graves, Joseph Scott, Dhananjay Ashok, Ian Li, Saeed Nejati, Behkish Nassirzadeh, Soham Mukherjee, Alaina Mahalanabis, and Murphy Berzish) for their help. It's unfortunate that our in-person collaboration was limited due to the pandemic, but I am thankful for all your help motivating me to keep going forward.

Dedication

This thesis is dedicated to my partner, Hayley Roberts, for her support, advice and faith. I am incredibly blessed to have you by my side.

Table of Contents

List of Figures **ix**

List of Tables **x**

1 Introduction **1**

1.1 Related Work 3

1.2 Contributions: 4

1.3 Thesis Organization: 5

2 Background **6**

2.1 How Neural Networks Learn: Gradient Descent 6

2.2 Generative Adversarial Networks (GANs) 7

2.3 Domain Knowledge 8

2.3.1 Domain Knowledge as Logical Constraints 8

2.3.2 Domain Knowledge as Neuron Importance 11

3 Constrained Gradient Descent Algorithm for Testing Deep Neural Networks **14**

3.1 Introduction 14

3.2 Additional Background 16

3.2.1 The DNN-SAT Problem 16

3.2.2 Adversarial Examples and the FGSM Method . . . 17
3.3 Constrained Gradient Descent (CGD) Algorithm . . . 18
3.4 Experimental Evaluation of the CGDTest Tool . . . 20
3.4.1 Qualitative Comparison of DNN TAV Tools . . . 20
3.4.2 Benchmark Suites . . . 22
3.4.3 Experimental Results . . . 23
3.4.4 Analysis of Results . . . 25
3.5 Related Work . . . 26
3.6 Conclusions and Future Work . . . 27

4 xAI-GAN: Enhancing Generative Adversarial Networks via Explainable AI Systems 28
4.1 Introduction . . . 28
4.1.1 An Overview of xAI-GAN . . . 30
4.1.2 Contributions . . . 31
4.2 Detailed Overview of xAI-GAN Systems . . . 32
4.2.1 Generator Training in Standard GANs . . . 33
4.2.2 xAI-guided Generator Training in xAI-GANs . . . 34
4.2.3 xAI-GAN Implementation Details . . . 34
4.3 Experimental Results . . . 35
4.3.1 Experimental Setup . . . 35
4.3.2 Evaluation Criteria . . . 35
4.3.3 Results on MNIST and Fashion MNIST . . . 36
4.3.4 Results on CIFAR10 Dataset . . . 37
4.3.5 Discussion of Experimental Results . . . 38
4.4 Related Work . . . 39
4.5 Conclusion and Future Work . . . 39
4.5.1 Controlling How Models Learn . . . 40

5 Conclusion **44**

5.1 Future Work 45

References **47**

List of Figures

4.1 FID scores on the same GAN architecture trained on different amounts of data. 29

4.2 System Architecture of xAI-GAN. 30

4.3 FID scores on MNIST Dataset 41

4.4 Sample of Images Generated by the xAI-GAN$_{lime}$ System using 35% Data on the MNIST Dataset 41

4.5 FID scores on Fashion MNIST Dataset 42

4.6 Sample of Images Generated by the xAI-GAN$_{shap}$ System using 35% Data on the Fashion MNIST Dataset 42

4.7 FID scores on the CIFAR10 Dataset 43

4.8 Sample of Images Generated by the xAI-GAN$_{saliency}$ System using 20% Data on CIFAR10 Dataset 43

List of Tables

3.1 Architecture of the CNN Model used in the Defended CNN benchmark . . 22

3.2 Results of all three experiments for the listed TAV tools on the three benchmarks. A * indicates that the tool was not able to test for the given type of adversarial example. 24

4.1 Model Architecture of Fully Connected GAN used in the context of MNIST and Fashion MNIST experiments. 33

4.2 Model Architecture of DC-GAN used in the context of the CIFAR10 experiments. 33

4.3 Average experiment time taken (in seconds) on MNIST and Fashion MNIST Datasets 36

4.4 Average experiment time taken (in seconds) by each GAN on MNIST and Fashion MNIST (FMNIST) datasets. 36

Chapter 1

Introduction

It is no exaggeration to claim that Deep Neural Networks (DNNs) have transformed many industries and research areas within the last decade, impacting several sub-fields of AI, computer science, engineering and even basic sciences [42, 59, 117]. This success has prompted industry practitioners and machine learning researchers to propose their use in various domains such as natural language processing [90], computer vision [130], and as autonomous vehicle controllers [47]. Their impact is so great that it is hard to find an industry where the use of DNNs has not been proposed and/or researched.

However, DNNs also present many unique challenges. For example, they pose security concerns since they are vulnerable to several kinds of attacks such as adversarial, data-poisoning, membership inference, and model inversion attacks [22, 120, 103, 54]. Further, DNNs can require a lot of data for training, acquiring which can be time consuming and expensive [126]. Additionally, DNNs present various interpretability and explainability challenges. Despite their considerable success, it is still unclear why DNNs are as effective as they are [14, 51, 127].

Recognizing the security vulnerabilities of DNNs, researchers have proposed a variety of testing, analysis and verification (TAV) methods tailored to them [62, 87, 160]. These methods can be broadly classified as testing, verification/program analysis and reachability methods. **Testing** methods aim to find inputs that satisfy some constraints on the given DNN [142]. **Verification/program analysis** methods, on the other hand, symbolically "capture" the model and often use a SAT or an SMT solver to either find inputs that satisfy the constraints or prove that the DNN adheres to the given properties [72, 70]. Finally, **reachability** methods define an allowable input region and propagate that region through the network, resulting in an output domain that encapsulates all possible outputs given any input described in the input region [87, 137].

DNN testing methods can scale very well and place no restrictions on the type of DNN analyzed. However, they lack the ability to test for a rich set of security properties (such as for different forms of adversarial robustness) and do not provide any certificates of guarantee. Verification methods are very useful when a user requires a certificate of guarantee and allow for rich set of properties to be specified, but place restrictions on the kinds of DNNs that can be analyzed and their scalability is poor at best. Reachability methods scale better than verification methods and are general, but score low on testing for a rich set of properties and certificates of guarantee measures (see Chapter 3 for more details on DNN TAV methods and their limitations). In many industrial and academic settings, model developers demand a method that is scalable to millions of parameters and is suitable for many architectures, while allowing to check for many types of DNN properties[1]. As a consequence, there is a need to find such a DNN TAV method that is scalable to these industrial sized DNNs while posing no restrictions on the DNN architecture and allowing to test a rich set of properties.

Further, DNNs can require lots of data based on their task. The collection of high-quality labelled data often requires a lot of resources such as time and money and is considered a 'major bottleneck' in machine learning research [126]. Notably, generative adversarial networks (GANs), a type of machine learning model that uses two DNNs in order to generate data that is indistinguishable from a target distribution, is notorious for requiring a lot of data for successful training [162]. As a result, researchers have recently looked at ways to make GANs more data-efficient [162, 25, 158]. However, while these methods are promising, there is still a need to find other ways of making GANs (and DNNs in general) more data-efficient (see Chapter 4 for more information).

A potential way to address both of the above DNN limitations is by leveraging domain knowledge, for example, in the form of logical constraints. While knowledge of a domain is indirectly used in DNNs (and machine learning in general) by annotating data or selecting features, domain knowledge by contrast refers to task-specific knowledge that is provided (e.g. in the form of logical constraints) as an additional source of information to the DNN model [149, 15]. This knowledge can be sourced either from a subject matter (human) expert or other software systems. An example is [129] where domain knowledge in the form of mathematical equations is used to create new labelled data.

[1]Note that, while being able to provide certificate of guarantees is useful, finding an incomplete DNN TAV method with the properties listed above is of tremendous value for both industry and research use.

Problem Statement:

In this thesis I address the following question, *can domain knowledge be leveraged to better test and train deep neural networks (DNNs)?*

More specifically, recent works suggest leveraging domain knowledge can help improve standard DNN training [140, 52]. Can domain knowledge be leveraged in order to create a DNN TAV method that improves over current state-of-the-art methods in terms of scalability and expressibility? Further, standard methods of leveraging domain knowledge to DNNs may not work for particular kinds of DNNs, such as generative adversarial networks (GANs), since GANs are a collection of two separate DNNs with an adversarial relationship (see Section 2.2 for more information). Can domain knowledge be leveraged to these types of DNNs to improve their training?

1.1 Related Work

There are several recent research works that investigate methods of injecting domain knowledge in DNNs [140, 52, 149, 15]. In many cases, domain knowledge is provided as physical simulations or equations and used to provide more annotated data [140, 149]. In other cases, domain knowledge is leveraged by converting to loss functions [36], regularization terms [30] or injecting into the DNN architecture [40]. Regardless, most methods serve to inject domain knowledge to standard DNN training. By contrast, I look at ways of leveraging domain knowledge to test DNNs, and train GANs - a particular type of DNN.

Note that leveraging domain knowledge is not exclusive to DNNs. In fact, there is a greater amount of research investigating ways to leverage domain knowledge in other machine learning models [149, 29, 83, 74]. Examples include integrating domain knowledge in support vector machines (SVMs) [83] and k-nearest neighbour (KNN) [74] models.

This work can be broadly positioned in the areas of research that aim to incorporate additional capability in Machine Learning. At the forefront of this is the field of Neurosymbolic AI, which aims to combine symbolic reasoning with learning capabilities [6, 27, 58]. Another notable line of research aims to integrate special layers in DNNs that can solve more complex problems, often using optimization solvers [73, 121]. These fields are in their early stages and are very active areas of research.

1.2 Contributions:

In summary, I show in this thesis that domain knowledge can be leveraged in order to better test and train DNNs (in particular, training GANs). More specifically, the contributions of this thesis are the following:

1. **Constrained Gradient Descent (CGD) Algorithm and its Implementation CGDTest Tool:** a novel algorithm (and a resultant tool called CGDTest) that leverages domain knowledge given as logical constraints, with the goal of checking whether DNN satisfy these user-specified logical constraints. I report on detailed experiments where I empirically evaluate the efficiency of aforementioned tools over three benchmark suites of 26 models with sizes ranging from 6000 to 11 million parameters (or number of neurons ranging from over 1000 to over 1.5 million) as measured by PAR2 score [56]. I show that the CGDTest tool is the most successful in producing the kinds of inputs as specified by standard definitions of adversarial examples, while other methods failed and scored much lower, either because they couldn't scale to some of the large DNNs or didn't allow properties to be specified or placed restrictions on the kinds of DNNs they could analyze. Further, I use two other notions of adversarial examples and show that CGDTest can test for constraints that other state-of-the-art tools cannot (Please see Chapter 3.)

2. **xAI-guided Gradient Descent Algorithm and xAI-GAN:** a novel gradient descent method (and a resultant GAN referred to as xAI-GAN) that utilizes xAI systems to focus the gradient descent algorithm on weights that are determined to be most influential by the xAI system. I implement several versions of xAI-GAN using different xAI systems. I perform experiments to evaluate the quality (as measured by Fréchet Inception Distance, abbreviated as FID [57]) of xAI-GANs relative to standard GANs, show that on MNIST and Fashion MNIST datasets, xAI-GANs achieve an improvement in FID score compared to standard GANs. Further, I extend the experiment to the CIFAR 10 dataset, using only 20% of the data for xAI-GAN while letting standard GAN use 100% of the data and show that xAI-GAN outperforms standard GAN in FID score even in this setting. I further compare xAI-GAN with Differentiable Augmentation [162] technique which has been shown to improve data-efficiency of GANs and show that xAI-GAN outperforms Differentiable Augmentation, resulting in a better FID score.

In both methods, I provide novel algorithms that leverage domain knowledge to better test and train DNNs. Note that while the xAI-guided gradient descent process is applied in the context of GANs, it can also be used to inject domain knowledge to standard DNNs.

1.3 Thesis Organization:

The rest of the thesis is organized as follows:

1. Chapter two covers **background** information necessary to understand the rest of the material covered. In particular, I provide information on gradient descent, the process by which DNNs learn. I then provide more details on the GAN architecture. Further, I cover the different forms of domain knowledge used in this work (logical constraints and explainable AI systems), and show how they are pre-processed.

2. Chapter three covers **constrained gradient descent (CGD)**, a novel DNN TAV method that addresses the most important concerns of model developers, i.e., a testing tool that scales to industrial-strength DNNs, is architecture-neutral, and most importantly enables them to specify and generate inputs to test a large number of DNN security and reliability properties. In this chapter, I provide an overview on the various DNN TAV methods before identifying the need for a better method. This is followed by a detailed information on the CGD algorithm that leverages domain knowledge to better test DNNs. I compare the implementation of this algorithm - CGDTest tool - with 9 state-of-the-art testing, verification/program analysis, and reachability methods along several vectors and show that it is closes the gap identified as it produces an improvement (in PAR2 score [56]) of over 1500% over the next best method in some cases. I then provide a detailed overview on related work before concluding the chapter and identifying potential future work.

3. Chapter four covers **xAI-GAN**, a novel method of injecting domain-knowledge (sourced from explainable AI systems) to make Generative Adversarial Networks (GANs) more data-efficient. In this chapter, I present a detailed overview on this method - termed xAI-guided gradient descent and contrast it against standard GAN training. I then report on experiments that evaluated the quality (as measured by FID [57]) of xAI-GANs relative to standard GANs and show that xAI-GANs outperforms standard GAN on the FID scores - especially when there is less data available. I then provide a summary of related work and contrast them with this method. In Section 4.5.1, I discuss how xAI-guided gradient descent methods can allow for greater control over the GAN training process before concluding this chapter.

4. Finally, chapter five **concludes** the thesis, summarizing the contributions of this work and contextualizing it with related research work. I also identify future research that can span from this line of work and describe it's potential benefits to both industry and academic settings.

Chapter 2

Background

2.1 How Neural Networks Learn: Gradient Descent

During the DNN training process, gradient descent[1] is used to fit the weights and bias parameters of the DNN to minimize a given loss function. This loss (or cost) function estimates the difference between the predicted label y' and the ground truth y, given batches of examples and labels. Common loss functions include the mean squared error function, the root mean squared error function (or ℓ_2 loss), and the cross-entropy loss function. A crucial property of loss functions is that they should be differentiable, which allows to easily calculate the gradient of loss with respect to the weights and biases.

More specifically, during the forward pass, a batch of example(s) x is provided to the neural network model M with weight and bias parameters θ_M. M propagates this batch through its layers to calculate the predicted label(s) y'. The loss between the ground truth and predicted labels $L(y', y)$ is then used to find the gradient of the loss with respect to the model parameters $\Delta_{\frac{\partial L}{\partial \theta^t_M}}$. Note that the gradient of the weights and parameters are calculated for a given layer l (denoted by $\theta_{M,l}$) using the gradient of the subsequent layer $\theta_{M,l+1}$. Once the gradients are calculated for all the layers, the model parameters are updated using a small learning rate α:

$$\theta^{t+1}_M = \theta^t_M - \alpha * \Delta_{\frac{\partial L}{\partial \theta^t_M}}$$

[1]Note that DNNs use back-propagation to efficiently calculate the gradients, and updating the parameters using this gradient information is usually considered as a separate process. Here I use the terms gradient descent and back-propagation inter-changeably.

This is the process that needs to be modified modify in order to inject domain knowledge to test and train DNNs. In the context of constrained gradient descent (CGD), domain knowledge is provided as logical constraints by adding it to the loss function(refer to 2.3.1). In the case of xAI-GAN, domain knowledge is given in the form of neuron importance matrix by modifying the DNN parameter update described above (refer to 2.3.2).

2.2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a type of DNNs that generate data which is *indistinguishable* from a target distribution. While GANs were introduced only a few years ago, they have been very successfully applied to generate many kinds of data such as text, images, music and other forms of data [113]. Briefly, standard GAN architectures consist of a system of paired DNNs, namely, a discriminator D and a generator G. A common analogy for GAN training is that of a counterfeiter (the generator G) and a detective (the discriminator D) playing an adversarial game where the counterfeiter makes a fake and the detective tries to tell if it's real or not. Over the training process, the detective and counterfeiter both get better at their jobs, with the end goal being that the counterfeiter is so proficient that their fakes can pass for the real thing.

In practice, the standard GAN training method involves alternate cycles of discriminator and generator training. During its training, the discriminator is provided a batch of examples drawn from both training data from the target distribution, as well as data generated by the generator (which initially is expected to be just noise). These examples are correctly labelled as "real" (if they were from the training set) or "generated" (if they are from the generator). The parameters of discriminator are modified so as to minimize this loss so that the discriminator can better distinguish real data from the generator output. By contrast, the generator is trained by providing noise (a vector randomly sampled from a prior distribution). The resultant output from the generator is given to the discriminator, and loss is calculated by comparing the discriminators prediction on this data with the "real" label. The parameters of generator are then modified so as to minimize this loss. The ideal termination condition for this process is when the generator produces high-quality data and the discriminator is unable to distinguish between "real" and "generated" examples[2].

[2]While such termination is ideal, the system can terminate due to other conditions such as in mode-collapse where the generator loses diversity, mapping all inputs to a small region of the feature space.

2.3 Domain Knowledge

Domain knowledge can be described as task-specific knowledge that is provided as an extra source of information to the DNN [15]. Note that this can be explicit information that is encapsulated by the examples provided to the DNN model, but it does not have to be. Different sources of domain knowledge include information provided by subject matter (human) experts, explainable AI (xAI), simulation software, or physics equations [140, 52]. Domain knowledge can be provided in various forms [149]. In this work, I focus on domain knowledge that is provided in the form of logical constraints (that are differentiable) as well as provided as neuron importance and describe methods to inject this domain knowledge to the gradient descent process of DNNs. Note that the methods described are independent of the source of the domain knowledge and only depend on its format.

2.3.1 Domain Knowledge as Logical Constraints

In this subsection, I describe how to inject domain knowledge provided as logical constraints to gradient descent in order to test whether a DNN satisfies the constraints. This is done by first converting these logical constraints differentiable loss functions, as described below.

Constraint Relaxation

In traditional logic, each constraint (or rule) is mapped to either 1 (true) and 0 (false) truth value. It is hard to capture uncertainty in this setting - a trait very important to many AI problems. Further, solving in traditional logic can be computationally intractable. By contrast, probabilistic soft logic (PSL) [75] - a framework for collective, probabilistic reasoning in relational domains - uses soft truth values in the interval [0,1]. This allows inference in this setting to be a continuous optimization problem, which can be solved efficiently. Additionally, one can encode similarity functions and other relationships as constraints in this domain.

In order to determine the degree that a constraint is satisfied, PSL uses the *Lukasiewicz t-norm* and its corresponding *co-norm* as the relaxation of logical connectives. These relaxations, as well as the truth values of the constraints, are designed to be exact at the extremes ('fully true' or 'fully false') - and more importantly - provide a consistent mapping for values in between. This allows to calculate smooth gradient information, which helps better update parameters.

The Language of Constraints

Similar to [36], the language of constraints in this setting consists of boolean combinations of comparisons between terms, where each term is either a constant or a differentiable function. More formally, the language of constraints can be given as follows:

- $\varphi \Rightarrow \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid$ comparison
- comparison $\Rightarrow t \leq t \mid t = t \mid t \neq t \mid t < t$
- $t \Rightarrow$ constant | differentiable function

Each term t can be a constant or a real-valued differentiable function that is defined over inputs, parameters (such as intermediate layer outputs) and outputs of the DNN. For the implementation, I included a number of popular differentiable functions (e.g. ℓ_∞ distance), which can be extended to more functions, as long as they are differentiable.

Translating Constraints to Differentiable Functions

The constraints are mapped to their equivalent differentiable functions using a recursive program. The mapping is as follows:

1. $\mathcal{L}(t \leq t') := max(t - t', 0)$,
2. $\mathcal{L}(t \neq t') := \epsilon * [t = t']$,
3. $\mathcal{L}(t = t') := \mathcal{L}(t \leq t' \wedge t' \leq t)$,
4. $\mathcal{L}(t < t') := \mathcal{L}(t \leq t' \wedge t \neq t')$.

The boolean combination of constraints are converted as follows:

1. $L(\varphi' \wedge \varphi'') := L(\varphi') + L(\varphi'')$,
2. $L(\varphi' \vee \varphi'') := L(\varphi') * L(\varphi'')$.

If φ is a negation $\neg\varphi'$, it is rewritten to a logically equivalent constraint before translation to loss as follows:

1. $\mathcal{L}(\neg(t = t')) := \mathcal{L}(t \neq t')$,
2. $\mathcal{L}(\neg(t \leq t')) := \mathcal{L}(t' < t)$,
3. $\mathcal{L}(\neg(t \neq t')) := \mathcal{L}(t = t')$,
4. $\mathcal{L}(\neg(t < t')) := \mathcal{L}(t' \leq t)$,
5. $\mathcal{L}(\neg(\varphi' \wedge \varphi'')) := \mathcal{L}(\varphi' \vee \varphi'')$,
6. $\mathcal{L}(\neg(\varphi' \vee \varphi'')) := \mathcal{L}(\varphi' \wedge \varphi'')$,
7. $\mathcal{L}(\neg(\neg\varphi')) := \mathcal{L}(\varphi')$.

Converting Constraints to Loss

The aforementioned boolean combination of comparison constraints are converted to a loss form recursively using the translations described above. This differentiable loss function is referred to as 'constraint loss'. This approach results in constraint loss with the following necessary properties:

1. the loss is zero exactly if the constraints are satisfied, and
2. the loss is differentiable everywhere.

Although gradient descent is the primary way in which DNNs learn, it can also be useful for other purposes. For example, a very common technique in DNN training is ℓ_2 *regularization*, a technique that adds the ℓ_2 norm of the set of weights to the loss function. During the learning process the weights and biases are optimized to both minimize the classification loss term, but also minimize the weights of the network. In such a way, regularization can be thought of as another "property" that needs to be respected and can be encoded using φ, similar to the adversarial robustness property definition above. This technique utilizes this concept of optimizing the network over any differentiable function (or combination of differentiable functions) to encode a set of constraints as differentiable functions and perform gradient descent over those constraints - a technique termed *constrained gradient descent*.

Primarily, the gradient of loss with respect to the features of the example is calculated using x: $\Delta_{\frac{\partial L}{\partial x}}$. This allows to alter the features in the direction of increased or decreased loss. When combined with the previous conception of performing gradient descent over any differentiable function, the degree to which to alter the features of an input can be calculated in order to increase or decrease the value of any differentiable function with regard to those features. One can encode a set of constraints as differentiable functions and perform gradient descent to alter an input in the direction that minimizes constraint violations. This is a crucial concept to the CGD method, which uses both loss and constraints to attempt to find counterexamples that violate desired properties. In order to use this constraint loss to test DNNs for property violations, I modify an example along the gradient of reduced constraint loss, and in this manner test for the existence of examples that satisfy desired properties.

2.3.2 Domain Knowledge as Neuron Importance

In this subsection, I describe how to inject domain knowledge provided as neuron importance to gradient descent so as to better train DNNs. Here, I use an explainable AI (xAI) system in order to source this domain knowledge. Note that xAI systems provide feature importance, which in the case of GANs can be considered as neuron importance for the generator (please see Chapter 4.2). This is not universal. However, any domain knowledge (sourced from a software system or human expert) that provides neuron importance information as a matrix format can be used by this method to inject into the gradient descent process (described below).

Explainable AI (xAI) Systems

As AI models become more complex, there is an increasing demand for *interpretability or explainability* of these models from decision makers, stakeholders, and lay users. In addition, one can make a strong case for a scientific need for explainable AI. Consequently, there has been considerable interest in xAI systems aimed at creating interpretable AI models that enable human understanding of AI systems [14].

One way to define xAI systems is as follows: they are algorithms that, given a model and a prediction, assigns values to each feature of an input that measures how important that feature is to the prediction. There have been several different xAI systems applied to DNNs with the goal of improving understanding of how these systems learn. These systems can do so in a variety of ways, and approaches have been developed such as ones using formal logic [63], game-theoretic approaches [93], or gradient descent measures [134]. These systems output explanations in a variety of forms such as ranked lists of features, select subsets of the feature sets, and values weighting the importance of features of input data used to train machine learning models. An overview of xAI systems is given below.

Saliency Map [135]**:** Inspired by the processes by which animals focus attention, saliency maps compute the importance of each feature in a given input to the resulting classification by a DNN model. In order to compute a saliency map, a DNN model M, image x and target label y are required. The loss of the prediction $M(x)$ is computed with respect to y and used to perform backpropagation to the calculate the gradient ∇x. This is then normalized to produce the saliency map. While there are similarities between saliency maps and the process by which the gradients passed by the discriminator to the generator is calculated, there are some differences. Notably, in the case of color images (such as CIFAR10 dataset), saliency map computes the maximum magnitude of ∇x for each pixel across all color channels, while the gradients are computed for each color channel separately.

Lime [125]**:** Short for Local Interpretable Model-Agnostic, LIME is used to explain the predictions of a ML classifier by learning an interpretable local model. Given a DNN model M and input x, Lime creates a set of new N inputs $x_1, ..., x_N$ by slightly perturbing x. It then queries M on these new inputs to generate labels $y_1, ..., y_N$. The new inputs and labels are used to train an interpretable model which is expected to approximate M well in the local vicinity of x. This local model is used to get the feature importance of x.

Feature Permutation [37]**:** This algorithm was originally introduced for random forests, but was modified to create a model-agnostic version. At a high level, it measures the difference in the prediction of a DNN model after permuting the feature's values. A feature is deemed "important" if the perturbation changes prediction confidence. Conversely, it is considered "unimportant" if the perturbation leaves the prediction unchanged. An overview of this algorithm can be found here [106].

DeepSHAP [93]**:** DeepSHAP is a combination of the DeepLIFT platform and *Shapley value explanations*. Introduced in 2017, the platform is well-suited for neural network applications and is freely available. DeepSHAP is an efficient Shapley value estimation algorithm. It uses linear composition rules and backpropagation to calculate a compositional approximation of feature importance values.

Shapley Value Estimation: Classic Shapley regression values are intended for linear models, where the values represent feature importance. Values are calculated by retraining models on every subset of features $S \subseteq F$ and valuing each feature based on the prediction values on models with that feature and without. Unfortunately, this method not only requires significant retraining but also requires at least $2^{|F|}$ separate models to cover all combinations of included features. Methods to approximate the Shapley values by iterating only over local feature regions, approximating importance using samples from the training dataset, and other approaches have been proposed to reduce computational effort.

The DeepLIFT xAI system: DeepLIFT uses a set of reference inputs and the consequent model outputs to identify the importance of features [134]. The difference between an output and a reference output, denoted by Δy, is explained in terms of the differences between the corresponding input and a reference input, given by Δx_i. The reference input is chosen by the user based on domain knowledge to represent a typical uninformed state. It is often a set of images from the original dataset that the model is trained on. Each feature x_i is given a value $C_{\Delta x_i \Delta y}$ which measures the effect of the model output on that feature being the reference value instead of its original value. The system uses a summation property where the sum of each feature's changes sum up to the change in the model output Δ_o of the original in comparison to the reference model: $\sum_{i=1}^{n} C_{\Delta x_i \Delta y} = \Delta_o$.

Leveraging Explainable Matrix in Gradient Descent

No matter what xAI system is used, we convert it's output to a matrix E that that encapsulates how important a particular feature is. More specifically, a value of 1 (or close to 1) means that particular feature is very important to the given task, while a value of 0 (or close to 0), represents that the feature is not as important to the given task. Note however, that the efficacy of the training depends on the efficacy of the xAI system and hence selecting the appropriate xAI system is crucial.

This matrix E generated by the xAI system is then used in a modified gradient descent algorithm to update the weights of the DNN by calculating the product (specifically, an element wise or a Hadamard product [60]) between E and the gradient of the particular layer output with respect to the loss $\Delta_{D_{\theta,l}}$. More precisely, the explanation matrix E is used to mask the gradient function, and consequently the "importance of the neurons" are taken into account in the modification of the DNN's weights during the application of the gradient descent.

Using this approach, one can foresee that users might be able to augment such explanation matrices with specifications that spell out relationships (using logical formulas) between the update methods for the various weights of a DNN. I would like to emphasize that the standard gradient descent method simply moves toward the greatest decrease in loss over an n-dimensional space, while by contrast, xAI-guided gradient descent algorithms can give users greater control over the learning process. The reason that this approach works well in training GANs is because there is already a procedure where feedback from the discriminator is provided to the generator(see Section 2.2), and so this can be augmented using the xAI-guided gradient descent. In this context, the matrix E is generated using the discriminator and applied to the last layer of the generator (see Algorithm 2).

Chapter 3

Constrained Gradient Descent Algorithm for Testing Deep Neural Networks

3.1 Introduction

Over the last decade, Deep Neural Networks (DNNs) have become ubiquitous and are being used in a variety of settings from image recognition to safety-critical systems [67, 24, 91]. Simultaneously and unsurprisingly, a plethora of attacks against DNNs have been recently introduced, including adversarial, data-poisoning, membership inference, and model inversion attacks, to name just a few [22, 120, 103]. More broadly, there is a clear recognition in the machine learning as well as in the software engineering, program analysis, and verification communities that the problem of lack of reliability and security of DNNs is of grave concern and needs to be addressed effectively [148]. As a result, researchers have proposed a variety of testing, analysis and verification (TAV), as well as reachability methods that have been specifically tailored to the DNN setting [62, 87, 160]. (An overview of each type of method is provided in Section 3.5.)

We can think of some desirable properties for DNN TAV methods. It should be **scalable** (i.e scale to industrial-sized DNNs that often contain 10's of millions of parameters), **expressible** (i.e allow for *expressing* any property of a DNN), **general** (i.e should not place any *restrictions* on the DNN architecture) and **provide correctness guarantees** (i.e provide a *certificate of guarantee* that asserts that the DNN model adheres to the input property or provide a counterexample of a violation). (Please refer to Section 3.4

for a more detailed discussion of these properties of the some of the leading DNN TAV methods.)

However, none of the above methods can be said to simultaneously possess the properties stated above. Testing methods can scale very well and place no restrictions on the type of DNN analyzed, but lack expressibility and do not provide any certificates of guarantee. Verification methods are very useful when a user requires a certificate of guarantee and allow for rich properties to be specified, but place restrictions on the kinds of DNNs that can be analyzed and their scalability is poor at best. Reachability methods scale well and are general, but score low on expressibility. Further, in the context of DNN verification and program analysis methods, the constraints (aka, specifications) and DNN-under-test are represented symbolically. It is widely accepted [80] that this requirement that DNNs be translated into symbolic form is a major reason for the lack of scalability of verification methods. On the other hand, DNN testing methods typically retain models in their original form, while no additional constraints specifying desired properties are allowed and therefore are limited from an expressibility point of view.

Problem Statement: Hence, the research questions addressed in this chapter are the following: *is it possible to create a hybrid method where model developers can specify properties (symbolically or as programs) as domain knowledge (provided as constraints), without requiring that DNN-under-test be converted into a symbolic form?* If so, *does this result in a testing and analysis framework that can scale to industrial-sized DNNs, while scoring high on the expressibility and generality properties stated above?*

Overview of Constrained Gradient Descent (CGD) Algorithm. To answer the above-stated questions, this chapter proposes a hybrid method, namely the Constrained Gradient Descent (CGD) algorithm, that combines optimization methods [1] with the ability to specify mathematical constraints, with the goal of generating DNN test inputs that satisfy (or alternatively, violate) the specified constraints. Specifically, CGD converts logical constraints into a differentiable loss function (called 'constraint loss') following the approach introduced in Probabilistic Soft Logic [75]. This loss is then minimized using a modified fast gradient sign method (FGSM) [46]. A detailed description is given in Section 3.3. Compared to 9 other state-of-the-art testing, verification, and reachability methods, CGD method finds test inputs that satisfy security constraints (specifically, adversarial inputs that are characterized by three different types of constraints) more efficiently on very large DNN models (i.e., scales better to DNNs with millions of parameters), allows for expressive properties, and generalizes well (i.e., places no restriction on the type of DNNs analyzed). In more detail, I make the following contributions in this chapter.

[1]Here the terms "optimization" and "gradient-descent (GD)" are used interchangeably.

Contributions.

1. **Constrained Gradient Descent (CGD) Algorithm and its Implementation CGDTest Tool:** The key contribution is a constrained gradient descent algorithm (and a resultant tool called CGDTest) that uses logical constraints as part of the loss function of the Fast Gradient Sign Method, with the goal of creating DNN test inputs that satisfy these user-specified logical constraints. This method addresses the most important concerns of model developers, i.e., a testing tool that scales to industrial-strength DNNs, is architecture-neutral, and most importantly enables them to specify and generate inputs to test a large number of DNN security and reliability properties. (Please see Section 3.3.)

2. **Comparison of CGDTest with 9 State-of-the-art Methods:** First, a qualitative comparison of CGDTest tool with state-of-the-art testing, verification/program analysis, and reachability methods along several vectors (refer Section 3.4) is conducted. Further, empirical evaluation of all tools is performed over three benchmark suites of 26 models with sizes ranging from 6000 to 11 million parameters (or number of neurons ranging from over 1000 to over 1.5 million) as measured by PAR2 score [56], and it is shown that the CGDTest tool is the most successful in producing the kinds of inputs as specified by standard definitions of adversarial examples (e.g., produces adversarial inputs for 97.7% of the experiments with an improvement in PAR2 score of over 1500% over the next best method using the Flow constraint definition of adversarial example), while other methods failed and scored much lower, either because they couldn't scale to some of the large DNNs or didn't allow properties to be specified or placed restrictions on the kinds of DNNs they could analyze. Further, two other notions of adversarial examples are used to show that CGDTest can test for constraints that other state-of-the-art tools cannot. (Please see Section 3.4.3.)

3.2 Additional Background

3.2.1 The DNN-SAT Problem

Here I define the problem, dubbed DNN-SAT, that the CGD method is designed to solve.

Definition 3.2.1. ***The DNN-SAT($M, y, \varphi(x)$) decision problem:*** *Given a DNN M, a label y, and specification φ over inputs, outputs and parameters of M, determine whether* *there exists an input to the DNN x s.t $(argmax(M(x)) = y) \wedge \varphi(x)$.* [2]

In other words, the DNN-SAT$(M, y, \varphi(x))$ problem determines whether there exists an input x to the DNN model M that results in a prediction of y (referred to as 'satisfying' the model), where x also satisfies the given logical constraints φ. Note that a better approach may be to re-define DNN-SAT to be an optimization problem since constraint φ may never be completely satisfied [3].

Definition 3.2.2. ***The DNN-SAT(**$M, x, y, \varphi(M, x, y, x', \delta)$**) adversarial robustness problem:*** *Given a DNN M, an input x, a label y, and specification φ over inputs to M, determine whether there exists a x' s.t $(argmax(M(x)) = y) \wedge \varphi(M, x, y, x', \delta)$, where $\varphi(M, x, y, x', \delta)$ represents $(argmax(M(x')) \neq y) \wedge (dist(x, x') < \delta)$ and $dist(x, x')$ computes a suitable similarity distance function (e.g ℓ_∞ distance). (While I specify only distance measure here, this method allows for other types of constraints besides distance or similarity measures.)*

The DNN-SAT adversarial robustness problem stated above checks whether the model M is adversarially robust within a δ range around input x. In other words, for a given input x, label y, this problem checks whether there exists an adversarial example x' such that the distance between x and x' is less than a given δ while M classifies x' as a different label compared to x. Note that the constraints φ could have additional restrictions. For example, one might wish to determine whether the adversarial input x' possesses certain qualities - such as containing features within a certain range. In such cases, these properties can be added to the encoding of φ. All of the DNN TAV methods listed in this chapter can be thought of as solving the DNN-SAT adversarial robustness problem.

3.2.2 Adversarial Examples and the FGSM Method

One can define adversarial examples as DNN inputs that are similar to a correctly classified input in a well-defined manner, and yet cause a DNN to classify them incorrectly. Often ℓ_p norms are used as a similarity metric, although any desired property can be used. In the case of an ℓ_p norm similarity measure with threshold δ, an example x' is said to be adversarial if:

[2] To be precise, φ is parametric in M and y and should ideally be written as $\varphi_{(M,y)}$. The subscripts are omitted for readability.

[3] I stress that the DNN-SAT problem is a general definition of verifying whether a DNN adheres to certain properties and is independent of the DNN architecture, the type of property to verify, and the solution method.

Algorithm 1: The Constrained Gradient Descent (CGD) algorithm

```
Data: DNN $M$, Label $y$, Weight $\epsilon$, Constraint $\varphi$
Result: DNN Input $x$ s.t. $(argmax(M(x)) = y) \wedge \varphi(x)$
1  $\varphi' =$ ConstraintLoss($\varphi$) ;                // convert $\varphi$ to constraint loss
2  initialize $x$ ;             // $x$ is often initialized to be a random vector
3  do
4      if $(argmax(M(x)) = y)$ $AND$ $\varphi'(x) = 0$ then
5          break ;      // terminate if $x$ satisfies the model and constraint
6      end
7      $l_1 = \varphi'(x)$ ;                                // compute constraint loss
8      $l_2 =$ Loss($M(x), y$) ;                     // compute misclassification loss
9      $\nabla x_1 = \frac{\partial l_1}{\partial x}$ ;           // compute gradient of constraint loss w.r.t $x$
10     $\nabla x_2 = \frac{\partial l_2}{\partial x}$ ;  // compute gradient of mis-classification loss w.r.t $x$
11     $x = x - \epsilon * (\nabla x_1 + \nabla x_2)$ ;   // update $x$ so as to minimize both losses
12 while $True$;
13 return $x$
```

$$\|x - x'\|_p \leq \epsilon \wedge argmax(M(x)) \neq argmax(M(x'))$$

Multiple methods exist for finding adversarial examples, and these methods often use gradient descent to add a perturbation to an example in the direction of maximized loss. An example of this is the *fast gradient sign method* [46]. By aligning the sign of the perturbation with the sign of the gradient, FGSM can achieve large deviations by exploiting locally linear behaviour of the model.

3.3 Constrained Gradient Descent (CGD) Algorithm

This section provides an overview of the constrained gradient descent (CGD) method (please refer Algorithm 1) which is a modification of the Fast Gradient Sign Method (FGSM) algorithm. The input to the CGD algorithm is a DNN-under-test M, a label y, a weight ϵ, and a constraint φ that encodes some property of the DNN to test. The output of the algorithm is an DNN input x that satisfies the constraints φ and $(argmax(M(x)) = y)$.

In the CGD algorithm, the user-specified constraints φ are first parsed and converted into a constraint loss φ' function (see Chapter 2 for more details on this process) via a short piece of code in TensorFlow (line 1). The constraint loss function is constructed in such a way that it results in 0 only when all constraints are satisfied. Input x to the DNN M is initialized as a random vector, whose size is equal to the size of input of M (line 2). The code then enters a loop that terminates only when the constraints are satisfied (i.e $\varphi'(x) = 0$) and the model classifies x as the label y (lines 4-6). If x satisfies these criteria, the loop is broken out of and x returned as the solution (line 13). Otherwise, gradient descent is used to modify x so that it can meet both criteria. Specifically, both constraint loss (line 7) and misclassification loss (line 8) are calculated, and then the gradient of both losses with respect to x is computed (lines 9 and 10). Finally, both gradient values are combined, and x is slightly altered (using a small ϵ value) in order to move in the direction of decreased loss (line 11). This process continues till either x satisfies both criteria or the process runs out of time.

Note that in certain settings, one might wish to make the following modifications to this algorithm for better performance. First, adding a small amount of carefully crafted noise may help this loss avoid local minima and allow to find results faster (this practice is often used in FGSM to achieve better results). Second, the constraints may never be fully satisfied, and in such cases one may wish to find inputs that 'mostly' satisfy the constraints. In this case, a small bound δ can be added to update the conditional statement in line 4 to check whether the constraints are mostly satisfied (i.e $\varphi'(x) < \delta$). Third, one may find better results if using separate weights ϵ_1, ϵ_2 for each of the gradient values. This can allow to control the effect of each of the gradient values in order to better alter x. Further, note that solving for x is in essence a multi-objective optimization problem, and so algorithms and methods used in this field may be leveraged to achieve better results.

Intuitively, CGD provides an advantage over pure optimization or verification/program analysis methods. While verification/program analysis methods allow for encoding a rich set of constraints, they do not scale well because they require that DNNs be presented in a symbolic form with millions of variables to solve and given as inputs to a solver (such as MILP or SMT) [81]. On the other hand, CGD converts constraints into a loss function and utilizes a modified optimization algorithm in order to find inputs that minimize this loss. This results in CGD having similar scaling capability as optimization methods while allowing for a richer set of constraints (expressibility), with no restriction on DNN architectures that can be analyzed (generality).

CGDTest Tool Implementation Details.

I implemented the CGDTest tool using Python 3.8 and Tensorflow 2.2 [1], a popular open source machine learning framework. CGDTest takes in as input the constraints given in the language of boolean combinations of comparisons between differentiable loss functions (created in Tensorflow) and floating point values, as described in 2.3.1. The allowable differentiable functions are restricted to a small set (including ℓ_∞ and ℓ_2 distances) that can be extended as needed to other loss functions as long as they maintain the required properties. These constraints are then converted to constraint loss using a simple parser as described in 2.3.1. This constraint loss is then used as input, along with the model M and label y as input to an implementation of Algorithm 1 in order to find examples that satisfy constraints. Since CGD can potentially be stuck in an infinite loop, it is terminated if it reaches a maximum time limit, returning "incomplete".

3.4 Experimental Evaluation of the CGDTest Tool

This section presents an extensive empirical evaluation of CGDTest tool with state-of-the-art testing, optimization-based, verification/program analysis, and reachability-based tools. This comparison is both qualitative along several vectors, and empirical over a set of 26 medium to large industrial-sized DNNs with parameters ranging from 6000 to 11 million.

Experimental Setup: Hardware and Software: All the experiments were performed on a machine consisting of four Intel i5-4300U (CPU 1.9 GHz) cores and 16 GB RAM running Ubuntu 20.04. In order to run the experiments, I implemented an evaluation framework using Python 3.8, Tensorflow and Keras and compared state-of-the-art DNN TAV tools on three benchmark suites: VNN-LIB, Defended CNN and Resnet-18. Refer to Section 3.4.1 for details on the tools and to Section 3.4.2 for information on benchmark suites.

3.4.1 Qualitative Comparison of DNN TAV Tools

After an extensive and thorough survey of various DNN TAV tools, I chose **Fast Gradient Sign Method (FGSM)** [46], **Basic Iterative Method (BIM)** [82], and **Carlini & Wagner Attack (C&W)** [20] from the optimization methods, **DLFuzz** [79] and **Random Fuzz** from the fuzzing based methods, and **Genetic Algorithm (GA)** [147] from genetic algorithm based methods. From the various verification/program analysis based methods, I selected **Marabou** [72] and **MIP Verify** [144] tools. From the reachability-based methods, I selected **ERAN** [137]. A qualitative comparison to contrast the strengths and weaknesses of these DNN TAV tools along several vectors that an industrial DNN developer would be interested in is provided below.

Scalability of Method: This refers to how well a method can efficiently generate test inputs for large industrial-sized neural networks. Testing methods are considered to have high scalability as they can be used to test DNNs with millions of parameters. Medium scalability indicates that a method can be suitable for medium-sized networks that may have multiple hidden layers with over 100,000 parameters. DLFuzz and ERAN are considered to have medium scalability because they have been used on medium sized models, while verification methods are known to have low scalability.

Expressibility of Properties: This refers to how expressible a method can be. Strong expressibility means a method is suitable for testing any property that can be encoded in the language of constraints accepted by the method. Verification methods are considered to have strong expressibility because they can encode many kinds of properties symbolically. Medium expressibility refers to a method that is able to test a large range of properties, but that some restrictions exist in terms of what properties are expressible in the method's input language. ERAN and CGDTest are considered to have medium expressibility. Similarly, GA is considered to have medium expressibility because it can check for constraints as long as they produce a fitness value. Conversely, low expressibility refers to methods that are suitable for a very small number of tasks (usually a single task). The rest of the testing based methods fall under this category.

Generality: It is common for methods to have restrictions on DNN architectures that define allowable hyper-parameters, activation functions, and other properties of a model-under-test. Often these restrictions are in the form of allowable activation functions, where the method allows for only very restricted kinds activation functions that it can test. Additional restrictions can come in the form of architectures (such as recurrent layers). All testing methods compared here do not place any restriction on the architecture of the DNN-under-test, making them very suitable to test most DNNs. By contrast, verification methods severely restrict the architecture of the DNN-under-test, often to only fully-connected layers and ReLu activation functions. Finally, while ERAN places some restrictions, it allows for a larger variety of DNN architectures as compared to verification methods.

Certificate of Guarantee: Certain methods can provide rigorous guarantees about network behaviour. In such cases, the method either produces a formal guarantee that a DNN satisfies a given property, or a model input that witnesses a property violation in the DNN-under-test. A certification is considered to be *strong* if the said method gives a provable strict guarantee that a property is verified and no counterexamples exist. Examples of such methods include MIP Verify. In some instances these certificates can be over-approximated to improve performance. Certificates are considered to be *weak* when a method uses relaxation to improve performance, but this relaxation comes at the cost of a strict guarantee. In other words, the certification only applies to a relaxed version of the model, and does not guarantee that the property holds for the model itself, as is the case for Marabou. ERAN can provide both strong and weak guarantees, based on the settings used. All other methods, including CGDTest, do not provide any guarantee.

3.4.2 Benchmark Suites

For the experimental evaluation, I chose a set of benchmark suites to perform empirical evaluation of the scalability of CGDTest and the tools identified in Section 3.4.1.We compare against a suite of benchmarks that consist of 26 models total with model complexity ranging from 6000 to 11 million parameters. The benchmarks used are:

Convolutional Neural Network (CNN)		
Layer Name	**Shape**	**Parameters**
Conv2D	(None, 14, 14, 20)	520
MaxPooling2D	(None, 7, 7, 20)	0
Conv2D	(None, 3, 3, 20)	10,020
MaxPooling2D	(None, 1, 1, 20)	0
Dense	(None, 64)	1344
Output	(64, 10)	650
	Total Parameters:	12,534

Table 3.1: Architecture of the CNN Model used in the Defended CNN benchmark

VNN-LIB: “an international initiative whose aim is to encourage collaboration and facilitate research and development in Verification of Neural Networks” [148]. The members of this community provided a collection of benchmarks intended to be used to compare various DNN TAV tools (refer to [48] for more details on this benchmark suite).

Defended CNN: the second benchmark suite consists of a convolutional neural network (CNN) trained on MNIST that is defended using projected gradient descent (PGD) algorithm described in [95]. This benchmark is intended to showcase the efficacy of these methods on DNNs that are defended. The model architecture for the CNN is given in Table 3.1.

Resnet: The third and final benchmark suite consists of a pre-trained Resnet-18 model (a detailed description of this architecture can be found in [53]) trained on the ImageNet dataset. The Resnet-18 model consists of 18 total layers including convolution layers and consists of over 11 million parameters. Due to the extensive number of parameters in this model, this benchmark is intended to showcase the scalability of the TAV methods to DNNs used in the industry.

3.4.3 Experimental Results

To empirically evaluate all the TAV tools, I created a framework that provides as input to each tool a triple of DNN model, an input, and a specification and checks whether the tool produces a result within an allotted time. For the inputs, 25 examples from the test sets of MNIST and ImageNet datasets were randomly selected. I used a timeout of 600 seconds for the evaluations. The framework was then set to run all TAV tools using these triples, and record the number of successful adversarial examples (or certificate of guarantees) generated by each tool.

PAR2 Score: In addition to the number of successful adversarial examples generated, the framework also calculated the PAR2 score for all tools on each benchmark suite. PAR2 scoring is used by the SAT community in order to effectively compare various SAT solvers on a suite of benchmarks. PAR2 score is calculated as follows: if a method produces a result within the time limit, the time taken is recorded (in seconds) to produce that result. Conversely, if a method is unable to produce a result in the given time, a penalty of 2*time-limit is added. A lower PAR2 score is considered better, as that signifies that the method was able to find violating inputs (or certify robustness) faster on average over a method that resulted in a higher PAR2 score. This time is aggregated on each benchmark suite and an overall PAR2 score for all three benchmark suites is also recorded. In order to showcase the *expressibility* of CGDTest, I ran three sets of experiments, each of which use different constraints to define adversarial examples:

1. **Standard Adversarial Constraint**: this constraint encodes adversarial examples as given in Definition 3.2.2. The resulting implementation is called CGDStandard.
2. **Natural Adversarial Constraint**: this constraint ensures that the original label y is now the second highest label by the model M on the adversarial input x'. This resulting implementation is called CGDNatural. The motivation for this is to show a great weakness of verification tools, that are unfortunately based on L_p-ball verification. I.e., if a DNN is robust against perturbed images within in an L_p-ball of the original image, then such DNNs would be certified as resistant to adversarial attacks by today's verification tools (assuming that these verification tools scale to the DNN-in-question). Unfortunately, such a certificate does not mean much because an attacker can still attack such DNNs via "natural adversarial constraints". Another

	Standard Adversarial	Natural Adversarial	Flow Adversarial
TAV Methods	**PAR2 (sec)**	**PAR2 (sec)**	**PAR2 (sec)**
CGDTest	**18,476.36**	**143,010.88**	**45,170.87**
C&W	21,713.19	*	*
BIM	23,120.65	*	*
FGSM	43,220.72	*	*
ERAN	600,040.7	647,970.55	*
MIP Verify	653,736.16	703,006.8	*
DLFuzz	707,066.27	*	*
Marabou	703,057.3	728,376.78	*
Random Fuzz	734,821.58	780,000	780,000
Genetic	703,211.67	735,509.32	761,186.62

Table 3.2: Results of all three experiments for the listed TAV tools on the three benchmarks. A * indicates that the tool was not able to test for the given type of adversarial example.

problem with verification tools is that it is not even clear to us how they will search the space of inputs for ”natural adversarial” attacks, given the fact that they don’t leverage gradient information in the way that CGDTest does.

3. **Flow Adversarial Constraint:** this constraint utilizes spatial transformations (flow) as given in Xiao et al. [154] to find adversarial examples. More specifically, given a DNN model M, label y, original input x and new input x', this constraint checks whether $argmax(M(x')) \neq y$ while x' and x have minimal *spatial transformation* perturbation. This implementation is referred to as CGDFlow.

For each of the three sets of constraints, I compared all tools on these benchmark suites. The results of these experiments is given in Table 3.2. For each experiment, the PAR2 score (in seconds) produced by each TAV method on all the three benchmark suites is provided, as well as an overall PAR2 score. The result of all three experiments is given in table 3.2.

Standard Adversarial Constraint Results: Overall, CGDTest was found to have the lowest PAR2 score of 18,476.36, resulting in an improvement of 14.9% over the next best tool (C&W). As a group, optimization tools resulted in the best overall PAR2 scores, with all other tools significantly lagging behind. While verification/program analysis and reachability tools performed well on the VNN-LIB benchmark suite, they could not scale to the other two suites. Fuzzing and GA performed consistently poorly on all three suites.

Natural Adversarial Constraint Results: Overall, CGDTest was again found to have the lowest PAR2 score of 143,010.88, resulting in an improvement of 353.1% over the next best tool (ERAN). CGDTest consistently resulted in the lowest PAR2 score across all benchmarks. Note that optimization based testing tools as well as DLFuzz can not test for adversarial definitions using these constraints. Verification/program analysis and reachability based methods resulted in an improvement over their performance in the previous experiment. However, they were still unable to produce results on the Resnet-18 benchmark.

Flow Constraint Results: Once again, optimization based tools as well as DLFuzz are unable to test for this property. While verification/program analysis based tools can theoretically test for spatial perturbations, the corresponding tools do not allow for this at the moment. The only tool (other than CGDTest) to produce results is the Genetic tool. CGDTest was again found to have the lowest PAR2 score of 45,170.87, which is over 1500% better than the next best tool (Genetic).

3.4.4 Analysis of Results

While optimization tools were observed to result in great performance w.r.t PAR2 scoring and the number of adversarial examples generated in experiment 1, they are unable to test for the constraints given in experiment 2 and 3. By contrast, verification/program analysis and reachability tools are able to test for the constraints in experiments 1 and 2, and produced good results on the smaller VNN-LIB suite, but were unable to scale to larger ones. Finally, fuzzing and GA tools showed consistently poor PAR2 results on all three suites. The CGDTest tool scales and generalizes better than even optimization tools (with a better overall PAR2 score) - because it uses the same gradient information in order to efficiently navigate the search space to find inputs while simultaneously taking into account user-specified constraints. Further, CGDTest is more expressible than other optimization tools because it allows for encoding of a rich language of constraints and these constraints also assist with efficiently searching the space of inputs. On the other hand, unlike verification tools, CGDTest does not produce a certificate of guarantee. Further, as mentioned previously, one can consider the case of constraints that are composed of multiple differentiable functions as a multi-objective function. In such cases, careful fine-tuning of weights may be required in order to better find inputs that minimize the overall constraint loss. Nevertheless, the scalability, generality and expressibility of CGDTest make it an ideal method to test DNN properties.

3.5 Related Work

DNN Testing Methods: As mentioned before, the class of DNN testing methods can be further sub-divided into three categories: fuzzing, genetic algorithm or GA, and optimization or GD methods. Optimization methods generally use some form of gradient descent to attempt to find examples that satisfy some property. Examples include the Fast Gradient Sign Method (FGSM) [46], the Carlini & Wagner (C&W) attack [20], the DeepFool attack [107] and Basic Iterative Method (BIM) [46]. These attacks are efficient, extremely generalizable, and require nothing more than access to the model-under-test (and possibly a input to modify). However, they come at the expense of both expressibility and certification. Other common testing methods include input fuzzing and genetic algorithms that attempt to do the same. Examples include random fuzzing, and DLFuzz [79] that utilizes neuron coverage statistics to guide fuzz in the direction of greater neuron coverage to find adversarial examples. DLFuzz has difficulty scaling to industrial-sized networks and is limited to searching for adversarial examples. By contrast, CGDTest is not only more efficient compared all competing testing methods, but also scores highly on expressibility and generality.

Verification/Program Analysis Methods: These methods encode the model-under-test symbolically and represent properties as formulas or constraints, with the aim of providing a certificate of guarantee (i.e., no violations of properties) or produce a counter-example. There are several tools such as the ones by Leofante et al. [85], Büning et al. [18] and MIP Verify [144] that encode DNNs and properties as MILP problems and use corresponding solvers to prove or disprove the existence of property-violating examples. They often restrict DNN activation functions and architectures to be piecewise-linear DNNs, which is not the case for CGDTest. On the other hand, tools like Reluplex [70] and Maribou [72] are simplex-based methods that can handle arbitrary piecewise linear activation functions. By contrast to these tools, CGDTest is very efficient and scales to large industrial-sized DNNs in the experimental evaluation, and also scores highly on generality, at the expense of certification. Also, CGDTest is less expressive than these verification methods considered.

Reachability Methods: These methods define an allowable input region and propagate that region through the network, resulting in an output domain that encapsulates any possible reachable output given one of the input values. Via this method, one can verify robustness by limiting the input region to a shape around an example (such as a perturbation bound) and verifying that the reachable outputs do not include incorrect classifications. Examples of such tools include ERAN [137] and NNV [146]. Unfortunately, in contrast to CGDTest, these methods do not scale well, and score low on generality and expressibility. Unlike verification methods, they do not provide any certificates of guarantees.

Learning via Constraint Loss: To the best of my knowledge, Kammig et al. were the first to introduce *probabilistic soft logic* [75], which allowed for encoding logical constraints as loss functions that are mostly continuous and used solvers to infer the most probable explanation (MPE). Xu et al. introduced *semantic loss*[156] that aims to capture how well the labels of a DNN align with constraints. This loss is derived using DNN outputs and logical constraints. Their experiments empirically show that leveraging semantic loss can result in lower DNN training time. By contrast, the constraints used in this work can be defined over DNN inputs and parameters, not just outputs. Further, constraint loss is used to test for property violations. Fisher et al. then introduced DL2 [36] in 2019, which allows for certain logical constraints to be converted into differential-everywhere loss functions. The resulting constraint loss function was used in these works to train a DNN in a manner that minimizes constraint violations by updating DNN parameters so as to minimize this loss. Further, this work used this loss in order to find training instances that resulted in a higher loss and added them to the next batch of training. CGD differs from the above-mentioned methods in the following ways. First, in their methods the constraints do not have to be defined over DNN parameters, unlike in the case of CGDTest. Second, this constraint loss is combined with a modified optimization based method (specifically FGSM). Finally, this method tests a DNN against a given property by finding inputs that satisfy certain constraints, instead of using it for DNN training.

3.6 Conclusions and Future Work

To address the reliability and security problems associated with DNNs, in this chapter I propose a new testing algorithm, called the Constrained Gradient Descent (CGD) method, that leverages domain knowledge provided as logical constraints. Via an extensive empirical evaluation of CGDTest against 9 other state-of-the-art methods (including testing, verification, and reachability), on a rich benchmark of 26 DNNs with sizes ranging from 6000 to 11 million parameters, I show that CGDTest scales better to large industrial-sized DNNs, allows users to specify a wide variety of DNN properties, as well as places no restrictions on the DNN architecture. While adversarial robustness is the most commonly researched property of DNNs at the moment, there are other important properties of DNNs such as equivalence, fairness and bias for which current TAV methods do not work very well. In these contexts, CGDTest could be an excellent choice as a TAV method.

Chapter 4

xAI-GAN: Enhancing Generative Adversarial Networks via Explainable AI Systems

4.1 Introduction

Generative Adversarial Networks (GANs), introduced only a few short years ago, already have had a revolutionary impact on generating data of varied kinds such as images, text, music, and videos [45]. The critical insight behind a GAN is the idea of corrective feedback loop from a deep neural network (DNN) called the *discriminator* back to a *generator*. However, a notable weakness of GANs is that they require a lot of data for successful training. For example, in order to learn to write digits, a human may only need a few ($\leq$10) examples before she or he learns to replicate them whereas a GAN would often need several orders of magnitude more data ($\geq$1000s) to learn the same task.

Quantitative evidence for GANs needing lots of data. While it is widely accepted that GANs need "a lot" of data to successfully learn, there isn't a lot of quantitative analysis to support this point of view. Hence, I provide quantitative evidence for this phenomena as a first step. More precisely, to investigate the impact of the size of data on GAN training, multiple GANs were trained using the same parameters on varying amounts of MNIST [84] dataset and compared them via Fréchet Inception Distance [57] (FID) scores. FID has been shown to be consistent with human evaluation of image quality and is the standard performance metric for GANs. I adjusted the number of training epochs so that all models were trained for the same number of iterations. More details on the GAN

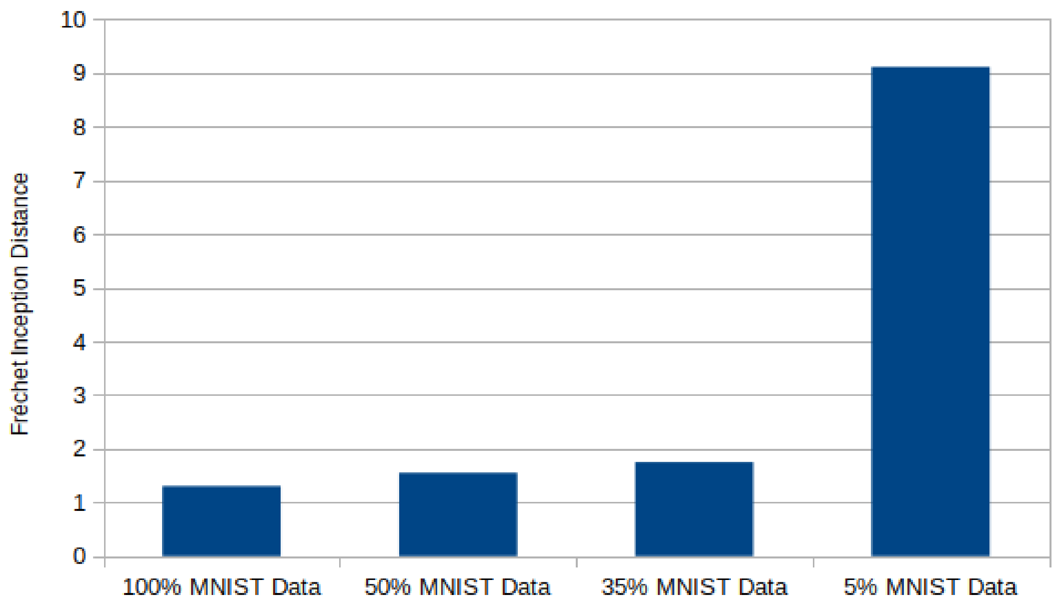

Figure 4.1: FID scores on the same GAN architecture trained on different amounts of data.

architecture and experiment set up can be found in Section 4.3. The results are found in Figure 4.1. The model trained on 100% of the data is observed to have the best FID score, while the model trained on only 5% of the data had the worst. There appears to be a correlation between the size of the data used and the FID score, with a GAN that uses more data resulting in a better performance. These results are consistent with [162] where the authors show how GAN training deteriorates given a limited amount of data.

The collection of high-quality labelled data for GANs and other machine learning algorithms often requires a lot of resources such as time and money and is considered a 'major bottleneck' in machine learning research [126]. As a consequence, there is a need for finding other ways of making GANs more data-efficient. Observe that, in the standard GAN architecture, the feedback provided by the discriminator to the generator is calculated using only one value (loss). This feedback is calculated as follows. The discriminator takes as input data from the generator to make a prediction. Next, loss is calculated based on this prediction and is used by the discriminator to provide feedback to the generator. This feedback is then used by the generator to perform gradient descent and update it's parameters so as to better fool the discriminator. The feedback provided thus originates from this single real-numbered value (loss). Hence, the research questions I address in this chapter are the following: *is it possible to leverage domain knowledge to provide a "richer"*

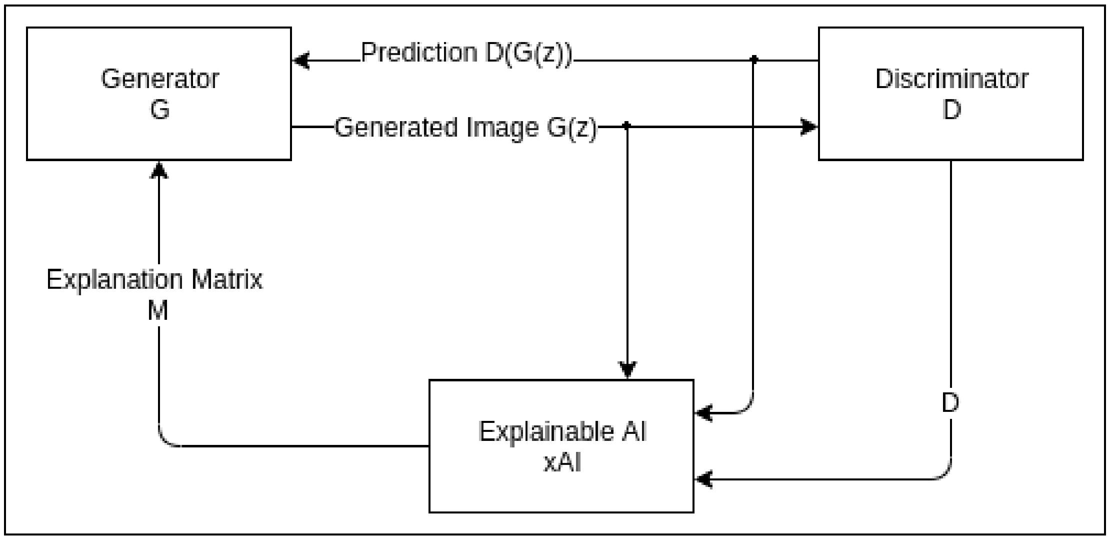

Figure 4.2: System Architecture of xAI-GAN.

corrective feedback from the discriminator back to the generator? If so, *does it enable GANs to be more data-efficient?*

4.1.1 An Overview of xAI-GAN

To answer the above-mentioned research questions, this chapter proposes a new class of GANs referred to as xAI-GAN wherein it is possible to provide "richer" corrective feedback (more than a single value) during training from discriminator to generators via Explainable AI (xAI) systems. A high-level system architectural overview of the xAI-GAN system is given in Figure 4.2. Consider the problem of training a GAN with the aim of producing images of digits (see Chapter 2 for more information on GAN training). Initially, the untrained generator G is given a noise sample z from a prior noise distribution p_z and produces an example $G(z)$ that is then given to discriminator D. The loss is calculated, and then the generated image $G(z)$, the discriminator D, and the output of the discriminator $D(G(z))$ are fed into an xAI system that produces an explanation as to why the image resulted in that loss. This explanation is then used to guide the training of the generator (refer Section 4.2 for details).

As mentioned in Section 2.2, a common analogy for GAN training is that of a counterfeiter (the generator) and a detective (the discriminator) playing an adversarial game. Over the training process, the detective and counterfeiter both get better at their jobs, with the end goal being that the counterfeiter is so proficient that their fakes can pass for the real thing. To extend this analogy to the xAI-GAN setting, this method works by using an expert in the field (the xAI system) to help improve the counterfeiter. When the detective recognizes a fake, the expert tells the counterfeiter what parts of the fake tipped off the detective. The counterfeiter is thus able to learn better why the detective detected a fake, and make better decisions to avoid pitfalls in future training.

Explanation Matrix: The explanation produced by xAI systems are converted to the form of an "explanation matrix" E, wherein, for every feature in an example (e.g., an input image), a value in the range [0,1] is assigned to the corresponding cell of the matrix E. If a feature (more precisely, the corresponding cell in E) is assigned value 0 (or close to 0), then it means that pixel had no impact on the classification decision made by the discriminator D. If a feature is assigned a value of 1 (or close to 1), then that means that feature is very important. This could be due to the feature being "determinative" in the classification made by D or when it "hurts" the classification made by D (more precisely, if the feature were to be changed, then the confidence of D in its classification would improve). Creating the explanation matrix in this manner helps to focus the learning process on the most influential features, regardless of whether those features were beneficial or harmful to the classification. Note that this is then used to inject to gradient descent (refer to 2.3.2).

4.1.2 Contributions

1. **xAI-guided Gradient Descent Algorithm and xAI-GAN:** The key contribution of this chapter is an xAI-guided gradient descent method (and a resultant GAN referred to as xAI-GAN) that injects domain knowledge sourced from xAI systems to focus the gradient descent algorithm on weights that are determined to be most influential by the xAI system (refer Section 4.2.2). Several versions of xAI-GANs are implemented using different state-of-the-art xAI systems (refer Section 4.2.2), namely saliency map [135], shap [94], and lime [125]. In Section 4.5.1, I discuss how xAI-guided gradient descent methods can give those training models greater control over the learning process.

2. **Experimental Evaluation of Quality of Images produced by xAI-GAN vs. Standard GAN:** I performed experiments to evaluate the quality (as measured by Fréchet Inception Distance, abbreviated as FID [57]) of xAI-GANs relative to standard GANs. I show that on MNIST and Fashion MNIST datasets, xAI-GANs achieve an improvement of up to 23.18% in FID score compared to standard GANs (refer Section 4.3.3).

Algorithm 2: Generator Training Algorithm. Note that the code block under the if *use_xAI* block only applies to the xAI-guided generator training.

```
   input  : generator G
   input  : discriminator D
   input  : explanation system xAI
   input  : boolean Flag use_xAI
   output: trained generator G
1  foreach noise sample $z$ do
2      Loss L = $Loss(1 - D(G(z))$ compute Discriminator Gradient $\Delta_D$ from L
        compute Generated Example Gradient $\Delta_{G(z)}$ from $\Delta_D$ if use_xAI is True
        then
3          compute Explanation Matrix $E$ using xAI  compute Modified Gradient
4          $\Delta'_{G(z)} = \Delta_{G(z)} + \alpha * \Delta_{G(z)} * E$ compute Generator Gradient $\Delta_G$ from $\Delta'_{G(z)}$
5      else
6          compute Generator Gradient $\Delta_G$ from $\Delta_{G(z)}$
7  end
8  update Generator parameters $\theta_G$ using $\Delta_G$
```

3. **Experimental Evaluation of Data Efficiency of xAI-GANs vs. Standard GANs:** I extend the experiments to the CIFAR 10 dataset, using only 20% of the data for xAI-GAN while letting standard GAN use 100% of the data. xAI-GAN outperforms standard GAN in FID score even in this setting. I further compare this work with Differentiable Augmentation [162] technique which has been shown to improve data-efficiency of GANs. Specifically, I show that xAI-GAN outperforms Differentiable Augmentation, resulting in a better FID score. Finally, I modify xAI-GAN to incorporate Differentiable Augmentation and show that the resulting model has better performance than either version.

4.2 Detailed Overview of xAI-GAN Systems

This section provides a detailed overview of the xAI-GAN system (please refer to Figure 4.2 for the system architecture of xAI-GAN and Algorithm 2 for the gradient descent algorithm for generator training) and contrast it with standard GAN architectures as well as the way they are trained. The intuition behind the xAI-guided generator training process is that the xAI system acts as a guide, shaping the gradient descent in a way that focuses generator

Generator	**Discriminator**
Dense (100, 256)	Dense (1024, 1296)
Dense (256, 512)	Dense (1296, 512)
Dense (512, 1296)	Dense (512, 256)
Dense (1296, (32, 32))	Dense (256, 1)

Table 4.1: Model Architecture of Fully Connected GAN used in the context of MNIST and Fashion MNIST experiments.

Generator	**Discriminator**
Conv (4 x 4)	Conv (16 x 16)
Conv (8 x 8)	Conv (8 x 8)
Conv (16 x 16)	Conv (4 x 4)
Conv (32 x 32)	Conv (1 x 1)

Table 4.2: Model Architecture of DC-GAN used in the context of the CIFAR10 experiments.

training on those input features that the discriminator recognizes as most important.

4.2.1 Generator Training in Standard GANs

Please refer to Section 2.2 for an overview on GAN training process. Here, I contrast the difference between xAI-guided generator training with standard GAN generator training. A single iteration of GAN generator is trained as follows (please refer to Algorithm 2): a selection of noise samples are drawn from the noise prior and passed through the generator to get a batch of generated examples (line 1). This batch is labelled as "real" and given to the discriminator, where the loss is found (line 2), and then used to update the generator parameters (the corrective feedback step). More precisely, the discriminator's gradient Δ_D is computed using the parameters of the discriminator and its loss (line 3), which is used to find the gradient of the generated example $\Delta_{G(z)}$ (line 4). Further, the gradients of all layers in the generator Δ_G are then computed using $\Delta_{G(z)}$ (line 10). Finally, the parameters Θ_G of the generator are updated using Δ_G (line 12) - completing one training iteration.

4.2.2 xAI-guided Generator Training in xAI-GANs

Observe that in the standard GAN setting the discriminator calculates the corrective feedback to the generator using only a single value (loss) per generated image. The entire point of xAI-guided training is to augment this feedback with the "reason" for the discriminator's decision, as determined by the xAI system.

During the *xAI-guided gradient descent* generator training process, the backpropagation algorithm is modified to focus generator training on the most meaningful features for the discriminator's prediction (please refer to lines 5-8 of Algorithm 2). Following with propagating the loss through the discriminator to find $\Delta_G(z)$, an xAI system is used to produce an explanation matrix E (line 6). E is a set of real values $\in [0, 1]$, where greater values represent features that are more important to the discriminator's prediction. The Hadamard (element wise) product of $\Delta_{G(z)}$ and M is calculated to get the modified gradient $\Delta'_{G(z)}$ (line 7). In an intuitive sense, E acts as a mask for $\Delta_{G(z)}$, focusing the gradient on the most important features and limiting the gradient on the less important ones. From there, the gradients of the generator Δ_G are calculated from $\Delta'_{G(z)}$ using a small value for α (line 8) and the parameters are then updated (line 12).

4.2.3 xAI-GAN Implementation Details

I implemented xAI-GAN using Pytorch 1.6 [116], an open source machine learning framework popular in deep learning research. For saliency and shap xAI systems I used Captum 0.2.0 [77], an open source interpretability framework developed by the team at Pytorch. For the lime-based xAI-GAN system I used the implementation from Lime 0.2.0.1 [125]. The explanation matrix E is generated by each of the xAI systems by taking the absolute value and normalizing the matrix to create a mask vector with values in range $[0, 1]$.

Pytorch notably has the `autograd` [115] package which handles automatic differentiation of all tensors. In order to provide xAI-guided feedback to the generator, I overrode the `register_backward_hook` function normally used to inspect gradients. I modified the gradients of the output layer of the generator using the resultant vector computed by the Hadamard (element wise) product with the computed mask. This modified gradient is back-propagated through the generator via the `autograd`. After extensive testing, I observed that switching on the xAI-guided gradient descent after half the number of training epochs gives the best results. This is because the discriminator would have learnt the distribution of the task at hand to a certain extent and consequently the xAI system is likely to produce better explanations.

4.3 Experimental Results

I performed extensive experimental evaluation of the xAI-GAN implementation that used 3 different xAI systems, comparing against standard GAN. These experiments were performed on three different datasets:

1. MNIST [84] a collection of 70,000 28x28 grayscale images of handwritten digits,
2. Fashion MNIST [155] a collection of 70,000 28x28 grayscale images of clothing, and
3. CIFAR10 [78] a collection of 60,000 3x32x32 color images of objects.

For the MNIST and Fashion MNIST datasets, the images were resized to 32x32 and fully connected GANs were used for both standard and xAI-GAN, the architecture for which is shown in Table 4.1. Leaky relu was the activation used for all but the last layers in the generator and discriminator. In the last layer, I used tanh for the generator, and sigmoid for the discriminator. A dropout rate of 0.3 was used during training in discriminator. For CIFAR10 dataset, I use a DC-GAN architecture for both standard and xAI-GAN as described in Table 4.2. The generator and discriminator use four convolutional layers, each with a stride of 2 and padding of 1. Each of them also use a batchnorm layer after every convolutional layer, except for the last one. The activation functions are identical to the fully-connected GAN architecture.

4.3.1 Experimental Setup

The batch size was selected to be 128 in the experiments. The Adam optimizer [76] was used for both generator and discriminator training. I used a learning rate of 0.0002, and ran experiments using Amazon's EC2 on a p2.xlarge instance which uses 1 Nvidia's K80 GPU with 64GiB RAM.

4.3.2 Evaluation Criteria

Based on a thorough literature survey of metrics [113] for the image domain, the following criteria was developed in order to perform a fair comparison of xAI-GANs vs. standard GANs:

Dataset	Standard	Shap	Lime	Saliency
MNIST	1221.19	9925.1	38865.93	2215.73
FMNIST	991.11	9694.81	39237.33	2162.24

Table 4.3: Average experiment time taken (in seconds) on MNIST and Fashion MNIST Datasets

Dataset	Standard	Shap	Lime	Saliency
CIFAR10	536.76	13290.34	12988.32	1214.44
+Diff	674.42	14319.87	13352.37	1175.99

Table 4.4: Average experiment time taken (in seconds) by each GAN on MNIST and Fashion MNIST (FMNIST) datasets.

1. **Fréchet Inception Distance (FID):** I opted to use Fréchet Inception Distance (FID) to measure quality since it has been shown to be consistent with human evaluation of quality [57]. FID was introduced by Heusel et al., to address the shortcomings of Inception Score (IS) such as the latter's inability to detect intra-class mode dropping and vulnerability to noise [57].

   At a high level, FID converts a set of images to the feature space provided by a specific layer in the Inception model. Various statistics, such as the mean and covariance are computed on the activation values of that layer to generate a multi-dimension Gaussian distribution. Finally, the Fréchet distance of the two distributions created using the generated and the training images is computed and provided as the output. In order to apply FID to MNIST and Fashion MNIST, I use the LeNet classifier, consistent with [13].

2. **Training Time:** I also measure the time required for training to identify the overhead added by xAI systems.

4.3.3 Results on MNIST and Fashion MNIST

I ran the experiments on both MNIST and Fashion MNIST datasets using two settings: 100% data and 35% data (to see the performance of xAI-GAN when data is scarce). The results of the experiments on MNIST dataset can be found in Figure 4.3. For 100% data, standard GAN produced an FID score of 1.31. The xAI-GAN$_{shap}$, xAI-GAN$_{lime}$ and xAI-GAN$_{saliency}$ systems resulted in scores of 1.36, 1.21 and 1.26 respectively. The xAI-GAN$_{lime}$ system had the best performance and resulted in an improvement of 7.35% in the FID score, as compared to standard GAN. For 35% data, standard GAN produced a score of 1.75 while the xAI-GAN$_{shap}$, xAI-GAN$_{lime}$ and xAI-GAN$_{saliency}$ systems produced scores of 1.50, 1.41 and 1.55 respectively. All three xAI-GAN systems outperformed standard GAN in this setting, with the xAI-GAN$_{lime}$ system resulting in an improvement of 19.62%. A sample of the images generated by the xAI-GAN$_{lime}$ system using 35% data can be seen in Figure 4.4.

The results of the experiments on the Fashion MNIST dataset can be found in Figure 4.5. For 100% data, standard GAN produced an FID score of 1.16. The xAI-GAN$_{shap}$, xAI-GAN$_{lime}$ and xAI-GAN$_{saliency}$ systems produced scores of 1.06, 0.97 and 1.1 respectively. Again, the xAI-GAN$_{lime}$ system had the best results, with an improvement of 16.67% over standard GAN. For 35% data, standard GAN produced a score of 1.61. The xAI-GAN$_{shap}$, xAI-GAN$_{lime}$ and xAI-GAN$_{saliency}$ systems produced scores of 1.24, 1.35 and 1.34 respectively. Here, the xAI-GAN$_{shap}$ system had the best performance, with an improvement of 23.18%. A sample of the images generated by the xAI-GAN$_{shap}$ system using 35% data can be seen in Figure 4.6.

The average time taken by each of the GANs on the respective dataset can be found in Table 4.3. The xAI-GAN$_{saliency}$ system runs in about 2x the time that standard GAN does, while the xAI-GAN$_{shap}$ and xAI-GAN$_{lime}$ systems take around 10x and 35x the time that standard GAN requires respectively. The reason for the discrepancy in times between the xAI-GANs systems is due to the difference between their implementation. Overall, xAI-GAN has been shown to outperform standard GAN in terms of FID scores, with improvements of up to 23.18%.

4.3.4 Results on CIFAR10 Dataset

I next ran the experiments on the CIFAR10 dataset using the parameters described earlier. In order to view the efficacy of xAI-GAN in the case where data is scarce, I used 100% of the data for the standard GAN while only using 20% data for xAI-GAN. Further, to compare with the work in [162], I used the Differential Augmentation implementation code linked in the paper to run another set of experiments. In the latter experiments, I added Differential Augmentation to all GANs - which will hereafter be referred to as “+ Diff”. Note that standard GAN+Diff still uses 100% of the data while all the xAI-GANs+Diff use 20% data. The results of these experiments are found in Figure 4.7. In the first run of the experiment (i.e without Differential Augmentation), standard GAN resulted in a FID score of 214.81 while the xAI-GAN$_{shap}$, xAI-GAN$_{lime}$ and xAI-GAN$_{saliency}$ systems resulted in scores of 218.48, 210.04 and 211.16 respectively. The xAI-GAN$_{lime}$ system showed the best results with around 2.22% improvement in FID score over standard GAN, even with 20% of the data.

Adding Differential Augmentation to standard GAN resulted in an FID score of 212.81, which is 0.93% improvement in FID score over standard GAN. As previously shown, xAI-GAN (in particular the xAI-GAN$_{lime}$ system) produced better results than Differential Augmentation - even when using 20% of the data. Moreover, both methods can be combined to produce complementary results. Running xAI-GANs with Differential Augmentation produced better scores for all GANs, with xAI-GAN$_{shap}$+Diff, xAI-GAN$_{lime}$+Diff and xAI-GAN$_{saliency}$+Diff resulting in scores of 214.27, 208.45 and 209.70 respectively.

The times taken (in seconds) by each of the GANs to run the experiments can be found in Table 4.4. xAI-GAN$_{saliency}$ takes around 2x the time and xAI-GAN$_{lime}$ and xAI-GAN$_{shap}$ take around 25x the time that standard GAN requires. The time taken by xAI-GAN$_{shap}$ system is similar to the xAI-GAN$_{lime}$ system as the implementation of shap is expensive in the case of colour images. Overall, xAI-GAN$_{lime}$+Diff shows an improvement of 2.96% in FID score over standard GAN while using 20% of the data. Note that the discrepancy in FID scores (and conversely, the training time) between xAI-GAN and standard GAN would be higher if xAI-GANs used 100% of the dataset for training.

4.3.5 Discussion of Experimental Results

I performed extensive experiments using MNIST, Fashion MNIST and CIFAR10 datasets on both fully connected and DC GANs and showed that xAI-GAN, particularly one using the lime xAI system, results in improvements of up to 23.18% in FID scores over standard GAN. I compared this work with [162] and showed that xAI-GAN resulted in improvement over Differential Augmentation in these experiments, and that both techniques are complementary and can be combined to result in even more improvement in FID scores. An important take-away is that xAI-GANs show an improvement over standard GANs in terms of FID score even when using less data.

Regarding the increased training time of xAI-GAN: While xAI-GAN requires more training time compared to standard GANs due to the overhead of the xAI system, I believe this is still an advantageous trade-off. GAN research focuses on improving the quality of the images and handling data scarcity, and consequently the time required to train is not as important. In addition, xAI systems scale linearly with the number of neurons in the DNN model. Therefore, the overhead caused by the xAI system will be linear to the model - allowing most hardware that can train standard GANs to be able to train xAI-GANs. Furthermore, with the advent of parallel and distributed computing as well as easier access to powerful computational resources, the time required to train xAI-GAN will be further mitigated.

4.4 Related Work

Goodfellow et al. were the first to introduce GANs in [45]. Since then, GANs have continued to be a popular research topic with many versions of GANs developed [113]. GANs can be broadly classified based on their architecture [124, 104, 23, 96] and the type of objective function used [101, 9, 99, 35, 161, 50]. To the best of my knowledge, there is no GAN that uses xAI feedback for training, thus making xAI-GAN the first of its kind. Further, the method of injecting domain knowledge to GAN training is novel to the best of my knowledge. I note that the xAI-guided gradient descent algorithm is independent of architecture or type of objective function used, and therefore can be applied to make any type of GAN an xAI-GAN.

Differentiable Augmentation [162] is a recent technique that aims to make GANs more data-efficient by augmenting the training data. The main idea behind this technique is to increase the data via various types of augmentations on both real and fake images during GAN training. These augmentations are differentiable and so the feedback from the discriminator can be propagated back to the generator through the augmentation. On the other hand, while xAI-GAN also aims to make GANs more data-efficient, this is done by passing "richer" information from the discriminator to the generator through an xAI system. I compare xAI-GANs with Differentiable Augmentation in section 4.3.4 and show that they can be combined to provide further improvements in data-efficiency.

ADAGRAD [33] is an optimization algorithm that maintains separate learning-rates for each parameter of a DNN based on how frequently the parameter is updated. On the other hand, xAI-GAN uses xAI feedback to determine how generator parameters are updated (refer Section 4.2.2).

4.5 Conclusion and Future Work

The empirical results suggest xAI-GAN can be leveraged in settings where data efficiency is important - such as where training data is limited or in privacy conscious settings. It can be also used in normal settings to produce better quality GANs.

4.5.1 Controlling How Models Learn

While standard GANs only use one value (loss) to calculate corrective feedback to the generator, there are many ways this feedback is used. For instance, several GANs vary the type of loss function [101, 9, 161, 35, 50] and the selection of the optimizer (such as Stochastic Gradient Descent) to control how the model learns. Similarly, I believe that the feedback provided by xAI system using xAI-GAN - which is "richer" compared to only using the loss value - can allow for greater control over this learning process. This control can be applied in various ways, such as in selecting the type of xAI system to use, varying the parameters of the chosen xAI system, offsetting the mask M to adjust the weight given to xAI feedback, alternating between xAI-guided and standard generator training, and selecting methods to combine xAI feedback with loss. I argue that xAI-GANs are a powerful way for users to gain greater control over the training process of GAN models, and that there are many avenues, applications, and extensions of this idea worth exploring in the future.

This chapter introduces xAI-GANs, a class of generative adversarial network (GAN) that leverage domain knowledge sourced from an explainable AI (xAI) system to provide "richer" feedback from the discriminator to the generator to enable more guided training and greater control. I next overview xAI systems and standard GAN training and then introduce the xAI-guided generator training algorithm, contrasting it's difference with standard generator training. To the best of my knowledge, xAI-GAN is the first GAN to utilize xAI feedback for training. I perform experiments using MNIST and Fashion MNIST datasets and show that xAI-GAN has an improvement in Fréchet Inception Distance of up to 23.18% as compared to standard GANs. In addition, I train xAI-GAN on the CIFAR10 dataset using only 20% of the data and compare it with standard GAN trained on 100% and show that xAI-GAN outperforms standard GAN even in this setting. I compare this work to the Differentiable Augmentation technique and show that xAI-GAN trained on 20% of the data outperforms standard GAN trained with Differential Augmentation. I further combine xAI-GAN with Differential Augmentation to produce even better results. There is a trade-off between data-efficiency, training time and quality of images in GANs and these experiments show that xAI-GAN$_{saliency}$ provides the best value out of the xAI systems compared. Ultimately, xAI-GAN may enable greater control over the GAN learning process - allowing for better performance as well as a better understanding of GAN learning.

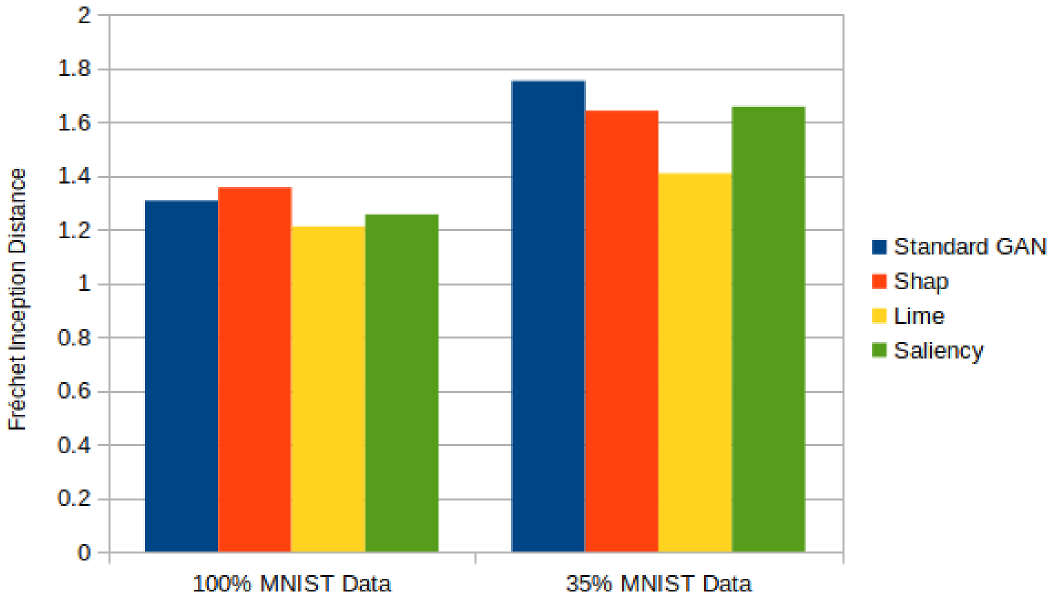

Figure 4.3: FID scores on MNIST Dataset

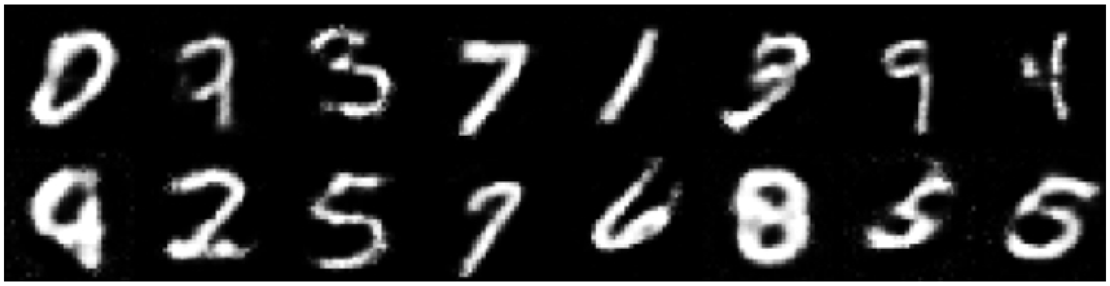

Figure 4.4: Sample of Images Generated by the xAI-GAN$_{lime}$ System using 35% Data on the MNIST Dataset

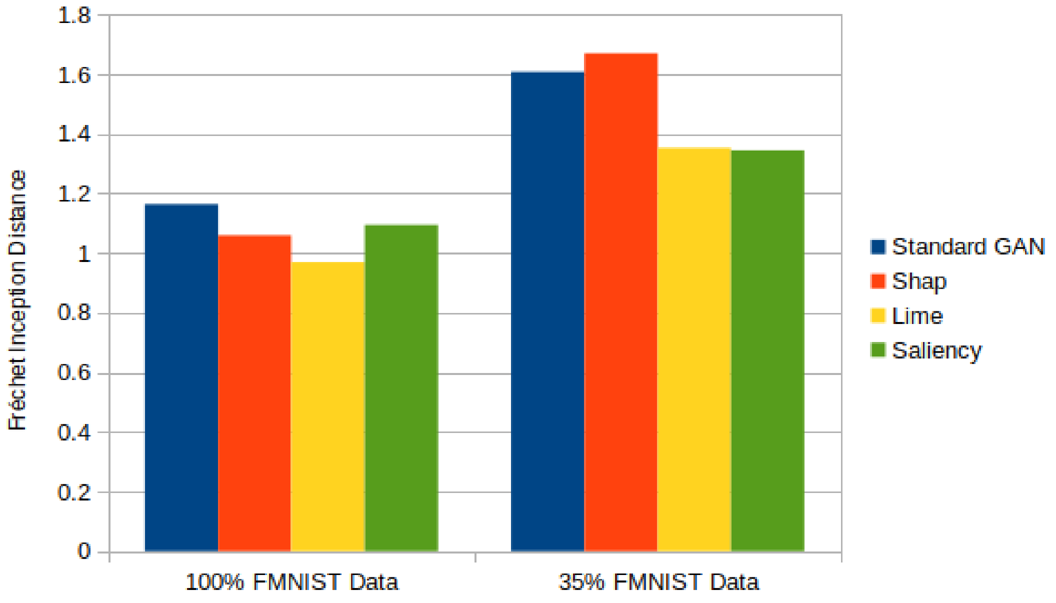

Figure 4.5: FID scores on Fashion MNIST Dataset

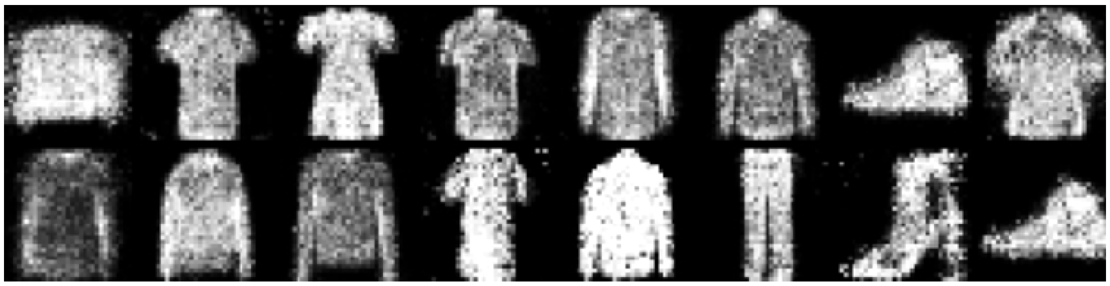

Figure 4.6: Sample of Images Generated by the xAI-GAN$_{shap}$ System using 35% Data on the Fashion MNIST Dataset

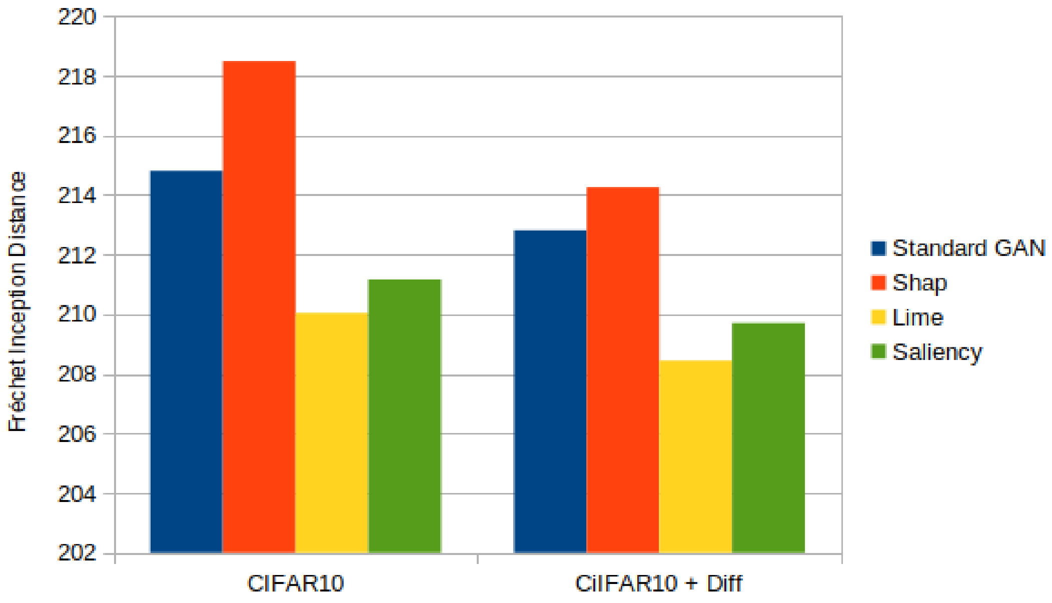

Figure 4.7: FID scores on the CIFAR10 Dataset

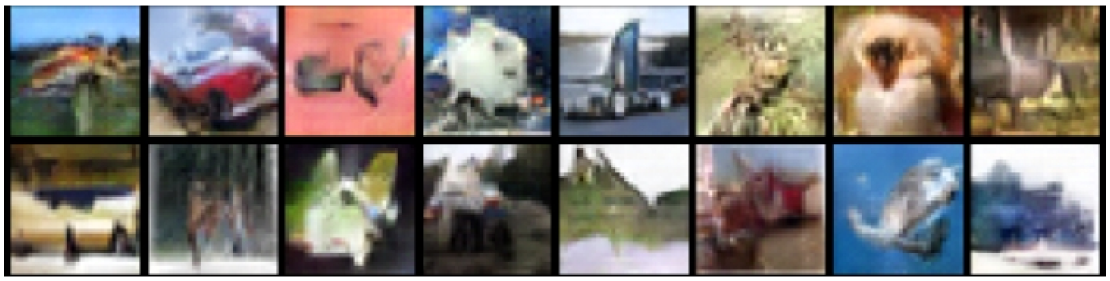

Figure 4.8: Sample of Images Generated by the xAI-GAN$_{saliency}$ System using 20% Data on CIFAR10 Dataset

Chapter 5

Conclusion

As DNNs are applied to increasingly safety-critical tasks, it is important to be able to test whether they comply with the given specifications. Case in point: DNNs have been proposed to be used as controllers for unmanned aerial vehicles (UAVs) [47]. In this setting, the specification for how the controllers should behave is provided by worldwide air traffic management regulatory bodies [97]. It is crucial to test whether a DNN trained on this task adheres to these specifications. Simultaneously, a different yet equally important task is to check whether a DNN is vulnerable to certain kinds of security attacks such as adversarial, model inversion, and availability attacks [54, 46, 95] or exhibit unwanted behaviours such as bias and unfairness [32]. For all of these cases, it is important to be able to find scalable and expressible methods to test, analyze and verify (TAV) DNNs. As shown in this thesis (specifically chapter 3), no current method exists and my work suggests that leveraging domain knowledge can help improve DNN testing by closing this gap.

Further, as DNNs are used in progressively complex tasks, the need for good quality training data is escalated. The recently introduced Wudao 2.0 is the largest language model ever trained and used around 5 terabytes of data for training [143]. The availability of training data is often considered a major bottleneck in machine learning research, and collecting good quality data often requires resources such as: time, money, and expert knowledge [126]. As a consequence, making models such as this more data efficient can allow for a huge reduction in training time, costs, and storage space requirements. In chapter 4, I show that leveraging domain knowledge provided in the form of neuron importance can help DNNs (particularly GANs) become more data-efficient. While this method is only applied to GANs in this work because of their distinct architecture, it can be applied to standard DNNs as long as we receive neuron importance information.

This thesis shows that leveraging domain knowledge can improve testing and training of DNNs, and help mitigate some of their weaknesses described above. More specifically, I present two novel algorithms that leverage domain knowledge in order to better test and train DNNs. In Chapter 3, I provided background information on gradient descent - the process by which DNNs learn as well as information on leveraging domain knowledge gathered from two sources: explainable AI (xAI) systems as well as logical constraints. In chapter 3, I introduced the constrained gradient descent (CGD) method that uses domain knowledge provided in the form of logical constraints to create a DNN testing, analysis and verification (TAV) method that is simultaneously scalable and expressible so as to be able to test for a rich set of properties in industrial sized DNNs. In chapter 4, I presented the xAI-guided gradient descent and the resultant GAN that incorporates domain knowledge provided in the form of neuron importance to make GANs more data-efficient.

Turing award winner Leslie Valiant famously said that a key challenge of Computer Science, and specifically Artificial Intelligence, is to combine learning with formal reasoning. He argued, "the aim is to identify a way of looking at and manipulating commonsense knowledge that is consistent with and can support what we consider to be the two most fundamental aspects of intelligent cognitive behaviour: the ability to learn from experience and the ability to reason from what has been learned. We are therefore seeking a semantics of knowledge that can computationally support the basic phenomena of intelligent behaviour" [39].

To that end, the field of neuro-symbolic AI (and the related fields of neuro-symbolic reasoning and neuro-symbolic learning) aim to integrate reasoning capability in machine learning models to realize this vision [6, 27, 58, 73, 151]. Applications of this field include creating specialized DNN models that can reason over existing knowledge to answer complex questions [69] and end-to-end task learning [73]. A number of these applications require the integration of domain knowledge in DNNs. As a consequence, the methods introduced in this thesis can allow for advances in the field of neuro-symbolic AI, and help have a transformative effect on the field of AI as a whole.

5.1 Future Work

The works described in this thesis require domain knowledge to be provided in a specific format for each use case. In the case of CGD, domain knowledge has to be provided as logical constraints that are differentiable. As for xAI-guided gradient descent, domain knowledge is required to be given as a neuron importance matrix for a particular layer, and so is suitable for the GAN setting where the feedback from a discriminator DNN is provided to a specific layer of the generator DNN. Currently, we do not understand well the trade-offs imposed by the format used to represent domain knowledge. As such, an interesting line of research would be to better understand how the format of domain knowledge used affects the quality of DNN testing and training.

Further, some formats can allow for more representation power over others. For example, we know that constraints provided as first order logic have more expressibility than constraints provided in boolean logic. If one can devise methods of integrating constraints provided in first order logic to DNNs, that would allow to encode a richer set of domain knowledge and can potentially improve DNN training and/or testing. Investigating methods of converting domain knowledge from one format to another may also be fruitful. And of course, finding other methods of injecting domain knowledge, and other formats of representing domain knowledge can be interesting avenues of future work. They may help incorporate more domain knowledge, or inject it with greater success into DNNs.

Note that injecting domain knowledge in machine learning models is not limited to just the DNNs [149, 29, 83, 74]. In fact, in other settings, injecting domain knowledge can be more straightforward, especially for more interpretable ML models such as decision trees and regression models. Finding ways to improve the integration of domain knowledge in other ML models is also an interesting and useful future work opportunity.

In this work, I show that the constrained gradient descent (CGD) method is scalable and expressible and I test DNNs for various adversarial robustness specifications. While this is valuable, we know that there exist other equally important specifications that we can test DNNs for, such as testing whether they adhere to certain specifications, and checking whether they exhibit unfair behaviour or bias. Using CGD to test for these properties can be a useful area of future research.

Finally, while the methods introduced in this thesis are empirically shown to produce great results, we do not have any formal guarantees whether the DNN is fully consistent with the given domain knowledge. Interpretability of DNNs is an open area of research, and our current understanding of what exactly DNNs learn is limited. Improving interpretability and explainability of DNNs can help us better isolate the effect of injecting domain knowledge in DNNs, and in doing so may allow us to better check whether a DNN is consistent with the given domain knowledge. This may also allows us to create new (or improve existing) methods of injecting domain knowledge to DNNs.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[3] Parnian Afshar, Shahin Heidarian, Farnoosh Naderkhani, Anastasia Oikonomou, Konstantinos N Plataniotis, and Arash Mohammadi. Covid-caps: A capsule network-based framework for identification of covid-19 cases from x-ray images. *Pattern Recognition Letters*, 138:638–643, 2020.

[4] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

[5] Jamil AlAgha. Expert system neural fuzzy system.

[6] Kay R Amel. From shallow to deep interactions between knowledge representation, reasoning and machine learning. In *Proceedings 13th International Conference Scala Uncertainity Mgmt (SUM 2019), Compiegne, LNCS*, pages 16–18, 2019.

[7] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2018.

[8] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[9] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[10] Alejandro Baldominos, Adráan Puello, Hasan Oğul, Tunç Aşuroğlu, and Ricardo Colomo-Palacios. Predicting infections using computational intelligence–a systematic review. *IEEE Access*, 8:31083–31102, 2020.

[11] Mislav Balunović, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. Certifying geometric robustness of neural networks. *Advances in Neural Information Processing Systems 32*, 2019.

[12] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*, pages 2613–2621, 2016.

[13] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *International Conference on Learning Representations*, 2018.

[14] Or Biran and Courtenay Cotton. Explanation and justification in machine learning: A survey. In *IJCAI-17 workshop on explainable AI (XAI)*, volume 8, page 1, 2017.

[15] Andrea Borghesi, Federico Baldo, and Michela Milano. Improving deep learning models via constraint-based domain knowledge: a brief survey. *arXiv preprint arXiv:2005.10691*, 2020.

[16] Rajdeep Borgohain and Sugata Sanyal. Rule based expert system for diagnosis of neuromuscular disorders. *International Journal of Advanced Networking and Applications*, 4(1):1509, 2012.

[17] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[18] Marko Kleine Büning, Philipp Kern, and Carsten Sinz. Verifying equivalence properties of neural networks with relu activation functions. In *International Conference on Principles and Practice of Constraint Programming*, pages 868–884. Springer, 2020.

[19] Yanshuai Cao, Gavin Weiguang Ding, Kry Yik-Chau Lui, and Ruitong Huang. Improving GAN training via binarized representation entropy (BRE) regularization. In *International Conference on Learning Representations*, 2018.

[20] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[21] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[22] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[23] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[24] Chih-Hong Cheng, Frederik Diehl, Gereon Hinz, Yassine Hamza, Georg Nührenberg, Markus Rickert, Harald Ruess, and Michael Truong-Le. Neural networks for safety-critical applications—challenges, experiments and perspectives. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1005–1006. IEEE, 2018.

[25] Joseph Paul Cohen, Paul Morrison, Lan Dao, Karsten Roth, Tim Duong, and Marzyeh Ghassemi. Covid-19 image data collection: Prospective predictions are the future. *MELBA*, page 18272, 2020.

[26] S. Das, P. K. Ghosh, and S. Kar. Hypertension diagnosis: A comparative study using fuzzy expert system and neuro fuzzy system. In *2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–7, 2013.

[27] Luc De Raedt, Sebastijan Dumancic, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In *IJCAI*, 2020.

[28] Thomas Demeester, T Rocktäschel, and S Riedel. Lifted rule injection for relation embeddings. In *2016 Conference on Empirical Methods in Natural Language Processing*, pages 1389–1399, 2016.

[29] Changyu Deng, Xunbi Ji, Colton Rainey, Jianyu Zhang, and Wei Lu. Integrating machine learning with human knowledge. *Iscience*, page 101656, 2020.

[30] Michelangelo Diligenti, Marco Gori, and Claudio Sacca. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165, 2017.

[31] Priya L Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5490–5500, 2017.

[32] Mengnan Du, Fan Yang, Na Zou, and Xia Hu. Fairness in deep learning: A computational perspective. *IEEE Intelligent Systems*, 2020.

[33] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[34] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 214–226, 2012.

[35] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.0396*, 2015.

[36] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. Dl2: Training and querying neural networks with logic. In *International Conference on Machine Learning*, pages 1931–1941, 2019.

[37] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019.

[38] Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694, 2020.

[39] Artur d'Avila Garcez and Luis C Lamb. Neurosymbolic ai: the 3rd wave. *arXiv preprint arXiv:2012.05876*, 2020.

[40] Artur S Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77, 1999.

[41] Artur S Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77, 1999.

[42] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[43] Biraja Ghoshal and Allan Tucker. Estimating uncertainty and interpretability in deep learning for coronavirus (covid-19) detection. *arXiv preprint arXiv:2003.10769*, 2020.

[44] Ian Goodfellow, Nicolas Papernot, Patrick McDaniel, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, et al. cleverhans v0. 1: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 1, 2016.

[45] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[46] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *stat*, 1050:20, 2014.

[47] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[48] Dario Guidotti, Francesco Leofante, Luca Pulina, and Armando Tacchella. Verification of neural networks: Enhancing scalability through pruning. In *ECAI 2020*, pages 2505–2512. IOS Press, 2020.

[49] Dario Guidotti, Luca Pulina, and Armando Tacchella. Never 2.0: Learning, verification and repair of deep neural networks. *arXiv preprint arXiv:2011.09933*, 2020.

[50] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[51] David Gunning. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2, 2017.

[52] Awni Y Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H Tison, Codie Bourn, Mintu P Turakhia, and Andrew Y Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature medicine*, 25(1):65–69, 2019.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[54] Yingzhe He, Guozhu Meng, Kai Chen, Xingbo Hu, and Jinwen He. Towards security threats of deep learning systems: A survey. *IEEE Transactions on Software Engineering*, 2020.

[55] Christoph S Herrmann. A hybrid fuzzy-neural expert system for diagnosis. In *IJCAI*, pages 494–501. Citeseer, 1995.

[56] Marijn Heule and Matti Jarvisalo. Sat competition 2020, 2020.

[57] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[58] Melanie Hilario. An overview of strategies for neurosymbolic integration. *Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, pages 13–36, 1997.

[59] Bhadeshia Hkdh. Neural networks in materials science. *ISIJ international*, 39(10):966–979, 1999.

[60] Roger A Horn. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pages 87–169, 1990.

[61] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H Hovy, and Eric P Xing. Harnessing deep neural networks with logic rules. In *ACL (1)*, 2016.

[62] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[63] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1511–1519, 2019.

[64] Jeremy Irvin, Pranav Rajpurkar, Michael Ko, Yifan Yu, Silviana Ciurea-Ilcus, Chris Chute, Henrik Marklund, Behzad Haghgoo, Robyn Ball, Katie Shpanskaya, et al. Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 590–597, 2019.

[65] Peter Jackson. *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[66] Kai Jia and Martin Rinard. Efficient exact verification of binarized neural networks. *arXiv preprint arXiv:2005.03597*, 2020.

[67] Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa. Food detection and recognition using convolutional neural network. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1085–1088, 2014.

[68] Bhavya Kailkhura, Brian Gallagher, Sookyung Kim, Anna Hiszpanski, and T Yong-Jin Han. Reliable and explainable machine-learning methods for accelerated material discovery. *npj Computational Materials*, 5(1):1–9, 2019.

[69] Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramon Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, et al. Question answering over knowledge bases by leveraging semantic parsing and neuro-symbolic reasoning. *arXiv preprint arXiv:2012.01707*, 2020.

[70] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[71] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[72] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The

marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[73] Kenji Kawaguchi. On the theory of implicit deep learning: Global convergence with implicit layers. In *International Conference on Learning Representations*, 2020.

[74] Sebo Kim, Varsha Sundaresan, Lei Zhou, and Tamer Kahveci. Integrating domain specific knowledge and network analysis to predict drug sensitivity of cancer cell lines. *PloS one*, 11(9):e0162173, 2016.

[75] Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, pages 1–4, 2012.

[76] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[77] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Jonathan Reynolds, Alexander Melnikov, Natalia Lunova, and Orion Reblitz-Richardson. Pytorch captum. https://github.com/pytorch/captum, 2019.

[78] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). *Unpublished Manuscript*, 2010.

[79] Reeda Kunhimangalam, Sujith Ovallath, and Paul K Joseph. A novel fuzzy expert system for the identification of severity of carpal tunnel syndrome. *BioMed research international*, 2013, 2013.

[80] Lindsey Kuper, Guy Katz, Justin Gottschlich, Kyle Julian, Clark Barrett, and Mykel Kochenderfer. Toward scalable verification for safety-critical deep networks. *arXiv preprint arXiv:1801.05950*, 2018.

[81] Lindsey Kuper, Guy Katz, Justin Gottschlich, Kyle Julian, Clark Barrett, and Mykel Kochenderfer. Toward scalable verification for safety-critical deep networks. *arXiv preprint arXiv:1801.05950*, 2018.

[82] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[83] Fabien Lauer and Gérard Bloch. Incorporating prior knowledge in support vector machines for classification: A review. *Neurocomputing*, 71(7-9):1578–1594, 2008.

[84] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[85] Francesco Leofante, Nina Narodytska, Luca Pulina, and Armando Tacchella. Automated verification of neural networks: Advances, challenges and perspectives. *arXiv preprint arXiv:1805.09938*, 2018.

[86] Bruno Lepri, Jacopo Staiano, David Sangokoya, Emmanuel Letouzé, and Nuria Oliver. The tyranny of data? the bright and dark sides of data-driven decision-making for social good. In *Transparent data mining for big and small data*, pages 3–24. Springer, 2017.

[87] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4, 2020.

[88] Wenqing Liu, Miaojing Shi, Teddy Furon, and Li Li. Defending adversarial examples via dnn bottleneck reinforcement. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1930–1938, 2020.

[89] Zihao Liu, Qi Liu, Tao Liu, Nuo Xu, Xue Lin, Yanzhi Wang, and Wujie Wen. Feature distillation: Dnn-oriented jpeg compression against adversarial examples. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 860–868. IEEE, 2019.

[90] Marc Moreno Lopez and Jugal Kalita. Deep learning applied to nlp. *arXiv preprint arXiv:1703.03091*, 2017.

[91] Marc Moreno Lopez and Jugal Kalita. Deep learning applied to nlp. *arXiv preprint arXiv:1703.03091*, 2017.

[92] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709, 2018.

[93] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[94] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[95] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[96] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[97] Guido Manfredi and Yannick Jestin. An introduction to acas xu and the challenges ahead. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–9. IEEE, 2016.

[98] Xudong Mao, Qing Li, Haoran Xie, Raymond Yiu Keung Lau, Zhen Wang, and Stephen Paul Smolley. On the effectiveness of least squares generative adversarial networks. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[99] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[100] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? *arXiv preprint arXiv:1801.04406*, 2018.

[101] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks, 2016.

[102] Thomas D Meyers, Ladislav Vagner, Katarina Janoskova, Iulia Grecu, and Gheorghe Grecu. Big data-driven algorithmic decision-making in selecting and managing employees: Advanced predictive analytics, workforce metrics, and digital innovations for enhancing organizational human capital. *Psychosociological Issues in Human Resource Management*, 7(2):49–54, 2019.

[103] Fatemehsadat Mirshghallah, Mohammadkazem Taram, Praneeth Vepakomma, Abhishek Singh, Ramesh Raskar, and Hadi Esmaeilzadeh. Privacy in deep learning: A survey. *arXiv preprint arXiv:2004.12254*, 2020.

[104] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[105] Christoph Molnar. *Interpretable Machine Learning*. self, 2019. https://christophm.github.io/interpretable-ml-book/.

[106] Christoph Molnar. *Interpretable Machine Learning*. Lulu. com, 2020.

[107] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[108] Abdel-Badeeh M.Salem, Mohamed Roushdy, and Rania Hodhod. A case based expert system for supporting diagnosis of heart diseases. *ICGST International Journal of Artificial Intelligence and Machine Learning AIML*, V1:33–39, 12 2004.

[109] Vineel Nagisetty, Laura Graves, Joseph Scott, and Vijay Ganesh. xAI-GAN: Enhancing generative adversarial networks via explainable ai systems. *AAAI Conference on Artificial Intelligence - Explainable Agency in AI Workshop*, 2021.

[110] Nina Narodytska. Formal analysis of deep binarized neural networks. In *IJCAI*, pages 5692–5696, 2018.

[111] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[112] Siegfried Nijssen and Elisa Fromont. Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 530–539, 2007.

[113] Zhaoqing Pan, Weijie Yu, Xiaokai Yi, Asifullah Khan, Feng Yuan, and Yuhui Zheng. Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 7:36322–36333, 2019.

[114] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE, 2016.

[115] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch, 2017.

[116] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[117] Jigneshkumar L Patel and Ramesh K Goyal. Applications of artificial neural networks in medical science. *Current clinical pharmacology*, 2(3):217–226, 2007.

[118] Brandon Paulsen, Jingbo Wang, and Chao Wang. Reludiff: Differential verification of deep neural networks. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 714–726. IEEE, 2020.

[119] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18, 2017.

[120] Nikolaos Pitropakis, Emmanouil Panaousis, Thanassis Giannetsos, Eleftherios Anastasiadis, and George Loukas. A taxonomy and survey of attacks against machine learning. *Computer Science Review*, 34:100199, 2019.

[121] Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

[122] Foster Provost and Tom Fawcett. Data science and its relationship to big data and data-driven decision making. *Big data*, 1(1):51–59, 2013.

[123] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.

[124] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[125] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[126] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: a big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[127] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[128] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[129] Joseph Scott, Maysum Panju, and Vijay Ganesh. LGML: logic guided machine learning (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13909–13910, 2020.

[130] Christin Seifert, Aisha Aamir, Aparna Balagopalan, Dhruv Jain, Abhinav Sharma, Sebastian Grottel, and Stefan Gumhold. Visualizations of deep neural networks in computer vision: A survey. In *Transparent data mining for big and small data*, pages 123–144. Springer, 2017.

[131] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

[132] Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. *arXiv preprint arXiv:1805.03364*, 2018.

[133] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[134] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3145–3153. JMLR. org, 2017.

[135] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[136] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T Vechev. Fast and effective robustness certification. *NeurIPS*, 1(4):6, 2018.

[137] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[138] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T Vechev. Boosting robustness certification of neural networks. In *ICLR (Poster)*, 2019.

[139] Matthew Sotoudeh and A Thakur. Correcting deep neural networks with small, generalizing patches. In *Workshop on Safety and Robustness in Decision Making*, 2019.

[140] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[141] Mu-Chun Su. Use of neural networks as medical diagnosis expert systems. *Computers in biology and medicine*, 24(6):419–429, 1994.

[142] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*, 2018.

[143] Market Research Telecast. What is wudao 2.0, china's artificial intelligence model capable of writing poems and generating recipes that surpassed google and musk's openai. *Market Research Telecast*, Jun 2021.

[144] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[145] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-based reachability analysis of deep neural networks. In *International Symposium on Formal Methods*, pages 670–686. Springer, 2019.

[146] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. *arXiv preprint arXiv:2004.05519*, 2020.

[147] Petra Vidnerová and Roman Neruda. Vulnerability of machine learning models to adversarial examples. In *ITAT*, pages 187–194, 2016.

[148] The Verification of Neural Networks Library (VNN-LIB). `www.vnnlib.org`, 2019.

[149] Laura Von Rueden, Sebastian Mayer, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. Informed machine learning–towards a taxonomy of explicit integration of knowledge into machine learning. *learning*, 18:19–20, 2019.

[150] Hai Wang and Hoifung Poon. Deep probabilistic logic: A unifying framework for indirect supervision. *arXiv preprint arXiv:1808.08485*, 2018.

[151] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pages 6545–6554. PMLR, 2019.

[152] TJ Wang, AK Skidmore, and AG Toxopeus. Improved understorey bamboo cover mapping using a novel hybrid neural network and expert system. *International journal of remote sensing*, 30(4):965–981, 2009.

[153] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.

[154] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018.

[155] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[156] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International Conference on Machine Learning*, pages 5502–5511. PMLR, 2018.

[157] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International Conference on Machine Learning*, pages 5502–5511, 2018.

[158] Xin Yi, Ekta Walia, and Paul Babyn. Generative adversarial network in medical imaging: A review. *Medical image analysis*, page 101552, 2019.

[159] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

[160] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.

[161] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

[162] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. *arXiv preprint arXiv:2006.10738*, 2020.