HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

Computer Science department

Machine Learning in Cybersecurity

T-710-MLCS

Report 5 - Adversarial Machine Learning

Saldana Giorgio
Email: giorgio24@ru.is

21st October 2024

Contents

Introduction

Adversarial machine learning has become an essential area of study in modern AI security, particularly with the rise of neural network models in sensitive applications. The ability to manipulate inputs subtly to deceive machine learning models—without altering human-perceived information—poses significant security challenges. In this assignment, we focus on adversarial attacks in the context of image recognition, leveraging the MNIST dataset of handwritten digits.

The core objective is to understand and generate adversarial samples that can fool image classifiers. This involves the implementation of a convolutional neural network (CNN) to classify the MNIST dataset, followed by the generation of adversarial examples using Fast Minimum Norm attacks from the Foolbox library. These adversarial techniques exploit vulnerabilities in model robustness, revealing potential weaknesses in machine learning systems.

The final goal is to assess the impact of different adversarial strategies on model performance and document how varying norms (L0, L1, L2, Linf) influence attack effectiveness. Through this exploration, we aim to gain a deeper understanding of machine learning vulnerabilities and evaluate the resilience of models in adversarial environments.

The report is structured as follow:

- Simple Data Preprocessing and Splitting
- Generate adversarial samples

For reproducibility, the code can be found here.

To run the experiment classifying mnist dataset using a CNN:

```
python train.py
```

Once the model is created, you can generate adversarial samples:

```
python adversarial.py
```

1 Simple Data Preprocessing and Splitting

In this implementation, a class-based approach is used to improve modularity and readability. The `MNISTModel` class encapsulates the key steps for loading data, building a Convolutional Neural Network (CNN), training, and evaluating the model.

Data Preprocessing

The MNIST dataset is loaded using Keras, reshaped into the required format for a CNN, and normalized to ensure consistent input. Labels are one-hot encoded for classification. Labels are one-hot encoded for classification to prepare the data for a multi-class classification problem. Additionally, the data is checked for any missing or NaN values, ensuring data integrity before training the model.

Model Architecture

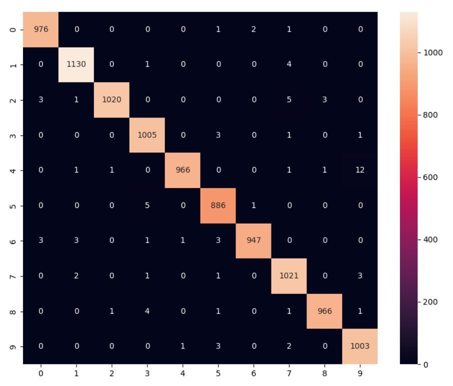
The CNN consists of several convolutional layers, followed by max-pooling and dropout layers to reduce overfitting. A fully connected layer and softmax output layer classify the digits. The model is compiled with categorical cross-entropy and optimized using RMSprop. To create the CNN model, a guideline provided by Kaggle was followed, available at the following link: `https://www.kaggle.com/code/amyjang/tensorflow-mnist-cnn-tutorial`.

Training and Evaluation

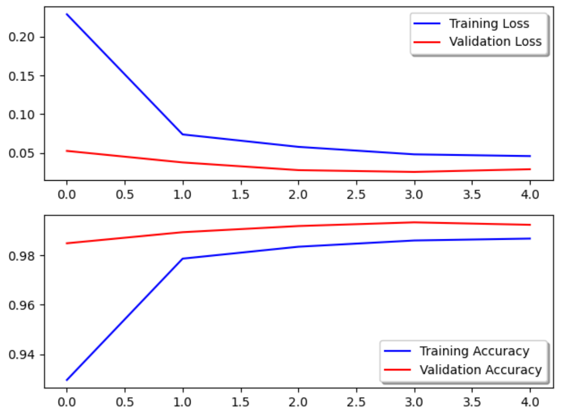
The model is trained for 5 epochs, and the training process is tracked for both accuracy and loss. The final model achieves a test accuracy of 99.13%, meeting the assignment's requirements. Importantly, after training, the model is saved to a file, `mnist_cnn_model.h5`, for future use in generating adversarial samples and further analysis.

Visualization

The model's performance is visualized using two key plots: the confusion matrix and the training history (loss and accuracy). The confusion matrix provides insight into the classification results for each digit, while the training history shows the model's progression over the training epochs.



(a) Confusion Matrix



(b) Training Loss and Accuracy

Figure 1: Model performance visualization: confusion matrix and training history.

2 Generate adversarial samples

This section explains the process used to generate adversarial samples with the Python library Foolbox, utilizing Fast Minimum Norm (FMN) adversarial attacks.

Fast Minimum Norm adversarial attacks Overview

Fast Minimum Norm (FMN) adversarial attacks were first introduced in the paper Fast Minimum-norm Adversarial Attacks through Adaptive Norm Constraints. These attacks aim to simplify the complexity of gradient-based methods, operating under different norms (0, 1, 2, and ∞) to find misclassified samples with maximum confidence. Several epsilon values were applied to observe how generated samples change, emphasizing the importance of choosing an appropriate epsilon, which represents the distance between original and perturbed samples.

Implementation

A class-based approach was employed to enhance code modularity and maintain adherence to the Open-Closed Principle, allowing new attacks to be integrated with minimal modifications.

Loading Model and Data Preparation

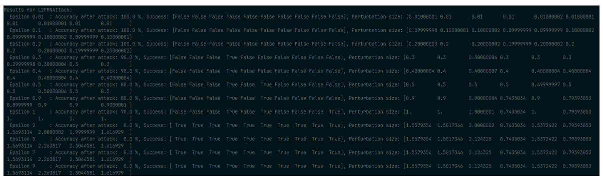
The model from the previous section, `mnist_cnn_model.h5`, was loaded using the Keras function `load_model()` and wrapped into a `TensorFlowModel` using Foolbox's built-in function. Data preparation involved selecting one sample for each digit (0-9) using a dictionary, ensuring the uniqueness of each digit. The dataset was then normalized and converted into `EagerPy` tensors backed by TensorFlow, as required by Foolbox. After these steps, the data was ready for adversarial attacks.

Performing Attacks

A list of tuples was defined, representing the various attacks executed by the `AdversarialTester` class. The attacks included:

- L0FMNAttack
- L1FMNAttack
- L2FMNAttack
- LInfFMNAttack

The attacks were divided into two main categories: those requiring an explicit epsilon value, such as L2FMNAttack, L1FMNAttack, and LInfFMNAttack, and the L0FMNAttack, which determines the optimal epsilon automatically. A similar evaluation of model accuracy, based on varying epsilon values, was conducted in line with the professor's provided code here. An example result for L2FMNAttack is shown below, more results are obtainable from the experiment reproduction detailed in the Introduction section.



```
Results for L2FMNAttack:
 Epsilon 0.01  : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0.01000001 0.01       0.01       0.01       0.01000002 0.01000001
 0.01       0.01000001 0.01       0.01      ]
 Epsilon 0.1   : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0.09999998 0.10000001 0.10000002 0.09999999 0.09999999 0.10000002
 0.09999999 0.10000002 0.09999999 0.10000001]
 Epsilon 0.2   : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0.20000003 0.2        0.20000002 0.19999999 0.20000002 0.2
 0.2        0.20000003 0.19999999 0.20000002]
 Epsilon 0.3   : Accuracy after attack: 90.0 %, Success: [False False False  True False False False False False False], Perturbation size: [0.3        0.3        0.30000004 0.3        0.3        0.3
 0.29999998 0.30000004 0.3        0.3       ]
 Epsilon 0.4   : Accuracy after attack: 90.0 %, Success: [False False False  True False False False False False False], Perturbation size: [0.40000004 0.4        0.40000007 0.4        0.40000004 0.40000004
 0.4        0.40000004 0.4        0.40000004]
 Epsilon 0.5   : Accuracy after attack: 80.0 %, Success: [False False False  True False  True False False False False], Perturbation size: [0.5        0.5        0.5        0.5        0.49999997 0.5
 0.5        0.50000006 0.5        0.5       ]
 Epsilon 0.9   : Accuracy after attack: 80.0 %, Success: [False False False  True False  True False False False False], Perturbation size: [0.9        0.9        0.90000004 0.7435034  0.9        0.79393053
 0.8999999  0.9        0.9        0.9000001 ]
 Epsilon 1     : Accuracy after attack: 70.0 %, Success: [False False False  True False  True False False False  True], Perturbation size: [1.         1.         1.0000001  0.7435034  1.         0.79393053
 1.         1.         1.         1.        ]
 Epsilon 2     : Accuracy after attack:  0.0 %, Success: [ True  True  True  True  True  True  True  True  True  True], Perturbation size: [1.5579354  1.5017346  2.0000002  0.7435034  1.5372422  0.79393053
 1.5693114  2.0000002  1.9999999  1.616929  ]
 Epsilon 5     : Accuracy after attack:  0.0 %, Success: [ True  True  True  True  True  True  True  True  True  True], Perturbation size: [1.5579354  1.5017346  2.124325   0.7435034  1.5372422  0.79393053
 1.5693114  2.263817   2.3044581  1.616929  ]
 Epsilon 7     : Accuracy after attack:  0.0 %, Success: [ True  True  True  True  True  True  True  True  True  True], Perturbation size: [1.5579354  1.5017346  2.124325   0.7435034  1.5372422  0.79393053
 1.5693114  2.263817   2.3044581  1.616929  ]
 Epsilon 9     : Accuracy after attack:  0.0 %, Success: [ True  True  True  True  True  True  True  True  True  True], Perturbation size: [1.5579354  1.5017346  2.124325   0.7435034  1.5372422  0.79393053
 1.5693114  2.263817   2.3044581  1.616929  ]
```

Figure 2: Accuracy L2FMNAttack's Result

Attack Visualization

Visualization of the attack results was done using Matplotlib. For each attack, a different grid was generated with various epsilon values, illustrating the degree of perturbation introduced to the samples. The plots obtained are displayed below.

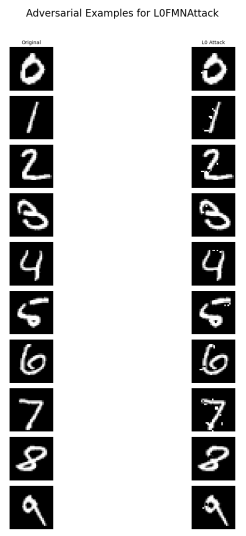


Figure 3: Adversarial Grid for L0FMNAttack

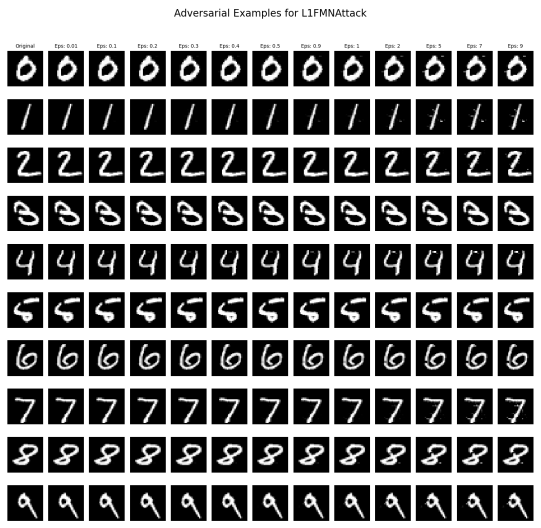


Figure 4: Adversarial Grid for L1FMNAttack

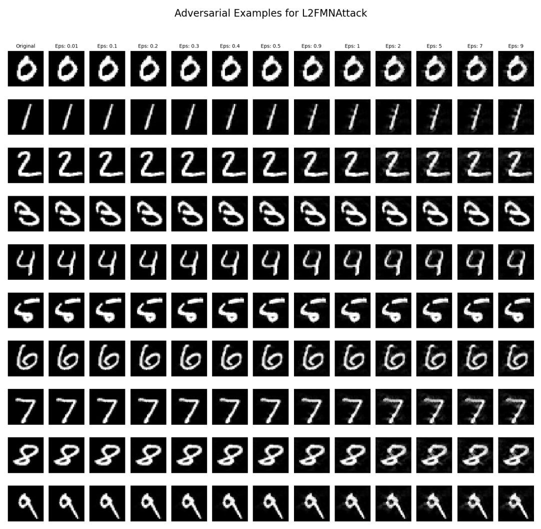


Figure 5: Adversarial Grid for L2FMNAttack

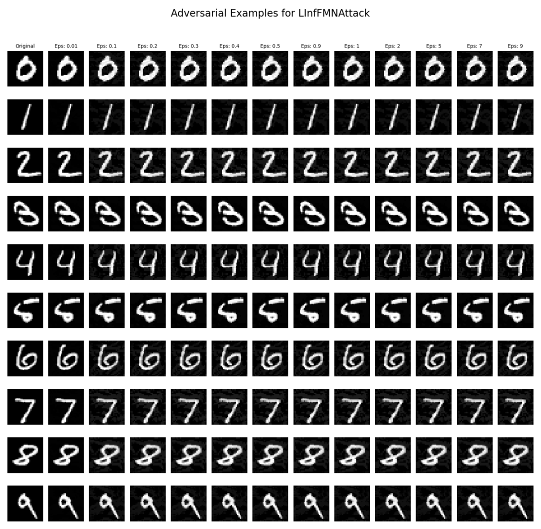


Figure 6: Adversarial Grid for LInfFMNAttack

3 Additional Attack

This section presents an additional attack added to the `adversarial.py` script to explore its performance in generating adversarial samples.

Selection of a New Attack

Adversarial Machine Learning is an evolving field, and finding suitable attacks for specific use cases requires an in-depth understanding of the domain. Given the exploratory nature of this project, the `L2ClippingAwareRepeatedAdditiveGaussianNoiseAttack` was chosen for its novelty and the author's interest in Gaussian-based methods. This attack, while less commonly discussed compared to Fast Minimum Norm (FMN) attacks, offers a unique approach by iteratively applying Gaussian noise under L2 constraints.

L2ClippingAwareRepeatedAdditiveGaussianNoiseAttack

Thanks to the class-based design of the implementation, integrating the `L2ClippingAwareRepeatedAddi` was straightforward. The attack was added to the list of attacks in the main execution code without requiring any structural changes, ensuring that the Open-Closed Principle is maintained.

This attack applies Gaussian noise repeatedly to the input image while ensuring that the perturbation remains within a predefined L2 bound. The algorithm works by progressively adding noise and clipping the result to maintain the desired L2 norm. This method is particularly effective in generating smooth perturbations that are difficult for human observers to notice, yet powerful enough to fool classifiers.

Attack Visualization

Below is the adversarial grid generated by applying the `L2ClippingAwareRepeatedAdditiveGaussianN` This grid visually demonstrates how the perturbations modify the original images to create adversarial samples.

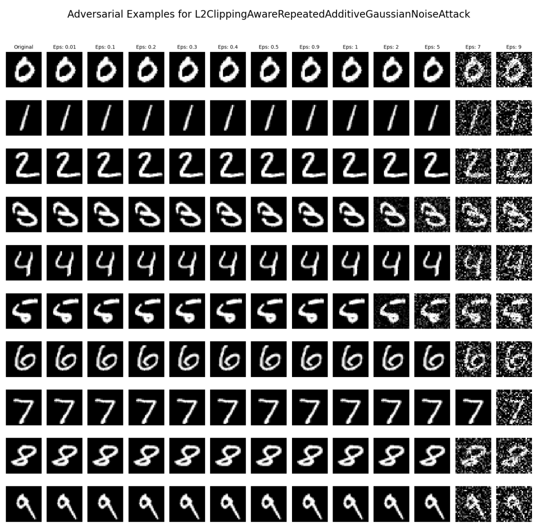
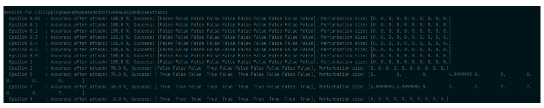


Figure 7: Adversarial Grid for L2 Clipping Aware Repeated Additive Gaussian Noise Attack



```
Results for L2ClippingAwareRepeatedAdditiveGaussianNoiseAttack:
  Epsilon 0.01  : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  Epsilon 0.1   : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  Epsilon 0.2   : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  Epsilon 0.3   : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  Epsilon 0.4   : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  Epsilon 0.5   : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  Epsilon 0.9   : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  Epsilon 1     : Accuracy after attack: 100.0 %, Success: [False False False False False False False False False False], Perturbation size: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  Epsilon 2     : Accuracy after attack: 90.0 %, Success: [False False False  True False False False False False False], Perturbation size: [0. 0. 0. 2. 0. 0. 0. 0. 0. 0.]
  Epsilon 5     : Accuracy after attack: 70.0 %, Success: [ True False False  True False  True False False False False], Perturbation size: [5.        0.        0.        4.9999995 0.        5.        0.
 0.        0.        0.        ]
  Epsilon 7     : Accuracy after attack: 30.0 %, Success: [ True  True False  True  True  True  True False False  True], Perturbation size: [6.9999995 6.9999995 0.        7.        7.        7.        7.
 0.        0.        7.        ]
  Epsilon 9     : Accuracy after attack:  0.0 %, Success: [ True  True  True  True  True  True  True  True  True  True], Perturbation size: [9. 9. 9. 9. 9. 9. 9. 9. 9. 9.]
```

Figure 8: L2 Clipping Aware Repeated Additive Gaussian Noise Attack's Result

Conclusion

This assignment explored the use of adversarial attacks in the context of image classification on the MNIST dataset. A Convolutional Neural Network (CNN) was trained to achieve high classification accuracy, and adversarial samples were generated using several Fast Minimum Norm (FMN) attacks, including L0, L1, L2, and LInf attacks, to assess the model's vulnerability.

The class-based approach adopted for the implementation enhanced modularity, allowing the addition of new attacks with minimal changes. In addition to the FMN attacks, the `L2ClippingAwareRepeatedAdditiveGaussianNoiseAttack` was introduced, offering a unique perspective on adversarial attacks by iteratively adding Gaussian noise while maintaining L2 constraints.

The experiments revealed the model's susceptibility to adversarial perturbations, underscoring the importance of adversarial robustness in machine learning systems. The results demonstrated how different attack strategies and epsilon values affect the effectiveness of adversarial samples.