University of Salerno

Department of Computer Engineering, Electrical Engineering and Applied Mathematics

Artificial Intelligence for Cybersecurity Project's Report

Group Members:

Iovaro Damiana	0622702017
Massaro Sara	0622702015
Nocerino Antonio	0622701912
Trotta Prisco	0622702014

Lecturer:
Prof. Greco Antonio

2023/2024

Contents

1 Project Goal

This report focuses on evaluating the security of a facial recognition system based on artificial neural networks. The main goal is to examine the robustness of this access control system against adversarial attacks and determine how effective the implemented defenses are in countering them.

The report will provide a detailed analysis of the performance evaluation procedures, including the generation of adversarial attacks, the creation and assessment of the implemented defenses, and thorough comments on the experimental results obtained.

2 Neural Networks and Test Set

2.1 NN1

The model being attacked is InceptionResnetV1 (referred to as NN1 from now on), pre-trained on the VGG-Face2 dataset (explained in later paragraphs). The model was originally trained in TensorFlow and then converted to PyTorch.

The model was trained on 8631 images.

The optimizer used is Adam, and the loss function is a custom function called Triplet Loss.

- `resnet` = InceptionResnetV1
- `loss` = Triplet Margin Loss
- `input_shape` = (3, 160, 160)
- `nb_classes` = 8631
- `optimizer` = Adam
- `clip_values` = (0, 1)

2.2 NN2

The model chosen to evaluate the transferability of attacks is a Senet50 model (referred to as NN2 from now on), also pre-trained on the VGG-Face2 dataset. In this case, the model is converted from a Caffe model.

The model was trained on 8631 images.

2.3 Creating the Test Set for Generating Attacks on NN1 and NN2

The data used in the subsequent steps is extracted from the VGG-Face2 dataset developed by the University of Oxford. The dataset contains images of 9,131 individuals, totaling about 3.31 million facial images. The images in the dataset exhibit a wide range of variations, including changes in pose, facial expressions, lighting conditions, age, and occlusions. This variety makes the dataset particularly useful for training robust and generalizable models.

Initially, 100 identities are selected from the training set of the VGG-Face2 dataset, and a test set is built with 1,000 images, 10 for each identity. This test set will be used to evaluate the accuracy of the face recognition network NN1 and NN2.

The process of creating the test set involves reviewing the entire dataset used for training VGG-Face2, which is cleaned of any improperly formatted rows.

At this point, 100 identities are randomly selected, using seed 42.

The images associated with the selected identities are extracted from the original dataset. To match the preprocessing performed on the training data for NN1 and NN2, the test set images are processed with the MTCNN module: in each image, the face is detected, cropped, and then resized to 160x160 pixels for NN1 and 224x224 pixels for NN2. NN1 uses the pixel in range (0,1), so the values are normalized. NN2 uses values ranging from 0 to 255 for each channel. However, the network subtracts the mean from the values of each channel BGR, which helps reduce dependency on brightness.

2.4 Evaluation on clean data

The first step after selecting the 2 networks and the test set is to evaluate the accuracy of the networks on the clean data. This evaluation is done with the understanding that the test set was derived from the samples used during the training phase of the models.

2.4.1 NN1 Accuracy

The accuracy value is 0.83. This result highlights the network's ability to correctly match a person's name to their face. This accuracy score also suggests that there is still room for improvement, especially in more challenging cases where recognition could be harder due to poor lighting conditions, significant changes in face angle, or lower image quality.

2.4.2 NN2 Accuracy

The accuracy value is 0.88. This result shows how NN2, even on clean samples, performs better than NN1. NN2's superiority in performance suggests it's better at correctly classifying unaltered samples, which sets a strong foundation for further evaluations in more complex scenarios.

3 Generation of Adversarial Samples

Once the test set is created and the network's performance on it is evaluated, the next step is to generate adversarial samples: adversarial samples (or adversarial examples) are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake. These inputs are typically created by making small, often imperceptible modifications to legitimate inputs, which result in the model producing incorrect or unintended outputs.

3.1 Error Generic vs Error Specific

Adversarial attacks can be categorized into two main types: error generic and error specific.

Error generic attacks aim to cause the model to make any incorrect prediction, without specifying which incorrect class the model should choose. The primary goal is simply to push the model's output away from the correct label.

On the other hand, error specific attacks are designed to mislead the model into predicting a specific incorrect class. In these attacks, the adversary selects a target class, and the adversarial sample is crafted to maximize the likelihood that the model will classify the input as this target class, rather than its true class.

Both types of attacks highlight the vulnerabilities of machine learning models, but error specific attacks demonstrate a more precise and controlled exploitation of these weaknesses.

The target class chosen for generating the attacks is class 0, which is a class not present in the test set used for generating the attacks.

The generated adversarial examples will be used to assess the impact of error-generic and error-specific attacks on NN1 and to see how well these attacks transfer to another classifier (NN2) trained on the same dataset.

Note that in this report we will refer to the error generic attack as the untargeted attack, and we will refer to the error specific attack as the targeted attack based on the operation of the ART library.

3.2 Security Evaluation Curve

To analyze the model's robustness against attacks, we used the Security Evaluation Curve. This curve aims to represent the model's robustness, with accuracy on the y-axis and attack strength on the x-axis.

In our experiment, we employed two types of security curves: one for untargeted attacks and another for targeted attacks.

Security Evaluation Curve for Untargeted Attacks: The y-axis represents the curve describing the attack's effectiveness, where points indicate the ratio of samples correctly labeled by the system to the total number of samples for each attack strength on the x-axis.

Formally:

- Let $D = \{(x_i, y_i)\}_{i=0}^{N-1}$ be the dataset with N samples.
- x_i is a sample and y_i is its ground truth label.
- pred(x_i) is the model's prediction for x_i.

The system's accuracy acc_{sys} is:

$$\text{acc}_{\text{sys}} = \frac{\sum_{i=0}^{N-1}(\text{pred}(x_i) = y_i)}{N}$$

We expect acc_{sys} to decrease as the attack strength increases, indicating the effectiveness of the untargeted attack.

Security Evaluation Curve for Targeted Attacks: In addition to acc_{sys}, the y-axis shows the curve describing the attack's effectiveness in inducing misclassification to the specific target class. Points represent the ratio of misclassified samples (predicted as the target class while the true class is not the target) to the total number of misclassified samples for each attack strength on the x-axis.

Formally:

- Let $D = \{(x_i, y_i)\}_{i=0}^{N-1}$ be the dataset with N samples.
- x_i is a sample and y_i is its ground truth label.
- t is the target class.

The accuracy $\text{acc}_{\text{mis\_target}}$ reflects the attack's effectiveness in misclassifying adversarial samples to the target class:

$$\text{acc}_{\text{mis\_target}} = \frac{\sum_{i=0}^{N-1}(\text{pred}(x_i) \neq y_i \wedge y_i \neq t \wedge \text{pred}(x_i) = t)}{\sum_{i=0}^{N-1}(\text{pred}(x_i) \neq y_i)}$$

We expect $\text{acc}_{\text{mis\_target}}$ to be high for effective targeted attacks, with the curve increasing as attack strength grows. Meanwhile, acc_{sys} should decrease as attack strength increases.

Since targeted attacks optimize a different cost function compared to untargeted attacks, we anticipate that acc_{sys} decreases more gradually in targeted attacks.

This analysis helps gauge how well the model resists adversarial attacks and highlights the differences between generic and targeted attack scenarios.

To perform a more in-depth analysis, within both types of security evaluation curves, the trend of perturbations made to the original images for the generation of adversarial attacks was also included.

3.3 Transferabily Analysis

The concept of transferability in adversarial attacks refers to the phenomenon where adversarial samples crafted to fool one machine learning model are also effective in deceiving other models, even if these models have different architectures or are trained on different datasets. This means that an adversarial sample designed to mislead a specific neural network can often cause other neural networks to make incorrect predictions as well.

Transferability is a significant concern because it suggests that creating robust defenses for one model may not be sufficient if similar adversarial examples can compromise other models, thus posing a broader security risk across different systems and applications.

Within this project, the NN2 network was used to evaluate the transferability of attacks carried out on the NN1 network.

3.4 Adversarial Robustness Toolbox (ART)

The ART library is used to generate adversarial samples: Adversarial Robustness Toolbox (ART) is a Python library for Machine Learning Security. ART provides tools that enable developers and researchers to evaluate, defend, certify and verify Machine Learning models and applications against the adversarial threats of Evasion, Poisoning, Extraction, and Inference.

3.5 Classifier for ART

The first step involved defining the classifier based on the characteristics of the NN1 network, which included relevant information such as input shape, the number of classes, clip value set to (0,1), the optimizer, and the loss function used. Subsequently, all attacks were defined, and adversarial samples were generated using these attacks. Since NN1 is a model trained in TensorFlow and then converted to PyTorch, the loss functions and optimizer used are not exactly the same as those used for training. Consequently, the generated attacks are of the gray-box type.

The classifier used is:

```
classifier = PyTorchClassifier(
    model=resnet,
    loss=loss,
    input_shape=input_shape,
    nb_classes=nb_classes,
    optimizer=optimizer,
    clip_values=(0, 1)
)
```

To verify the transferability of attacks from NN1 to NN2, black-box attacks are conducted as follow:

- The classifier remains the same as used for generating attacks on NN1.
- The data dimensions are adjusted to 224x224.
- During pre-processing, the clip values interval is changed to [0, 255].

3.6 Gradient Based Attacks

Machine learning and deep learning models rely on the Gradient Descent algorithm to optimize an objective function, typically the loss function.

Gradient descent is an iterative algorithm used to minimize a loss function $J(\Theta)$.

The parameter update rule for the model parameters is given by:

$$\Theta_{t+1} = \Theta_t - \alpha \cdot \nabla J(\Theta_t)$$

Where:

- Θ_0: Initial model parameters
- α: Learning rate
- $J(\Theta)$: Loss function
- $\nabla J(\Theta)$: Gradient of the loss function

While Gradient Descent optimizes by adjusting the model weights to steer input images towards the correct decision region by rearranging features, gradient-based attacks keep the model weights fixed and move the image towards a different decision region. This can be a generic region in a generic error attack or a specific region in a targeted attack.

The steps involved in a gradient-based attack are:

1. Selection of the Original Input
2. Gradient Calculation
3. Adversarial Sample Generation

Depending on the method used to generate the adversarial sample, we distinguish three types of gradient-based attacks: **Fast Gradient Sign Method, Basic Iterative Method, Projected Gradient Descent**.

3.6.1 Fast Gradient Sign Method

Attack Description **Fast Gradient Sign Method (FGSM)** is a technique used to create adversarial examples to deceive machine learning models, particularly neural networks.

The main objective of FGSM is to leverage the knowledge of the model's loss function to generate a modified input (adversarial example) that causes the model to make a misclassification error.

The algorithm is the following for each test sample x':

1. Take the original image x of the true class y.
2. Make predictions on x using the known model $h(x, w)$.
3. Compute the loss of the prediction based on the true class label y, i.e. $\mathcal{L}(h(x, w), y)$.
4. Calculate the gradients of the loss with respect to x, i.e. $\nabla_x(\mathcal{L}(h(x, w), y))$.
5. Compute the sign of the gradient, i.e. sign$(\nabla_x \mathcal{L}(h(x, w), y))$.
6. Use the signed gradient to construct the output adversarial image with a perturbation proportional to the noise magnitude ϵ.

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(h(x, w), y))$$

In each case we distinguish the case of error generic and error specific attack:

1. **Error Generic**: $x'_t = x_{t-1} + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(h(x, w), y))$
2. **Error Specific**: $x'_t = x_{t-1} - \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(h(x, w), C))$ where C is the target class.

Fast Gradient Sign Method attack was originally implemented by Goodfellow et al. (2015) with the infinity norm (and is known as the Fast Gradient Sign Method). In ART library, the implementation of this attack extends the attack to other norms, and is therefore called the Fast Gradient Method.

To perform the attack, the following attack parameters were considered:

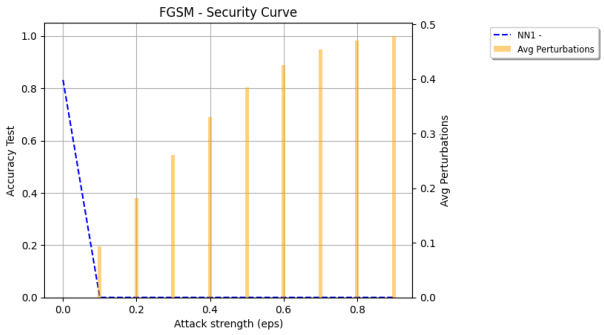
- **param norm**: The norm of the adversarial perturbation. Possible values: inf, np.inf, 1 or 2. Default value is inf
- **param eps**: Attack step size (input variation).
- **param eps_step**: Step size of input variation for minimal perturbation computation.
- **param targeted** (`bool`): Indicates whether the attack is targeted (True) or untargeted (False)

Chosen Parameters The selection of eps was made to explore the effectiveness of the attack at different levels, analyzing how the effectiveness of the attack varies with the increase of this value. The parameter eps_step was proportional to eps to ensure a gradual and controlled change in the input at each iteration, thus increasing the probability of success of the attack without introducing large jumps that could be easily detected as anomalies. It's always fixed at eps/10. We proceeded to implement an attack procedure as described in the following block:

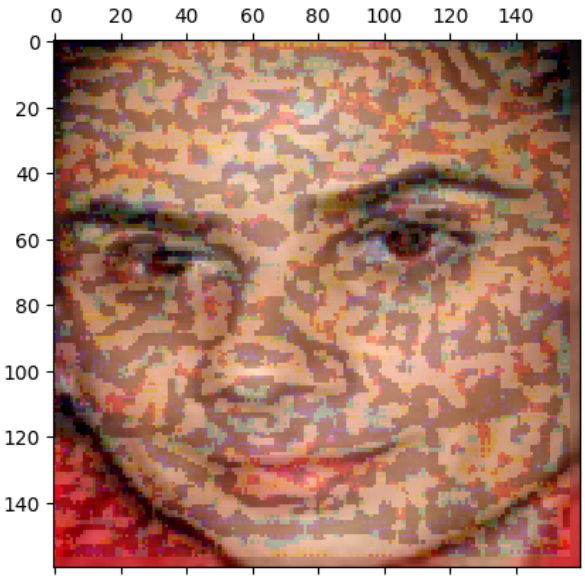
```
    # param for ART
    # norm not setted --> default L_inf
    epsilons = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
    eps_step = eps / 10

    # pseudo code for generate attack
    for eps in epsilons:
        x_perturbed = generat_attack_FGSM(eps, eps / 10, samples)
```

Attack Gray-Box NN1

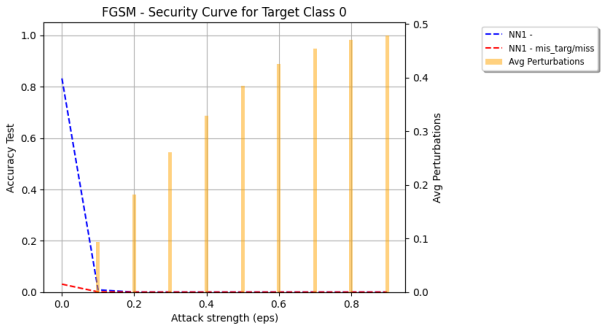
Untargeted attack FGSM is an effective attack to undermine the classifier's ability to classify samples correctly. Particularly, even with a very small value of epsilon like 0.1, the accuracy of the network diminishes. Conversely, the increasing trend of perturbation on the input sample is dictated by the strength of the attack: increasing epsilon amplifies the perturbation.



Targeted Attack Such low accuracy values are perfectly justified by looking at the images produced by this attack. An example is shown with eps=0.1

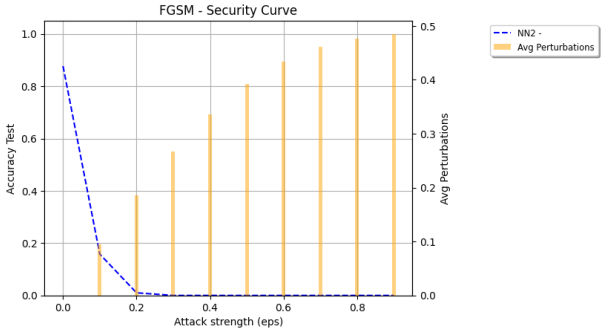


On the other hand, when performing a targeted attack, the limitations of this method become apparent. The constraint of modifying the input sample only once prevents the attack from achieving its primary goal, which is to make the model predict the target class.

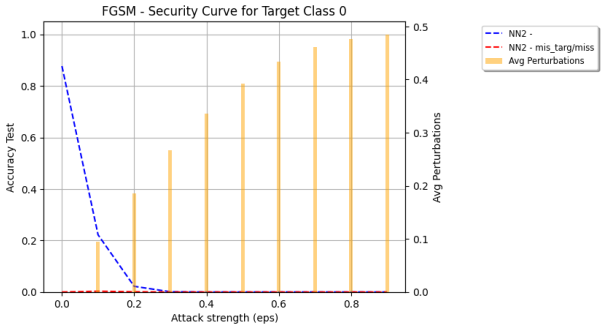


Attack Black-Box NN2

Untargeted attack The security curve for analyzing the transferability of the attack on NN2 reveals that such an attack is transferable for the error generic attack category from a certain epsilon value onwards. Unlike NN1, NN2 is indeed less robust against attacks with lower strength.



Targeted Attack The accuracy of the NN2 network for an error-specific attack remains almost similar to that of an error-generic attack, but the attack's ability to confuse the network with the target class are practically zero.



This shows that such an attack has little ability to direct classification to the target class. This was highly predictable looking at the performance of the targeted attack performed on NN1.

Attack feasibility Assessment From the analysis of the security evaluation curves, it was inferred that the FGSM attack achieves good results in confusing the network both in the classification of NN1 as well as in the case of NN2. The analysis on the adversary samples produced, however, brings out the limitations of such an attack and its poor feasibility in environments where imperceptible changes are to be made to the input samples.

3.6.2 Basic Iterative Method

Attack Description Basic Iterative Method (BIM) is a variant of FGSM that iteratively adds noise to the input image x over multiple iterations and clips pixel values of intermediate results after each step to ensure they remain within an ϵ-neighborhood of the original image.

$$X_0^{adv} = X, X_{N+1}^{adv} = Clip_{X,\epsilon}\{X_{adv}^N + \alpha * \text{sign}\nabla_x(\mathcal{J}(X_N^{adv}, y_{true}))\}$$

To perform the attack, the following attack parameters were considered:

- **param eps**: Maximum perturbation that the attacker can introduce.
- **param eps_step**: Attack step size (input variation) at each iteration.
- **param max_iter** (`int`): The maximum number of iterations.
- **param targeted** (`bool`): Indicates whether the attack is targeted (True) or untargeted (False).

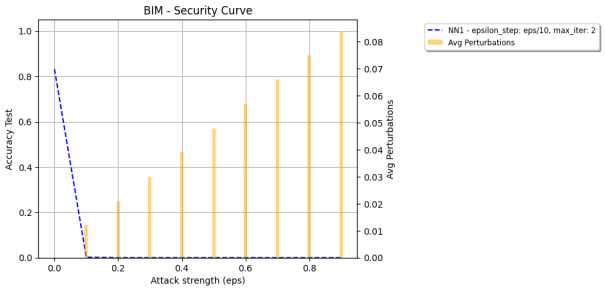
Chosen Parameters The selection of max_iter (maximum number of iterations) was made to explore the effectiveness of the attack at different iteration levels, analyzing how the effectiveness of the attack varies with the number of iterations and the computational time balance. The parameter eps_step was proportional to eps to ensure a gradual and controlled change in the input at each iteration, thus increasing the probability of success of the attack without introducing large jumps that could be easily detected as anomalies. It's always fixed at eps/10.

We proceeded to implement an attack procedure as described in the following block:

```
# param for ART
# norm not setted --> default L_inf
epsilons = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
eps_step = eps / 10
max_iters = [2, 5, 10]

# pseudo code for generate attack
for max_iter in max_iters:
    for eps in epsilons:
        x_perturbed = generate_attack_BIM(max_iter, eps, eps /
            10, samples)
```

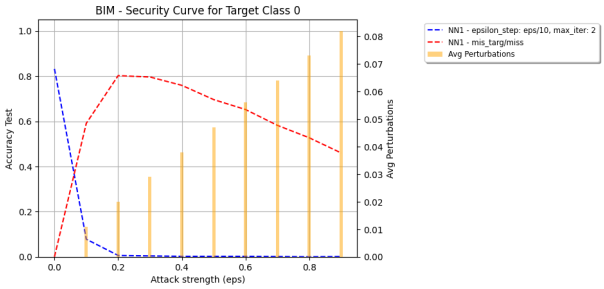
Attack Gray-Box NN1 For the analysis of Gray-Box attacks on NN1, the same distinction between untargeted and target attacks adopted for the FGSM attack analysis is maintained.

Untargeted Attack As with the FGSM attack, BIM also works very well in untargeted attacks. In fact, even in this case, already after values of 0.1 for the epsilon step the accuracy of the classifier is negated.



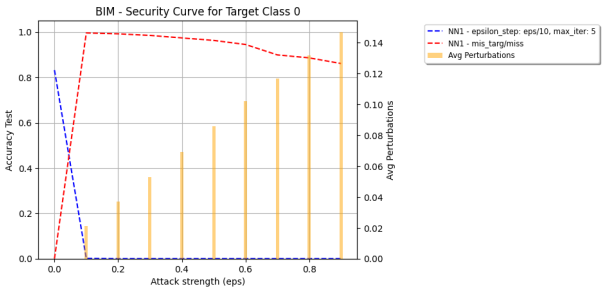
What is noticeable for the security evaluation curves with max_iter 5 and max_iter 10, respectively, is that the SEC trend remains almost unchanged, with the only difference being that as max_iter increases, the perturbation applied to the input sample increases. This is justified by the fact that by increasing the number of iterations to modify the sample, it can be modified multiple times, causing the noted effect. Specifically, we go from a maximum perturbation of about 0.08 for max_iter 2 to a maximum perturbation of about 0.175 for max_iter 10.

Targeted Attack Regarding targeted attacks, however, the characterization it can do is more detailed. Unlike FGSM, such an attack is able to perform well on the targeted attack.

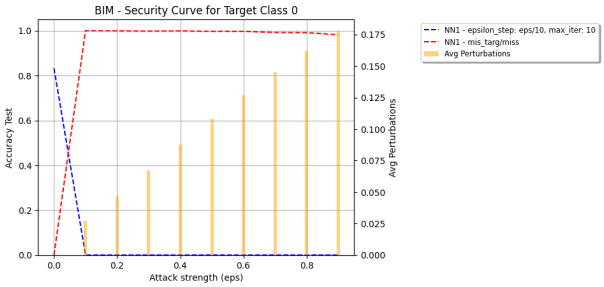


From the Security Evaluation Curves shown above, it can be seen that as max_iter increases, the accuracy on the target class tends to stabilize on 1 already when the value of epsilon is low. In particular, for max_iter 2 the accuracy on the target class reaches a maximum value of 0.8, decreasing as epsilon increases. This is justified by the fact that the number of iterations performed, based on high epsilon values, turns out to be insufficient to allow the attack to succeed.

With max_iter 5 this trend still results, but unit accuracy values begin to be reached.



With max_iter 10 the accuracy stabilizes at the unit value right away.



Attack Black-Box NN2

Untargeted Attack The evaluation of the Security Evaluation Curves for NN2 shows that the untargeted attack appears to be well transferable, especially as the value of max_iter increases.

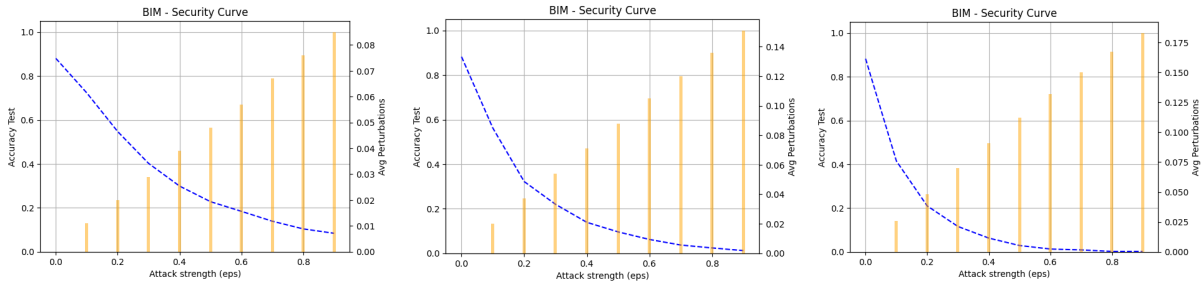


Figure 1: Security Evaluation Curves as textmax_iter increases, respectively equal to 2, 5 and 10. The blue curve indicates the accuracy of the system, the red curve indicates the accuracy on the target class

Targeted Attack Analysis of the targeted attack, on the other hand, shows that the accuracy on the target class always remains close to 0. Unlike the targeted attack with FGSM, however, one is able to target some samples to the target class, bringing out an improvement in the performance of this attack compared to its non-iterative version. As the value of max_iter increases, the peak achieved for accuracy on the target class increases.

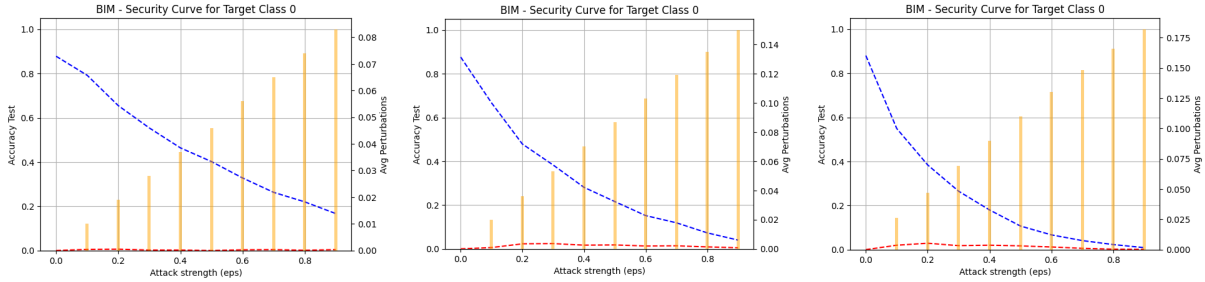
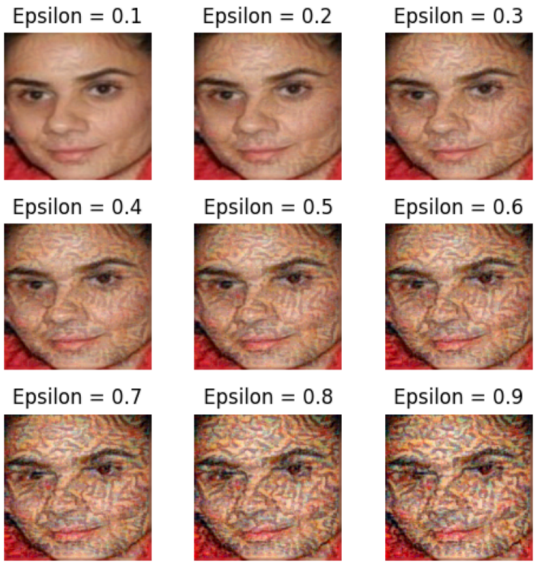


Figure 2: Security Evaluation Curves as textmax_iter increases, respectively equal to 2, 5 and 10. The blue curve indicates the accuracy of the system, the red curve indicates the accuracy on the target class

Attack feasibility Assessment As with FGSM, the feasibility of such an attack is evaluated. Again, the Security Evaluation Curve shows good results in confusing networks. The problem always carries over to the created adversary samples.

The feasibility of the BIM attack was evaluated based on a qualitative analysis of the perturbed images generated by the attack. The perturbed images were visually inspected to assess the extent of the perturbations introduced by the attack and their impact on the image quality. The feasibility assessment aimed to determine whether the perturbations were imperceptible to the human eye and whether the attack was successful in generating adversarial examples that could deceive the target model.

(a) Original image



(b) Perturbed image with different ϵ

The perturbations applied to the input sample result to be such that the appearance of the image is greatly changed. Therefore, BIM attack is hardly feasible in environments where imperceptible changes are to be made to the input samples.

3.6.3 Projected Gradient Descent

Attack Description Projected Gradient Descent (PGD) is an iterative technique used to generate adversarial examples, which are intentionally modified inputs designed to deceive a machine learning model. In formal terms, given an original image x and an associated label y, the objective of PGD is to find a perturbation ϵ^* that maximizes the loss function $J(\theta, x+\epsilon^*, y)$, while keeping ϵ^* within a set of allowable perturbations S. This can be formulated as the following optimization problem:

$$x_{t+1} = \Pi_{x+S}(x_t + \alpha \cdot \text{sign}(\nabla_x J(\theta, x, y)))$$

where:

- x_t: Represents the perturbed image at the t-th iteration.
- α: Hyperparameter that determines the magnitude of the perturbation applied at each iteration.
- $\text{sign}(\nabla_x J(\theta, x, y))$: Indicates the sign of the gradient of the loss function $J(\theta, x, y)$ with respect to the image x. This gradient points in the direction of maximum change in the loss.
- Π_{x+S}: Projection function that projects the perturbed image onto the set S, ensuring it stays within the bounds of a sphere (typically L_∞ norm) around the original image x.

Iterative process of PGD

1. **Initialization**:
    - Start with the original image x as x_0.
2. **Computing the gradient**:
    - For each iteration t, compute the gradient of the loss function with respect to the current image x_t.
3. **Updating the image**:
    - Perturb the current image by adding $\alpha \cdot \text{sign}(\nabla_x J(\theta, x_t, y))$.
4. **Projection**:
    - Project the perturbed image onto the set S to keep the perturbation within defined bounds.

In the case where the attack aims to induce the target model to predict a specific target class, the attack is formalized similarly but with a substantial difference:

$$x_{t+1} = \Pi_{x+S}(x_t - \alpha \cdot \text{sign}(\nabla_x J(\theta, x, C)))$$

It's no longer a matter of moving away from the correct class, but rather generating a sample $x+\epsilon^*$ that is predicted as C, the target class where $C \neq y$. Therefore, in this case, the objective is to minimize $J(\theta, x + \epsilon^*, C)$.

Chosen Parameters To perform the attack, the following attack parameters were considered:

- **param norm**: L_∞ default.
- **param eps**: Maximum perturbation that the attacker can introduce.
- **param eps_step**: Attack step size (α) at each iteration.
- **param max_iter**: Maximum number of iterations.
- **param num_random_init**: Number of random initializations within the epsilon ball.

We proceeded to implement an attack procedure as described in the following block:

```
# Parameters for ART
# norm not setted --> default L_inf
epsilons = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
eps_step = eps / 10
max_iters = [2, 5, 10]
random_init = 5

# Pseudocode for generating the PGD attack
for max_iter in max_iters:
    for eps in epsilons:
        x_perturbed = generat_attack_PGD(max_iter, eps, eps /
           10,
                                         random_init, samples)
```

The choice of attack hyperparameters was the result of a feasibility and relevance analysis specifically aimed at balancing between attack effectiveness and maintaining image quality. Parameters such as **eps** (maximum perturbation) were selected to ensure that the introduced perturbations are realistic and not easily perceptible to the human eye. The selection of **max_iter** (maximum number of iterations) was made to explore the attack's effectiveness under different iteration levels, analyzing how attack efficacy varies with the number of iterations and balancing computational time. The **eps_step** was proportioned to **eps** to

ensure a gradual and controlled change in the input at each iteration, thereby increasing the likelihood of attack success without introducing large jumps that could be easily detected.

Setting **num_random_init** to 5 was aimed at ensuring a thorough exploration of the perturbed input space, enhancing attack robustness through multiple random initializations within the epsilon ball.

Attack Gray-Box NN1

PGD attack untargeted In the attack on NN1, the model used to generate the attack is the same as the target model. Therefore, the security evaluation curves related to effective attacks are reported, one SEC for each **max_iter**.

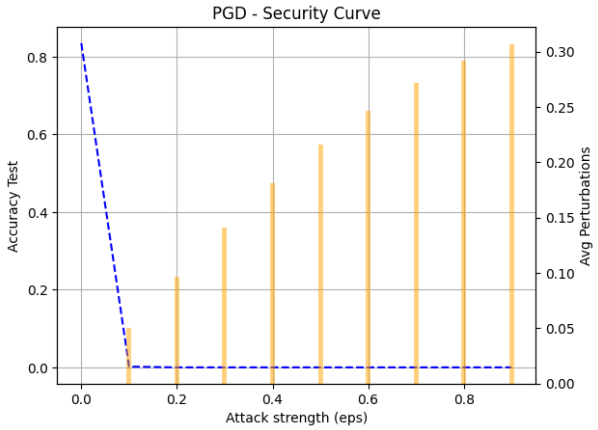
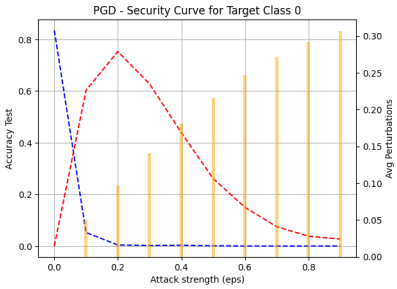


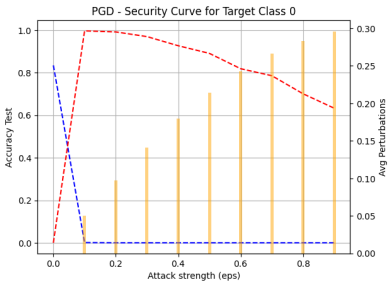
Figure 4: SEC PGD with **max_iters** $= 2$

From the analysis of the SEC, it emerges that the gray-box untargeted attack is highly effective starting from $\epsilon = 0.1$. This highlights NN1's low robustness against a gray-box attack. Moreover, this analysis suggests that attacks with higher maximum iterations will perform similarly, as even with max_iters = 2, the model's accuracy converges to around 0^+. The security evaluation curves for max_iters $= 5$ and max_iters $= 10$ exhibit the same trend. This is because the earlier analysis indicates that only a few iterations are sufficient to optimize the attack.

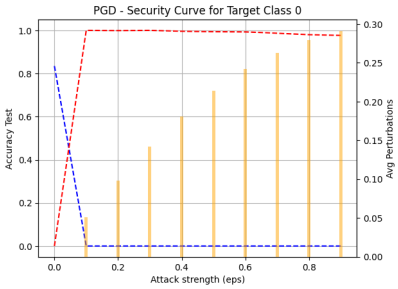
PGD attack targeted In the attack on NN1, the model used to generate the attack is the same as the target model. Therefore, the security evaluation curves related to effective attacks are reported, one SEC for each max_iter.



(a) SEC with max_iters $= 2$



(b) SEC with max_iters $= 5$



(c) SEC with max_iters $= 10$

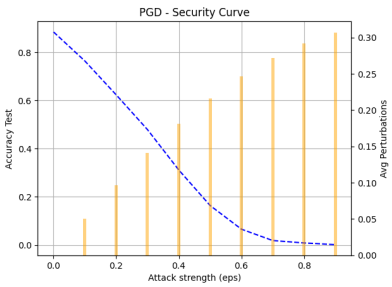
From the analysis of the Security Evaluation Curve (SEC), a substantial difference in the system's behavior is evident regarding a gray-box untargeted attack. In this case, the SEC shows the overall accuracy trend of the system evaluated on adversarial samples in blue, which appears slightly smoother. This is because the adversarial samples are not generated to mislead the sample away from its true class, but rather to prompt the system to classify the sample into the target class. The effectiveness of the attack is depicted in red, illustrating the ratio of samples incorrectly classified by the system with the target label to the total number of misclassified samples.

With max_iters $= 5$, it is noticeable that the targeted attack proves to be more effective across the range of epsilon values but still exhibits a decreasing trend as epsilon values increase. With max_iters $= 10$, we obtain attack effectiveness across the entire epsilon space considered, indicating that the attack converges to a solution close to optimal for each value of ϵ.

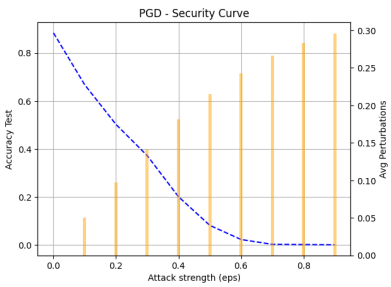
Transferability Analysis In this phase, the goal is to evaluate if the attack is transferable, meaning whether constructing an attack using a surrogate network can undermine the robustness of a target black-box network. To this end, the attack on the network NN2 was analyzed by constructing the attack using the network NN1.

Attack Black-Box NN2

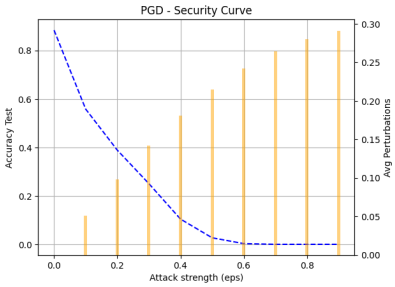
PGD attack untargeted In a first phase, black-box untargeted attacks were analyzed under the same conditions as gray-box attacks.



(a) SEC with max_iters $= 2$



(b) SEC with max_iters $= 5$

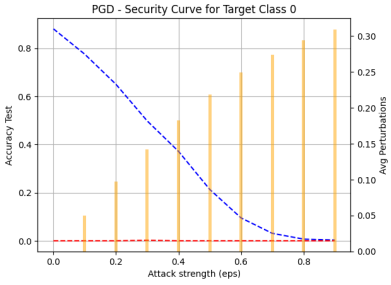


(c) SEC with max_iters $= 10$

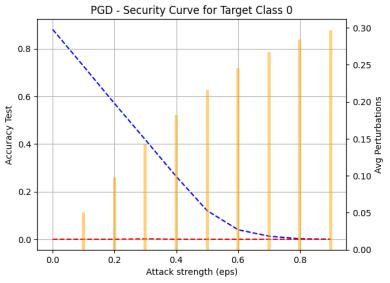
The analysis demonstrates the transferability of the attack. The trend of the acc_{sys} curve appears smoother compared to the white-box attack. Although these iterations are performed on the NN1 model, the graph indicates that the optimizations are not significantly different, because this attack is Gradient-based and the optimization is performed on the gradient of the loss function.

The graphs show that the attack is not only transferable but also more effective as both epsilon and max iters increase, confirming the robustness of the PGD attack method even in a black-box untargeted context.

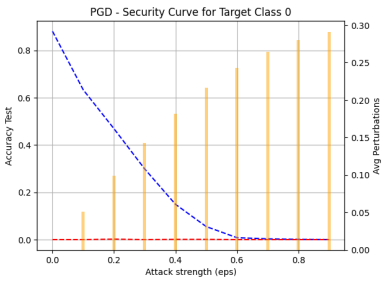
PGD attack targeted In a second phase, targeted black-box attacks were analyzed under the same conditions as the white-box attacks.



(a) SEC with max_iters $= 2$



(b) SEC with max_iters $= 5$



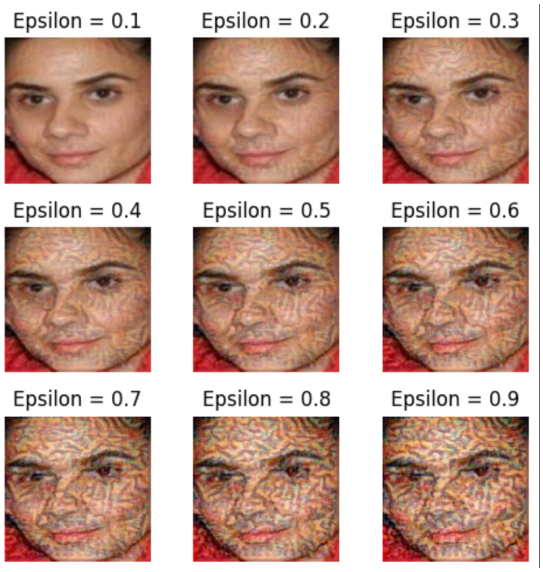
(c) SEC with max_iters $= 10$

From the analysis of the SEC, a substantial difference in the system's behavior compared to an untargeted attack is evident. In this case, the SEC shows in blue the trend of the overall accuracy of the system evaluated on adversarial samples, and it is noted that the curve is slightly smoother. This occurs because the adversarial samples are not generated to move the sample away from its true class, but rather to induce the system to label the sample with the target class.

The curve describing the effectiveness of the attack is represented in red and illustrates the ratio of samples incorrectly labeled with the target label to the total number of misclassified samples. It is noted that this ratio remains fixed at 0, demonstrating that the targeted black-box attack is not transferable. From the graphs of max_iters $= 5$ and max_iters $= 10$, it is evident that the attack's effect is only on the acc_{sys} (system accuracy), likely due to different significant features used by NN1 and NN2 to classify samples into the target class.

Attack Feasibility Assessment Feasibility analysis for the PGD attack follows the same structure as the BIM attack.

(a) Original image



(b) Perturbed image with different ϵ

From the visual inspection of the perturbed images, it can be observed that the perturbations introduced by the PGD attack are imperceptible to the human eye with value of $\epsilon \leq 0.3$. The perturbed images with $\epsilon \geq 0.4$ show slight distortions in the image, but the perturbations are still not easily so the previous analysis suggests that the attack is feasible and effective in generating adversarial examples that can deceive the target model.

3.7 Deep Fool Attack

Attack Description DeepFool is an untargeted attack and not gradient based. Its objective is to move the sample towards the nearest decision boundary so that it is misclassified.

For this reason, implementing a targeted attack is not feasible. The goal is to generate an adversarial input by adding the minimum amount of perturbation. In fact, very small and often imperceptible perturbations of samples are sufficient to deceive state-of-the-art classifiers and cause misclassification.

Formally, for a given classifier, we define an adversarial perturbation as the minimal perturbation r that is sufficient to change the estimated label $\hat{k}(x)$:

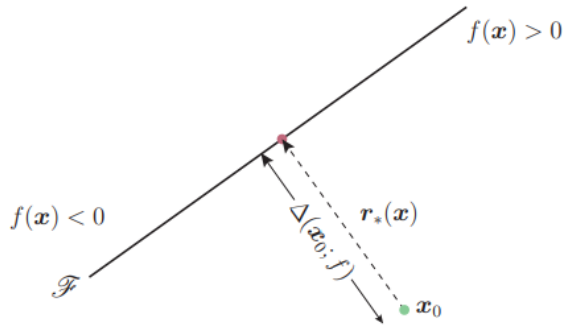
$$\Delta(x; \hat{k}) := min_r ||r||_2 \text{ subject to } \hat{k}(x + r) \; ! = \hat{k}(x)$$

where x is an image and $\hat{k}(x)$ is the estimated label. Calling $\Delta(x; \hat{k})$ the robustness of $\hat{k}$ at the point x. The robustness of the classifier $\hat{k}$ is then defined as:

$$\rho_{adv}(\hat{k}) = \mathbb{E}_x \left(\frac{\Delta(x; \hat{k})}{\|x\|_2} \right)$$

where $\mathbb{E}_x$ is the expectation with respect to the distribution of data.

Below is a graphical representation of $Delta(x; \hat{k})$ in the case of a binary classifier:



The implementation of DeepFool provided by the ART (Adversarial Robustness Toolbox) library uses the L_2 norm as the measure of perturbation.

To evaluate performance under different attack strengths, images were corrupted using various values of the characteristic DeepFool parameters. Specifically, the parameters of interest are ϵ and max_iter.

- The value ϵ represents the amount added to the gradient direction to perturb the image; increasing this value escalates the perturbation in each iteration.

- On the other hand, max_iter indicates the maximum number of iterations the DeepFool attack can execute to perturb the sample. Increasing this value gives the attack more time to find a perturbation that pushes the sample beyond the decision boundary of the classifier. A higher value of this parameter can be useful if the sample is far from the decision boundaries. Empirical observation shows that DeepFool converges in few iterations; therefore, during experimentation, reduced values of the max_iter parameter were evaluated.

3.7.1 Chosen Parameters

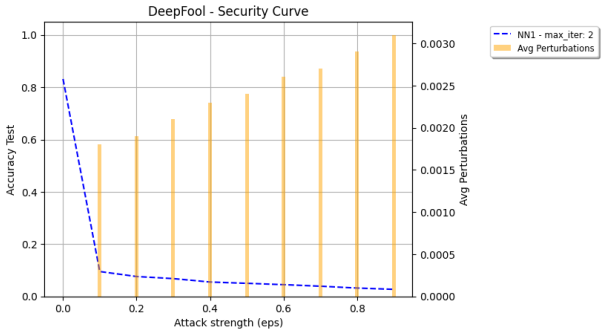
The chosen values for max_iter are $[2, 5, 10]$ and for each of these values, performance was evaluated across eps values ranging from $[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$

The results were not produced with higher values of max_iter for two main reasons:

- As mentioned earlier, DeepFool converges in few iterations.
- Due to time constraints and limited computational resources, generating attacks with higher values of max_iter would have required too much machine time.

3.7.2 Attack Gray-Box NN1

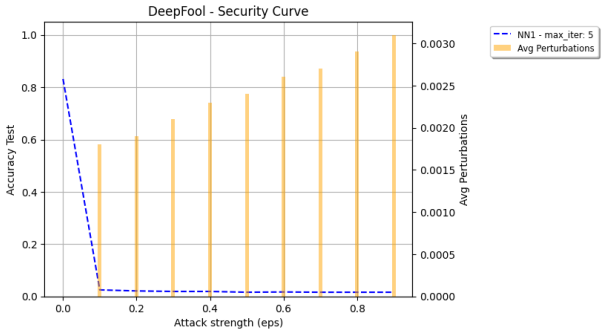
As observed, samples corrupted with DeepFool manage to deceive the NN1 network even with low values of eps and max_iter, leading to an immediate significant drop in accuracy. Regarding average perturbations, it naturally remains low because DeepFool aims to corrupt samples using minimal perturbation.



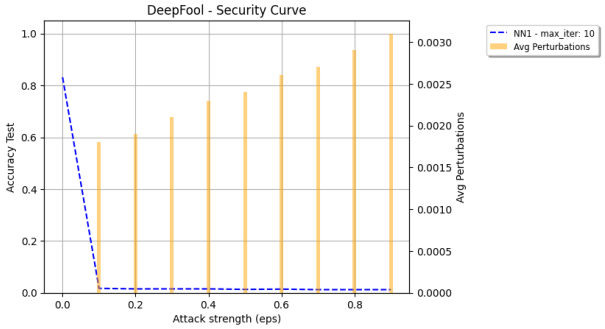
As max_iter increases, one would expect a further reduction in accuracy because the attack has more time to find a perturbation that corrupts the sample and causes the model to misclassify

it. Indeed, from the graphs, we observe that accuracy decreases further. The difference is not stark because the value of max_iter is not excessively high compared to the previous graph; presumably, with higher values of max_iter, accuracy would have decreased even more.

Increasing max_iter can lead to slightly larger perturbations, especially if the initial sample is far from the decision boundary. In this case, average perturbations do not change compared to max_iter=2. This could be due to the minimal difference between the two max_iter values in various runs or because there are not enough samples far from the decision boundaries.



When max_iter increases again, the results obtained are consistent with the previous ones and the same considerations apply.



3.7.3 Attack Black-Box NN2

To evaluate the transferability of attacks generated using NN1 as the classifier, another network named NN2 was defined.

After generating adversarial samples using NN1 as the classifier, the performance in terms of accuracy on the NN2 model was evaluated.

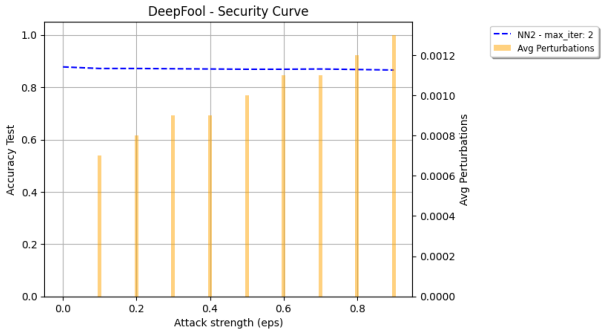
Similar to NN1, accuracy on NN2 was evaluated while varying eps and max_iters using SEC curves. The ranges for changing attack parameters are the same as those used for NN1, specifically to assess transferability. Therefore, max_iter varies within the range [2, 5, 10], while eps varies within the range [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9].

In the first case, with max_iters=2, it can be clearly seen that the accuracy does not drop drastically as it did for NN1. Indeed, the attack is optimized for the characteristics, weights, and structure of the NN1 model.

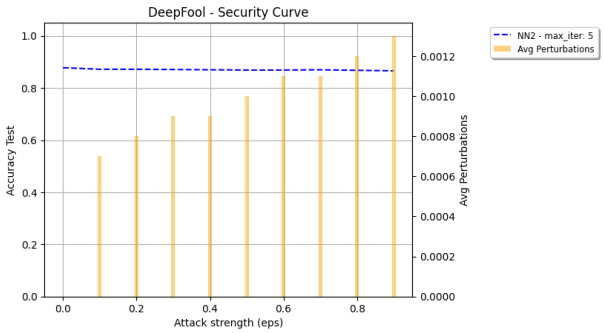
This leads to the conclusion that the DeepFool attacks generated for NN1 do not transfer to the NN2 model, which therefore appears to be more robust to the perturbations introduced by the DeepFool attack. In fact, the accuracy values obtained on corrupted samples are not very different from the accuracy obtained on clean samples.

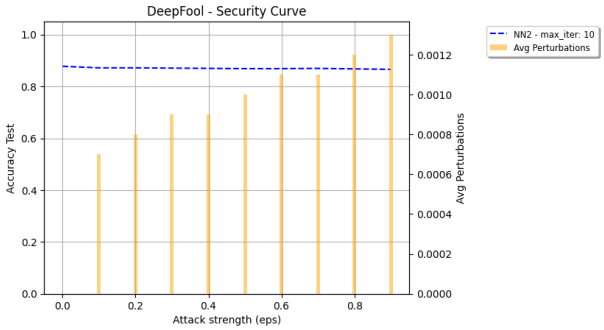
This analysis does not rule out the possibility that increasing max_iters to values like 100 or 1000 could lead to more pronounced decreases in accuracy, as the attack would have more time to find a perturbation sufficient to corrupt the sample enough to mislead the NN2 model.

As for the average perturbations obtained in this case, they are consistent with those obtained in NN1, as expected.



Similarly, in this case, increasing max_iters doesn't show significant differences, which is probably because the max_iters parameter doesn't change much compared to previous runs.

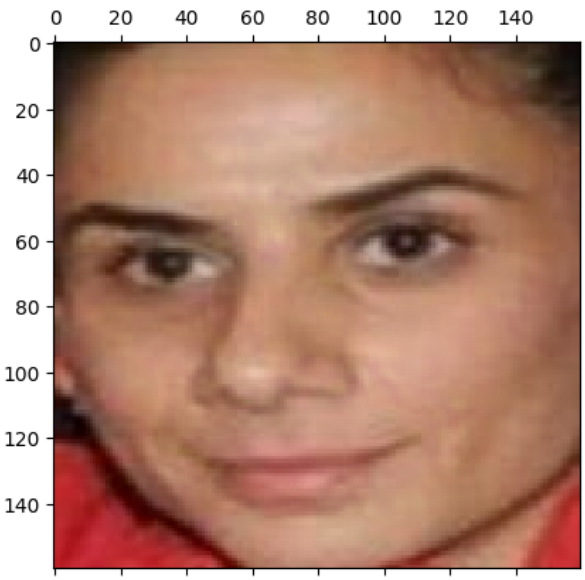


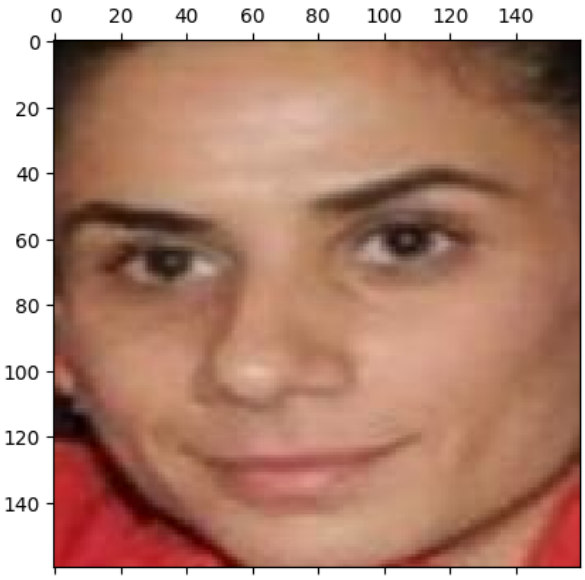


3.7.4 Attack feasibility

As for the feasibility of the attack, it's clear that the attack is feasible because the perturbations introduced into the image are minimal, making them realistic and unaltered.

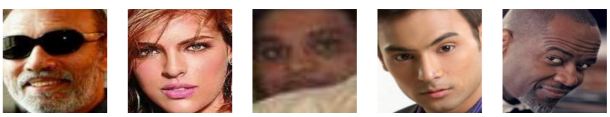
Here's an example of a corrupted image using DeepFool with eps=0.9 and max_iter=2:





The first image depicts the corrupted sample, with a perturbation level of 0.003. The second image depicts the original sample.

Below other corrupted images with DeepFool attack:



3.8 Carlini-Wagner Attack

3.8.1 Attack Description

Out of the Carlini-Wagner pair, three versions of algorithms for generating adversarial samples have been proposed, each based on a different distance measure:

- L_0, which counts the number of pixels altered in the image;
- L_2, which measures the standard Euclidean distance;
- L_∞, which measures the maximum change in each coordinate.

Among these three implementations, L_2 was chosen. The reasons for this choice are:

- This implementation uses different starting points for gradient descent to avoid the greedy search for the minimum that could lead to getting stuck in a local minimum.
- L_2 is the only one among the three proposals that has been parallelized, allowing hundreds of attacks to be executed simultaneously on the GPU.

Formally, the problem to find an adversarial instance for an image x is:

$$\text{minimize } D(x, x+\delta)$$

$$\text{such that } C(x+\delta) = t \text{ and } x+\delta \in [0,1]^n$$

Where x is fixed and the goal is to find δ that minimizes $D(x, x+\delta)$, in other words, finding small values of δ such that the resulting image t will be misclassified by C, but still valid. The proposed formulation is challenging for existing algorithms mainly because the constraint $C(x+\delta) = t$ is highly non-linear. Therefore, it is reformulated in a different way to make it more easily optimizable. Thus, we define an objective function f such that $C(x+\delta) = t$ holds if and only if $f(x+\delta) \leq 0$. The formulation of the problem used:

$$\text{minimize } D(x, x+\delta) + c \cdot f(x+\delta)$$

$$\text{such that } x+\delta \in [0,1]^n$$

Where:

- x is the input sample,
- D is a distance metric, in this case L_2 distance,
- δ is the perturbation added to the sample,
- $c > 0$ is a constant,
- $f(\cdot)$ is the objective function.

To ensure that modifications result in a still valid image, a box constraint on δ must be added.

Moving to the implementation present in ART of the CarliniWagner algorithm, there is the possibility to vary five parameters to find the best combination for generating the most effective attack. The parameters are:

- initial const: initial value of the constant c, used as a starting point for tuning,
- binary search step: number of times the constant c is adjusted via binary search; c is a regularization parameter that balances between minimizing the perturbation and maximizing the probability of misclassification. A larger binary search step makes the algorithm less sensitive to the initial const value,
- confidence: confidence of the adversarial example in fooling the model; a higher value corresponds to producing an adversarial example farther from the original sample but closer to the target class,
- learning rate: initial learning rate for the attack algorithm; smaller values produce better results but converge slower,
- max iter: maximum number of optimization iterations; more iterations can lead to a more precise perturbation at the cost of longer adversarial sample generation time.

3.8.2 Chosen Parameters

Here's an analysis of how tweaking attack parameters affects the **NN1** network. We've tried both targeted and untargeted attacks.

Below are the default values used for generating these attacks:

```
binary_search_steps = 1
confidence = 0.5
learning_rate = 0.01
initial_const = 1000
max_iter = 5
```

We decided to iterate over the values [1, 2, 3, 4, 5, 6, 7, 8] for analyzing binary search step. We tested such a limited number of values for two reasons: first, increasing the parameter value did not worsen the model's classification accuracy; second, increasing the number of binary searches significantly extended the attack generation time, which isn't practical for real-world application.

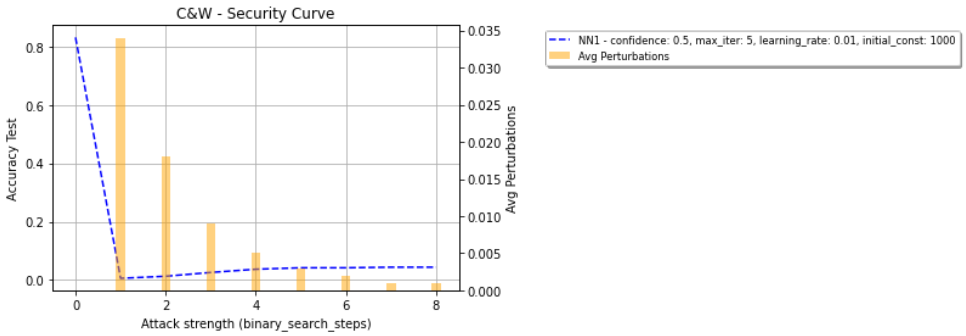
For the confidence parameter, the interval chosen is [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]. This range was selected to cover an entire order of magnitude.

The range of values used for the experiment of learning rate parameter is [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]. Here too, we iterate over values spanning an entire order of magnitude.

The range of values used for the experiment is [5, 6, 7, 8, 9]. The main reason for such a narrow interval is the computational time required as the number of iterations to generate the adversarial sample increases.

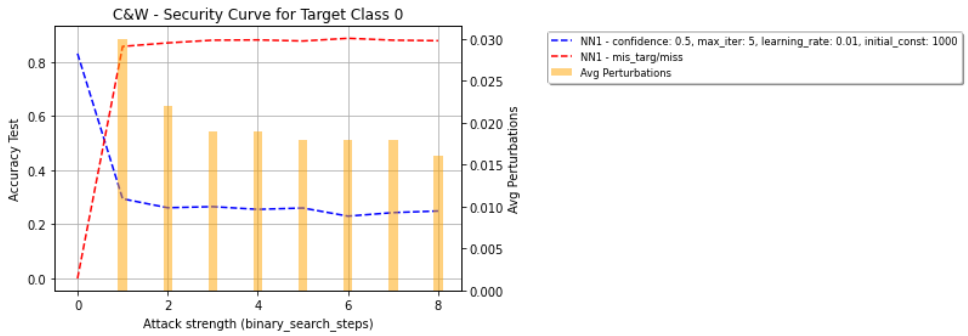
3.8.3 Attack Gray-Box NN1

Binary search step Untargeted execution graph:



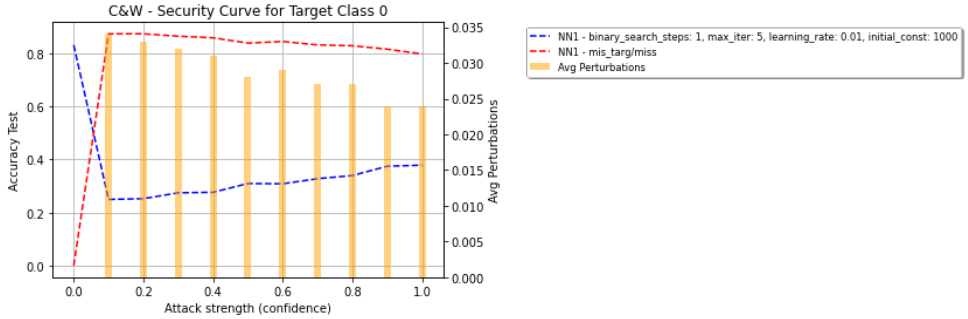
As shown in the graph, we observe a drop from 83% accuracy on the test set without attacks to 0.6% with a binary_search_step of 1. This stabilizes around 4% for parameter values 6, 7, and 8. Additionally, while not clearly visible in the graph, it's evident from the data that increasing the parameter values reduces the perturbation of the adversarial sample, eventually dropping below 1%.

Targeted execution graph:



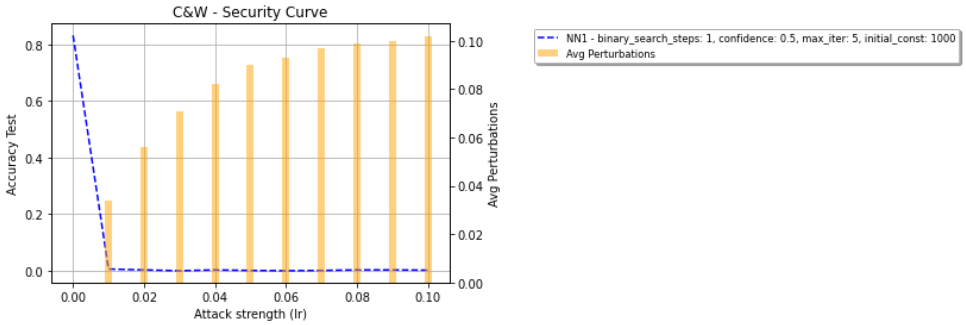
The range of iteration for binary search steps is the same as in the untargeted case. The accuracy of the attacked model is higher compared to the untargeted scenario, settling around 23%. A notable insight from the graph is that using a low value of binary search steps results in an accuracy on the target class of 85%.

Confidence Targeted execution graph:



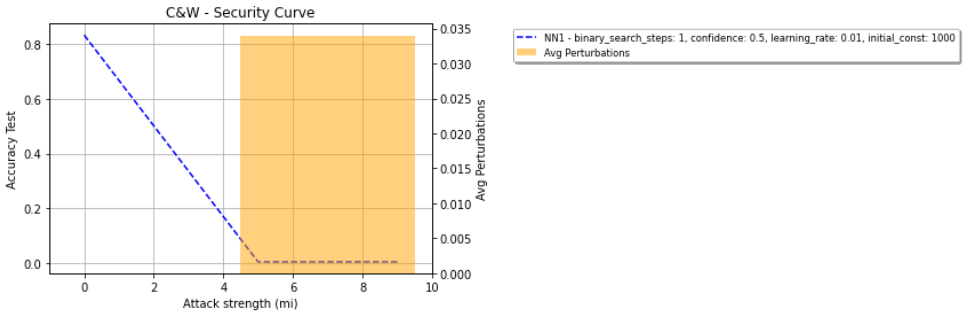
Again, there is higher accuracy on the samples (maintaining the same range of test values), peaking at 37% with maximum confidence. The perturbation of adversarial samples decreases from about 3% to 2.4% as confidence increases.

Learning rate Untargeted execution graph:



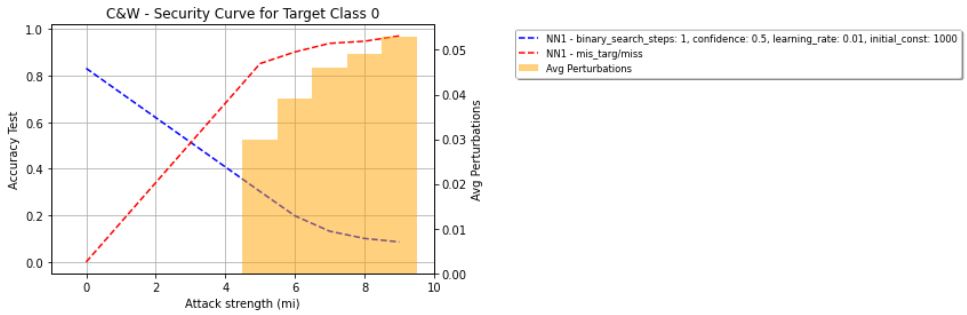
From this run, it's evident that for certain values (0.03 and 0.06), the accuracy was 0%, meaning none of the 998 samples were classified correctly. In other cases, the average accuracy was around 2%. Regarding perturbation values, consistent with the parameter definition, smaller learning rates resulted in smaller perturbations, whereas larger values led to perturbations as high as 10%.

Max iter Untargeted execution graph:



From the graph, it's evident that as the number of iterations increases, the model accuracy remains at 0.6%. It can be inferred that the current combination of parameters is already optimal with a maximum of 5 iterations, and increasing this number only extends the attack generation time without actually worsening the model's performance. Consequently, the perturbation also remains stable at 3.4%.

Targeted execution graph:



Unlike the untargeted attack, here we observe a variation in model accuracy with respect to the parameter. Initially, there is a 30% accuracy which decreases as the number of iterations increases, eventually dropping to 8%. Consequently, the perturbation on the samples increases. Regarding accuracy on the target class, there is a 97% accuracy when the parameter is set to 9.

3.8.4 Attack Black-Box NN2

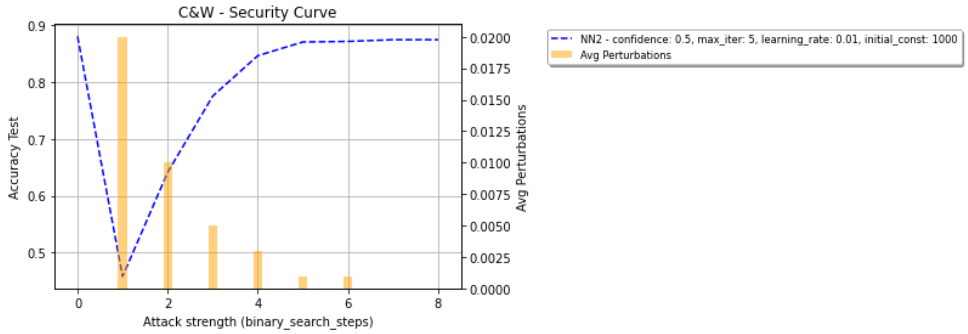
Here's an analysis of how the generation of attacks, varying the parameters, impacts the NN2 network under study. Both targeted and untargeted attacks were conducted.

Below are the default parameter values used for generating these attacks:

```
binary_search_steps = 1
confidence = 0.5
learning_rate = 0.01
initial_const = 1000
max_iter = 5
```

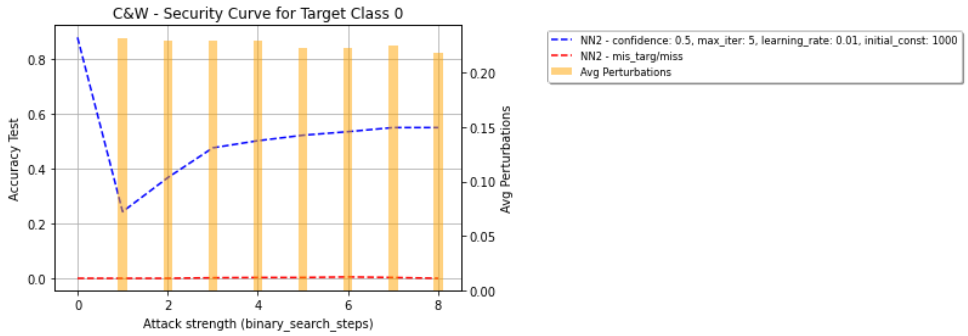
The experiment ranges used for NN2 are the same as those used for NN1, so this analysis focuses on the transferability of attacks generated to target NN1 against NN2.

Binary search step Untargeted execution graph:



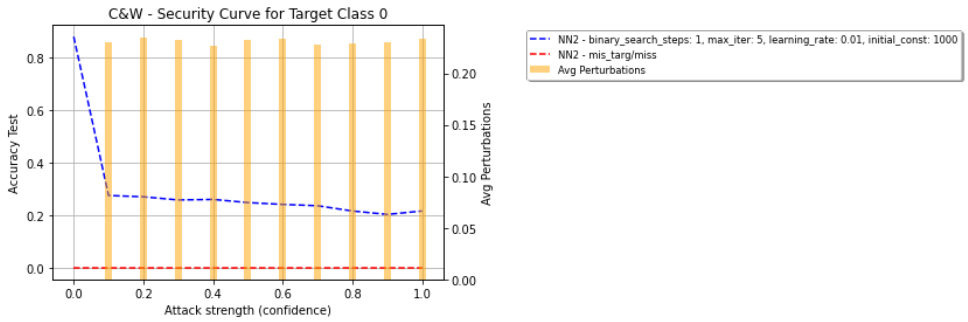
Looking at this graph, it's noticeable that as the number of iterations increases to find the optimal value of *c*, the attack becomes less transferable. This is because the adversarial samples are less altered, making NN2 more robust and thus the generated attacks less transferable (more details in the final transferability analysis).

Targeted execution graph:



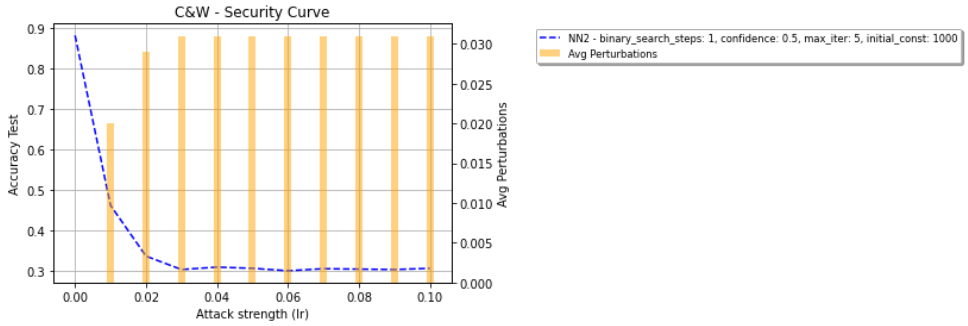
Notice how all adversarial samples generated for the target class are never recognized as belonging to that class, unlike what happened with NN1. It can be stated that the attack is not transferable, even though the accuracy values of the inference made on the model are lower compared to those obtained in the untargeted case.

Confidence Targeted execution graph:



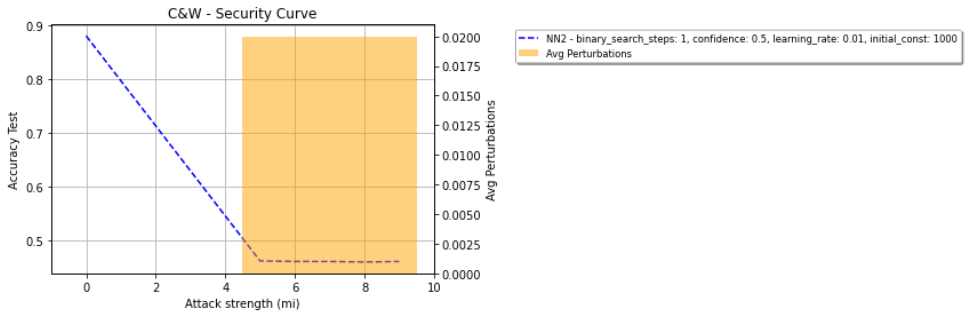
Similarly, the generated samples are never recognized as belonging to the target class. The attack proves to be non-transferable, given the nearly nonexistent accuracy on the target class, despite maintaining a similar behavior regarding the model's overall accuracy.

Learning rate Untargeted execution graph:



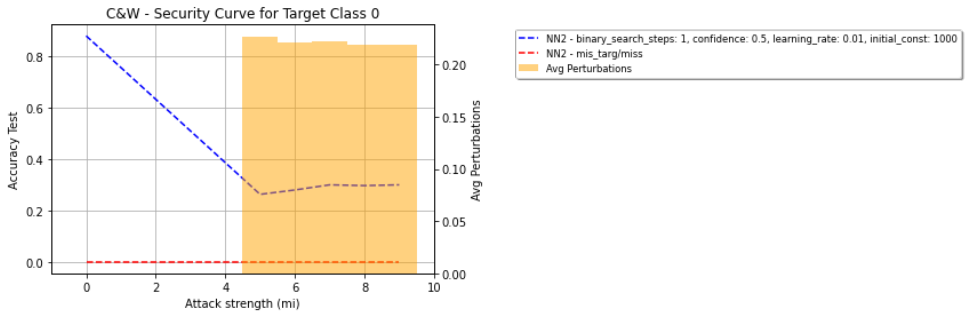
From the graph, it's observed that as the learning rate increases, the model accuracy decreases, reaching a minimum of 30%. In this case as well, this decrease in accuracy is entirely different from what was observed for NN1. Therefore, the generated attacks appear to be partially transferable.

Max iter Untargeted execution graph:



Given the results for the chosen values of iterations for gradient descent, a minimum accuracy of 45% is observed, indicating partial transferability to NN2 in this case as well.

Targeted execution graph:



In this latest case, as in the previous ones, all adversarial samples are not classified as belonging to the target class, indicating non-transferability of the attack. Here too, there is a different trend in the accuracy descent compared to the untargeted case.

3.8.5 Transferability Analysis

The graphs presented for the untargeted attacks generated with the Carlini-Wagner algorithm for NN1 show that they are partially transferable to NN2. In fact, for almost all the untargeted attacks, the minimum accuracy is around 45%, with only one instance showing a minimum accuracy of 30% as the learning rate varies. The attacks are partially transferable because both models were trained on the same dataset, and although they are different, the gradient descent during training produced similar results. Therefore, attacks on NN1 also achieve a certain degree of success on NN2.

On the other hand, generating adversarial samples for a targeted attack showed that NN2 is "robust," as it does not classify any of these samples as belonging to the target class, even

though it is more affected by the attacks (i.e., having a lower overall accuracy). Since the objective function for generating targeted attacks aims to make the samples more easily classified as belonging to the target class, it can be stated that the attack is not transferable from NN1 to NN2. This is justified by the fact that targeted attacks are generated to affect NN1; consequently, for NN2, the generated adversarial samples are simply adversarial samples and not samples to be classified as belonging to the target class.

An unusual behavior is observed as the value of the binary search step increases: the increase in this variable corresponds to a greater number of steps in the binary search performed to find the best value of *c*, which represents the weight given to the function to be minimized in the algorithm. What happens is that this function is minimized with respect to NN1, which can be attacked with a pattern of altered pixels different from those needed for NN2. Consequently, by minimizing the image perturbation more precisely, specific pixels are altered that will cause misclassification in NN1, while NN2 will simply see it as noise that the network can handle.

3.8.6 Attack feasibility assessment

The optimal parameter choice should strike a balance where the adversarial sample generation isn't overly time-consuming, isn't too weak, and doesn't deviate too much from the original sample. For the second point, there aren't particular issues because the Carlini&Wagner algorithm performs well even with 'weak' parameters.

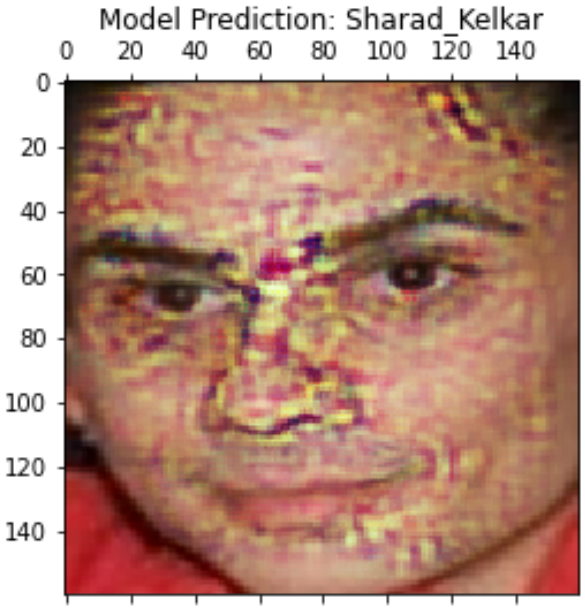
Unlike other algorithms that randomly perturb samples, this algorithm specifically modifies the face.

We begin with an analysis of the untargeted case followed by the targeted case.

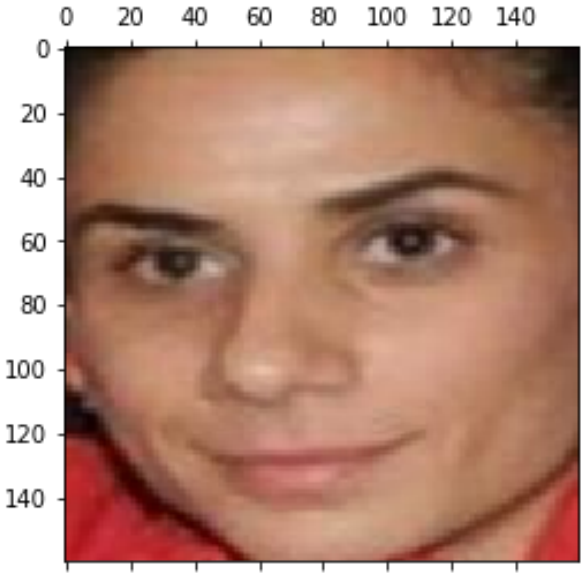
Starting with an evaluation of the default parameters:

```
binary_search_steps = 1
confidence = 0.5
learning_rate = 0.01
initial_const = 1000
max_iter = 5
```

The resulting adversarial image is classified as Sharad Kelkar instead of Nevin Yanıt:



Starting from the original image:



As you can see, despite a 3.8% perturbation, the image is visibly altered.

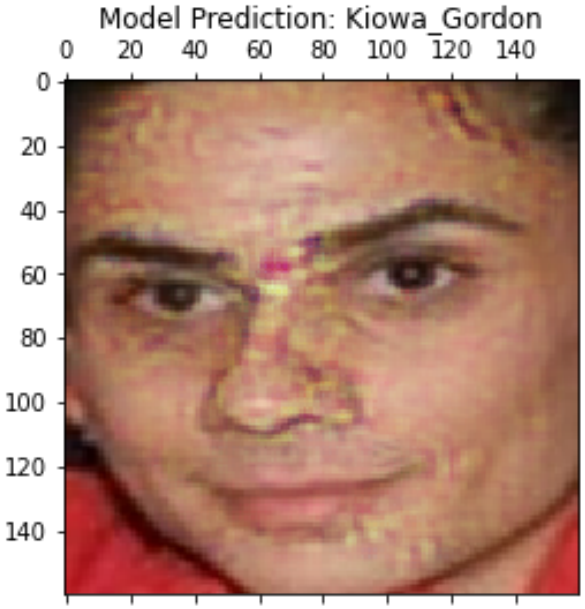
Looking back at the experiments presented earlier, optimal values for an untargeted attack could be:

```
binary_search_steps = 1
confidence = 0.5
learning_rate = 0.01
initial_const = 400
max_iter = 5
```

Changing the binary search steps parameter would have increased the generation time of the adversarial image and reduced the likelihood of the model misclassifying. The max_iter parameter was not modified for the same reason, to avoid increasing the attack generation time.

Adjusting the confidence parameter would not have produced significant changes in attack performance, for the same reason as the learning rate parameter.

Reducing the initial_const parameter benefited both in terms of perturbation and the time taken to generate the attack. In fact, the adversarial image successfully deceived the model into classifying Nevin Yanıt as Kiowa Gordon:

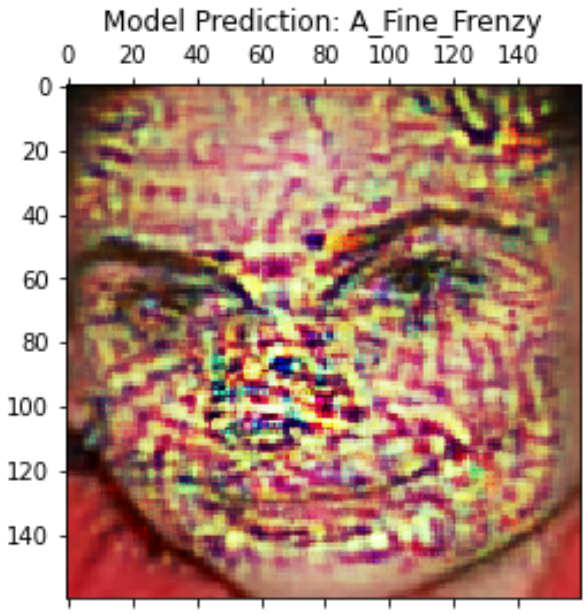


In particular, producing a perturbation of 1.5%.

Moving on to the generation of targeted attacks, using the same parameters initially used to analyze the untargeted attack:

```
binary_search_steps = 1
confidence = 0.5
learning_rate = 0.01
initial_const = 1000
max_iter = 5
```

The target class chosen is the first one in the test set (A Fine Frenzy), resulting in the following adversarial image:

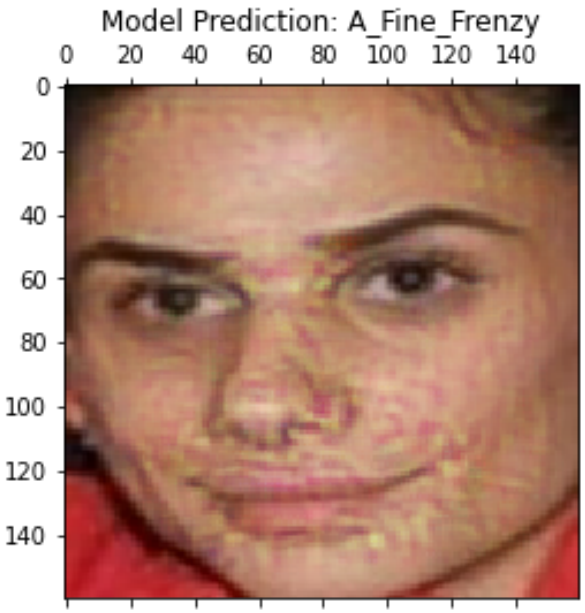


Again, the image is visibly altered (about 9% perturbation), but it is classified by the model as belonging to the target class.

Based on observations from previous experiments, the parameters are adjusted as follows:

```
binary_search_steps = 3
confidence = 0.5
learning_rate = 0.01
initial_const = 1000
max_iter = 7
```

This results in the following image:



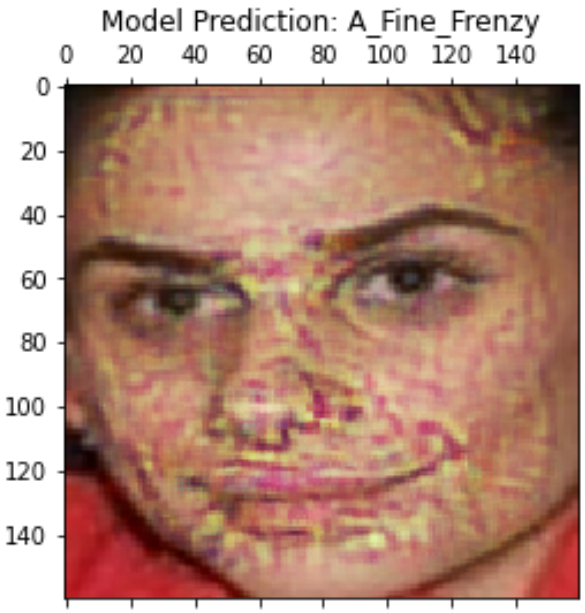
The image is classified as belonging to the target class, even though it doesn't appear very altered (1.3% perturbation). This parameter combination might be considered optimal, but its generation time is significantly increased due to the higher binary search steps and maximum iterations.

Maintaining the same parameters except for max_iter (reduced to 5), results in a very similar outcome in terms of both time and perturbation as the previous one. However, with the combination of parameters binary_search_steps = 1 and max_iter = 7, an altered image with 9% perturbation (like the first case) is obtained.

A good parameter combination turns out to be:

```
binary_search_steps = 2
max_iter = 5
```

which produces:



With a perturbation of 2.8%.

Regarding the other three parameters, they were not otherwise modified or there would have been a higher accuracy of the model on the generated adversarial samples, as shown in the graphs of previous experiments.

This combination of parameters turns out to be the best for two reasons: a lower value of binary search steps produces higher perturbation, while a lower number of iterations does not modify the image at all (practically the sample remains unaltered).

4 Defense Mechanisms

To make the NN1 network more robust against implemented attacks, a Detector was implemented upstream of NN1. This detector receives the image as input and determines whether the sample is corrupted by an attack or if it is clean. Only if the sample is not corrupted will it be passed to the NN1 network, which then assigns an identity to the face. This approach hypothetically ensures that no corrupted sample is passed to the NN1 network.

The Detector was built using a pre-trained neural network, MobileNetV2, as the base. MobileNetV2 was chosen because it is known for being a lightweight and highly efficient network. Using heavier neural networks, like ResNet, would have resulted in a more complex and computationally expensive system. Additionally, due to time constraints, choosing a lighter network reduced the training time. The pre-trained weights were loaded using the ImageNet dataset.

The final classifier layer of the network was then replaced with a new linear layer to adapt it to the specific binary classification task, with outputs corresponding to “corrupted” and “clean.”

The optimizer used was AdamW, with a learning rate set to 0.001, and the loss function used was Cross Entropy.

To optimize the training process’s efficiency and leverage the model’s existing knowledge, only the last classifier layer was trained. During the model’s training phase, 100 epochs were conducted, but to avoid overfitting and improve efficiency, an early stopping criterion was set at 15 epochs.

At the end of the training process, the model was evaluated using a validation set composed of both corrupted and clean samples. The model achieved an accuracy of 0.99, indicating that the detector is highly effective at recognizing whether a face is corrupted or not.

4.1 Train and Validation set for detector

The data used for training the detector was taken from the VGG-Face2 dataset. Among all the identities present in the dataset, the 100 identities used for the Test Set for NN1 and NN2 were excluded.

Subsequently, an additional 1000 identities were randomly selected with seed 84. For each of these identities, 10 images were randomly selected (seed 84).

At this point, the 10,000 samples were processed by the MTCNN module, producing cropped images with a size of 160x160 (since the defense will be applied to NN1). The images were saved in the folder `training_set_detector/0`.

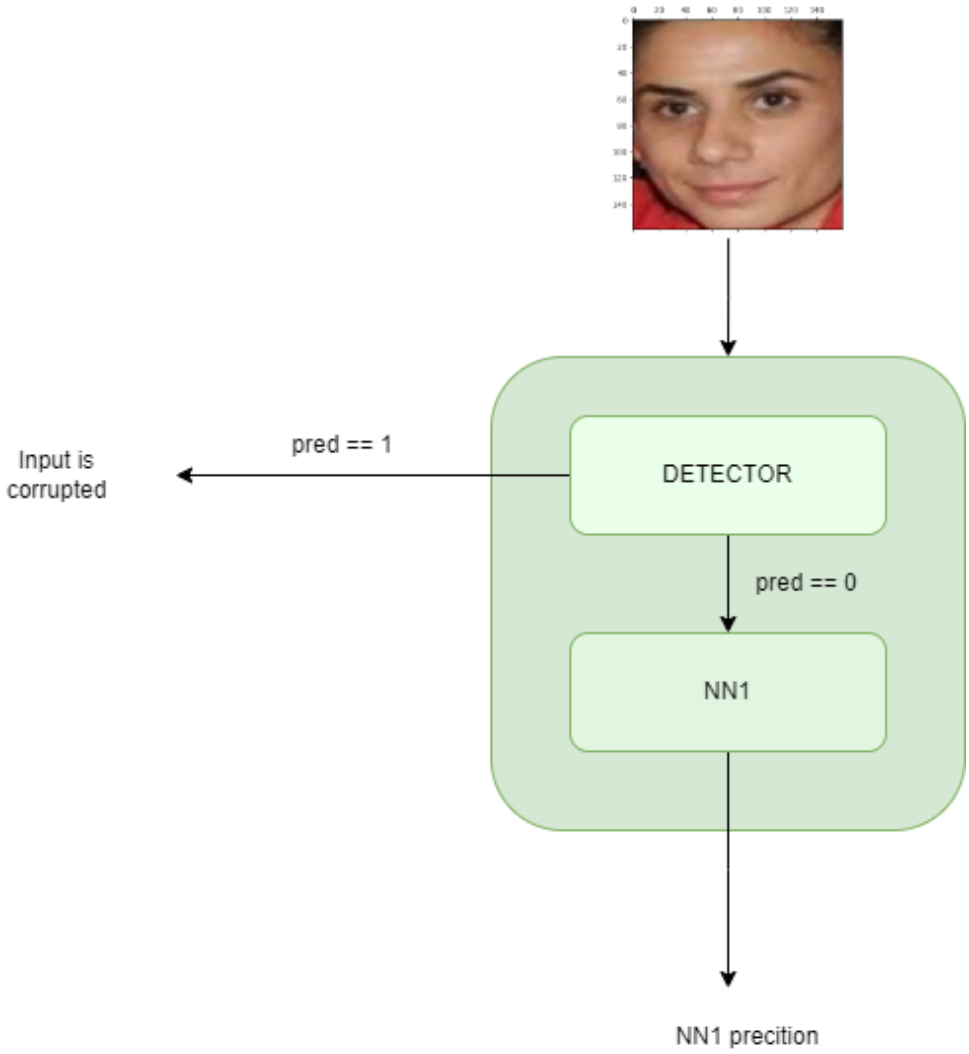
These 10,000 samples were equally divided among different types of attacks: 2000 samples for FGSM, 2000 samples for BIM, 2000 samples for PGD, 2000 samples for DeepFool, and 2000 samples for CarliniWagner. For each type of attack, these samples were further split in half to generate both targeted and untargeted attacks, except for DeepFool, which does not perform targeted attacks. Thus, adversarial samples for each type and each attack were generated and then saved in the folder `training_set_detector/1`.

Finally, from the generated adversarial data, a validation set for each attack was created following the 80-20 rule: 80%, or 800, of the samples were used for the training set, and 20%, or 200, of the samples were used for the validation set. The final folder structure for the detector training data (`training_set_detector`) is as follows:

```
./training_set_detector
    ./train
        ./0
        ./1
    ./val
        ./0
        ./1
```

4.2 Full System Architecture

The full system architecture is shown below:

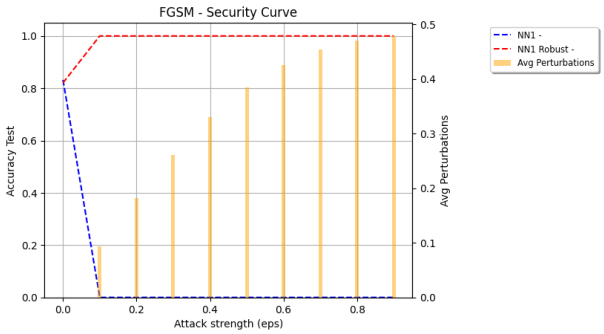


The system is compose from a detector that perform a detection of adversarial samples, if the sample is detected as adversarial, it is discarded, otherwise it is passed to the NN1 network. If the discard is performed for the adversarial sample, then the number of corrected samples is increased by one, else if the prediction of the NN1 network is correct, the number of correct samples is increased by one.

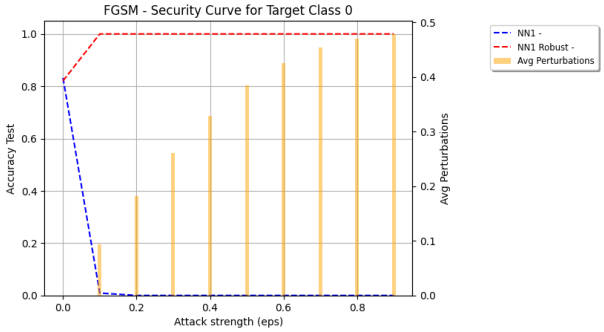
The test set used for evaluating the performance of the trained detector is the same one used for generating attacks on NN1 and NN2.

4.3 FGSM Defense

We proceed with an analysis of the overall system's performance (detector and model) against attacks generated by the FGSM.



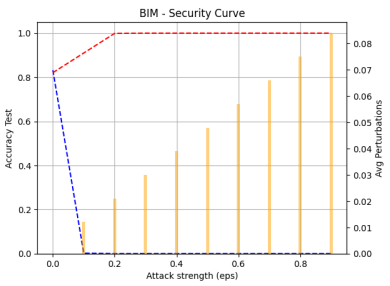
From this Security Evaluation Curve, it is evident how the corrupted sample detector works particularly well in the case of samples generated by this type of attack. The performance of the detector is such that it blocks all corrupted samples before they reach the NN1 network, leading to an increase in overall system performance. This behavior is also evidenced in the case of targeted attacks.



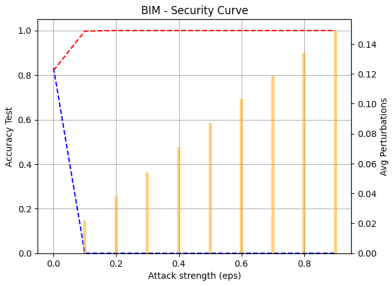
4.4 BIM Defense

The system with integration of pre-processing Detector was analyzed against the PGD attack, respect target and untargeted attacks.

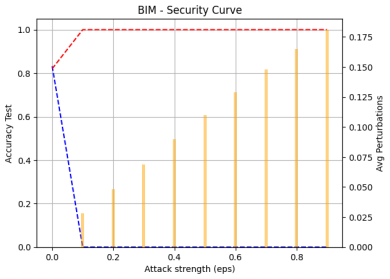
Untargeted Defense Below are the results of the system with the SEC curve for the untargeted attack.



(a) SEC with max_iters $= 2$



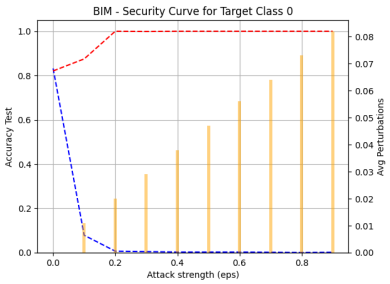
(b) SEC with max_iters $= 5$



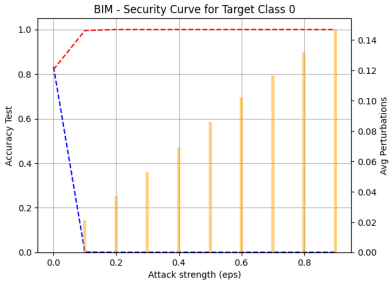
(c) SEC with max_iters $= 10$

Based on the analysis of the SEC curve, it is clear that the detector successfully identifies adversarial samples produced by the BIM attack. The detector's effectiveness prevents all compromised samples from reaching the NN1 network, thereby enhancing the overall system performance. This capability is similarly demonstrated in the context of targeted attacks.

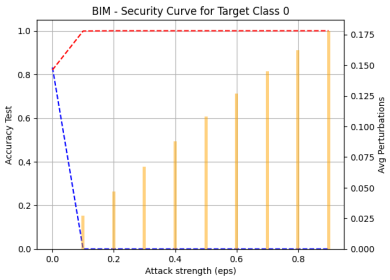
Targeted Defense Below are the results of the system with the SEC curve for the targeted attack.



(a) SEC with max_iters $= 2$



(b) SEC with max_iters $= 5$



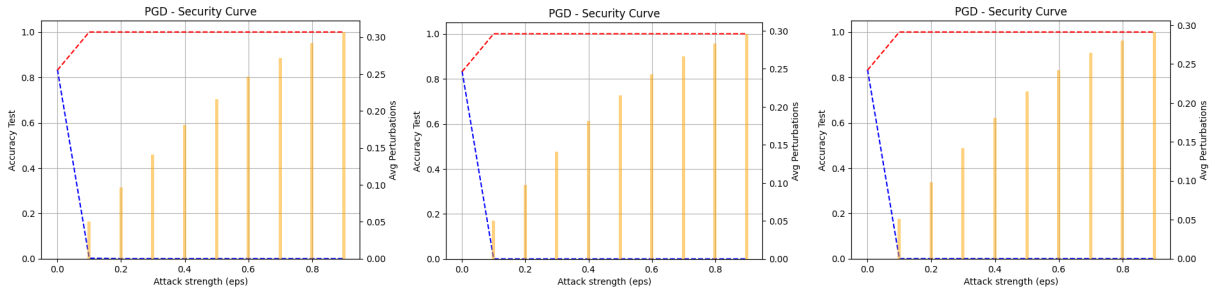
(c) SEC with max_iters $= 10$

Similar to the untargeted attack, the detector successfully identifies the adversarial samples produced by the BIM attack. The detector's performance ensures that all compromised samples are intercepted before reaching the NN1 network, resulting in an overall improvement in system performance. This behavior is also observed in the case of targeted attacks.

4.5 PGD Defense

The system with integration of pre-processing Detector was analyzed against the PGD attack, respect target and untargeted attacks.

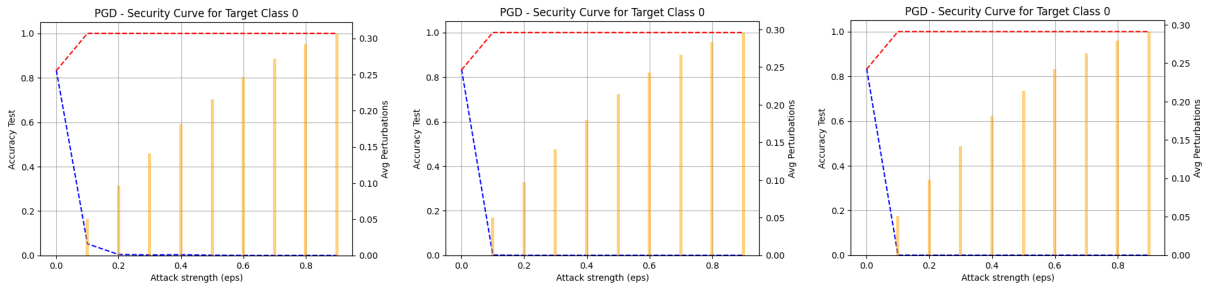
Untargeted Defense Below are the results of the system with the SEC curve for the untargeted attack.



(a) SEC with max_iters $= 2$ **(b)** SEC with max_iters $= 5$ **(c)** SEC with max_iters $= 10$

From analysis of the SEC curve, it is evident that the detector is able to detect the adversarial samples generated by the PGD attack. The performance of the detector is such that it blocks all corrupted samples before they reach the NN1 network, leading to an increase in overall system performance. This behavior is also evidenced in the case of targeted attacks.

Targeted Defense Below are the results of the system with the SEC curve for the targeted attack.



(a) SEC with max_iters $= 2$ **(b)** SEC with max_iters $= 5$ **(c)** SEC with max_iters $= 10$

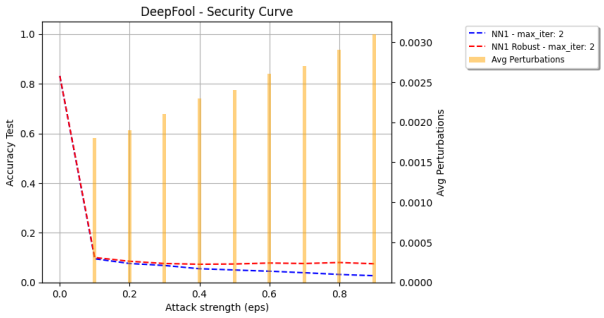
Like the untargeted attack, the detector is able to detect the adversarial samples generated by the PGD attack. The performance of the detector is such that it blocks all corrupted samples before they reach the NN1 network, leading to an increase in overall system performance. This behavior is also evidenced in the case of targeted attacks.

4.6 DeepFool defense

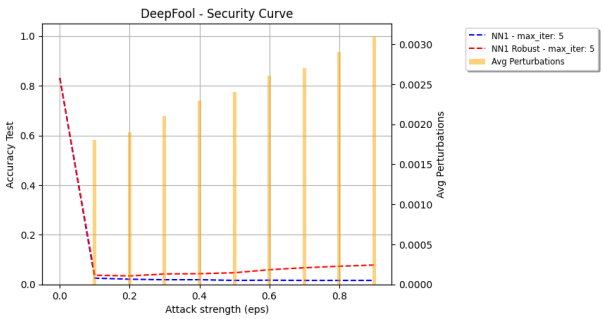
SEC curves are analyzed to evaluate the overall system accuracy, which includes the detector upstream of the NN1 network.

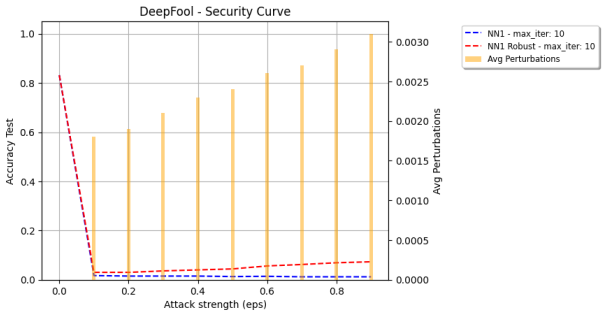
As you can clearly see, the detector is not able to detect samples attacked with the DeepFool attack. This is because the perturbations applied to the image are very minimal and difficult to detect. A slight improvement, compared to the system with only NN1, is slightly visible as epsilon increases since, for these values, the image is more perturbed compared to lower epsilon values.

The number of corrupted samples that the detector recognizes as such for epsilon = 0.1 and max_iter=2 is 21. Therefore, the remaining corrupted samples are passed to the NN1 model, which assigns them to the wrong class, resulting in lower accuracy. As the attack strength increases, the number of corrupted samples recognized by the detector rises, reaching a maximum of 105.



As the max_iter parameter increases, the performance of the system using the detector decreases even further. This is because the attack has more time to find the perturbation to effectively corrupt the sample, making it even harder for the detector to recognize adversarial samples.





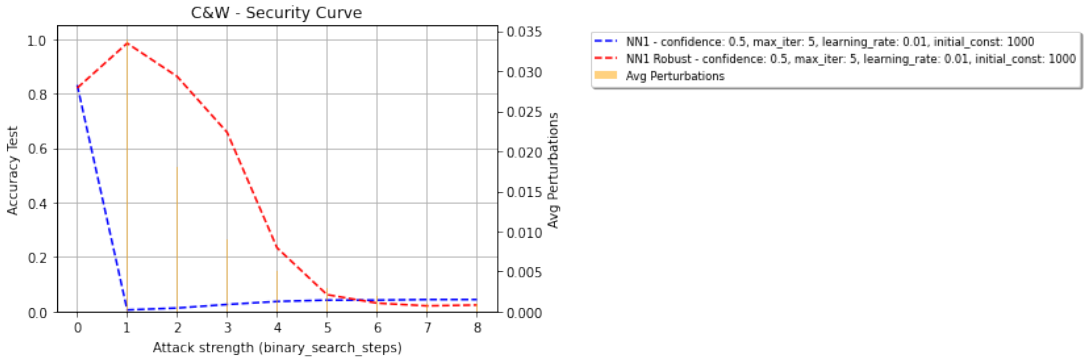
4.7 Carlini-Wagner Defense

We proceed with an analysis of the overall system's performance (detector and model) against attacks generated by the Carlini-Wagner algorithm. The default parameters for generating the attacks are:

```
binary_search_steps = 1
confidence = 0.5
learning_rate = 0.01
initial_const = 1000
max_iter = 5
```

Some of these five parameters are iterated over the same ranges analyzed previously in the untargeted scenario.

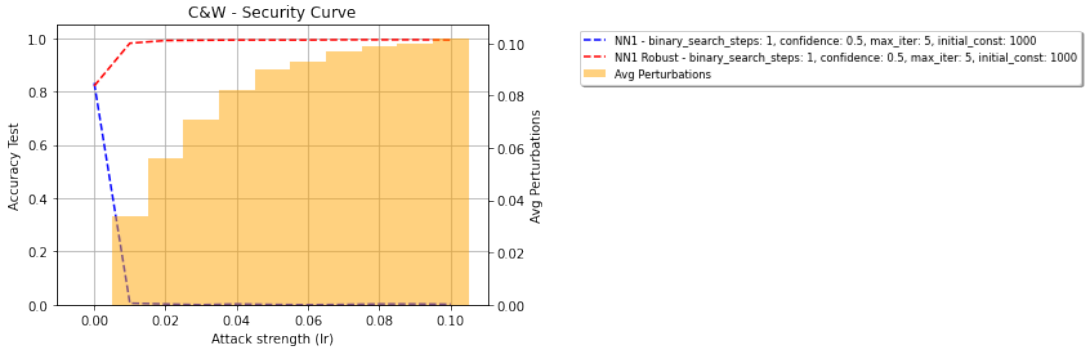
Binary search step Untargeted execution graph:



The behavior observed with the defense is completely different from that obtained without

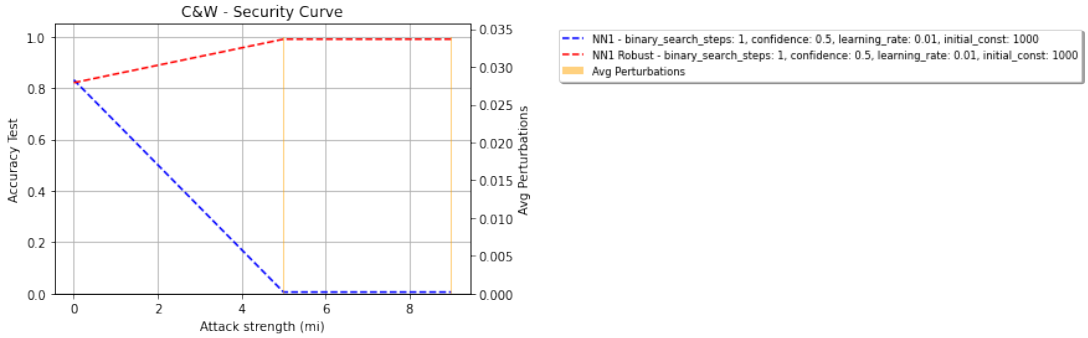
the defense. What can be noted is that when the attack becomes finer the system decreases in its performance, in according to decreasing of perturbation on adversarial samples. In fact, with higher values of this parameter, the number of samples discarded is less than 3%.

Learning rate Untargeted execution graph:



During the inference of the generated samples, almost all samples are ignored. This means that the adversarial component added to the image is detectable by the detector.

Max iter Untargeted execution graph:



In this case with almost all samples being rejected by the detector, this indicates that the detector is effective in identifying adversarial samples.

5 Conclusions and Future Developments

5.1 Conclusions

In conclusion, this work has evaluated the robustness of two neural networks (referred to as NN1 and NN2) in face recognition, focusing on their resilience against adversarial attacks. First, a test set of face images was created using VGG-Face2. Then, the performance of NN1 and NN2 was evaluated on uncorrupted test samples. To test the robustness of the networks, adversarial samples were generated using FGSM, BIM, PGD, DeepFool, and CarliniWagner attacks, with the help of the ART library, varying the parameters of each attack to apply different attack intensities to the networks. The performance of NN1 was then evaluated on these corrupted samples, followed by the evaluation of NN2's performance to study the transferability of the attacks. It was found that NN1's accuracy drops significantly on adversarial samples generated by all attacks. NN2 also shows low accuracy on samples corrupted with PGD, BIM, and FGSM, indicating significant transferability of these attacks between the two networks. However, attacks generated with DeepFool are not transferable, while those generated with CW show partial transferability.

A defense mechanism was subsequently implemented for NN1, introducing a detector in front of the network to identify and filter out adversarial samples before they can compromise NN1. The performance of the overall model, consisting of the detector and NN1, shows that the defense is effective against gradient-based attacks, but not against DeepFool attacks, where most of the corrupted samples manage to deceive the detector. Regarding CW attacks, a good number of samples are filtered out by the detector, but this is not sufficient to achieve high performance.

The results highlight the importance of implementing effective defense measures to ensure the robustness of facial recognition systems against adversarial threats, while also emphasizing the ongoing challenges in improving protection against sophisticated attacks like DeepFool and CW.

5.2 Future Developments

- Due to time constraints, a training dataset of approximately 20,000 samples was used for the Detector. This limited number of samples was sufficient to initiate model training and achieve promising preliminary results. However, to further improve classifier

performance, increasing the dataset size is possible. A larger dataset would enhance the model's generalization ability, reduce the risk of overfitting, and increase the accuracy and robustness of the Detector on unseen data.

- Due to limited computational resources, high values of certain parameters were not tested in attack generation. For instance, in the DeepFool attack, the maximum value of max_iter used was 10, as higher values required significantly more time to generate corrupted samples. It might be considered to increase this value to assess any potential decrease in performance on NN2.