Attacking Robust Defenses Trained on MNIST and Fashion-MNIST with AutoAttack

Kevin McMahon
Department of Computer Science
Michigan State University
East Lansing, MI 48823
`mcmah111@msu.edu`

Tyler Lovell
Department of Computer Science
Michigan State University
East Lansing, MI 48823
`lovellty@msu.edu`

Abstract

In recent years, adversarial defenses have been advancing side by side with adversarial attacks. There are many existing defenses that have impressive bounds when attacked with PGD, but the more recent AutoAttack has proven many defenses to be lacking in terms of certified robustness. Recent discussion by top researchers suggest that MNIST is too easy to use as a baseline. We devised an experiment to see if using Fashion-MNIST will further decrease the certified robustness of chosen classifiers. To the best of our knowledge, this is the first work that empirically tests this swapping of datasets in the field of adversarial machine learning. Our results are strong and show that using a more complicated dataset will in fact make the trained models less robust against an identical attack.

1 Introduction

Machine learning is an ever growing field within computer science. Machines are being taught to make predictions on everyday things around them. They are given data to train off that then when given new data the trained model can make a decision as to what the new data is. However, there are ways to fool the model into thinking the data is something else. The data is slightly perturbed so as to fool the model into believing what it sees is actually something else. This is what adversarial machine learning is. Attacking a trained model into misclassifying the data. To prevent these attacks from working, special defenses and training methods are being developed. In this paper, we will be looking into the robust accuracy reported using a standardized attack system called AutoAttack [2] on four of these adversarial machine learning models that are trained on two different databases. The four models we used were TRADES, CROWN-IBP, MMA and FAST which we will describe later on.

We compared two different datasets. MNIST and Fashion MNIST. The MNIST database consists of labeled images of handwritten digits. MNIST is commonly used for training machine learning models. The database contains 60,000 training images and 10,000 testing images [6]. The Fashion-MNIST database was created to serve as a direct drop-in replacement for the MNIST dataset. Fashion-MNIST is a database containing article images from Zalando. The dataset uses fashion items such as shirts and shoes instead of written digits. Fashion-MNIST has the same split of 60,000 training images and 10,000 testing images as MNIST [11].

Many of the defenses evaluated in [2] featured both MNIST and CIFAR-10/100 classifiers. This range makes sense historically since MNIST has traditionally been the baseline for testing a classifier, while CIFAR acts as a more realistic evaluation both in terms of applications and resolution [5]. However, it has been known for quite some time that it is fairly straightforward to train a classifier with 99% accuracy on MNIST. Since Fashion-MNIST was created as a harder version of MNIST, our

Preprint. Under review.

$$\min_f \mathbb{E}\left\{ \underbrace{\phi(f(\boldsymbol{X})Y)}_{\text{for accuracy}} + \underbrace{\max_{\boldsymbol{X}'\in\mathbb{B}(\boldsymbol{X},\epsilon)} \phi(f(\boldsymbol{X})f(\boldsymbol{X}')/\lambda)}_{\text{regularization for robustness}} \right\}.$$

Figure 1: Proposed minimization function from TRADES paper [12]

project testsif training robust classifiers on Fashion-MNIST will result in worse performance than on MNIST.

2 Background

The field of adversarial machine learning has seen a recent explosion in research work. Almost every defense tested by the creators of AutoAttack was developed after 2017, and some of the defenses have been actively improved upon even after AutoAttack was released.

2.1 PyTorch

PyTorch is an open source machine learning framework released in 2016 [9]. Along with TensorFlow and Keras, it has enabled the rapid acceleration of machine learning and specifically deep learning. One of the key components of PyTorch is CUDA compatibility which allows for code acceleration on NVIDIA GPUs [8]. CUDA specifically is responsible for moving complex calculations off of CPU and onto much more powerful GPU architectures. We use Pytorch with CUDA enabled GPUs to train and test defenses in this project.

2.2 AutoAttack

Within adversarial machine learning there are many different defenses to test the robustness of a model. The problem with having so many different defenses is that there is no standard for how well one defense performs compared to another. AutoAttack looks to rectify this problem. AutoAttack is an ensemble of Diverse Parameter-free Attacks developed by Croce and Hein to standardize how adversarial defenses are tested. With every defense using the same testing method, the robustness's of the defenses can be better compared to each other. AutoAttack consists of four different attack methods to be run on the trained model to get a standardized robust accuracy. These attacks are two different versions of automatic projected gradient descent with different loss functions being used, an FAB attack and a square attack. [2]

2.3 TRADES

This defense spawns from an identified trade of between robustness and accuracy. The authors perform a theoretical breakdown of this trade-off and decompose "the robust error as the sum of the natural error and boundary error" [12]. The theoretical build up suggests Figure 1 be used as the new function for optimization. This equation encapsulates the trade-off between natural and robust errors. The model is made more accurate while also promoting smoothness, which is "an indispensable property of robust models" [1].

2.4 CROWN-IBP

Interval Bound Propagation is an efficient training method that for the most part works quite well [4]. Prior work shows that IBP outperforms relaxation based methods for robustness guarantees yet suffers from crucial stability issues. To attempt to solve this issue, IBP was combined with CROWN [13] to create the aptly named defense CROWN-IBP. This new defense uses two bounding passes: a forward bounding pass using the highly efficient IBP followed by a backward bounding pass with a "tight relaxation based bound" [13]. The authors of this work prove that CROWN-IBP is a steadfast improvement on IBP in terms of robustness guarantees. According to the creators of AutoAttack [2], CROWN-IBP was the most robust MINST defense they tested.

2.5 MMA

Max-Margin Adversarial training is a neural network training method to protect against adversarial attacks. Instead of using a fixed epsilon, MMA uses an adaptive selection to get the "correct" epsilon as the margin individually for each data point. Here, margins are defined as the distances from inputs to a classifier's decision boundary. The authors [3] found that there is a close connection between adversarial losses and the margins for a classifier where when the adversarial loss on the decision boundary for the shortest successful perturbation is minimized, the margins can be maximized.

2.6 FAST Adversarial Training

This adversarial training method prioritizes speed. The authors proved that is is possible to train an empirically robust model by using a weaker and cheaper adversary even though this method was previously believed to be ineffective. By using the fast gradient sign method along with random initialization values, an adversarial trained model can be as robustly effective as if it were trained on projected gradient descent based adversaries. The benefit to this is the drastic decrease in the time it would take to train the models. [10]

3 Experimental Setup

To test our hypothesis we evaluated 4 adversarial defense methods from [2] to train on MNIST and FashionMNIST. These defenses were chosen for their familiarity from CSE 891 lectures and their open availability on GitHub. One notable exclusion from the defenses used was IBP [4]. We reasoned that this defense could be excluded since CROWN-IBP [13] was done by a similar group of researchers that built on IBP and proved that adding the bounding used in CROWN [13] produced better robustness results.

We used the default model training and testing parameters supplied by the authors of each defense in all cases as it would be more reproducible in future experiments. Every model trained on MNIST in [2] tested using the L infinity norm with a radius of 0.3, so we elected to use this value for training and testing our models on both datasets. The MNIST and FashionMNIST datasets were obtained using PyTorch [9] datasets module.

3.1 Azure Environment

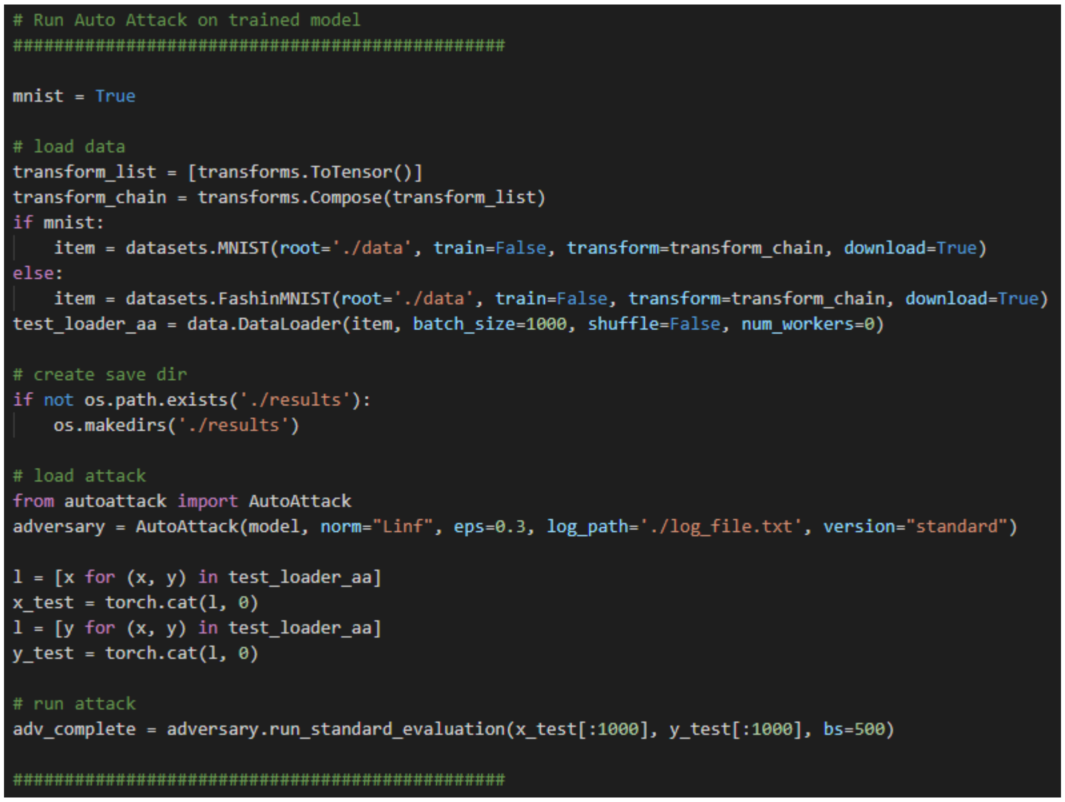
We used two different environments to test two defenses a piece. The Azure environment was created using a free Azure account with $200 of credits. Since it was our first time using this service, the straightforward Azure ML Studio was used to create experiments, runs, and models for this project. The URL for this service is `https://ml.azure.com/`.

The reason to use cloud computing is cheap access to computing resources that exceed the power of local machines. Computing tasks run on Azure were done using a STANDARD_DS3_V2 compute instance (CPU) and one NVIDIA Tesla V100 to run CUDA [8] enabled code. At around $0.80 per hour of usage, this environment was much preferred to purchasing an expensive GPU and setting up a local computing rig.

A list of key dependencies used to train the TRADES and CROWN-IBP models in Azure is listed below. Several Azure specific modules are left out as they are unnecessary when training these models outside of Azure. Note that this list may not be comprehensive for all environments, and some packages may not be required if not using PyTorch with CUDA enabled.

> "python=3.6.2", "pip=20.2.4", "cmake==3.18.2", "torch==1.6.0", "torchvision==0.5.0", "mkl==2018.0.3", "horovod==0.20.0", "tensorboard==1.14.0", "future==0.17.1", "joblib"

Naturally, several aspects of the code had to be modified in order to function correctly in Azure. Changes made for this reason are left out of this discussion for clarity with two exceptions. First,



```
# Run Auto Attack on trained model
#################################################

mnist = True

# load data
transform_list = [transforms.ToTensor()]
transform_chain = transforms.Compose(transform_list)
if mnist:
    item = datasets.MNIST(root='./data', train=False, transform=transform_chain, download=True)
else:
    item = datasets.FashinMNIST(root='./data', train=False, transform=transform_chain, download=True)
test_loader_aa = data.DataLoader(item, batch_size=1000, shuffle=False, num_workers=0)

# create save dir
if not os.path.exists('./results'):
    os.makedirs('./results')

# load attack
from autoattack import AutoAttack
adversary = AutoAttack(model, norm="Linf", eps=0.3, log_path='./log_file.txt', version="standard")

l = [x for (x, y) in test_loader_aa]
x_test = torch.cat(l, 0)
l = [y for (x, y) in test_loader_aa]
y_test = torch.cat(l, 0)

# run attack
adv_complete = adversary.run_standard_evaluation(x_test[:1000], y_test[:1000], bs=500)

#################################################
```

Figure 2: Code snippet used to attack a trained robust model using AutoAttack [2]

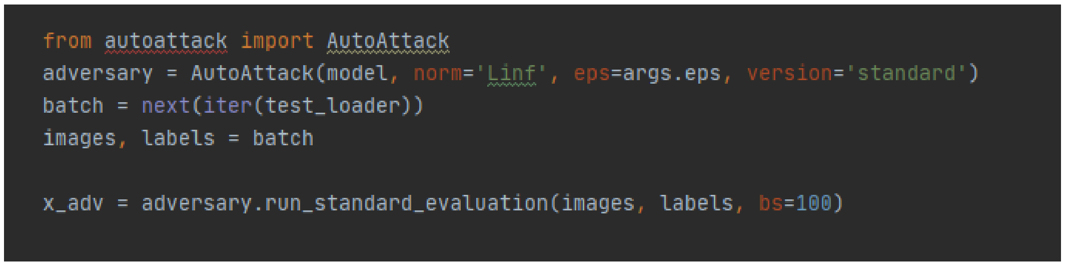
Figure 3 shows the code snippet that was added to each training file for TRADES and CROWN-IBP. This snipped was obtained from an example of AutoAttack found in file `eval.py` at `https://github.com/fra31/auto-attack/tree/master/autoattack/examples`. We relied on text logging to output the results of our experiments. Additionally, a small code snippet was added to relevant training files due to a HTTP 503 error when downloading MNIST via PyTorch.

TRADES The code used was obtained from `https://github.com/yaodongyu/TRADES`. The provided code trained a CNN with 4 convolutional layers and three fully-connected layers [12]. For each dataset, one state-of-the-art model was trained and subsequently tested. Very few changes to the code were necessary. A Python file was created in Azure that ran the file `train_trades_mnist.py`.

CROWN-IBP The code used was obtained from `https://github.com/huanzhang12/CROWN-IBP`. The authors of [13] claimed their reasoning behind training 10 small models to test would discourage hand tuning of hyperparameters and give a volume based evaluation of how effective CROWN-IBP is beyond the reported SOTA. We decided to use that same logic and train 10 small models to test for each dueling dataset to obtain an unbiased comparison in a reasonable amount of time.

3.2 Local Environment

The other setup we used was on one of our own machines. The machine has a NVIDIA GeForce RTX 2080 Ti GPU with 8 cores so running CUDA on it went relatively smooth. There were some difficulties getting the virtual environment initially set up and making sure it had all of the proper packages installed, but afterwards, the environment worked perfectly with no troubles in the environment itself. The only real difficulty in set up came from the lack of using virtual machines before. Tyler had never had a need to use them as everything he has done so far could be run within the IDE itself such as Jupyter Notebook or PyCharm. It was a good learning experience and something



```
from autoattack import AutoAttack
adversary = AutoAttack(model, norm='Linf', eps=args.eps, version='standard')
batch = next(iter(test_loader))
images, labels = batch

x_adv = adversary.run_standard_evaluation(images, labels, bs=100)
```

Figure 3: Code snippet used to attack the MMA and FAST trained robust models using AutoAttack [2]

Table 1: Accuracy of models trained with TRADES on both MNIST and Fashion MNIST datasets. The reported accuracy values were obtained from Table 5 of [12] using PGD [7]

TRADES Experiment	MNIST	Fashion-MNIST
Reported Clean Accuracy	99.48	-
Reported Robust Accuracy	95.60	-
Experimental Clean Accuracy	99.51	84.25
Experimental Robust Accuracy	91.10	**34.10**

that will be used again in the future. The environment had the following packages to run both MMA and FAST adversarial training:

> "python=3.9.4", "pip=20.2.3", "advertorch=0.2.3", "autoattack=0.1", "numpy=1.20.2", "pillow=8.2.0", "torch=1.8.1+cu102", torchaudio=0.8.1", "torchvision=0.9.1+cu102"

Unlike the Azure environment, much of the original code did not need to be changed for the models to be created. Both models had their own evaluation methods. These were changed to also include AutoAttack [2]. Unlike the code snippet above, all that was used for AutoAttack to evaluate the MMA and Fast models was from AutoAttacks [2] read me on how to use it. Below is what was used for both models.

MMA The code used was obtained from `https://github.com/BorealisAI/mma_training`. This code trained a LeNet5Madry neural network model from the advertorch package using the MMA training algorithm [3]. The only changes needed to be made to the training code was to allow the inclusion of the FashionMNIST database. To allow this, more options were added so that when the dataset specified in the arguments to begin training the model was given, it could also include FashionMNIST.

A separate file was used to evaluate the MMA trained model. The original evaluation was used and this file was modified to also include AutoAttack [2] to evaluate the model. Figure 3 contains the added lines to perform AutoAttack.

FAST Adversarial Training The code used was obtained from `https://github.com/locuslab/fast_adversarial`. Similar to the MMA model, the adversarial training for FAST also created a neural network. The Fast gradient sign method they mentioned was used to quickly train the model [10]. For both the MNIST and FashionMNIST datasets, the model was created without any setbacks. However, when evaluating the models, only results for MNIST were able to be attained. The fashionMNIST model when tested with the FAST repositories default evaluator and with AUtoAttack returned a robust accuracy of 0% percent. Thinking it was potentially the settings for the MNIST model that was originally used, we changed how the neural network was being initiated and even incorporated a new class function for it but none of these methods changed the results. When being evaluated for robust accuracy, the fashion MNIST models would return 0%.

Table 2: Robust accuracy of MMA and FAST trained models on both MNIST and Fashion MNIST datasets using the included evaluations and AutoAttack for both.

Evaluation Method	MNIST	Fashion-MNIST
MMA Included Evaluation	98.77	77.02
MMA AutoAttack	100	73.00
FAST Included Evaluation	85.20	0.00
FAST AutoAttack	76.00	0.00

4 Results

As we were not able to gather any worthwhile results for the FAST model on the Fashion MNIST dataset, we will skip that. Other than that though we got some interesting results for these two models. Let's look at the relationship between the models included robust accuracy evaluaters and with AutoAttack. The purpose of AutoAttack [2] again was to create a standardized adversarial defense tester so as to better compare one defense towards another. Here, we can see that the MMA included evaluation method actually calculated a slightly worse robustness score than AutoAttack did. However, when looking at the FAST model, the robustness score for the included evaluation script was much better than AutoAttacks grade on the model. Because of AutoAttack, we are better able to see how well the models perform when compared to each other. While the FAST model still had a lower robust accuracy than MMA with the included evaluater, we can see that when using a standardized method, it perform even worse when compared. At the same time though, the accuracy reported by the FAST model is still tremendous to see. The FAST model would take around 6 minutes to train where as the MMA model would take over 4 hours to train. Even with that little of a time for trianing, the accuracy calculated with AutoAttack is still very impressive.

We had predicted that there would be a difference between the accuracies of models trained using the MNIST dataset and those trained on the Fashion MNIST dataset, but we were not sure how much this difference would be. With the included evaluation script for the MMA model, we saw that the robust accuracy went down by about 23%. For the AutoAttack accuracy score, we observed a drop of 27% between the datasets. This difference does make sense when you look at the datasets themselves. While there are many different ways to write a number, the amount of ways that different fashion pieces can be constructed and designed is much more. ALong with that, the similarity between different types of clothing can be much closer than that of different numbers. It is hard at times to distinguish between a t-shirt and a shirt for humans, so when it is being tested on a machine that has been perturbed by an adversarial attack, it will have more of a challenge to distinguish the two.

Additionally, the results on TRADES show a similar trend. The results of our experiments on TRADES are shown in Figure 1. Note that the experimental robust accuracy is 91.10% on MNIST but only 34.10% on Fashion-MNIST! One could easily note that improving the defense or making the underlying neural network more complex would shrink this margin. That is on of the goals of the creators of Fashion-MNIST, and our results show that this applies to adversarial machine learning as well.

5 Conclusions

We consider our experiments successful on multiple levels. The original hypothesis was addressed using several freely available defenses and shows that the dataset chosen may make a bigger difference than initially thought. Additionally, we were able to apply material from lectures to a novel problem statement. These results provide yet another reason to move beyond MNIST for training proof-of-concept classifiers. The existing literature on Fashion-MNIST combined with our results demonstrates that the field of machine learning will eventually move beyond MNIST. Future adversarial defenses should consider adopting different datasets to see how much the choice of dataset affects the robust accuracy.

References

[1] Moustapha Cisse et al. *Parseval Networks: Improving Robustness to Adversarial Examples*. 2017. arXiv: `1704.08847 [stat.ML]`.

[2] Francesco Croce and Matthias Hein. *Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks*. 2020. arXiv: `2003.01690 [cs.LG]`.

[3] Gavin Weiguang Ding et al. "MMA Training: Direct Input Space Margin Maximization through Adversarial Training". In: *International Conference on Learning Representations*. 2020. URL: `https://openreview.net/forum?id=HkeryxBtPB`.

[4] Sven Gowal et al. *On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models*. 2019. arXiv: `1810.12715 [cs.LG]`.

[5] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (2009). URL: `http://www.cs.toronto.edu/~kriz/cifar.html`.

[6] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: `10.1109/5.726791`.

[7] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: `1706.06083 [stat.ML]`.

[8] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: `https://developer.nvidia.com/cuda-toolkit`.

[9] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[10] Eric Wong, Leslie Rice, and J. Zico Kolter. *Fast is better than free: Revisiting adversarial training*. 2020. arXiv: `2001.03994 [cs.LG]`.

[11] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: `cs.LG/1708.07747 [cs.LG]`.

[12] Hongyang Zhang et al. *Theoretically Principled Trade-off between Robustness and Accuracy*. 2019. arXiv: `1901.08573 [cs.LG]`.

[13] Huan Zhang et al. *Towards Stable and Efficient Training of Verifiably Robust Neural Networks*. 2019. arXiv: `1906.06316 [cs.LG]`.