

Adversarial Examples for Visual Decompilers

James Wei



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-81

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-81.html>

May 12, 2017

Copyright © 2017, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to thank my advisor, Dawn Song, for her guidance during my undergraduate and graduate careers. I would also like to thank Alyosha Efros for his advice, and for introducing me to computer vision and machine learning. Moreover, this work would not have been possible without my collaborators, Warren He and Mitar Milutinović. Finally, I would like to thank my friends at 2609, in whom I found my second family, and my parents, whose love and support will always continue to inspire me.

Adversarial Examples for Visual Decompilers

by James Wei

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

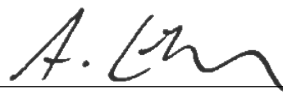
Committee:



Professor Dawn Song
Research Advisor

May 12, 2017

(Date)



Professor Alexei A. Efros
Second Reader

May 12, 2017

(Date)

Adversarial Examples for Visual Decompilers

by

James Chachen Wei

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Dawn Song, Chair

Professor Alexei A. Efros

Spring 2017

Abstract

Adversarial Examples for Visual Decompilers

by

James Chachen Wei

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Dawn Song, Chair

Deep learning models are vulnerable to adversarial examples: maliciously perturbed inputs that compel models to make incorrect predictions with high confidence. We present an analysis of adversarial examples in the context of visual decompilers. Using the image-to-L^AT_EX task as a baseline for structured prediction problems, we show that targeted and non-targeted adversarial examples can fool the model using a minimal amount of perturbations. Additionally, we apply and discuss the limitations of two detection schemes. Finally, we propose—and subsequently break—two prevention strategies, one of which involves a novel attack for quantized adversarial examples.

Contents

Contents	ii
Acknowledgements	iii
1 Introduction	1
2 Problem Definition and Motivation	3
2.1 Adversarial Examples	3
2.2 Optical Character Recognition	3
2.3 The Image-to-L ^A T _E X Task	4
2.4 What You Get Is What You See	4
3 Background and Related Work	6
3.1 Generating Adversarial Examples	6
3.2 Transferability	8
3.3 Detection	9
3.4 Prevention	9
4 Model Evaluation	10
4.1 WYGIWYS Model Architecture	10
4.2 Network Sparsity	10
4.3 Visualizing Network Activations	11
4.4 Adding More Data	11
5 Adversarial Examples for Image-to-L^AT_EX	13
5.1 Non-targeted Adversarial Examples	13
5.2 Targeted Adversarial Examples	14
5.3 Regularization	17
5.4 Activation Differences	19
5.5 Robustness Test	19
6 Attack Detection	22
6.1 PCA Comparison	22
6.2 Out-of-distribution Softmaxes	23

7	Defenses and Their Limitations	27
7.1	Types of Adversaries	27
7.2	Segmentation	27
7.3	Input Quantization	29
8	Future Work	34
9	Conclusion	36
	Bibliography	37

Acknowledgements

I would like to thank my advisor, Dawn Song, for her guidance during my undergraduate and graduate careers. I would also like to thank Alyosha Efros for his advice, and for introducing me to computer vision and machine learning. Moreover, this work would not have been possible without my collaborators, Warren He and Mitar Milutinović. Finally, I would like to thank my friends at 2609, in whom I found my second family, and my parents, whose love and support will always continue to inspire me.

Chapter 1

Introduction

Advancements in the architectures of deep neural networks have revolutionized data modeling and analysis across a wide number of applications. However, recent analysis indicates that deep learning models are vulnerable to *adversarial examples*: inputs generated by adding small, worst-case perturbations that force the model to produce incorrect answers with high confidence [21, 4]. For image inputs, successful adversarial perturbations can be so small that they are virtually imperceptible to the human eye.

Researchers have discovered that adversarial examples are pervasive and can easily be found using a number of gradient- and optimization-based methods. Moreover, adversarial examples are transferable [18, 12]: malicious inputs for one model can be used to compromise another designed for the same task, even if there are significant differences in the models' architectures or training methods.

As the use of deep learning in practical applications grows more ubiquitous, many efforts have tried to address the problem of adversarial examples. Some propose methods for detecting the presence of malicious inputs, while others investigate ways to make networks robust against adversarial noise. However, as the research community evaluates these detection schemes and defenses, they find that strategies for thwarting adversarial examples are limited in a number of ways. Common flaws include a failure to generalize beyond simple tasks, easy circumvention, and detrimental performance trade-offs on non-adversarial inputs. As such, the issue of detecting and defending against adversarial examples remains an open question.

Despite the growing research interest in adversarial examples, a large portion of the literature evaluates adversarial examples on simple image classification problems (e.g., MNIST, CIFAR-10). Given that the use of neural networks has progressed far beyond basic classification, it is important to consider adversarial examples in a wider scope. In particular, deep learning has been shown to be very successful for *structured prediction tasks*, as evidenced by their application to breakthroughs in natural language processing, bioinformatics, and computer vision.

In this work, we extend the study of adversarial examples to a class of structured prediction models: visual decompilers. Visual decompilers take compiled images as input and return sequences of high-level markup tokens. Specifically, we study the image-to- \LaTeX task, where images of compiled \LaTeX equations are decompiled into their corresponding \LaTeX source code. We focus on the image-to- \LaTeX task because of several interesting prop-

erties that make it a useful baseline for structured prediction tasks in general; an extended motivation for the image-to-L^AT_EX task is given in Chapter 2.

Adversarial examples for visual decompilers can be designed to manipulate the performance of the model. By adding trace amounts of adversarial perturbations to the input image, an attacker can fool the model into producing uncompileable markup or an arbitrary output sequence of the attacker’s choosing. Figure 1.1 shows how a visual decompiler responds differently to valid and adversarial inputs.

We summarize our contributions in this work as follows:

- We adapt methods for generating adversarial examples to visual decompilers. We show that the perturbations required to produce successful adversarial examples are visually imperceptible.
- We consider two recently published detection schemes for adversarial examples and demonstrate their limitations with respect to visual decompilers.
- We propose—and subsequently break—two defenses against adversarial examples. In the process, we introduce a novel method for crafting quantized adversarial examples.

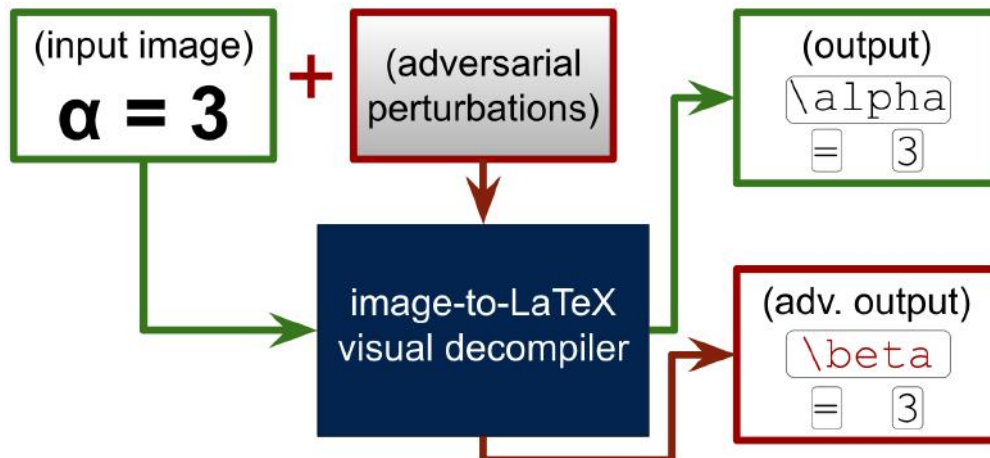


Figure 1.1: Adversarial examples for visual decompilers.

Chapter 2

Problem Definition and Motivation

We begin with a discussion of several important concepts that motivate this work.

2.1 Adversarial Examples

The existence of adversarial examples presents a salient challenge for machine learning researchers. From a theoretical perspective, the abundance of adversarial examples underscores the brittleness of state-of-the-art models. Furthermore, adversarial examples in image space demonstrate a fundamental disconnect between human visual understanding and the artificial representations captured by deep networks. From a practical perspective, this problem presents a potent attack surface for real-world adversaries. As more applications start to make use of deep neural networks, adversarial examples threaten the integrity of digital content filters, biometric authentication systems, autonomous vehicles, and more.

Determining the cause of adversarial examples and formulating a generalizable defense strategy remain open questions. With a better understanding of adversarial examples, we hope to simultaneously mitigate their impact on deep neural networks and improve our ability to create robust models.

2.2 Optical Character Recognition

Optical character recognition (OCR) refers to the broad task of converting images of text into some machine-interpretable encoding. Modern-day OCR systems first emerged as a subset of shape detection in signal processing. Early approaches involve filtering input signals in Fourier space [14]. Additional work has significantly improved the quality of these conventional models by means of integrating character segmentation [13] and generalizing to handwritten characters [5].

With the rapid adoption of convolutional and recurrent neural networks, the performance of OCR models has significantly improved. Not only do these deep learning approaches achieve higher character-by-character accuracies, they also understand a larger range of inputs (e.g., text extraction and recognition from scene images [22]). In addition to their ability to capture visual patterns, deep learning approaches can also learn representations of the underlying language model to help produce coherent transcriptions.

2.3 The Image-to-L^AT_EX Task

For the purpose of studying adversarial examples for structured prediction tasks, we choose to focus on a specific OCR subtask: image-to-L^AT_EX. A model for this task takes images of compiled L^AT_EX equations as input and produces the corresponding L^AT_EX markup. L^AT_EX is a powerful typesetting language that is capable of accessing files on the system that compiles it. In some configurations, it can even run shell commands. Thus, it is important for a L^AT_EX decompiler system to be robust against adversarial examples because an unattended deployment may emit malicious L^AT_EX source code which unsuspecting users will try to compile. At best, this could lead to an incorrect formula, and at worst, a system compromise.

Beyond the practical system security concerns, we choose to study the image-to-L^AT_EX task because of several interesting properties that are important for research:

- The input space for the image-to-L^AT_EX task is highly constrained. Compared to natural images used for text transcription, image captioning, and object detection, images of compiled L^AT_EX equations contain significantly fewer degrees of freedom. In general, they are encoded by a single channel, contain minimal amounts of noise, display glyphs from a well-defined vocabulary, and reflect a strict syntactic structure.
- Image-to-L^AT_EX comes with a perfect function for the reverse of the task. By rendering the output markup using a standard L^AT_EX compiler, we can directly compare the input image with the rendered output, which is helpful for detecting adversarial examples and for improving the robustness of the model in general.
- It is easy to augment the existing dataset with novel samples of varying size and complexity.

We hope that understanding adversarial examples in a tightly controlled environment like image-to-L^AT_EX can serve as a useful baseline for exploring and evaluating generalizable defenses.

For our experiments, we use the `im2latex-100k` dataset [9], a collection of 94,630 L^AT_EX equation images and their corresponding markup, extracted from papers in the HEP-Th (high energy physics – theory) portion of arXiv between 1992 and 2003. We also collect and evaluate our own image-to-L^AT_EX dataset that contains 256,742 equations from a different portion of arXiv; additional details about the augmented dataset will be given in Chapter 4.

2.4 What You Get Is What You See

In order to evaluate attacks, detection strategies, and defenses for adversarial examples, we use Deng et al.’s visual markup decompiler system “What You Get Is What You See” (WYGIWYS) [3].

The model uses a six-layer convolutional neural network to capture visual features. The output of the CNN is passed into a bilinear LSTM encoder, which produces a new feature grid. Finally, the feature grid is converted to the generated markup output using a bilinear LSTM decoder with attention. By leveraging both a CNN and an RNN, the WYGIWYS model

learns visual characteristics of different glyphs in the \LaTeX vocabulary while capturing an underlying language model for syntactically correct \LaTeX equations.

At the time of writing, the WYGIWYS model is the state-of-the-art solution for the image-to- \LaTeX task. A summary of the model’s performance is given in Table 2.1. The original WYGIWYS model was built using Torch; for our experiments, we use a Tensorflow reimplementation [11]. We evaluate the Tensorflow implementation and find that it matches the performance of the Torch model.

BLEU score	0.8773
Exact match	0.7746

Table 2.1: Performance of the WYGIWYS model on the `im2latex-100k` test set. Results are taken from [3].

Chapter 3

Background and Related Work

Since their introduction in seminal work by Szegedy et al. [21] and Goodfellow et al. [4], adversarial examples for deep learning models have been explored in a number of different contexts. Key results indicate that adversarial examples are pervasive, transferable across several major network configurations (e.g., MNIST, QuocNet, AlexNet), and can be found using gradient- and optimization-based methods.

With the hope of making neural networks more robust, many additional efforts have been made to (1) efficiently find vulnerable samples in a given input space, (2) attack a variety of network architectures, (3) detect the presence of maliciously perturbed inputs, and (4) formulate defenses against adversarial examples.

3.1 Generating Adversarial Examples

Suppose we have a target classifier F with model parameters θ . Let x be an input to the classifier with corresponding ground truth label y . For a well-trained classifier, $F_\theta(x)$ outputs a category or label that matches y with high probability. An adversarial example x^* is some instance in the input space that is close to x , but causes F_θ to produce an incorrect output. Prior work considers two classes of adversarial examples.

First, a *non-targeted* adversarial example is an x^* that causes the target classifier to produce any incorrect output: $F_\theta(x^*) \neq y$. Second, a *targeted* adversarial example is an x^* that causes the target classifier to produce a specific incorrect output y^* : $F_\theta(x^*) = y^*$ where $y \neq y^*$.

Adversarial examples can be trivially constructed by setting x^* equal to the value of some other sample x' in the training set with a different label $y' \neq y$. As such, we enforce an additional constraint $d(x, x^*) \leq \epsilon$, where $d(\cdot, \cdot)$ quantifies some distance metric between an input sample x and its corresponding adversarial example x^* . ϵ should be sufficiently small so that the generated adversarial perturbation does not change the ground truth identity of input. For image inputs, this means the perturbation should be small enough that a human can still interpret the image correctly.

In general, adversarial examples are created by learning some small perturbation δ that is added to the original input: $x^* = x + \delta$. Several methods for generating non-targeted and targeted adversarial examples are described below.

3.1.1 Fast gradient sign method

Goodfellow et al. propose the *fast gradient sign method* [4] for generating adversarial examples. The adversarial examples found using this method can be computed quickly, but are not optimized; there may exist many other adversarial examples for a particular x that trick the model into making the same misclassification but require a smaller amount of perturbations.

For a non-targeted attack, the fast gradient sign method computes the adversarial perturbation following the gradient sign of the model’s loss function:

$$\begin{aligned}\delta &= \epsilon \cdot \text{sign}(\nabla_x J(F_\theta(x), y)) \\ x^* &= \text{clip}(x + \delta)\end{aligned}$$

Here, $J(\cdot, \cdot)$ denotes the loss function that was used to train the model F_θ . ϵ controls the magnitude of the permitted perturbation; it functions as a bound on the ℓ_∞ norm of the perturbation. At a high level, this method computes the perturbation δ that yields the greatest increase in loss, thereby maximizing the probability of misclassification.

For the targeted case, rather than simply perturbing x away from the ground truth label y , we seek to find a perturbation that minimizes the loss of classifying x^* as a specific incorrect label y^* . Thus, we alter the formulation:

$$\begin{aligned}\delta &= -\epsilon \cdot \text{sign}(\nabla_x J(F_\theta(x), y^*)) \\ x^* &= \text{clip}(x + \delta)\end{aligned}$$

3.1.2 Fast gradient method

While the fast gradient sign method perturbs inputs in the gradient sign direction of the model’s loss function, the *fast gradient method* [12] perturbs inputs in the gradient direction. Both the fast gradient method and the fast gradient sign method are “one-step” algorithms. That is, both methods can produce an adversarial example after a single gradient computation. Later, we will explore optimization-based approaches that require more computation time but produce adversarial examples with higher average rates of success using comparatively fewer perturbations.

For the non-targeted case, the fast gradient method computes:

$$\begin{aligned}\delta &= \epsilon \cdot \frac{\nabla_x J(F_\theta(x), y)}{\|\nabla_x J(F_\theta(x), y)\|} \\ x^* &= \text{clip}(x + \delta)\end{aligned}$$

For the targeted case, the fast gradient method finds:

$$\begin{aligned}\delta &= -\epsilon \cdot \frac{\nabla_x J(F_\theta(x), y^*)}{\|\nabla_x J(F_\theta(x), y^*)\|} \\ x^* &= \text{clip}(x + \delta)\end{aligned}$$

The intuition for the targeted and non-targeted calculations follows from the fast gradient sign method.

3.1.3 Optimization method

In addition to the gradient-based methods described previously, Szegedy et al. show that adversarial examples can be found by iteratively optimizing a given objective function [21]. The optimization-based approach carries a larger computation cost compared to gradient-based methods, but produces adversarial examples that minimize the amount of perturbations.

For the non-targeted case, adversarial examples can be crafted by optimizing:

$$\delta = \underset{\delta}{\operatorname{argmin}} \lambda \|\delta\|_p - J(F_\theta(x + \delta), y)$$

Here, we seek to find the smallest perturbation that yields the greatest loss. $\|\delta\|_p$ is the p -norm of the perturbation; λ is a tunable hyperparameter denoting the regularization strength. Different choices of p -norms (e.g., ℓ_0 , ℓ_1 , ℓ_2 , ℓ_∞) generate adversarial examples with different properties. For instance, for image inputs, optimizing the ℓ_2 -norm creates adversarial examples that perturb a large number of pixels by a very small amount, whereas optimizing the ℓ_0 - or ℓ_1 -norm produces adversarial examples that maximally perturb a small number of pixels.

For the targeted case, we optimize:

$$\delta = \underset{\delta}{\operatorname{argmin}} \lambda \|\delta\|_p + J(F_\theta(x + \delta), y^*)$$

The initial experiments conducted by Szegedy et al. use a box-constrained L-BFGS optimizer to accomplish the minimizations above. However, more recent experiments by [2, 12] suggest that using the Adam optimizer [10] yields much faster convergence while producing adversarial examples of the same quality. For the experiments conducted in this work, we use the optimization-based approach to generate adversarial examples.

3.2 Transferability

The approaches for generating adversarial examples discussed previously require white-box access to a target model’s gradients. However, from a practical viewpoint, attackers in the real world typically only have black-box access to the system.

Building off this idea, several works investigate the creation of adversarial examples in black-box scenarios. Goodfellow et al. suggest that adversarial examples are *transferable*; adversarial examples for a single task can be used to compromise multiple models with different architectures and training procedures.

The issue of transferability is further studied by Papernot, McDaniel, and Goodfellow [18] and Liu et al [12]. Their investigations produce targeted and non-targeted adversarial examples for black-box models; this is achieved by training and attacking a substitute model using minimal information about the properties of the victim model’s architecture and behavior.

Evaluating the transferability of adversarial examples for visual decompilers falls outside of the scope of this work. Attacks and defenses will be discussed only in the context of a white-box scenario.

3.3 Detection

As a first step towards mitigating the impact of adversarial examples, a number of strategies have been proposed for distinguishing inputs perturbed with adversarial noise from valid inputs.

One body of work suggests that adversarial examples exist in a different principal component space than valid inputs. Hendrycks and Gimpel perform principal component analysis and compare the variance of the coefficients for clean and adversarial images [7]. They find that adversarial images emphasize low-ranked principal components much more than clean images. Using this method, they build detectors that achieve high AUROC and AUPR for three image classification tasks: Tiny-ImageNet, CIFAR-10, and MNIST.

Another approach for detecting adversarial examples is based on the idea that adversarial examples induce intermediate outputs in deep networks that are statistically distinct from the outputs produced by valid inputs. For instance, Hendrycks and Gimpel [6] suggest that it is possible to detect incorrectly classified samples by examining output softmax values because correctly classified samples tend to have greater maximum softmax probabilities than incorrectly classified samples. With this approach, they build baseline classifiers that detect incorrectly predicted samples for tasks in computer vision, natural language processing, and speech recognition.

Finally, adversarial examples can be detected by directly augmenting the network architecture of the target model. Metzen et al. propose training a detector subnetwork that distinguishes adversarial inputs from valid inputs based off the intermediate activations in the deep network [16].

3.4 Prevention

Beyond detecting adversarial examples, several ideas for defending networks against adversarial examples have been evaluated. However, they contain inherent flaws and have not been shown to generalize well across different tasks.

Goodfellow et al. introduce *adversarial retraining* as a first-order defense [4]. This strategy involves augmenting the training pipeline with adversarial examples found using fast gradient methods. While this defense is successful in reducing the effectiveness of known adversarial examples, it ceases to work once an adversary adopts a novel algorithm for constructing malicious inputs. Other defenses, like defensive distillation [20], have been proven to be easily circumventable by modifying a minimal number of training parameters [1].

Although a thorough investigation of detection and prevention schemes falls largely outside the scope of this work, we will briefly consider and evaluate a couple of detection strategies and defenses as a part of our discussion.

Chapter 4

Model Evaluation

As a first step, we investigate the image-to-L^AT_EX model that we use to carry out experiments in this work. These results help us better understand subsequent findings.

4.1 WYGIWYS Model Architecture

The WYGIWYS model is divided into three main components that are trained end-to-end. First, the model captures visual features using a six-layer convolutional neural network. Second, the filtered output of the CNN is encoded into a feature grid using a bilinear LSTM row encoder. Finally, a bilinear LSTM decoder with attention produces the markup output. Please refer to Deng et al.’s paper for additional details [3].

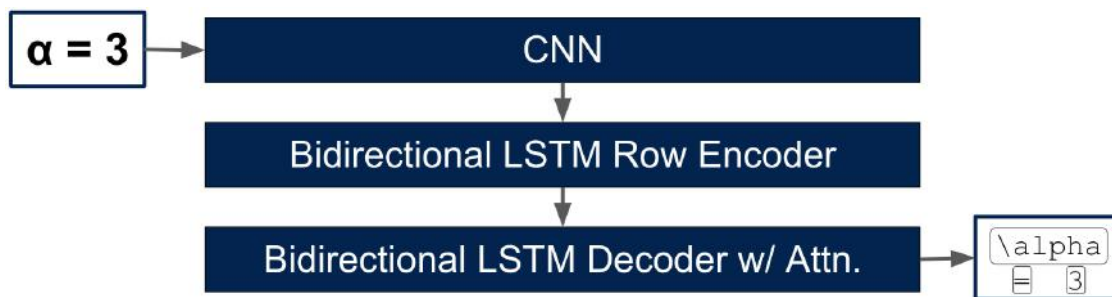


Figure 4.1: WYGIWYS architecture.

4.2 Network Sparsity

To better understand the visual representations being captured by the convolutional neural network at the front of the WYGIWYS model, we first record metrics about the sparsity of the six convolutional layers; the results are shown in Table 4.1. The values in Table 4.1 were computed by averaging the activations produced by 633 images from the test set portion of the `im2latex-100k` dataset [9]. Additional sparsity measurements were taken using adversarial images and images perturbed with random Gaussian noise. We find that

Layer	Num. filters	Filter height	Filter width	Activation density	Mean value	Mean activated value
1	64	50	120	0.3817	0.0653	0.2458
2	128	25	60	0.3336	0.0469	0.1708
3	256	13	30	0.6197	0.2787	0.4638
4	256	13	30	0.0417	0.0163	0.4383
5	512	13	15	0.6086	0.3095	0.5209
6	512	7	15	0.4421	0.3517	0.8096

Table 4.1: WYGIWYS model convolutional layer sparsity. “Activation density” refers to the average fraction of nodes that are turned on by the ReLU activation function for a single input image.

changing the type of input image (i.e., valid, adversarial, noisy) does not alter the activation density pattern.

It is interesting to note that while most layers are fairly dense (activation densities between 30% and 60%), the fourth convolutional layer yields very sparse activations (a little over 4% of the nodes).

4.3 Visualizing Network Activations

In addition to analyzing the sparsity of convolutional layers, we visualize the activations at each convolutional layer and at the bilinear LSTM row encoder. These visualizations can be viewed in Figure 4.2.

Most of the filters we observe are difficult to interpret visually; however, we find that the activations at the sparse convolutional layer (layer 4) are sensitive to the shapes of different glyphs in the equation image (see Figure 4.3). Thus, we hypothesize that successful adversarial examples for the image-to-L^AT_EX task will alter the activation patterns in filters corresponding to the glyph(s) under attack at this layer.

4.4 Adding More Data

The performance of many deep learning models can be improved by simply training on more data. We confirm that this holds true for the WYGIWYS model as well by assembling an augmented dataset that is more than 2.5 times larger than the dataset on which the WYGIWYS model was originally trained. A comparison between the datasets and subsequent model performance can be found in Table 4.2.

We constructed our new augmented dataset by collecting equations from arXiv papers published between November 2016 and December 2016 across all portions. For reference, the `im2latex-100k` dataset [9] originally used to train the WYGIWYS model was constructed using papers in the HEP-Th (high energy physics – theory) portion of arXiv between 1992 and 2003.

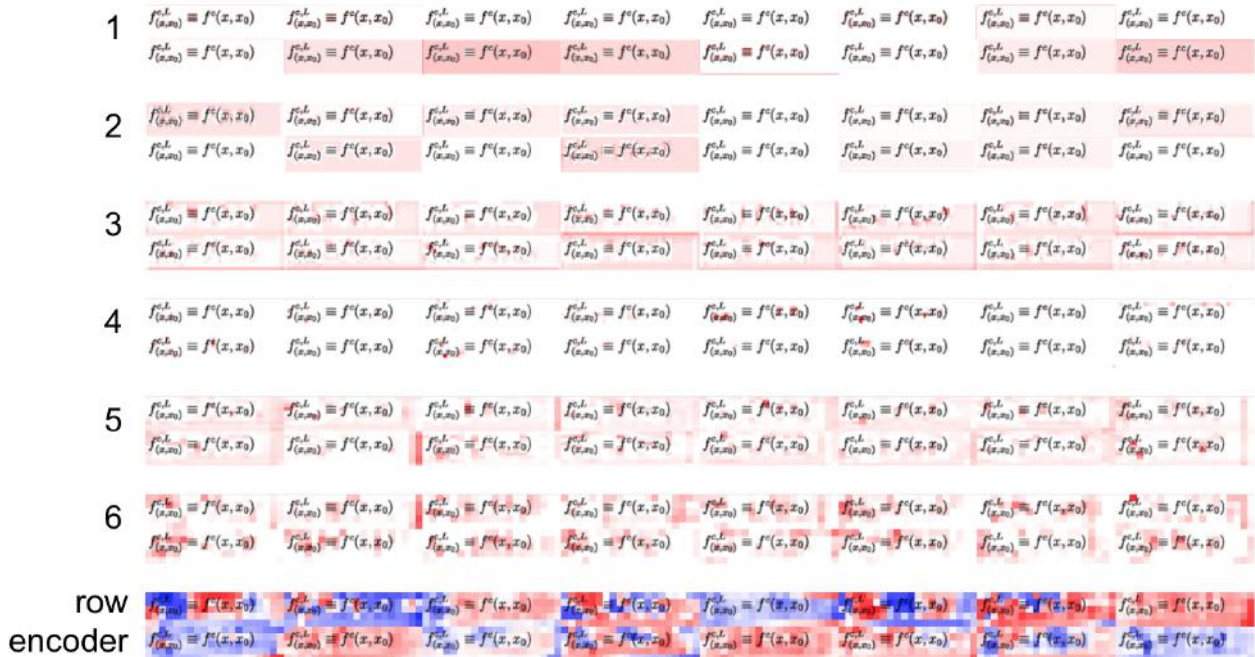


Figure 4.2: Activations of the first 16 filters in each of the convolutional layers and the LSTM row encoder. The red overlay denotes positive values and the blue overlay denotes negative values (color intensity is scaled up by approximately eight times for clarity). Note that there are no negative values in the convolutional layer activations because a ReLU activation function is used.

$$\begin{aligned}
 f_{(x,x_0)}^{c,L} &\equiv f^c(x, x_0) & f_{(x,x_0)}^{c,L} &\equiv f^c(x, x_0) \\
 f_{(x,x_0)}^{c,L} &\equiv f^c(x, x_0) & f_{(x,x_0)}^{c,L} &\equiv f^c(x, x_0)
 \end{aligned}$$

Figure 4.3: Selected filter activations from the sparse convolutional layer (layer 4). The red overlay indicates the activation area; as before, the color intensity is scaled up to facilitate viewing. The top left filter appears to activate on right parentheses; the top right filter appears to activate on left parentheses; the bottom left filter appears to activate on the character x ; the bottom right filter appears to activate on commas.

	Num.	Num.	Num.	Mean	Mean
Dataset	training	validation	test	loss	perplexity
im2latex-100k [9]	76,444	8,491	9,695	0.2082	1.2416
Our dataset	205,388	25,673	25,681	0.1437	1.1573

Table 4.2: Results from testing our extended dataset.

Chapter 5

Adversarial Examples for Image-to-LATEX

In this chapter, we show that the optimization-based approach can be used to find successful adversarial examples that use very small perturbations.

5.1 Non-targeted Adversarial Examples

Non-targeted adversarial examples are the easiest type of adversarial examples to find because the optimization function enforces a relatively smaller set of constraints. As a reminder, to craft a non-targeted adversarial example for a valid input sample x with ground truth label y , we find the necessary perturbation δ by optimizing:

$$\delta = \underset{\delta}{\operatorname{argmin}} \lambda \|\delta\|_p - J(F_\theta(x + \delta), y)$$

The resulting non-targeted adversarial example is given by $x^* = x + \delta$. As suggested by Liu et al. [12], adversarial examples with small perturbations can be successfully found even when omitting the regularization term ($\lambda = 0$); we follow this recommendation to produce a baseline. Later, we will investigate the effect of applying regularization.

We use the Adam optimizer [10] over 100 iterations with a constant learning rate of $\eta = 0.3$. Table 5.1 contains summary metrics for the effectiveness of non-targeted adversarial examples and Figure 5.1 shows two examples of non-targeted adversarial examples. Even without regularization, the average maximum perturbation magnitude across the non-targeted adversarial examples is 38.78 for images encoded in 8-bit space (i.e., pixel values fall in the range $[0, 255]$).

Input type	BLEU score	Exact match
Valid	0.8773	0.7746
Non-targeted adversarial examples	0.3807	0.0000

Table 5.1: Performance degradation caused by non-targeted adversarial examples.

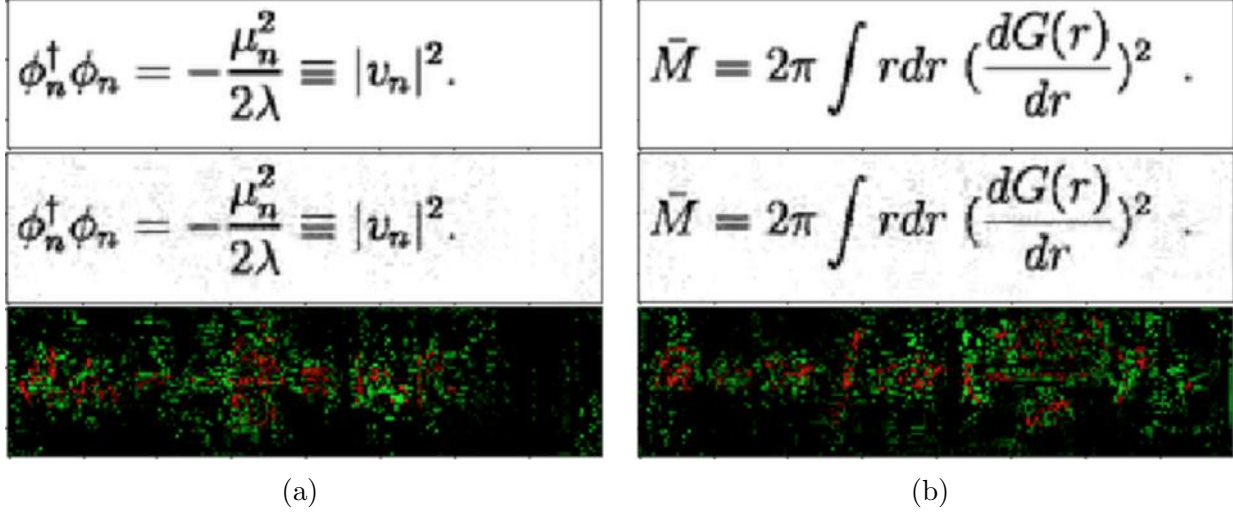


Figure 5.1: Two non-targeted adversarial examples. The top row is the original image, the middle row is the adversarial example, and the bottom row is the visualization of the perturbations (green denotes the addition of ink and red denotes the removal of ink; color intensities are scaled up to the maximum perturbation magnitude). In Figure 5.1a (left), the ground truth markup is $\phi_n^\dagger \phi_n = -\frac{\mu_n^2}{2\lambda} \equiv |v_n|^2$; its corresponding non-targeted adversarial example causes the model to instead output $z_n^z \phi_n = -\frac{\mu_a^2}{2\lambda} \equiv |\omega_a|^2$. In Figure 5.1b (right), the ground truth markup is $\bar{M} = 2\pi \int r dr (\frac{dG(r)}{dr})^2$; its corresponding non-targeted adversarial example causes the model to instead output $\bar{1} = 2\pi \int r dr (\frac{dG(r)}{dr})^2$.

5.2 Targeted Adversarial Examples

In addition to non-targeted adversarial examples, targeted adversarial examples can also be generated for image-to-L^AT_EX. Unlike image classification tasks (e.g., MNIST, ImageNet), an equation image is not defined by a single label; rather, each image sample is assigned a series of ordered labels corresponding to the token sequence of the reference markup. Hence, a targeted attack on this structured prediction task involves modifying either the classification or sequence order of one or more of the token labels.

As such, a targeted attack for image-to-L^AT_EX can be viewed as “editing” the ground truth label sequence. Therefore, we divide target attacks into three categories: substitution, insertion, and deletion.

We generate targeted adversarial examples using the optimization method:

$$\delta = \underset{\delta}{\operatorname{argmin}} \lambda \|\delta\|_p + J(F_\theta(x + \delta), y^*)$$

Just as with non-targeted adversarial examples, we initially omit the regularization term ($\lambda = 0$) [12]. For each of the three types of targeted attacks, we generate adversarial examples using the Adam optimizer [10] over 100 iterations with a constant learning rate of $\eta = 0.3$. We choose to keep these hyperparameters constant in order to compare the

difficulty of each type of attack; higher success rates could have been achieved by tailoring these hyperparameters to each attack type.

5.2.1 Substitution attack

A basic substitution attack involves replacing one character in the ground truth label with an arbitrary character of the attacker's choosing. Figure 5.2 shows an adversarial example that performs a substitution attack: the adversarial perturbations cause the model to misinterpret the `\alpha` token as a `\beta` token while leaving the predictions on all other tokens unchanged.

Substitution attacks are the easiest of the three targeted attacks. Using the hyperparameters described previously, we achieve an adversarial success rate of 72%.

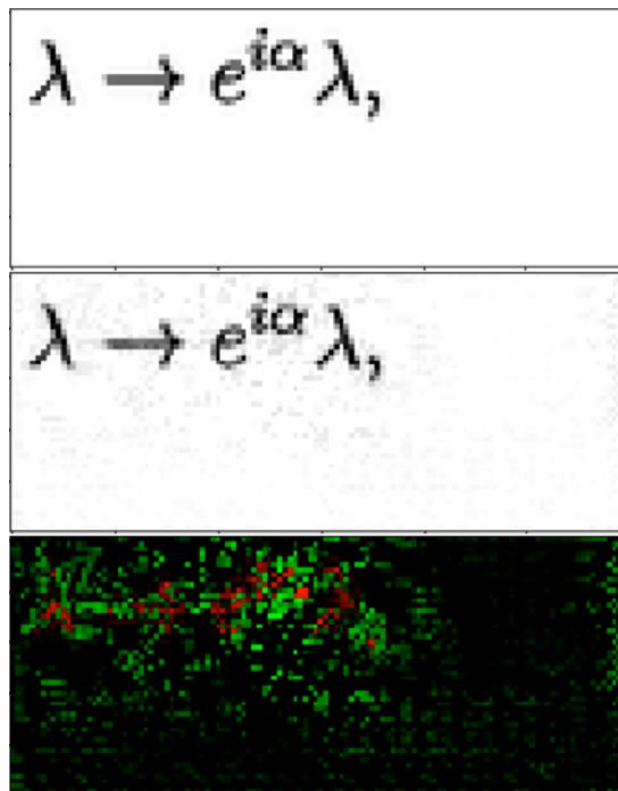


Figure 5.2: A targeted substitution attack. The top row is the original image, the middle row is the adversarial example, and the bottom row is the visualization of the perturbations (green denotes the addition of ink and red denotes the removal of ink; color intensities are scaled up to the maximum perturbation magnitude). The adversarial example successfully compels the model to erroneously substitute the `\alpha` token with a `\beta` token so that the final output prediction reads `\lambda \rightarrow e^{i\beta}\lambda ,.` Note that all other tokens are predicted correctly.

5.2.2 Insertion attack

While a substitution attack keeps the number of tokens across the adversarial label and the ground truth label constant, an insertion attack tricks the model into adding one or more tokens to the structured prediction output. Figure 5.3 gives an example of an insertion attack: the adversarial perturbations cause the model to output the digit 3 (three) immediately following the = (equal sign) token.

Using the set of common hyperparameters, we achieve an adversarial success rate of 54%. This suggests that insertion attacks are more difficult than substitution attacks, most likely because more precise perturbations are required to fool the model into detecting the presence of a non-existent glyph than to alter the identity of an existing glyph.

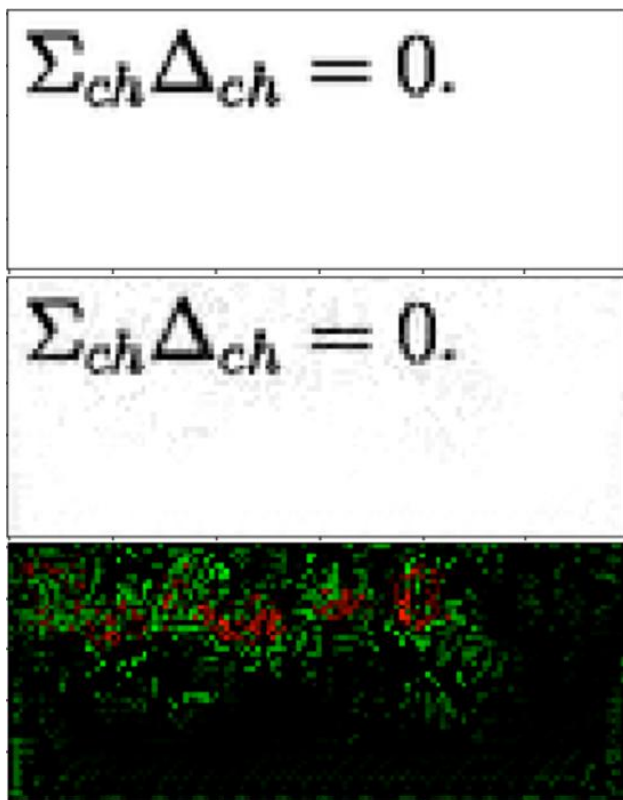


Figure 5.3: A targeted insertion attack. The top row is the original image, the middle row is the adversarial example, and the bottom row is the visualization of the perturbations (green denotes the addition of ink and red denotes the removal of ink; color intensities are scaled up to the maximum perturbation magnitude). The adversarial example successfully compels the model to erroneously insert a 3 (three) token between the = (equal sign) token and the 0 (zero) token so that the final output prediction reads $\Sigma_{ch} \Delta_{ch} = 30 \dots$

5.2.3 Deletion attack

A deletion attack allows an adversary to remove one or more tokens from the output prediction. Figure 5.4 shows an example of a deletion attack: the adversarial perturbations

prevent the model from recognizing the = (equal sign) token.

Using the set of common hyperparameters, our deletion attack achieves an adversarial success rate of 26%. This shows that deletion is the most difficult of the three targeted attack types. Masking the visual presence of an existing character in an equation image—especially given the amount of ink used to encode an equal sign—requires high-precision adversarial perturbations. We can achieve higher adversarial success rates for deletion attacks by increasing the number of iterations and adjusting the learning rate.

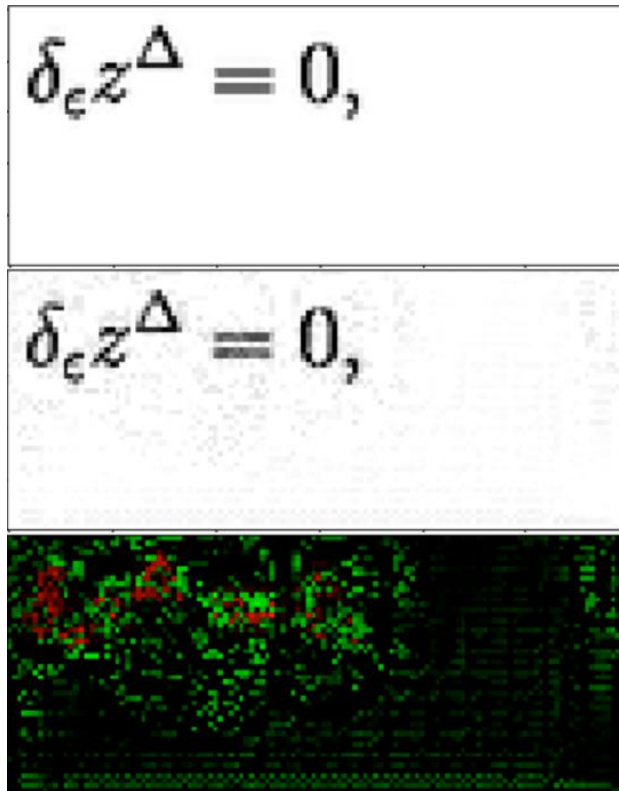


Figure 5.4: A targeted deletion attack. The top row is the original image, the middle row is the adversarial example, and the bottom row is the visualization of the perturbations (green denotes the addition of ink and red denotes the removal of ink; color intensities are scaled up to the maximum perturbation magnitude). The adversarial example successfully compels the model to erroneously omit the = (equal sign) token so that the final output prediction reads $\delta_{\epsilon} z^{\Delta} 0$.

5.3 Regularization

The adversarial examples created previously were optimized without a regularization term. Although they were successful, we observe a large number of perturbations that fall outside of the area of the token under attack (see the perturbation visualizations in Figures 5.2, 5.3, and 5.4).

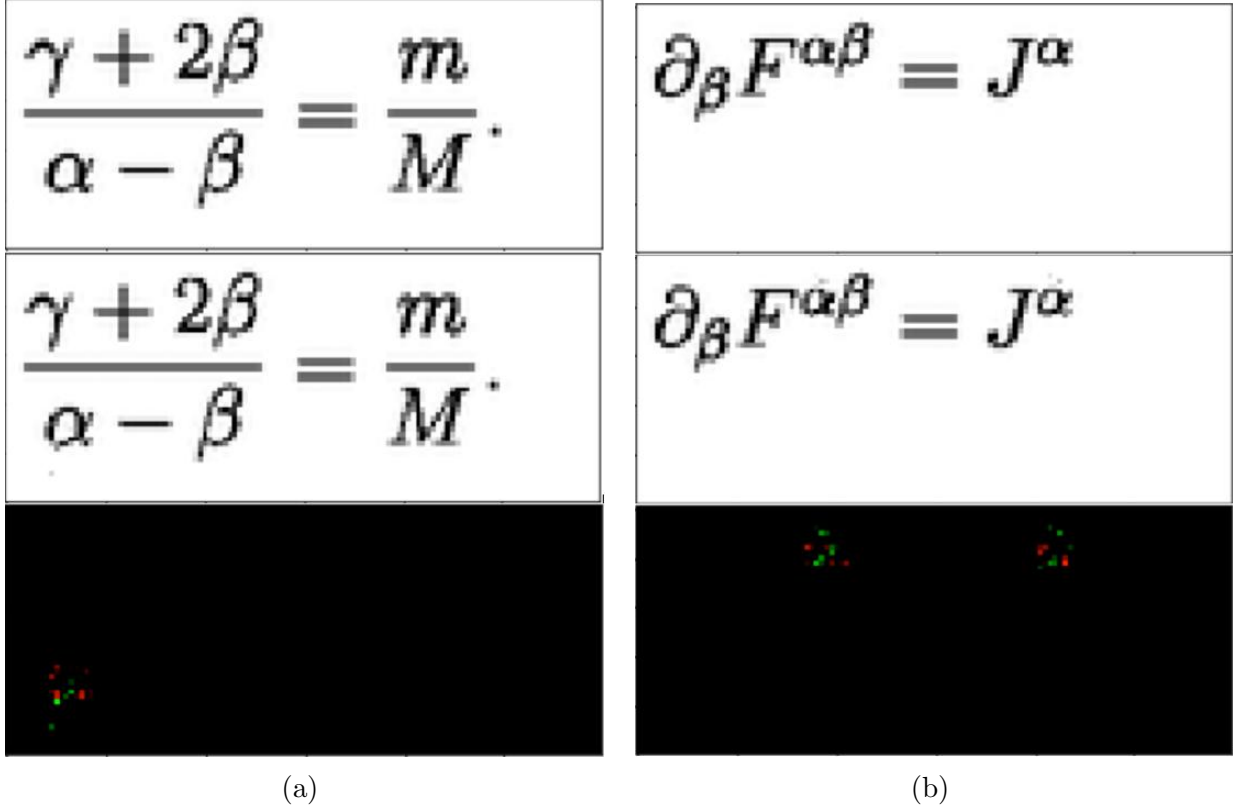


Figure 5.5: Two targeted adversarial examples for substitution attacks optimized with an ℓ_1 regularization term. The top row is the original image, the middle row is the adversarial example, and the bottom row is the visualization of the perturbations (green denotes the addition of ink and red denotes the removal of ink; color intensities are scaled up to the maximum perturbation magnitude). In each of the two examples, the model predicts each `\alpha` character as a `\beta` character while predicting every other token correctly. The perturbation visualizations (last row) show that the ℓ_1 regularization term effectively constrains the adversarial perturbations to the area around the target glyph. This is especially apparent in Figure 5.5b (right), where there are two `\alpha` characters present.

To minimize the magnitude of perturbations and the number of perturbed pixels, we introduce an ℓ_1 regularization term to our optimization function. Specifically, for a targeted attack, we find:

$$\delta = \underset{\delta}{\operatorname{argmin}} \lambda \|\delta\|_1 + J(F_\theta(x + \delta), y^*) \text{ with } \lambda > 0$$

For this experiment, we generate regularized substitution attacks. We use the Adam optimizer [10] over 500 iterations with a learning rate $\eta = 0.35$ and a regularization strength $\lambda = 0.0032$. Two examples are shown in Figure 5.5. Overall, this approach achieves an adversarial success rate of 86%. Comparable success rates can be achieved for insertion and deletion attacks by tuning the hyperparameters accordingly.

These regularized results show that the image-to-L^AT_EX model can be compromised by adversarial perturbations that are virtually imperceptible.

5.4 Activation Differences

Our analysis in Chapter 4 included visualizations of different convolutional layers in the WYGIWYS model. We drew particular attention to the fourth convolutional layer, which exhibited very sparse activations. As shown in Figure 4.3, we hypothesized that the filters learned in the fourth convolutional layer activate on different glyphs in the \LaTeX vocabulary.

To confirm this hypothesis, we compare the difference between the filter activations between valid inputs and adversarial images. As shown in Figure 5.6, filters sensitive to the character under attack activate and deactivate in line with the adversarial substitution.

$$\begin{aligned}
 f_{(\mathbf{x}, \mathbf{x}_0)}^{c,L} &\equiv f^c(\mathbf{x}, \mathbf{x}_0) \\
 f_{(\mathbf{x}, \mathbf{x}_0)}^{c,L} &\equiv f^c(\mathbf{y}, \mathbf{x}_0) \\
 f_{(\mathbf{x}, \mathbf{x}_0)}^{c,L} &\equiv f^c(\mathbf{x}, \mathbf{x}_0)
 \end{aligned}$$

Figure 5.6: Selected filter activations at the fourth convolutional layer on an adversarial example. The first image (top row) is the activation of a filter sensitive to the \mathbf{x} character on a valid input: $f_{\mathbf{x}}(I_{\text{valid}})$. The second image (middle row) shows the difference between the activation of $f_{\mathbf{x}}$ on an adversarial example and the valid image: $f_{\mathbf{x}}(I_{\text{adversarial}}) - f_{\mathbf{x}}(I_{\text{valid}})$. Here, the adversarial example substitutes the first \mathbf{x} character on the right-hand side with a \mathbf{y} character. We observe that the nodes of $f_{\mathbf{x}}$ around the target \mathbf{x} character deactivate (blue shading indicates a negative difference value). The third image (bottom row) is the activation of a filter sensitive to the \mathbf{y} character on the adversarial image: $f_{\mathbf{y}}(I_{\text{adversarial}})$. Note that only the targeted \mathbf{x} character activates, while unperturbed \mathbf{x} characters do not activate.

5.5 Robustness Test

We measure the resiliency of adversarial examples against random noise. That is, does a targeted adversarial example remain effective if perturbed by random Gaussian noise?

A targeted adversarial example perturbed by random noise can either (1) maintain its effectiveness as a targeted adversarial example, (2) return to its ground truth label, or (3) degrade into some non-targeted adversarial example. We experimentally evaluate each of these three probabilities by perturbing a varying fraction of pixels in a collection of targeted adversarial examples (substitution, insertion, and deletion) with random Gaussian noise over a range of variances.

Figure 5.7 shows that targeted adversarial examples remain effective when a large number of pixels are perturbed by noise with small variance, or when a small number of pixels are perturbed by noise with large variance. Unfortunately, just because targeted adversarial examples lose their effectiveness in the face of larger amounts of random Gaussian noise does not mean that these noisy adversarial examples revert back to their reference labels. Figure 5.8 indicates that adversarial examples that become ineffective because of random Gaussian noise overwhelmingly degrade to non-targeted adversarial examples.

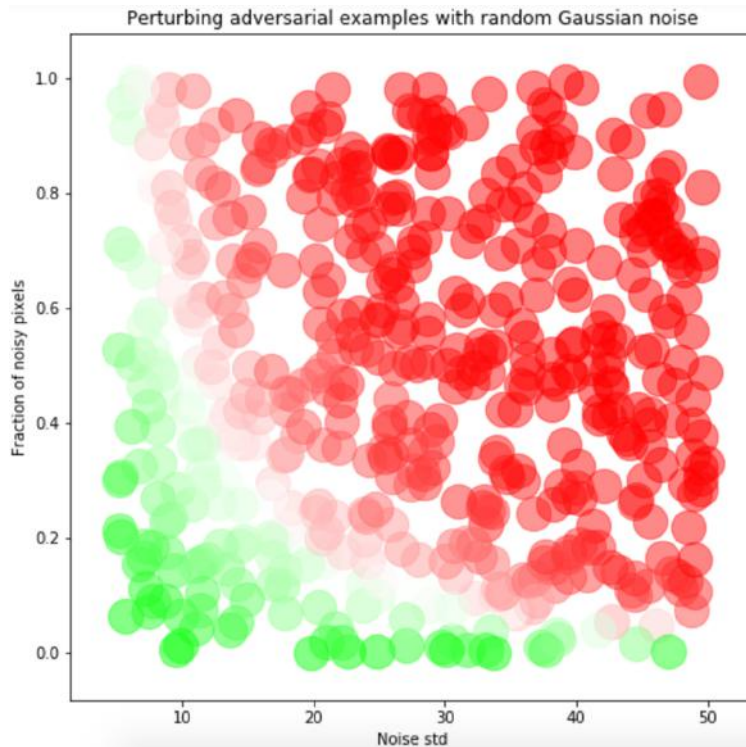


Figure 5.7: The likelihood of targeted adversarial examples remaining effective as a function of noise density (vertical axis) and noise intensity (horizontal axis). Green indicates that the adversarial example is likely to remain effective ($p > 0.5$), white indicates that the adversarial example is equally likely to maintain or lose effectiveness ($p = 0.5$), and red indicates that the adversarial example is likely to become ineffective ($p < 0.5$).

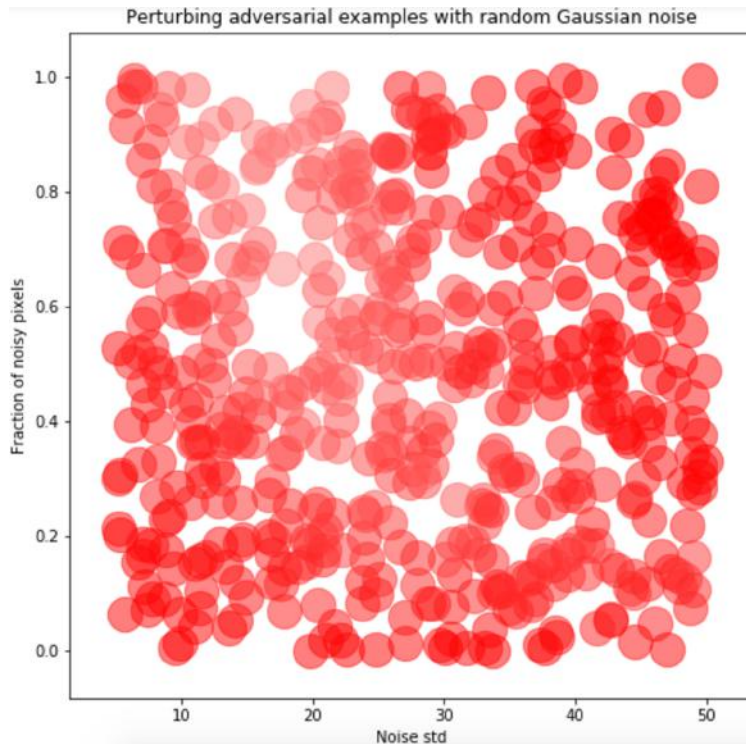


Figure 5.8: The likelihood of targeted adversarial examples reverting back to their ground truth labels as a function of noise density (vertical axis) and noise intensity (horizontal axis). Green indicates that the adversarial example is likely to revert back to its ground truth label ($p > 0.5$) and red indicates that the adversarial example is unlikely to revert back to its ground truth label ($p < 0.5$). The plot is entirely red because the probability that an adversarial example reverts back to its reference label is low across all measured data points (the average probability of reverting back to the ground truth label is 0.0845).

Chapter 6

Attack Detection

In this chapter, we evaluate two recently proposed methods for detecting adversarial examples and discuss their limitations with respect to adversarial examples for the image-to-L^AT_EX task.

6.1 PCA Comparison

Hendrycks and Gimpel argue that valid inputs and adversarial examples exist in different principal component spaces [7]. They perform PCA on the pixels of valid training set images and show that mapping adversarial examples to this space results in an abnormal emphasis on lower-rank principal components.

Figure 6.1 gives a visualization of the comparison proposed by Hendrycks and Gimpel for a single valid and adversarial image pair. The coefficients of lower-rank principal components for adversarial images (blue) is several orders of magnitude larger than their corresponding coefficients for valid images (orange). The PCA coefficients of twelve additional pairs of valid and adversarial images for the image-to-L^AT_EX task are compared in Figure 6.2.

At first glance, this method may seem like an effective way to distinguish between valid and adversarial images. However, we find that Hendrycks and Gimpel’s approach is brittle:

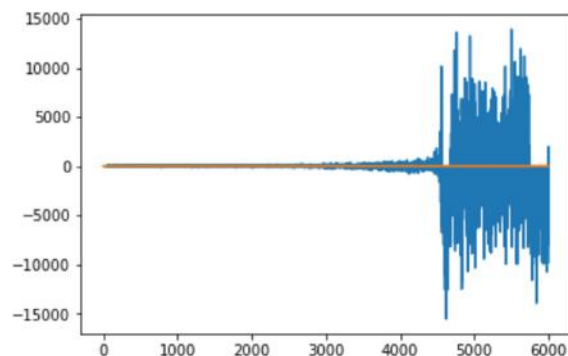


Figure 6.1: The PCA coefficient comparison between a single valid equation image (orange) and its corresponding adversarial example (blue). The principal components are arranged along the horizontal axis in order of rank; the vertical axis denotes the coefficient value.

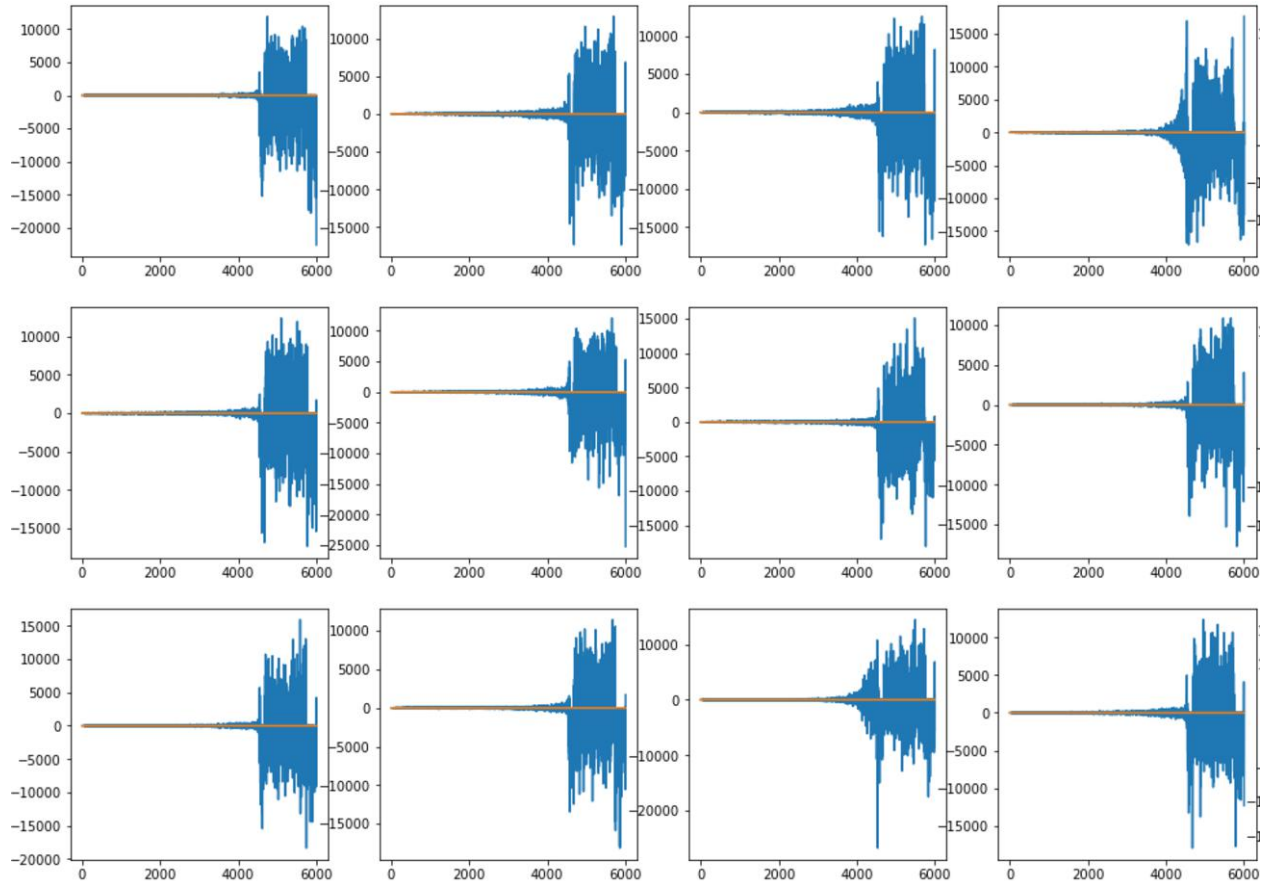


Figure 6.2: The PCA coefficient comparison between twelve randomly selected pairs of valid and adversarial images. The adversarial images were computed using the optimization-based approach with ℓ_1 regularization.

even the smallest amount of random Gaussian noise causes the coefficients for lower-rank principal components to explode. As an experiment, we take random images from the test set and perturb a 5% of the pixels in each image using random Gaussian noise with $\sigma = 5$. Recall that the images in the dataset are scaled between 0 and 255, so random Gaussian noise with $\sigma = 5$ is very small. The WYGIWYS model produces the correct markup for each of these noisy images, but the coefficients of the lower-rank principal components behave as if the noisy image were adversarial, as shown in Figure 6.3. A practical deployment of this detection method would yield a high false positive rate in the face of minuscule amounts of noise. These results highlight the importance of a detection strategy’s ability to distinguish between adversarial noise and random noise.

6.2 Out-of-distribution Softmaxes

In another work, Hendrycks and Gimpel suggest that incorrectly classified inputs—including adversarial examples—yield softmax outputs that fall in a different distribution than softmax outputs for correctly classified samples because correctly classified samples tend to have a

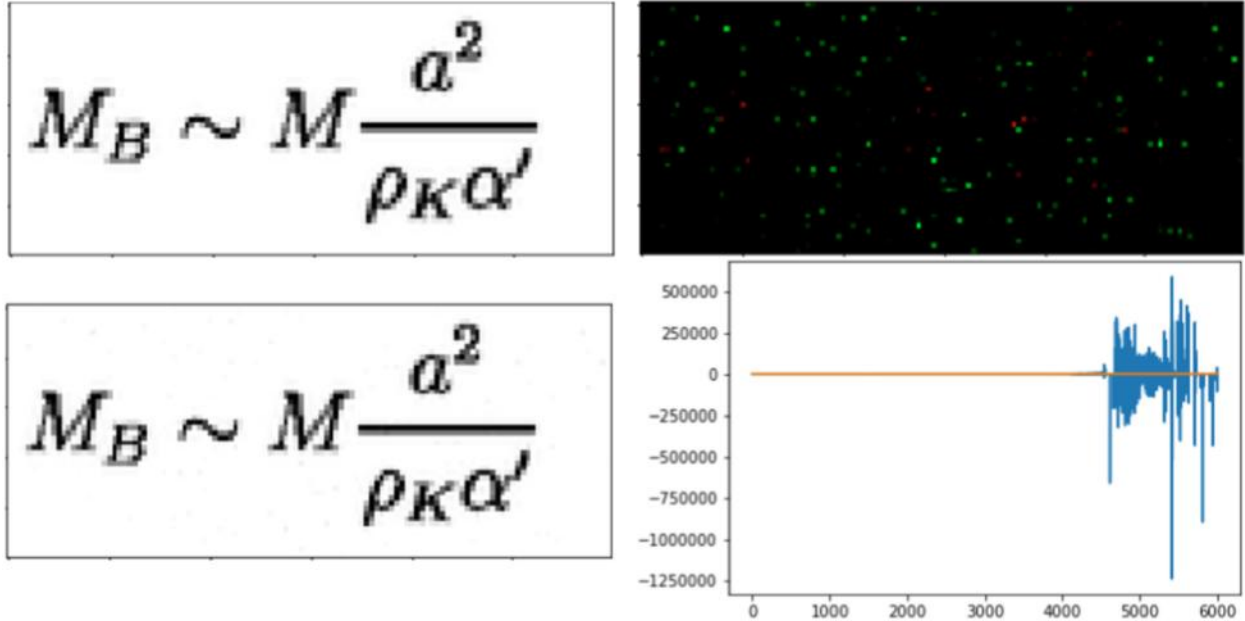


Figure 6.3: The PCA coefficient comparison between a clean image and a noisy image. (1) Top left: the clean image. (2) Bottom left: the noisy image; 5% of the pixels have been perturbed with random Gaussian noise ($\sigma = 5$). (3) Top right: a visualization of the added noise (green denotes the addition of ink and red denotes the removal of ink; color intensities are scaled up to the maximum perturbation magnitude). (4) Bottom right: the PCA coefficient comparison between the clean image and the noisy image.

higher maximum softmax value than incorrectly classified samples [6]. As such, a classifier for out-of-distribution samples can be built using observed softmax probabilities.

We apply this approach to the image-to-L^AT_EX model. Because we are making structured predictions, the WYGIWYS outputs a set of softmax probabilities for each output token. We divide the tokens into four sets: (1) all tokens across a collection of valid images, (2) all tokens across a symmetric collection of adversarial images, (3) the tokens in the valid image collection that are targeted by adversarial examples, (4) the tokens in the adversarial image collection that are maliciously modified. For clarity, set 3 is a subset of set 1; similarly, set 4 is a subset of set 2.

We note the model’s maximum softmax output for each of the tokens. Table 6.1 summarizes the maximum softmax probability distributions for each set. The distribution metrics shown in Table 6.1 indicate that there is a clear difference between the softmax probabilities produced by valid, pre-perturbation target characters and those produced by adversarial, post-perturbation target characters (set 3 versus set 4). However, if we consider the softmax probabilities of all characters across the valid and adversarial images (set 1 versus set 2), the distinction is much less clear.

Table 6.2 shows the results of building classifiers to detect adversarial tokens with respect to the output softmax values. If we assume that the defender knows exactly which token or tokens the attacker wants to manipulate, then this detection scheme works perfectly since the ranges of the softmax probabilities in set 1 and set 2 are disjoint (see row 1 in Table 6.2).

Token set	Num. samples	Mean	Standard deviation	Max.	Min.
(1) All, valid	761	0.9793	0.0647	1.0000	0.4096
(2) All, adversarial	761	0.9582	0.1122	1.0000	0.2452
(3) Target, valid	49	0.9892	0.0196	0.9997	0.9101
(4) Target, adversarial	49	0.6291	0.1776	0.8970	0.2452

Table 6.1: A summary of the maximum softmax probability distributions for the four token sets.

However, if the defender does not have prior knowledge about which tokens are under attack, Hendryck and Gimpel’s approach does not work well. In the case where we consider the softmax probabilities from all tokens in both valid and adversarial images (row 3 in Table 6.2), the AUPR and AUROC fall between 50% and 55%, which means that the detector is only performing slightly better than random. If we consider larger equation images containing a greater number of tokens, the performance will likely degrade further.

It is still possible that residual adversarial noise affects the distribution of softmax values in neighboring, non-targeted characters. If so, we can use this information to make stronger predictions about the validity of an equation image. To test this hypothesis, we compare corresponding non-targeted tokens from the collections of valid and adversarial images to see if there is a difference in their maximum softmax distributions that can be used to differentiate the two sources. The base rate for each of the entries in Table 6.3 is 50/50 since a token in the valid image that is not subject to attack should be identically decompiled in the adversarial image.

The results in Table 6.3 show that such a difference does not exist: the maximum softmax probability distribution for non-targeted characters in the valid image collection is indistinguishable from the maximum softmax probability distribution for non-targeted characters in the adversarial image collection. This means that without precise knowledge of the adversary’s intent, softmax probabilities cannot be used to deduce whether an image is valid or adversarial.

Moreover, given that an attack may only manipulate a small number of tokens across a small number of images, even if our detectors work with high AUROC in the 50/50 base rate case, the prior distribution of valid and adversarial images will heavily bias a practical detector to return a non-adversarial verdict.

In class	Out class	Base rate	Normality AUPR	Abnormality AUPR	AUROC
Target (3)	Target (4)	50/50	1.0000	1.0000	1.0000
All (1)	Target (4)	94/6	0.9989	0.7613	0.9834
All (1)	All (2)	50/50	0.5077	0.5522	0.5210

Table 6.2: Detecting out-of-distribution token predictions using softmax probabilities.

Non-targeted token	Num. samples	Normality AUPR	Abnormality AUPR	AUROC
= (equal sign)	46	0.4776	0.4740	0.4802
\frac	30	0.5348	0.4747	0.5022
^ (caret)	76	0.5017	0.4882	0.4761
_ (underscore)	90	0.4570	0.4608	0.4467

Table 6.3: Distinguishing non-targeted tokens in valid and adversarial images using softmax probabilities.

Chapter 7

Defenses and Their Limitations

In this chapter, we consider several defensive measures against adversarial examples for image-to-L^AT_EX and their respective limitations.

7.1 Types of Adversaries

We begin with a discussion of two threat models to consider when evaluating defenses.

First, we define a *weak adversary* to be an attacker that is not aware of any defenses that may be in place to protect the model from adversarial examples. A weak adversary can generate adversarial examples using known methods. The ability to successfully defend against a weak adversary is the minimum requirement for a defense.

Second, we define a *strong* or *adaptive adversary* to be an attacker that is aware of any and all defensive measures, understands exactly how the defenses work, and actively tries to circumvent them. Following a key tenet in computer security, an ideal defense should remain effective against an adaptive attacker.

For each of the following defenses, we show that it successfully defends against weak adversaries, but fails against adaptive ones.

7.2 Segmentation

In Chapter 5, we gave several examples of non-targeted and targeted adversarial examples. In Figures 5.1, 5.2, 5.3, and 5.4, a significant portion of the adversarial perturbations falls in the whitespace area surrounding the equation. Even in the regularized case (Figure 5.5), not all of the adversarial perturbations fall strictly over the inked areas of the glyph in question.

This motivates the use of image segmentation as a defense against adversarial examples. Given a mask over the equation image, we can nullify adversarial perturbations that fall outside of the inked glyphs. By doing so, we reduce the number of pixels an adversary can perturb.

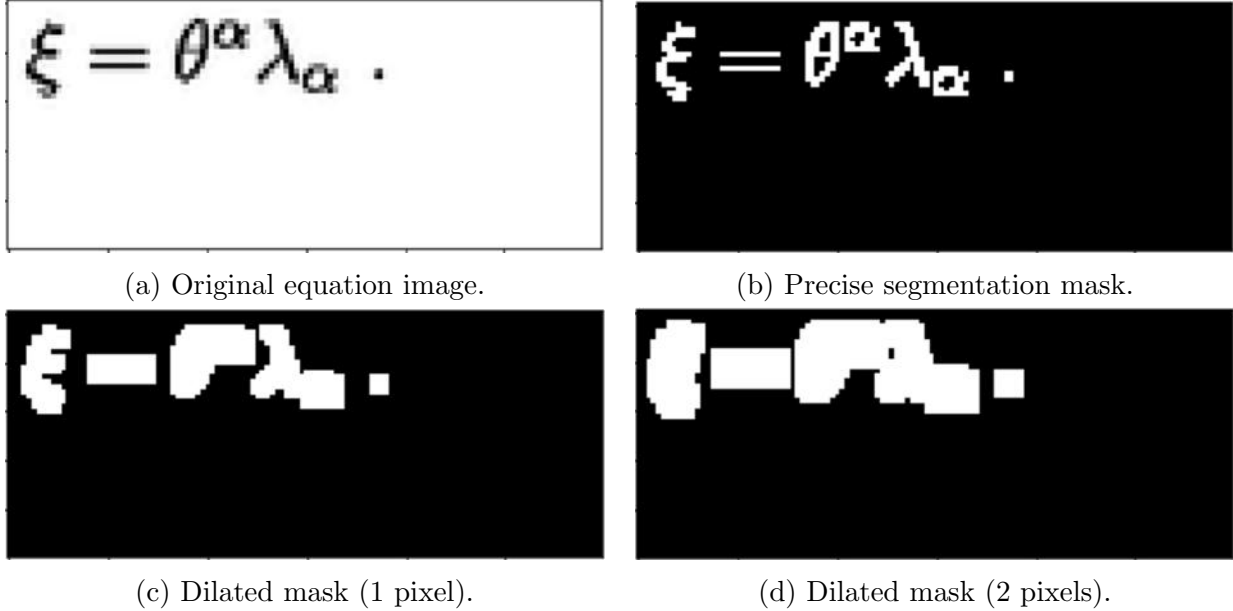


Figure 7.1: Segmentation masks with varying degrees of precision.

7.2.1 Computing masks

Naturally, this defense brings up questions about how masks can be obtained. Furthermore, if we use a machine learning model for image segmentation, can the attacker instead compromise the segmentation system?

Because we have access to the clean, pre-perturbed images in the test set, we devise a simple segmentation oracle for the sake of this evaluation. For any valid 8-bit encoded equation image I_{valid} , we can find a precise segmentation mask M_0 by calculating:

$$M_0 = I_{\text{valid}} < 200 \quad (0 \text{ is black and } 255 \text{ is white})$$

In addition to calculating a precise mask, we can construct masks of varying precision by thresholding the result of convolving the precise mask with a ones matrix. For instance, to compute a mask with one-pixel dilation, we calculate

$$M_1 = (M_0 * J_3) > 0$$

where J_3 is the 3×3 ones matrix. Similarly, to compute a mask with two-pixel dilation, we calculate:

$$M_2 = (M_0 * J_5) > 0$$

A visualization of the resulting segmentation masks is given in Figure 7.1.

7.2.2 Weak adversary

We first test this defense on a weak adversary that generates adversarial examples following the optimization-based approach used in Chapter 5. After the weak adversary submits a set

of adversarial examples, we use a precise mask to set all areas of the image that fall outside the inked equation to white (255).

Without the segmentation defense, the weak adversary achieves a 74% adversarial success rate. With the segmentation defense, the weak adversary achieves a 0% success rate; none of the segmented targeted adversarial examples match their intended adversarial labels. The defense is comparably effective using when using suboptimal masks with one- or two-pixel dilation.

7.2.3 Adaptive adversary

Next, we consider mounting this defense against an adaptive attacker. Because an adaptive attacker is aware of the segmentation defense, he or she modifies the generation procedure for adversarial examples to account for this change. Specifically, the adaptive attacker restricts the update step and the regularization computation to operate exclusively within the segmented area. To emulate an adaptive attacker, we use the Adam optimizer [10] over 500 iterations with a constant learning rate $\eta = 0.35$ and an ℓ_1 regularization strength $\lambda = 0.0032$.

Using the precise segmentation masks, our results show that predictions on the adversarial images exactly match the adversarial label 76% of the time across a batch of fifty adversarial examples. Moreover, if we only consider the subset of adversarial images whose valid counterparts yield predictions that exactly match their respective reference labels, then the adversarial success rate increases to 89.2%.

Figure 7.2 gives an example of an adversarial example produced by an adaptive adversary. The attacker successfully restricts all adversarial perturbations to the space within the segmentation mask. These results look similar to the adversarial examples generated with ℓ_1 regularization. However, they obey a stronger constraint here since not all of the perturbations in ℓ_1 -regularized adversarial examples fall precisely over the inked glyphs.

As expected, using masks with lower precision (e.g., one-pixel dilation or two-pixel dilation) simply serves to increase the attack surface for the adaptive adversary. For instance, in the one-pixel dilation case, the adaptive adversary achieves an adversarial success rate of 86%.

With our artificial segmentation oracle, we produce perfect (or near-perfect) segmentations of equation images; yet, an adaptive attacker can easily circumvent these masks. This means that in a practical deployment where we do not have access to such an oracle, the segmentation quality can only decrease. An imperfect segmentation system that is too liberal fails to mask out all whitespace outside of the inked equation, thereby giving an adaptive adversary access to more pixels to perturb. On the other hand, an imperfect segmentation system that is too conservative may mistakenly mask out parts of the inked equation, which would undermine the performance of the model on valid inputs.

7.3 Input Quantization

The adversarial examples we have produced thus far rely on perturbations with small floating point magnitudes. However, the WYGIWYS model is trained and evaluated on images

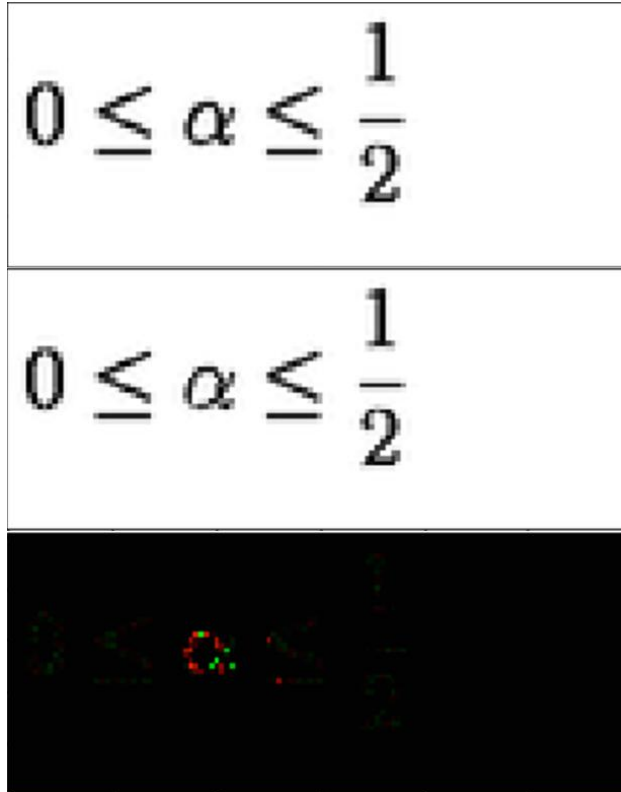


Figure 7.2: An adversarial example produced by an adaptive adversary that is aware of the segmentation defense. The top row is the original image, the middle row is the adversarial example, and the bottom row is the visualization of the perturbations (green denotes the addition of ink and red denotes the removal of ink; color intensities are scaled up to the maximum perturbation magnitude). The adversary successfully restricts all adversarial perturbations to the space within the precise segmentation mask.

that are quantized to 8-bit integer values. By allowing inputs to take on arbitrary floating point values, the model unnecessarily leaves itself vulnerable to the space of floating point adversarial inputs.

As such, a proposed defense involves limiting the precision of input values to the quantization space of the training set. For the image-to-L^AT_EX task, this means small perturbations that fall between the 8-bit quantization boundaries will be rounded accordingly, thereby reducing the precision and effectiveness of adversarial examples optimized in floating point space.

To make the defense more robust, we can take this one step further by reducing the precision of the training data itself. Rather than training and testing in an 8-bit image space, we binarize the image data (i.e., reduce to single-bit precision) using a simple thresholding scheme and retrain the network. The binarized model achieves comparable performance on the test set with a mean loss of 0.2302 and a mean perplexity of 1.2709; for reference, the model achieves a mean loss of 0.2082 and a mean perplexity of 1.2416 when trained and evaluated on 8-bit images. In fact, this behavior can be observed in other models for different tasks: models for MNIST and CIFAR-10 experience minimal performance losses

when trained and evaluated on low-precision images [23].

7.3.1 Weak adversary

Given this new binary quantization space, adversarial perturbations are forced to fully activate or deactivate pixel values in the image. Suppose the weak adversary generates adversarial examples with floating point values between 0 and 255 as before. Because the majority of the adversarial perturbations are not large enough to change their respective binary activation states, the binarized versions of these adversarial images are more or less identical to the binarized versions of their valid counterparts. Therefore, by performing binarization as a pre-processing step, the adversarial examples lose their effectiveness.

7.3.2 Adaptive adversary

Now, suppose we are faced with an adaptive adversary that is aware of the binary quantization scheme. Naturally, the attacker can add the binary quantization constraint to the adversarial example generation pipeline. However, the discrete nature of the desired perturbations presents several problems for optimizers that operate in continuous space (e.g., stochastic gradient descent, Adam).

To begin, gradients cannot backpropagate smoothly across the quantization layer. Moreover, if the magnitude of the accumulated gradients over a single training step are not large enough to flip the binary activation state at any given pixel, the optimizer receives no useful feedback because the binarization nullifies these undersized perturbations. That is, the optimizer cannot see how much progress it has made towards generating a binarized adversarial example, or if it is even making steps in the right direction. This makes convergence extremely difficult. Ultimately, the only part of the objective function that the optimizer can control is the regularization term, so applying an unmodified optimization procedure for binarized adversarial images trivially returns an empty perturbation mask ($\delta = 0$).

These observations are all empirically verified. We also find that the optimizer fails to produce any meaningful adversarial noise even with a large learning rate and no regularization term. Because current approaches for generating adversarial examples fail to adapt to quantized scenarios, we introduce a new method for generating quantized adversarial examples.

Categorical reparameterization with Gumbel-Softmax

Jang et al. and Maddison et al. demonstrate an approach for learning latent categorical variables using continuous optimizers [8, 15]. Rather than attempting to explicitly backpropagate gradients through a non-differentiable discrete-valued matrix, they propose a continuous relaxation of categorical variables by reparameterizing them as differentiable samples from a Gumbel-Softmax distribution.

To give a simple high-level example, suppose we want to optimize a 1×1 categorical variable X that takes on one of three values in the set $\{-1, 0, 1\}$. Conventional optimization approaches that involve backpropagating gradients to X fail because X is not differentiable.

However, following the ideas presented in [8, 15], we can solve this problem by instead reparameterizing X as $X' = [X'_{-1}, X'_0, X'_1]$, a 1×3 matrix that represents a probability distribution over the categorical values of X . Note that the values of X'_k are continuous. In the forward pass, we randomly sample a discrete value from the set $\{-1, 0, 1\}$ based on the values in X' ; in the backward pass, we backpropagate gradients to each X'_k . To help the model converge, we anneal the sampling temperature as the number of optimization steps grows large.

For a complete explanation of the reparameterization trick that includes more details about temperature annealing and a derivation of the Gumbel-Softmax distribution, please refer to [8] and [15].

Quantized adversarial examples for MNIST

Using Gumbel-Softmax reparameterization, we show that an adaptive attacker can successfully produce adversarial examples that resist the quantization defense. Here, we evaluate the Gumbel-Softmax attack by generating targeted adversarial examples for an MNIST model that only accepts binarized inputs.

In the binary quantization case, there are a total three possible perturbation values: -1 , 0 , and 1 . As such, we reparameterize the discrete perturbation mask δ as $\delta' = [\delta'_{-1}, \delta'_0, \delta'_1]$, which encodes three unnormalized probabilities at each pixel. In general, for an n -bit quantization scheme, the number of possible noise values is given by $2^{n+1} - 1$.

For the continuous case, we initialize the perturbation mask δ at 0 . For the reparameterized case, we instead encode the zero initialization by setting $\delta'_k = \epsilon, \forall k \neq 0$ and $\delta'_0 = 1 - (\epsilon \cdot (2^{n+1} - 2))$, where ϵ is some small value (we used 0.001) and n is the bit depth.

We use the Adam optimizer [10] over 750 iterations with a constant learning rate $\eta = 0.09$ and an ℓ_1 regularization weight $\lambda = 0.5$. We initialize the Gumbel-Softmax sampling temperature at $\tau = 1.0$ and exponentially anneal it at a rate of 0.003 until it reaches a minimum temperature of $\tau = 0.1$.

Across a batch of 10,000 targeted adversarial examples, the attacks match their target adversarial labels at a rate of 0.8560 , their reference labels at a rate of 0.1309 , and some non-targeted adversarial label at a rate of 0.0131 . For reference, the binary input MNIST model achieves a test set accuracy of 0.9898 . Figure 7.3 gives some examples resulting from the Gumbel-Softmax attack.

Adapting the Gumbel-Softmax attack to the image-to-L^AT_EX task remains a piece of future work.

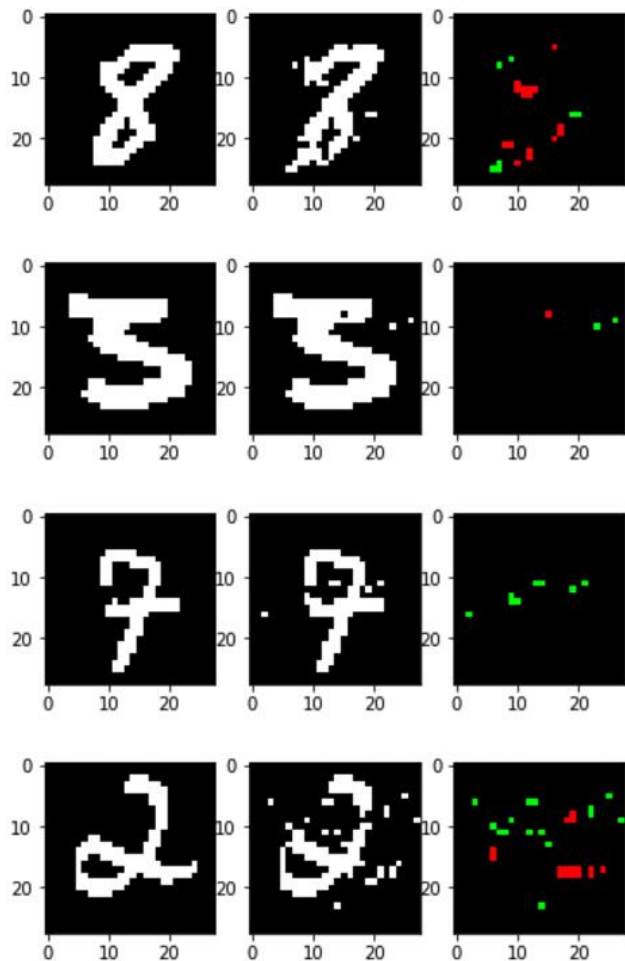


Figure 7.3: Binarized adversarial examples generated using the Gumbel-Softmax attack. The left column gives the original binarized images; the middle column gives the binarized adversarial examples; the right column gives a visualization of the adversarial perturbations (green denotes addition and red denotes subtraction). The reference labels for the four images are $[8, 3, 7, 2]$ and the adversarial target labels are $[7, 5, 9, 9]$. The adversarial examples exactly match their adversarial labels.

Chapter 8

Future Work

This work is framed primarily as a preliminary baseline evaluation of adversarial examples for structured prediction tasks. As such, we leave many avenues open for future work.

High-level categories for future work include:

- Generating adversarial examples for image-to- \LaTeX using other methods not considered by this work (e.g., JSMA [19], DeepFool [17])
- Evaluating the transferability of adversarial examples for image-to- \LaTeX
- Investigating adversarial examples in different types of visual decompilers (e.g., image-to-HTML), or in other structured prediction tasks
- Testing the robustness of visual decompilers on larger images (i.e., equations with multiple lines and/or more tokens)
- Adapting and evaluating upcoming and existing detection strategies and defenses on structured prediction tasks

As a piece of immediate follow-up work, we take advantage of the highly controlled nature of the image-to- \LaTeX task by integrating feedback from a \LaTeX compiler at test time. We hope that the process of rendering the output prediction and comparing it against the input image can help the model detect discrepancies that arise from either adversarial examples or non-adversarial misclassification.

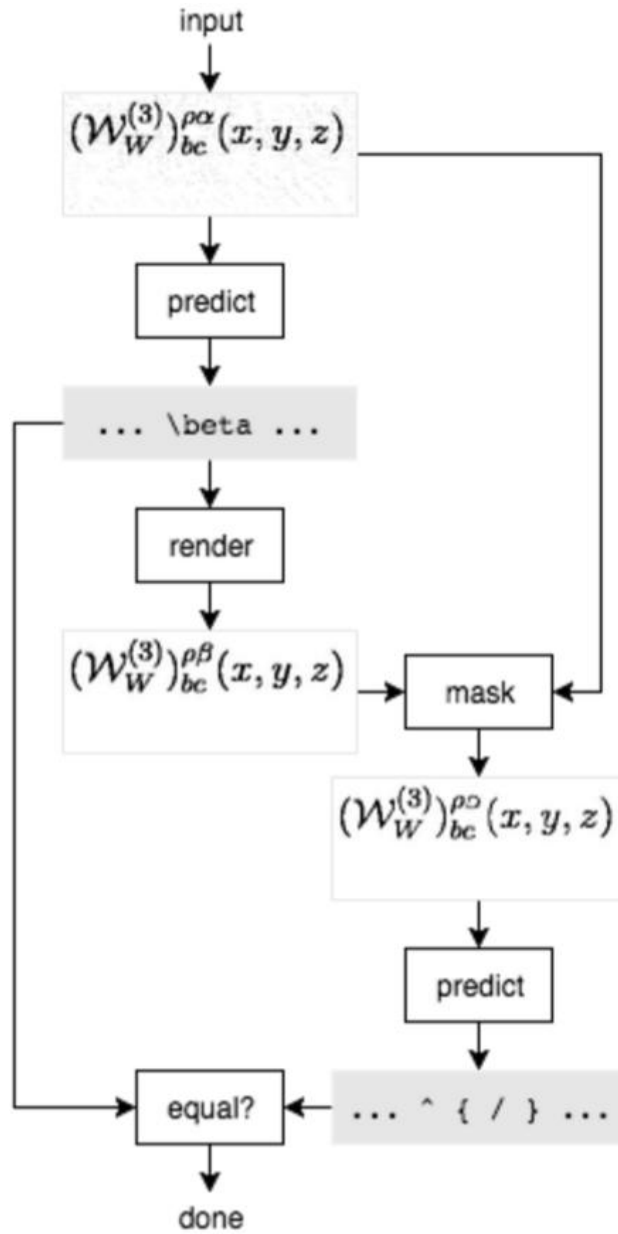


Figure 8.1: An example pipeline that uses the re-rendered output as additional feedback to validate the correctness of the prediction.

Chapter 9

Conclusion

We presented adversarial examples for visual decompilers. In our investigation of the image-to- \LaTeX structured prediction task, we adapt methods for generating adversarial examples and demonstrate that targeted and non-targeted adversarial examples can be produced using virtually imperceptible perturbations.

In an effort to mitigate the impact of adversarial examples, we evaluate two detection schemes. We show that PCA comparison is brittle in the face of random noise, and that the out-of-distribution softmax detection strategy fails without knowledge of the adversary’s intent.

Additionally, we propose and break two defenses. Image segmentation is an effective defense against weak adversaries, but fails against adaptive adversaries because it is possible to constrain the necessary adversarial perturbations within the masked equation area. Input quantization trades off a small degree of model performance to drastically shrink the space of adversarial examples. While previous methods for generating adversarial images cease to work in the quantized case, we successfully carry out a novel attack using Gumbel-Softmax reparameterization.

Finally, we outline some points of future work. Primarily, we plan to use a \LaTeX compiler at test time as a source of feedback about the quality of the output prediction; this will help detect misclassified samples (both valid and adversarial).

As deep learning makes its way into more and more mission-critical applications, understanding and combating adversarial examples become increasingly important. We hope that this work can inform future investigations into generating, detecting, and preventing adversarial examples for structured prediction tasks.

Bibliography

- [1] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint*, 2016.
- [2] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *arXiv preprint arXiv:1608.04644*, 2016.
- [3] Yuntian Deng, Anssi Kanervisto, and Alexander M Rush. What you get is what you see: A visual markup decompiler. *arXiv preprint arXiv:1609.04938*, 2016.
- [4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [5] Gösta H Granlund. Fourier preprocessing for hand print character recognition. *IEEE transactions on computers*, 100(2):195–201, 1972.
- [6] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.
- [7] Dan Hendrycks and Kevin Gimpel. Early methods for detecting adversarial images. *arXiv preprint arXiv:1608.00530*, 2016.
- [8] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *stat*, 1050:1, 2017.
- [9] Anssi Kanervisto. im2latex-100k, arxiv:1609.04938, June 2016.
- [10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [11] Rithesh Kumar. im2latex-tensorflow, 2017.
- [12] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.
- [13] Yi Lu. Machine printed character segmentation—; an overview. *Pattern recognition*, 28(1):67–80, 1995.
- [14] A Vander Lugt. Signal detection by complex spatial filtering. *IEEE Transactions on information theory*, 10(2):139–145, 1964.

- [15] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.
- [16] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.
- [17] 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.
- [18] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.
- [19] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.
- [20] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016.
- [21] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.
- [22] Tao Wang, David J Wu, Adam Coates, and Andrew Y Ng. End-to-end text recognition with convolutional neural networks. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3304–3308. IEEE, 2012.
- [23] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.