Report Progetto

Valutazione della Sicurezza di un Sistema di Speaker Recognition

Corso
Artificial Intelligence for Cybersecurity 2022/2023

Professori
Prof. Vincenzo Carletti vcarletti@unisa.it
Prof. Antonio Greco agreco@unisa.it

Gruppo 04

Coppola Carmine	c.coppola79@studenti.unisa.it	0622701699
Cuzzocrea Allegra	a.cuzzocrea2@studenti.unisa.it	0622701707
D'Andrea Anna	a.dandrea26@studenti.unisa.it	0622701682
Di Mauro Enrico Maria	e.dimauro5@studenti.unisa.it	0622701706

Sommario

Introduzione

Si ipotizza di avere un sistema di controllo accessi basato su speaker recognition, in grado di permettere l'accesso solo agli utenti autorizzati.

- **Problema**: il sistema può essere soggetto ad eventuali attacchi che potrebbero comportare dei malfunzionamenti nel riconoscimento audio. Si ipotizza un attaccante in grado di poter eseguire solo **attacchi di evasione** (attacchi in grado di manipolare solo i dati del test set) con lo scopo di permettere l'accesso ad un utente non autorizzato o negarlo ad uno autorizzato; dunque, si tratta di un problema di classificazione binaria (autorizzato vs non autorizzato).

- **Obiettivo**: realizzare e valutare le performance del sistema di controllo accessi basato su speaker recognition, il tutto seguendo un **approccio proattivo**, ovvero immergendosi sia lato attaccante che difensore.

Preparazione: Colab & Librerie

Per la realizzazione e l'analisi del sistema è stato necessario **Google Colab**, uno strumento gratuito presente nella suite Google che consente di scrivere codice Python direttamente dal proprio browser; inoltre, è stato necessario installare ed importare le seguenti librerie (requirements):

- **TensorFlow**: una libreria Python open source per l'apprendimento automatico

- **Adversarial Robustness Toolbox (ART)**: una libreria Python open source per la sicurezza in ambito machine learning. ART fornisce i tool per permettere a sviluppatori e ricercatori di difendere e valutare applicazioni e modelli di machine learning rispetto a minacce avversarie (Evasion, Poisoning etc.). ART supporta tutti i framework di machine learning più noti (TensorFlow, Keras, PyTorch, MXNet, scikit-learn etc.), tutti i tipi di dati (immagini, audio, video etc.) e tutti i task di machine learning (classificazione, rilevamento di oggetti, riconoscimento vocale etc.)

Analisi preliminare

Database

Il Database originario (dataset.csv) comprende un totale di circa 150.000 campioni audio appartenenti a 1251 utenti differenti (id10001, …, id11251), mentre il Dataset fornito, per ovvie questioni di praticità e memoria, ne comprende 4874 appartenenti a 40 utenti differenti (id10270, …, id10309). Come già accennato nell'introduzione, si tratta di un problema di classificazione binaria. Nelle fasi successive, quindi, verrà mostrata la procedura di suddivisione in classe positiva (utente autorizzato) e classe negativa (utenti non autorizzati).

	Unnamed: 0	split	path	video	VoxCeleb1 ID	VGGFace1 ID	Gender	Nationality	Set
0	0	3	id10003/na8-QEFmj44/00003.wav	na8-QEFmj44	id10003	Aamir_Khan	m	India	dev
1	1	1	id10003/tCq2LcKO6xY/00002.wav	tCq2LcKO6xY	id10003	Aamir_Khan	m	India	dev
2	2	1	id10003/K5zRxtXc27s/00001.wav	K5zRxtXc27s	id10003	Aamir_Khan	m	India	dev
3	3	1	id10003/bDxy7bnj_bc/00004.wav	bDxy7bnj_bc	id10003	Aamir_Khan	m	India	dev
4	4	1	id10003/E_6MjfYr0sQ/00011.wav	E_6MjfYr0sQ	id10003	Aamir_Khan	m	India	dev
...	...	...	...	...	...	...	...	...	...
152794	153511	1	id11251/s4R4hvqrhFw/00002.wav	s4R4hvqrhFw	id11251	Zulay_Henao	f	USA	dev
152795	153512	1	id11251/gFfcgOVmiO0/00006.wav	gFfcgOVmiO0	id11251	Zulay_Henao	f	USA	dev
152796	153513	3	id11251/7GtZpUtReJ8/00001.wav	7GtZpUtReJ8	id11251	Zulay_Henao	f	USA	dev
152797	153514	2	id11251/5-6lI5JQtb8/00001.wav	5-6lI5JQtb8	id11251	Zulay_Henao	f	USA	dev
152798	153515	3	id11251/7GtZpUtReJ8/00006.wav	7GtZpUtReJ8	id11251	Zulay_Henao	f	USA	dev

Figura 1. Database

Modello SpeakerID

Il modello SpeakerID è stato utilizzato per l'estrazione delle features e per questo motivo è stato necessario modificarlo in modo da ottenere 2 output: predizione riguardante l'id utente e il feature vector. Dunque, il modello prevede un campione audio ".wav" in input da cui ricava lo Spettrogramma di Mel, ovvero una rappresentazione visiva dell'audio stesso; in seguito, lo Spettrogramma viene dato in pasto alla rete ResNet, che lavora con le immagini ed estrae le feature.

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, 1)]         0

 lambda (Lambda)             (None, None, 1)           0

 log_melgram_layer (LogMelgr  (None, None, 128)        0
 amLayer)

 z_score_normalization (ZSco  (None, None, 128)        0
 reNormalization)

 model (Functional)          [(None, 1251),            11822374
                              (None, 512)]

=================================================================
Total params: 11,822,374
Trainable params: 11,814,436
Non-trainable params: 7,938
_________________________________________________________________
```

Figura 2. Architettura modello SpeakerID

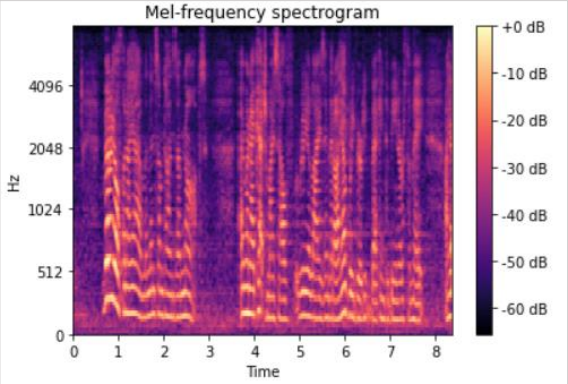

Figura 3. Spettrogramma di Mel

Per il caricamento e l'interazione con i file audio è stato necessario sfruttare **Librosa**, una libreria Python per l'analisi audio. Di fianco viene riportato un esempio di Spettrogramma di Mel ottenuto grazie al metodo apposito di Librosa.

Popolamento delle Classi

I classificatori binari utilizzati operano sulle seguenti classi:

- **classe positiva**: campioni audio appartenenti all'utente autorizzato
- **classe negativa**: campioni audio appartenenti agli utenti non autorizzati

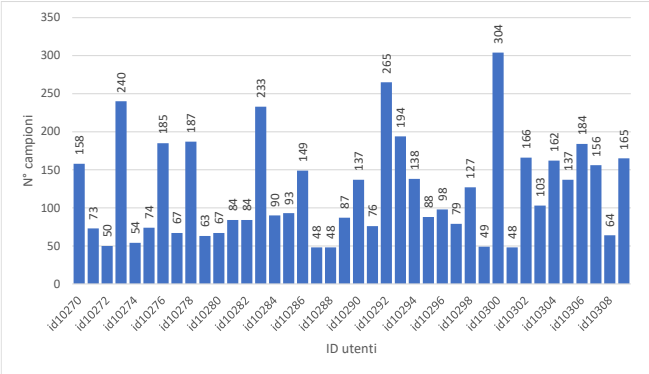

Figura 4. ID - n° campioni

Prima di popolare correttamente le classi sopra citate, è stata effettuata un'analisi del dataset a disposizione. In particolare, è stato verificato che il numero dei campioni a disposizione è pari a 4874 divisi in maniera non equa tra 40 utenti. Ogni utente è rappresentato da un id univoco.

Dal momento che l'utilizzo del dataset a disposizione comportava uno sbilanciamento eccessivo a favore della classe negativa, è stato deciso di perseguire l'obiettivo di **utilizzare il maggior numero di campioni a disposizione** e, allo stesso tempo, di **ottenere un dataset bilanciabile attraverso una data augmentation accettabile**[1].

Per tale motivo, è stato scelto l'utente con il maggior numero di campioni audio a disposizione per popolare la classe positiva; per la classe negativa, invece, è stato individuato l'utente con il minor numero di campioni audio a disposizione in modo da prelevare randomicamente tale quantità da tutti gli utenti non autorizzati.

Il risultato di tale selezione ha prodotto un dataset costituito da un sottoinsieme del dataset originale, formato da 304 campioni audio per la classe positiva (utente con **"id10300"**) e 1872 campioni audio per la classe negativa. In questo modo, quando sarà effettuata la data augmentation si otterrà un numero di campioni augmentati positivi pari a circa 6 volte quello dei campioni originali positivi. Questo numero è stato considerato adeguato in quanto di poco superiore al limite utilizzato per il concetto di accettabilità.

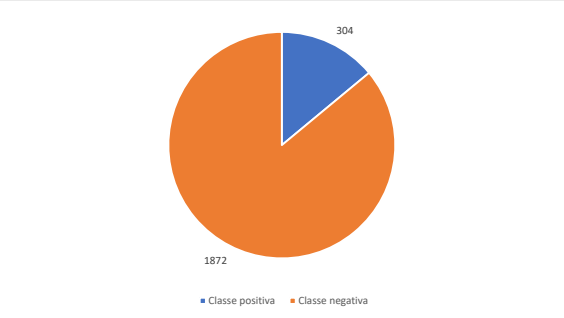

Figura 5. Classe positiva - Classe negativa

Dataset Splitting

Una volta determinate le classi è stata effettuato lo splitting in Training Set, Validation Set e Test Set.

In particolare, ognuna delle due classi è stata divisa in modo randomico in tre sottoinsiemi, quello per il training, quello per il validation e quello per il test, contenenti rispettivamente il 60%, il 20% ed il 20% dei campioni.

Tale divisione ha prodotto i seguenti insiemi:

- **Training Set**: 182 campioni positivi e 1123 campioni negativi
- **Validation Set**: 61 campioni positivi e 374 campioni negativi
- **Test Set**: 61 campioni positivi e 375 campioni negativi

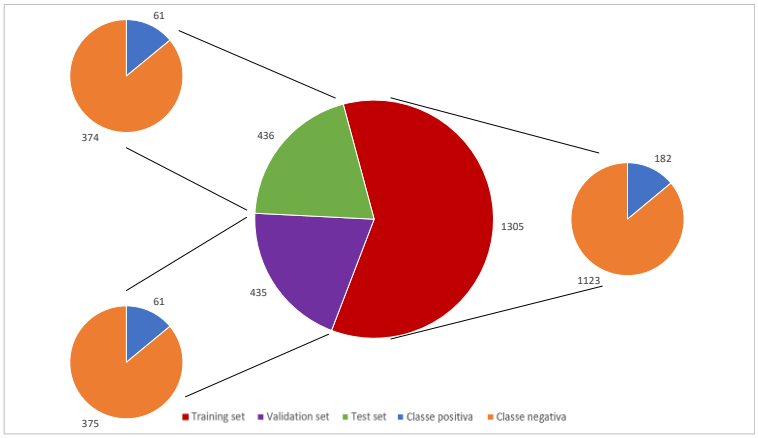

Figura 6. Training, Validation e Test set con suddivisione in classe positiva e negativa

[1] Il numero dei campioni prodotti dalla data augmentation deve essere al massimo circa 5 volte il numero dei campioni originali della classe augmentata

Data Augmentation

Dopo aver ottenuto i sottoinsiemi di ogni classe è stato possibile procedere con la data augmentation citata durante la fase di popolamento delle classi. In questo modo, è stato ottenuto un dataset completamente bilanciato.

In particolare, l'obiettivo è stato far coincidere il numero di campioni positivi presenti in ogni set con il corrispondente numero di campioni negativi. Per fare ciò, in ogni set, sono stati creati dei campioni augmentati attraverso il seguente procedimento:

1. Selezionare un campione positivo
2. Applicare una trasformazione casuale tra quelle scelte generando un nuovo campione
3. Salvare il campione ottenuto
4. Ripetere dal punto 1. fino a saturare la differenza tra campioni positivi e negativi

Nota: la selezione del campione positivo avviene in maniera ordinata e ciclica tra quelli originali

Le trasformazioni scelte per effettuare la data augmentation sono:

- **Noise Injection**: introduce all'interno dell'audio del rumore randomico
- **Shifting Time**: trasla l'audio casualmente verso destra o sinistra dei secondi indicati
- **Changing Pitch**: modifica casualmente la tonalità dell'audio
- **Changing Speed**: velocizza o rallenta l'audio del fattore indicato
- **Random Gain**: amplifica casualmente il guadagno dell'audio in un intervallo indicato
- **Polarity Inversion**: inverte l'ampiezza del segnale audio trasformando i valori positivi in negativi e viceversa

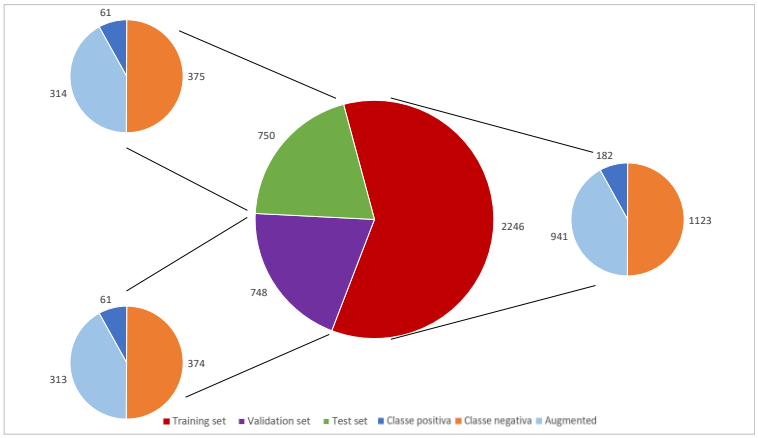

Figura 7. Training, Validation e Test set con suddivisione in classe positiva e negativa dopo la data augmentation

Sarebbe stato possibile invertire le fasi di dataset splitting e data augmentation, ma ciò avrebbe comportato l'introduzione di correlazioni tra i campioni di Training, Validation e Test Set. In questo modo, quindi, si sarebbe commesso un errore in quanto i risultati del classificatore scelto come modello White-Box sarebbero stati falsati.

Estrazione delle Feature

In seguito alla fase di data augmentation è stata effettuata l'estrazione delle feature. In particolare, sono state estratte le feature di tutti i campioni mantenendo la divisione di Training, Validation e Test Set di classe positiva e negativa. Tale suddivisione è stata mantenuta in modo da effettuare verifiche sulla correttezza delle estrazioni ottenute.

Oltre ai vettori delle feature sono stati prodotti anche quelli delle label mantenendo una corrispondenza tra feature di ogni audio e classe di appartenenza.

Infine, è stata realizzata la concatenazione di tutte le feature e label ottenendo due vettori separati per ognuno dei tre set. Tali vettori saranno utilizzati durante la fase di training del modello. Dopodiché, ognuno dei feature vector risultanti è stato salvato (ad es. come `final_train_features.npy`), in modo da farne poi una load quando serve, per evitare di rieseguire l'estrazione delle feature che richiede tempo.

Training & Testing

Le feature estratte precedentemente vengono usate per l'addestramento di 2 classificatori binari: MLP e KNN. Il classificatore MLP verrà sfruttato sia come White-Box che come Black-Box, quello KNN solo come Black-Box

Aggiornamento Label: One-Hot Encoding

Prima dell'addestramento effettivo, però, è stato necessario aggiornare le label, dato che si tratta di un problema binario. È stato effettuato un One-Hot Encoding nel seguente modo:

- Classe Positiva: [0, 1]
- Classe Negativa: [1, 0]

MLP (White-Box)

Il primo classificatore sfruttato è un MLP, che nelle fasi successive fungerà da classificatore White-Box. In input sono previsti vettori di feature contenenti 512 elementi, mentre in output la predizione sulla classe di appartenenza.

L'apprendimento è stato realizzato con ottimizzatore Adam e loss Binary Crossentropy.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 350)               179550

 dense_1 (Dense)             (None, 100)               35100

 dense_2 (Dense)             (None, 50)                5050

 dense_3 (Dense)             (None, 2)                 102

=================================================================
Total params: 219,802
Trainable params: 219,802
Non-trainable params: 0
_________________________________________________________________
```

Figura 8. Architettura del modello white-box

I risultati ottenuti sul Test Set costituito da campioni non corrotti sono i seguenti:

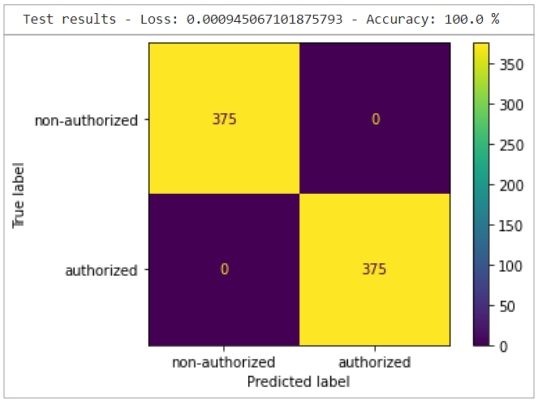

Figura 9. Matrice di confusione MLP_wb

KNN (Black-Box)

Il secondo classificatore sfruttato è un KNN con k = 1 di **Scikit-learn** (una libreria open source di apprendimento automatico per Python), che nelle fasi successive fungerà da classificatore Black-Box.

I risultati ottenuti sul Test Set costituito da campioni non corrotti sono i seguenti:

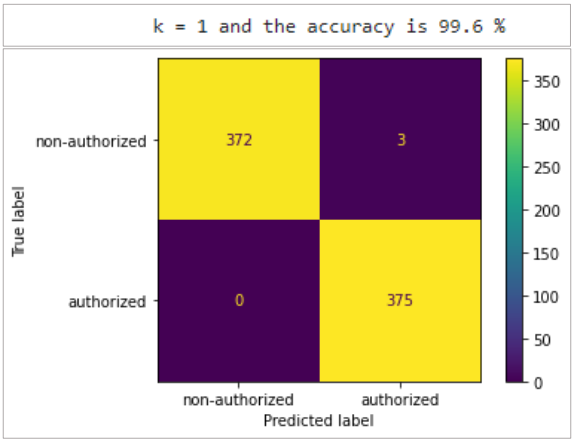

Figura 10. Matrice di confusione KNN_bb

MLP (Black-Box)

Per una maggiore completezza della successiva analisi di trasferibilità, è stato sfruttato anche un terzo classificatore (Black-Box), ovvero un MLP di **Scikit-learn**.

I risultati ottenuti sul Test Set costituito da campioni non corrotti sono i seguenti:

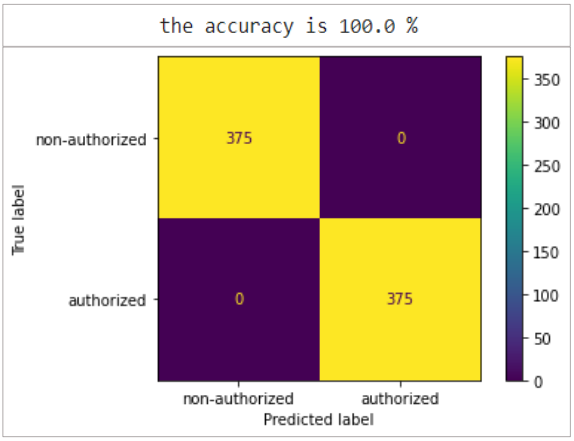

Figura 11. Matrice di confusione MLP_bb

Simulazione Offensiva & Analisi

Descrizione degli attacchi

Come già accennato nell'introduzione, è stato deciso di valutare solo **attacchi di evasione**, con cui l'attaccante può manipolare solo i dati del Test Set, al fine di ingannare il sistema. È stato seguito il paradigma di progettazione **Security by Design**, ovvero un **approccio proattivo** che prevede di agire sia come attaccante, valutando gli effetti degli attacchi sul sistema e la loro trasferibilità, che come difensore, progettando e sviluppando un meccanismo di difesa contro tali attacchi.

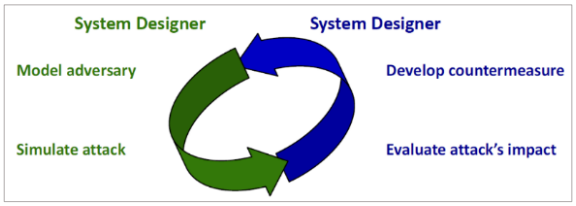

Figura 12. Approccio proattivo

Tutti gli attacchi presi in considerazione sono **Targeted** in quanto vengono presi di mira i campioni del Test Set; poi, sono state considerate le due seguenti categorie:

1. **Attacchi Error-Generic**: un attaccante mira ad ottenere un accesso illecito o a negare l'accesso ad un utente autorizzato, facendolo scambiare per un non autorizzato (DoS); dunque, non c'è alcuna classe target e l'obiettivo è far predire sempre una classe diversa da quella reale
2. **Attacchi Error-Specific**: un attaccante mira ad ottenere un accesso illecito facendosi scambiare per un utente autorizzato; dunque, la classe target è quella Positiva [0, 1] e l'obiettivo è far predire la classe negativa come positiva. Si noti che si potrebbe ottenere un DoS anche con un attacco Error-Specific impostando la classe Negativa come classe target

Ogni attacco è stato eseguito sul classificatore **MLP White-Box** e successivamente è stato soggetto ad un'analisi di trasferibilità attraverso l'utilizzo dei classificatori **KNN Black-Box** e **MLP Black-Box**; dunque, i campioni generati durante l'attacco al classificatore White-Box sono stati utilizzati per attaccare anche i classificatori Black-Box.

Con classificatore White-Box s'intende un sistema per cui l'attaccante possiede il massimo livello di conoscenza, ovvero egli conosce l'intera quadrupla $\boldsymbol{\theta_{PK} = (D, X, f, w)}$ del sistema bersagliato, dove D è il Training Set, X è il set di feature, f è l'algoritmo di apprendimento e w rappresenta i parametri addestrati; infatti, in questo caso, si parla di **Perfect Knowledge**. Gli attacchi su questo tipo di sistema permettono di ottenere le prestazioni effettive del sistema, cioè come esse degradano rispetto alla forza dell'attacco.

Con classificatore Black-Box, invece, s'intende un sistema per cui l'attaccante non ha conoscenza, o meglio conosce solo l'obiettivo del sistema ma non la quadrupla sopra citata; infatti, in questo caso, si parla di **Zero Knowledge**. Per attaccare un sistema di questo tipo viene valutato il comportamento di altri sistemi simili, così da scegliere il miglior attacco in termini di trasferibilità.

La diagnosi di sicurezza del sistema (analisi di robustezza e trasferibilità pre e post attacchi) è stata eseguita generando la **Security Evaluation Curve**, che mostra l'andamento dell'accuracy del sistema al variare della forza dell'attacco. Per accuracy si intende il numero di campioni correttamente classificati rispetto al numero totale di campioni del Test Set. Inoltre, per ogni attacco, è stato generato anche un istogramma che mostra l'andamento della perturbazione applicata al variare della forza dell'attacco.

Attacchi

In ART, Untargeted e Generic, Targeted e Specific sono usati come sinonimi. Infatti, tutti gli attacchi Error-Generic hanno il parametro targeted settato a False mentre quelli Error-Specific a True.

Gli attacchi sferrati sono FGSM, BIM, PGD e CW.

- FGSM, BIM e PGD sono detti attacchi del primo ordine o attacchi Gradient Based dato che si basano sulla conoscenza del Gradiente della funzione di costo. A differenza di FGSM e BIM, PGD non è un attacco totalmente controllabile a causa dell'introduzione della casualità
- CW mira ad ottenere il massimo risultato con la minima perturbazione, ma si tratta di un attacco più lento rispetto a quelli Gradient Based

Sono state considerate sia la versione Error-Generic che quella Error-Specific di tutti gli attacchi sopra citati.

Rispetto agli attacchi Error-Generic, che riescono ad azzerare l'accuracy, gli attacchi Error-Specific riescono al massimo a dimezzarla. Questo accade perché l'attacco mira a convertire solo i campioni della Classe Negativa [1, 0] nella Classe Positiva [0, 1], ovvero mira solo alla metà dei campioni.

FGSM (Fast Gradient Sign Method)

Per ogni campione di test, l'algoritmo FGSM utilizza il gradiente (rispetto all'input) con segno per generare un campione corrotto avente una perturbazione proporzionale al valore di ε. Il parametro settato in questo attacco è proprio **epsilon** e, infatti, all'aumentare del suo valore aumenta la perturbazione e, perciò, le modifiche saranno più evidenti e l'attacco sarà più efficace.

I valori testati sono:

```
[0, 0.001, 0.003, 0.008, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45,
 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.5, 2, 2.5]
```

Error Generic

Di seguito viene mostrata la Security Evaluation Curve del classificatore MLP White-Box e si può notare che l'attacco riesce ad abbattere completamente l'accuracy (0%) a partire da $\varepsilon \simeq 0.6$.

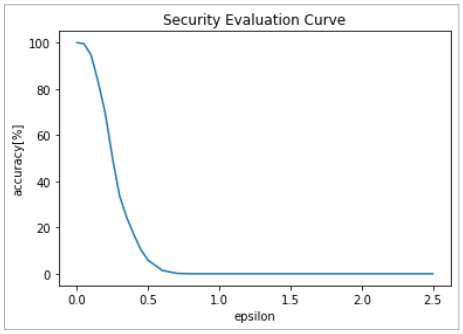

Figura 13. FGSM - S.E.C. MLP_wb

Invece, nel caso di KNN Black-Box e MLP Black-Box, si ha la seguente situazione:

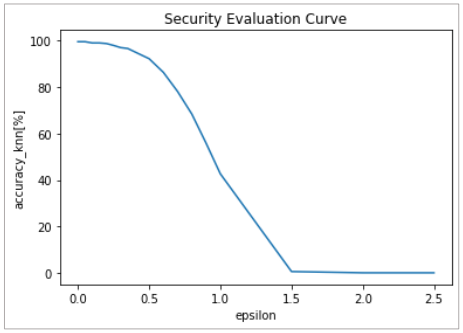

Figura 14. FGSM - S.E.C. KNN_bb

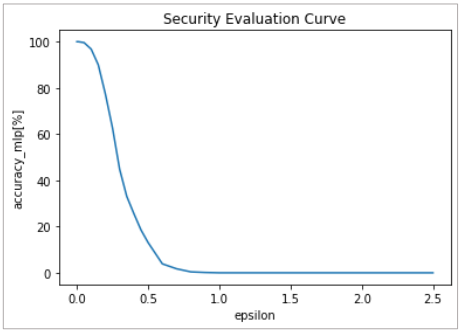

Figura 15. FGSM - S.E.C. MLP_bb

La Security Evaluation Curve del classificatore MLP Black-Box è molto simile a quella del classificatore White-Box e ciò indica la piena trasferibilità dell'attacco. I due modelli, infatti, sono molto simili tra loro. Invece, nel caso del classificatore KNN Black-Box, con valori piccoli di ε (inferiori a 1) non si riesce ad abbattere considerevolmente l'accuracy e ciò indica una minore trasferibilità su questo modello. In quest'ultimo caso, per indurre il classificatore a sbagliare completamente le predizioni, è necessaria una perturbazione maggiore e, quindi, un ε pari ad almeno 1.5.

Inoltre, l'istogramma riportato di seguito permette di osservare che all'aumentare di ε aumenta la perturbazione.

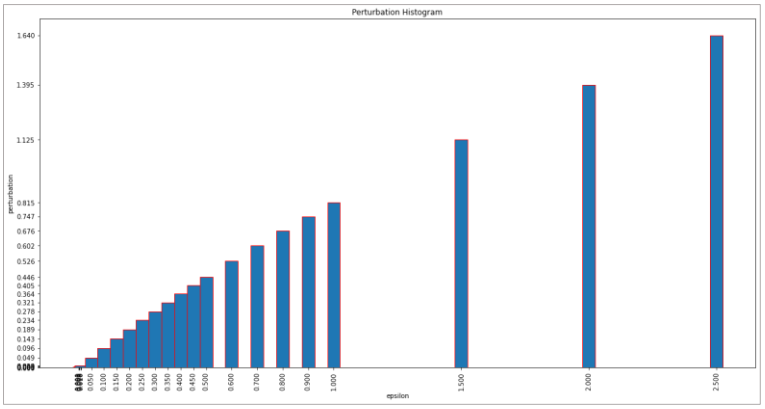

Figura 16. FGSM - Perturbation histogram MLP_wb

Error-Specific

Come già detto in precedenza, il miglior risultato raggiungibile con un attacco di tipo Error-Specific prevede il dimezzamento dell'accuracy del classificatore.

Le curve ottenute dall'attacco FGSM Error-Specific sono molto simili a quelle della versione Error-Generic. Infatti, come è possibile osservare dai grafici riportati di seguito, l'accuracy dei classificatori MLP White-Box e Black-Box si dimezza a partire da un valore di $\varepsilon \simeq 0.6$. Invece, per ottenere la minore accuracy possibile sul classificatore KNN Black-Box, è necessario un valore di $\varepsilon = 1.5$.

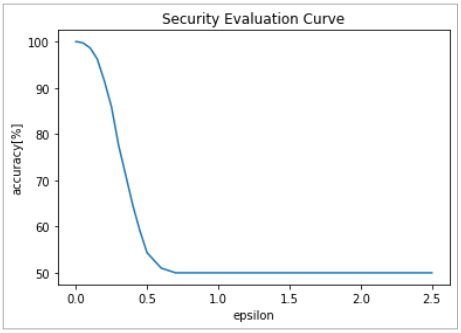

Figura 17. FGSM - S.E.C. MLP_wb

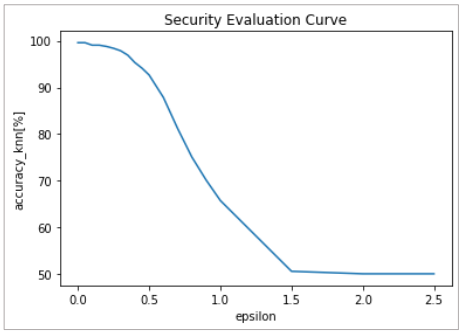

Figura 18. FGSM - S.E.C. KNN_bb

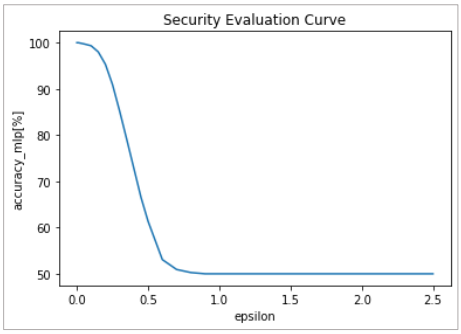

Figura 19. FGSM - S.E.C. MLP_bb

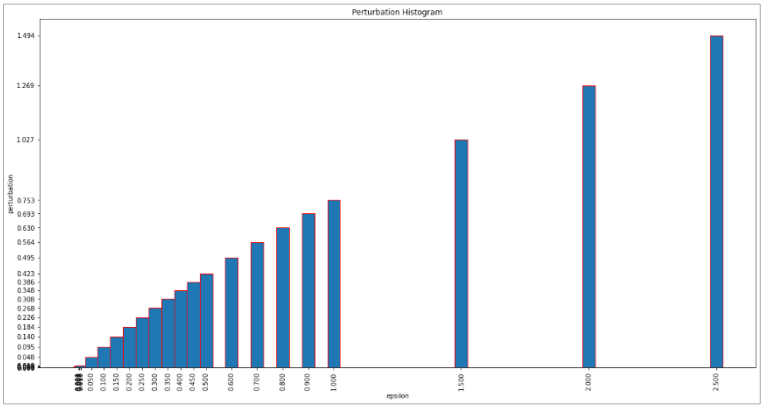

Figura 20. FGSM - Perturbation histogram MLP_wb

BIM (Basic Iterative Model)

L'algoritmo BIM è una variante iterativa di FGSM, che aggiunge ripetutamente rumore al campione al fine di causare una classificazione errata. A differenza di FGSM, dato che è possibile effettuare più iterazioni utilizzando valori piccoli di ε, in BIM il rumore non è uniforme.

I parametri settati in questo attacco sono:

- **max_iter**: numero massimo di iterazioni. Questo parametro è indipendente dalle iterazioni effettive, che dipendono da epsilon ed eps_step. Ad esempio, con epsilon = 0.1 ed eps_step = 0.05 si avranno 2 iterazioni effettive, anche se max_iter = 10
- **epsilon_step**: incremento di epsilon ad ogni iterazione
- **epsilon**: massima perturbazione raggiungibile

I valori testati sono:

- `max_iters = [5, 10, 20, 30]`
- `epsilon_steps = [0.05, 0.1, 0.3, 0.5, 0.7, 1]`
- `epsilons = [0, 0.1, 0.3, 0.5, 0.7, 1, 1.5, 2]`

Error-Generic

Testando tutte le combinazioni dei valori sopra riportati è emerso che la prima configurazione che causa un totale abbattimento dell'accuracy del classificatore MLP White-Box è costituita dai seguenti parametri: max_iter = 5, epsilon_step = 0.3.

Come si evince dalla Security Evaluation Curve, infatti, l'accuracy si azzera a partire da un valore di $\varepsilon \simeq 0.7$.

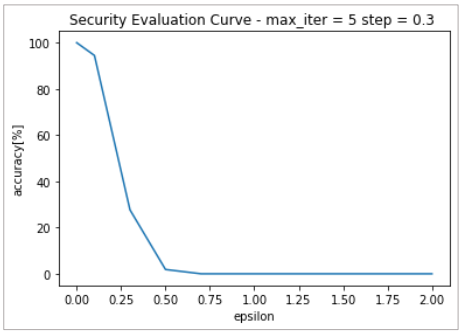

Figura 21. BIM - S.E.C. MLP_wb

L'analisi della trasferibilità ha rivelato che l'accuracy del classificatore KNN Black-Box è portata a 0%, a partire da un valore di $\varepsilon = 1.5$. Invece, la trasferibilità sul classificatore MLP Black-Box è maggiore e da $\varepsilon \simeq 0.75$ si riesce a portare a 0% l'accuracy.

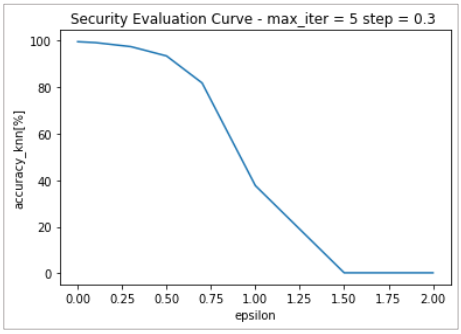

Figura 22. BIM - S.E.C. KNN_bb

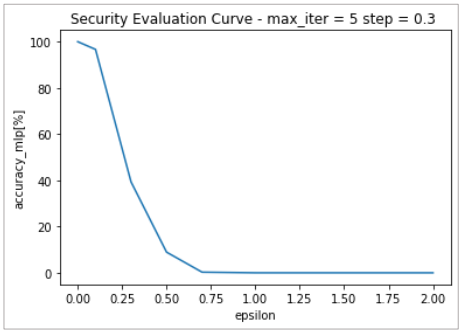

Figura 23. BIM - S.E.C. MLP_bb

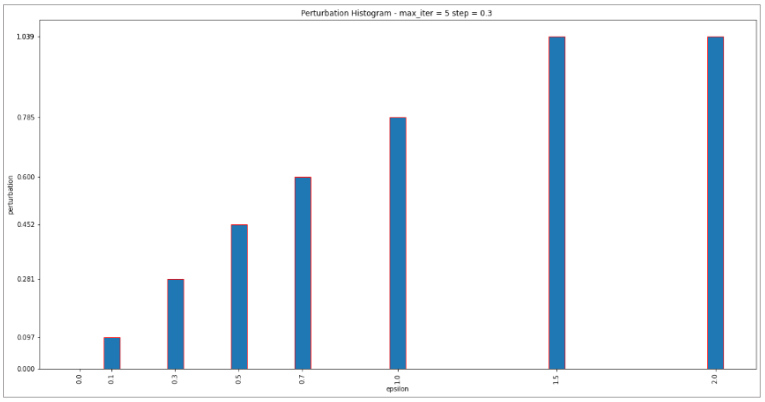

Figura 24. BIM - Perturbation histogram MLP_wb

Mantenendo costante max_iter a 5 e aumentando epsilon_step a valori maggiori di 0.3, si hanno grafici simili a quelli sopra riportati.

Se considerassimo l'attacco solo riferito al classificatore MLP White-Box, con max_iter = 5 basta epsilon_step = 0.1 per portare a 0% l'accuracy a partire da ε = 0.5.

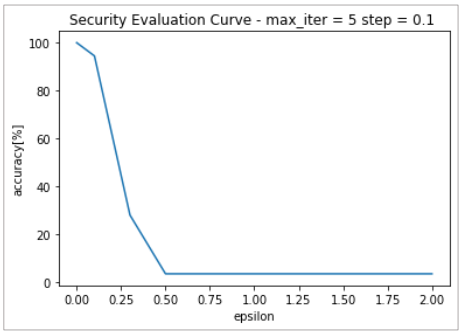

Figura 25. BIM - S.E.C. MLP_wb

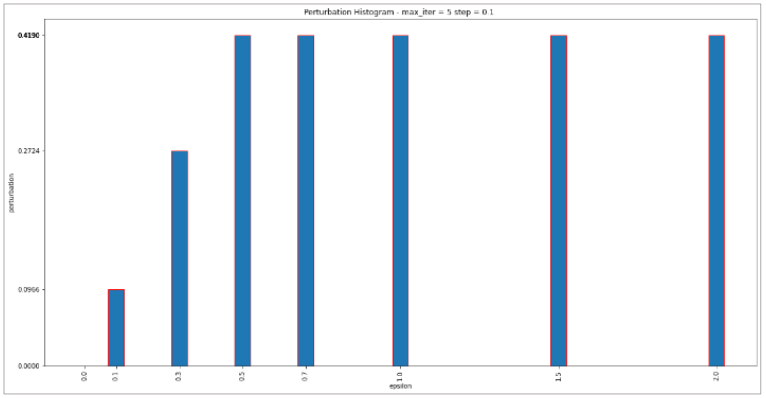

Figura 26. BIM - Perturbation histogram MLP_wb

Tuttavia, con max_iter = 5 ed epsilon_step ≤ 0.1, l'attacco risulta essere poco trasferibile.

Da ulteriori analisi sulle prestazioni del classificatore MLP White-Box è stato verificato che, mantenendo costante il valore di max_iter a 10, con epsilon_step = 0.05 si hanno prestazioni simili al grafico di sopra; però, anche con questa configurazione, l'attacco risulta poco trasferibile.

Infine, è stato verificato che, aumentando i valori di max_iter ed epsilon_step, non si ottengono miglioramenti significativi; infatti, con max_iter = 20 ed epsilon_step = 0.7, si hanno le stesse prestazioni della prima configurazione mostrata.

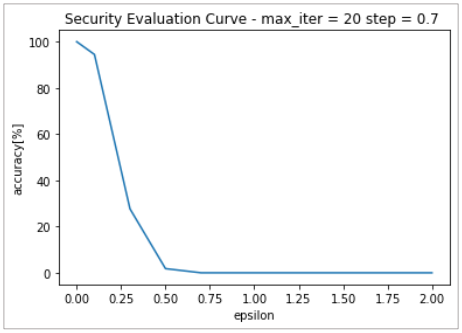

Figura 27. BIM - S.E.C. MLP_wb

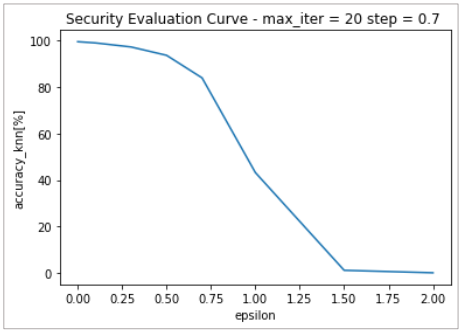

Figura 28. BIM - S.E.C. KNN_bb

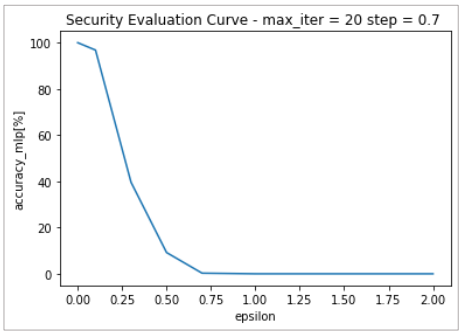

Figura 29. BIM - S.E.C. MLP_bb

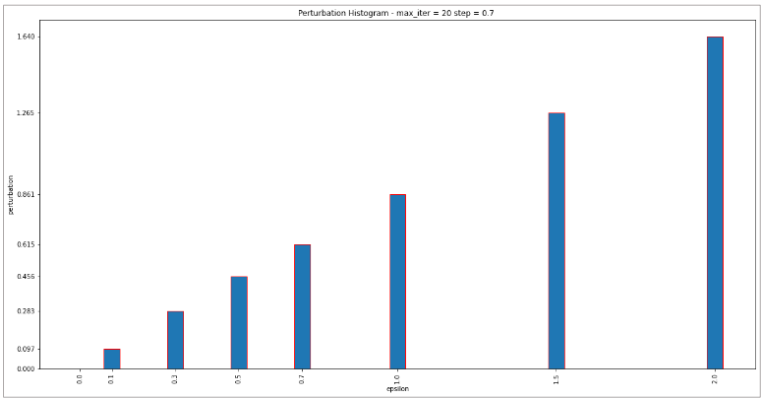

Figura 30. BIM - Perturbation histogram MLP_wb

Error-Specific

I risultati relativi alla versione Error-Specific sono molto simili a quelli della versione Error-Generic. Infatti, come nel caso Error-Generic, la configurazione con max_iter = 5 e epsilon_step = 0.3 permette di ottenere un ottimo risultato sia attaccando il classificatore White-Box che in termini di trasferibilità.

Di seguito sono riportati i grafici ottenuti.

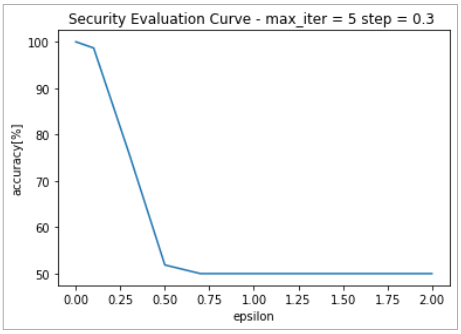

Figura 31. BIM - S.E.C. MLP_wb

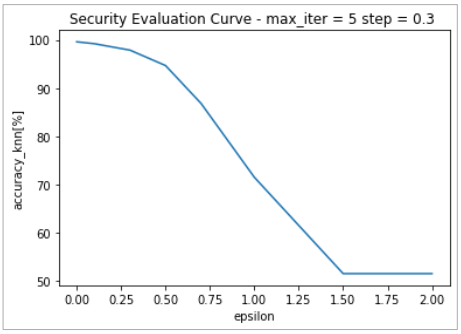

Figura 33. BIM - S.E.C. KNN_bb

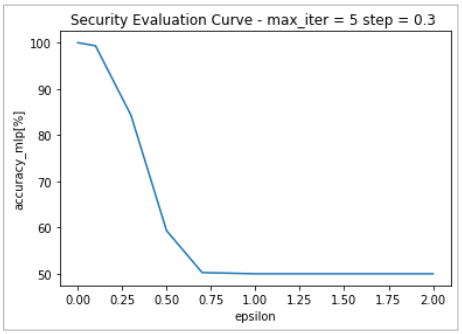

Figura 32. BIM - S.E.C. MLP_bb

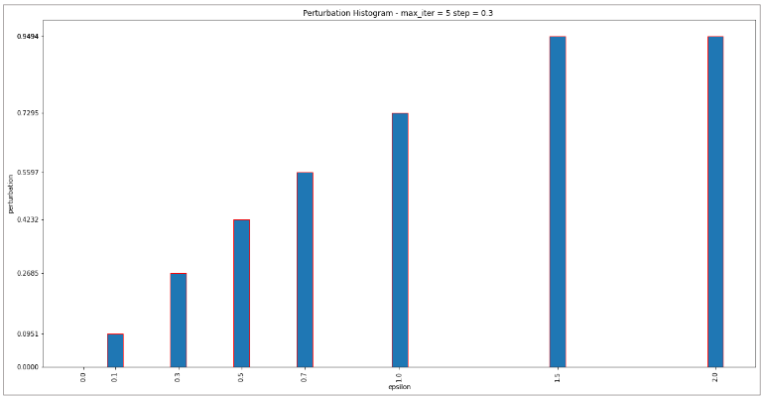

Figura 34. BIM - Perturbation histogram MLP_wb

Con ε = 0.7 si porta l'accuracy al 50% nei classificatori MLP White-Box e Black-Box; invece, nel caso del KNN Black-Box, serve un ε = 1.5.

Come nel caso Error-Generic, aumentando max_iter ed epsilon_step, non si ottengono miglioramenti rispetto alla configurazione sopra riportata.

PGD (Projected Gradient Descent)

L'algoritmo PGD è un'estensione di BIM. In particolare, BIM implica sempre movimenti in direzioni specifiche mentre PGD introduce un elemento di causalità per ogni iterazione. Quindi, aggiungendo del rumore casuale da una distribuzione uniforme con valori nell'intervallo (– ε, + ε), PGD risulta più efficace, in quanto permette di superare le limitazioni sul movimento che si hanno con BIM.

I parametri settati in questo attacco sono i medesimi di BIM, ma con l'aggiunta di **num_random_init**, che indica il numero di inizializzazioni random.

I valori testati sono:

- `epsilons = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1, 1.5, 2]`
- `epsilon_steps = [0.1, 0.3, 0.5, 1]`
- `max_iters = [5, 10, 20]`
- `num_random_init = [1, 5, 10]`

Di seguito sono proposte le configurazioni ottimali dal punto di vista dell'attaccante, ottenute attraverso il testing delle combinazioni dei valori.

Error-Generic

Utilizzando la configurazione random_init = 1, max_iter = 5, ed epsilon_step = 0.3, nella SEC riferita al classificatore MLP White-Box l'accuracy viene abbattuta completamente a partire da un valore di $\varepsilon \simeq 0.7$.

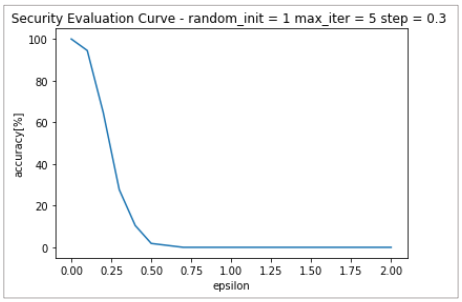

Figura 35. PGD - S.E.C. MLP_wb

Osservando le SEC relative ai classificatori Black-Box, è possibile affermare che l'attacco considerato è poco trasferibile sul classificatore KNN, in quanto un valore di $\varepsilon = 2$ non è sufficiente ad azzerare completamente l'accuracy, mentre risulta essere trasferibile sul classificatore MLP Black-Box con un valore di $\varepsilon \simeq 0.75$.

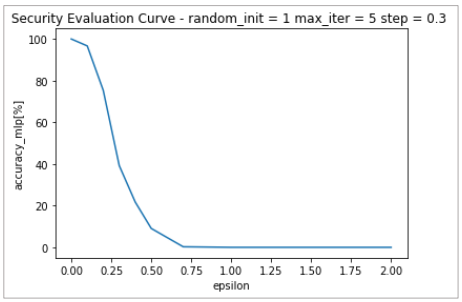

Figura 36. PGD - S.E.C. MLP_bb

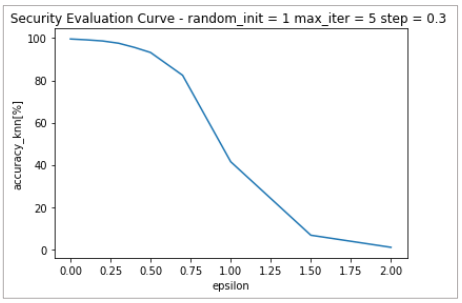

Figura 37. PGD - S.E.C. KNN_bb

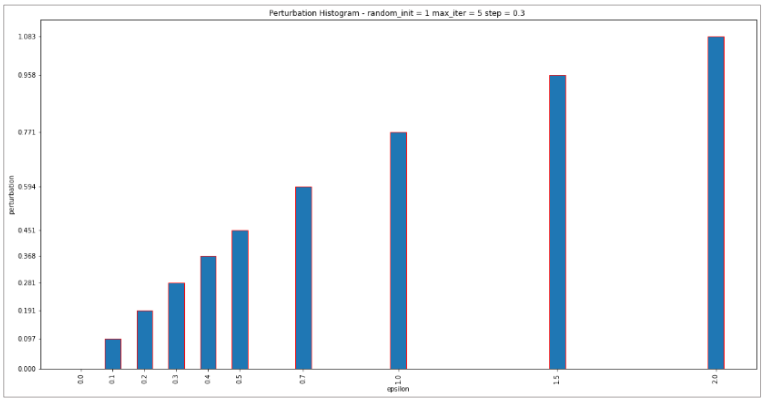

Figura 38. PGD - Perturbation histogram MLP_wb

Per ottenere un'accuracy nulla anche sul classificatore KNN è necessario un valore di epsilon_step = 0.5. Si ottiene l'accuracy a 0% considerando un valore di ε = 1.5

Di seguito, è riportata la SEC relativa all'attacco generato attraverso la configurazione costituita da max_iter = 5 e epsilon_step = 0.5.

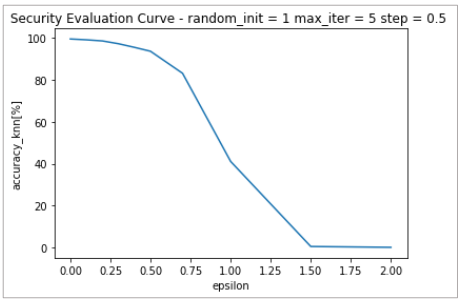

Figura 39. PGD - S.E.C. KNN_bb

Ulteriori analisi hanno dimostrato che aumentando i valori di num_random_init, max_iter e eps_step, si hanno situazioni analoghe. Si è notato, però, che per valori di max_iter = 5 ed eps_step < 0.3 l'attacco risulta poco trasferibile e, quindi, poco efficace.

Error-Specific

I risultati relativi alla versione Error-Specific sono molto simili a quelli della versione Error-Generic.

Si può vedere che con la stessa configurazione del caso Error-Generic (random_init = 1, max_iter = 5 ed epsilon_step = 0.3) l'accuracy del classificatore MLP White-Box si azzera a partire da un valore di $\varepsilon \simeq 0.7$.

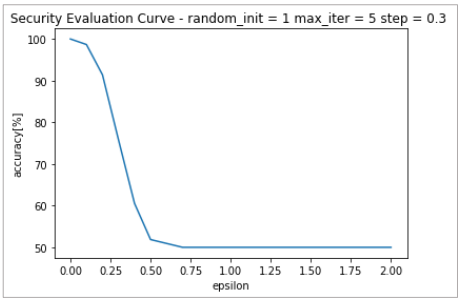

Figura 40. PGD - S.E.C. MLP_wb

Anche in questo caso, l'accuracy del classificatore KNN Black-Box, per un valore di $\varepsilon = 2$, non raggiunge il 50%.

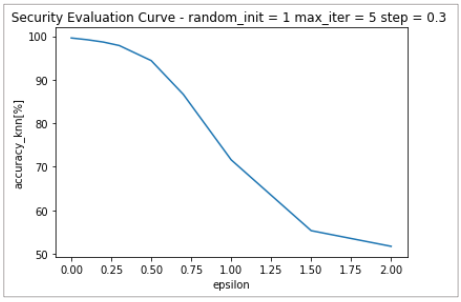

Figura 41. PGD - S.E.C. KNN_bb

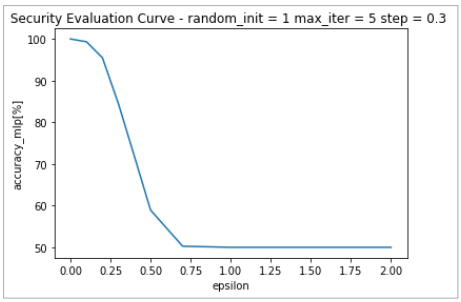

Figura 42. PGD - S.E.C. MLP_bb

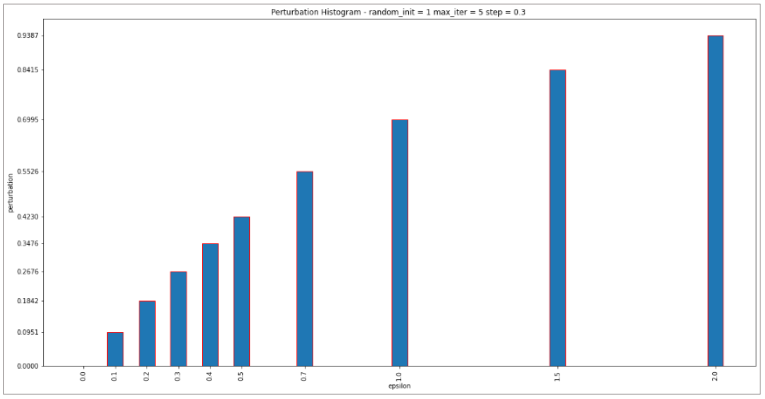

Figura 43. PGD - Perturbation histogram MLP_wb

Per ottenere accuracy allo 0% nel KNN Black-Box è necessario un valore di epsilon_step = 0.5.

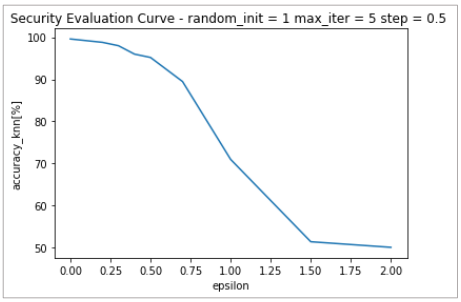

Figura 44. PGD - S.E.C. KNN_bb

Mantenendo costante random_init = 1 e aumentando max_iter ed epsilon_step, si ottengono prestazioni simili a quelle sopra riportate.

Anche aumentando random_init a 5, affinché l'attacco abbia successo, occorre almeno max_iter = 5 e epsilon_step = 0.5, in modo da ottenere anche una buona trasferibilità. Invece, aumentando random_init a 10 e max_iter a 10, basta avere un epsilon_step = 0.3.

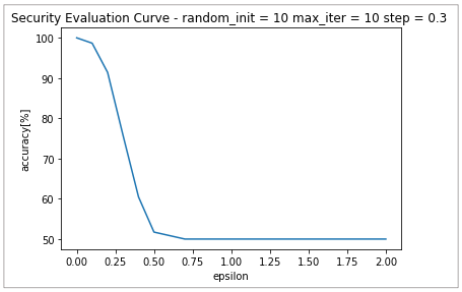

Figura 45. PGD - S.E.C. MLP_wb

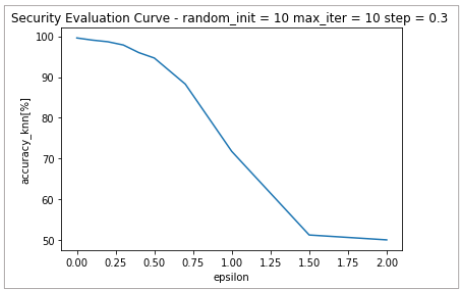

Figura 46. PGD - S.E.C. KNN_bb

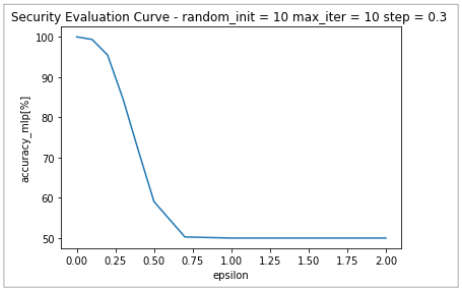

Figura 47. PGD - S.E.C. MLP_bb

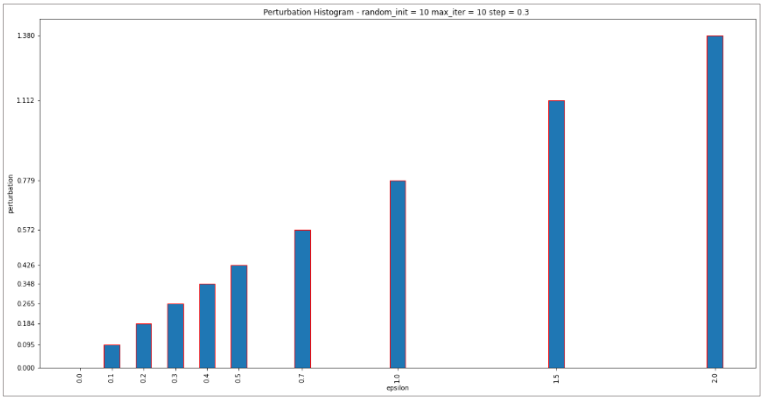

Figura 48. PGD - Perturbation histogram MLP_wb

CW (Carlini-Wagner)

L'algoritmo CW ha l'obiettivo di generare un campione corrotto con la minima perturbazione possibile. In particolare, si tratta di un problema di ottimizzazione, che prevede di minimizzare una funzione di costo che rappresenta la distanza tra il campione originale e quello corrotto, in modo che quest'ultimo sia riconosciuto come un campione avente una classe differente da quella.

Il problema di questo algoritmo è l'elevato tempo di generazione degli attacchi. Inoltre, attacchi di questo tipo sono molto efficaci rispetto al classificatore bersagliato, ma non godono di molta trasferibilità. In più, devono essere sferrati escludendo l'ultimo livello della rete.

I parametri settati in questo attacco sono:

- **max_iter**: numero massimo di iterazioni.
- **confidence**: confidence degli adversarial sample. Valori alti producono sample lontani da quelli originali, che sicuramente faranno sbagliare la predizione.
- **initial_consts**: costante di trade-off iniziale, da utilizzare per regolare l'importanza relativa alla distanza e alla confidence.
- **binary_search_steps**: numero di volte per regolare la costante con la ricerca binaria. Se ha un valore alto, l'algoritmo non sarà molto sensibile al valore di initial_consts.
- **learning rate**: il tasso di apprendimento iniziale per l'algoritmo di attacco. Valori più piccoli producono risultati migliori ma sono più lenti a convergere.

I parametri settati in questo attacco sono:

- `initial_consts = [0, 50, 100, 500, 1500]`
- `binary_search_steps = 7`
- `max_iter = 5`
- `confidence = 0.5`
- `learning_rate = 0.01`

Error-Generic

Con i parametri appena citati, si ottengono i risultati mostrati di seguito.

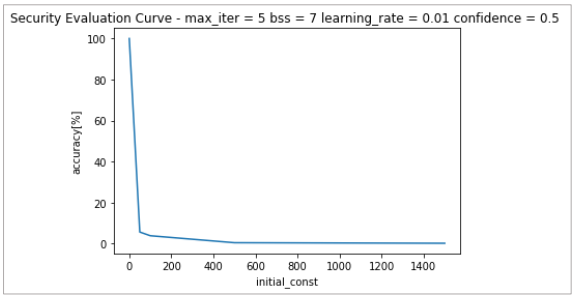

Figura 49. CW - S.E.C. MLP_wb

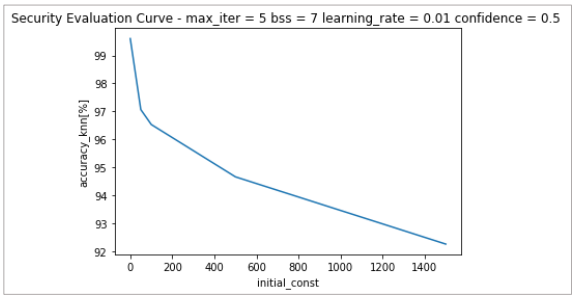

Figura 50. CW - S.E.C. KNN_bb

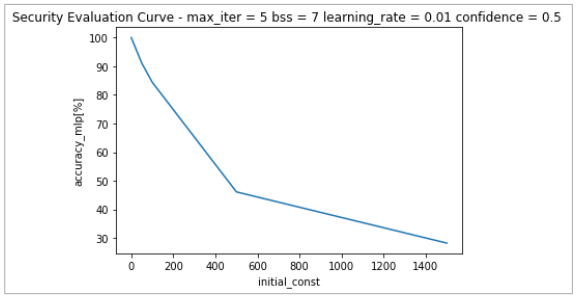

Figura 51. CW - S.E.C. MLP_bb

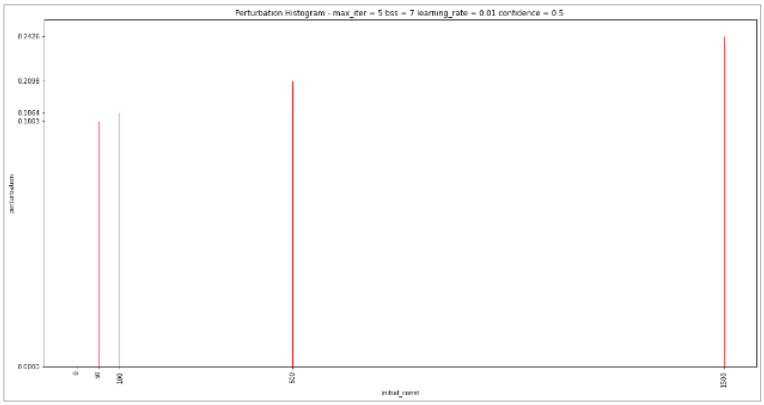

Figura 52. CW - Perturbation histogram MLP_wb

Osservando questi grafici, si può notare che sul classificatore MLP White-Box, l'attacco è efficace con alti valori di initial_consts (almeno pari a 500) con una perturbazione di 0.2. Come volevasi dimostrare, per ottenere un attacco efficace in grado di azzerare l'accuracy del classificatore, serve una perturbazione minore rispetto agli attacchi visti precedentemente.

Come previsto, l'attacco è poco trasferibile nel caso del classificatore MLP Black-Box (che è il classificatore più simile a quello White-Box) e per niente trasferibile nel caso del classificatore KNN Black-Box (che è il classificatore più lontano a quello White-Box).

Error-Specific

Lo stesso discorso fatto per il caso Error-Generic vale per quello Error-Specific.

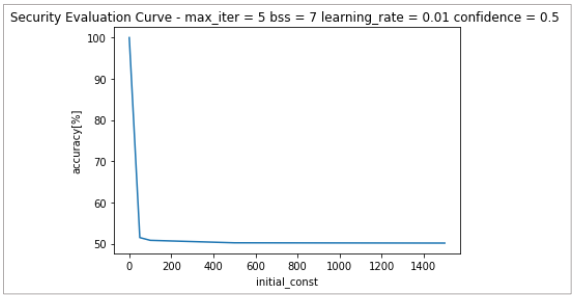

Figura 53. CW - S.E.C. MLP_wb

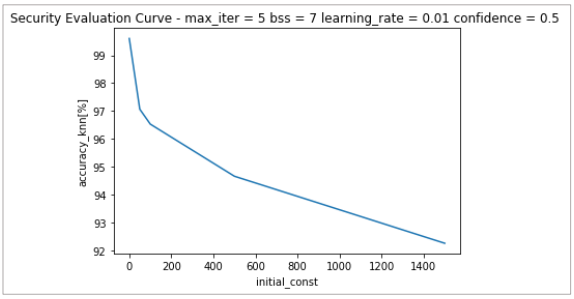

Figura 54. CW - S.E.C. KNN_bb

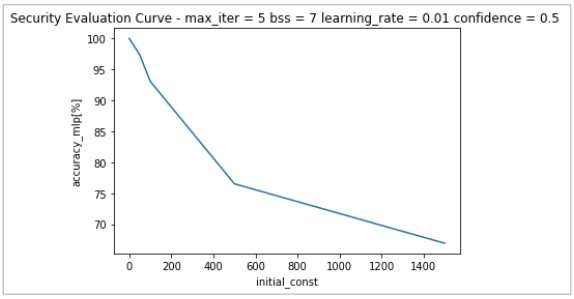

Figura 55. CW - S.E.C. MLP_bb

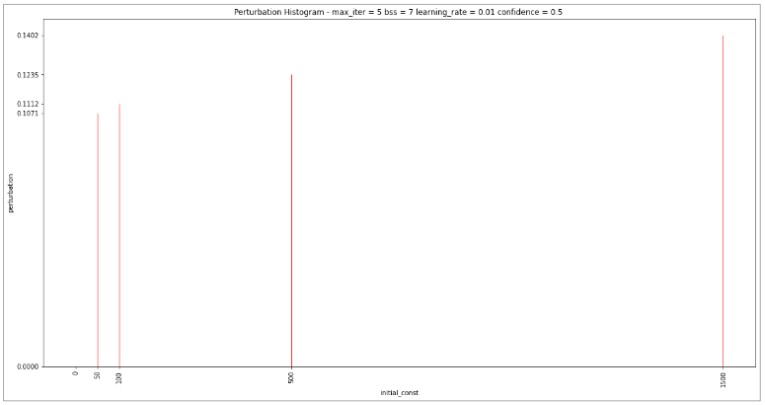

Figura 56. CW - Perturbation histogram MLP_wb

Dopo aver analizzato i risultati prodotti, si può affermare che l'attacco più trasferibile è FGSM.

Simulazione Difensiva & Analisi

Esistono due approcci difensivi: reattivo e proattivo. Il primo rende immuni solo rispetto ad attacchi passati, mentre il secondo aiuta a prevenire gli attacchi futuri, dato che si considera la sicurezza in fase di progettazione (Security by Design).

Esistono diverse tipologie di difesa proattiva. Alcuni esempi:

- Adversarial Training
- Image Pre-Processing before classification
- Detection of Adversarial Sample
- Gradient Masking
- Robust Optimization

In questa fase, sono state implementate e testate due strategie di difesa: **Adversarial Training** e **Detection of Adversarial Sample**.

Detection of Adversarial Sample

Questa tecnica mira a identificare i campioni corrotti in input e a evitare che siano utilizzati dal classificatore bersagliato. Dunque, la difesa si basa sulla costruzione di un classificatore binario in grado di distinguere i campioni in input originali da quelli manipolati dagli attaccanti, ovvero un detector.

Il detector viene addestrato utilizzando i campioni in input originali come classe positiva e quelli corrotti come classe negativa; in questo modo, agendo da filtro, evita che al classificatore arrivino campioni corrotti in input.

Per la realizzazione di questo meccanismo, è stata sfruttata la classe **BinaryInputDetector** fornita dalla libreria ART, addestrando un detector su un Training Set costituito da campioni originali e campioni avversariali opportunamente generati. In particolare, il Training Set è stato costruito attraverso la concatenazione tra i campioni del Training Set usato per allenare il classificatore da difendere e i campioni avversariali generati da attacchi di tipo FGSM, BIM e PGD. I parametri utilizzati per la creazione delle istanze degli attacchi sono stati scelti in seguito ad una fase di tuning ed un'analisi dei risultati, con l'obiettivo di ottenere la miglior difesa possibile.

Per valutare l'efficacia della difesa, sono state create le istanze degli attacchi considerati e sono stati generati i campioni avversariali a partire dai campioni del Test Set (750). Successivamente, i campioni corrotti sono stati forniti in input al classificatore MLP White-Box e al detector e sono stati messi a confronto il numero di campioni corrotti classificati in modo scorretto dal classificatore MLP White-Box e il numero di campioni corrotti classificati come tali dal detector al variare della forza dell'attacco. I risultati ottenuti sono accettabili, in quanto il detector riesce a riconoscere in quasi tutti i casi la totalità dei campioni avversariali fornitigli in input; infatti, le curve rappresentanti il numero di campioni riconosciuti come avversariali si trovano sempre al di sopra di quelle rappresentanti il numero di campioni sbagliati dal classificatore.

Infine, in seguito alla classificazione dei campioni avversariali di tutti gli attacchi generati, è stata calcolata l'accuracy del detector, che è risultata pari circa a 68%.

Di seguito, sono riportate le curve che mostrano le prestazioni del detector ottenuto rispetto agli attacchi considerati.

FGSM

Error-Generic

Si consideri che il primo valore di ε considerato è 0. Questa scelta è stata effettuata per valutare le prestazioni anche in assenza di attacco. Infatti, la curva relativa al detector assume valore 0 per ε = 0, dato che si è in assenza di campioni corrotti.

Per un valore di ε ≃ 0.1, il detector riconosce quasi la totalità dei campioni di test dati in input come corrotti (750). Inoltre, all'aumentare della forza dell'attacco la curva si assesta al valore 750.

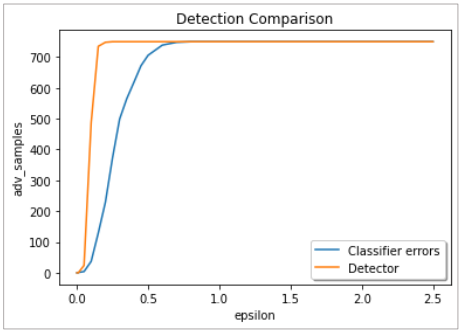

Figura 57. FGSM - Detector MLP_wb

Error-Specific

L'andamento della curva relativa al detector rivela prestazioi leggermente minori rispetto al caso Error-Generic. In particolare, il detector riesce a riconoscere la quasi totalità dei campioni corrotti a partire da ε ≃ 0.7; inoltre, all'aumentare della forza dell'attacco le sue prestazioni migliorano.

Si ricordi che, siccome il Test Set è bilanciato (375 positivi e 375 negativi), l'attacco raggiunge la massima intensità quando il classificatore MLP White-Box sbaglia 375 classificazioni. È questo il motivo per cui la curva blu non supera questo valore.

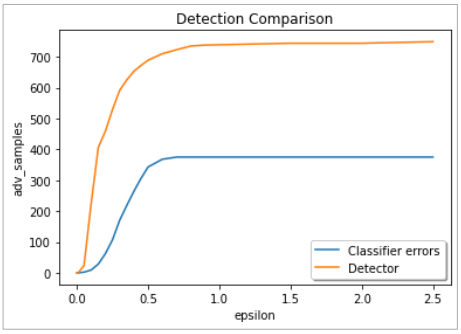

Figura 58. FGSM - Detector MLP_wb

BIM

Error-Generic

Il grafico riportato di seguito mostra i risultati relativi alla configurazione costituita da max_iter = 5 e epsilon_step = 0.3, ovvero una di quelle che produce il risultato migliore in fase di attacco.

Anche in questo caso, è possibile osservare che in assenza di attacchi il detector non rileva campioni corrotti e, all'aumentare della forza dell'attacco, li rileva tutti a partire da un valore di $\varepsilon \simeq 0.3$.

Inoltre, la curva relativa al detector si trova sempre al di sopra di quella relativa al classificatore MLP White-Box.

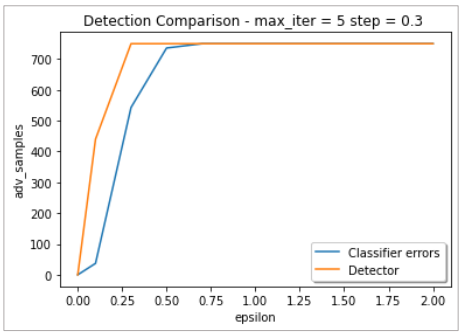

Figura 59. BIM - Detector MLP_wb

Error-Specific

L'andamento della curva gialla dimostra che le performance del detector nel caso di attacco BIM Error-Specific sono minori di quelle relative al caso Error-Generic.

In particolare, la curva assume valori sempre maggiori all'aumentare della forza dell'attacco fino ad assestarsi ad un valore alto in corrispondenza di un valore di $\varepsilon \simeq 1.5$.

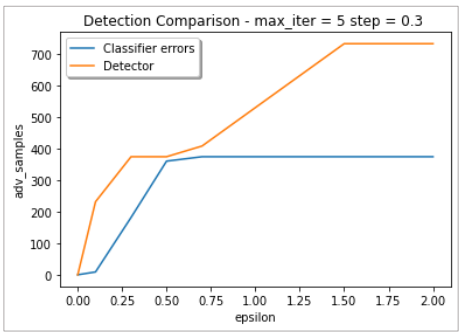

Figura 60. BIM - Detector MLP_wb

PGD

Error-Generic

Dal grafico è possibile osservare che in assenza di attacchi le due curve assumono valore 0. Inoltre, a partire da un $\varepsilon \simeq 0.25$, il detector riconosce la totalità dei campioni corrotti.

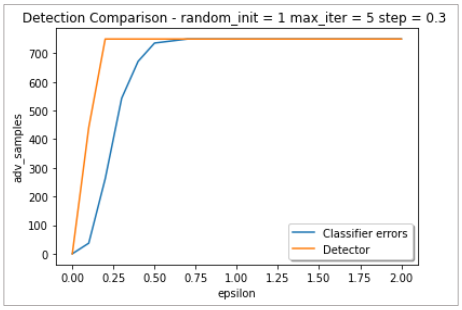

Figura 61. PGD - Detector MLP_wb

Error-Specific

Anche in questo caso, nella versione Error-Specific si ha risultato meno efficace rispetto alla versione Error-Generic. Il detector, infatti, riconosce la totalità dei campioni corrotti solo a partire da un valore di $\varepsilon = 2$.

Come nel caso dell'attacco BIM Error-Specific analizzato in precedenza, la curva gialla presenta un andamento sempre crescente all'aumentare della forza dell'attacco.

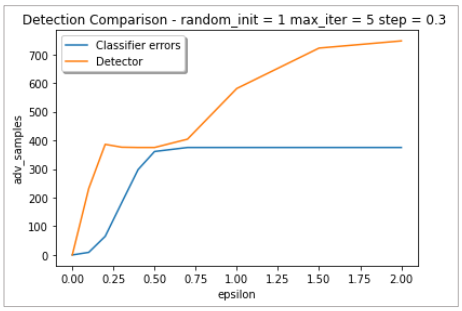

Figura 62. PGD - Detector MLP_wb

CW

Error-Generic

Nonostante il detector non sia stato allenato su campioni corrotti generati da attacchi Carlini-Wagner, è stata effettuata anche un'analisi relativa a questo tipo di attacco. In particolare, la curva relativa al detector dimostra che quest'ultimo riconosce una quantità di campioni corrotti elevata a partire da un valore di initial_const = 50.

Questo è un risultato accettabile considerando che si tratta di un attacco molto efficace, come è possibile notare dall'andamento della curva relativa agli errori commessi dal classificatore MLP White-Box.

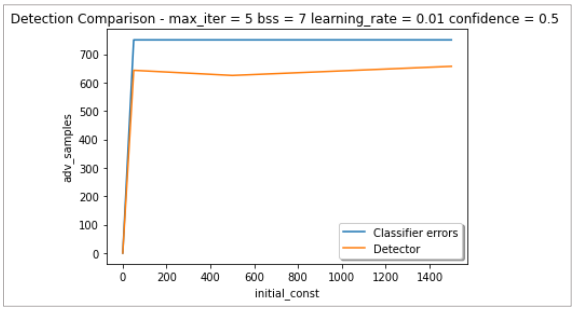

Figura 63. CW - Detector MLP_wb

Error-Specific

Anche in questo caso, il comportamento del detector è meno efficace rispetto alla versione Error-Generic.

Inoltre, come nel caso Error-Generic, è possibile notare la rilevante efficienza dell'attacco.

Per valori di initial_const > 50, però, il detector riesce a riconoscere un numero tollerabile di campioni corrotti.

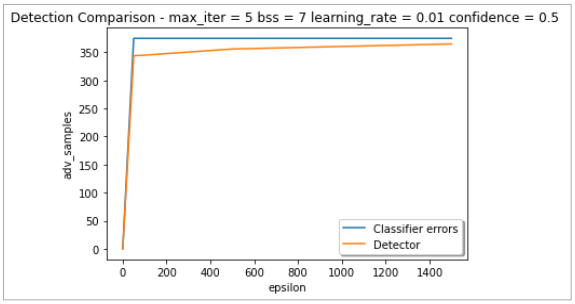

Figura 64. CW - Detector MLP_wb

Adversarial Training

Questa tecnica prevede di generare degli adversarial sample e di aggiungerli al Training Set, in modo da allenare il classificatore a riconoscere nel modo corretto anche i sample corrotti. In particolare, gli adversarial sample vengono ottenuti con una data augmentation sui campioni originali, effettuata automaticamente grazie alle funzioni di libreria utilizzate.

In questo modo la generalizzazione viene aumentata, ma è possibile che il classificatore perda accuratezza sui campioni originali; per questo motivo, è sempre necessario verificare anche le prestazioni su questi ultimi.

Grazie alla classe **AdversarialTrainer** fornita dalla libreria ART, a partire dal modello del classificatore iniziale, è stato creato un nuovo classificatore che in seguito è stato allenato su campioni avversariali generati sulla base di alcuni attacchi. Precisamente, sono state utilizzate le istanze di attacchi di tipo BIM e PGD (come consigliato da Madry et al. 2017, *Towards Deep Learning Models Resistant to Adversarial Attacks*), con parametri scelti opportunamente in seguito ad una fase di tuning.

Prima di valutare le prestazioni del classificatore robusto ottenuto sui campioni avversariali, sono state valutate quelle sul Test Set originale, così da assicurarsi di non aver perso accuratezza sui campioni originali. Il risultato di questa valutazione ha prodotto un'accuracy del 99.8% e ciò vuol dire che il classificatore non ha subito un calo significativo di prestazioni sul dataset originale in seguito all'Adversarial Training.

In seguito, il nuovo classificatore è stato testato sugli attacchi considerati, sia nelle versioni Error-Generic che in quelle Error-Specific. Infine, è stato calcolato l'accuracy del classificatore robusto in seguito alla classificazione dei campioni avversariali di tutti gli attacchi generati. Il valore ottenuto è circa 77%.

Di seguito sono riportati i grafici che confrontano la Security Evalutaion Curve del classificatore iniziale e quella del classificatore robusto per tutti gli attacchi considerati.

FGSM

Error-generic

Anche in questo caso, come per la Detection of adversarial sample, si consideri che il primo valore di ε considerato è 0. In questo caso, però, per tale valore non si ha un'accuracy pari al 100% a causa della piccola perdita dovuta all'Adversarial Trainer.

Per un valore di $\varepsilon \simeq 0.5$, il modello riconosce quasi la totalità dei campioni di test dati in input e riesce a classificarli correttamente come positivi o negativi; al contrario, è possibile notare come il classificatore originale non riesca ad attuare tale distinzione. All'aumentare di ε, la curva decresce mostrando una perdita di accuratezza, fino ad assestarsi attorno al 50% per $\varepsilon \simeq 1$. Questo evidenzia il vantaggio di tale tecnica, poiché il classificatore originale avrebbe sbagliato la classificazione in ogni caso, mentre il classificatore addestrato al più commette un errore ogni due classificazioni.

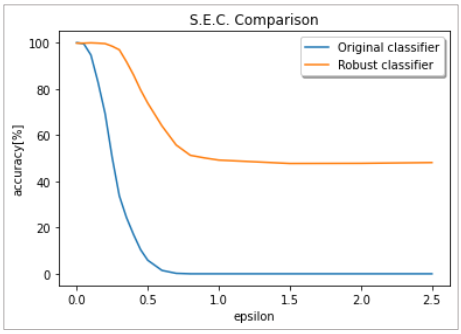

Figura 65. FGSM - Adversarial MLP_wb

Error-Specific

L'accuratezza del modello rileva un netto miglioramento rispetto al caso Error-Generic. In particolare, la curva non si discosta di molto dal caso ottimo, ossia quello costante al 100%, indicando una quasi perfetta identificazione dei campioni negativi che tentano di farsi classificare come positivi.

Si ricordi che, siccome il Test Set è bilanciato (375 positivi e 375 negativi), il classificatore originale non scende mai al di sotto del 50%, anche in caso di ε elevate.

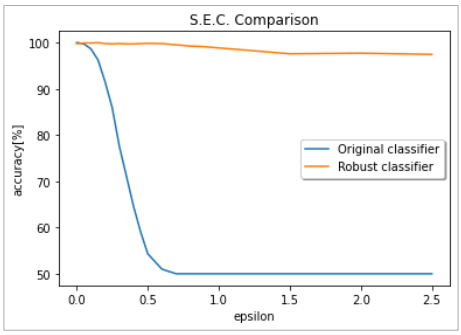

Figura 66. FGSM - Adversarial MLP_wb

BIM

Error-Generic

Come per il caso di FGSM, la curva arancione riesce a classificare correttamente per ε piccole, fino ad assestarsi su un valore di circa il 50% per $\varepsilon > 1$.

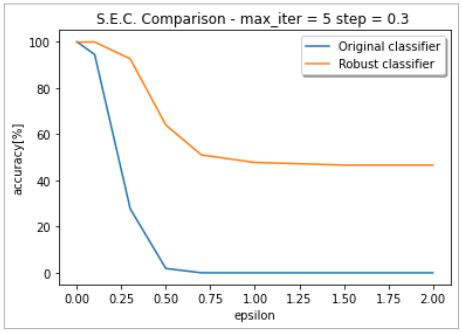

Figura 67. BIM - Adversarial MLP_wb

Error-Specific

L'accuratezza nel caso di campioni della classe negativa che tentano di farsi identificare come positivi non scende di molto al di sotto del 100%, indicando così un'ottima capacità di difesa.

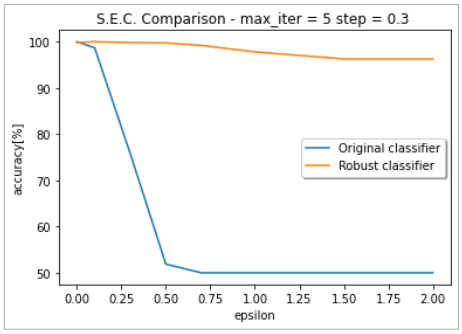

Figura 68. BIM - Adversarial MLP_wb

PGD

Error-Generic

L'aggiunta della randomness non evidenzia un miglioramento significativo rispetto al caso analizzato con BIM. Anche in questo caso, infatti, l'accuratezza è alta per ε basse e decresce fino a circa il 50% per $\varepsilon \simeq 1$.

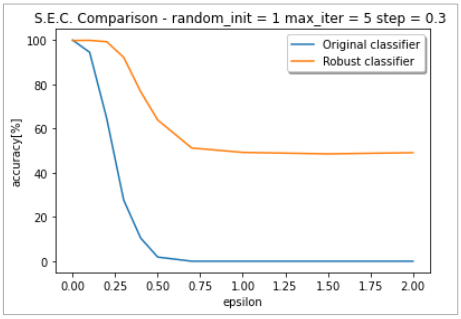

Figura 69. PGD - Adversarial MLP_wb

Error-Specific

L'accuratezza a differenza del BIM mostra una leggera decrescita per $\varepsilon \simeq 1.5$, ma la difesa continua ad essere estremamente valida.

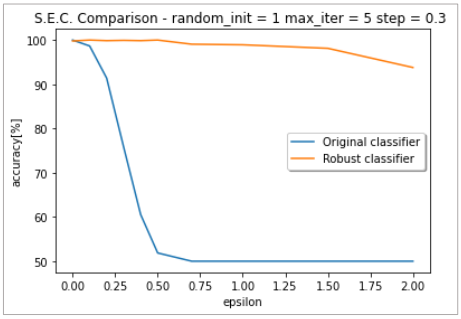

Figura 70. PGD - Adversarial MLP_wb

CW

Error-Generic

In questo caso, non avendo utilizzato CW per addestrare il classificatore, si ottengono prestazioni inferiori rispetto agli attacchi gradient based, ma sono ugualmente presenti dei lievi miglioramenti. La curva arancione, infatti, decresce in maniera più delicata rispetto a quella blu. Per valori di initial_const fino a 100 si riesce ad ottenere un miglioramento che decresce fino al 40%. Da initial_const = 100 fino a initial_const = 500 si ha una lieve decrescita che porta ad avere un'accuratezza di circa il 10% che resta costante all'aumentare di initial_const.

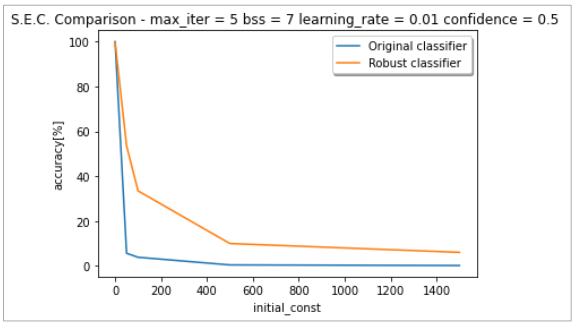

Figura 71. CW - Adversarial MLP_wb

Error-Specific

L'andamento è molto simile a quello che si ottiene nel caso di Error-Generic con la differenza che i valori decrescono fino ad un massimo del 50% mantenendo per valori iniziali di initial_const un guadagno di accuratezza maggiore. Per initial_const elevati si ottiene un leggero decadimento delle prestazioni che riporta allo stesso guadagno del caso precedente

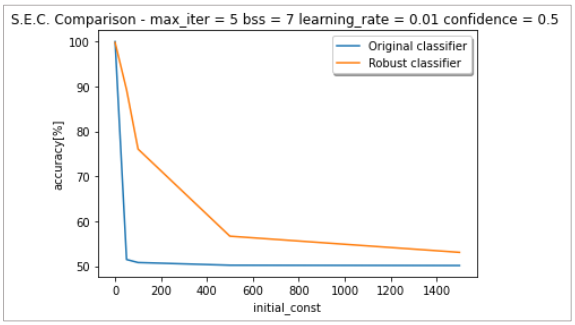

Figura 72. CW - Adversarial MLP_wb

Riferimenti

- https://adversarial-robustness-toolbox.readthedocs.io/en/latest/modules/attacks/evasion.html
- https://adversarial-robustness-toolbox.readthedocs.io/en/latest/modules/defences/detector_evasion.html
- https://adversarial-robustness-toolbox.readthedocs.io/en/latest/modules/defences/trainer.html
- https://librosa.org/doc/latest/index.html
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html