Assignment 5
Adversarial Machine Learning

T-710-MLCS, Machine Learning in Cyber Security

Reykjavik University - School of Computer Science

Federico Maria Cruciani

`federico24@ru.is`

27. October 2024

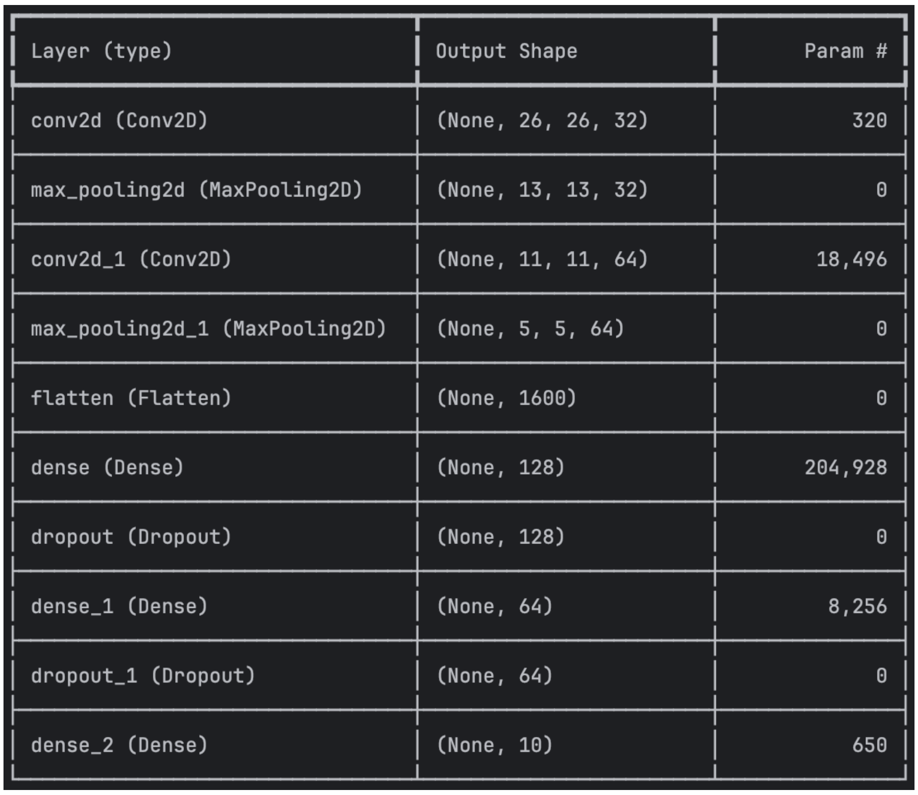
1 Introduction

This report covers the work done for Assignment 5 of the Machine Learning in Cyber Security course. The objective of the assignment was to experiment with the generation of adversarial samples using different types of attack. Experiments were performed on the MNIST dataset of handwritten digits using a simple CNN model.

2 Simple Data Preprocessing and Splitting

The file **train.py** handles training the CNN model, the code was obtained from a Kaggle notebook[1]. It has the structure shown in figure 1. The classifier obtains an accuracy of 99%.

The trained model is also saved in the file **mnist_cnn.keras** to be used for the adversarial attacks.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204,928
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650

Figure 1: The structure of the CNN

Before training the model, the input data went under two preprocessing steps, done in the **preprocessing.py** file:

- Normalisation
- Reshaping into a 28x28 pixel matrix with grey scale values

Also the labels were transformed into one-hot encoded vectors.

3 Generate Adversarial Samples

The generation of the adversarial samples is done in the file **adversary.py**.

[1]`https://www.kaggle.com/code/imdevskp/digits-mnist-classification-using-cnn`

First, the model and input images are transformed into data types that Foolbox can use, then the different variants of the Fast Minimum Norm attack are performed against the model by running the `perform_attack()` function, once for each of the ten classes which are given one at a time to the attack as a misclassification target. The resulting images are shown in figures 2-5.

The attacks were performed with epsilon values $[0.004, 0.01, 0.1, 1.0, 10.0, 30, 100]$ except for the L0 variant which doesn't support setting epsilons, so `None` was used in that case. The code to calculate the accuracy for each value was taken and adapted from the Python script provided in class.

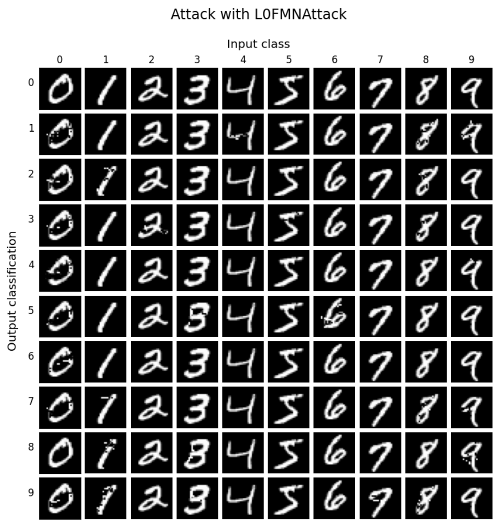


Figure 2: Results for L0FMNAttack

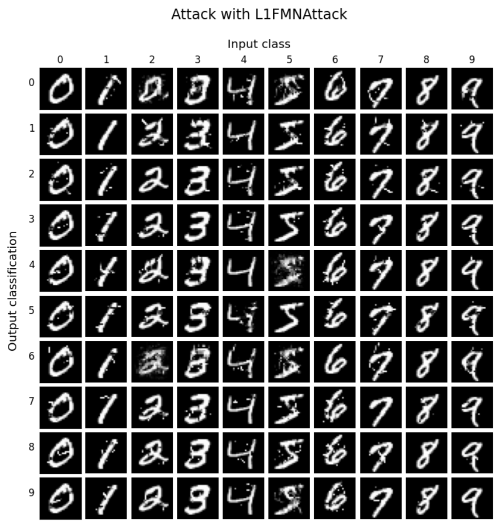


Figure 3: Results for L1FMNAttack

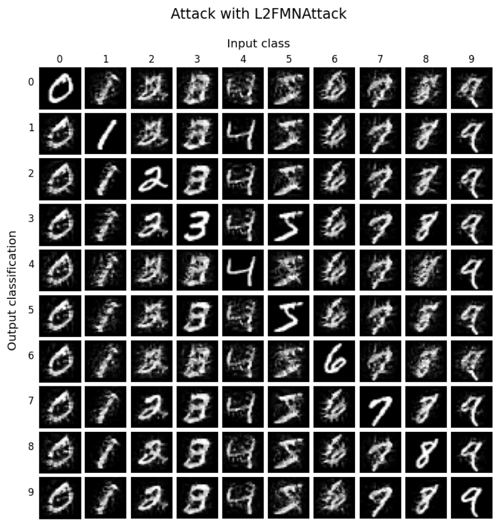


Figure 4: Results for L2FMNAttack

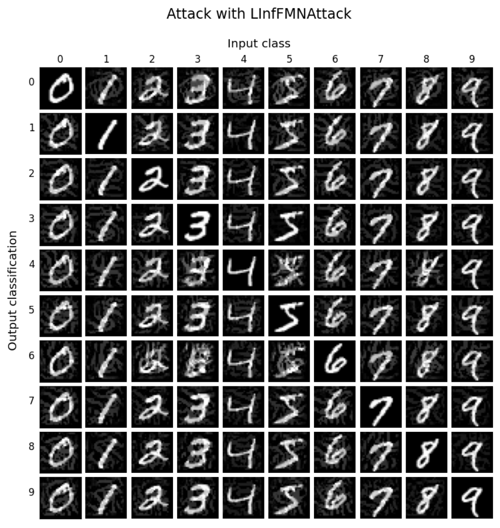
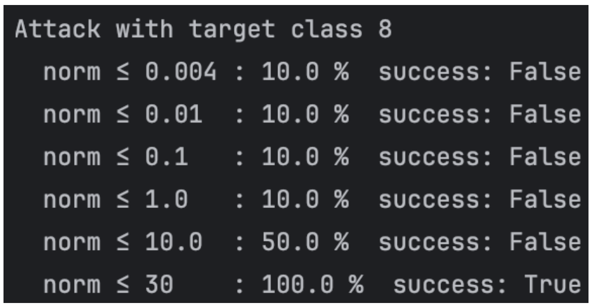


Figure 5: Results for LInfFMNAttack

3.1 Observations

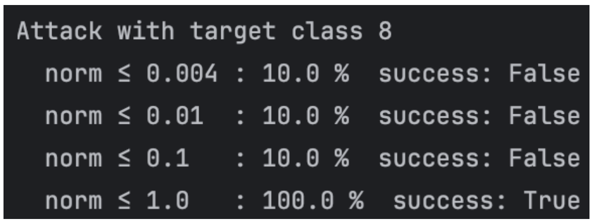
As seen in the example console outcomes shown in figure 6, as the norm goes "higher" the attacks need a smaller epsilon to achieve success on all labels, with LInf already reaching 100% success with epsilon equal to 1.0. This, however, doesn't mean that the images produced have less visible perturbations, in fact the opposite is true, seeing how some numbers are almost unrecognisable.

The L0 attack didn't always manage to create effective adversarial samples, only reaching a maximum of six successful samples for the class 9.



```
Attack with target class 8
  norm ≤ 0.004 : 10.0 %  success: False
  norm ≤ 0.01  : 10.0 %  success: False
  norm ≤ 0.1   : 10.0 %  success: False
  norm ≤ 1.0   : 10.0 %  success: False
  norm ≤ 10.0  : 50.0 %  success: False
  norm ≤ 30    : 100.0 %  success: True
```

(a) L1FMNAttack

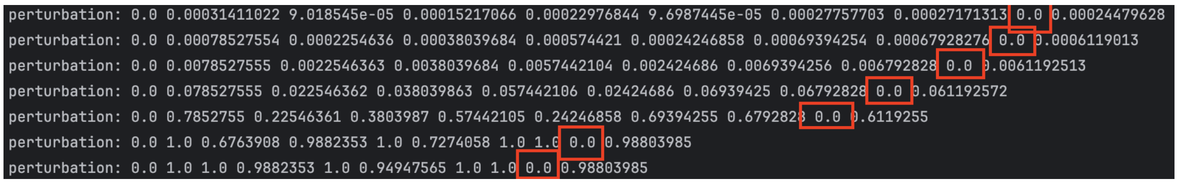


```
Attack with target class 8
  norm ≤ 0.004 : 10.0 %  success: False
  norm ≤ 0.01  : 10.0 %  success: False
  norm ≤ 0.1   : 10.0 %  success: False
  norm ≤ 1.0   : 100.0 %  success: True
```

(b) LInfFMNAttack

Figure 6: Example outcomes for different epsilon values

Moreover, for some of the target labels (0, 4 and 6) the L1 attack failed to generate samples for all the images. In particular, it produced no perturbation at all for the images that have the true label 8, as seen in figure 7. In fact the images in figure 3 with input class 8 and output classification 0, 4 and 6 are identical to the true 8 image in the diagonal.



```
perturbation: 0.0 0.00031411022 9.018545e-05 0.00015217066 0.00022976844 9.6987445e-05 0.00027757703 0.00027171313 0.0 0.00024479628
perturbation: 0.0 0.00078527554 0.0002254636 0.00038039684 0.000574421 0.00024246858 0.00069394254 0.00067928276 0.0 0.0006119013
perturbation: 0.0 0.0078527555 0.0022546363 0.0038039684 0.0057442104 0.002424686 0.0069394256 0.006792828 0.0 0.0061192513
perturbation: 0.0 0.078527555 0.022546362 0.038039863 0.057442106 0.02424686 0.06939425 0.06792828 0.0 0.061192572
perturbation: 0.0 0.7852755 0.22546361 0.3803987 0.57442105 0.24246858 0.69394255 0.6792828 0.0 0.6119255
perturbation: 0.0 1.0 0.6763908 0.9882353 1.0 0.7274058 1.0 1.0 0.0 0.98803985
perturbation: 0.0 1.0 1.0 0.9882353 1.0 0.94947565 1.0 1.0 0.0 0.98803985
```

Figure 7: Example perturbation values for L1FMNAttack with target label 0, in red the values showing no perturbation