Università degli Studi di Padova

DIPARTIMENTO MATEMATICA

Università degli Studi di Padova

Master Degree in Cybersecurity

Course: Digital Forensics and Biometrics

Programming Project #2:

Fooling Synth Detector: Can you spot this?

Report and Project:
Nicola Busato 2119291
Jacopo Momesso 2123874

Academic year 2023/2024

Contents

1 Aim of the project

The main aim of this project is to develop one or more strategies that can fool a synthetic image detector. To achieve this, we will use adversarial techniques to exploit the vulnerabilities of the machine learning model. By doing so, we aim to manipulate the output of the detector and cause it to misidentify synthetic images as real.

2 Materials

The materials used are:

- **Google Colab**: used to program our strategies to fooling the synth detector
- **Cycle-GAN**: GAN used to generate our fake images
- **Horse2zebra Dataset**: dataset used by the GAN to turn horses into zebras
- **GAN Image Detector** software: the synth detector we wanted to fool
- **AI or Not** online detector: used to understand how to proceed in order to fool a synth detector
- **Laboratory Experience number 2**: Adversarial Attack

3 Methods

For this project, three distinct methods to evade detection were used:

- **Adversarial noise**: The first method involves the addition of adversarial noise, a type of disturbance designed to fool detection, which refers to carefully crafted perturbations or alterations strategically applied to input data. The primary goal is to exploit vulnerabilities in the model's decision boundaries, tricking it into making incorrect predictions.
  In our project, we added adversarial noise using the Fast Gradient Sign Method (FGSM). The FGSM generates an adversarial example by utilizing the gradients of the neural network. It creates a new image that maximizes the loss by using the gradients of the loss with according to the input image. The process can be summarized as follows:

  $$\tilde{x}_k = x_k + \epsilon \cdot sing(\Delta_x J(\Theta, x_k, y_k))$$

  where:

  - $\tilde{x}$: Adversarial image
  - x : Original input image
  - y: Original input label
  - ϵ : Multiplier to ensure the perturbations are small
  - Θ : Model parameters
  - J : Loss

- **Random noise**: A naive approach of adversarial noise, consisted of simply adding random noise, similar to adversarial noise but without a criterion for efficient addition. Unlike adversarial noise, which is crafted to mislead the model, random noise is introduced without a targeted strategy, making it less effective for deceiving the detector.
- **Image manipulating**: The last method involves manipulating the image by introducing rotation, resizing, and distortion. This approach aims to confuse the detector by altering the appearance of the original image.

3.1 Implementation

We downloaded from GitHub the project (GAN Image Detector) and its relative weights, which we are going to use as a detector to detect real or fake images.
Then we downloaded the CycleGAN from GitHub, which we are goingo to use, after pre-training it, to generate fake images using the horse2zebra dataset imported from Kaggle.
Once all this was downloaded, we used the CycleGAN to generate the fake images using the *trainA* and *testA* of the horse2zebra dataset and then saved them in the appropriate directories. (Using the CycleGAN we generated 1187 images).

In order to make the code clearer and simpler, we have defined some helper functions for the operations we want to perform on the generated images. The main helper functions are:

- *adversarialNoise()*: generates adversarial noise (using the function *createAdversarialPattern()*) for a given image and saves the perturbed images with varying epsilon values (using the function *saveAsImg()*)
- *randomNoise()*: adds random gaussian noise to a given image and saves it with varying noise scale values (using the function *saveAsImg()*)
- *imageManipulation()*: applies image manipulation techniques and saves the manipulated images (using the function *saveAsImg()*)
- *processImages()*: processes images according to the specified method passed as a parameter. There are three type of methods:
  - Adversarail Noise (section 3)
  - Random Noise (section 3)
  - Image Manipulation (section 3)
- *RunGANimageDetection()*: runs the previously downloaded GANimage Detector on images in a specified directory and saves the output in a .csv file

Once we had defined the helper functions, we created a subset of the generated images (15% of the generated images) and we used it to create our deceptive images with different strategies that we previously explained.
Firstly, we generated images with adversarial noise using various epsilon values (epsilon from 0 to 0.1 in steps of 0.005), then we generated images with random noise (with different noise level from 0 to 100 in steps of 5), then we generated images with manipulation and finally we combined the three different

strategies. We decided not to apply Gaussian and random noise together, as they are both types of noise and this operation results in a sum of noise. So it will be more convenient to increase epsilon instead of using two types of noise.

After modifying the images, we first ran the detector on the unmodified images and printed the results, and then ran the detector on the modified images and printed the results.
Based on the results obtained and plotted, we selected the best strategy and then applied it to the entire dataset of the generated images (for more details, see section 4.3).

3.2 Altered Images

These strategies allowed us to obtain different variants of the same image, 44 different versions are created per image, these are some of those obtained with the various strategies.

Some images obtained by Adversarial Noise with different epsilon:

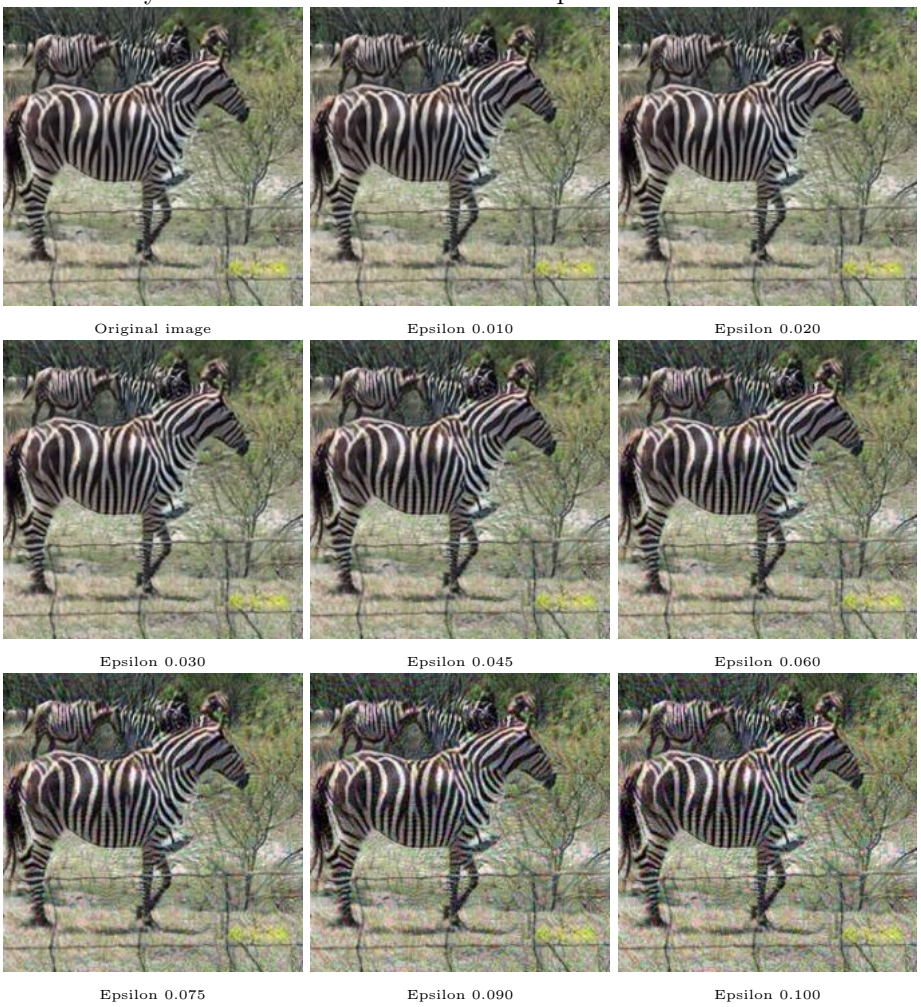

Figure 1: Some images with different adversarial noise level (epsilon).

Some images obtained by Random Noise with different epsilon:

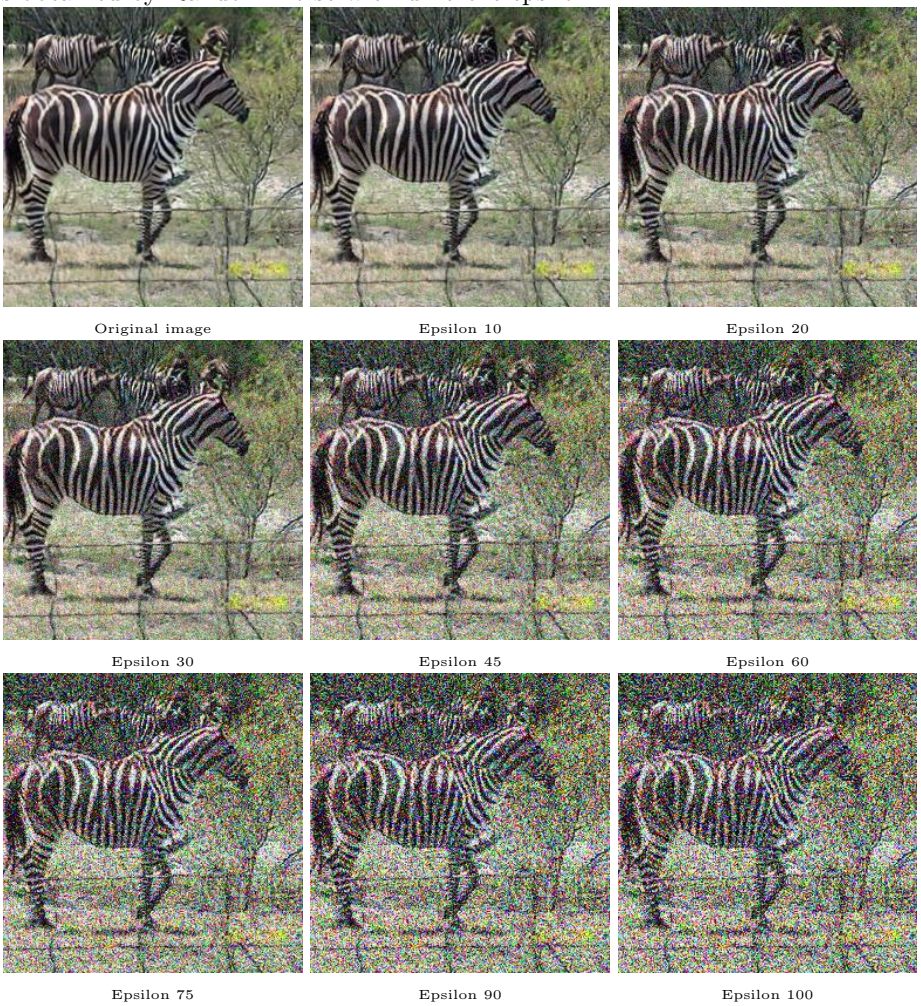

Figure 2: Some images with different random noise level (epsilon).

Images obtained by Manipulation and Combine strategies:

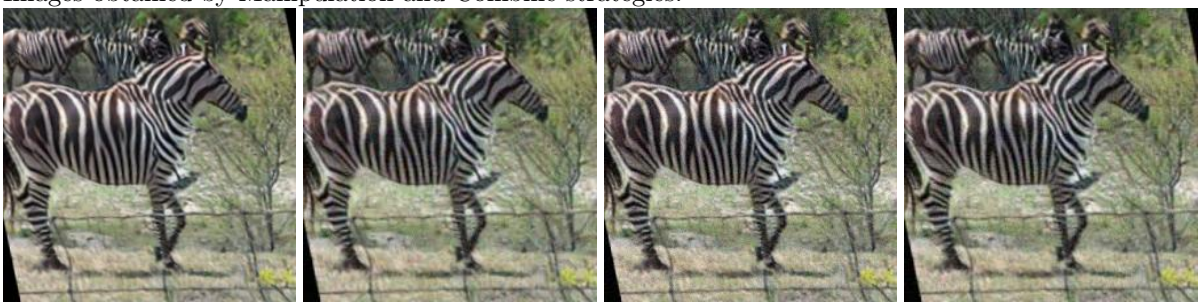

Figure 3: Some images with different strategies.

4 Results

4.1 Detector output

The GAN Image Detector provides classification scores indicating whether an image is categorized as *Real* or *Fake*. The detector returns a CSV file containing the classification scores for each image.

Filename	Confidence	Classified
0001.jpg	6.001	Fake
0002.jpg	-1.659	Real
0003.jpg	-4.019	Real
...	...	...
0176.jpg	5.195	Fake
0177.jpg	2.144	Fake
0178.jpg	-0.335	Real

Table 1: Example of GAN detector output

The table [1] is divided into 3 columns

- **Filename**: Analyzed image
- **Confidence**: Indicates the degree to which the detector perceives the image as *Real* or *Fake*.
- **Classified**: A customized element added to facilitate data interpretation, it's *Real* if the *Confidence* is ≥ 0; otherwise, it's considered *Fake*.

The value *Confidence*, in the Table 1, represents how much the detector thinks that the image is *Real* or *Fake*. A higher value indicates a stronger belief that the image is Fake, while a lower value suggests a stronger conviction that the image is Real.

From this table we extrapolate also some other information:

```
Element misclassified: 12 (6.74%)
MEAN: 4.3934; MIN: -1.842; MAX: 11.862
```

Although the images are all Fake and not yet edited by us, some of them are already considered Real

4.2 First staggering of the data

To begin, we chose a subset of images, specifically 178 out of 1187 images, which accounts for 15% of the dataset. For each image, we applied various alteration strategies with different strengths. This included 20 epsilon values for both adversarial and random noise techniques, as well as manipulation of the image through scratch, rotation and shearing. Combinations of different strategies were also used to further manipulate the images.
After processing the data obtained from the detector, we demonstrated through graphs how the images are classified as the parameters change or different strategies are adopted.

Adversarial Noise:

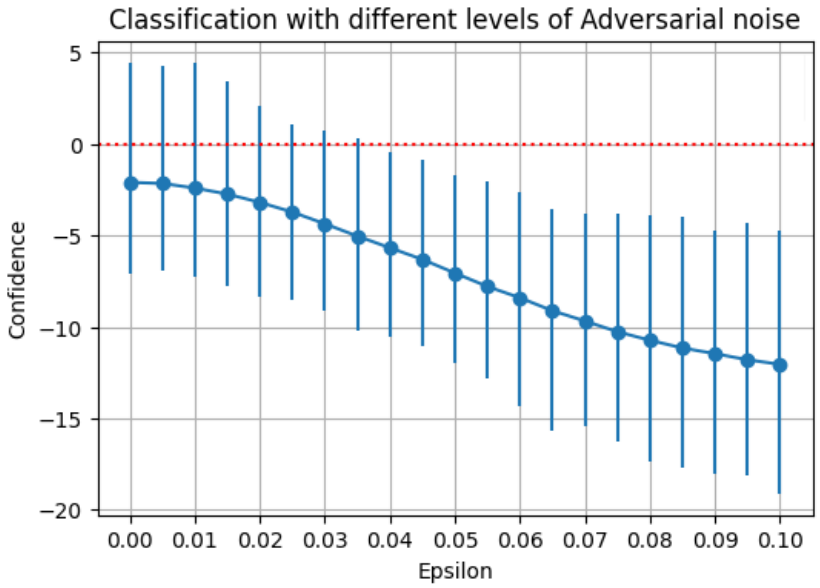

Figure 4: Plot of the image classification using Adversarial Noise with different Epsilon values.

The blue dots in the graph represent the average classification of each image altered using the same strategy. The vertical lines represent the spread, indicating the maximum and minimum classification for each parameter.
In this case (graph [4]), increasing epsilon increases also the confidence that the image is classified as *Real*.

After analysing the graph [4], we have decided to exclude epsilon values that are less or equal to 0.035, as some elements were classified as *Fake*. Additionally, values between 0.04 and 0.05 were too close to zero, making them prone to misclassification as *Fake*. We observed that the maximum confidence of the images did not decrease significantly when considering epsilon values that are greater than or equal to 0.07. Therefore, we narrowed down our range to values between 0.05 and 0.07. We've chosen 0.06 as the epsilon value because it represents a good compromise between misclassifications and noise reduction.

Random Noise:

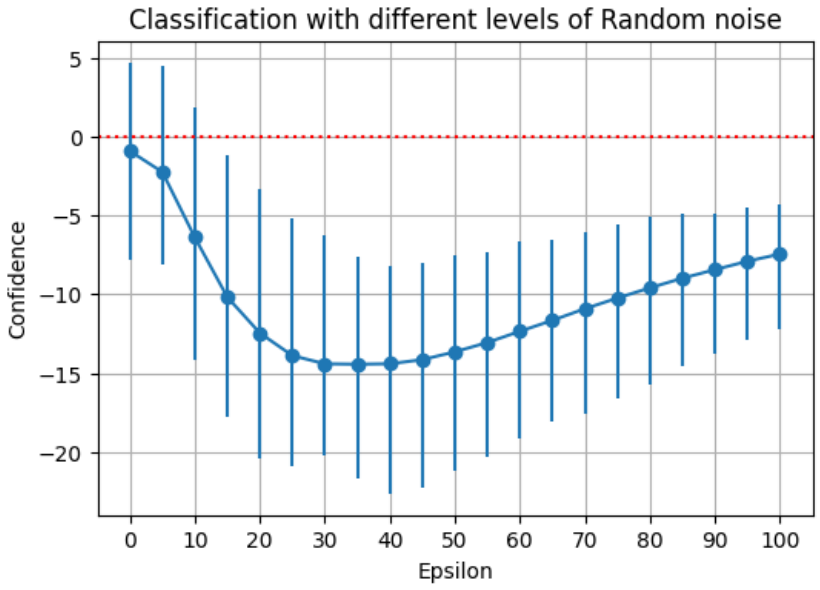

Figure 5: Plot of the image classification using Random Noise with different Epsilon values.

The image classification is affected by the intensity of the noise. It's interesting to see that the classification as real increases with the noise intensity, reaching a peak at epsilon = 35, before decreasing and gradually increasing again.

After analysing the graph [5], we decided to exclude epsilon values below 10 as they resulted in misclassification of some images as fake. Values exceeding 20 introduced excessive noise, making the images less discernible. Therefore, we selected a range of epsilon between 15 and 20, ultimately deciding to keep 17, as it strikes a balance between the epsilon value and noise level.

Manipulation and combined strategies:

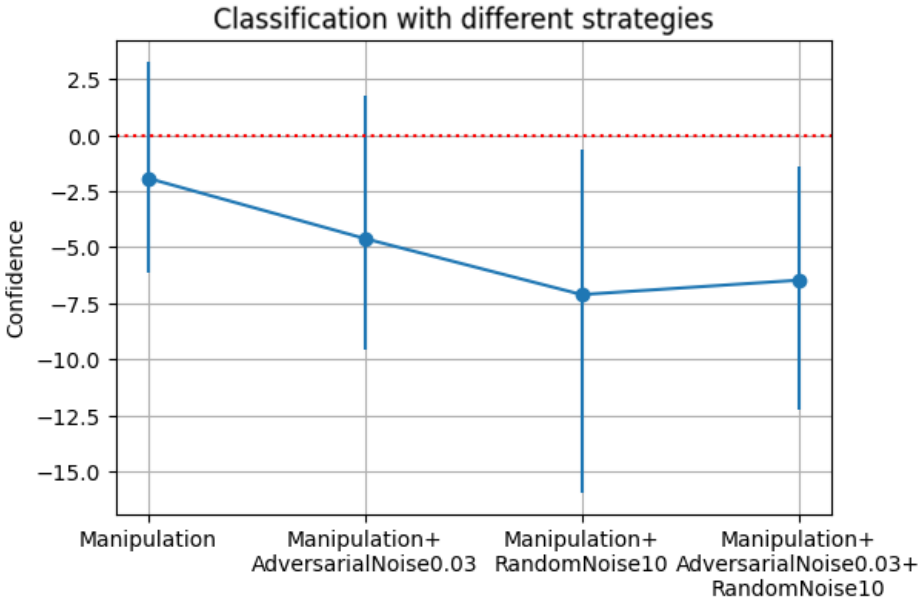

Figure 6: Plot of the image classification using Manipulation and the combinations of the strategies.

The third graph [6] shows the output of the Discriminator for the Manipulation images and for the combinations of different strategies.
In this case, we decided to discard Manipulation and Manipulation+AdversarialNoise because some images were misclassified, and we discarded Manipulation+RandomNoise because we had too much errors in the classification, so we kept Manipulation+AdversarialNoise+RandomNoise as a strategy.

4.3 Select of best strategy

Now there were only three strategies to compute, instead of 44. This allowed us to increase the subset size, for a more precise analysis. At this point we used 712 images, which were 60% of the total.

The following strategies were considered to determine the optimal one:

- **Adversarial Noise** with epsilon 0.06
- **Random Noise** with epsilon 17
- **Manipulation + Adversarial Noise 0.04 + Random Noise 10**, the adversarial epsilon was increased from 0.03 to 0.04.

From these three strategies, we obtained the following graph:

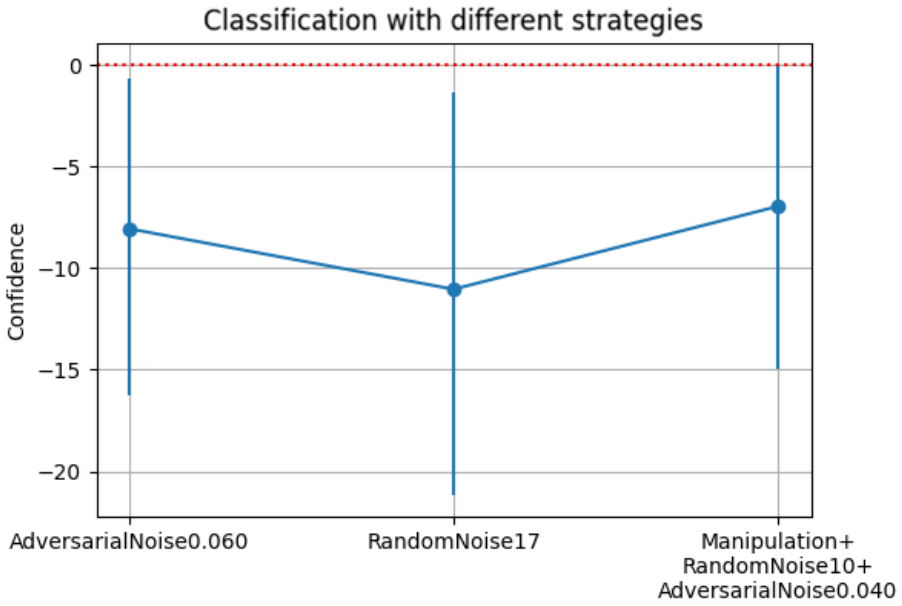

Figure 7: Plot of the image classification using the selected strategies.

Images obtained with the three best strategies:

Adverarial Noise 0.060 | Random Noise 17 | Manipulated, RndNoise 10 and AdvNoise 0.040

Figure 8: Images form the best strategies that we are considered.

After analysing the graph [7], we have determined that our optimal strategy was to use the Adversarial Noise, but we decided to increase epsilon from 0.06 to 0.065 since 0.06 is only slightly lower than zero. Although the Random Noise approach results in a lower average *Confidence*, it introduces excessive variance in the classification values. The Random Noise approach's maximum value is only slightly lower than Adversarial Noise's maximum value. However, the combined strategy involving Manipulation, Adversarial Noise, and Random Noise shows promising misclassification rates and comparable spread to Adversarial Noise alone. Nevertheless, it slightly increases the probability of classifying images as fake. Although this combined strategy could potentially be improved by adjusting epsilon values, we have decided against pursuing it further due to its complexity compared to the straightforward implementation of Adversarial Noise.

4.4 Best strategy

As the best strategy we chose Adversarial Noise with epsilon 0.065, so we computed the alteration for all 1187 images. It is important to note that the displayed results are based solely on outcomes obtained during the specific session. Program results may vary across different executions.
The detector output obtained was:

Filename	Confidence	Classified
0001.jpg	-11.162	Real
0002.jpg	-5.871	Real
0003.jpg	-10.552	Real
...	...	...
1185.jpg	-7.917	Real
1186.jpg	-8.980	Real
1187.jpg	-10.942	Real

Table 2: GAN detector output from the best strategy

```
Element misclassified: 1187 (100.00%)
Mean: -8.780, Min: -18.507, Max: -0.129
```

Images with the lowest and highest confidence levels:

```
0193.jpg   -0.128804   Real
0833.jpg   -18.506683  Real
```

Img 0193.jpg without alteration Img 0193.jpg with Adversrial Noise 0.065

Figure 9: Image with the minimum confidence, before and after alteration

Img 0833.jpg without alteration Img 0833.jpg with Adversrial Noise 0.065

Figure 10: Image with the maximum confidence, before and after alteration

By using **adversarial noise** with an epsilon parameter set to **0.065**, we were able to manipulate our dataset to produce false results. This demonstrates the efficacy of adversarial techniques, leading to a complete misclassification of the dataset's samples.

5 Possible Future Improvements

One of the future improvements that could be adopted to increase the usefulness and efficiency of the fooling, could be to develop an automated selection mechanisms that determines the optimal strategy, based on the characteristics of the dataset. Currently, the most suitable strategy is often chosen through manual intervention or by implementing predetermined rules. However, machine learning algorithms or optimization techniques could be used to design systems capable of autonomously identifying the most effective approach for a given dataset. This could result in more efficient and accurate outcomes. The system will adjust its strategy according to the specific characteristics and requirements of each dataset. These improvements will simplify the process and improve the system's overall performance and effectiveness in various data scenarios.

For this project we used the Fast Gradient Sign Method (FGSM), but other adversarial strategies that could be used include: DeepFool, Carlini-Wagner (CW) Attack, PGD (Projected Gradient Descent) Attack, and JSMA (Jacobian-based Saliency Map Attack). Each technique has its advantages and limitations. Integrating multiple strategies could lead to an easier to fool Detector.

A further approach could be to merge multiple datasets to identify generalisable parameters. By pooling information from diverse datasets, common trends and patterns across various data domains could be extracted. This aggregation aims to derive parameters with broader applicability and reduce the risk of bias. Such an approach enhances the scalability and versatility of parameter settings without the need for a learning system.