Detecting Adversarial Examples with Anomaly Detector Enhanced GANs (GAN-ADE)

Michael Salceda
School of Engineering and Applied Sciences
The George Washington University
Washington, D.C., USA
msalceda@gwu.edu

***Abstract*—The increase in data availability and need for quick analysis has accelerated the push for machine learning solutions. With the heightened adoption of ML applications, securing and protecting those applications has been increasingly prioritized. To aid in those efforts, this paper attempts to construct a model- and domain-agnostic defense amid the sea of image and neural network-based attacks and defenses in the adversarial ML field. Using two different attacks to generate effective adversarial examples for testing, I achieve some promising results utilizing a novel architecture called GAN-ADE for detecting those adversarial examples, hopefully paving the way for some new ideas to expand this area of research.**

Keywords—anomaly detection, adversarial examples, generative adversarial networks, variational autoencoders

I. Introduction

In today's world, there is no shortage of data being generated and collected. Due to the increase in data volume and complexity, efforts to analyze that data are relying more heavily on machine learning (ML) to get to actionable insights quickly. However, people are becoming more privacy- and security-conscious, causing both public and private sectors to become more cognizant or the inputs and outputs of their ML applications. Because of this attention, research into adversarial ML is becoming more and more important.

A key issue in adversarial ML is defending against attacks where the attacker can generate adversarial examples for a model to make the model perform in an unexpected way, e.g., misclassifying an input with high confidence [13]. The best defense against these kinds of attacks ("evasion" attacks) is preventing the adversarial data from reaching the target model in the first place so, in this paper, I explore and test a way to detect and filter out adversarial examples with the additional challenges of: (1) only using the input training data and (2) not being able to access and/or modify the target model.

II. Background

The motivations behind this paper and the challenges laid out are based on a preliminary search of adversarial ML papers. Much of the literature and research in adversarial ML is focused on adversarial attacks and defenses for deep neural networks – particularly in the image domain. Consequently, the attacks and defenses proposed are often about image manipulation or other image-related perturbations. However, many applications in industry do not involve images so the transferability of defenses across domains can be a challenge. What may be an effective defense of an image-classification model may not be an effective defense for a tabular-based classification model and vice-versa. So, this begs the question: can a model- and domain-agnostic adversarial defense be created?

To limit the scope of this paper, I focused specifically on evasion attacks – attacks to make a target model behave incorrectly. Effectively crafting a defense for this kind of attack necessitates the restrictions set by the challenges mentioned in the introduction.

A. *Challenge 1: Using Input Training Data Only*

The use of only input training data to craft the defense is motivated by the fact that a defender may not even realize an attack has happened before adversarial examples are sent to the model. Or perhaps the defender is brought on after an attack, so the adversarial data used is no longer available. Whatever the case may be, for the sake of this scenario, I assume that, as a defender, I only have access to the input training data. This allows for a potentially more robust defense as it makes no assumptions on the evasion method or adversarial data – I just know that there may or may not be adversarial data present.

B. *Challenge 2: Limited/No Access to the Target Model*

Several defenses assume that, as a defender, one would be able to access the internals of a model (e.g., model weights, gradients, layers, etc.). While certainly true for models created internally by one's own data science team or research team, this is not universally true. Like how there are "black-box" adversarial attacks (i.e., the attacker cannot see/access/know about the internals of a model), there are certainly situations where a black-box defense needs to be constructed. Enforcing this limitation forces one to, just like the first challenge, make little to no assumptions. The aim is a truly robust and generalizable defense, no matter the target model or attack.

III. Methodology

In this section, I will describe overall methodology for developing my defense, namely: (1) the data used, (2) the attacks used, (3) the defense developed, (4) and the experiment setup.

A. *Data - Covertype Data Set*

The data used in my experimentation is the "Covertype" data set [2]. The dataset has a total of 581,012 samples with 54 features and seven classes – each class corresponds to a particular forest cover type (1 = "Spruce/Fir", 2 = "Lodgepole Pine", etc.). More details on the dataset can be found at [2].

The reason I chose to use this dataset was because there are already many papers utilizing image datasets such as MNIST [9] and CIFAR-10 [10]. I wanted to move away from using just images and utilize tabular data instead.

B. Evasion Attacks

For my experiments, I utilized two evasion attacks for testing: the “Fast Gradient Sign Method” (FGSM) [5] and the decision tree attack described in [11].

These two attacks were chosen due to their speed in generating adversarial examples as well as the models they target. A brief overview of how each attack works will be given in the upcoming subsections. In general, FGSM can target models in which gradients are used to train them, e.g., logistic regression, support vector machines, deep neural networks, etc. The decision tree attack is, per its name, able to attack decision tree models – which, by extension, means an attack using the same or similar principles can be applied to attack other tree-based models like random forest models.

1) FGSM: As its name implies, the FGSM attack utilizes the gradients in a model to create a new adversarial example by perturbing an input’s features such that the loss for that input sample increases. Equation (1) shows this in mathematical form where x_{adv} is the adversarial output, x_{orig} is the original input, ϵ determines the size of the perturbation, and $J(\theta, x_{orig}, y_{orig})$ is the cost function for the model.

$$x_{adv} = x_{orig} + \epsilon \text{sign}(\nabla_x J(\theta, x_{orig}, y_{orig})) \tag{1}$$

[5] shows that, despite its simplicity, this attack can generate effective adversarial examples with very small values of ϵ, making it a good way to test if a defense can identify those adversarial examples.

2) Decision Tree Attack: The decision tree attack is also quite simplistic in its design yet highly effective. The attack is executed by modifying the value(s) of one or more features such that the adversarial input traverses down a different path taken by the original input to a new leaf node. The algorithm simply searches for a neighboring incorrect leaf node close to the original leaf node and adjusts the features of the input sample such that it fulfills the if-then conditions of each node in the decision tree path to traverse to the incorrect leaf node.

While [11] shows that adversarial examples created with this technique are not transferable to other models (i.e., adversarial examples crafted with this attack do not fool other models), it also shows that decision trees are one of the most vulnerable to having attacks transferred to it from models. Therefore, if I can defend a decision tree successfully with this defense with an attack created specifically for this decision tree, it should ideally be able to defend against adversarial examples transferred from other models.

An important observation to make is that FGSM and the decision tree attack are both “white-box” attacks – the attacker has access to the internals of the target model. This makes it more challenging as a defender since the attacker has the advantage of being able to access the model while I cannot. Therefore, even if I had access to the model internals, since the attacker does too, it is quite feasible for the attacker to easily break through any internal target model mechanisms I come up with.

C. Defense Development

The way my proposed defense works boils down to the use of two components: (1) an anomaly detector and (2) a generative adversarial network (GAN).

These two components are closely related – after all, the anomaly detector component could be a GAN itself, and the detection of adversarial examples can be thought of as being under the umbrella of “anomaly detection” since adversarial examples are anomalies. I separated the two here because the anomaly detector component can be *any* model: isolation forest, GAN, VAE, etc., whereas the GAN component is specifically that: a GAN.

While there are papers that use GANs for adversarial protection [12] and anomaly detection models for adversarial detection [1], my approach, to my knowledge, is novel in that it

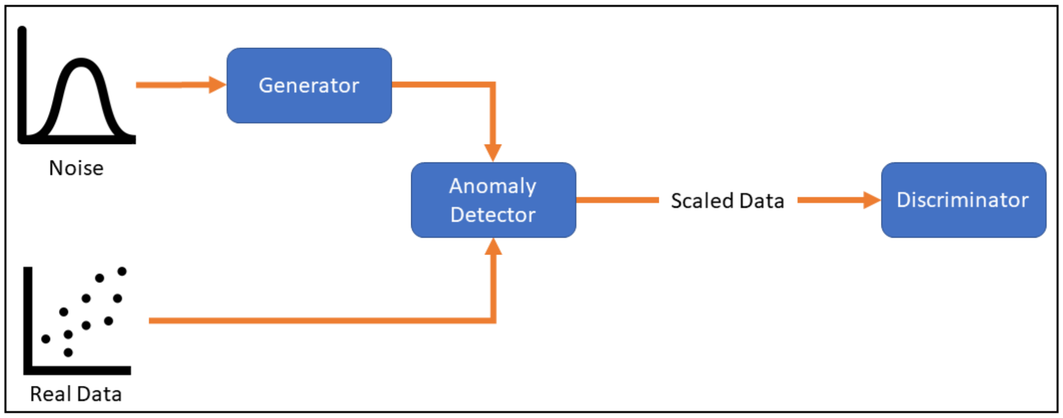

Fig. 1. The combined GAN-ADE model of using an anomaly detector between the generator and discriminator of a GAN. The anomaly detector is pre-trained on the real training data so it can learn the proper representation of real data. It is then used as a scaler for generated and real inputs in the GAN to aid the discriminator and challenge the generator.

combines the two components into one single architecture which I call GAN-ADE.

1) Overall Model: The combined architecture of GAN-ADE (Fig. 1) attempts to combine the probabilistic outputs of a pre-trained anomaly detector with the discriminative ability of the GAN.

Before training the GAN part of the model, an anomaly detector is trained on the training data. This way, it learns to accurately represent the training data. Ideally, adversarial data would be far enough from the representation learned by the anomaly detector that it would be flagged as an anomaly. Real data should be close to the anomaly detector's learned representation and not be flagged. However, rather than using the anomaly detector as a binary classifier for "outlier"/"not-outlier", I rely on the probabilistic output of the anomaly detector, i.e., I use the probabilities of being an outlier instead.

The GAN component of my defense is used to the discriminating between real and adversarial as well. The generator in the GAN tries to generate similar looking samples to the training data while the discriminator in the GAN attempts to tell real samples apart from generated samples. The difference here between a traditional GAN structure and mine is that the anomaly detector now sits between the generator and the discriminator of the GAN.

This "anomaly detection layer" allows for information about the "real" distribution of the data to be used in training a knowledgeable discriminator. When the generator creates samples, those samples are passed into the anomaly detector. The output probabilities of being non-anomalous is used to scale the generator inputs before being passed into the discriminator. Theoretically, due to them being generated samples, the probabilities of being non-anomalous should be small so the feature values get scaled down.

Real samples are processed the same way – they are first passed into the pre-trained anomaly detector and then scaled by their probability of being non-anomalous. Since they are real samples, they should be scaled down by very little or not at all since their non-anomalous probabilities should be close to 100%.

What this should do, ideally, is make more obvious which samples are real and which samples are fake to the discriminator and force the generator to work harder to generate real-looking samples since it has to "fool" two things: the anomaly detector *and* the discriminator.

When it comes to deploying the defense, only the anomaly detector and discriminator are used (Fig. 2). Incoming samples are first passed through the anomaly detector, scaled by their non-anomalous probabilities, and finally passed into the discriminator which would output some score. This output score can then be used to filter out adversarial examples before passing the rest of the data into the target model.

2) Anomaly Detector: There are a variety of anomaly detector models I could have used; however, the anomaly detector I decided to use was a variational autoencoder (VAE) [7].

The reason I chose to use a VAE was because they are similar in nature to a GAN – VAEs attempt to learn the distribution of the training data so it can reconstruct it with some degree of fidelity. While VAEs can be used to generate new examples, they can also be used to find anomalies by utilizing the reconstruction error of a trained VAE. If an input sample has a high reconstruction error from the trained VAE, it can be reasoned that that sample is not like the data used to train it. That reconstruction error can be turned into a probability by scaling based on the training reconstruction error.

3) GAN: For the GAN component, there were two types of GANs I experimented with: (1) the vanilla GAN [4] and (2) the Wasserstein GAN with gradient penalty (WGAN-GP) [6].

I chose these GANs for comparing a base GAN with a more state-of-the-art architecture to see if it would make a difference. There were some slight modifications I made to the architecture such as number of layers, activation functions, etc. since the original papers built them for usage on images. I also modified the WGAN-GP by making it conditional, i.e., making the generator create outputs based on the training labels. In essence, I combined parts of the Conditional GAN (CGAN) [10] with the WGAN-GP to make it a Wasserstein Conditional GAN with gradient penalty (WCGAN-GP).

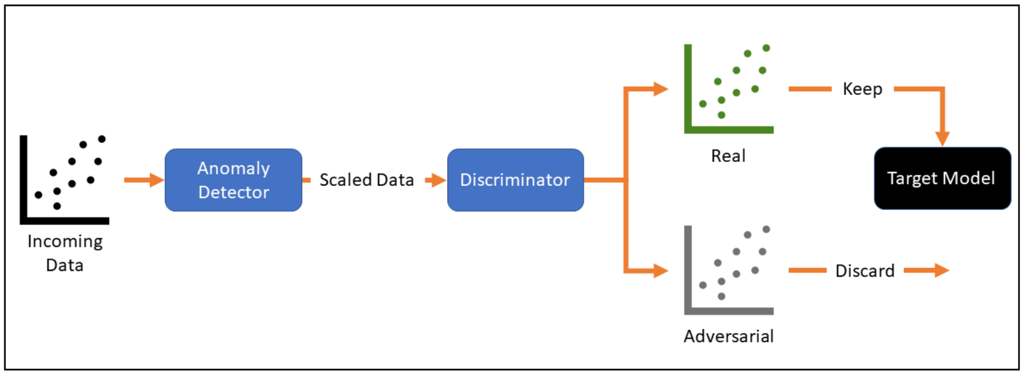

Fig. 2. The deployment of the defense uses just the anomaly detector and the discriminator. Incoming data is passed into the anomaly detector, scaled, run through the trained discriminator, and then filtered based on the discriminator score. Real data is kept and sent to the target model while adversarial data is discarded.

D. *Experiment Setup*

The experimental setup is broken into three phases: (1) baseline performance and adversarial example generation, (2) defense building, and (3) defense testing.

1) Phase 1: For the first phase, I set up my experiment like so:

1. Create two baseline models: a logistic regression model and decision tree model. These models are trained on 80% of the entire dataset. 20% is used for testing.
2. Get predictions with the held-out test data to get a baseline performance score for each model.
3. Create adversarial examples using FGSM and the decision tree attack (from the `adversarial-robustness-toolbox` Python library) for both models.
4. Combine adversarial examples with the test examples and pass them through their respective models to get predictions. The new performance scores are used as an indicator of how effective the adversarial examples were in fooling the models.

If the adversarial examples are good – i.e., they drop the performance of the models by a substantial amount – I move to the next step of crafting the defense. The adversarial examples are used to test how effective my defense is.

2) Phase 2: The first step in my defense is training the anomaly detector. I utilize `PyOD` Python library which has a VAE implementation for anomaly detection. Using the training data split out previously, I train the VAE for 100 epochs with a batch size of 512 and using the default architecture parameters. Architecture details can be found in `PyOD`'s documentation.[1]

Once the VAE is trained, I train the GAN models with the VAE embedded. Both the WCGAN-GP and vanilla GAN are trained for 100 epochs with a batch size of 512. The differences between them are, of course, the underlying architecture, as well as the optimizer parameters. The complete architecture and parameter setup can be found in the accompanying code repository.[2] After training completes for each GAN, I report out two sets of metrics: (1) approximate Leave-One-Out (LOO) accuracy and (2) discriminator scores on test data.

Approximate LOO accuracy is a metric adapted from [3] for determining how good a GAN is. The basic idea is to generate data using the generator of a trained GAN, combine it with real data, and assign the appropriate binary labels of "1" or "0" ("1" for real data; "0" for generated data). Then, passing that data into a 1-NN classifier and using Leave-One-Out cross-validation, get the overall LOO score. If the LOO score is close to or approximately 50%, that indicates that the generator replicated the real data well. Intuitively, if the generator copied the data exactly (i.e., overfit the training data), the LOO score would be 0% because the nearest neighbor of each generated point would be the real point it copied (distance of 0). If the LOO score is 100%, that means that the generator generated samples so different that the nearest neighbor was another generated sample. Due to runtime limits in the Google Colab environment my experiments were run in, I opted to approximate the LOO score by doing 10,000-fold cross-validation instead of Leave-One-Out cross-validation.

For the discriminator score reporting, the scores I reported differed due to the differing outputs of the GAN discriminators. The WCGAN-GP discriminator outputs some number generated by its last dense linearly activated layer. The vanilla GAN discriminator has a sigmoid-activated final layer, so the outputs are between 0 and 1.

For the WCGAN-GP discriminator, the training data is passed through the anomaly detector, scaled by the probability of being non-anomalous, and then passed into the discriminator. The average and standard deviation of the training scores is calculated and reported out. The standard deviation and mean of the WCGAN-GP training scores are then used in thresholding the discriminator output for filtering out adversarial examples during the defense testing phase. The WCGAN-GP discriminator scores cannot be objectively classified as "good" or "bad" since the discriminator score is used as a comparison measure, not as a direct measure of quality.

The vanilla GAN discriminator scoring uses the test data from the LOO scoring (the combined generated and real test data). Following the same procedure as for the WCGAN-GP discriminator, the LOO test data is passed through the anomaly detector. It is then scaled by the probability output and sent to the trained vanilla GAN discriminator. Rather than calculating average and standard deviation of the test scores, since the sigmoidal output can be interpreted as a probability, I round the outputs to the nearest integer (0 or 1) and use those as labels. Then I generate a classification report. If the discriminator was trained well, it should be able to differentiate between the generated samples and the real samples.

3) Phase 3: To test my trained defense model, I utilize the same adversarial test and real test data used in phase 1. For each model, (decision tree and logistic regression), I take the respective combined test data (adversarial plus real examples) and follow the process outlined in Fig. 2 as if this was a deployed defense: the data is passed through the anomaly detector, scaled by the detector probabilities, and passed into the discriminator. The output scores are then thresholded to determine what gets labeled as "real" and "adversarial".

For both WCGAN-GP and vanilla GAN, a variable called γ is used as the "thresholding" variable. The thresholding strategy for each model was different due to the differing outputs of the discriminators.

[1] https://pyod.readthedocs.io/en/latest/pyod.models.html#module-pyod.models.vae

[2] https://github.com/msalceda/csci-6364-final-project

For the WCGAN-GP, the threshold for a real or adversarial sample was:

$$\mu_{train\_dis\_score} \pm \gamma\ \sigma_{train\_dis\_score} \quad (2)$$

$\mu_{train\_dis\_score}$ is the mean WCGAN-GP discriminator training score and $\sigma_{train\_dis\_score}$ is the standard deviation of the WCGAN-GP discriminator training scores. If the output discriminator score of a sample falls outside the range defined by (2), it would be classified as adversarial. By varying γ, one can get stricter (lower γ) or looser (higher γ) constraints on what is considered adversarial or not.

The vanilla GAN uses a different and simpler thresholding scheme: γ is used as a threshold score. If the output discriminator score is greater than γ, it is classified as real; otherwise, it is classified as adversarial.

γ is varied for both the WCGAN-GP and vanilla GAN. The final accuracy, F1, and recall scores across γ values of the respective discriminators are reported in the "Results" section.

IV. Results

The following sections break down the results from each phase of the experiment.

A. Phase 1: Baseline Modeling & Adversarial Generation

For the first phase of the experiment, I created a baseline decision tree model and logistic regression model trained on 80% of the original Covertype data set. Then I created adversarial test examples for both models using their respective attacks, combined the adversarial test examples with the real test examples, and ran them through the models. Table I shows the performance results of the original test set and the combined test set.

TABLE I. BASELINE MODEL RESULTS

Models	**Model Performance**			
	Normal Test		***Normal + Adversarial Test***	
	Accuracy	***F1***[a]	***Accuracy***	***F1***[a]
Logistic Regresion (LR)	0.60	0.50	0.31	0.32
Decision Tree (DT)	0.94	0.89	0.49	0.48

[a] F1 scores reported are **unweighted macro averages** across all seven classes.

Even with small perturbations ($\epsilon = 0.3$ for the FGSM attack and perturbations of 0.001 for the decision tree attack), one can see that both attacks were able to successfully generate adversarial examples that degenerated the performance of the baseline models.

B. Phase 2: Defense Training

Phase 2 involved the training of the defense – first of that is to train the anomaly detector. The anomaly detector used, as previously mentioned, was a VAE. To test if using a VAE by itself was sufficient for finding adversarial examples, I ran the real test examples through and labeled examples above 0.5 probability anomalous to be adversarial. I did the same for the logistic regression adversarial examples and the decision tree adversarial examples. The results are in Table II.

TABLE II. VAE ADVERSARIAL DETECTION RESULTS

	% of Data Labeled as Adversarial
Real Test Examples	0.00
Adversarial LR Test Examples	0.00
Adversarial DT Test Examples	0.00

Ideally, the percentage of data labeled as adversarial would be 0% for the real test data and 100% for both the logistic regression and decision tree adversarial examples. Obviously, this was not the case: while the VAE model correctly classified 0% of the real test data as adversarial, it completely missed on any of the adversarial examples generated by the FGSM and decision tree attacks. It can be argued that the VAE underfit the training data since it was not able to identify the adversarial examples on its own, but it can also be argued that perhaps the attacks generated examples so well that they fit within the learned distributions of the training data. In either case, the results indicate the need for something else to aid in identifying adversarial examples besides just the VAE.

Following the training of the VAE was the training of the GANs. For each GAN, the LOO accuracy score and the GAN-specific metrics are recorded in Table III.

TABLE III. GAN TRAINING METRICS

Models	**Model Performance**				
	LOO	$\mu_{train\_dis\_score}$	$\sigma_{train\_dis\_score}$	***F1***	***Accuracy***
WCGAN-GP	100.00%	-423.43	863.22	-	-
Vanilla GAN	100.00%	-	-	1.00	1.00

From the LOO scores, it seems that the generators for both WCGAN-GP and vanilla GAN did not train well as the 1-NN classifier model was able to completely distinguish between real and generated samples. Based on the vanilla GAN test F1 and accuracy scores, the vanilla GAN discriminator looks to be able effectively distinguish between generated and real samples which, as one can see from the vanilla GAN LOO score, is most likely due to the generator not generating good examples.

C. Phase 3: Defense Testing

With the training completed, the testing of the finished defense was done. As previously explained, to test the defense, the adversarial examples generated by the FGSM and decision tree attacks were each combined into a dataset containing real test data and the adversarial test data, i.e., one test set had FGSM adversarial examples with real examples and the other test set had decision tree adversarial examples with real examples.

Each respective dataset, one for the LR model and one for the DT model, were processed according to the defense deployment outlined in Fig. 2, and various threshold values, γ, were tested for filtering out adversarial and real examples. The results of the testing phase are in Table IV and Table V.

TABLE IV. TEST RESULTS FOR WCGAN-GP

γ	**GAN-ADE Model Performance**					
	Logistic Regression Real + Adversarial Data			***Decision Tree Real + Adversarial Data***		
	Accuracy	***F1***[b]	***Recall***[c]	***Accuracy***	***F1***[b]	***Recall***[c]
0.5	0.59	0.57	**0.82**	**0.63**	**0.60**	**0.91**
1.0	**0.64**	**0.64**	0.64	0.52	0.52	0.40
1.5	0.60	0.58	0.35	0.47	0.39	0.10
2.0	0.54	0.44	0.11	0.50	0.36	0.03
2.5	0.51	0.35	0.01	0.50	0.33	0.00
3.0	0.50	0.33	0.00	0.50	0.33	0.00

b. F1 scores reported are **unweighted macro averages** across the "real" and "adversarial" classes.

c. Recall scores (i.e., true positive rate) is reported for the **"adversarial"** class.

TABLE V. TEST RESULTS FOR VANILLA GAN

γ	**GAN-ADE Model Performance**					
	Logistic Regression Real + Adversarial Data			***Decision Tree Real + Adversarial Data***		
	Accuracy	***F1***[d]	***Recall***[e]	***Accuracy***	***F1***[d]	***Recall***[e]
0.01	0.50	0.33	0.00	0.50	0.33	0.00
0.05	0.50	0.33	0.00	0.50	0.33	0.00
0.10	0.50	0.33	0.00	0.50	0.33	0.00
0.25	0.50	0.33	0.00	0.50	0.33	0.00
0.50	0.50	0.33	0.00	0.50	0.33	0.00
0.75	0.50	0.33	0.00	0.50	0.33	0.00
0.99	0.50	0.33	0.00	0.50	0.33	0.00

d. F1 scores reported are **unweighted macro averages** across the "real" and "adversarial" classes.

e. Recall scores (i.e., true positive rate) is reported for the **"adversarial"** class.

The WCGAN-GP performed much better compared to the vanilla GAN. Even though the threshold γ varied from 0.01 to 0.99 for the vanilla GAN, it looks as if the discriminator was underfit and not trained well since it classified everything as "real" and did not identify *any* adversarial samples.

On the other hand, the WCGAN-GP did decently well as its threshold γ was varied. If one wanted to optimize the F1 score for the logistic regression model, a γ of 1.0, i.e., any discriminator score that fell outside of -423.43 ± 863.22, would be best for balance. If one wanted to be very strict, the best would be a γ of 0.5 which is a much tighter range of -423.43 ± 431.61 since it has a high adversarial recall score of 0.82. For the decision tree adversarial examples, the WCGAN-GP peaked at a lower γ of 0.5 and adversarial recall of 0.91.

V. CONCLUSION

My work has shown that it is possible to detect adversarial examples without modifying or accessing the target model. I found that utilizing a simple VAE for adversarial detection is not enough and neither is using a vanilla GAN combined with a VAE anomaly detector. It was only with the WCGAN-GP architecture that I was able to get some promising results in being able to successfully filter out adversarial examples.

Future work includes the further tuning and development of both components of this defense: the anomaly detector and the GAN. I did not heavily invest in tuning the anomaly detector or the GAN, and there are many parameters that could be modified.

For the anomaly detector, since I was using a VAE, there are many ways to build a VAE by varying the decoder-encoder layers, varying the latent dimensions for the encoding, tuning the parameters of the optimizer, etc. One could also experiment with replacing the VAE with another anomaly detector model like an isolation forest model or a 1-class SVM model.

The GANs can be modified in many ways as well, from adding layers to playing with different learning rates for the discriminator and generator and adapting different architectures.

In the end, there is still much room for improvement in this space. Detecting adversarial examples is a challenging and ever-evolving area of research so hopefully this work can serve as a starting point for some new ideas.

REFERENCES

[1] D. A. Bierbrauer, A. Chang, W. Kritzer, and N. D. Bastian, "Cybersecurity Anomaly Detection in Adversarial Environments," *arXiv e-prints*, p. arXiv:2105.06742, 2021.

[2] J. A. Blackard and D. J. Dean. *Covertype Data Set*, UCI Machine Learning Repository

[3] A. Borji, "Pros and Cons of GAN Evaluation Measures: New Developments," *arXiv e-prints*, p. arXiv:2103.09396, 2021.

[4] I. J. Goodfellow *et al.*, "Generative Adversarial Networks," *arXiv e-prints*, p. arXiv:1406.2661, 2014.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv e-prints*, p. arXiv:1412.6572, 2014.

[6] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein GANs," presented at the Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, California, USA, 2017.

[7] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv e-prints*, p. arXiv:1312.6114, 2013.

[8] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.

[9] Y. LeCun and C. Cortes. *The MNIST Database of Handwritten Digits*

[10] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv e-prints*, p. arXiv:1411.1784, 2014.

[11] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples," *arXiv e-prints*, p. arXiv:1605.07277, 2016.

[12] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models," *arXiv e-prints*, p. arXiv:1805.06605, 2018.

[13] C. Szegedy *et al.*, "Intriguing properties of neural networks," *arXiv e-prints*, p. arXiv:1312.6199, 2013.