Università degli Studi di Salerno

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Corso di Laurea Magistrale in Ingegneria Informatica

Artificial Intelligence for Cybersecurity

Project Work

Security evaluation of a speaker recognition system

Group n. **05**

Surname and Name	Id
Buonaiuto Mariassunta	0622701677
Candela Raffaele	0622701673
Clarizia Emilio	0622701667
Paolillo Alessandro	0622701812

Academic Year 2022-2023

Summary

1 Executive summary

The objective of this project is the implementation of an access control system based on speech recognition and the subsequent evaluation of its security and performance.

It was necessary to train and validate two binary classifiers (one as a white box and the other as a black box) capable of recognizing the audio of the authorized user (positive class) and those of all other users (negative class).

A dataset was provided from which to select a set of uncorrupted audios to be divided into training, validation, and test sets. It was also required to select the identity of a user to be used as the reference class (the authorized user).

After training and validation of the classifiers on the uncorrupted audios, it was planned to generate adversary samples on one of the two classifiers (chosen as white box classifier) and then to evaluate this classifier on these corrupted samples as well as the transferability of the attack (adversary samples) on the other classifier (chosen as black box).

Thus, it was necessary to implement and evaluate at least one defense mechanism to try to limit the effects of the attack.

1.1 Dataset

Regarding the choices made on the data, we started from a dataset containing 4874 audios of 40 different identities. We chose as the target identity the one with id=10300, corresponding to Ernest Borgnine, because of his nationality (USA), which is the one most present in the dataset, and also because of the number of his available audios, which allowed us to create a balanced dataset. In fact, 304 audios of Ernest Borgnine were available (to be labeled as positive) and taking 8 audios for each other identity resulted in 312 audios (to be labeled as negative).

Once this new dataset was created, it was divided into a Train part (70 percent of the samples) and a Validation part (30 percent of the samples).

To perform the attacks, a Test-set was created consisting of another 78 samples of the negative class (2 for each person other than Ernest), randomly chosen (excluding those already included in the other sets): the samples of this class will be modified through the ART library attacks, to try to trick the classifier into passing them off as elements of the target class.

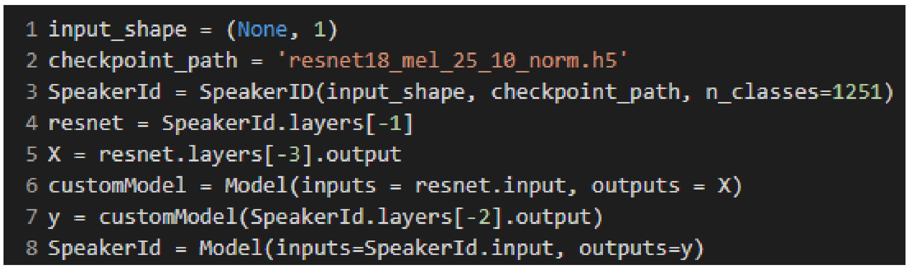
No Data Augmentation technique was performed since the purpose of the project is to evaluate classifier attacks, and augmentation could have introduced a robustness of the classifier against attacks.

1.2 Pre-processing, Features extraction and Post-Processing

Initially, the audios were resized to (130118, 1) so that they could all have the same length (average length of the samples in the dataset) and features could be extracted with the provided network (SpeakerId).

The SpeakerID model was modified in such a way that features could be easily extracted. This was done by removing the last layers of ResNet18 (the last part that makes up the SpeakerId model), so that this change would have the effect that the network itself provides with the include_top = False option. When include_top = False in a machine learning model, it means that you are excluding the fully connected layer (top layer) at the end of the model and keeping only the lower layers that are used for feature extraction. By excluding the top layer, you can extract features from the inputs instead of making predictions using the entire model. This allows the lower levels of the model to be used as a fixed feature extractor, where the extracted features can then be inserted into a new model for training.

However, the include_top = False option was not usable directly within the SpeakerId model construction because the weights of a checkpoint_path (resnet18_mel_25_10_norm.h5) were loaded via the load_weights function.



```python
input_shape = (None, 1)
checkpoint_path = 'resnet18_mel_25_10_norm.h5'
SpeakerId = SpeakerID(input_shape, checkpoint_path, n_classes=1251)
resnet = SpeakerId.layers[-1]
X = resnet.layers[-3].output
customModel = Model(inputs = resnet.input, outputs = X)
y = customModel(SpeakerId.layers[-2].output)
SpeakerId = Model(inputs=SpeakerId.input, outputs=y)
```

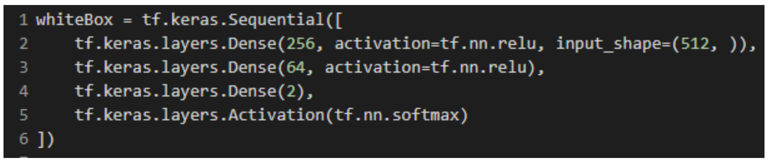
The feature vector extracted from this network has 512 elements that were normalized because, since they no longer represented the waveforms of the original audio, they had ranges of values in an inaccurate range.
The features were then brought into the range (0, 1) to make them comparable and to have the same range in which to perform fits on different samples.
The type of normalization used is as follows (MinMax):

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

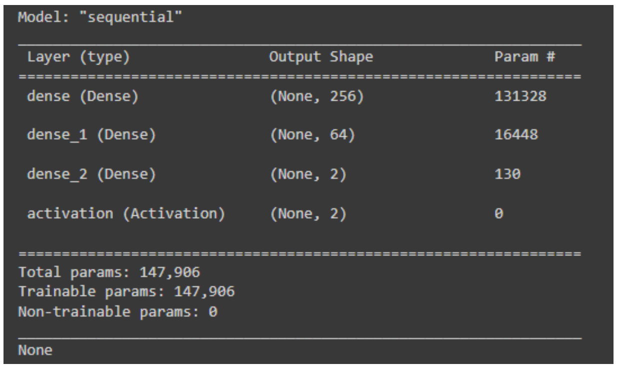
1.3 Classifiers training and evaluation

The classifiers chosen were two different MLPs: a first MLP (WhiteBox) as the classifier on which to perform the attacks and a second MLP (BlackBox) as the classifier on which to evaluate the transferability of the same attacks.
The problem in question was treated, not as a binary problem but as a 2-class multi-class problem: this choice is since some attacks of the ART library were not implemented to be used on binary classifiers.
This small change had no side effects: the use of categorical_crossentropy instead of binary_crossentropy and the use of 2 output neurons instead of 1 did not produce changes in the performance of the network.

1.3.1 White Box



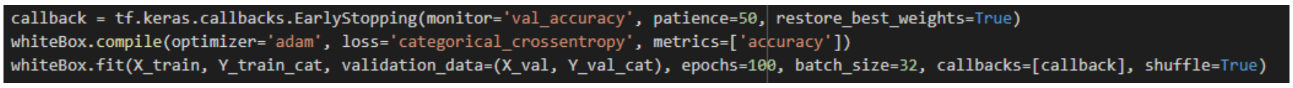
```
whiteBox = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation=tf.nn.relu, input_shape=(512, )),
    tf.keras.layers.Dense(64, activation=tf.nn.relu),
    tf.keras.layers.Dense(2),
    tf.keras.layers.Activation(tf.nn.softmax)
])
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               131328

 dense_1 (Dense)             (None, 64)                16448

 dense_2 (Dense)             (None, 2)                 130

 activation (Activation)     (None, 2)                 0

=================================================================
Total params: 147,906
Trainable params: 147,906
Non-trainable params: 0
_________________________________________________________________
None
```

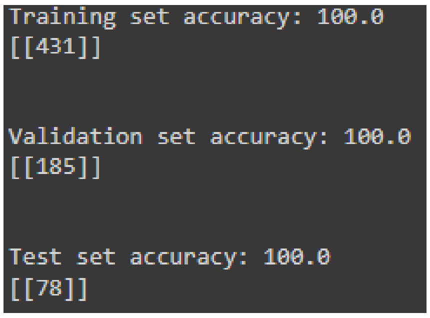
Given this network architecture, different tests were made for the training of this classifier.
For each training, the EarlyStopping callback based on the accuracy produced on the validation set was used, with a patience of 50 epochs. Furthermore, as previously mentioned, the loss function used is the categorical_crossentropy and all the trainings provided a batch size of 32 samples and a maximum number of epochs equal to 100: all the tests triggered the callback within the 100 epochs. The different tests were done with the following optimizers: [adagrad, adam, rmsprop].

The chosen network was achieved with the following configuration:



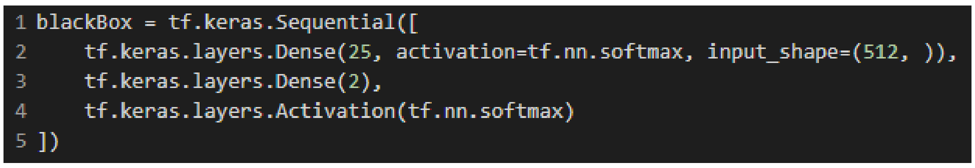
```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=50, restore_best_weights=True)
whiteBox.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
whiteBox.fit(X_train, Y_train_cat, validation_data=(X_val, Y_val_cat), epochs=100, batch_size=32, callbacks=[callback], shuffle=True)
```

The performances achieved are as follows:

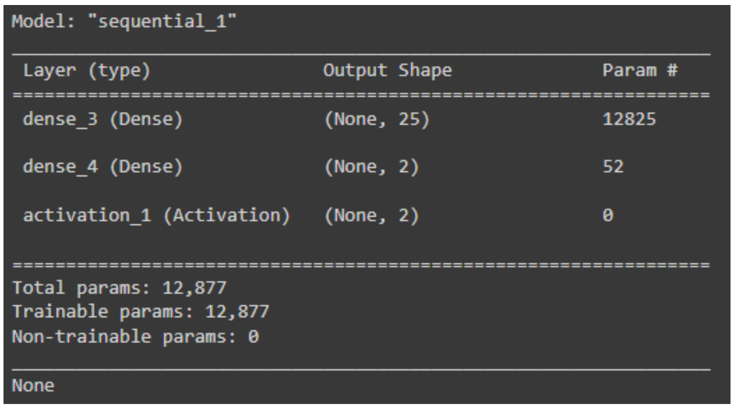


```
Training set accuracy: 100.0
[[431]]


Validation set accuracy: 100.0
[[185]]


Test set accuracy: 100.0
[[78]]
```

1.3.2 Black Box



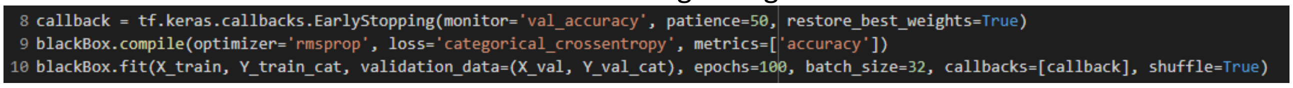
```
blackBox = tf.keras.Sequential([
    tf.keras.layers.Dense(25, activation=tf.nn.softmax, input_shape=(512, )),
    tf.keras.layers.Dense(2),
    tf.keras.layers.Activation(tf.nn.softmax)
])
```



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 25)                12825

 dense_4 (Dense)             (None, 2)                 52

 activation_1 (Activation)   (None, 2)                 0

=================================================================
Total params: 12,877
Trainable params: 12,877
Non-trainable params: 0
_________________________________________________________________
None
```

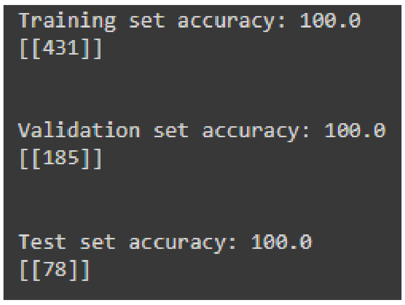
Given this network architecture, different tests were made for the training of this classifier.
For each training, the EarlyStopping callback based on the accuracy produced on the validation set was used, with a patience of 50 epochs. Furthermore, as previously mentioned, the loss function used is the categorical_crossentropy and all the trainings provided a batch size of 32 samples and a maximum number of epochs equal to 100: all the tests triggered the callback within the 100 epochs. The different tests were done with the following optimizers: [adagrad, adam, rmsprop].

The chosen network was achieved with the following configuration:



```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=50, restore_best_weights=True)
blackBox.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
blackBox.fit(X_train, Y_train_cat, validation_data=(X_val, Y_val_cat), epochs=100, batch_size=32, callbacks=[callback], shuffle=True)
```

The performances achieved are as follows:



```
Training set accuracy: 100.0
[[431]]


Validation set accuracy: 100.0
[[185]]


Test set accuracy: 100.0
[[78]]
```

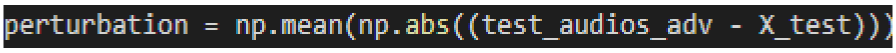
The models were created so that the activation layer could be easily separated to access the logits of the previous layer that are used by DeepFool and Carlini - Wagner attacks designed on a classifier that uses logits.

2 Attacks

The attacks used to generate the adversarial examples are all part of the Adversarial Robustness Toolbox (ART) library.
As for the gradient attacks (FGSM, BIM, PGD), they were applied on a KerasClassifier, i.e., the ART class used to create a classifier on which to apply the attack from the model, with the white box model passed through the model attribute and the clip values attribute set to (0,1). This attribute is a tuple of the form (min, max) representing the minimum and maximum values allowed for features.

As for DeepFool and Carlini - Wagner attacks, they were applied to a KerasClassifier with the white box model deprived of its last layer passed through the model attribute and the clip values attribute set to (0,1).

The SEC was constructed to assess the effectiveness of attacks by evaluating both attack accuracy and classifier accuracy as the attack strength varied. Attack strength is represented by a single parameter of the specific attack or by the mean perturbation if multiple parameters are present; the mean perturbation is calculated as the average over all samples in the Test Set of the absolute value of the difference between the original sample and the post-attack sample:

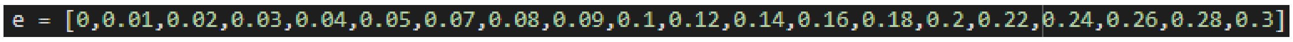


```
perturbation = np.mean(np.abs((test_audios_adv - X_test)))
```

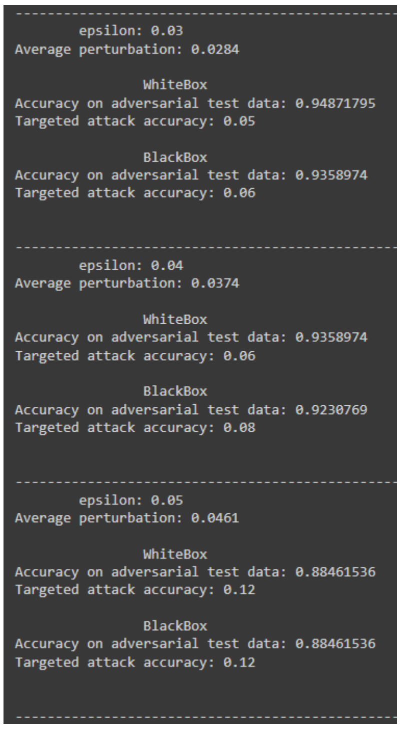
Obviously, since the Test Set consists only of samples of the negative class, the accuracy of the attack is complementary to the accuracy of the classifier (their sum is always equal to 1).
In evaluating the attacks, leaving out FGSM for which graphs had already been produced, we chose to keep only the accuracy of the classifier for ease of reading.

2.1 FGSM

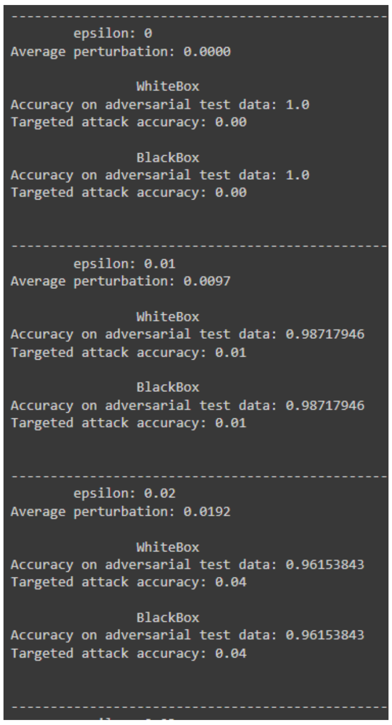
In this attack, the strength is represented by the increasing **epsilon** (e) value, directly proportional to the average perturbation. The tested values vary in the interval (0, 1) but, noting that the stall point is reached with epsilon (e) equal to 0.2, only the epsilon values in the interval (0, 0.3) have been reported in such a way to make the accuracy trend more visible as the strength of the attack varies. We set the targeted value to True since we have the authorized user as target class.



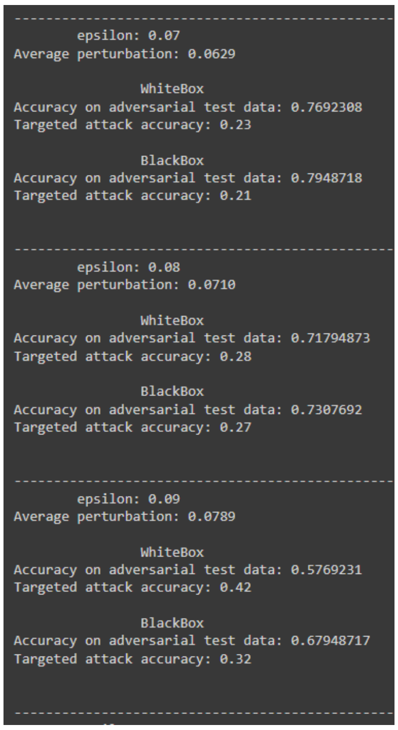
```
e = [0,0.01,0.02,0.03,0.04,0.05,0.07,0.08,0.09,0.1,0.12,0.14,0.16,0.18,0.2,0.22,0.24,0.26,0.28,0.3]
```



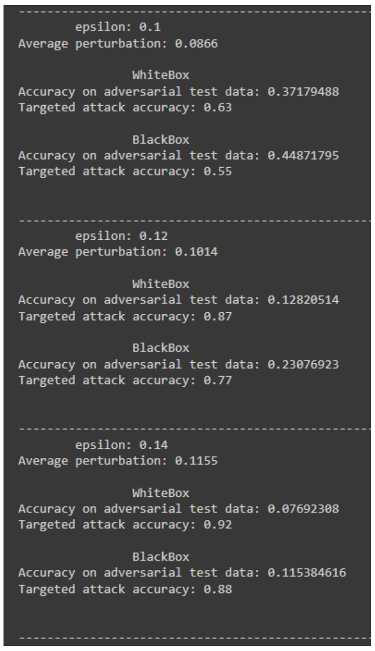
```
-----------------------------------------------
        epsilon: 0.03
Average perturbation: 0.0284

                WhiteBox
Accuracy on adversarial test data: 0.94871795
Targeted attack accuracy: 0.05

                BlackBox
Accuracy on adversarial test data: 0.9358974
Targeted attack accuracy: 0.06


-----------------------------------------------
        epsilon: 0.04
Average perturbation: 0.0374

                WhiteBox
Accuracy on adversarial test data: 0.9358974
Targeted attack accuracy: 0.06

                BlackBox
Accuracy on adversarial test data: 0.9230769
Targeted attack accuracy: 0.08


-----------------------------------------------
        epsilon: 0.05
Average perturbation: 0.0461

                WhiteBox
Accuracy on adversarial test data: 0.88461536
Targeted attack accuracy: 0.12

                BlackBox
Accuracy on adversarial test data: 0.88461536
Targeted attack accuracy: 0.12


-----------------------------------------------
```



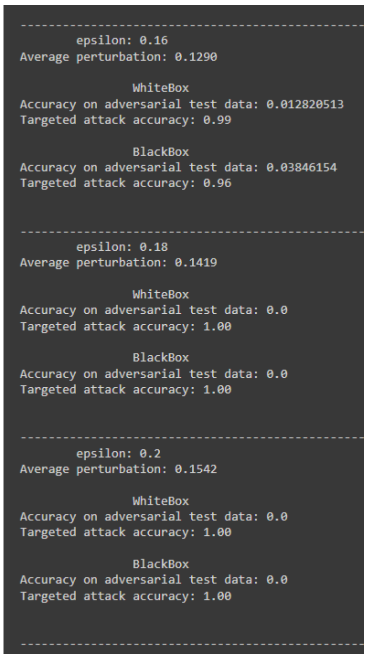
```
-----------------------------------------------
        epsilon: 0
Average perturbation: 0.0000

                WhiteBox
Accuracy on adversarial test data: 1.0
Targeted attack accuracy: 0.00

                BlackBox
Accuracy on adversarial test data: 1.0
Targeted attack accuracy: 0.00


-----------------------------------------------
        epsilon: 0.01
Average perturbation: 0.0097

                WhiteBox
Accuracy on adversarial test data: 0.98717946
Targeted attack accuracy: 0.01

                BlackBox
Accuracy on adversarial test data: 0.98717946
Targeted attack accuracy: 0.01


-----------------------------------------------
        epsilon: 0.02
Average perturbation: 0.0192

                WhiteBox
Accuracy on adversarial test data: 0.96153843
Targeted attack accuracy: 0.04

                BlackBox
Accuracy on adversarial test data: 0.96153843
Targeted attack accuracy: 0.04


-----------------------------------------------
```



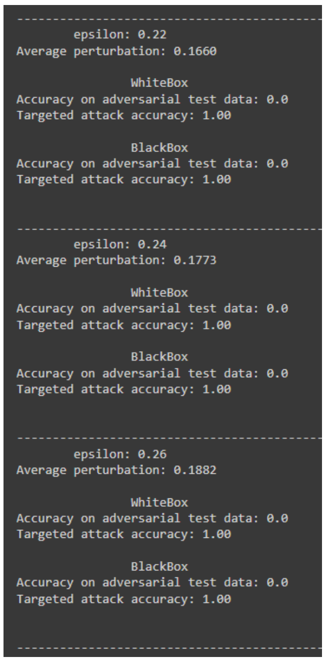
```
-----------------------------------------------
        epsilon: 0.07
Average perturbation: 0.0629

                WhiteBox
Accuracy on adversarial test data: 0.7692308
Targeted attack accuracy: 0.23

                BlackBox
Accuracy on adversarial test data: 0.7948718
Targeted attack accuracy: 0.21


-----------------------------------------------
        epsilon: 0.08
Average perturbation: 0.0710

                WhiteBox
Accuracy on adversarial test data: 0.71794873
Targeted attack accuracy: 0.28

                BlackBox
Accuracy on adversarial test data: 0.7307692
Targeted attack accuracy: 0.27


-----------------------------------------------
        epsilon: 0.09
Average perturbation: 0.0789

                WhiteBox
Accuracy on adversarial test data: 0.5769231
Targeted attack accuracy: 0.42

                BlackBox
Accuracy on adversarial test data: 0.67948717
Targeted attack accuracy: 0.32


-----------------------------------------------
```



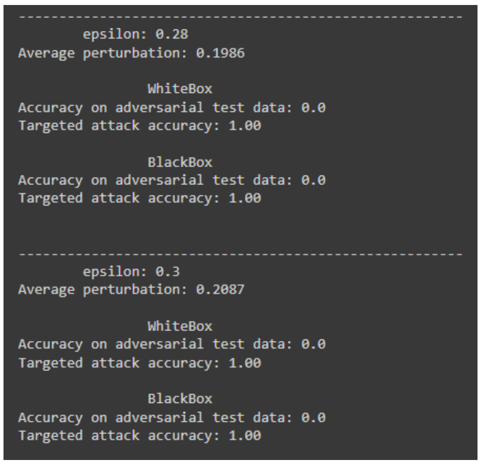
```
-----------------------------------------------
        epsilon: 0.1
Average perturbation: 0.0866

                WhiteBox
Accuracy on adversarial test data: 0.37179488
Targeted attack accuracy: 0.63

                BlackBox
Accuracy on adversarial test data: 0.44871795
Targeted attack accuracy: 0.55


-----------------------------------------------
        epsilon: 0.12
Average perturbation: 0.1014

                WhiteBox
Accuracy on adversarial test data: 0.12820514
Targeted attack accuracy: 0.87

                BlackBox
Accuracy on adversarial test data: 0.23076923
Targeted attack accuracy: 0.77


-----------------------------------------------
        epsilon: 0.14
Average perturbation: 0.1155

                WhiteBox
Accuracy on adversarial test data: 0.07692308
Targeted attack accuracy: 0.92

                BlackBox
Accuracy on adversarial test data: 0.115384616
Targeted attack accuracy: 0.88


-----------------------------------------------
```

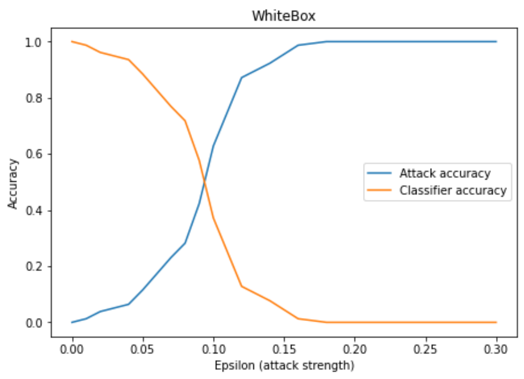


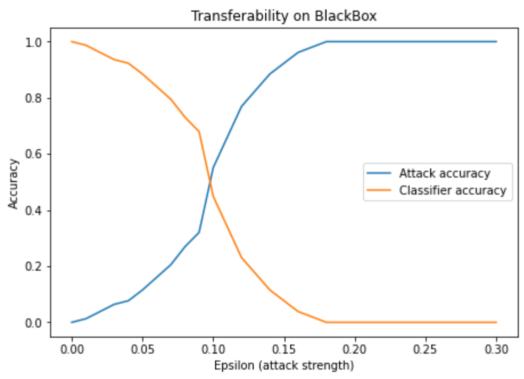
```
-----------------------------------------------
        epsilon: 0.16
Average perturbation: 0.1290

                WhiteBox
Accuracy on adversarial test data: 0.012820513
Targeted attack accuracy: 0.99

                BlackBox
Accuracy on adversarial test data: 0.03846154
Targeted attack accuracy: 0.96


-----------------------------------------------
        epsilon: 0.18
Average perturbation: 0.1419

                WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

                BlackBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00


-----------------------------------------------
        epsilon: 0.2
Average perturbation: 0.1542

                WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

                BlackBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00


-----------------------------------------------
```

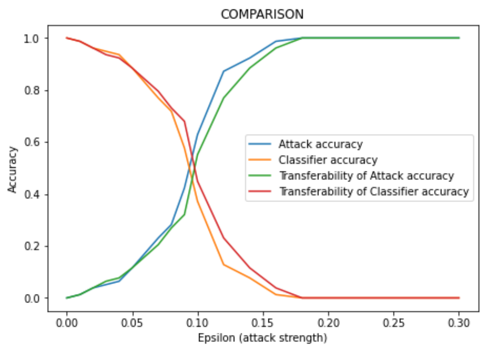


```
-----------------------------------------------
        epsilon: 0.22
Average perturbation: 0.1660

                WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

                BlackBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00


-----------------------------------------------
        epsilon: 0.24
Average perturbation: 0.1773

                WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

                BlackBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00


-----------------------------------------------
        epsilon: 0.26
Average perturbation: 0.1882

                WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

                BlackBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00


-----------------------------------------------
```


```
------------------------------------------------------------
        epsilon: 0.28
Average perturbation: 0.1986

                WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

                BlackBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00


------------------------------------------------------------
        epsilon: 0.3
Average perturbation: 0.2087

                WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

                BlackBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00
```







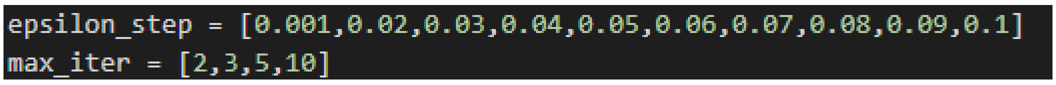
Since an MLP network takes advantage of the gradient in training, it was expected from the outset that gradient-based attacks could be very effective on the WhiteBox. It can be noted that the average perturbation, as already mentioned, follows the epsilon parameter hand in hand.
It is also noted that the attack moves with an almost identical trend on the BlackBox classifier, the reason for this could be given by the fact that the classifiers have in any case been trained on the same features of the same audios and it is also an MLP network.

2.2 BIM

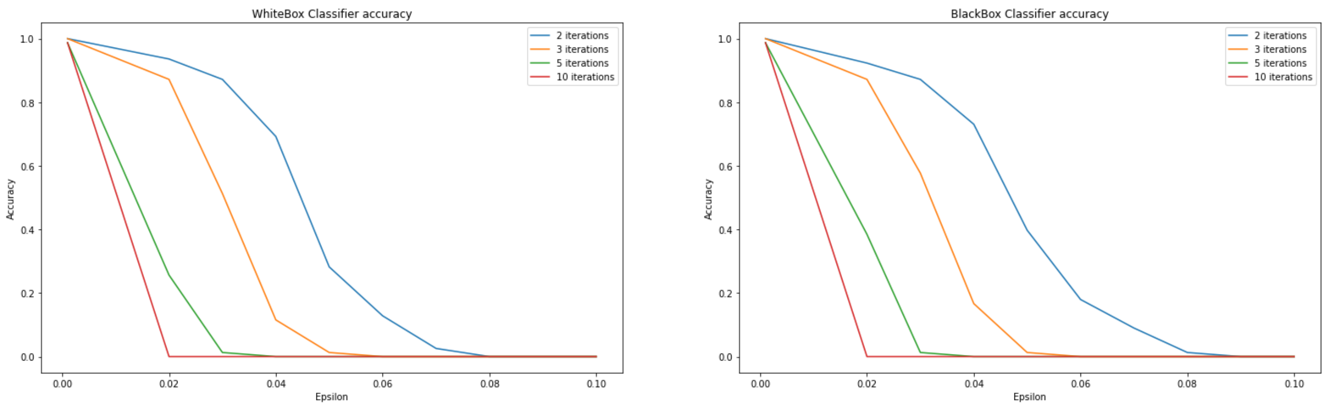
The Basic Iterative Method attack is the iterative version of FGSM.
In this attack the strength is represented by the parameters:

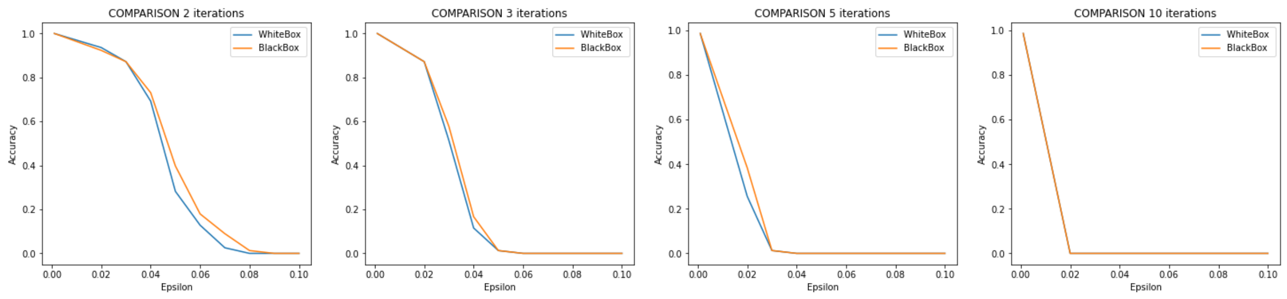
- **e** indicates the maximum applicable perturbation, 1 was chosen in order not to have limits on the maximum applicable perturbation (the values will however remain in the range (0, 1)).
- **epsilon_step** indicates the perturbation applied to each iteration.
- **max_iter** indicates the number of iterations.

To test this attack, all the combinations that foresee an epsilon_step between (0, 1) and max_iter in [2, 3, 5, 10] have been tried. It was noted that the stall point was reached with epsilon equal to 0.1, consequently only the epsilon values in the interval (0, 0.1) were reported, to make the trend of the graphs more understandable.

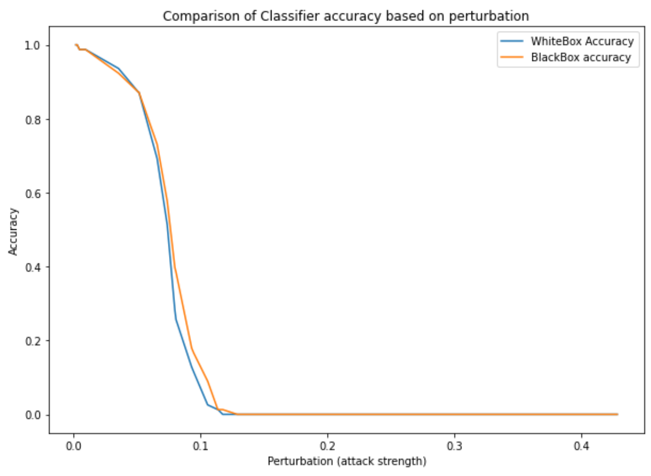


```
epsilon_step = [0.001,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1]
max_iter = [2,3,5,10]
```





Unlike FGSM you need a lower epsilon to completely collapse the performance of the classifier. BIM performs more iterations with a given epsilon and obviously the more iterations, the smaller the epsilon can be to have excellent results.
Given that the strength of the attack in this case is dependent on both epsilon and the number of iterations, in order to correctly build an evaluable SEC, it was decided to consider the average perturbation on the samples as the strength of the attack.

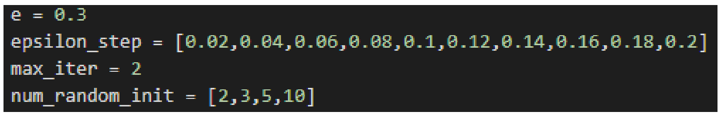


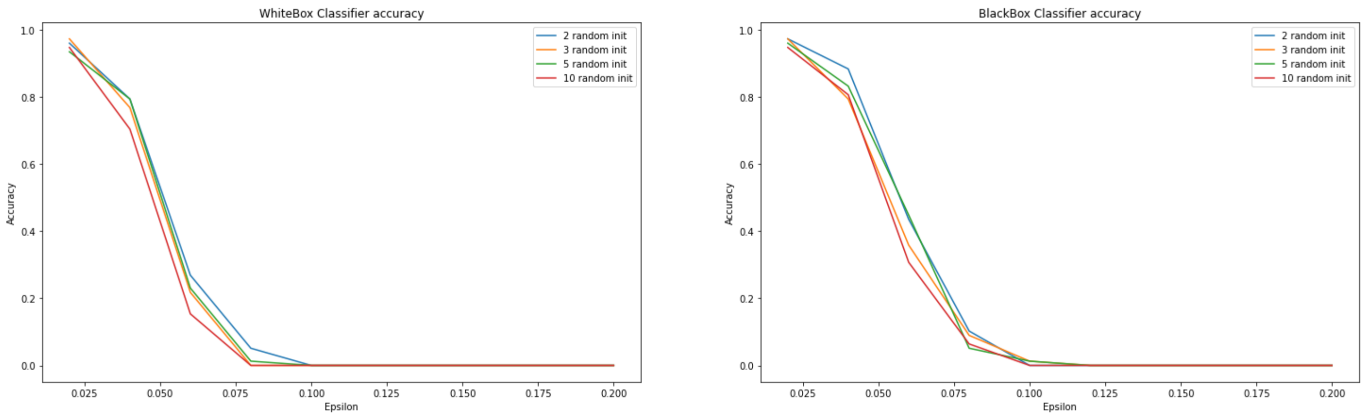
This graph considers all combinations of epsilon and number of iterations, evaluating the average perturbation.
In agreement with the previous attack, performance completely collapses with the average perturbation being around 0.13.
It is also noted that again the attack is transferable: the accuracy of the BlackBox classifier follows closely behind that of WhiteBox.

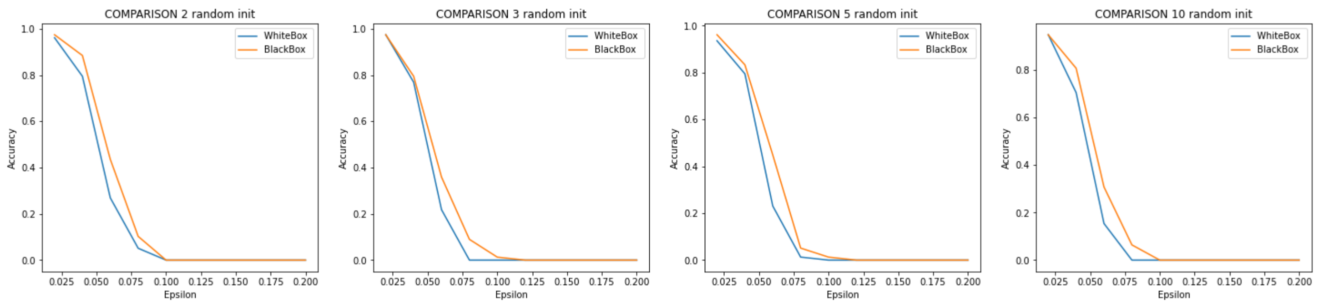
2.3 PGD

In this attack the strength is represented by the parameters:

- **e** indicates the maximum perturbation applicable to the sample and determines the epsilon-ball on which the projection function acts. By acting, unlike BIM, on the epsilon-ball, a too large 'e' produces high perturbations which in some tests led to the failure of the attack. For these reasons, this parameter has been set to 0.3.
- **epsilon_step** indicates the perturbation applied to each iteration. The values tested for this parameter vary in the interval (0, 1) but given that the attack settles around lower values, to make the trend of the SEC more understandable, the interval (0, 0.2) has been reported.
- **max_iter** indicates the number of iterations before the action of the projection function.
- **num_random_init**: Number of random initialisations within the epsilon ball. The tested values are in [2, 3, 5, 10].

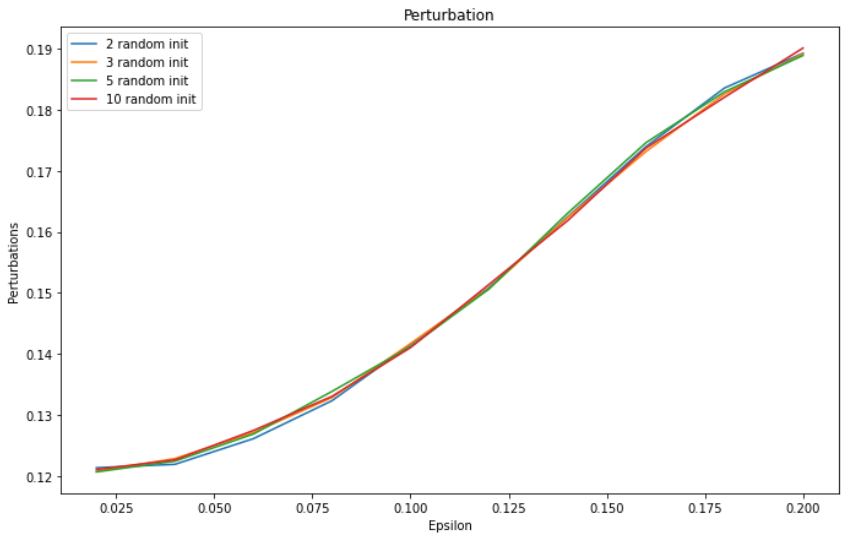

```
e = 0.3
epsilon_step = [0.02,0.04,0.06,0.08,0.1,0.12,0.14,0.16,0.18,0.2]
max_iter = 2
num_random_init = [2,3,5,10]
```

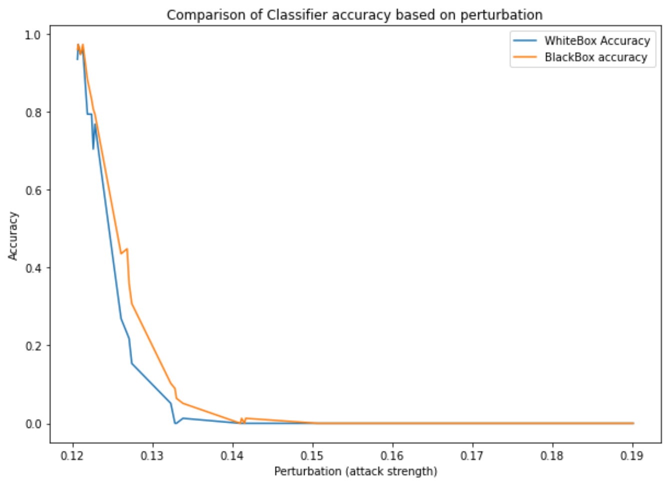




Unlike BIM, a higher number of iterations does not lead to the "destruction" of the network first, but as the number of iterations changes, the trend remains almost identical. This is probably due to the fact of the projection on the epsilon-ball, which inserts a randomness that in this case "worsens" (compared to BIM) the cases in which there are more iterations.

As for BIM, the strength of the attack is dependent on various parameters: to correctly build an evaluable SEC, it was decided to consider the average perturbation on the samples as the strength of the attack. Unlike BIM, however, the randomity added in PGD means that the average perturbation produced as the epsilon_step varies, is very similar for each num_random_init value.

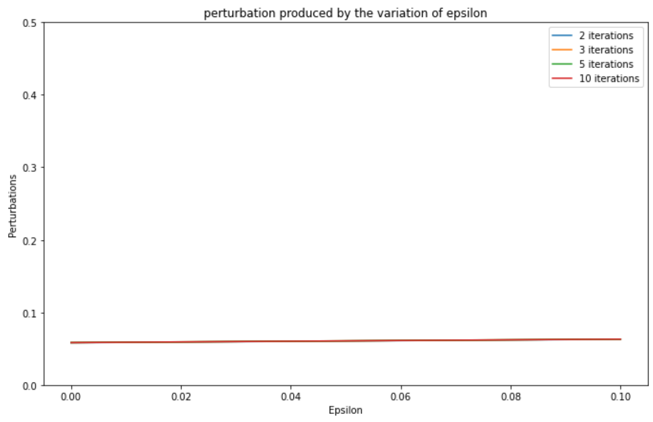


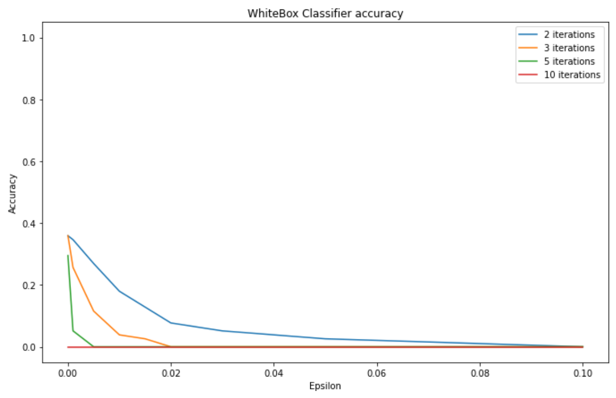


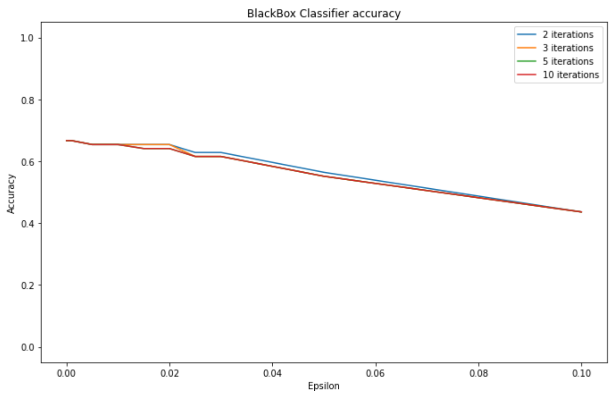
The presence of “jumps” in the graph is because attacks made with different num_random_inits produce the same perturbation but slightly different classifier performance.
It is noted that also in this case, the accuracy of the classifier touches 0 with average perturbations around 0.135, as was the case with both BIM and FGSM.
Also in this case the attack is transferable to the BlackBox classifier.

2.4 DeepFool

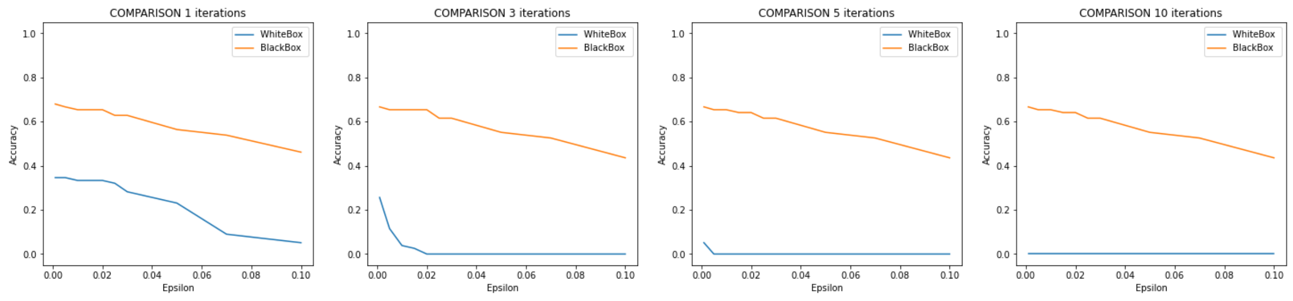
In this attack the strength is represented by the parameters **epsilon** and **max_iter**.
In this case the attacks were applied to a Keras classifier with the white box model deprived of its last layer passed through the model attribute and the clip values attribute set to (0,1).
It was noted that as these parameters varied, the average perturbation always remained constant in a neighborhood of 0.06, despite epsilon having low (0.00001) or high (0.1) values.







As can be seen, the attack in this case is more successful than previous attacks even with a lower perturbation. Using values of a number of iterations equal to 10 allows for an attack that destroys the network even with very low epsilon values.

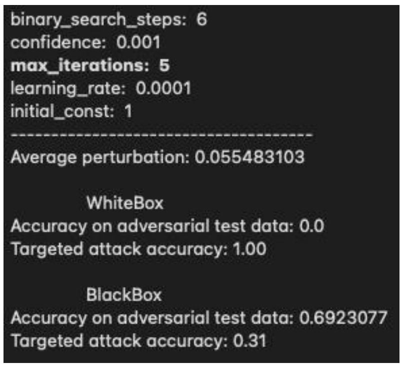


As expected, this type of attack turns out to be less transferable than basic gradient attacks.

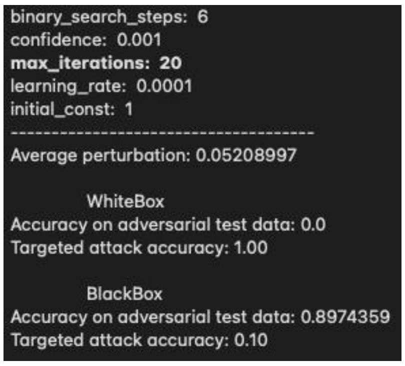
2.5 Carlini-Wagner

There are various types of Carlini-Wagner, which differ according to the type of norm used (L0, L2, Linf). In our case we have chosen CW with L2.
In this case the attacks were applied to a Keras classifier with the white box model deprived of its last layer passed through the model attribute and the clip values attribute set to (0,1).
Initially, for this attack, an experimental parameter tuning was tried to try to understand what the best combinations could be. Once the parameters that returned the best values among the tests made were ascertained, some were fixed, keeping only one variable.
The goal in the end was to produce an attack that would destroy the network, trying to produce the SEC based on the variable parameter.
It was not possible to try all the combinations as this attack takes a long time: some tests took up to 45 minutes to perform.

First has been investigated the relationship between **learning_rate** and **max_iterations**: the two parameters are obviously linked as a lower learning rate needs more iterations to reach convergence and vice versa.



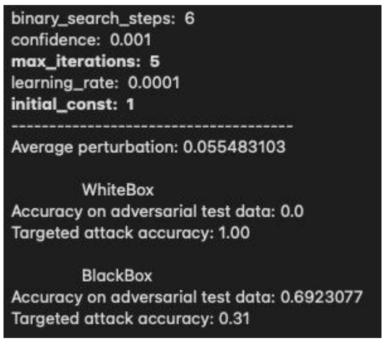
```
binary_search_steps:  6
confidence:  0.001
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.055483103

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.6923077
Targeted attack accuracy: 0.31
```



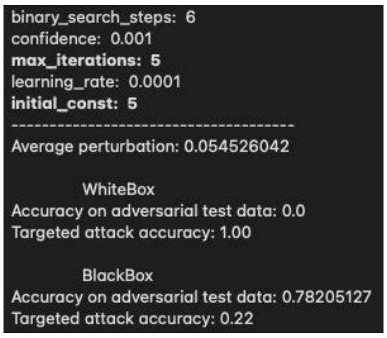
```
binary_search_steps:  6
confidence:  0.001
max_iterations:  20
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.05208997

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.8974359
Targeted attack accuracy: 0.10
```

From the first experiments it was noted that, keeping the learning rate around 0.0001, 5 iterations (max_iterations) were sufficient. Increasing the value of max_iter showed how the performance (the success of the attack) of the BlackBox model worsened.

Subsequent tests were done to evaluate the usefulness of the **initial_const** parameter, and it was noted that although the value was set to 1 the attack was executed correctly.



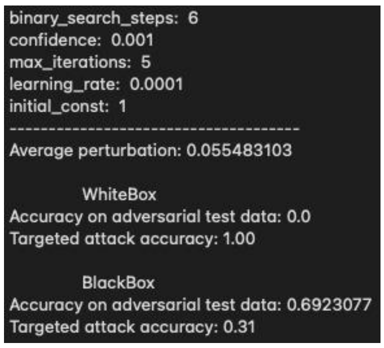
```
binary_search_steps:  6
confidence:  0.001
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.055483103

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.6923077
Targeted attack accuracy: 0.31
```



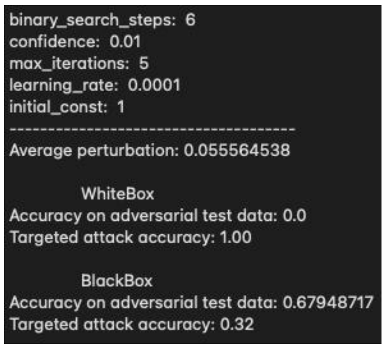
```
binary_search_steps:  6
confidence:  0.001
max_iterations:  5
learning_rate:  0.0001
initial_const:  5
------------------------------------
Average perturbation: 0.054526042

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.78205127
Targeted attack accuracy: 0.22
```

Increasing the value of initial_const showed how the performance (the success of the attack) of the BlackBox model worsened.

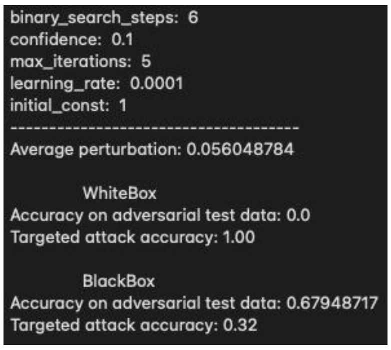
To further improve the situation, the **confidence** parameter was changed: it was seen that by taking it to 0.9, the attack maintained the same performance on WhiteBox (with a similar mean perturbation) and improved significantly on BlackBox.



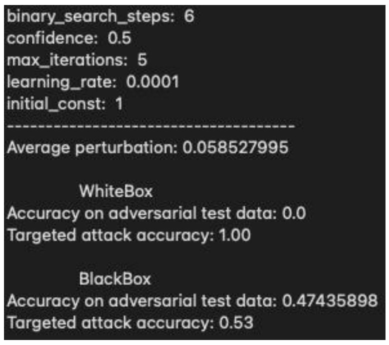
```
binary_search_steps:  6
confidence:  0.001
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.055483103

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.6923077
Targeted attack accuracy: 0.31
```



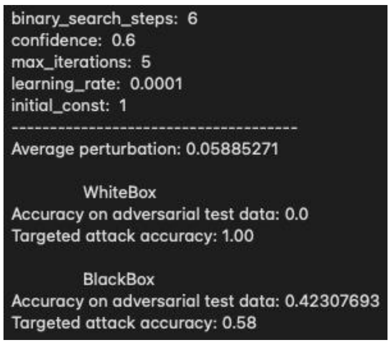
```
binary_search_steps:  6
confidence:  0.01
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.055564538

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.67948717
Targeted attack accuracy: 0.32
```



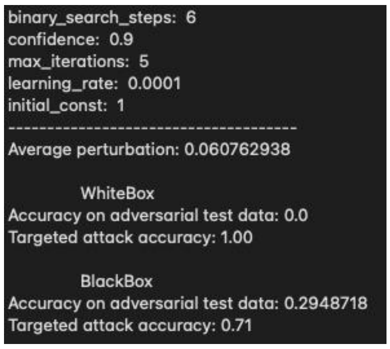
```
binary_search_steps:  6
confidence:  0.1
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.056048784

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.67948717
Targeted attack accuracy: 0.32
```



```
binary_search_steps:  6
confidence:  0.5
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.058527995

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.47435898
Targeted attack accuracy: 0.53
```

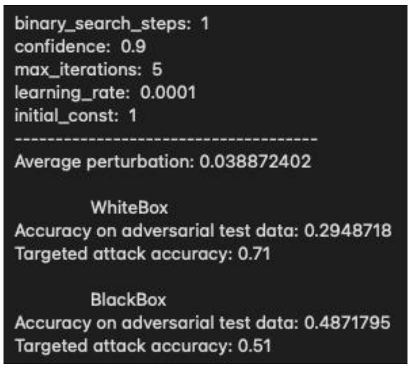


```
binary_search_steps:  6
confidence:  0.6
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.05885271

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.42307693
Targeted attack accuracy: 0.58
```



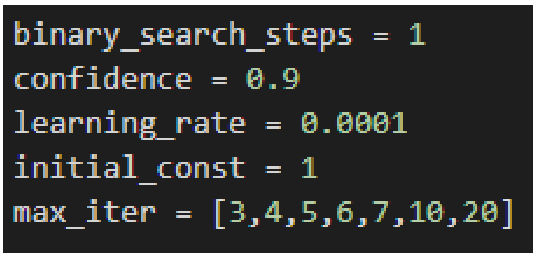
```
binary_search_steps:  6
confidence:  0.9
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.060762938

           WhiteBox
Accuracy on adversarial test data: 0.0
Targeted attack accuracy: 1.00

           BlackBox
Accuracy on adversarial test data: 0.2948718
Targeted attack accuracy: 0.71
```

It was later noted that after setting the value of the confidance parameter, the **binary_search_step** parameter could take the value 1 to achieve a performant attack.

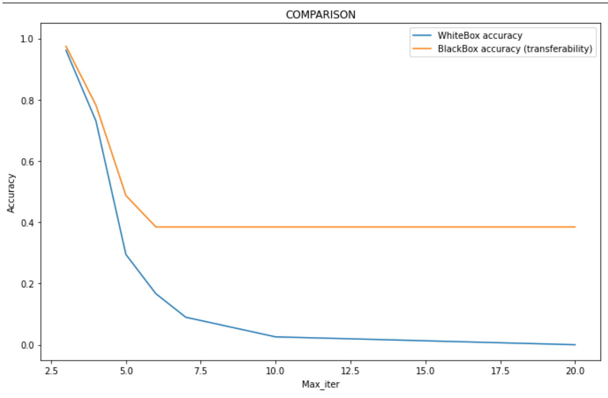


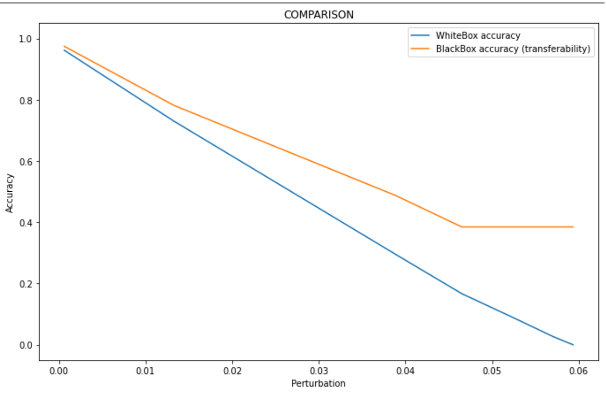
```
binary_search_steps:  1
confidence:  0.9
max_iterations:  5
learning_rate:  0.0001
initial_const:  1
------------------------------------
Average perturbation: 0.038872402

           WhiteBox
Accuracy on adversarial test data: 0.2948718
Targeted attack accuracy: 0.71

           BlackBox
Accuracy on adversarial test data: 0.4871795
Targeted attack accuracy: 0.51
```

It was noted that what affects the attack is the max_iter parameter. Therefore, it was decided to make this parameter variable, to create the SEC, because it achieves a good compromise between average applied perturbation and attack accuracy.



```
binary_search_steps = 1
confidence = 0.9
learning_rate = 0.0001
initial_const = 1
max_iter = [3,4,5,6,7,10,20]
```





It is certainly possible to make a better tuning of the parameters, leading to a decrease in the average perturbation but given the high timing in the execution of the tests, this result seemed satisfactory anyway. Note that it is the attack that manages to completely break down the performance of the classifier by applying the smallest average perturbation in comparison to the other attacks (0.059). This was not an unexpected result as this attack just tries to optimize this.
It was noticed that the attack was less transferable than the gradient-based attacks: the Carlini-Wagner attack that destroyed the WhiteBox classifier was not performing in the same way on the BlackBox.

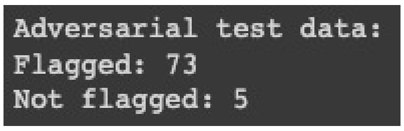
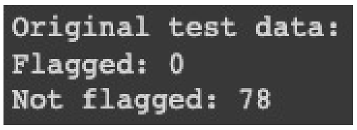
3 Defenses

To guarantee a robust defense to our WhiteBox model, it was decided to use adversary sample detectors as defenses to be able to distinguish adversary samples and discard them if they are fed to our system. After individually training one detector for each attack, they were put together into an ensemble of detectors. Then, the model inputs will be a mix of original data and adversary data, and then only those that the detector ensemble considers original will be filtered out. To ensure the highest accuracy in detecting attacks, the unanimity of detectors was considered.

The training was performed using data as generic as possible, from attacks as generic as possible, made using various ranges of parameters generated. For detector training, we used original training set plus modified training set (which in turn was divided by the number of attacks and each subset was modified with one attack), so in the end we will have a training set consisting of twice as many data of the original training set (half original ones and half modified ones).
For the detectors, we did not use the original labels, but new labels were created, to be precise label 0 was assigned to the original samples and label 1 to the modified ones. Before the training of the detectors, the samples were shuffled (with random state set to 42 to be able to repeat the exact same shuffling). The evaluation of the single detectors was done on: adversarial data (adversarial samples created using the aforementioned attacks) and original samples (the samples of the test set, obviously not used during training).

3.1 FGSM

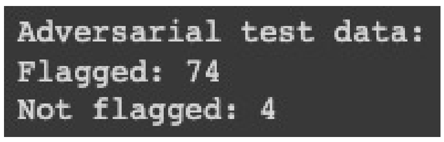
For the generation of training adversarial samples using the FGSM attack, we used 20 epsilon values in the range 0 through 0.3. This was made to generalize as much as possible, starting from a small perturbation (0.01) up to an acceptable perturbation (0.3).
The detector used is a RandomForestClassifier with n_estimators = 150. Training was carried out using both the original samples and those modified with the FGSM attack with the various espisoln values in such a way as to make the classification more robust.


```
Adversarial test data:
Flagged: 73
Not flagged: 5
```

```
Original test data:
Flagged: 0
Not flagged: 78
```

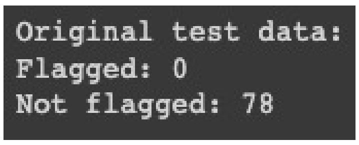
As can be seen, the detector trained on the previously created data manages to perform well both on the "original" test set consisting of the non-flagged samples (it recognizes all samples as non-flagged) and on the test set consisting of adversarial samples (it recognizes 73 out of 78 samples as flagged). The adversarial samples classified as not flagged by the detector are those characterized by very low perturbation (In fact, some of these cannot even fool the WhiteBox).

3.2 BIM

For the generation of training adversarial samples using the BIM attack we chose the epsilon values equals to 1 (in order not to have limits on the maximum applicable perturbation), 10 epsilon steps between 0.001 and 0.1 and 4 max_iter values equals to [2,3,5,10], for a total of 40 attacks.
The detector used is a RandomForestClassifier with n_estimators = 150. Training was carried out using both the original samples and those modified with the BIM attack with the various parameters in such a way as to make the classification more robust.



```
Adversarial test data:
Flagged: 74
Not flagged: 4
```

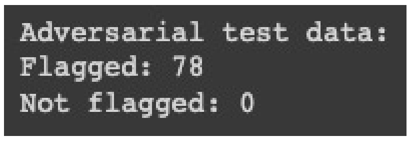


```
Original test data:
Flagged: 0
Not flagged: 78
```

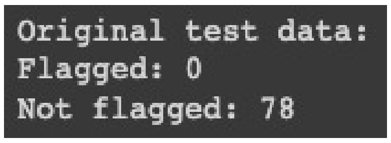
As can be seen, the detector trained on the previously created data manages to perform well both on the "original" test set consisting of the non-flagged samples (it recognizes all samples as non-flagged) and on the test set consisting of adversarial samples (it recognizes 74 out of 78 samples as flagged). The adversarial samples classified as not flagged by the detector are those characterized by very low perturbation (In fact, some of these cannot even fool the WhiteBox).

3.3 PGD

For the generation of training adversarial samples using the PGD attack we chose epsilon value equals to 0.3 (a too large epsilon produces high perturbations which in some tests led to the failure of the attack), 10 epsilon steps between 0.02 and 0.2, max_iter value equals to 2 (higher values did not produce significant differences but were computationally more onerous) and 4 num random init parameters [2,3,5,10], for a total of 40 attacks.
The detector used is a RandomForestClassifier with n_estimators = 150. Training was carried out using both the original samples and those modified with the PGD attack with the various parameters in such a way as to make the classification more robust.



```
Adversarial test data:
Flagged: 78
Not flagged: 0
```

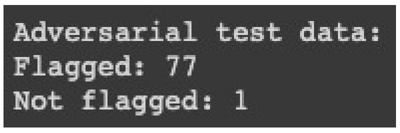


```
Original test data:
Flagged: 0
Not flagged: 78
```

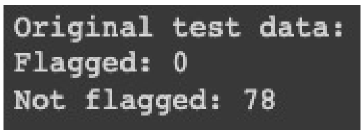
As can be seen, the detector trained on the previously created data manages to perform well both on the "original" test set consisting of the non-flagged samples (it recognizes all samples as non-flagged) and on the test set consisting of adversarial samples (it recognizes all samples as flagged).

3.4 DeepFool

For the generation of training adversarial samples using the DeepFool attack we chose 10 epsilon values between 0.000001 and 0.9 and 4 max iter values [2,3,5,10].
The detector used is a RandomForestClassifier with n_estimators = 150. Training was carried out using both the original samples and those modified with the DeepFool attack with the various parameters in such a way as to make the classification more robust.



```
Adversarial test data:
Flagged: 77
Not flagged: 1
```

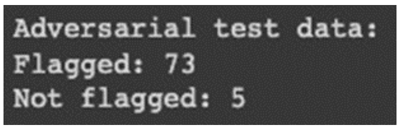


```
Original test data:
Flagged: 0
Not flagged: 78
```

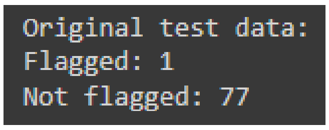
As can be seen, the detector trained on the previously created data manages to perform well both on the "original" test set consisting of the non-flagged samples (it recognizes all samples as non-flagged) and on the test set consisting of adversarial samples (it recognizes 77 out of 78 samples as flagged).

3.5 Carlini-Wagner

For the generation of training adversarial samples using the Carlini-Wagner attack we chose the binary_search_step = 1, confidence = 0.9, learning_rate = 0.0001, initial_const = 1 and 7 max_iter values [6,7,10,12,15,18,20].
The detector used is a RandomForestClassifier with n_estimators = 100. Training was carried out using both the original samples and those modified with the DeepFool attack with the various parameters in such a way as to make the classification more robust.



```
Adversarial test data:
Flagged: 73
Not flagged: 5
```



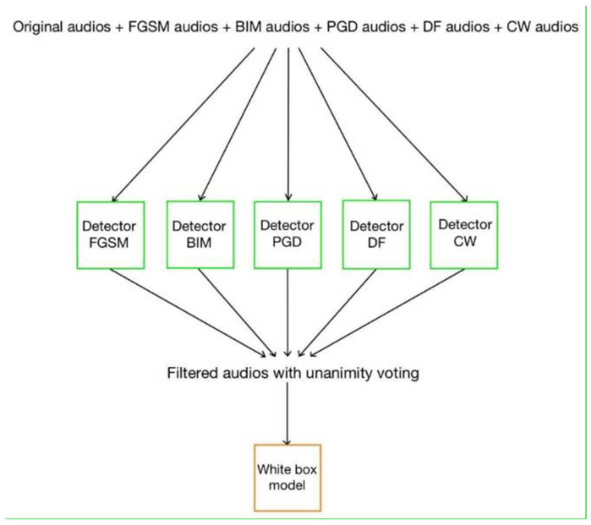
```
Original test data:
Flagged: 1
Not flagged: 77
```

As can be seen, the detector trained on the previously created data manages to perform well both on the "original" test set consisting of the non-flagged samples (it recognizes 77 out of 78 as non-flagged) and on the test set consisting of adversarial samples (it recognizes 73 out of 78 samples as flagged). The adversarial samples classified as not flagged by the detector are those characterized by very low perturbation (In fact, some of these cannot even fool the WhiteBox).

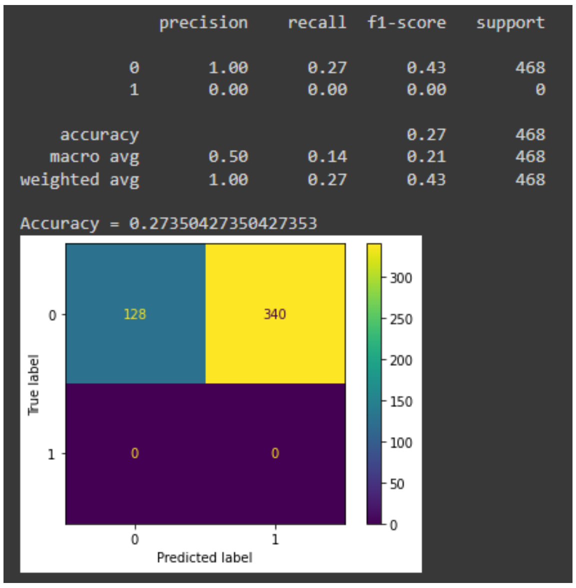
3.6 Total defences

To perform the total defense, we use the original samples (the test set) and the samples generated by the various attacks as inputs to the WhiteBox model. We first evaluate the model without the detector with all data (originals and adversarial), then with the detector to filter the data, and finally with only the original samples (thus only the test set).

To evaluate the white box model and detector, we put together the six test sets (original, modified FGSM samples, modified BIM, modified PGD, modified DeepFool, modified Carlini-Wagner), shuffling it both to create data to give it as input to the model and to give as input to the detector (label 0 for the original audios, label 1 for the adversarial ones).
We give input audios to the ensemble of detectors (composed of the 5 detectors, the FGSM one, the BIM one, the PGD one, the DeepFool one and Carlini-Wagner one) which filter the audios with unanimity voting mechanism: if the totality (so the 5 detectors) classifies an audio as original, then it passes the check and will be given as input to the model, otherwise it will be discarded. In this way we ensure the highest accuracy in detecting attacks.


Original audios + FGSM audios + BIM audios + PGD audios + DF audios + CW audios
Detector FGSM
Detector BIM
Detector PGD
Detector DF
Detector CW
Filtered audios with unanimity voting
White box model

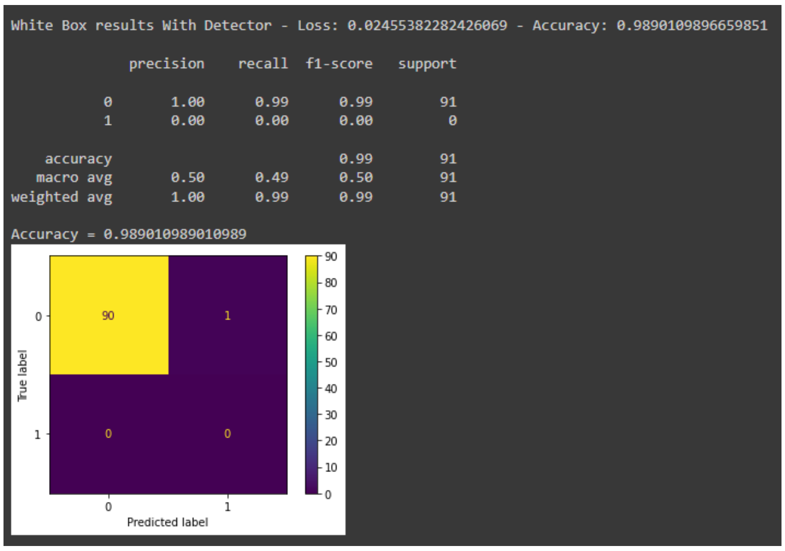
3.6.1 WhiteBox without detector



```
              precision    recall  f1-score   support

           0       1.00      0.27      0.43       468
           1       0.00      0.00      0.00         0

    accuracy                           0.27       468
   macro avg       0.50      0.14      0.21       468
weighted avg       1.00      0.27      0.43       468

Accuracy = 0.27350427350427353
```

Obviously, unsurprisingly, without any defense mechanisms the performance on the model is low.

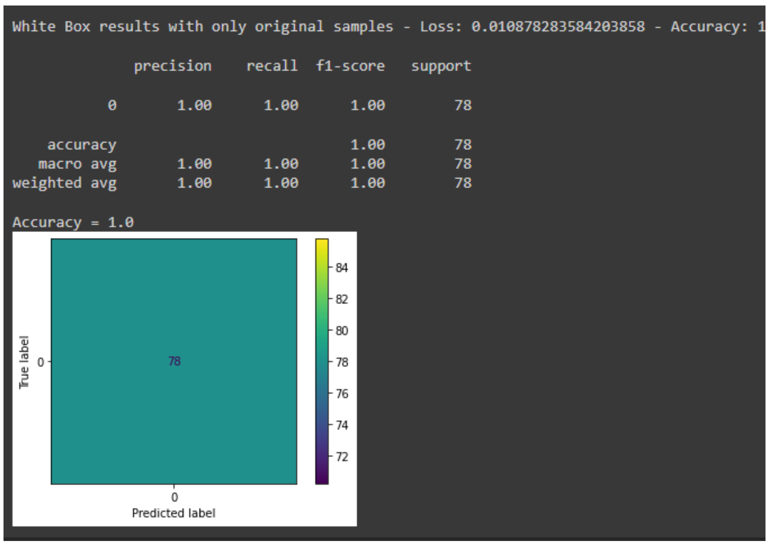
3.6.2 WhiteBox with detector



```
White Box results With Detector - Loss: 0.02455382282426069 - Accuracy: 0.9890109896659851

              precision    recall  f1-score   support

           0       1.00      0.99      0.99        91
           1       0.00      0.00      0.00         0

    accuracy                           0.99        91
   macro avg       0.50      0.49      0.50        91
weighted avg       1.00      0.99      0.99        91

Accuracy = 0.989010989010989
```

The result obtained is not perfect, but it is acceptable. As we can see, some adversarial audio (13) is not recognized by the detectors. This occurs because of their low perturbation. In fact, despite being adversarial audios, 12 of them are classified correctly.

3.6.3 WhiteBox with only original test set



```
White Box results with only original samples - Loss: 0.010878283584203858 - Accuracy: 1

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        78

    accuracy                           1.00        78
   macro avg       1.00      1.00      1.00        78
weighted avg       1.00      1.00      1.00        78

Accuracy = 1.0
```

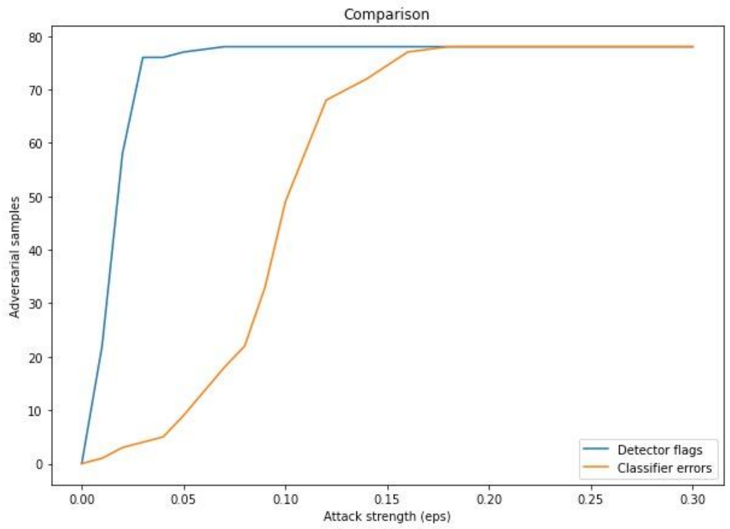
In this case the values obtained during the construction of the model are confirmed, in fact the accuracy is the same. By comparing the results with those obtained with the detector for filtering the samples, we can confirm an acceptable result.

4 Certified defenses

In order to evaluate the certified defenses on the final network, attacks made on the network itself were evaluated, with a wider range of parameters, in such a way as to evaluate a higher case history.

4.1 FGSM

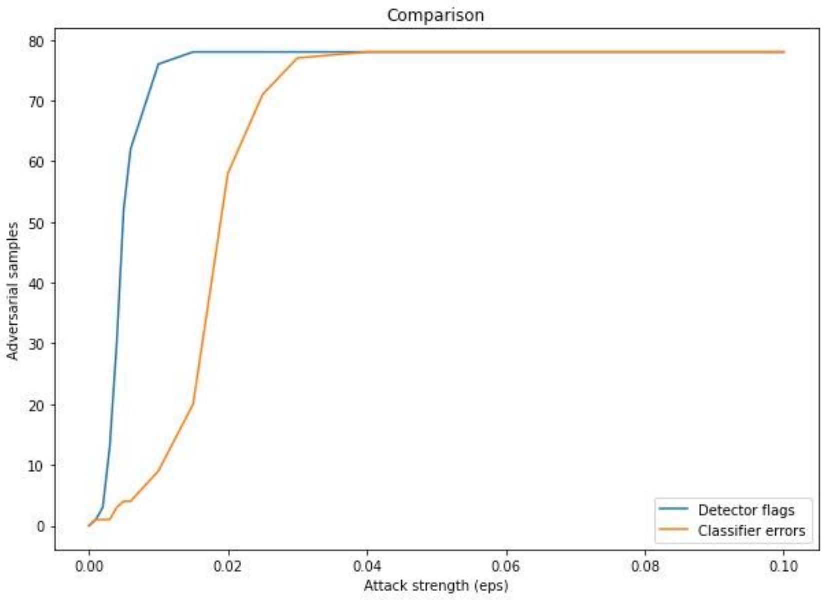
```
eps_range = [0,0.01,0.02,0.03,0.04,0.05,0.07,0.08,0.09,0.1,0.12,0.14,0.16,0.18,0.2,0.22,0.24,0.26,0.28,0.29,0.3]
```



The network is almost immune to this attack; it is able to evaluate samples well when the perturbations created are very large; it finds some difficulty for minimal perturbations but is able to maintain satisfactory accuracy because these samples (epsilon<0.06) are evaluated correctly by the WhiteBox model.

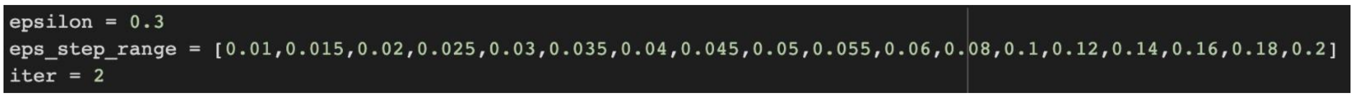
4.2 BIM

```
epsilon = 1
iter = 5
eps_step_range = [0.001,0.002,0.003,0.004,0.005,0.006,0.01,0.015,0.02,0.025,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1]
```

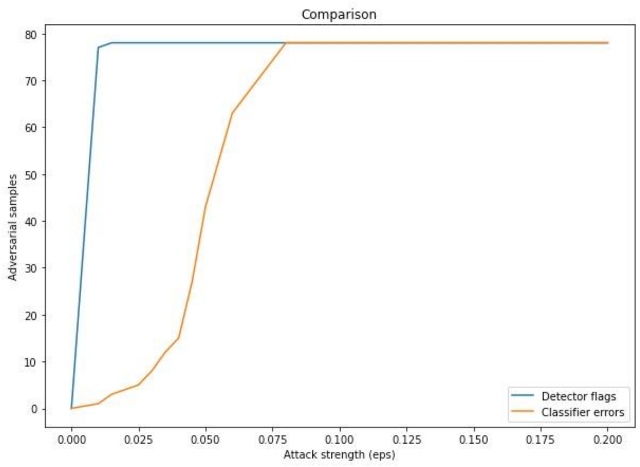


The network is almost immune to this attack; it can evaluate samples well when the perturbations created are very large; it finds some difficulty for minimal perturbations but is able to maintain satisfactory accuracy because these samples (epsilon<0.015) are evaluated correctly by the WhiteBox model.

4.3 PGD

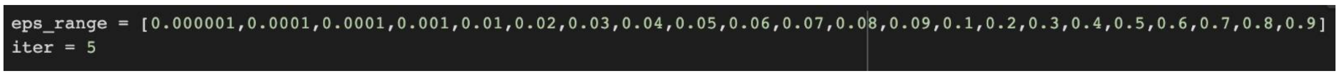


```
epsilon = 0.3
eps_step_range = [0.01,0.015,0.02,0.025,0.03,0.035,0.04,0.045,0.05,0.055,0.06,0.08,0.1,0.12,0.14,0.16,0.18,0.2]
iter = 2
```

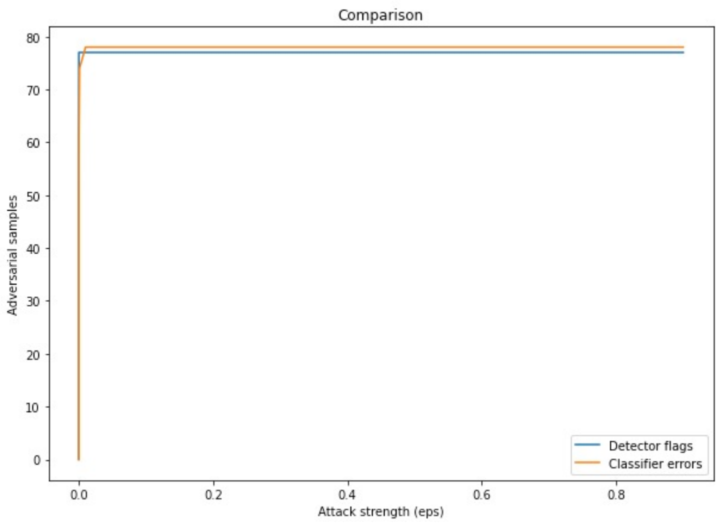


The network is immune to this attack (for attacks belonging to the range of selected parameters).

4.4 DeepFool

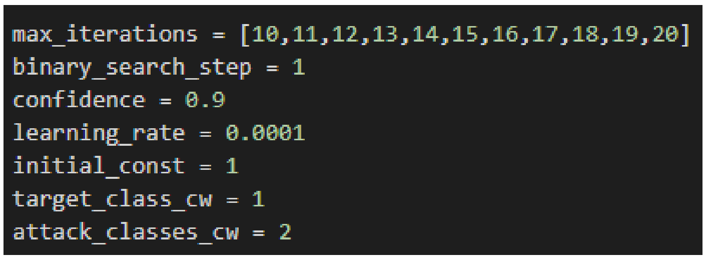


```
eps_range = [0.000001,0.0001,0.0001,0.001,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
iter = 5
```

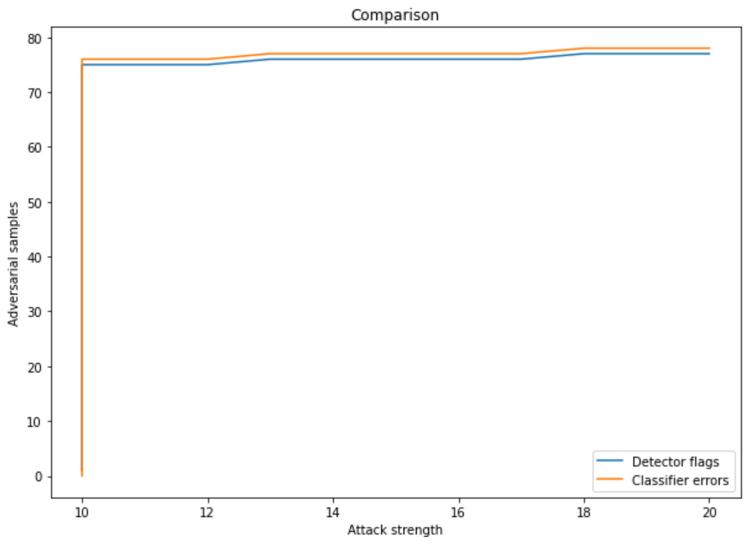


The network is almost immune to this attack. With the various tests done, it turns out that for only one sample the network fails to classify it correctly.

4.5 Carlini-Wagner



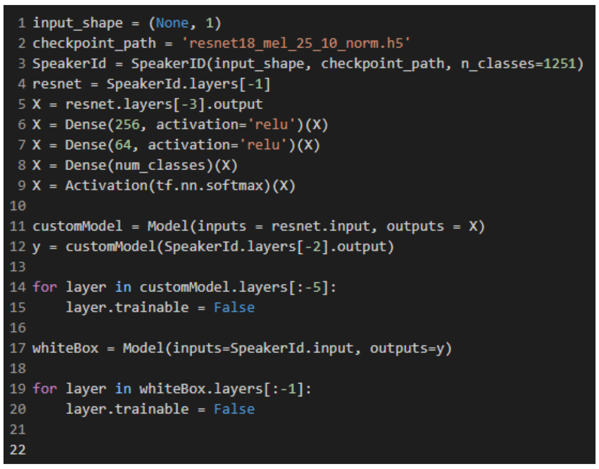
```
max_iterations = [10,11,12,13,14,15,16,17,18,19,20]
binary_search_step = 1
confidence = 0.9
learning_rate = 0.0001
initial_const = 1
target_class_cw = 1
attack_classes_cw = 2
```



With these parameters, the network is almost immune to this attack.

5 Another test

In addition to the experiments carried out previously, an additional experiment was chosen, within which the three gradient-based attacks (FGSM, BIM, PGD) were carried out directly on the audios. This was done in such a way as to have the possibility of being able to listen and visualize (in terms of a spectrogram) the actual difference between an "original" audio and an adversarial audio, since the previous experiment, acting directly on the features returned by the SpeakerId model, did not allow for this.
To do this, the previous model was repurposed to allow it to perform this task, trying to maintain it.

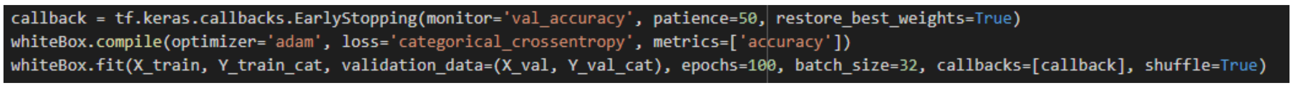


```python
input_shape = (None, 1)
checkpoint_path = 'resnet18_mel_25_10_norm.h5'
SpeakerId = SpeakerID(input_shape, checkpoint_path, n_classes=1251)
resnet = SpeakerId.layers[-1]
X = resnet.layers[-3].output
X = Dense(256, activation='relu')(X)
X = Dense(64, activation='relu')(X)
X = Dense(num_classes)(X)
X = Activation(tf.nn.softmax)(X)

customModel = Model(inputs = resnet.input, outputs = X)
y = customModel(SpeakerId.layers[-2].output)

for layer in customModel.layers[:-5]:
    layer.trainable = False

whiteBox = Model(inputs=SpeakerId.input, outputs=y)

for layer in whiteBox.layers[:-1]:
    layer.trainable = False

```

Trainable = False values were added so as not to change the weights of the supplied SpeakerId model already trained.

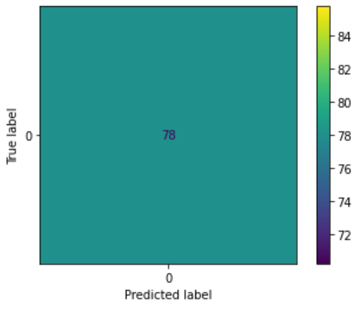
Given this network architecture, different tests were made for the training of this classifier.
For each training, the EarlyStopping callback based on the accuracy produced on the validation set was used, with a patience of 50 epochs. The loss function used is the categorical_crossentropy and all the trainings provided a batch size of 32 samples and a maximum number of epochs equal to 100: all the tests triggered the callback within the 100 epochs.
The different tests were done with the following optimizers: [adagrad, adam, rmsprop].

The chosen network was achieved with the following configuration:



```python
callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=50, restore_best_weights=True)
whiteBox.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
whiteBox.fit(X_train, Y_train_cat, validation_data=(X_val, Y_val_cat), epochs=100, batch_size=32, callbacks=[callback], shuffle=True)
```

The trained model manages to achieve a performance on the test_set of 1.0.



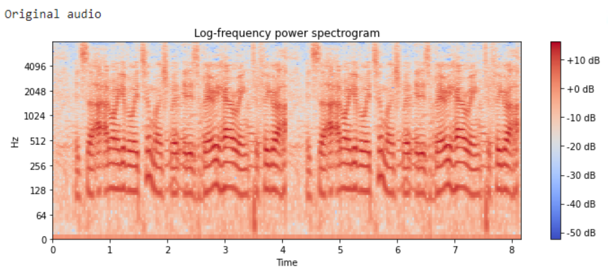
Therefore, the three attacks with their respective results will be analyzed below.

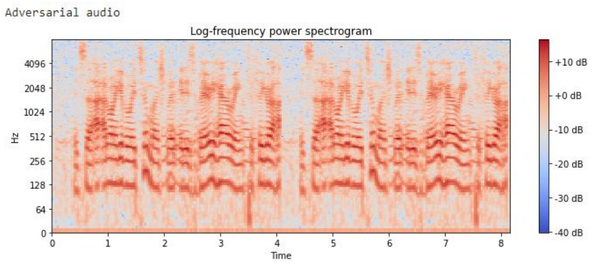
5.1 FGSM

For the FGSM attack, adversarial audio was obtained with an average perturbation of 0.0019 with an epsilon_fgsm = 0.00191. This values obviously cannot be compared with previous attacks since we are in a completely different situation, where attacks are launched directly on the output of librosa rather than on features extracted from SpeakerId.
Below are links to listen to the original and attached audio.

Original audio:
https://drive.google.com/file/d/1ekHUsacHDnG7fan99I3qzvf-yHkwN_pr/view?usp=share_link

Adversarial audio:
https://drive.google.com/file/d/1_So02e83dQ3jYWDfQRH09jg-CqOual-L/view?usp=share_link





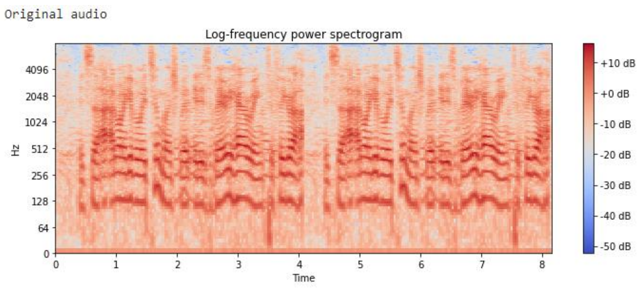
Analyzing the spectrogram of the two audios, we notice a small variation in the log-frequency spectogram. This indicates a greater presence of noise, which justifies how this sample is recognized as belonging to target class 1, despite belonging to another person.

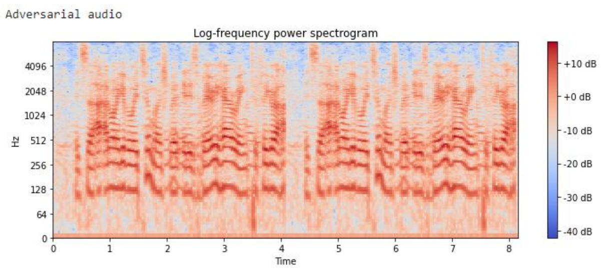
5.2 BIM

For the BIM attack, adversarial audio was obtained with an average perturbation of 0.00042 with
epsilon_bim = 0.001
epsilon_step_bim = 0.00001
max_iter_bim = 100
This values obviously cannot be compared with previous attacks since we are in a completely different situation, where attacks are launched directly on the output of librosa rather than on features extracted from SpeakerId.
Below are links to listen to the original and attached audio.

Original audio:
https://drive.google.com/file/d/1irbg9aOoedM8hNobmyd00VjfguQ3LZAq/view?usp=share_link

Adversarial audio:
https://drive.google.com/file/d/1lVy_SGiqtySrvAi_XEEcKlAupod9Ln5R/view?usp=share_link





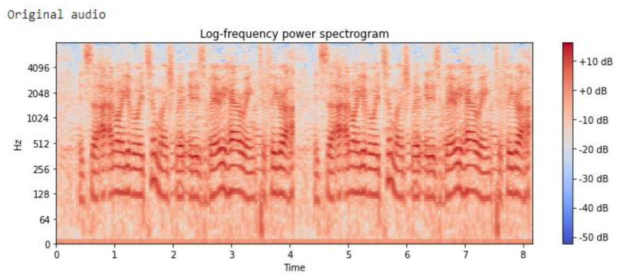
Analyzing the spectrogram of the two audios, we notice a small variation in the log-frequency spectogram. This indicates a greater presence of noise, which justifies how this sample is recognized as belonging to target class 1, despite belonging to another person.

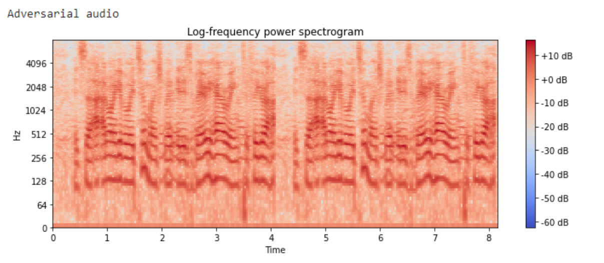
5.3 PGD

For the PGD attack, adversarial audio was obtained with an average perturbation of 0.00041. This value obviously cannot be compared with previous attacks since we are in a completely different situation, where attacks are launched directly on the output of librosa rather than on features extracted from SpeakerId.
Below are links to listen to the original and attached audio.

Original audio:
https://drive.google.com/file/d/1e1eWqxKWcaSzOOHlnXkAsXatTABfFuUi/view?usp=share_link

Adversarial audio:
https://drive.google.com/file/d/1AfK0bGQjk5y1_ptHo_WcgofK9vS3Z1Pp/view?usp=share_link





Analyzing the spectrogram of the two audios, we notice a small variation in the log-frequency spectogram. This indicates a greater presence of noise, which justifies how this sample is recognized as belonging to target class 1, despite belonging to another person.