Adversarial Machine Learning With Image Classification

Aya Hourani
Software Engineering
Rutgers University
New Brunswick, NJ, USA
aah173@scarletmail.rutgers.edu

[1] ***Abstract*—The utilization of machine learning in various applications has improved problem-solving capabilities in revolutionary ways. However, the widespread use of machine learning, particularly in applications such as image and speech recognition, data analysis, and product recommendations, proves the need to strengthen the security of these algorithms. This is due to their susceptibility to various adversarial attacks that compromise data integrity and confidentiality. Adversarial machine learning serves as a pivotal tool in identifying and mitigating vulnerabilities within machine learning models. This paper delves into the exploration of adversarial attacks, focusing on white-box scenarios, and introduces the Projected Gradient Descent method as an iterative optimization-based adversarial attack; it describes the process of creating an image classifier model using convolutional neural networks. The results demonstrate the model's susceptibility to adversarial perturbations, leading to misclassifications. Finally, the paper discusses potential defenses, shedding light on the importance of retraining models with adversarial examples to enhance robustness.**

***Index Terms*—Adversarial Machine Learning, Image Classification, Projected Gradient Descent**

I. INTRODUCTION

A. Motivation

Machine learning has become very prevalent in modern technology, allowing industries to solve problems more effectively without explicitly telling the machine what to do. Many industries utilize machine learning for image and speech recognition, data analysis, product recommendations, and so much more. This emphasizes the need for improving security within machine learning, as they are vulnerable to many types of attacks. Preserving the integrity and confidentiality of this data is crucial to maintaining these machine learning algorithms safely and at no cost to both the consumers and developers. Machine learning algorithms require access to an abundance of data to work. Adversarial machine learning helps identify vulnerabilities and weaknesses in machine learning models. Understanding how models behave under adversarial conditions allows developers to enhance the models' resilience to attacks and unexpected inputs. Adversarial learning contributes to the development of more secure machine learning systems.

[1]

B. Related Work

Much work has been done relating to adversarial machine learning and its applications. One area explored is adversarial deep reinforcement learning, which focuses on the vulnerabilities of learned policies [4]. Reinforcement learning is a machine learning example where an agent interacts with an environment, taking actions to maximize a cumulative reward signal. Deep reinforcement learning uses deep neural networks to represent complex mappings from states to actions or values. Adversarial examples in reinforcement learning refer to perturbations or modifications to the input space that are designed to mislead or deceive the learning agent. The main goal of Adversarial Deep Reinforcement Learning is to enhance the robustness and generalization capabilities of the reinforcement learning agent.

Another area explored is adversarial natural language processing, which studies adversarial attacks on speech recognition for speech-to-text applications [3]. Adversarial Natural Language Processing (ANLP) involves the integration of adversarial techniques into the development and improvement of natural language processing (NLP) models. Similar to adversarial deep reinforcement learning, the goal is to enhance the robustness, security, and capabilities of natural language processing systems, which are also susceptible to adversarial attacks and vulnerabilities.

Finally, there has been extensive research conducted concerning machine learning, particularly for image classification. Machine learning for image recognition has many applications, including face recognition, video surveillance analysis, 3D image vision, and medical image diagnosis [1]. Many techniques are utilized for image recognition, including convolutional neural networks, feature extraction, machine learning, deep learning, transfer learning, and data augmentation.

II. ADVERSARIAL ATTACK METHODS

This section will explore the adversarial attack methods used in machine learning.

A. Adversarial Machine Learning

Adversarial machine learning is the study of attacks on machine learning algorithms, as well as the defenses against them. Evasion attacks are the most common and can be classified into white-box and black-box attacks, specifying the level of knowledge the adversary has. For example, the

adversary can have complete knowledge, limited knowledge, or zero knowledge of the system.

B. White-Box Attacks

In a white-box scenario, the attackers have detailed knowledge of the machine learning algorithm, model, and architecture. The testing process is based on the understanding of how the software functions internally. The adversary has read and write access to the training dataset of the targeted learning system [2]. The adversary can modify the learning model configurations. One of the advantages is that white-box testing allows for thorough coverage of code and logic, making it effective for finding internal errors, optimizing code, and ensuring complete code coverage.

C. Black-Box Attacks

In a black-box scenario, the attackers know almost nothing about the machine learning system. Attackers in a black-box scenario do not have access to the internal code, algorithms, or implementation details of the system. The adversary doesn't have access to the training dataset of the targeted system [2]. They focus on understanding the inputs, outputs, and expected behaviors of the system. One of the advantages is that black-box testing is useful for assessing the system from a user's perspective and validating that it meets the specified requirements.

D. Projected Gradient Descent

For the scope of this project, we're going to assume a white-box scenario in that the adversary has detailed knowledge of the internal structure of the model. The projected gradient descent will be used to generate the adversarial example and demonstrate how the model can incorrectly predict the label of an image. Projected Gradient Descent is an iterative optimization-based attack that aims to find adversarial examples by making small perturbations to input data in the direction of the gradient of the loss function with respect to the input. Unlike traditional gradient descent used for model training, PGD uses the model's predictions on adversarial examples to estimate gradients during each iteration.

PGD is often applied iteratively, with each iteration refining the perturbation to increase the model's loss, and this also increases the likelihood of misclassification.
Given that the success of PGD relies on the ability to compute or estimate gradients with respect to the input, it requires knowledge of the target model's internal parameters, making it a white-box attack. PGD is commonly used as a tool for evaluating the robustness of machine learning models. It helps assess how well a model withstands adversarial perturbations, providing insights into potential vulnerabilities. PGD can be defined using the following equation:

$$x^{t+1} = \Pi_{x+S}(x^t + \alpha sgn(\nabla_x L(\Theta, x, y))) \text{ [7]}$$

Parameter Descriptions:

- x^{t+1}: The perturbed version of the original input at iteration t+1. It's the updated input that results from applying an adversarial perturbation.
- Π_{x+S}: A projection operator that enforces certain constraints on the perturbed input. It ensures that the perturbed input lies within a specific feasible region.
- $\nabla_x L(\Theta, x, y)$: The gradient of the loss function L with respect to the input x. It provides information about how the loss changes concerning small changes in the input.
- sgn: Computes the sign (direction) of the gradient. It indicates the direction in which the loss function increases the fastest.
- α: Hyperparameter that scales the adversarial perturbation. It controls the magnitude of the perturbation applied to the input.
- S: The scaling of the perturbation. It adjusts the perturbation to ensure it is within certain bounds or satisfies specific constraints.

III. METHODOLOGY

This section will describe the methodology used to implement the image classifier model.

A. Importing and Preprocessing Data

The image classifier will take 2 types of images as input, shapes and numbers. To obtain the data, images were extracted from Google using the Download All Images Extension; this allows us to search for an image and automatically download the first 100 images and create a zip folder containing them. We use this process to obtain the images for both the shapes and the numbers. Once we collect the labeled dataset, the next step is to filter out the invalid image file types (file types outside the common jpeg/png file types). This is to ensure that the classifier only takes input images with the correct file type. Figure 1 visualizes a set of 4 randomly selected images from the dataset, along with their class labels. These labels are depicted as binary numbers 0 and 1, with 1 representing the shapes and 0 the numbers. We end this preliminary stage with preprocessing the data by scaling the images accordingly.

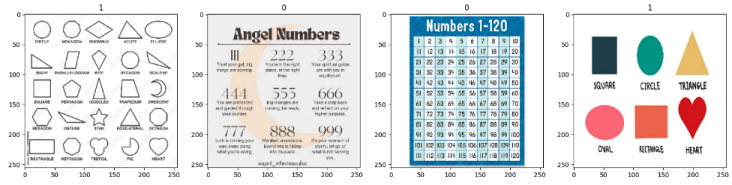

Figure 1: Visualization of images with class labels (0 and 1)

The next step is to split the data and establish appropriate partitions. The dataset contains approximately 200 images which are partitioned into training, validation, and test sets. Figure 2 describes the number of batches allocated to each set. Each batch contains approximately 28 images so we're assigning each data size a total number of:

- Training size: 112 images
- Validation size: 56 images
- Test size: 28 images

Data	Batches
Training	4
Validation	2
Test	1

Figure 2: Data partitions

B. Building the Deep Learning Model

Before feeding images into a machine learning model, features are extracted to represent the characteristics of the images. In traditional machine learning, these features might include color histograms, texture features, or other descriptors. In deep learning, convolutional neural networks (CNNs) are commonly used to automatically learn relevant features from raw pixel values. CNNs are designed to automatically learn hierarchical features from images, making them well-suited for image classification tasks. For this classifier, we will use Convolutional Neural Networks to learn the features of our data. The Tensorflow library is used to build the deep learning model using CNNs. TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but especially focuses on the training and inference of deep neural networks.

We introduce multiple layers in our convolutional neural network, each layer serving its own purpose, and we can model our CNN as shown in Figure 3.

```
# Adding the layers to the network
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
model.add(MaxPooling2D())

model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

# Condense rows and width and # of channels
model.add(Flatten())

# Fully connected layers
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Figure 3: CNN structure

Layer Functions:

- MaxPooling: Condenses the values and returns the maximum values.
- Flatten: Used to flatten the inputs and keep the batch size the same. It condenses the rows, width, and number of channels.
- Dense: Contains all the neurons that are deeply connected within themselves.

In addition to the layers, we also apply activation functions ReLU and Sigmoid. Their behaviors are shown as modeled in Figure 4.

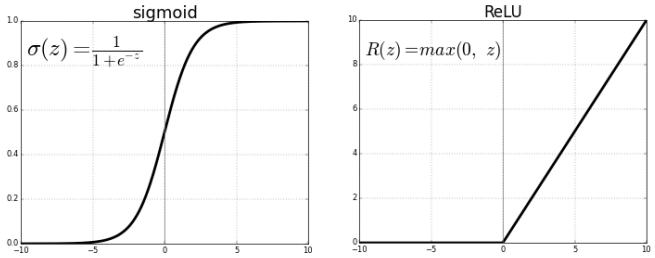

Figure 4: Behaviors of activation functions

Activation Functions:

- ReLU: Referred to as a rectified linear unit. It outputs the input value for any positive input and zero for any negative input. It introduces nonlinearity to the model, which enables it to learn from data and make more complex predictions. [5]
- Sigmoid: The output is in the range 0-1, which is beneficial for binary classification problems or any task where the output needs to be interpreted as probabilities. This is ideal for our image classifier, as the model classifies images using binary markers. [5]

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 16)      448

max_pooling2d (MaxPooling2D) (None, 127, 127, 16)      0

conv2d_1 (Conv2D)            (None, 125, 125, 32)      4640

max_pooling2d_1 (MaxPooling2D) (None, 62, 62, 32)      0

conv2d_2 (Conv2D)            (None, 60, 60, 16)        4624

max_pooling2d_2 (MaxPooling2D) (None, 30, 30, 16)      0

flatten (Flatten)            (None, 14400)             0

dense (Dense)                (None, 256)               3686656

dense_1 (Dense)              (None, 1)                 257
```

Figure 5: Final deep learning model

Figure 5 displays a summary of the deep learning model generated, detailing the layers of the network, and the output of their shapes.

C. Model Training

After generating the deep learning model, it's time to train it using the training data set. The model is fit over the training size (4 batches) and runs over 20 epochs. Figure 6 shows the process of training the model over the first 10 epochs, and metrics measured are the accuracy and loss. Ideally, as the model runs more epochs, the loss value should be decreasing while the accuracy is increasing.

```
Epoch 1/20
4/4 [==============================] - 6s 1s/step - loss: 2.1957 - accuracy: 0.5234 - val_l
oss: 1.2105 - val_accuracy: 0.5469
Epoch 2/20
4/4 [==============================] - 5s 953ms/step - loss: 1.0887 - accuracy: 0.5781 - va
l_loss: 0.6951 - val_accuracy: 0.5625
Epoch 3/20
4/4 [==============================] - 5s 989ms/step - loss: 0.6665 - accuracy: 0.5938 - va
l_loss: 0.6244 - val_accuracy: 0.6719
Epoch 4/20
4/4 [==============================] - 5s 1s/step - loss: 0.6713 - accuracy: 0.6797 - val_l
oss: 0.4833 - val_accuracy: 0.8281
Epoch 5/20
4/4 [==============================] - 5s 1s/step - loss: 0.5867 - accuracy: 0.6484 - val_l
oss: 0.4958 - val_accuracy: 0.7969
Epoch 6/20
4/4 [==============================] - 6s 1s/step - loss: 0.4988 - accuracy: 0.7734 - val_l
oss: 0.4662 - val_accuracy: 0.7656
Epoch 7/20
4/4 [==============================] - 5s 1s/step - loss: 0.4469 - accuracy: 0.8594 - val_l
oss: 0.3580 - val_accuracy: 0.9062
Epoch 8/20
4/4 [==============================] - 5s 1s/step - loss: 0.3550 - accuracy: 0.8359 - val_l
oss: 0.3734 - val_accuracy: 0.8281
Epoch 9/20
4/4 [==============================] - 5s 1s/step - loss: 0.2748 - accuracy: 0.8672 - val_l
oss: 0.1805 - val_accuracy: 0.9688
Epoch 10/20
4/4 [==============================] - 5s 1s/step - loss: 0.2213 - accuracy: 0.9141 - val_l
oss: 0.1444 - val_accuracy: 0.9531
```

Figure 6: Model training over the first 10 epochs.

After the model has finished training, we can plot its performance by measuring its loss and accuracy over the 20 epochs, as modeled in Figure 7. As previously mentioned, the loss should be decreasing with increasing epochs, and the accuracy should be increasing. From the plots, we can observe that this behavior is well defined, as the loss approaches 0 by the end of the last epoch and the accuracy approaches 1.

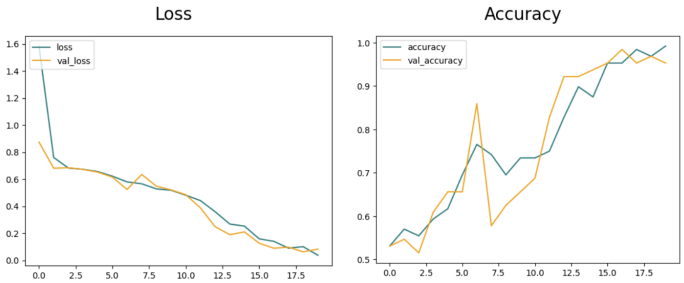

Figure 7: Performance plots for the image classifier

D. Model Testing

Now that we've generated our image classifier model, trained it using the training dataset, and modeled its performance, it's time to test the model using test images outside our dataset. To test the accuracy of the model, 2 test images will be used, a shape image and a number image. After importing a shape image and running the classifier, the model generates a prediction of 0.86675847. The model is configured to output a label of 1 if the prediction is greater than 0.5, and 0 if it's less than 0.5. In this scenario, the output is significantly greater than 0.5, so the model generates a class label of 1 for the image. This is correct as if we recall in the preliminary stage, a shape image is assigned the class label 1 and the number image 0. We can also test our model using a number image. In this case, the model generates a prediction of 0.01677191, which is very close to 0, thus the model labels this image with a 0 label.

E. Generating an Adversarial Example

Now that we've created our image classifier model, trained it on our dataset, and tested its accuracy, it's time to apply the adversarial attack. As previously stated, we are utilizing the Projected Gradient Descent method to generate an adversarial example. We are going to use both a shape and number image as our test images, so our original prediction should generate a label close to the value 1 for the shape and 0 for the number, while our adversarial model should generate a label closer to 0 for the shape and 1 for the number. We are going to measure our image using different values of epsilon (hyperparameter), and observe how that affects not only the prediction, but also the image itself. Figures 8 and 9 show the results of applying PGD to our shape and number images using multiple values for epsilon.

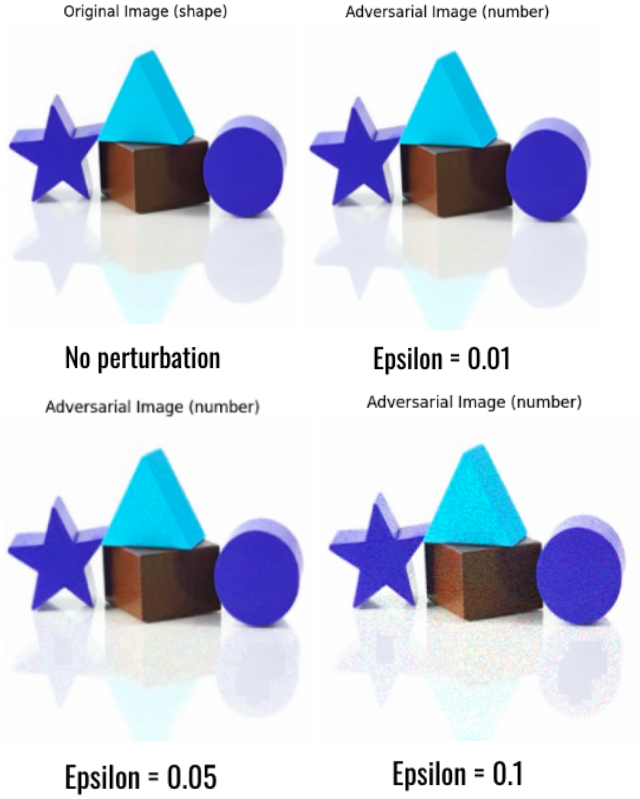

Figure 8: The adversarial shape image generated at different epsilon values

Figure 9: The adversarial number image generated at different epsilon values

IIII. RESULTS

A. Discussion of Results

From Figure 8, we observe greater noise in the image as we increase epsilon, indicating more perturbation has been applied to the image; the perturbation is most visible on the triangle. We can also see that the adversarial model generated incorrect labels for the image, as the plots indicate that the image is a number as opposed to a shape. To understand the true impact of the perturbation applied, we can measure how the model becomes less accurate in classifying the image as we increase the amount of perturbation by looking at the actual predictions, as shown in Figure 10.

Epsilon Value (Amount of Perturbation)	Prediction
0.01	0.49899125
0.05	0.28940347
0.1	0.18573707

Figure 10: Epsilon Value vs. Prediction

We can compare these predictions using the prediction of the original image, which is **0.65125793**. From Figure 10, we can conclude that stronger perturbations result in much less accurate predictions. Configuring the epsilon value to 0.01 doesn't significantly impact the accuracy of the model, however it does affect it to the point where the image is incorrectly labeled. This phenomenon is only strengthened when we increase the perturbation. We can also measure the behavior of the adversarial model using a test number image, and Figures 9 and 11 demonstrate the experimental results.

Epsilon Value (Amount of Perturbation)	Prediction
0.01	0.6468667
0.05	0.99909645
0.1	0.99996257

Figure 11: Epsilon Value vs. Prediction

Similar to the adversarial shape image, we can compare these predictions with the prediction of the original number image, which is **0.03178492**. Unlike the adversarial example using the shape image, we see a much more significant change in the prediction from just applying a perturbation of 0.01 to the image. Remember as previously mentioned, to distinguish between class labels, the model is configured to round the prediction up or down depending on whether or not it's greater than 0.5. Just like the shape image, even an epsilon value of 0.01 is enough to disturb the image classifier and cause an incorrect classification. The difference in the amount of perturbation required to generate an incorrect label for a number and shape image can be further explored by studying the features of the images themselves; for the scope of this project, this topic is left for future discussion. It's more difficult to see the perturbation on the number image, however it is most visible on the actual numbers (143).

From the generated adversarial examples, we can draw an important conclusion regarding the effect of adding perturbations to images. While increasing the amount of perturbation decreases the model's ability to accurately predict the image, it also greatly alters the original image, making the perturbation more noticeable. This indicates that more robust models will require greater perturbations to decrease the model's accuracy, in which case the adversarial attack will be deemed ineffective since the attack will become apparent.

B. Possible Defense

Although the focus of this project was to demonstrate an adversarial attack on our image classifier model, it's worth mentioning how we can mitigate this attack. Retraining with adversarial images is a technique used in machine learning to enhance the robustness of a model against adversarial attacks [6]. In this case, we would generate adversarial examples using an attack method like projected gradient descent from the original training data. We would then modify the original training dataset by including the generated adversarial examples to create a more diverse training dataset. From here, we would just retrain the model and assess its performance and robustness. As previously mentioned, the projected gradient descent is a great method for testing the robustness of machine learning models.

V. CONCLUSION

In conclusion, this paper emphasizes the need for enhancing the security of machine learning systems, given their applications in modern technology. The exploration of adversarial machine learning sheds light on the vulnerabilities inherent in these systems. The application of Projected Gradient Descent as an adversarial attack method on an image classifier model demonstrates the susceptibility of such models to perturbations, leading to incorrect predictions. The results highlight the motivation to develop robust defenses against adversarial attacks. Retraining with adversarial images was introduced as a defense method for generating more robust models and defending against adversarial attacks. As machine learning continues to advance, safeguarding these technologies becomes essential, ensuring their reliability, integrity, and resilience in the face of evolving adversarial landscapes. This paper contributes valuable insights toward the ongoing mission for securing and advancing the field of machine learning.

ACKNOWLEDGMENT

I would like to thank Prof. Xiong Fan for introducing various computer system security measures and techniques that are widely used in many applications.

References

[1] C. Li, X. Li, M. Chen and X. Sun, "Deep Learning and Image Recognition," 2023 IEEE 6th International Conference on Electronic Information and Communication Technology (ICEICT), Qingdao, China, 2023, pp. 557-562, doi: 10.1109/ICEICT57916.2023.10245041.

[2] S. Y. Khamaiseh, D. Bagagem, A. Al-Alaj, M. Mancino and H. W. Alomari, "Adversarial Deep Learning: A Survey on Adversarial Attacks and Defense Mechanisms on Image Classification," in IEEE Access, vol. 10, pp. 102266-102291, 2022, doi: 10.1109/ACCESS.2022.3208131.

[3] Zhang, W. E., Sheng, Q. Z., Alhazmi, A., & Li, C. (2019). Adversarial Attacks on Deep Learning Models in Natural Language Processing: A Survey. ArXiv. /abs/1901.06796

[4] Korkmaz, E. (2023). Adversarial Robust Deep Reinforcement Learning Requires Redefining Robustness. ArXiv. /abs/2301.07487

[5] dishashree26. "Fundamentals of Deep Learning - Activation Functions and When to Use Them?" Analytics Vidhya, 3 Nov. 2023, www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/.

[6] Thakur, N., Ding, Y., & Li, B. (2020). Evaluating a Simple Retraining Strategy as a Defense Against Adversarial Attacks. ArXiv. /abs/2007.09916

[7] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards Deep Learning Models Resistant to Adversarial Attacks. ArXiv. /abs/1706.06083