



Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale in Data Mining

Fingerprint Adversarial Presentation Attacks: From a Digital Issue To a Real World Concern

Anno Accademico 2020/2021

Relatori

Ch.mo Prof Carlo Sansone

Ing. Stefano Marrone

Candidato

**Umberto Gagliardini
matr. M63001023**

[Dedica]

SOMMARIO

Abstract.....	4
Capitolo 1: Introduzione.....	6
1.1 Sistemi di riconoscimento biometrici.....	7
1.1.1 Sistema basato su impronte digitali.....	8
1.2 Fingerprint-based Authentication System.....	8
1.3 Presentation Attacks.....	9
1.4 Liveness Detection.....	9
1.5 Adversarial Attacks.....	10
1.6 Stampa impronte perturbate.....	11
Capitolo 2: concetti base e letteratura.....	13
2.1 Fingerprint Liveness Detection.....	14
2.2 Ingannare una convolutional neural network.....	17
2.2.1 Tipologie di attacchi.....	17
2.3 Evasion attacks.....	18
2.3.1 Iterative Fast Gradient Sign Method.....	20
2.3.2 DeepFool.....	22
2.3.3 Auto - Projected Gradient Descent.....	23
2.4 Authentication System.....	29
2.4.1 Mindtct: Minutiae Detection.....	29
2.4.2 Bozorth: Fingerprint Matcher.....	33
Capitolo 3: Approccio proposto.....	35

3.1 Fingerprint Liveness Dataset.....	36
3.1.1 Scanners.....	37
3.1.2 Materiali.....	38
3.2 CNN per Liveness Detection.....	39
3.2.1 Testing.....	42
3.2.2 Scanner GreenBit e Latex.....	43
3.3 Attacco liveness detector.....	44
3.3.1 Fingerprint adversarial perturbations.....	44
3.3.2 V1-V2: Enhancement.....	51
3.3.3 V3: Random ROI.....	60
3.3.4 V4: Dilatation.....	64
3.3.5 V5: Pepper Noise.....	67
3.5 Authentication.....	70
3.6 Printed Fingerprints.....	72
Capitolo 4: Risultati.....	76
4.1 Digital.....	77
4.1.1 Binarized Clean.....	77
4.1.2 Attacks.....	78
4.2 Studio rumore.....	80
4.3 Printed.....	81
4.3.1 Studio relativo: Binarized vs Perturbed.....	83
4.3.2 Studio incrementale.....	84
Conclusioni.....	86
Sviluppi futuri.....	87
Bibliografia.....	88

ABSTRACT

La crescente disponibilità di scanner economici ed affidabili per l'acquisizione delle impronte ha causato una crescente diffusione di sistemi di autenticazione basati su impronte digitali (Fingerprint-based Authentication System: FAS) nell'elettronica di consumo.

Ciò ha dato vita ad una nuova ondata di ricerca riguardo attacchi intelligenti, volti a bypassare un FAS utilizzando un'impronta digitale contraffatta, ovvero un'impronta creata artificialmente che presenta le stesse particolarità dell'originale (*Presentation Attack*).

Come conseguenza, dal punto di vista della sicurezza, è diventata essenziale l'introduzione in ogni FAS di un “rilevatore di vitalità”, ovvero un *liveness detector* (LD), in grado di discernere l'autentico dal contraffatto.

Come in molti altri compiti di computer vision, le reti neurali convoluzionali (CNN) si sono dimostrate molto efficaci anche per il rilevamento della vitalità delle impronte digitali. Tuttavia, è stato dimostrato che è possibile adattare approcci perturbativi contraddittori per fuorviare LD basati sulle CNN (*Adversarial Attacks*).

Lo scopo di questa tesi è quello di effettuare un ulteriore passo in avanti verso la progettazione di un attacco *white-box* completo che sia in grado

di far sbagliare i *liveness detectors* che rappresentano lo stato dell'arte ed allo stesso tempo riesca ad ingannare anche il sistema di autenticazione.

Inoltre, in collaborazione con l'Università degli Studi di Cagliari è stata effettuata un'attività di stampa delle immagini perturbate al fine di studiare il comportamento delle immagini e di conseguenza degli scanner.

I risultati mostrano che è possibile sfruttare le perturbazioni contraddittorie per fuorviare sia il rilevatore di vitalità che il sistema di autenticazione, dando origine a delle impronte stampate su lattice che possono essere utilizzate concretamente contro uno dei migliori scanner in circolazione: GreenBit.

Lo scopo ultimo del lavoro realizzato è quello di mostrare le vulnerabilità dei migliori *liveness detectors* in circolazione, non solo nel mondo digitale, ma anche nel mondo fisico, enfatizzando in questo modo le problematiche relative all'utilizzo di CNN per i LD.

CAPITOLO 1: INTRODUZIONE

Nel corso del primo capitolo si vuole fornire un'introduzione ai sistemi di riconoscimento biometrici ed alla possibilità di aggirarli utilizzando i *presentation attacks*, attacchi volti ad ingannare il sistema presentando un dato biometrico contraffatto.

Si introduce il tipo di sistema biometrico considerato durante il lavoro, ovvero quello basato su impronte digitali.

Inoltre, viene descritto un concetto alla base del lavoro svolto: la *liveness detection*, la quale nasce per contrastare gli attacchi precedentemente accennati e, di conseguenza, vengono menzionati ulteriori attacchi creati per aggirare quest'ultimo sistema di sicurezza: attacchi basati su perturbazioni.

Infine, il tutto converge verso la possibilità di stampare i risultati di questi attacchi nel modo migliore possibile per ingannare i sistemi biometrici nel mondo reale.

È importante menzionare la collaborazione con l'Università degli Studi di Cagliari, la quale è da anni esperta nella ricerca dei sistemi biometrici ed ha effettuato la stampa delle impronte perturbate create con la metodologia descritta in seguito.

1.1 Sistemi di riconoscimento biometrici

Nell'era della connettività mobile, della tecnologia IoT e del cloud computing i tradizionali metodi di accesso basati su ID utente e password per autenticare l'identità digitale non riescono a fronteggiare in maniera efficiente le nuove minacce informatiche, per questo si sta puntano sempre più sul **riconoscimento biometrico** con lo scopo di aumentare la sicurezza dei sistemi collegati in rete e di proteggere i dati.

Un **sistema di riconoscimento biometrico** è un particolare tipo di sistema informatico che ha lo scopo di identificare una persona sulla base di una o più caratteristiche fisiologiche e/o comportamentali (**biometria**) confrontandole con i dati precedentemente acquisiti e presenti nel database del sistema, tramite degli algoritmi e dei sensori di acquisizione dei dati in input.

La **biometria** è la «*disciplina che studia le grandezze biofisiche allo scopo di identificarne i meccanismi di funzionamento, di misurarne il valore e di indurre un comportamento desiderato in specifici sistemi tecnologici*».

La biometria permette di stabilire che ogni individuo ha caratteristiche:

- universali, tutti devono averla;
- uniche, due o più individui non possono avere la stessa uguale caratteristica;
- permanenti, questa non varia nel tempo;
- collezionabili, deve essere misurata quantitativamente.

Gli identificatori biometrici sono caratteristiche distintive e misurabili usate per etichettare e descrivere un individuo. Gli identificatori biometrici sono spesso classificati come caratteristiche fisiche o caratteristiche comportamentali. Le **caratteristiche fisiche** si riferiscono alla forma del corpo, sono abbastanza stabili, soggette solo a piccole

variazioni nel tempo. Alcuni esempi possono essere le impronte digitali, le vene delle dita o del palmo della mano, la forma del viso, il DNA, la retina dell'occhio. Le **caratteristiche comportamentali**, invece, sono strettamente legate alle abitudini di una persona, come la firma e l'impronta vocale e per questo possono essere influenzate dalla situazione psicologica dell'individuo, quindi devono essere aggiornate spesso.

I metodi più tradizionali del controllo sugli accessi includevano sistemi di identificazione basati su carte di riconoscimento e sistemi basati sulla conoscenza di informazioni, quali password o PIN. Dato che gli identificatori biometrici sono unici da individuo a individuo, sono molto più affidabili nel verificare l'identità rispetto a metodi basati su documenti o informazioni.

1.1.1 Sistema basato su impronte digitali

L'impronta digitale è probabilmente la più utilizzata e accettata forma di riconoscimento biometrico. L'ampio utilizzo dell'impronta digitale quale tecnologia di riconoscimento biometrico si basa su due principi basilari: l'immutabilità e l'individualità.

Con il termine **immutabilità** ci si riferisce al fatto che le impronte digitali, che si formano durante la gravidanza, non cambiano attraverso il tempo; l'**individualità**, invece, consiste nella certezza che le macro e micro-caratteristiche dell'impronta sono uniche da individuo a individuo.

1.2 Fingerprint-based Authentication System

Mediante un apposito scanner, i dati biometrici riguardanti le impronte digitali del soggetto vengono riprodotti su un supporto elettronico per poi essere convertiti attraverso appositi software in forma digitale mediante specifici algoritmi. Il sistema di rilevazione biometrica, successivamente, confronta le fattezze maggiori e minori e consente di ricostruire le

particolarità dell'anatomia dell'impronta e individuare l'identità del soggetto.

I vantaggi di questo tipo di tecnologia biometrica consistono in primo luogo nella rapidità e semplicità di rilevamento, che può essere effettuato mediante uno scanner relativamente economico. In secondo luogo, le procedure di rilevamento sono state sviluppate e migliorate nel corso di oltre un secolo, pertanto si può affermare che l'affidabilità della procedura è buona. Infine, la cattura dell'impronta digitale non è influenzata dalle caratteristiche dell'ambiente esterno. Tra gli svantaggi, invece, si colloca la possibilità che una piccola percentuale della popolazione potrebbe avere dei deterioramenti cutanei che renderebbero difficile o addirittura impossibile il rilevamento delle impronte digitali.

1.3 Presentation Attacks

Con l'avanzare della ricerca è stato scoperto che i sensori biometrici possono essere aggirati attraverso l'utilizzo di riproduzioni artificiali di una determinata caratteristica fisiologica e, ovviamente, ciò vale anche per le impronte digitali.

Questi attacchi sono chiamati ***presentation attack*** e sono definiti come l'attività di presentazione di dati biometrici al sottosistema di acquisizione con l'obiettivo di interferire con il funzionamento del sistema biometrico.

Le impronte digitali, in particolare, possono essere replicate artificialmente generando dei falsi, detti **artefatti**, capaci di aggirare il sistema di autenticazione grazie alla conservazione di caratteristiche fondamentali per il riconoscimento, come creste e solchi.

Gli artefatti possono essere generati in maniera **cooperativa**, ovvero l'individuo partecipa alla generazione della replica ponendo il proprio dito su un materiale malleabile (come silicone o gelatina) per creare lo stampo, oppure **non cooperativa**, in cui si cerca di estrarre l'impronta

da oggetti senza la collaborazione dell'individuo (per esempio tramite fotografie) per poi successivamente passare alla stampa.

1.4 Liveness Detection

La *liveness detection* nasce per contrastare i *presentation attacks* e la sua funzionalità consiste nel classificare il dato biometrico presentato in ingresso come **live**, cioè appartenente ad una persona reale, o **spoof**, ovvero un artefatto.

È possibile, quindi, rappresentare un sistema di riconoscimento biometrico come la serie di due sottosistemi: **Liveness Detector** e **Authentication System**; in questo modo il dato biometrico sarà sottoposto all'AS solo se supera il controllo della *liveness*.

Di seguito è riportato un esempio riguardante un FAS:

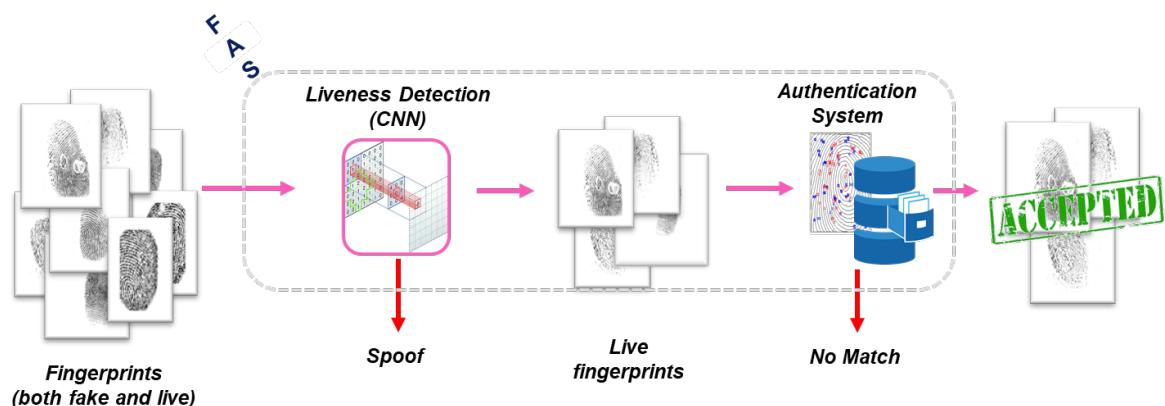


Figura 1 - FAS

La rilevazione di vitalità in un'impronta digitale si basa sul principio secondo il quale è possibile raccogliere informazioni aggiuntive dai dati acquisiti dal sensore e sfruttare tali informazioni per determinare se l'impronta digitale è autentica o falsa.

1.5 Adversarial Attacks

Con i costanti progressi ottenuti nel tempo, le tecniche di *deep learning* si sono affermate sempre di più in molti ambiti diversi, uno di questi è proprio la *liveness detection*. In particolare, nel 2015, durante la LivDet¹ sono state utilizzate le *Convolutional Neural Network (CNN)* per l'implementazione dei *liveness detectors* e ciò ha portato ad un aumento dell'*accuracy*: si passa da un 70% degli anni precedenti al 90%. Da quell'anno, quindi, le CNNs si sono affermate come stato dell'arte per la realizzazione di LD efficienti.

Negli ultimi anni l'utilizzo delle CNNs ha condotto verso un miglioramento del grado di precisione nel rilevamento della vitalità rispetto agli approcci di *image processing* utilizzati in passato introducendo, però, alcuni problemi e vulnerabilità tipiche delle reti neurali. Esse, infatti, sono molto sensibili agli input; un qualsiasi classificatore può essere ingannato per fornire delle predizioni completamente errate attraverso un'opportuna perturbazione degli input. Queste perturbazioni vengono chiamate **adversarial perturbation** e gli algoritmi in grado di generarle vengono detti **adversarial attack**.

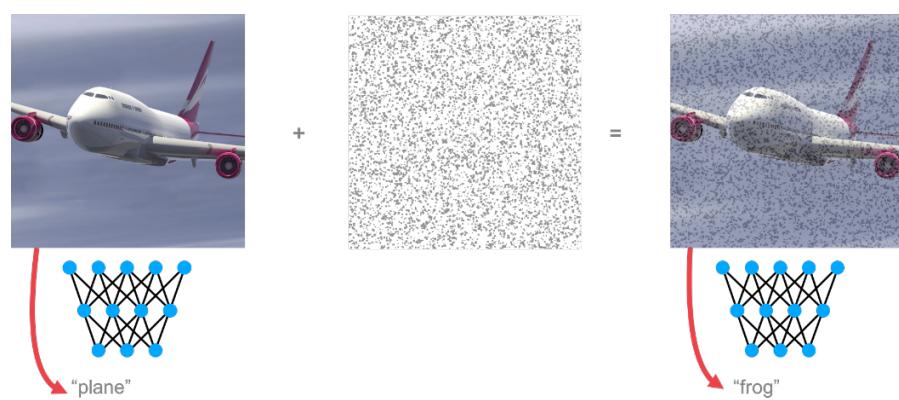


Figura 2 - Esempio adversarial attack

¹ *Fingerprint Liveness Detection Competition*, una competizione che si svolge con cadenza biennale finalizzata alla ricerca e sviluppo di nuove tecniche nell'ambito della *fingerprint liveness detection*

La vulnerabilità accennata ha portato i ricercatori a sviluppare molte tecniche di attacco che possono essere classificate in attacchi **white-box** e **black-box**: i primi sono attacchi in cui l'aggressore ha piena conoscenza dei parametri e dei gradienti della rete (o una loro stima), mentre i secondi sono attacchi in cui l'aggressore non ha alcuna informazione riguardo la rete da attaccare. Gli attacchi white-box, inoltre, si dividono in mirati e non mirati, dove i primi tentano di spingere i classificatori a sbagliare senza tener conto della classe di output, mentre i secondi cercano di indurre i classificatori a predire una determinata classe finale.

È facile intuire che gli attacchi *white-box* hanno maggiore probabilità di successo poiché utilizzano i valori della rete negli algoritmi matematici, ma allo stesso tempo nella realtà è difficile ottenere la struttura interna di una rete senza averne l'autorità. Viceversa, gli attacchi *black-box* hanno meno possibilità di successo ma possono essere eseguiti senza particolari conoscenze. Bisogna precisare che con l'avanzare della ricerca stanno nascendo sempre più tecniche che permettono di "stimare" una rete per poter sferrare un attacco *white-box* e, allo stesso tempo, stanno nascendo anche tecniche che permettono di migliorare le reti, per esempio addestrandole con immagini perturbate.

1.6 Stampa impronte perturbate

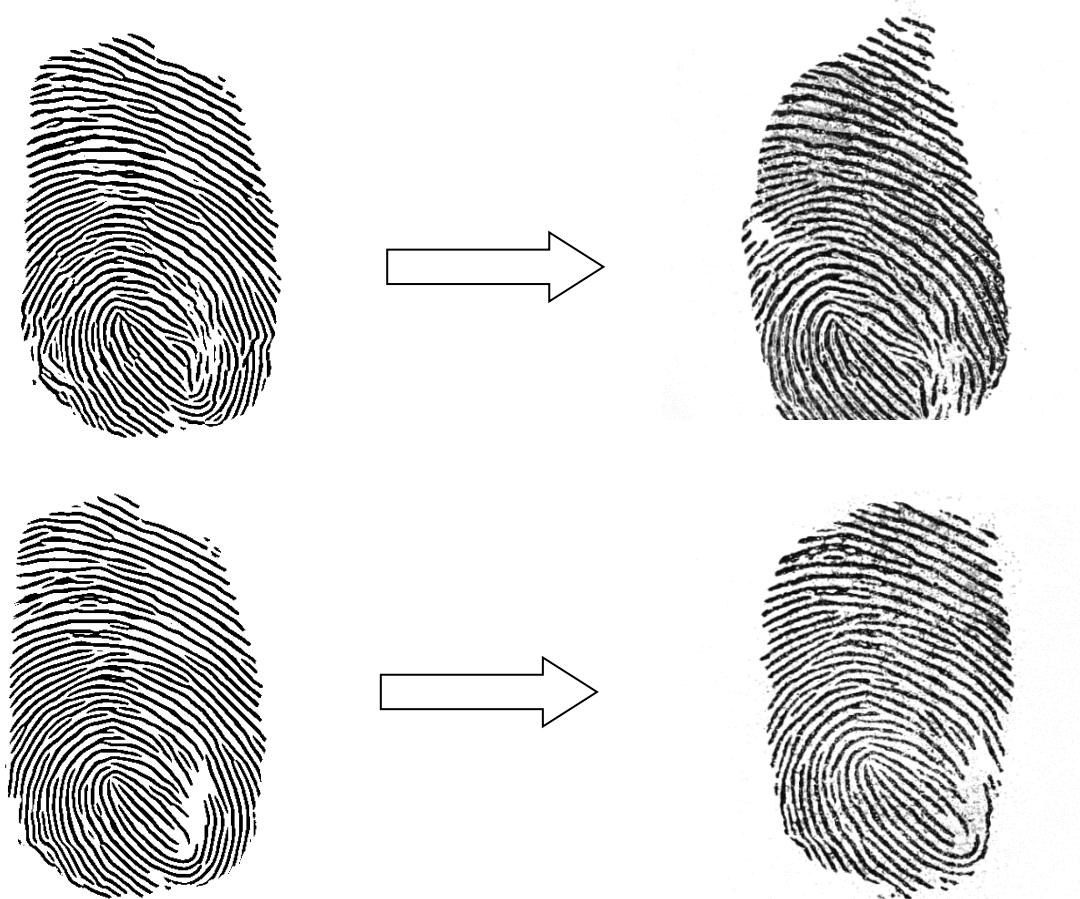
La fase finale di un attacco ad un sistema biometrico basato su impronte consiste nella stampa dell'artefatto opportunamente perturbato su un determinato materiale.

Durante la stampa sono vari i fattori che incidono, come la capacità dell'operatore umano, la qualità dei materiali e la robustezza dell'immagine perturbata da stampare; infatti, non è esagerato dire che la stampa è la fase più importante dato che se l'impronta non viene

stampata bene si perdono tutti i progressi ottenuti durante l'*adversarial attack*.

I problemi principali che si possono riscontrare in seguito alla stampa sono due: introduzione del rumore di fondo e ritaglio di parte dell'impronta. A causa di questi problemi l'impronta può essere riconosciuta come *spoof* o può non superare il *match* del sistema di autenticazione.

Di seguito sono riportati due esempi in cui due immagini digitali perturbate, create a partire dalla stessa immagine originale (ma con attacchi diversi), vengono stampate e poi riacquisite per testarle:



È semplice capire che la prima è riconosciuta *spoof*, mentre la seconda *live*, anche se stampate a partire da due impronte *live* simili; questo perché la prima presenta del rumore più accentuato ed è ritagliata, mentre la seconda, anche se presenta dei leggeri ritagli ai bordi,

conserva la sua forma originale e le sue caratteristiche non sono oscurate dal rumore.

CAPITOLO 2: CONCETTI BASE E LETTERATURA

In questo capitolo si vuole descrivere l'aspetto teorico dei concetti alla base della soluzione proposta.

Come già detto nel capitolo introduttivo, per aumentare la sicurezza di un sistema di autenticazione biometrico è necessario creare un *liveness detector*, quindi, *in primis* viene mostrato cosa fornisce la letteratura riguardo quest'argomento ed in particolare qual è lo stato dell'arte.

In seguito, si descrivono i vari modi in cui è possibile ingannare un rilevatore di vitalità e quale tipologia di attacco sarà al centro della tesi, ovvero gli *evasion attacks*.

Infine, dopo aver introdotto teoricamente i principali attacchi che saranno utilizzati nella soluzione, si descrive il sistema di autenticazione ed il suo funzionamento nel dettaglio.

2.1 Fingerprint Liveness Detection

Una strategia comune per rilevare l'uso di un “artefatto” come dato biometrico in ingresso consiste nel sottomettere una replica artificiale del dito al sensore, ovvero far imparare al sensore la struttura delle impronte contraffatte, dette *spoof*. Molti materiali possono essere utilizzati per questo scopo, compresi quelli economici e molto accessibili come la colla per legno.

La tecnica che permette di determinare se l'impronta digitale presentata appartiene ad un individuo reale o ad una replica è nota come *Liveness Detection* (LD); essa può sia sfruttare solo l'impronta digitale acquisita dallo scanner, sia sfruttare i dati proveniente da sensori aggiuntivi (come temperatura, pressione sanguigna, umidità). Se da un lato le informazioni fornite da hardware esterno possono migliorare il tasso di riconoscimento, dall'altro acquisire dati esterni potrebbe non essere sempre fattibile, ad esempio non è possibile utilizzare questo approccio con sensori già in circolazione. Invece, l'uso di un approccio puramente software consente di utilizzare un LD su una gamma più ampia di dispositivi. Per questa ragione, la maggior parte degli sforzi è stata spesa per la prima soluzione.

Nel corso degli anni sono stati proposti LD sempre più sofisticati per far fronte a sfide sempre più impegnative relative a tecniche di *spoofing*. Per supportare questo processo, nel 2009 è stata avviata la prima competizione di *liveness detection* (**LivDet**), un contest biennale in cui i partecipanti si sfidano per distinguere impronte digitali contraffatte da campioni vivi. Lo scopo del contest è quello di consentire ai ricercatori di confrontarsi su un protocollo comune sperimentale standardizzato, fornendo loro un gran numero di campioni raccolti in un ambiente noto. Nel corso delle varie edizioni della competizione i *liveness detectors* sono migliorati costantemente; in particolare, l'edizione del 2015 ha segnato una svolta poiché le impronte digitali contraffatte del *test set* sono state

prodotte utilizzando anche materiali non inclusi nel *training set* e soprattutto perché per la prima volta viene utilizzata una CNN per implementare un LD.

Infatti, il concorso è stato vinto da Nogueira² proprio con una CNN **VGG19** pre-addestrata su ImageNet e, quindi, perfezionata (*fine-tuning*) sul rilevamento della vitalità delle impronte digitali.

Da allora in poi non solo i migliori risultati di LivDet includono sempre approcci basati su CNN, ma è interessante notare che soluzioni simili sono risultate efficaci anche con altre biometrie.

Di seguito vengono descritte la struttura della CNN vincitrice del concorso LivDet 2015 e l'algoritmo utilizzato per addestrarla poiché saranno alla base della soluzione proposta.

In dettaglio, VGG-19 è una CNN composta da 16 *convolutional layer* e da 3 *fully connected layer*. L'ultimo livello della rete presenta di *default* un livello *softmax* composto da 1000 unità, in quanto la rete è stata generata per effettuare la classificazione delle 1000 classi di ImageNet e, siccome la *liveness detection* è un problema di classificazione binaria, l'ultimo strato della VGG-19 è stato rimpiazzato da un livello *softmax* con sole due unità in modo da calcolare il punteggio delle classi *live* e *spoof*.

² Cicero Nogueira dos Santos, ricercatore specializzato in machine learning

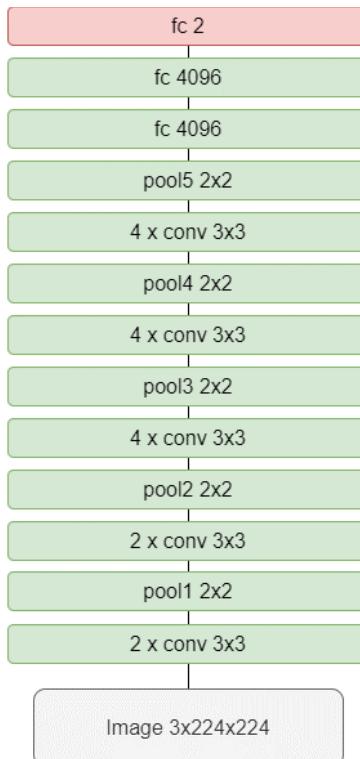


Figura 3 - VGG19 adattata ad un problema binario

La rete pre-addestrata su ImageNet è stata successivamente addestrata sul *training set* di impronte fornito da LiveDet utilizzando come algoritmo la Discesa Stocastica del Gradiente (SGD) con mini-batch di dimensione 5, momento 0.9 e *learning rate* costante pari a 10^{-6} .

Inoltre, altro fattore importante è la *data augmentation* utilizzata per aumentare la varietà e la dimensione del *set*. L'autore afferma di essersi ispirato ad un articolo di Krizhevsky e di aver aumentato la dimensione del *training set* di 10 volte prelevando da ogni immagine 5 *patch* con dimensione pari all'80% dell'originale: quattro relative agli angoli e una al centro e, in seguito, per ogni *patch* viene effettuata una riflessione orizzontale per ottenere un totale di 10 immagini:

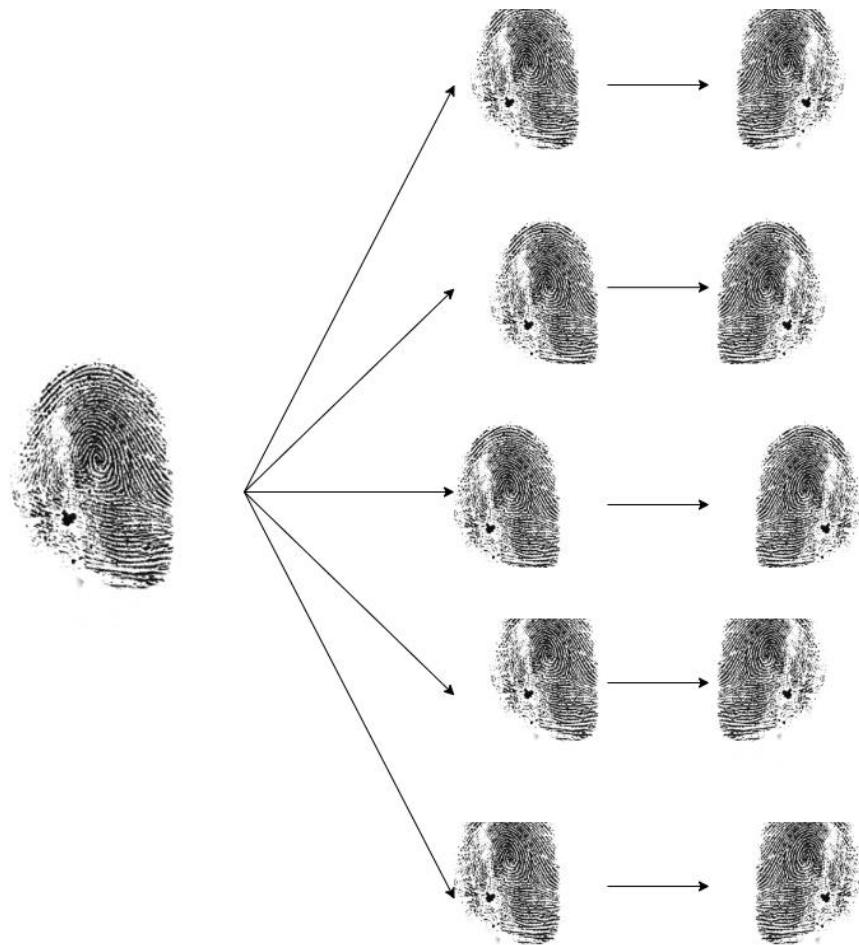


Figura 4 - Esempio estrazione 10 patch

Dopo aver terminato l'addestramento lo stesso metodo è utilizzato anche in fase di predizione per ogni impronta presentata in ingresso: vengono prelevate le 10 *patch* e predette singolarmente; la media delle predizioni viene poi utilizzata come predizione finale per l'immagine originale, in questo modo aumenta la robustezza del rilevatore.

2.2 Ingannare una convolutional neural network

Nel 2013 è stato dimostrato in un articolo che data una rete neurale convoluzionale, è possibile creare immagini in grado farla classificare male arbitrariamente. Vale la pena notare che gli autori dell'articolo non solo hanno dimostrato l'esistenza di vulnerabilità delle CNNs, ma hanno

anche introdotto un metodo per la generazione di esempi contraddittori basato sull'algoritmo Limited Memory Broyden-Fletcher-Goldfarb-Shanno (LM-BFGS) e sul valore della funzione di perdita della rete. Da quel momento in poi la ricerca ha pubblicato numerosi attacchi diversi e sono state ideate anche altre tecniche oltre all'*adversarial attack*.

In generale, si parla di **Adversarial Machine Learning** per indicare una serie di tecniche volte a compromettere il corretto funzionamento di un sistema informatico che faccia uso di algoritmi di apprendimento automatico tramite la costruzione di input speciali in grado di ingannare tali algoritmi: nello specifico, lo scopo di tali tecniche è quello di causare l'errata classificazione in uno di questi algoritmi.

Inoltre, nel caso dell'apprendimento supervisionato, è possibile costruire particolari input in grado di far trapelare informazioni sul *training set* usato, o di permettere la clonazione del modello stesso.

2.2.1 Tipologie di attacchi

Gli autori dell'**Adversarial Robustness ToolBox (ART)**, un toolbox python creato per testare la robustezza degli algoritmi di machine learning, suddividono gli attacchi nelle seguenti tipologie:

- **Poisoning** - lo scopo di un attacco di tipo poisoning è quello di diminuire il più possibile l'accuratezza di un determinato modello per tutti i possibili input. Varianti di questa tipologia di attacco esistono per diversi modelli, sia nel caso dell'apprendimento supervisionato che di quello non supervisionato: nel primo caso, il poisoning del modello avviene durante la fase di addestramento dell'algoritmo. Da un punto di vista della sicurezza informatica, il poisoning fa parte di una classe di attacchi definiti come *denial of service*.
- **Evasion** - l'obiettivo di un attacco di tipo evasion è quello di violare l'integrità di un modello di apprendimento automatico. Durante un attacco di questo tipo, l'attaccante modifica un determinato

campione con l'obiettivo di ottenere come output dal classificatore una classe che è diversa dalla sua reale classe di appartenenza; in alternativa, l'attacco potrebbe più semplicemente tentare di diminuire la confidenza del modello per quel campione. Quindi non si va ad attaccare il modello in fase di addestramento, ma si modifica un determinato input per farlo classificare male.

- **Extraction** - Mentre il poisoning e l'evasion rientrano nelle categorie di attacchi che violano, rispettivamente, il funzionamento generale di un modello e l'integrità di un modello per determinati input, nel caso del model extraction si considera una violazione della sua *confidentiality* (riservatezza). Tali attacchi possono essere usati sia per clonare un determinato modello che per inferire (**inference attack**) informazioni private o sensibili sul dataset usato durante il suo training, ad esempio gli attributi.

Da questa divisione si può intuire come l'adversarial attack sia di tipo evasion e questi ultimi, come già accennato nell'introduzione, possono essere divisi a loro volta in *white* e *black box*.

Inoltre, attacchi di tipo extraction possono essere usati come fase preliminare di un attacco *white-box* per aumentare le possibilità di successo rispetto ad un *black-box*.

2.3 Evasion attacks

Durante lo sviluppo del lavoro ci si è soffermati sull'analisi degli *evasion attacks* poiché lo scopo è generare degli artefatti che riescano ad essere riconosciuti come *live*.

Nel dettaglio un attacco di questo tipo, come formulato nel 2013, può essere descritto come un problema di ottimizzazione vincolata in cui si cerca la perturbazione minima (che non alteri troppo il dato originale) la quale aggiunta all'input faccia classificare in maniera diversa il modello:

Minimize $\|r\|$ subject to:

$$f(x+r) = l$$

$$\text{con } x+r \in [0,1], f(x) \neq l$$

in questo caso r è detta **perturbazione**.

Tutti gli attacchi di questo tipo si differenziano tra loro per le tecniche con cui calcolano la perturbazione o cercano di semplificare il problema per poi risolverlo con determinati algoritmi.

Anche tra gli *evasion attacks* è possibile individuare delle categorie, di seguito le più diffuse:

- **Gradient-based attack** - richiedono l'accesso ai gradienti del modello e sono quindi di tipo *white-box*. Sono senza dubbio i più pericolosi perché l'attaccante può usare i valori dei gradienti per ottimizzare matematicamente l'attacco. La ricerca ha dimostrato che se l'attaccante ha accesso ai gradienti del modello sarà sempre in grado di creare una nuova serie di esempi contraddittori per ingannare il modello.
- **Confidence score attack** - utilizzano la confidenza generata dal classificatore per stimare i gradienti del modello e quindi, eseguono un'ottimizzazione intelligente simile agli attacchi basati su gradiente. Questo approccio non richiede che l'attaccante sappia nulla del modello e quindi è di tipo *BlackBox*.
- **Hard label attack** - si basano esclusivamente sull'etichetta emessa dal modello ("gatto", "cane") e non richiedono i confidence score. Ciò rende l'attacco più debole, ma probabilmente più realistico poiché i confidence score sono difficili da ottenere.
- **Surrogate model attack** - sono molto simili agli attacchi basati sul gradiente, tranne per il fatto che richiedono un passaggio aggiuntivo. Quando l'avversario non ha accesso agli interni del modello ma vuole comunque eseguire un attacco *white-box* può

provare a ricostruire prima il modello del bersaglio sulla sua macchina.

- **Brute force attack** - generano casualmente degli *adversarial samples* con tecniche semplici.

Tra le categorie descritte si è scelto di utilizzare i *gradient-based* poiché più accurati e, inoltre, sono realizzabili visto che il modello vittima è conosciuto poiché verrà creato utilizzando il *training set* di LivDet.

Tra l'elevato numero di attacchi presenti in questa categoria, ne sono stati selezionati tre tra quelli messi a disposizione da ART e di seguito sono riportate le motivazioni della scelta:

- **IFGSM**, il più veloce tra gli attacchi a disposizione;
- **DeepFool**, scelto poiché ha riportato ottimi risultati in articoli che trattano argomenti simili alla tesi proposta;
- **APGD**, è il più complesso tra i tre attacchi ed è stato scelto poiché utilizza perturbazioni variabili che si adattano al problema.

2.3.1 Iterative Fast Gradient Sign Method

L'attacco IFGSM è una versione avanzata dell'attacco base FGSM, quindi, per poterne descrivere le caratteristiche bisogna introdurre prima la sua versione base.

Fast Gradient Sign Method

FGSM calcola i gradienti di una funzione di perdita (ad esempio, errore quadratico medio) rispetto all'immagine di input ed utilizza il segno dei gradienti per creare una nuova immagine che massimizza la perdita.

È possibile descrivere FGSM usando la seguente equazione:

$$adv_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

con $\|r\|_\infty < \epsilon$

dove:

- **adv_x** : immagine contraddittoria in output;
- **x** : immagine originale da perturbare;
- **y** : classe reale dell'immagine;
- **ϵ** : valore piccolo per cui moltiplicare il segno del gradiente, serve ad assicurare che le perturbazioni siano abbastanza piccole da non essere rilevate dall'occhio umano, ma abbastanza grandi da ingannare la rete neurale;
- **θ** : modello di rete neurale a disposizione (per questo white-box);
- **J** : *loss function*.

L'attacco è stato implementato inizialmente da Goodfellow utilizzando la norma infinito per limitare il valore di ϵ e non fa altro che applicare la formula indicata precedentemente. Intuitivamente, per ciascun pixel, FGSM utilizza il segno del gradiente della funzione di perdita per determinare in quale direzione l'intensità dei pixel dovrebbe essere cambiata (aumentata o diminuita) per massimizzare la funzione di perdita e con un singolo step cambia tutti i pixel contemporaneamente.

Bisogna precisare che FGSM nella versione base è un attacco a singolo step e non è detto che riesca a perturbare abbastanza l'immagine da farla classificare male; questo perché FGSM è stato progettato per essere veloce piuttosto che ottimale e non è destinato a produrre le minime perturbazioni contraddittorie.

IFGSM

È un modo semplice per estendere FGSM, esso non fa altro che applicare più volte (*iterative*) il metodo base con piccoli valori di perturbazione e ritaglia i valori dei pixel dei risultati intermedi per assicurare che siano contenuti in un intorno ampio ϵ dell'immagine originale:

$$\mathbf{X}_0^{adv} = \mathbf{X}, \quad \mathbf{X}_{N+1}^{adv} = Clip_{X,\epsilon} \left\{ \mathbf{X}_N^{adv} + \alpha \operatorname{sign}(\nabla_X J(\mathbf{X}_N^{adv}, y_{true})) \right\}$$

IFGSM introduce un semplice perfezionamento del metodo descritto precedentemente dove invece di fare un unico passo di grandezza epsilon nella direzione del gradiente, vengono eseguiti vari piccoli step di grandezza α ed il risultato è ritagliato dallo stesso ϵ . L'algoritmo procede finché la classe predetta non è diversa da quella reale, in questo modo si individua una “minima” perturbazione contraddittoria. Inoltre, l'algoritmo contiene un numero massimo di iterazioni da eseguire ed una perturbazione totale massima applicabile per arrestare l'algoritmo con fallimento.

L'unico svantaggio rispetto al metodo base è il tempo di esecuzione che aumenta, ma in confronto a tutti gli altri attacchi pubblicati dalla ricerca è comunque uno dei più veloci.

2.3.2 DeepFool

DeepFool si basa sull'idea che la robustezza di un modello binario f (dato il problema della liveness basta considerare solo due classi) per un input x_0 è uguale alla distanza di x_0 dal piano che separa le 2 classi:

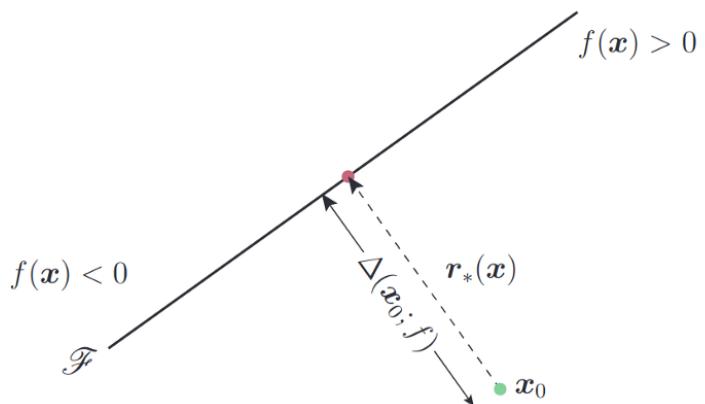


Figura 5 - Esempio piano separatore delle classi

Quindi l'attacco non fa altro che generare una serie di perturbazioni additive che spostano l'input verso il bordo del piano separatore.

Sia f il classificatore e x_i il punto in cui si trova l'input all'iterazione corrente, la perturbazione all'iterazione i -esima può essere calcolata come:

$$r_i = -\frac{f(x_i)}{\|\nabla f(x_i)\|_2^2} \nabla f(x_i)$$

Ad ogni iterazione la perturbazione ottenuta viene sommata con il punto corrente x_i per generare x_{i+1} :

$$x_{i+1} = x_i + r_i$$

Se la classe predetta per x_{i+1} è diversa da quella di x_0 (immagine di partenza), allora l'algoritmo termina e restituisce la perturbazione minima \hat{r} , altrimenti passa all'iterazione successiva. La perturbazione minima restituita dall'algoritmo è la perturbazione totale, ovvero la somma di tutte le perturbazioni calcolate:

$$\hat{r} = \sum r_i$$

Si noti che l'algoritmo può convergere in prossimità dell'iperpiano separatore, pertanto, la perturbazione \hat{r} viene moltiplicata per una costante $\eta \ll 1$ in modo che l'esempio contraddittorio generato vada oltre l'iperpiano separatore e, settando opportunamente questo parametro è possibile assicurare un certo livello di confidenza. Superato tale iperpiano, l'immagine perturbata viene classificata male rendendo l'attacco riuscito.

Da notare che DeepFool è un attacco iterativo che ad ogni step applica una nuova perturbazione finché il classificatore non cambia la sua decisione, quindi l'attacco dovrebbe riuscire sicuramente. In alcuni casi, però, l'algoritmo necessita di troppe iterazioni e quindi converge troppo lentamente, per questo è possibile settare un numero massimo di iterazioni per fermarlo.

2.3.3 Auto - Projected Gradient Descent

Come per IFGSM, anche APGD è un'evoluzione della versione base FGSM e, in particolare, è una versione migliorata e più complessa dell'attacco Projected Gradient Descent.

Projected Gradient Descent (PGD)

PGD è una variante di IFGSM che utilizza la discesa del gradiente proiettata sulla funzione di perdita (teoricamente è l'ascesa del gradiente poiché massimizza una funzione piuttosto che minimizzarla, ma è comune riferirsi semplicemente al processo come discesa del gradiente):

$$x^{t+1} = \Pi_{x+S} (x^t + \alpha \operatorname{sgn}(\nabla_x L(\theta, x, y)))$$

dove Π è la funzione proiezione in un intorno di grandezza S dell'immagine iniziale x .

L'attacco PGD a differenza di IFGSM inizializza l'algoritmo su un punto casuale nella sfera di interesse (deciso dalla norma L_∞) ed esegue riavvii casuali per poi proiettare il punto finale sulla sfera stessa, mentre IFGSM inizializza al punto originale ed esegue un *clipping* al posto della proiezione.

Sia IFGSM che PGD essendo iterativi applicano delle perturbazioni più “precise” che non distruggono l’immagine anche con un ϵ maggiore e allo stesso tempo hanno un tasso di successo più elevato.

Algoritmo

1. Partenza da un punto casuale nella sfera L^p intorno al campione di input x ;
2. “Passo del gradiente” in direzione della perdita maggiore, ovvero somma la perturbazione calcolata al punto casuale cercando di aumentare la *loss function*;
3. Se il valore ottenuto è uscito dalla sfera lo proietta al suo interno;

4. Ripetizione punti 2 e 3 finché non converge, ovvero finché in iterazioni successive non trova sempre lo stesso massimo della funzione di perdita o finché non raggiunge il numero massimo di iterazioni.

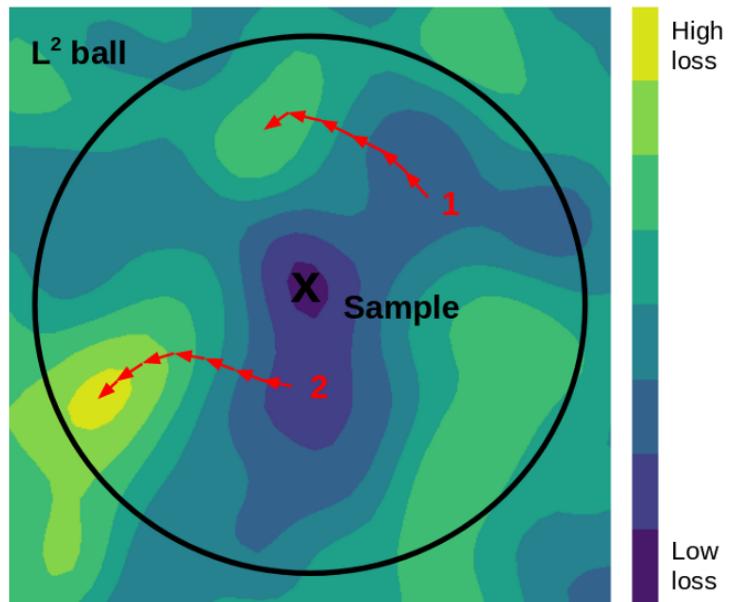


Figura 6 - Esempio PGD

Di seguito è riportato un esempio grafico in 2D della proiezione del punto nella sfera:

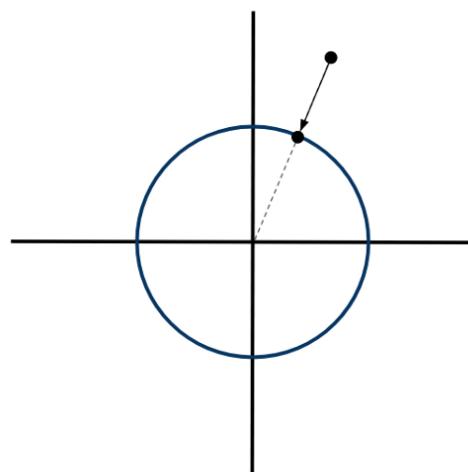


Figura 7 - Esempio proiezione

Auto-PGD

La versione avanzata di PGD mira a risolvere i seguenti problemi:

- dimensione del passo α costante subottimale poiché non garantisce la convergenza e le prestazioni dell'algoritmo sono altamente influenzate dalla scelta del suo valore;
- lo schema complessivo è in generale agnostico del budget dato all'attacco (ovvero di quanto si possono modificare i pixel);
- l'algoritmo è inconsapevole delle tendenze, cioè non considera se l'ottimizzazione si sta evolvendo con successo e non è in grado di reagire a questo.

L'idea principale di Auto-PGD è quella di partizionare le N_{iter} disponibili in una fase iniziale di **esplorazione**, dove si cerca un insieme ammissibile di buoni punti iniziali, ed una fase di **sfruttamento**, durante la quale si cerca di utilizzare la conoscenza finora accumulata.

La transizione tra le due fasi è gestita riducendo progressivamente la dimensione del passo. In effetti, una grande dimensione del passo permette di muoversi velocemente, mentre una più piccola massimizza la funzione obiettivo in maniera *greedy* a livello locale; quindi, è bene partire da un valore più grande per poi ridurlo gradualmente. La riduzione della dimensione del passo non è programmata a priori, ma piuttosto è governata dall'andamento dell'ottimizzazione: se il valore dell'obiettivo cresce sufficientemente veloce la dimensione del passo è molto probabilmente corretta, altrimenti è ragionevole ridurla. Inoltre, una volta ridotta la dimensione del passo, l'algoritmo riparte dal punto migliore finora trovato.

Algoritmo

```

1: Input:  $f, S, x^{(0)}, \eta, N_{\text{iter}}, W = \{w_0, \dots, w_n\}$ 
2: Output:  $x_{\max}, f_{\max}$ 
3:  $x^{(1)} \leftarrow P_S(x^{(0)} + \eta \nabla f(x^{(0)}))$ 
4:  $f_{\max} \leftarrow \max\{f(x^{(0)}), f(x^{(1)})\}$ 
5:  $x_{\max} \leftarrow x^{(0)}$  if  $f_{\max} \equiv f(x^{(0)})$  else  $x_{\max} \leftarrow x^{(1)}$ 
6: for  $k = 1$  to  $N_{\text{iter}} - 1$  do
7:    $z^{(k+1)} \leftarrow P_S(x^{(k)} + \eta \nabla f(x^{(k)}))$ 
8:    $x^{(k+1)} \leftarrow P_S(x^{(k)} + \alpha(z^{(k+1)} - x^{(k)})$ 
       $+ (1 - \alpha)(x^{(k)} - x^{(k-1)}))$ 
9:   if  $f(x^{(k+1)}) > f_{\max}$  then
10:     $x_{\max} \leftarrow x^{(k+1)}$  and  $f_{\max} \leftarrow f(x^{(k+1)})$ 
11:   end if
12:   if  $k \in W$  then
13:     if Condition 1 or Condition 2 then
14:        $\eta \leftarrow \eta/2$  and  $x^{(k+1)} \leftarrow x_{\max}$ 
15:     end if
16:   end if
17: end for

```

Di seguito sono analizzate le principali fasi dell'algoritmo.

Gradient step

È la fase di aggiornamento dell'immagine:

$$\begin{aligned}
 z^{(k+1)} &= P_S \left(x^{(k)} + \eta^{(k)} \nabla f(x^{(k)}) \right) \\
 x^{(k+1)} &= P_S \left(x^{(k)} + \alpha \cdot (z^{(k+1)} - x^{(k)}) \right. \\
 &\quad \left. + (1 - \alpha) \cdot (x^{(k)} - x^{(k-1)}) \right)
 \end{aligned}$$

dove alfa (appartenente a $[0,1]$) regola l'influenza del precedente aggiornamento su quello attuale. Dal momento che nelle prime iterazioni di APGD la dimensione del passo è particolarmente grande, si vuole inserire un'influenza dei passaggi precedenti.

Step size selection

L'algoritmo parte con una certa dimensione del passo $\eta = 2\epsilon$ e, dato un budget di iterazioni N_{iter} , si identificano i checkpoint $w_0 = 0, w_1, \dots, w_n$ in cui il l'algoritmo decide se è necessario dimezzare la corrente dimensione del passo.

Il passo viene dimezzato se sono soddisfatte le seguenti due condizioni:

1. $\sum_{i=w_{j-1}}^{w_j-1} \mathbf{1}_{f(x^{(i+1)}) > f(x^{(i)})} < \rho \cdot (w_j - w_{j-1})$,
2. $\eta^{(w_{j-1})} \equiv \eta^{(w_j)}$ and $f_{\max}^{(w_{j-1})} \equiv f_{\max}^{(w_j)}$,

dove $f_{\max}^{(k)}$ è il valore obiettivo più alto trovato nelle prime k iterazioni.

Condizione 1: conta in quanti casi dall'ultimo checkpoint w_{j-1} il *gradient step* ha avuto successo nell'aumento di f . Se ciò è accaduto per almeno una frazione ρ del numero di passi effettuati in totale, allora la dimensione del passo viene mantenuta poiché l'ottimizzazione sta procedendo correttamente (usa $\rho = 0,75$).

Condizione 2: è vera se la dimensione del passo non è stata ridotta all'ultimo checkpoint e non si sono verificati miglioramenti nel miglior valore di f trovato dall'ultimo checkpoint (ovvero se $f_{\max}^{(k)}$ non viene aggiornato dall'ultimo checkpoint). In questo modo si impedisce di rimanere bloccati in potenziali cicli.

Se una delle due condizioni è vera, allora la dimensione del passo all'iterazione $k=w_j$ è dimezzata e si ha: $\eta_k := \eta_k/2$ per ogni $k = w_j + 1, \dots, w_{j+1}$.

Restarts from the best point

Se ad un checkpoint w_j la dimensione del passo viene dimezzata, si pone $x^{(w_j+1)} = x_{\max}$, cioè si riparte dal punto con valore massimo di f finora. Ciò ha

senso in quanto ridurre la dimensione del passo porta ad una ricerca più localizzata, e questo dovrebbe essere fatto in un intorno della migliore soluzione attualmente candidata.

Exploration vs exploitation

L'idea è che l'algoritmo transiti gradualmente dall'esplorazione dell'intero insieme ammissibile S ad un'ottimizzazione locale e questa transizione è regolata dalla progressiva riduzione della dimensione del passo e dalla scelta di quando diminuirlo, ovvero i checkpoint w_j . In pratica, si vuole consentire una fase di esplorazione iniziale relativamente lunga, per poi eventualmente aggiornare la dimensione del passo più spesso andando verso la fase di sfruttamento. In effetti, con dimensioni del passo più piccole i miglioramenti nella funzione obiettivo sono probabilmente più frequenti ma anche di minore entità, mentre l'importanza di prendere il vantaggio dell'intero spazio di input è testimoniato dai risultati sperimentali ottenuti con i riavvii casuali nel classico attacco PGD.

La scelta dei checkpoint è fatta nel seguente modo:

$$w_j = \lceil \rho_j N_{iter} \rceil \leq N_{iter}$$

dove $\rho_j \in [0, 1]$, $\rho_0 = 0$, $\rho_{j+1} = \rho_j + \max \{ \rho_j - \rho_{j-1} - 0.03, 0.06 \}$

Nonostante siano presenti molti parametri da poter scegliere, gli autori dell'attacco affermano che è consigliato usare i valori indicati nel loro articolo e di lasciare libero solo N_{iter} .

Loss function

Infine, APGD permette anche di scegliere diverse tipologie di *loss functions*:

- funzione del modello vittima;
- *cross-entropy*:

$$\text{CE}(x, y) = -\log p_y = -z_y + \log \left(\sum_{j=1}^K e^{z_j} \right),$$

- difference of logits ratio:

$$\text{DLR}(x, y) = -\frac{z_y - \max_{i \neq y} z_i}{z_{\pi_1} - z_{\pi_3}},$$

2.4 Authentication System

Esistono vari modi per implementare un sistema di autenticazione biometrico basato su impronte digitali e per lo sviluppo della tesi è usato il *software NBIS* sviluppato dal National Institute of Standards and Technology (NIST) per il Federal Bureau of Investigation (FBI) e il Dipartimento della Sicurezza Nazionale americana (DHS).

Il software NBIS è composto da vari elementi che permettono l'analisi delle impronte, ma per realizzare un *matcher* sono stati utilizzati: Mindtct, per estrarre le caratteristiche dell'impronta (rilevatore di minuzie) e Bozorth, che permette di confrontare minuzie di impronte diverse per classificarle come appartenenti, o non, allo stesso individuo.

2.4.1 Mindtct: Minutiae Detection

Questo paragrafo descrive prima cosa sono le minuzie di un'impronta digitale e poi come il *software* Mindtct riesce ad estrarle e salvarle.

Definizione di minuzie

Un'impronta digitale è costituita da due tipologie di linee chiamate **ridges** (creste) e **valleys** (solchi), le quali formano una sorta di schema chiamato *ridge pattern*.

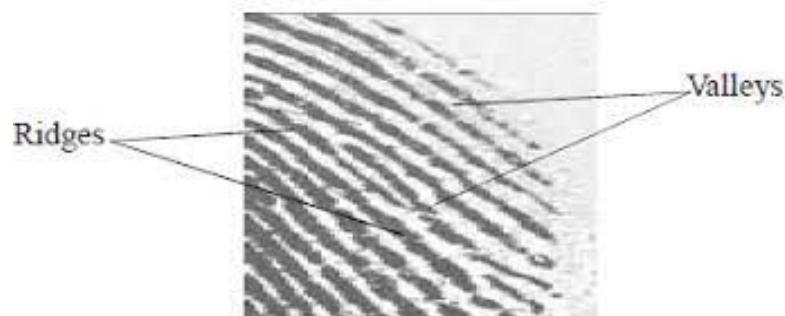


Figura 8 - Ridges & Valleys

Tradizionalmente, due impronte digitali sono confrontate utilizzando caratteristiche discrete, chiamate **minuzie**. Queste caratteristiche includono punti nella pelle di un dito dove le creste terminano (*ridge ending*) o si dividono (*ridge bifurcation*).

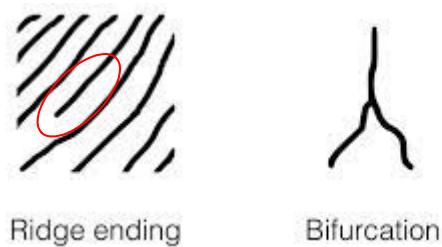


Figura 9 - Esempio minuzie

Per verificare la corrispondenza delle impronte digitali è necessario registrare le coordinate e l'orientamento di ogni minuzia.

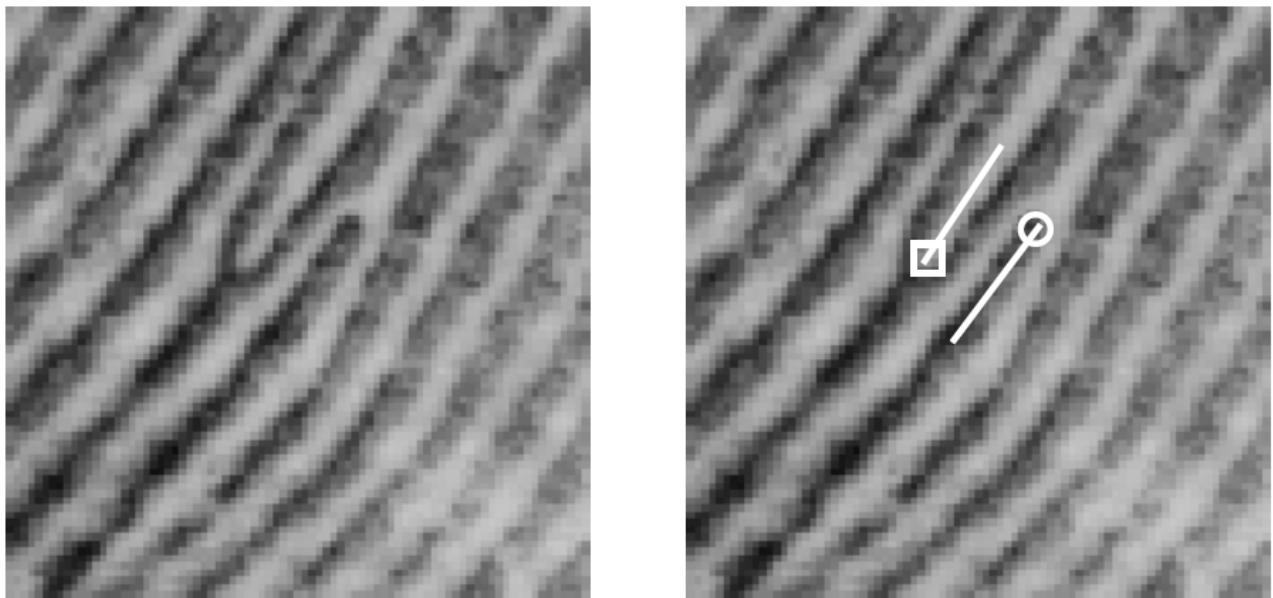


Figura 10 - Coordinate minuzie

La posizione di ogni minuzia è rappresentata con una posizione all'interno dell'immagine; esistono vari modi per indicare questa posizione e lo standard ANSI/NIST specifica le unità di distanza in termini di 0,01 mm dall'angolo inferiore sinistro dell'immagine.

L'orientamento, invece, è rappresentato in gradi, con zero gradi che puntano in orizzontale verso destra, mentre gradi crescenti procedendo in senso antiorario.

L'orientamento di una *ridge ending* è determinato misurando l'angolo tra l'asse orizzontale e la linea che parte dall'estremità della minuzia ed esce da essa.

L'orientamento di una biforcazione è determinato misurando l'angolo tra l'asse orizzontale e la linea che inizia dalla biforcazione e che attraversa la parte dove le due creste sono unite.

Nell'esempio l'orientamento è indicato da A:

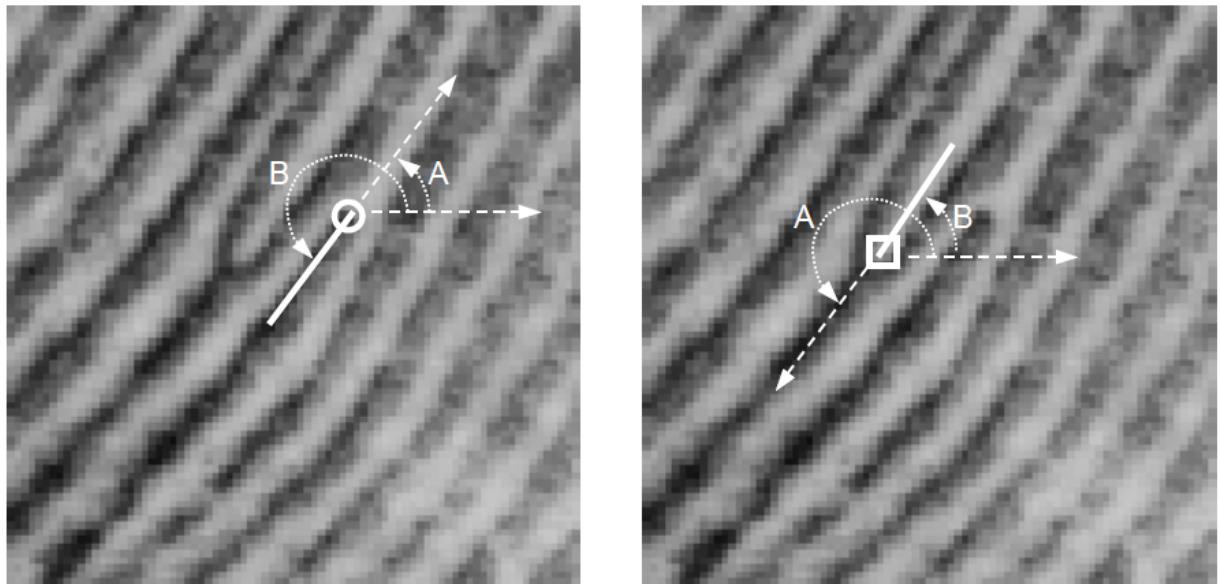


Figura 11 - Orientamento minuzie (sinistra terminazione, destra biforcazione)
A) angolo standard
B) angolo FBI/IAFIS

Algoritmo

Una volta ottenuta un'impronta digitale in ingresso, per registrare le minuzie Mindtct effettua un processo diviso in più fasi.

1. Generate image maps

Poiché la qualità dell'immagine di un'impronta digitale può variare, soprattutto nel caso di impronte digitali latenti, è fondamentale essere in grado di analizzare l'immagine e determinare le aree che sono degradate e che potrebbero causare problemi. È possibile misurare diverse caratteristiche per trasmettere informazioni riguardo la qualità di regioni localizzate nell'immagine. Queste caratteristiche includono: la determinazione del flusso direzionale delle creste nell'immagine ed il rilevamento di regioni a basso contrasto, basso flusso delle creste ed alta curvatura. Queste ultime tre condizioni rappresentano aree instabili nell'immagine in cui il rilevamento delle minuzie è inaffidabile.

Utilizzando le mappe generate per individuare regioni diverse, questa fase genera un file finale che assegna ad ogni punto dell'immagine un livello di qualità.

2. Binarize image

L'algoritmo di rilevamento delle minuzie è progettato per operare con immagini binarie in cui i pixel neri rappresentano le creste e i pixel bianchi rappresentano le valli. Per creare questa immagine binaria, è necessario analizzare ogni pixel dell'immagine di input in scala di grigi e determinare se deve essere assegnato un pixel bianco o nero. Questo processo è indicato come immagine **binarizzazione**.

A un pixel viene assegnato un valore binario basato sulla direzione del flusso della cresta associata al blocco a cui appartiene il pixel. Se non è presente un flusso di cresta rilevabile per il blocco del pixel corrente, allora il suo valore è impostato su bianco. Se viene rilevato un flusso di cresta, allora le intensità dei pixel che circondano il flusso appartenente a quel blocco vengono analizzate per determinare se il valore finale sarà bianco o nero.

3. Detect Minutiae

Questo passaggio scansiona metodicamente l'immagine binaria, analizzando coppie consecutive di pixel nell'immagine alla ricerca di sequenze che combaciano con i pattern relativi alle minuzie. I pixel che corrispondono ai pattern vengono considerati come “potenziali” minuzie.

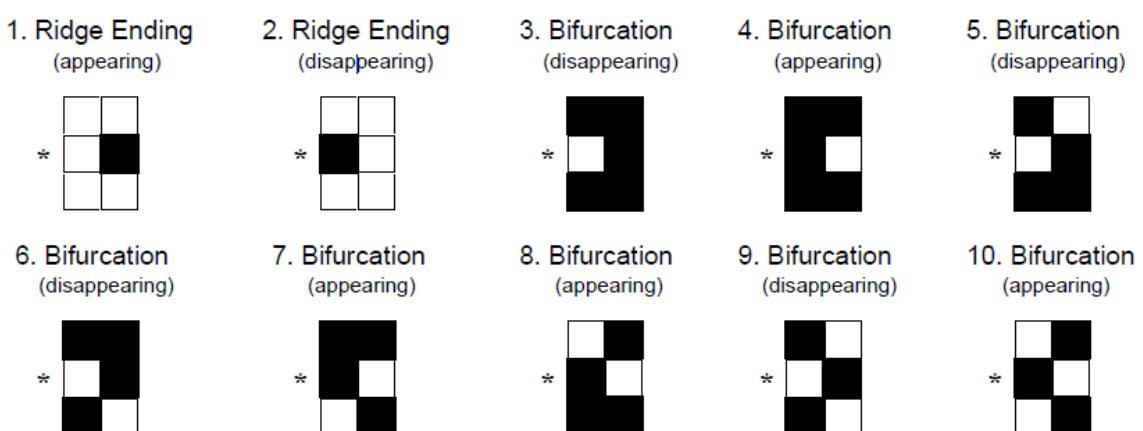


Figura 12 - Esempio riconoscimento minuzie

4. Remove False Minutiae

Nella lista dei candidati sono incluse molte false minutie, per questo molto sforzo viene speso per rimuovere quest'ultime. Questo passaggio include la rimozione di isole, laghi, fori, minutie in regioni di scarsa qualità dell'immagine, minutie laterali, ganci, sovrapposizioni, minutie troppo larghe e minutie troppo strette (pori).

5. Count Neighbor Ridges

Mindtct, come molti altri *matcher* di impronte digitali, usa altre informazioni oltre ai soli punti di minuzia. Le informazioni ausiliarie di solito includono la direzione della minuzia, il suo tipo e possono includere informazioni relative alle minutie dei vicini.

6. Assess Minutiae Quality

Nonostante il lungo processo di rimozione di false minutie, molte di esse non vengono eliminate per non correre il rischio di perdere informazioni; per questo, l'ultima fase consiste nel quantificare la qualità di ogni minutia in modo da riuscire a dare un peso minore alle minutie "sospette" durante il *matching*.

Alla fine di questa fase il *software* rilascia una serie di file che poi potranno essere usati dal *matcher* vero e proprio per effettuare il confronto: Bozorth.

2.4.2 Bozorth: Fingerprint Matcher

L'algoritmo di corrispondenza Bozorth calcola un punteggio di corrispondenza tra le minutie di due qualsiasi impronte digitali per determinare se appartengono allo stesso individuo.

Il *matcher* costruisce tabelle separate per le impronte digitali da confrontare, le quali definiscono la distanza e l'orientamento delle minutie in ogni impronta digitale. Queste due tabelle vengono quindi confrontate per la compatibilità e ne viene costruita una nuova che

memorizza le informazioni relative alla compatibilità tra le impronte. La tabella di compatibilità, infine, è utilizzata per creare un punteggio di corrispondenza osservando la dimensione e il numero di gruppi di minuzie compatibili.

Algoritmo

È importante precisare due cose riguardo il *matcher*:

- i. Utilizza solo la posizione (x,y) e l'orientamento t delle minuzie, rappresentate come una tripla {x,y,t};
- ii. L'algoritmo è designato per essere invariante a rotazione e traslazione.

Di seguito sono descritte le tre fasi dell'algoritmo Bozorth.

1. Construct Intra-Fingerprint Minutia Comparison Tables

Il primo passo dell'algoritmo consiste nel calcolare per ogni minuzia delle misurazioni relative a tutte le altre minuzie nella stessa impronta digitale. Queste misurazioni relative vengono memorizzate in una tabella di confronto e sono ciò che fornisce invarianza di rotazione e traslazione dell'algoritmo.

2. Construct an Inter-Fingerprint Compatibility Table

Il passaggio successivo dell'algoritmo consiste nel cercare voci "compatibili" tra le tabelle di confronto delle minuzie di due impronte separate. Se la distanza e gli angoli relativi tra le due voci della tabella di confronto sono accettabili, allora viene inserita una voce nella tabella di compatibilità delle due impronte.

3. Traverse the Inter-Fingerprint Compatibility Table

A questo punto del processo è stata costruita una tabella di compatibilità che consiste in un elenco di associazione di compatibilità tra minuzie potenzialmente corrispondenti. Queste associazioni rappresentano singoli collegamenti in un grafo di compatibilità. Per determinare quanto bene le

due impronte digitali corrispondono si attraversa il grafo con il percorso più lungo delle associazioni. Il risultato finale, quindi, è la lunghezza del percorso più lungo.

Il risultato restituito dall'algoritmo rappresenta solo approssimativamente il numero di minuzie che hanno superato il *match*; quindi, come regola generale si adotta che un numero superiore a 40 sia sufficiente per ottenere una corrispondenza tra due impronte digitali.

CAPITOLO 3: APPROCCIO PROPOSTO

Nel corso di questo capitolo si vuole riportare nel dettaglio la soluzione adottata per risolvere il problema introdotto inizialmente, ovvero la realizzazione di un processo che permetta di generare delle impronte digitali perturbate, le quali una volta stampate su lattice siano capaci di ingannare uno dei migliori scanner in circolazione.

Durante l'esposizione della soluzione vengono descritti i *dataset* iniziali, gli scanner creati con l'utilizzo di CNN (come accennato nel capitolo precedente) ed il modo in cui vari attacchi, forniti dalla letteratura, sono stati modificati per migliorare le perturbazioni applicate alle impronte digitali inizialmente classificate come *spoof*.

Vengono descritte varie versioni di processo di attacco al fine di migliorare non solo l'*accuracy*, ma anche il processo successivo di stampa.

Il progetto è stato realizzato utilizzando Google Colab e, in particolare, il *framework open source* specializzato nel *machine learning*: **Pytorch**.

3.1 Fingerprint Liveness Dataset

Il dataset utilizzato per la realizzazione del progetto è stato fornito durante la competizione LivDet del 2015.

LivDet ha l'obiettivo di confrontare metodologie di rilevamento della vitalità delle impronte digitali basate su software o hardware. Nell'edizione del 2015 undici istituzioni si sono iscritte con dodici proposte per la parte basata su software ed una per la parte hardware-based.

Il dataset è costituito da immagini provenienti da quattro diversi dispositivi ottici: GreenBit, Biometrika (modello HiScan), Digital Persona e Crossmatch. Per ciascuno di questi dispositivi sono fornite più di 4000 immagini.

Le immagini *live* provengono da acquisizioni multiple di tutte le dita di soggetti diversi. Ogni dito è stato acquisito con modi diversi per imitare scenari reali, ovvero con dita bagnate e asciutte e con elevata e bassa pressione del dito. Le immagini *spoof* sono state acquisite con modalità cooperativa ed i materiali utilizzati per la loro realizzazione sono ecoflex, gelatina, lattice, colla per legno, ecoflex liquido e RTV (una gomma siliconica bicomponente) per Green Bit, Biometrika e Digital Persona; mentre per CrossMatch sono stati utilizzati playdoh, body bouble, ecoflex, OOMOO (gomma siliconica) e una nuova forma di gelatina.

Gli interi set di dati sono stati divisi in due parti: *training* e *test*, uno per la configurazione degli algoritmi e l'altro per valutazione delle prestazioni. I set di test includono immagini contraffatte create usando materiali sconosciuti, ovvero materiali che non sono stati inclusi nel set di addestramento. I materiali sconosciuti sono: ecoflex liquido e RTV per Green Bit, Biometrika e Digital Persona; OOMOO e gelatina per Crossmatch. Questa pratica è stata adottata per valutare l'affidabilità dei modelli attaccati con materiali sconosciuti.

Il *training set* è costituito da 8983 immagini, mentre il *test set* da 10448.

Di seguito sono riportate informazioni divise per le varie tipologie di *scanner* e, per quanto riguarda il *test set*, è presente anche una divisione per materiale utilizzato per la generazione degli artefatti.

3.1.1 Scanners

Come già accennato, sono presenti quattro tipologie di scanner:

Scanner	Model	Resolution [dpi]	Format images
GreenBit	DactyScan26	500	PNG
Biometrika	HiScan-PRO	1000	BMP
Digital Persona	U.are.U 5160	500	PNG
CrossMatch	L Scan Guardian	500	BMP

Nella tabella sottostante sono riportate le informazioni principali del *dataset* riguardanti ogni *liveness detector*, ovvero numero totale di immagini (divise in *live* e *spoof*) e dimensione di ogni immagine.

Scanner	Training Size (Live+Spoof)	Test Size (Live+Spoof)	Image Size (height x width)
CrossMatch	2983 (1510+1473)	2948 (1500+1448)	749x799 Live 750x800 Spoof
Digital Persona	2000 (1000+1000)	2500 (1000+1500)	324x252
GreenBit	2000 (1000+1000)	2500 (1000+1500)	500x500
Biometrika	2000 (1000+1000)	2500 (1000+1500)	1000x1000

È possibile notare come il *dataset* di CrossMatch sia l'unico ad avere dimensioni diverse da tutti gli altri e, inoltre, ha anche immagini con dimensioni diverse tra *live* e *spoof*.

Di seguito sono mostrate anche alcune immagini *live* del *training set* per ogni *scanner*:



Figura 13
CrossMatch



Figura 14
DigitalPersona

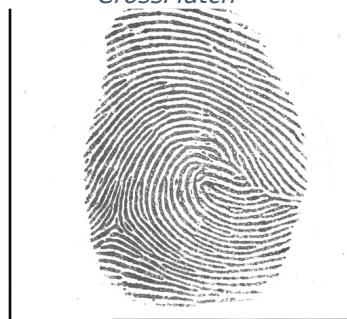


Figura 15 - GreenBit



Figura 16 - HiScan

3.1.2 Materiali

Per quanto riguarda le impronte *spoof*, esse sono state realizzate stampando impronte digitali, ottenute con metodo cooperativo, su diverse tipologie di materiali in modo da studiare la capacità degli scanner di riconoscere meglio un determinato materiale rispetto ad un altro. Inoltre, nei *test sets*, oltre ai materiali del *training*, sono presenti anche altri materiali in modo da poter verificare la capacità dei *detectors* di trattare materiali sui quali non sono stati addestrati.

Dataset	Live Image	Ecoflex	Gelatine	Latex	WoodGlue	Liquid Ecoflex	RTV
Green Bit	1000	250	250	250	250	250	250
Biometrika	1000	250	250	250	250	250	250
Digital Persona	1000	250	250	250	250	250	250
	Live Image	Body Double	Ecoflex	Playdoh	OOMOO	Gelatin	-
Crossmatch	1500	300	270	281	297	300	-

Figura 17 - Datasets divisi in immagini Live e Spoof divise per materiale

Anche in questo caso CrossMatch si differenzia dagli altri tre per la presenza di materiali diversi.

3.2 CNN per Liveness Detection

La creazione dei vari *liveness detectors* è stata effettuata utilizzando il metodo proposto da Nogueira e descritto in [2.1 Fingerprint Liveness Detection], ma con parametri diversi. Quindi, come prima operazione viene effettuata una *data augmentation offline*, in cui, per ogni immagine presente nei *training sets*, vengono estratte e salvate 10 *patch* con dimensione pari all'80% della dimensione iniziale, ottenendo dei *training sets* dieci volte più grandi degli originali. La fase successiva riguarda la creazione della rete a partire da una CNN pre-addestrata ed il *fine tuning* utilizzando i *dataset* realizzati. Durante il *tuning* dei parametri l'unico scanner ad aver dato problemi è stato DigitalPersona a causa della bassa dimensione delle immagini (324x252), per questo è l'unico per cui non è stata utilizzata una VGG19 ma una DenseNet.

Pytorch fornisce una collezione di reti pre-addestrate su ImageNet; quindi, l'unica operazione da effettuare è il caricamento di una determinata CNN modificando il numero di neuroni di *output* per adattarla al problema binario.

Per quanto riguarda il *fine tuning*, le reti fornite da *pytorch* utilizzano come *input* immagini con determinate caratteristiche:

- RGB;
- dimensione 224x224;

- Valori *float* compresi nell'intervallo [0,1];
- Normalizzazione con $mean = [0.485, 0.456, 0.406]$ e $std = [0.229, 0.224, 0.225]$;

per questo, ogni immagine presente nel *training set* prima di essere sottomessa alla rete deve subire una trasformazione che la adatti alle caratteristiche descritte sopra. In particolare, le immagini vengono caricare già come RGB con valori in [0,1] e prima di essere sottoposte alla rete bisogna effettuare solo un *resize* e la normalizzazione proposta. Bisogna precisare che a seconda dell'interpolazione usata si hanno diverse tipologie di ridimensionamento con risultati differenti e, sperimentalmente, quella che ha restituito risultati migliore è la *nearest*.

Quindi, le CNN usate per la creazione degli *scanners* utilizzano solo immagini 224x224, ma a seconda della dimensione iniziale dell'immagine si ottengono *patch* di dimensione diversa, per esempio nel caso di GreenBit le *patch* hanno dimensione 400x400 poiché le impronte iniziali sono 500x500.

L'addestramento delle reti per i vari *scanner* è stato effettuato utilizzando il *mini-batching* ed una *validation split* con rapporto 8:2, ovvero utilizzando l'80% del *traning set* per l'addestramento ed il restante 20% per la validazione.

Di seguito, sono riportate le reti utilizzate per ogni *scanner* con i valori degli iperparametri utilizzati in fase di addestramento ed i risultati finali ottenuti.

Scanner	CNN	Algorithm	Learning rate	Batch size	# Epochs
CrossMatch	VGG19	Adam	10^{-5}	200	10
DigitalPersona	DenseNet 201	Adam	10^{-4}	200	15
GreenBit	VGG19	Adam	10^{-5}	200	10

HiScan	VGG19	Adam	10^{-5}	200	10
--------	-------	------	-----------	-----	----

Scanner	Accuracy	Loss	Validation Accuracy	Validation Loss	Test Accuracy
CrossMatch	99.34%	0.1	98.56%	0.1	93.25%
DigitalPersona	96.69%	0.11	96.65%	0.1	89.24%
GreenBit	99.77%	0.1	99.2%	0.1	95.84%
HiScan	99.68%	0.1	99.35%	0.1	93.52%

Tra i vari scanner creati è stato deciso di analizzarne solo uno, ovvero GreenBit. Questo perché, non solo ha restituito l'accuracy maggiore sul set di prova, ma attualmente è anche il migliore in circolazione ed il più diffuso. Quindi, dato che lo scopo del lavoro è creare un processo di attacco realizzabile ed utilizzabile concretamente e non semplicemente testare attacchi su scanner diversi, GreenBit risulta la scelta migliore su cui focalizzarsi.

È stato scelto di riportare anche un esempio di impronte GreenBit per ogni materiale:



Figura 18 - Latex

Figura 19 - Ecoflex



Figura 20 - Gelatine

Figura 21 - Liquid Ecoflex

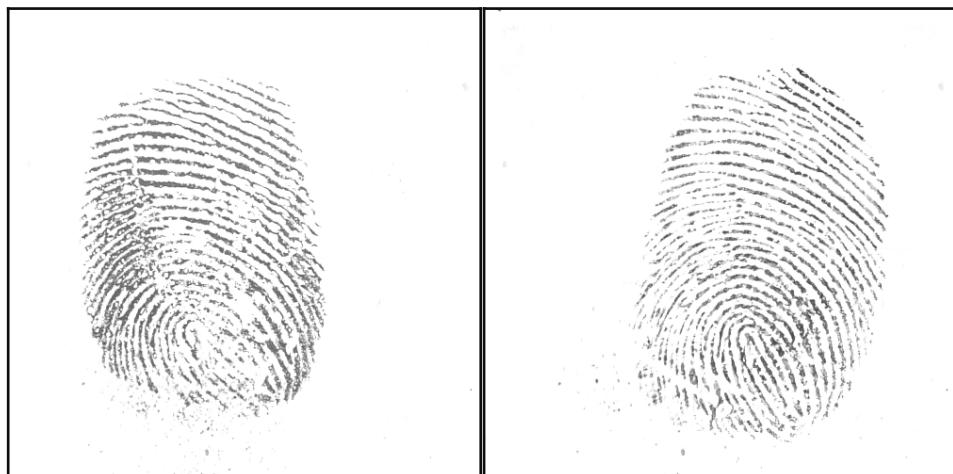


Figura 22 - RTV

Figura 23 - WoodGlue

Tra i vari materiali a disposizione è stato scelto di utilizzare solo il lattice poiché con le prime versioni degli attacchi, insieme ad ecoflex, ha restituito risultati più incoraggianti e, inoltre, collaborando con l'Università degli Studi di Cagliari, è risultato molto più semplice da trattare nel processo di stampa.

3.2.1 Testing

Bisogna ricordare che per il *test set* viene utilizzata una funzione che effettua la media delle predizioni su dieci patch dell'immagine originale, come proposto nell'articolo di Nogueira. Infatti, un *testing* normale ha riportato risultati peggiori da quelli mostrati nella tabella precedente.

Di seguito è riportato uno schema che descrive la funzione utilizzata per testare le immagini e che sarà utilizzata anche in tutti gli attacchi nei prossimi paragrafi:



Figura 24 - Schema testing per GreenBit

Da notare come il *resize* a 224x224 viene effettuato solo sulle singole patch e non sulle immagini iniziali, poiché sono le uniche ad essere sottoposte alla rete. Effettuando un ridimensionamento sulle immagini originali, l'estrazione delle patch creerebbe *patches* con dimensione inferiore a 224x224 e ciò necessiterebbe di un ulteriore *resize* per aumentarne le dimensioni portando quindi ad una perdita eccessiva delle informazioni contenute nell'immagine. Sperimentalmente, infatti, il metodo mostrato nello schema si è dimostrato il migliore.

3.2.2 Scanner GreenBit e Latex

Data la scelta di un solo scanner e di un solo materiale nel proseguimento del progetto, è stato deciso di riportare solo per essi informazioni più dettagliate riguardanti il *dataset*.

Come già detto, GreenBit è stato ottenuto addestrando con 20000 immagini una VGG19 pre-addestrata su ImageNet; mentre, per quanto riguarda la porzione di *test set* utilizzata, si parla di 250 impronte digitali *spoof* stampate su lattice e di seguito sono riportati i risultati del *testing*:

- Numero di immagini riconosciute *spoof* sul numero totale di immagini *spoof latex*: **248/250**;
- Accuracy: **99.2%**.

Quindi, lo scanner sbaglia a riconoscere come *spoof* solo 2 immagini su 250, ovvero le seguenti:

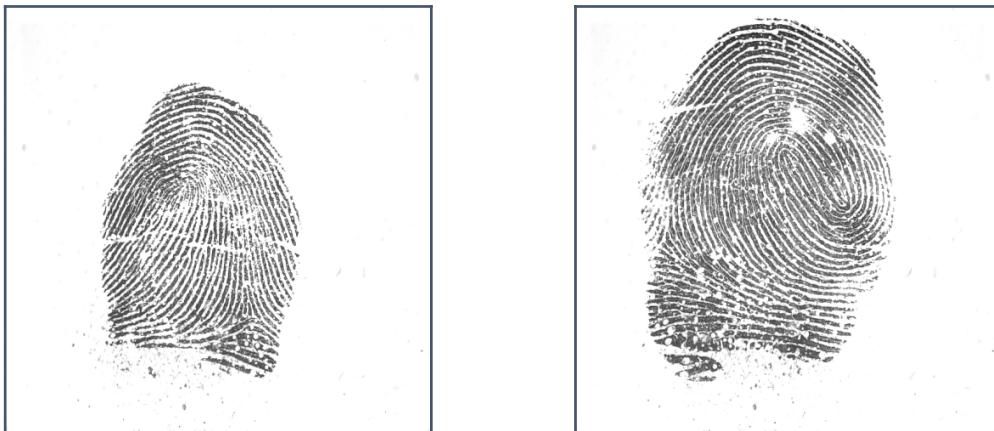


Figura 25 – Impronte 007_0_9 e 007_6_9 riconosciute live

Le restanti 248 immagini saranno quindi le impronte da perturbare ed in seguito stampare per fuorviare lo scanner.

3.3 Attacco liveness detector

Una volta definito il modello da utilizzare ed il set di dati da attaccare, il lavoro si sposta sulla generazione di perturbazioni capaci di fuorviare la VGG19 utilizzata. Per effettuare gli attacchi è stata utilizzata ART (*Adversarial Robustness Toolbox*), una libreria Python sviluppata da IBM che mette a disposizione diversi strumenti sia per la generazione di attacchi volti a testare la robustezza di un sistema, sia per la creazione di difese contro le minacce presentate dagli *adversarial attacks*.

3.3.1 Fingerprint adversarial perturbations

Gli attacchi presenti in ART sono molto generici in modo da poter essere utilizzati in domini differenti, per questo è necessario modificare i tre attacchi accennati in [2.3 Evasion attacks] per adattarli al dominio delle impronte digitali e, in particolare, per migliorarne le prestazioni.

In generale, tutti e tre gli attacchi calcolano le perturbazioni basandosi sul gradiente della rete per poi utilizzarle per modificare l'immagine in *input* iterativamente. Nello specifico, però, le immagini sono RGB, ovvero matrici $3 \times n \times m$ con tre: numero di canali; di conseguenza anche i

gradienti saranno tridimensionali e, quindi, si ottiene una perturbazione diversa per ogni canale. Ciò genera delle immagini perturbate che risultano avere colori diversi dal classico effetto in scala di grigi delle immagini originali; per questo la prima modifica da effettuare ad ogni attacco è la **trasformazione delle perturbazioni in scala di grigi** (matrici bidimensionali $1 \times n \times m$) subito dopo la loro generazione, per poi andare ad aggiungerle singolarmente ad ogni canale dell'immagine, in questo modo non si altera il colore originario dell'impronta.

L'esempio sotto mostra come una perturbazione in scala di grigi riesca a creare un'immagine molto più fedele all'originale.



Figura 26 - Immagine originale da perturbare

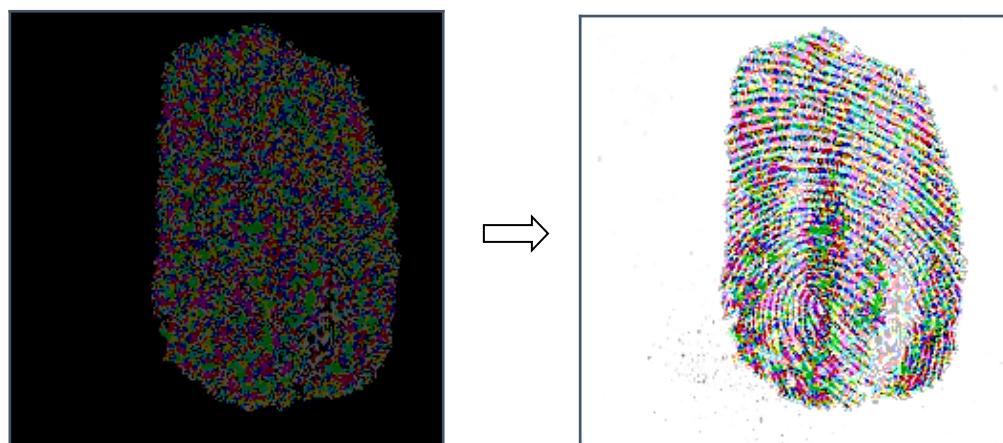


Figura 27 - Perturbazione originale e immagine perturbata

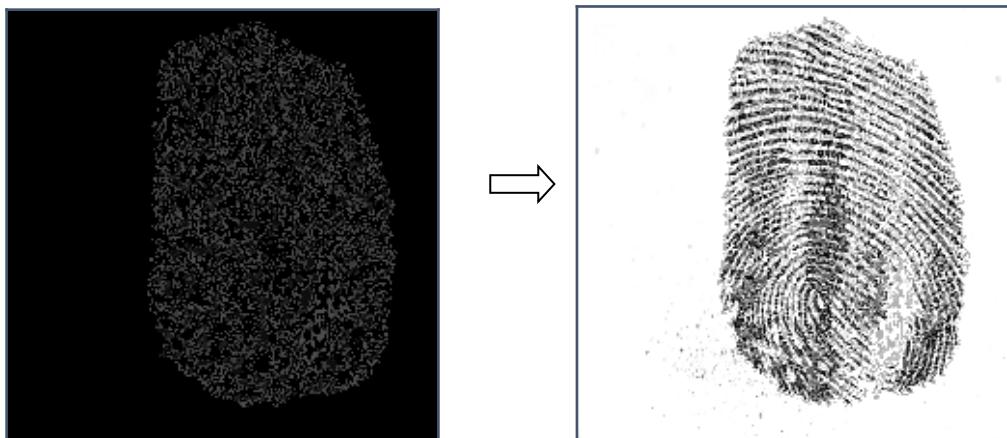


Figura 28 - Perturbazione trasformata in scala di grigi e immagine perturbata

Altro aspetto importante è la “localizzazione” della perturbazione, vuol dire che anche se l’immagine ha dimensione 500×500 , non tutti i pixel costituiscono realmente l’impronta, ma molti fanno parte dello sfondo. Il calcolo del gradiente, e quindi della perturbazione, però, riguarda tutti i pixel, per questo si ottiene una perturbazione che altera anche lo sfondo bianco. ART, permette di usare come parametro di ogni attacco una **maschera o ROI mask (Region of Interest Mask)**, ovvero un’immagine (con stessa dimensione dell’immagine da perturbare) binaria che associa 0 ai pixel dello sfondo e 1 ai pixel dell’impronta. Grazie ad essa è possibile annullare la perturbazione in prossimità dello sfondo: dove la maschera vale 0 assegna 0 alla perturbazione, mentre dove vale 1 lascia il valore inalterato. In questo modo, si ottiene una perturbazione localizzata solo in prossimità del *foreground*, ovvero l’impronta vera e propria. Questa caratteristica non è una modifica effettuata agli attacchi poiché è già supportata, ma è necessario creare un algoritmo che permetta di ottenere una maschera precisa che non vada a ritagliare parti dell’impronta o ad includere parti dello sfondo. L’idea che sta alla base della creazione della maschera è la *binarizzazione*, ovvero la capacità di assegnare 0 ai pixel dello sfondo ed 1 ai pixel del soggetto, minimizzando il margine di errore.

La generazione delle *ROI Mask* è stata effettuata utilizzando la libreria OpenCV. In particolare, è stato utilizzato il metodo di **Otsu**, un algoritmo

che calcola automaticamente la soglia ottima per dividere i pixel in bianchi e neri minimizzando la varianza intra-classe: a tutti i pixel con valore inferiore alla soglia viene associato il valore 0, viceversa a tutti i pixel con valore superiore alla soglia viene associato 1. Una volta ottenuta l'immagine binaria, sono state applicate ulteriori trasformazioni per estendere la zona bianca intorno all'impronta e non solo alle creste. Di seguito un esempio di creazione della *ROI mask* a partire da un'impronta, per poi applicare la maschera all'impronta iniziale con lo scopo di rimuovere il rumore di fondo:

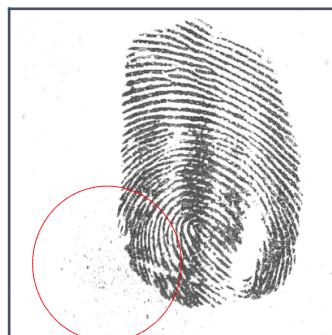


Figura 29 - Immagine iniziale con rumore

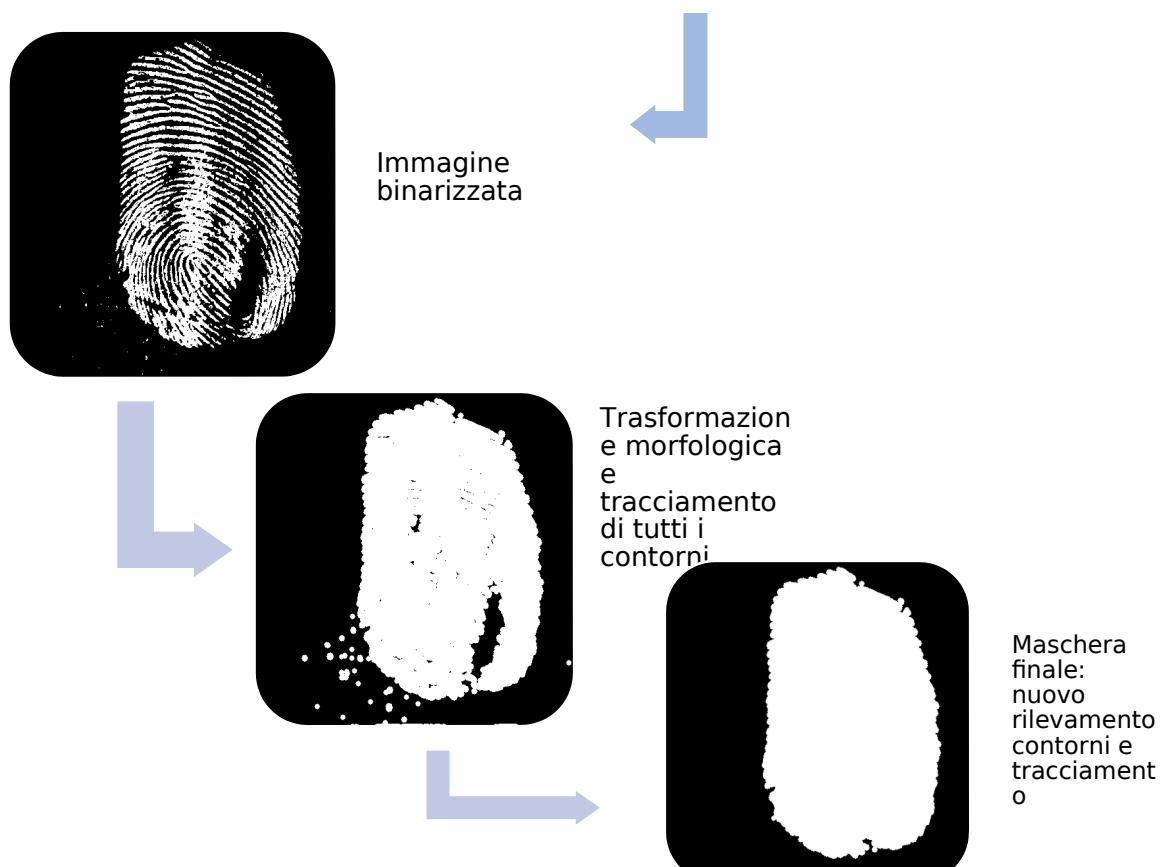


Figura 30 - Calcolo ROI mask

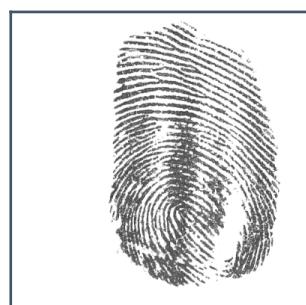


Figura 31 - Immagine finale pulita

La seconda modifica della versione iniziale degli attacchi riguarda l'introduzione di un nuovo parametro: **confidence**. Ogni volta che viene testata un'immagine la rete produce due risultati in uscita che costituiscono le predizioni e a partire da essi si calcolano le probabilità di appartenenza alle due classi; in particolare, quella relativa alla classe */ive* è detta **liveness** ed indica "quanto" l'immagine è considerata viva. Per confidence si intende proprio la *liveness* minima che si desidera avere in un'immagine perturbata affinché l'attacco possa considerarsi riuscito. Quindi, non basta che il *detector* sbagli la classe, ma deve anche predire l'immagine */ive* con una determinata probabilità minima.

L'ultima modifica consiste nell'introduzione della **funzione di testing basata su media**. Gli attacchi ad ogni iterazione effettuano la verifica della classe predetta dal modello dopo aver introdotto una perturbazione, ma in questo caso lo *scanner* creato utilizza una media di predizioni sulle *patches* e non una semplice predizione; quindi, è necessario utilizzare in ogni attacco la funzione di *testing* creata *ad hoc*.

Bisogna anche chiarire l'aspetto della dimensione delle immagini; come già detto gli attacchi sono basati su gradiente e per calcolarlo bisogna sottoporre le immagini al classificatore, il quale supporta solo immagini 224×224 ma le impronte originali sono 500×500 . Quindi, è necessario effettuare un primo *resize* di riduzione da 500×500 a 224×224 per dare l'immagine in ingresso all'attacco in modo da calcolare ed applicare le perturbazioni. In seguito all'applicazione della perturbazione all'immagine è necessario introdurre un *resize* da 224×224 a 500×500 all'interno della funzione *testing* per essere fedeli alle specifiche dello scanner. Questo approccio riguardante il primo *resize* dell'intera immagine è definito **Image Resize** in un articolo di Marrone³, nel quale descrive anche un secondo approccio che consiste nell'effettuare il *resize* della sola perturbazione, ovvero viene data in ingresso l'immagine ridimensionata a

³ Marrone Stefano, ricercatore dell'Università degli Studi di Napoli

224×224 , si calcola la perturbazione e quest'ultima viene data in uscita effettuando un *resize* a 500×500 per poi sommarla all'immagine originale. Questa tecnica è detta ***Noise Resize***, ma come afferma Marrone stesso il primo approccio restituisce risultati migliori e per questo è preferito rispetto al secondo.

Prima di passare alla descrizione dei vari attacchi e delle loro versioni, si vogliono mostrare due viste del processo di attacco realizzato:



Figura 32 - Processo di attacco vista alto livello

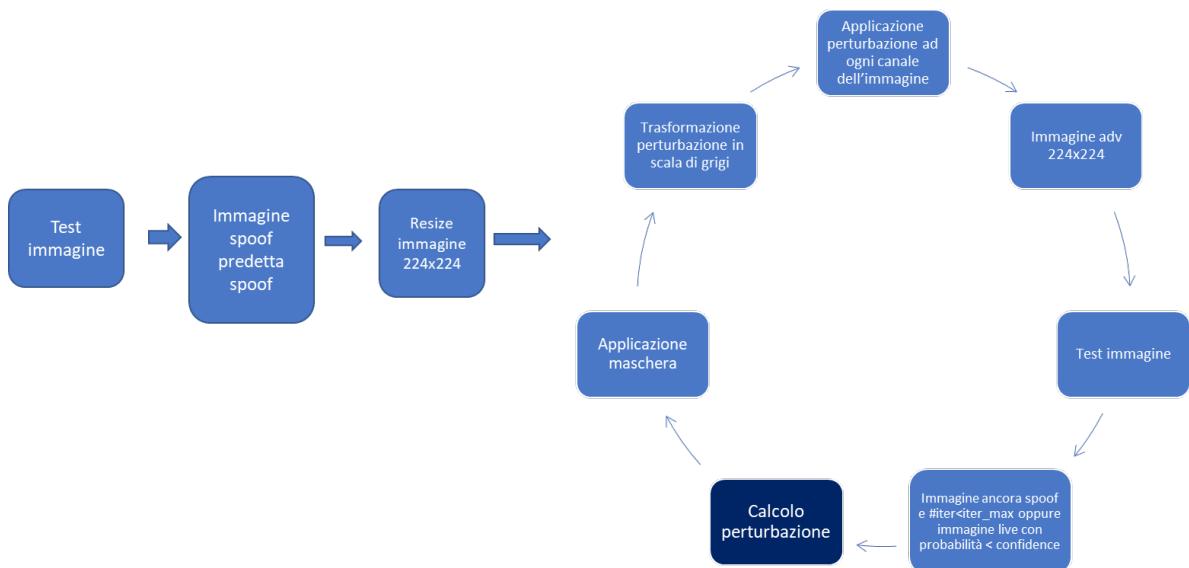


Figura 33 - Processo di attacco nel dettaglio (in blu scuro la prima fase del ciclo)

La prima fase prevede un *test* dell'immagine in modo da scartare le immagini *live* e le 2 immagini *spoof* predette erroneamente *live*; quindi, l'attacco si concentra sulle restanti 248 immagini *spoof*.

Infine, quando l'immagine perturbata soddisfa la condizione di uscita, essa ha dimensione 224x224, per questo viene effettuato un *resize* finale fuori dall'attacco per riportarla alla dimensione originale 500x500 prima di passare alla stampa.

Prima di descrivere le varie versioni degli attacchi si vogliono riportare le modalità di utilizzo dei tre attacchi forniti da ART e le modifiche specifiche effettuate singolarmente.

IFGSM

L'Iterative Fast Gradient Sign Method è fornito da ART all'interno dell'attacco generico Fast Gradient Method; infatti, ART permette di usare la versione base o la versione *iterative* semplicemente settando un parametro *booleano* di input: *minimal*.

Per questo attacco non sono necessarie modifiche nell'algoritmo poiché abbastanza semplice, veloce ed efficiente. Quindi basta porre *minimal=True* per utilizzarlo direttamente nella versione iterativa.

DeepFool

Per quanto riguarda DeepFool fornito da ART è stata inserita una modifica basandosi sul funzionamento di IFGSM e sul parametro *confidence* inserito. Infatti, come già accennato, spesso DeepFool converge in prossimità dell'iperpiano separatore ed è possibile "allontanare" il punto da esso moltiplicando la perturbazione con una costante, in ART detta *epsilon*. Il problema di questo approccio è che per ottenere una determinata *liveness* minima è necessario spesso utilizzare *epsilon* troppo grandi che perturbano troppo l'immagine; quindi, invece di effettuare un singolo *step* riguardante la moltiplicazione finale, viene inserito un ciclo in cui si effettua più molte la moltiplicazione con una

epsilon più piccola finché non si ottiene la probabilità minima richiesta da *confidence* o finché non si raggiunge il numero massimo di iterazioni: *max_over*.

APGD

L'attacco APGD fornito da ART non necessita di modifiche importanti, l'unica effettuata consiste nel rendere opzionale il parametro *random init*, ovvero la perturbazione casuale applicata inizialmente, in questo modo è possibile partire dall'immagine originale per perturbarla e non da un punto casuale all'interno della sfera iniziale.

Il processo di attacco appena mostrato, però, presenta dei problemi in fase di stampa:

- difficoltà nel creare stampe realistiche;
- rumore e ritagli introdotti dalla stampa distruggono le perturbazioni generate.

Per questo è necessario effettuare delle migliorie al processo e di seguito vengono proposte cinque versioni diverse che tentano di risolvere problemi differenti.

Da questo punto in poi le varie migliorie applicate al processo di attacco riguardano in ugual modo tutti e tre gli attacchi utilizzati; quindi, vengono mostrate varie versioni dell'attacco mostrato in Figura 32 ed ogni versione culmina con la stampa delle immagini perturbate su lattice.

3.3.2 V1-V2: Enhancement

La prima versione degli attacchi nasce con lo scopo di facilitare e migliore il processo di stampa; infatti, gli attacchi come mostrati finora realizzano

delle impronte come mostrate in Figura 28 le quali presentano varie sfumature di grigio (composto da 3 canali). Inoltre, gli attacchi non modificano solo i pixel corrispondenti alle creste ma anche le valli e non ha senso stampare uno “sfondo non bianco” per le impronte.

Per questo, la prima versione realizzabile dell'attacco si basa sul concetto di ***enhancement***, ovvero una trasformazione che rende l'impronta binaria andando ad associare il valore 0 alle creste e 1 al resto. Una volta creata l'immagine binaria a partire dall'originale, poi, bisogna replicare la matrice bidimensionale per ogni canale, poiché la rete supporta solo immagini RGB.

Di seguito un esempio di impronta finale:



Figura 34 - Impronta che ha subito l'enhancement

La funzione che effettua la binarizzazione dell'immagine è stata ottenuta modificando il lavoro di Utkarsh-Deshmukh⁴, il quale si basa a sua volta su un articolo del 1998 di Lin Hong.

L'enhancement utilizza un banco di filtri *gabor* volto a migliorare l'immagine dell'impronta digitale. L'orientamento dei filtri gabor è deciso dall'orientamento delle creste nell'immagine di input e la forma del filtro gabor si basa sulla frequenza e lunghezza d'onda delle creste.

⁴ <https://github.com/Utkarsh-Deshmukh/Fingerprint-Enhancement-Python>

Algoritmo

L'algoritmo utilizzato per la realizzazione della trasformazione è costituito da cinque fasi principali ed utilizza immagini input in scala di grigi ($2xnxm$):

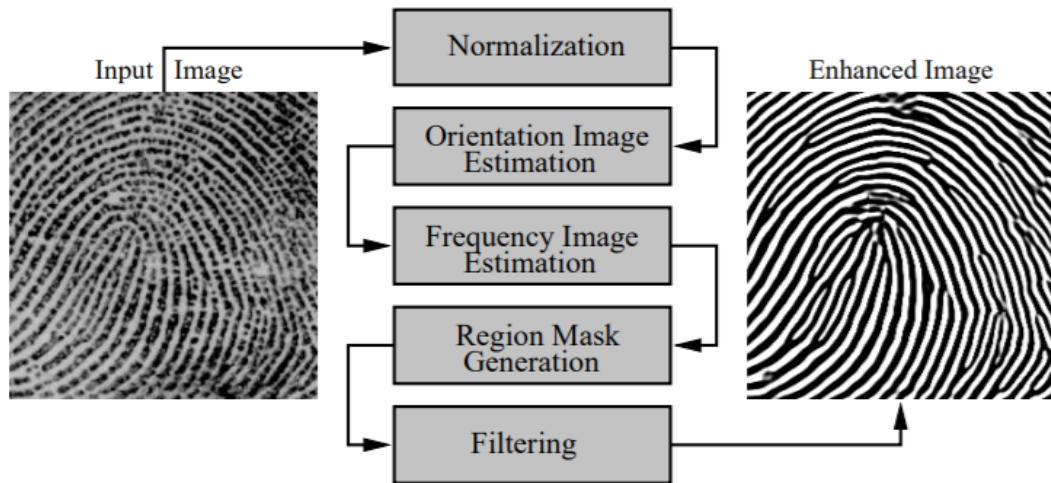


Figura 35 - Algoritmo enhancement

- Normalizzazione:** l'immagine in input viene normalizzata in modo che abbia delle specifiche media e varianza. Siano $I(i, j)$ il valore del livello di grigio del pixel (i, j) , M e VAR media e varianza stimate di I e $G(i, j)$ il valore normalizzato del livello di grigio del pixel (i, j) ; l'immagine normalizzata è definita come:

$$G(i, j) = \begin{cases} M_0 + \sqrt{\frac{VAR_0(I(i,j)-M)^2}{VAR}}, & \text{if } I(i, j) > M \\ M_0 - \sqrt{\frac{VAR_0(I(i,j)-M)^2}{VAR}}, & \text{otherwise,} \end{cases}$$

Dove M_0 e VAR_0 sono i valori di media e varianza desiderati.

La normalizzazione è un'operazione a livello pixel e non cambia la nitidezza delle strutture di creste e solchi. Lo scopo principale della normalizzazione è ridurre la variazione dei valori del livello di grigio lungo creste e solchi per facilitare i passi successivi.



Figura 36 - Impronta prima e dopo la normalizzazione

2. **Stima dell'orientamento locale:** viene stimata un'immagine di orientamento a partire dall'immagine normalizzata. L'immagine di orientamento rappresenta una proprietà intrinseca delle impronte digitali e definisce coordinate invarianti per creste e solchi in una zona localizzata. Esistono vari metodi per calcolare le coordinate, ma gli autori dell'articolo hanno sviluppato un algoritmo di stima dell'orientamento basato sulla media quadrata minima.



Figura 37 - Orientamento creste

3. **Stima della frequenza locale:** viene calcolata un'immagine di frequenza a partire dall'immagine normalizzata e da quella dell'orientamento. In determinate zone dove non compaiono minuzie, i livelli di grigio lungo creste e solchi possono essere modellati come un'onda di forma sinusoidale lungo una direzione normale all'orientamento della cresta locale e a partire da essa è possibile stimare le frequenze. Pertanto, la frequenza della cresta locale è un'altra proprietà intrinseca dell'impronta.

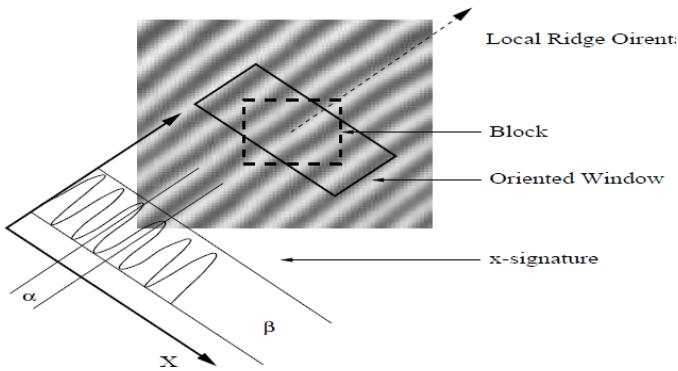


Figura 38 - Sinusoide lungo l'orientamento di una cresta

4. **Stima della maschera di regione:** la maschera di regione si ottiene classificando ogni pixel o blocco dell'immagine di input normalizzata in un blocco recuperabile, regione in cui creste/solchi sono ben definiti o corrotti leggermente da rumore e si possono ancora estrarre informazioni, o irrecuperabile. La classificazione dei pixel in recuperabili e irrecuperabili può essere eseguita in base alla valutazione della forma dell'onda formata dalle creste e dai solchi locali. Da notare che l'algoritmo può fallire se la percentuale di regioni recuperabili è inferiore alla soglia posta pari a 40.
5. **Filtraggio:** un banco di filtri Gabor, sintonizzato sull'orientamento e sulla frequenza delle creste locali, viene applicato ai pixel di cresta e solco nell'impronta digitale di input normalizzata per ottenere un'immagine migliorata (*enhanced*). Le configurazioni di creste e solchi paralleli con frequenza e orientamento ben definiti in un'immagine forniscono informazioni utili che aiutano a rimuovere rumore indesiderato. Le onde sinusoidali di creste e solchi variano lentamente in un orientamento locale costante. Pertanto, un filtro passa-banda sintonizzato sulle corrispondenti frequenza e orientamento può rimuovere efficacemente il rumore indesiderato e preservare le vere strutture di creste e solchi. I filtri Gabor hanno proprietà selettive in frequenza e orientamento e hanno una risoluzione particolarmente ottimale sia nel dominio spaziale che in quello della frequenza. Pertanto, è opportuno utilizzare i filtri Gabor

come filtri passa banda per rimuovere il rumore e preservare la struttura di creste e solchi.

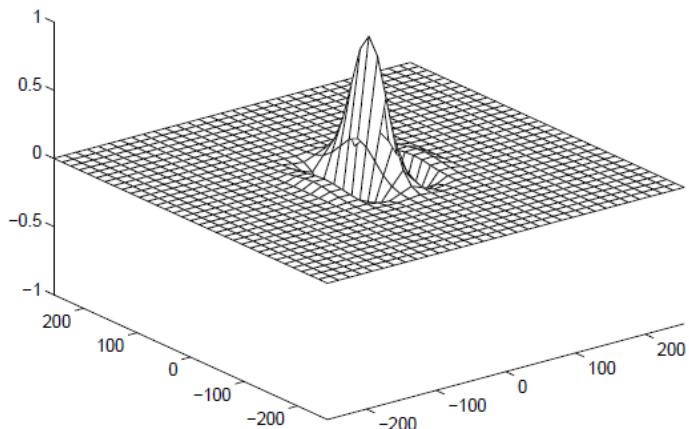


Figura 39 - Esempio filtro Gabor

Grazie all'*enhancement* le impronte finali risultano molto più semplici da stampare poiché sono presenti solo due valori per i pixel: 0,1 e si può decidere a quale mappare l'operazione di stampa e a quale l'operazione di non stampa.

Inoltre, la binarizzazione riesce a risolvere anche un altro problema importante, ovvero la perdita di informazioni nel processo di salvataggio e lettura; infatti, come già detto, l'Università degli Studi di Cagliari si è occupata della stampa delle impronte e, nel processo di salvataggio su disco e trasmissione di quest'ultime per poi essere lette e stampate, risulta presente una perdita di informazioni a causa dell'elevato numero di sfumature presenti nell'immagine. Ciò comporta che le predizioni effettuate sulle immagini prima e dopo il salvataggio/lettura risultano diverse. Ma, grazie alla semplicità delle immagini post-*enhancement* la perdita si annulla.

Come accennato, l'algoritmo originale è stato leggermente modificato:

- Inserimento trasformazione iniziale delle immagini in scala di grigi;
- Dato che l'algoritmo originale fallisce per alcune immagini con dimensioni troppo piccole come 224x224, è stato inserito un *resize*

intermedio obbligatorio alla dimensione 350x350 (*size* suggerita dall'autore ma opzionale) per immagini con dimensione inferiore a 350 oppure è stata inserita anche la possibilità di scegliere la dimensione finale che deve avere l'immagine;

- Trasformazione dell'immagine finale binaria bidimensionale in una matrice RGB ripetendo la stessa immagine per ogni canale;
- Applicazione maschera finale per rimuovere rumore intorno all'impronta.

L'ultima incognita rimanente è in quale parte del processo di attacco inserire la binarizzazione e, inoltre, quanto quest'ultima distrugge le perturbazioni introdotte dall'attacco.

V1: *enhancement* dopo l'attacco

Una prima versione del processo d'attacco prevede l'inserimento dell'*enhancement* alla fine dell'attacco in modo da realizzare l'attacco normalmente per poi "pulire" l'immagine per far risaltare le creste (*enhancement*). In questo modo però si è notato che per molte impronte i progressi ottenuti durante l'attacco vengono eliminati e le impronte continuano ad essere classificate bene. Per questo viene inserito un ulteriore ciclo contenente l'attacco, in cui, ogni volta che quest'ultimo termina viene effettuato l'*enhancement* e se l'immagine risultante non soddisfa i requisiti: classificata *live* e *liveness* minima, si serra nuovamente l'attacco sull'immagine risultante, ovvero binaria.

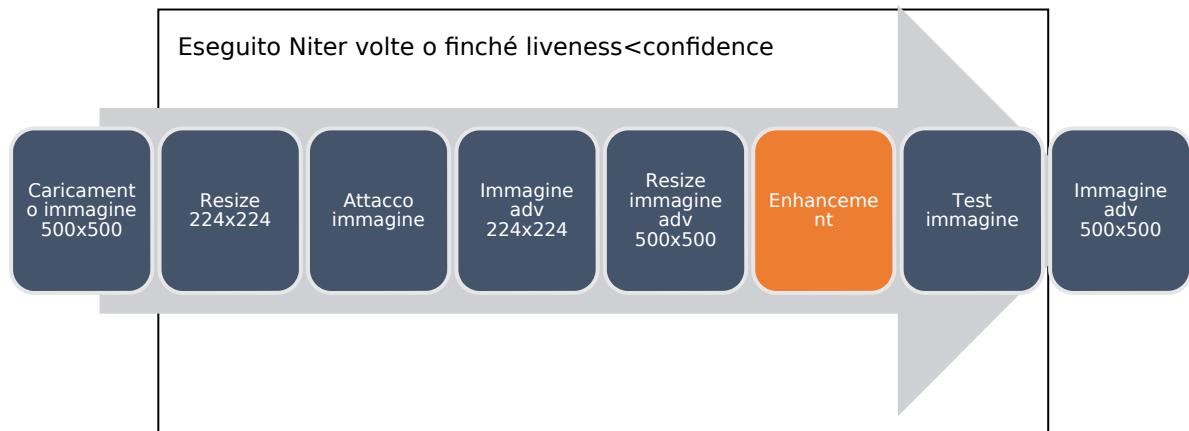


Figura 40 - Attacco versione 1

L'attacco così ottenuto ha portata alla stampa dei primi set di immagini i cui risultati, insieme agli altri relativi alle successive versioni, verranno approfonditi nel prossimo capitolo.

Di seguito vengono riportati i parametri scelti per i tre attacchi per realizzare la prima versione:

Attack	Parameters						
IFGSM	Eps=100 00	-	Eps_step=0 .05	Max_iter= 20	Confidence= 0.7	Niter (ciclo esterno) =5	
DeepFool	Epsilon= 0.1	Max_over=5	-	Max_iter= 10	Confidence= 0.7	Niter=5	
APGD	Eps=-	Nb_andom_init =0	Eps_step=0 .3	Max_iter= 20	Confidence= 0.7	Niter=5	

La versione 1 *digital*, ovvero non stampata, ha riportato valori abbastanza buoni di *accuracy* totale, si parla di circa il 50% delle 248 immagini classificate come *live* per ogni attacco.

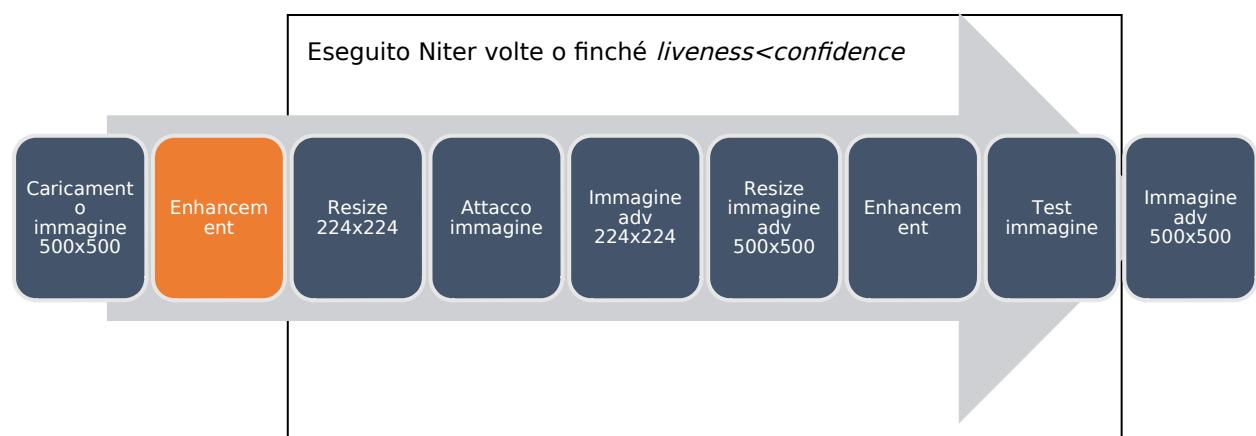


Figura 41 - Esempio impronta perturbata con APGD V1

V2: enhancement prima dell'attacco

Con lo scopo di migliorare i risultati viene effettuata una prova di inserimento dell'*enhancement* fin da subito, ovvero si applica all'immagine originale, per poi presentarla in ingresso all'attacco. Inoltre, dati i risultati buoni restituiti nella versione 1, diventa obbligatorio alzare il valore dei parametri degli attacchi per cercare di aumentare l'*accuracy* anche se a discapito del tempo di esecuzione. Infatti, mentre gli attacchi della versione 1 come tempo di esecuzione si aggirano intorno a 1-2 ore, in questo caso si parla di 5-6 ore.

In questo modo nasce la versione 2:



Bisogna precisare che dopo l'*enhancement* esterno al ciclo (riquadro arancione), prima di entra in quest'ultimo, si effettua un test dell'immagine per verificare se essa è ancora predetta *spoof*; questo perché per il 16% delle immagini basta l'*enhancement* per essere predette *live* e ciò fa risparmiare tempo all'attacco.

I risultati della V2 dimostrano valida l'intuizione di inserimento della binarizzazione fin da subito, poiché essa permette di guidare meglio l'attacco con la prima iterazione; infatti, per la maggior parte delle immagini risultano decisive le prime iterazioni esterne dell'attacco, se non si riesce a perturbare con successo l'immagine in queste iterazioni difficilmente si riuscirà con le successive.

Durante la creazione della versione due si è tentato l'inserimento dell'*enhancement* anche all'interno dei singoli attacchi (quindi all'interno della fase “attacco immagine”) ma ciò ha portato a tempi di esecuzione troppo elevati e non si hanno avuti miglioramenti nei risultati finali, ciò perché la binarizzazione in ogni iterazione interna dell'attacco distrugge letteralmente le lievi perturbazioni applicate.

Attac k	Parameters						
IFGSM	Eps=100 00	-	Eps_step=0. 007	Max_iter=1 00	Confidence=0.8	Niter (ciclo esterno) =30	
DeepFool	Epsilon=0.1	Max_over=10	-	Max_iter=5 0	Confidence=0.8	Niter=30	
APGD	Eps=0.5	Nb_andom_init =30	Eps_step=0. 3	Max_iter=1 50	Confidence=0.8	Niter=30	

Tutti gli attacchi hanno superato il 50% di *accuracy* ma APGD si è dimostrato il migliore raggiungendo l'80% in *digital*.



Figura 42 - Esempio impronta perturbata con APGD V2

3.3.3 V3: Random ROI

La terza versione dell'attacco nasce da un'analisi effettuata sui tre attacchi e sulle relative impronte stampate. Gli attacchi, infatti, applicano perturbazioni maggiori in determinati punti dell'immagine a seconda del valore del gradiente; per questo, se durante il processo di stampa le zone in cui si è concentrato l'attacco vengono corrotte dal rumore o ritagliate, i progressi fatti nel processo d'attacco vengono persi.

A tale scopo è stata ideata una versione che permette ai vari attacchi di concentrarsi su singole zone localizzate dell'impronta in ogni iterazione per ottenere perturbazioni finali uniformemente distribuite.

Per la realizzazione della distribuzione uniforme della perturbazione è stata introdotta in ogni attacco una **random ROI**, ovvero un ritaglio casuale circolare della maschera, in maniera tale da ritagliare la perturbazione generata. Ciò, ovviamente, fa sviluppare l'attacco in maniera diversa: mentre prima si perturbava interamente l'immagine, ora si perturbano singole zone alla volta ed è necessario utilizzare più iterazioni per ottenere un risultato simile al precedente, ma in questo modo ad ogni iterazione l'attacco si troverà a calcolare il gradiente su un'immagine non interamente perturbata e ciò assicura una distribuzione uniforme delle varie perturbazioni. In seguito, poi, è necessario verificare

se l'aumento del tempo di esecuzione è compensato da un aumento dell'*accuracy* delle immagini stampate e tutto ciò verrà discusso nel prossimo capitolo.

Quindi, la versione 3 modifica il ciclo interno dell'attacco e non il processo generale, il quale resta lo stesso utilizzato dalla V2:

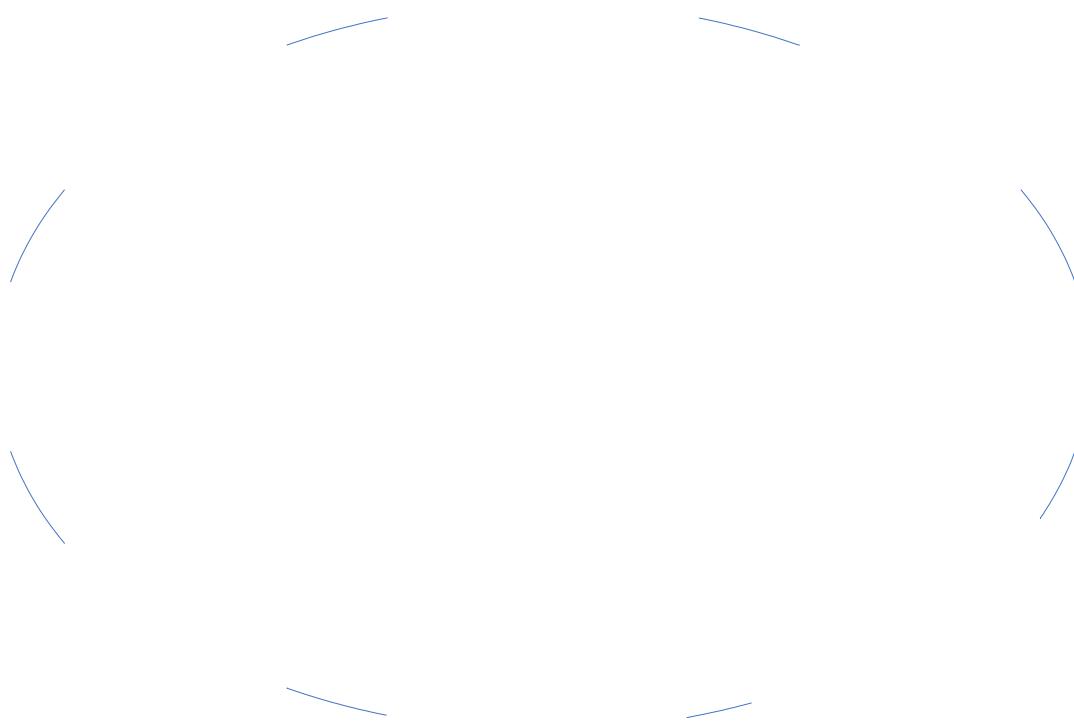


Figura 43 - Attacco con random ROI

Si mostra, di seguito, la creazione di una *random ROI*, la quale viene realizzata grazie all'utilizzo della libreria OpenCV. Il ritaglio casuale circolare della maschera è sviluppato in più passi:

- 1) Individuazione raggio della circonferenza contenente interamente l'impronta, il quale è calcolato nel seguente modo:
 - a. Individuazione centro impronta;
 - b. Individuazione punto di estremo dell'impronta;
 - c. Calcolo differenza tra i punti a e b per ottenere il raggio r.



Figura 44 - Circonferenza circoscritta all'impronta

- 2) A questo punto si individua un punto casuale all'interno della maschera relativa all'impronta e si utilizza come centro della circonferenza di raggio r/a (il centro è scelto tra i punti interni alla maschera per non ottenere *random ROI* finali vuote, ovvero nere);

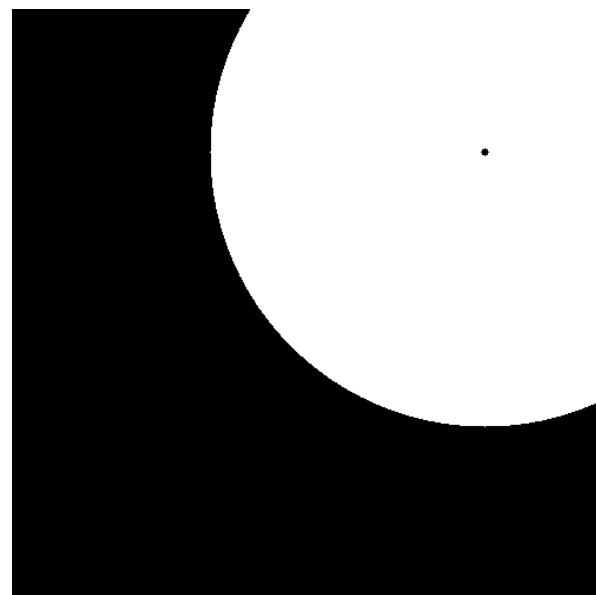


Figura 45 - Circonferenza casuale con centro all'interno dell'impronta; raggio= r

- 3) Moltiplicazione tra circonferenza casuale e maschera originale per ottenere la *random ROI*.

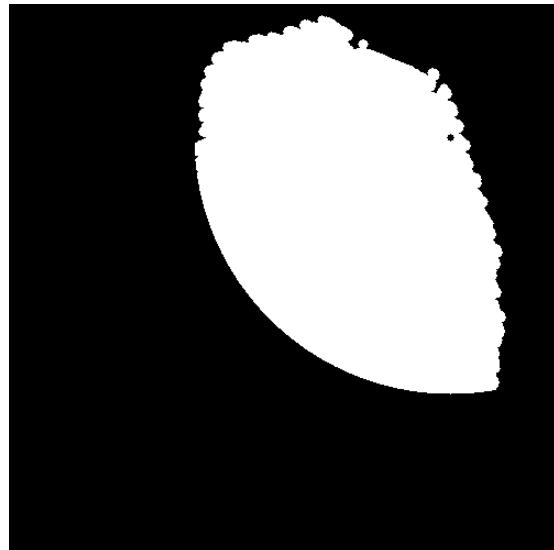


Figura 46 - Random ROI

Quindi, la parte dell'impronta che verrà modificata durante l'i-esima iterazione dell'attacco è la seguente:

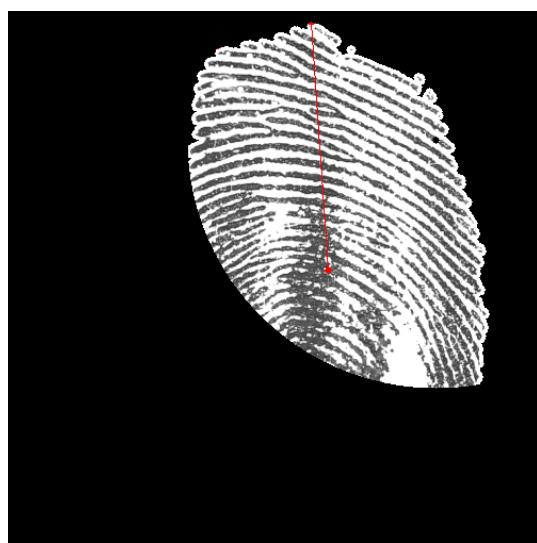


Figura 47 - Zona dell'impronta da perturbare

La funzione che genera la maschera casuale ha come parametro in ingresso il valore **a**, ovvero il fattore per cui dividere il raggio per ottenere il raggio della circonferenza casuale. Durante la generazione delle immagini per la versione 3 è stato usato come fattore $a=1$ per non ottenere random ROI troppo piccole e quindi velocizzare l'attacco.

Attack	Parameters					
k	Eps=100 00	-	Eps_step=0. 007	Max_iter=200	Confidence =0.8	Niter (ciclo esterno) =30
IFGSM	Eps=100 00	-	Eps_step=0. 007	Max_iter=200	Confidence =0.8	Niter (ciclo esterno) =30
DeepFool	Epsilon=0.1	Max_over=15	-	Max_iter=100	Confidence =0.8	Niter=30
APGD	Eps=0.5	Nb_andom_init =15	Eps_step=0. 3	Max_iter=100	Confidence =0.8	Niter=15

Per i primi due attacchi è stato possibile alzare il numero di iterazioni, mentre per APGD è stato necessario diminuirlo dato il tempo di esecuzione troppo elevato, ma su un *set* di dati ridotto è risultato inutile utilizzare un numero di iterazioni più elevato con questo approccio.

L'*accuracy* ottenuta da questa versione è risultata leggermente inferiore in confronto a V2, ma superiore a V1.



Figura 48 - Esempio impronta perturbata con APGD V3

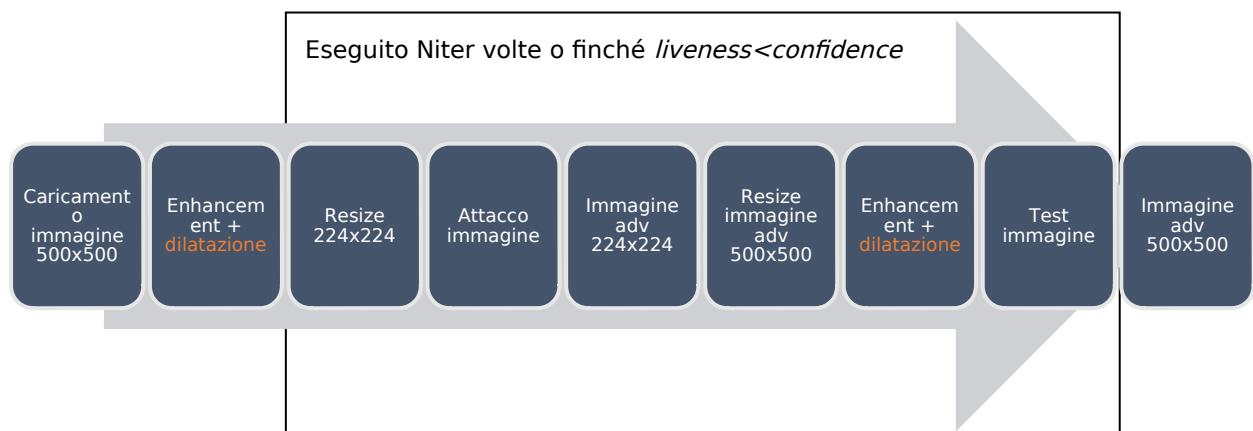
3.3.4 V4: Dilatation

A causa del tempo di esecuzione elevato ed il peggioramento dell'*accuracy* della versione 3, si è cercato un modo alternativo per ridurre l'influenza del rumore introdotto dalla stampa. Dall'analisi delle immagini perturbate digitali e stampate è sorta spontanea l'idea di

cercare di ridurre lo spazio in cui possa inserirsi il rumore, ovvero nei solchi dell'impronta.

Per questo la versione quattro utilizza una **dilatazione** delle creste dell'impronta in modo da ridurre la dimensione dei solchi (zone bianche) ma mantenendo comunque le minuzie.

Il processo di dilatazione è stato realizzato grazie all'ausilio della libreria Scikit-Image ed è stato inserito immediatamente dopo ogni *enhancement*; in questo modo, essendo un'operazione veloce, il tempo di esecuzione non viene alterato. La V4 è stata ottenuta inserendo la dilatazione nel processo della versione due:



In realtà la dilatazione è stata realizzata con il processo di **erosione** della libreria accennata, poiché le creste sono nere e quindi i relativi pixel valgono 0. L'erosione morfologica pone un pixel in posizione (i,j) pari al minimo valore tra tutti i pixel nell'intorno centrato in (i,j); in questo l'erosione restringe le regioni luminose e ingrandisce le regioni scure.

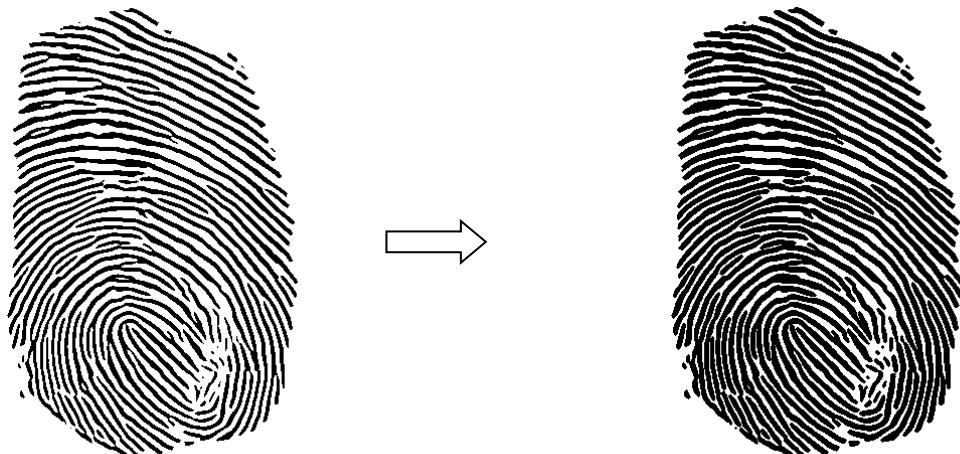


Figura 49 - Dilatazione

La funzione per individuare l'intorno di ogni pixel utilizza un elemento detto *footprint*, il quale può avere varie forme, ma per la realizzazione dell'attacco è stato scelto un elemento circolare: *ball* con raggio 1. Per raggi con valori superiori ad 1 si ottengono sovrapposizioni tra le creste e ciò corrompe troppo l'impronta, come nel seguente esempio:



Figura 50 - Dilatazione eccessiva

La dilatazione, come sarà mostrato in seguito, restituisce valori di *accuracy* che superano la versione 2; infatti, per ogni attacco si supera il 90%. Inoltre, restituisce buoni valori calcolati in previsione di stampa ma riduce l'*accuracy* dell'*authentication*.

La dilatazione è stata inserita all'interno del processo di attacco e non è stata effettuata sulle immagini risultanti dell'attacco V2; in questo modo essa guida l'attacco nell'applicazione delle perturbazioni. È stata effettuata una prova applicando la dilatazione a immagini già perturbate ed i risultati, seppur buoni, sono stati peggiori.

I parametri utilizzati per i vari attacchi sono gli stessi mostrati per la versione due quindi non vengono riportati.



Figura 51 - Esempio impronta perturbata con APGD V4

3.3.5 V5: Pepper Noise

L'ultima versione del processo di attacco prevede l'applicazione di rumore artificiale al processo definito nella versione quattro.

Durante l'analisi delle immagini stampate si è cercato di emulare il rumore introdotto in fase di stampa per poter in qualche modo predire il comportamento delle immagini; infatti, nel capitolo successivo sarà mostrato lo studio effettuato ed i conseguenti risultati.

Oltre alla previsione del comportamento, però, si è cercato di utilizzare il rumore per guidare il processo d'attacco in modo da ottenere immagini che, una volta stampate, siano abbastanza robuste da essere predette ancora *live*. Sempre in relazione al processo di attacco mostrato precedentemente, il rumore artificiale è stato introdotto solo prima del

testing in modo da avere una previsione del comportamento post-stampa, in questo modo se l'immagine non supera il test si esegue un'ulteriore iterazione del ciclo esterno.



Le impronte una volta stampate, se non troppo corrotte, assumono un aspetto simile alla seguente immagine:



Figura 52 - Esempio impronta stampata

Come si può notare, il rumore consiste nell'introduzione di pixel più scuri nelle zone relative ai solchi; per questo come rumore artificiale, sempre grazie all'utilizzo della libreria Scikit-Image, è stato scelto di utilizzare un **random noise** di tipo **pepper**, ovvero un rumore che si ottiene ponendo a 0 pixel casuali con il valore 1 (solchi).

Il problema però è che in questo modo si ottengono solo pixel totalmente neri, mentre nella realtà si hanno varie sfumature di grigio; a tal

proposito è stato necessario sommare al rumore dei valori casuali compresi tra 0 ed 1 per attenuare i pixel totalmente neri.

Il processo di applicazione del rumore può essere riassunto nei seguenti passi:

- 1) Creazione immagine bianca $500x500$ (composta da soli pixel posti ad 1);
- 2) Applicazione *pepper random noise* all'immagine bianca. La funzione permette anche di specificare un parametro che indica la quantità di pixel da modificare: *amount*;

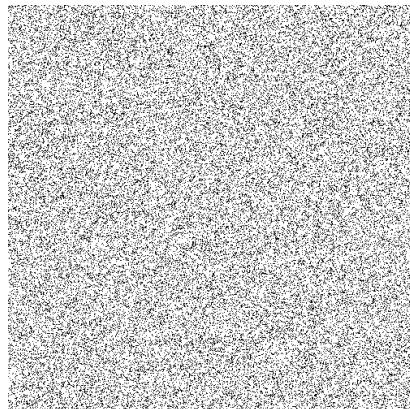


Figura 53 - Pepper random noise

- 3) Somma di valori casuali in $[0,1]$ all'immagine rumorosa per attenuarne l'intensità;

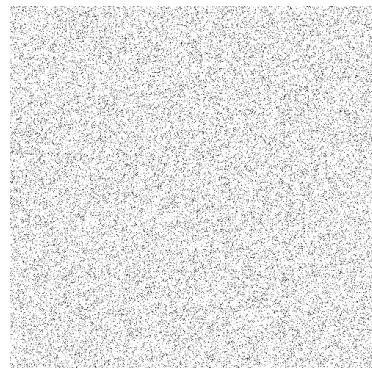


Figura 54 - Rumore attenuato

- 4) Moltiplicazione del rumore per l'impronta dopo l'*enhancement*;
- 5) Applicazione maschera e *clipping* valori in $[0,1]$.



Figura 55 – Esempio immagine con rumore artificiale; amount=0.2

La versione 5 è stata realizzata solo per l'attacco APGD poiché è quello che ha riportato i migliori valori di *accuracy* in ogni versione e come parametro *amount* è stato utilizzato il valore 0.3, in modo da simulare una situazione in cui la stampa altera molto l'impronta.

Inoltre, bisogna precisare che per considerare realizzato l'attacco e per uscire dal ciclo esterno, l'impronta deve superare il *testing* con e senza rumore ed è stata definita una *confidence* diversa per il *test* del rumore.

Attack	Eps	Nb_random_init	Eps_step	Max_iter	Confidence	Confidence noise	Number	Amount
APGD	0.5	30	0.2	150	0.8	0.65	20	0.3

L'*accuracy* risulta leggermente inferiore alla versione quattro (sempre superiore al 90%) ma le previsioni di stampa migliorano nettamente.



Figura 56 - Esempio impronta perturbata con APGD V5

3.5 Authentication

Lo scopo principale del lavoro svolto è aggirare il sistema di *liveness detection* come mostrato precedentemente, ma è anche molto importante verificare se le impronte perturbate vengono alterate troppo, al punto da non superare il sistema di autenticazione.

Dato che ci si è soffermati sui rilevatori di vitalità, all'interno del processo di attacco non è stata introdotta nessuna fase che assicura il superamento del *match*; quindi, per ogni impronta digitale o stampata ottenuta bisogna verificare se poi è realmente utilizzabile per aggirare anche il sistema di autenticazione.

Come già descritto in [2.4 Authentication System], per la realizzazione di un sistema di autenticazione molto semplice si utilizzano i due componenti: Mindtct e Bozorth; il primo è utilizzato per estrarre le minuzie dalle impronte ed il secondo per verificarne la corrispondenza. Quindi, si verifica solo se l'impronta perturbata corrisponde all'impronta originale presente nel *test set*, in questo modo se quest'ultima supera il match, anche la perturbata lo supererà.

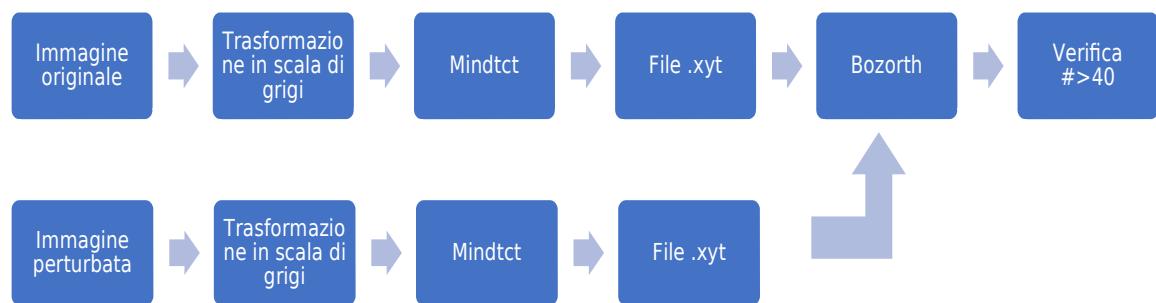
I due componenti utilizzati sono stati compilati per l'ambiente Windows ed utilizzati con Matlab su set di dati offline, ovvero set presenti in locale

sulla macchina; quindi, una volta ottenute le impronte perturbate è necessario metterle a disposizione di Matlab per verificarne la corrispondenza con le originali.

Nello specifico il funzionamento dei due componenti è molto semplice:

- Mindtct prende in ingresso due immagini (una originale e l'altra perturbata) in scala di grigi e restituisce in uscita otto file per ogni immagine, i quali contengono informazioni riguardanti le minuzie, come già descritto nel capitolo 2;
- Bozorth utilizza i file **xyt** delle due immagini per contare quante minuzie corrispondono. Il file di testo **xyt** riporta i risultati del rilevamento delle minuzie come coordinate x, y, theta, e la qualità dei punti di minuzia rilevati; ogni riga nel file corrisponde ad una singola minuzia.

Infine, una volta ottenuto il numero di minuzie corrispondenti bisogna verificare se è maggiore di 40, in questo modo le due impronte superano il *match*.



Per migliorare la valutazione del sistema di autenticazione, il processo appena descritto, il quale può essere definito come standard, viene modificato aggiungendo l'*enhancement*, ovvero viene applicata la binarizzazione come già vista ad entrambe le immagini in input, in modo da far risaltare le minuzie e rimuovere il rumore se presente. Questa modifica è importante soprattutto per la verifica delle immagini stampate le quali conterranno del rumore e con la binarizzazione si ottiene un

aumento dell'*accuracy*, mentre per le immagini digitali non si hanno molti cambiamenti poiché esse hanno già subito l'*enhancement* e le immagini originali sono abbastanza definite.

Nel caso in cui si voglia inserire la verifica del *match* all'interno del processo d'attacco è possibile utilizzare i due componenti anche su Colab; infatti, è bastato compilare Mindtct e Bozorth direttamente tramite il terminale messo a disposizione dalla piattaforma per ottenere due file eseguibili utilizzabili nel codice. Ciò potrebbe essere interessante per uno sviluppo futuro.

3.6 Printed Fingerprints

L'ultimo passaggio del progetto realizzato riguarda lo studio delle impronte perturbate stampate. Come già accennato, grazie alla collaborazione con l'Università degli Studi di Cagliari, è stato possibile stampare le impronte ottenute con i vari attacchi per poi studiarne le immagini ottenute da una loro scannerizzazione. Si ricorda che il processo di stampa/scannerizzazione introduce ritagli e rumore.

In questo paragrafo si vuole descrivere il processo di creazione degli artefatti e la struttura del set di impronte risultante.

L'Università degli Studi di Cagliari ha collaborato alla creazione del progetto realizzando stampe delle impronte perturbate su diversi materiali ed in questo paragrafo si vuole descrivere il processo di creazione degli artefatti con lattice. Il processo può essere diviso in quattro fasi principali:

- 1) Creazione di un foglio digitale contenente il negativo di massimo 70 immagini digitali perturbate;

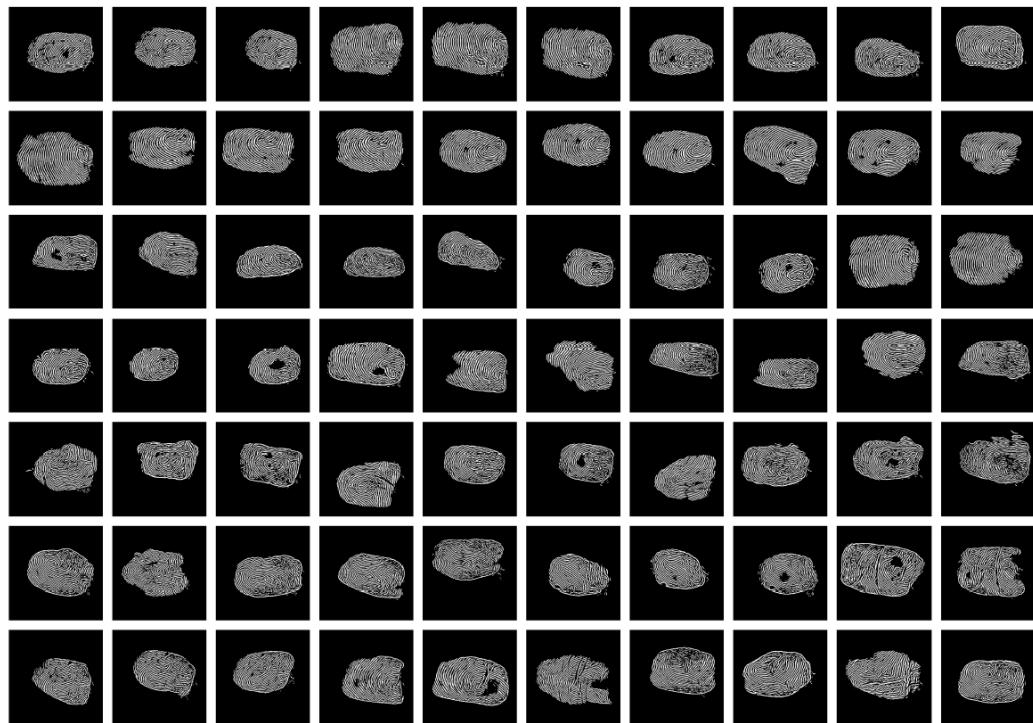


Figura 57 - Esempio foglio con negativo impronte

- 2) Stampa del foglio con una stampante laser;
- 3) Uno strato di lattice viene depositato sopra le stampe delle impronte digitali e si aspetta che si asciughi;



Figura 58 - Esempio applicazione lattice su stampo

4) Acquisizione degli artefatti tramite scannerizzazioni ripetute della stessa impronta.

Viene utilizzato il negativo delle immagini poiché la stampante andrà a stampare solo il nero e lascerà il bianco vuoto, in questo modo per ogni cresta dell'impronta si otterrà un solco nel foglio dato dalla presenza dello sfondo nero rialzato e, in seguito, versando il lattice sul foglio esso si deposita nei solchi del foglio, ovvero le creste.

Una volta asciugato il lattice viene rimosso dal foglio e si ottengono gli artefatti che possono essere acquisiti in digitale.

Per ogni impronta vengono prelevate dieci acquisizioni variando leggermente la posizione dell'impronta sullo scanner in modo da simulare uno scenario reale in cui l'impronta non si trova sempre nella stessa posizione.

Per le immagini relative alla versione 1 dell'attacco il processo di stampa ha generato artefatti leggermente più grandi rispetto alle impronte originali e ciò, come si vedrà nel prossimo capitolo, porta ad un calo dell'*accuracy*, sia per il *liveness detector* sia per il sistema di autenticazione; per questo dalla versione 2 in poi è stato ideato un nuovo algoritmo⁵ di creazione dei fogli che realizza stampi contenenti impronte della stessa dimensione degli originali e ciò ha portato ad un miglioramento dei risultati.

Per quanto riguarda i set di impronte stampate ricevuti dall'Università degli Studi di Cagliari, essi sono organizzati in cartelle, una per ogni foglio, ed ogni foglio contiene massimo 700 acquisizioni, divise in 10 per ogni impronta e contenute in una cartella dedicata.

Di seguito è riportato un esempio di dieci acquisizioni diverse della stessa impronta stampata per far notare le differenze di posizione, inclinazione, rumore e ritaglio.

⁵ Algoritmo creato da Stefano Marrone



Figura 59 - Esempio 10 acquisizioni di un artefatto

CAPITOLO 4: RISULTATI

In questo capitolo si descrivono i risultati ottenuti applicando le varie versioni degli attacchi al *set* di immagini in lattice. *In primis*, si descrive il comportamento delle immagini digitali, ovvero le immagini perturbate prima del processo di stampa, per poi passare all'analisi di alcuni *set* di immagini stampate. Bisogna precisare che non tutte le immagini digitali sono state stampate a causa dei lunghi tempi necessari ed alla disponibilità limitata del materiale durante lo sviluppo del progetto.

Per guidare la scelta dei set di immagini da stampare viene effettuato uno studio del rumore per ogni set, in modo da prevedere il comportamento post-stampa e scegliere le migliori immagini da stampare.

Infine, viene descritto uno studio incrementale dei risultati nel quale si cerca di massimizzare l'*accuracy* delle impronte stampate combinando immagini con solo *enhancement* a immagini relative ad attacchi diversi.

Per calcolare il *match* delle impronte viene utilizzato il metodo con *enhancement* descritto in precedenza ed un altro parametro utile alla descrizione dei risultati è il **Mean Liveness Score (MLS)**, ovvero la media delle *liveness* relative alle sole impronte predette *live* di un set di immagini.

4.1 Digital

Il primo risultato che bisogna evidenziare, anche se già accennato precedentemente, riguarda le immagini originali: **clean**. Si ricorda che la porzione di test set in lattice è composta da 250 immagini *spoof* e, senza alterarne nessuna, il *liveness detector* sbaglia la predizione su 2 di esse:

- **2/250** con *accuracy* **0.80%**;
- **MLS=96.11%**.

Quindi, in seguito si farà riferimento ad un set di dati composto da 248 immagini predette *spoof*.

Le 2 immagini predette *live* senza subire alterazioni non vengono considerate per il *match* poiché il risultato è ritenuto non rilevante per lo scopo del progetto.

Di seguito, sono riportati i risultati relativi al solo *enhancement* ed ai vari attacchi.

4.1.1 Binarized Clean

Come detto in precedenza, una porzione delle 248 immagini è predetta *live* in digitale applicando solo la binarizzazione e la maschera. In particolare, si ottengono:

- **40/248** impronte *live* con un'*accuracy* del **16.13%**;
- **30/40** hanno una *liveness* superiore all'80%;
- **MLS=87.97%**;
- **Match: 100%**, poiché si confrontano le *binarized clean* con loro stesse, dato il metodo usato.

Per i vari attacchi descritti, escludendo la versione 1, si ha un *enhancement* iniziale seguito da un test dell'immagine, in questo modo per ogni attacco si evitano le 30-40 (dipende dalla *confidence* scelta) immagini accennate poiché già *live* e ciò permette di risparmiare tempo.

Discorso a parte viene fatto per la versione 5, poiché si effettua un test anche sulle versioni rumorose delle impronte. In generale, quindi, si può affermare che la maggior parte degli attacchi produce un sottoinsieme di almeno 30 immagini uguali, poiché già *live* con la binarizzazione.

4.1.2 Attacks

In questo paragrafo vengono riportati i risultati in digitale di tutte le versioni di attacco descritte ed una loro comparazione. Inoltre, per ogni versione si ricorda che vengono sferrati i 3 attacchi modificati: IFGSM, DeepFool e APGD.

Dato l'elevato numero di valori da riportare come: versioni, attacchi, *liveness* e *match*, sono state create delle tabelle riassuntive per ogni attacco.

IFGSM

L'attacco IFGSM nonostante sia il più veloce tra i tre attacchi è risultato non essere il peggiore in termini di accuracy delle impronte *live*, ma a causa della disponibilità limitata dei materiali ed ai risultati peggiori del *match* è stato deciso di non stampare alcun set di queste impronte.

Dalla tabella sottostante si può notare come la prima versione sia la peggiore e l'ultima la migliore, sia in termini di accuracy che in termini MLS e match.

VERSION	LIVE/TOT	ACCURACY	MLS	MATCH/LIVE	MATCH ACCURACY
1 D	136/248	54,84%	86.83%	71/136	52,21%
2 D	177/248	71,37%	91.11%	127/177	71,75%
3 D	169/248	68,15%	92.78%	126/169	74,56%
4 D	224/248	90,32%	94.52%	192/224	85,71%

DeepFool

DeepFool è risultato essere il peggiore tra i tre attacchi in termini di accuracy, ma, nonostante ciò, in seguito al processo di stampa ha restituito buoni risultati come verrà mostrato nei paragrafi successivi.

In questo caso si può notare come versioni successive migliorino sempre più i risultati, tranne per la versione tre che ha un leggero calo per quanto riguarda l'*accuracy*.

VERSION	LIVE/TOT	ACCURACY	MLS	MATCH/LIVE	MATCH ACCURACY
1 D	98/248	39,52%	89.62%	87/98	90,82%
2 D	147/248	59,27%	91.64%	109/147	74,15%
3 D	146/248	58,87%	92.48%	112/146	76,71%
4 D	220/248	88,71%	94.49%	162/220	73,64%

APGD

L'ultimo attacco considerato, APGD, ha restituito il maggior numero di immagini */ive* per quasi ogni versione del processo d'attacco, per questo è stato deciso di calcolare solo per questo attacco la versione 5. Come si può notare dalla tabella sottostante, però, il *match* cala drasticamente con l'ultima versione, ciò perché si individua la versione dell'immagine perturbata più resistente al rumore senza tener conto di quanto viene distorta l'immagine e quindi si perdono minuzie.

Inoltre, il *mean liveness score* per la versione 5 raggiunge quasi il 100%, ciò fa capire l'elevata robustezza delle impronte di questo set.

VERSION	LIVE/TOT	ACCURACY	MLS	MATCH/LIVE	MATCH ACCURACY
1 D	140/248	56,45%	88.98%	56/140	40,00%
2 D	210/248	84,68%	90.28%	165/210	78,57%
3 D	160/248	64,52%	94.23%	130/160	81,25%
4 D	235/248	94,76%	95.78%	199/235	84,68%
5 D	228/248	91,94%	98.44%	123/228	53,95%

Infine, è stato scelto di riportare una tabella per comparare le versioni uno e quattro; in particolare, la tabella mostra di quanto la versione 4 migliora determinati parametri rispetto alla versione 1:

Attack	Δ accuracy	Δ MLS	Δ numero match
IFGSM	+35.48%	+7.69%	+121
Deepfool	+49.19%	+4.87%	+75
APGD	+38.31%	+9.46%	+67

Nonostante l'MLS per la versione quattro sia calcolato su un numero maggiore di immagini rispetto alla versione 1, esso aumenta lo stesso e ciò dimostra il miglioramento introdotto con l'aumento del parametro *confidence*. Per il *match* è stato necessario parlare di “numero” di immagini e non di percentuale poiché non è calcolato sullo stesso numero di immagini totali.

4.2 Studio rumore

Durante la stampa dei primi set di immagini è risultato evidente il problema del rumore introdotto, per questo è stato deciso di effettuare uno studio riguardante quest'ultimo, come accennato nel capitolo precedente.

Lo studio si occupa di aggiungere un rumore artificiale (identico a quello descritto nella versione cinque degli attacchi) ai vari set di impronte digitali per ottenere una stima dei risultati che si ottengono post-stampa.

In particolare, per ogni set sono stati calcolati due valori:

- Numero di immagini *live* con parametro *amount* pari a 0.1;
- Numero di immagini *live* con parametro *amount* pari a 0.2.

Combinando questi due risultati con i valori *Live/Tot* digitali senza rumore si ottiene un intervallo di valori *Live/Tot* in cui potrebbe trovarsi il valore reale post-stampa.

Di seguito sono mostrati gli intervalli calcolati per i vari set di dati, dove ogni elemento $a_{ij} - b_{ij}$ della matrice indica l'intervallo dato da *amount*=0.2 e *amount*=0.1.

Att ack	IFGSM	APGD	DeepFool
Version			
Versione 1	30-50	20-50	20-45

Versione 2	60-90	45-85	40-75
Versione 3	60-90	45-90	50-85
Versione 4	40-85	35-95	30-60
Versione 5		100-190	

I valori mostrati indicano il numero di immagini *live* previste post-stampa, ma il numero può aumentare fino al valore massimo ottenuto in *digital* se non viene introdotto rumore durante la stampa, o diminuire ad un valore inferiore ad amount=0.2 se il rumore introdotto è eccessivo.

Da notare come le versioni 4 restituiscano previsioni migliori delle altre e, in particolare, come la versione 5 sia l'unica a superare la soglia di 100 immagini come limite inferiore dell'intervallo.

Lo studio del rumore è stato effettuato poiché inizialmente si avevano a disposizione i set di impronte stampate relativi alle versioni uno e due di APGD e DeepFool e bisognava capire se valesse la pena proseguire con la stampa delle versioni successive.

Data la quantità limitata di lattice si è scelto di proseguire con la stampa delle versioni successive degli stessi attacchi poiché hanno riportato un incremento nei valori stimati e, nonostante IFGSM riportasse ottimi valori previsti post-stampa, è stato deciso di focalizzarsi sul confronto tra due soli attacchi per versioni successive piuttosto che confrontare tre attacchi su un numero inferiore di versioni.

Nonostante ciò, è stato deciso di riportare anche i risultati di IFGSM poiché incoraggianti e quindi utili per uno studio futuro.

4.3 Printed

Per quanto riguarda le immagini stampate viene riportata una singola tabella contenente tutti i valori in modo da effettuare fin da subito un confronto.

È stata effettuata una prova di stampa anche delle immagini originali e delle *binarized* per studiarne il comportamento:

- Clean: solo 2 immagini su 248 risultano *live* post-stampa e sono differenti dalle immagini effettivamente *live* in digitale;
- Binarized: circa il 23% delle immagini risulta *live* post-stampa ed è un valore anche superiore a quello ottenuto in digitale.

I risultati appena mostrati sono dovuti al rumore ed ai ritagli introdotti in fase di stampa; quindi, si può affermare senza problemi che sono risultati dati dalla casualità e non sono rilevanti. Però è naturale chiedersi di quanto gli attacchi migliorano il risultato già buono dato dalle *binarized* stampate e per questo in seguito viene effettuato un ulteriore studio al riguardo.

Attacco	Versione	Live/Tot	Acc%	MLS	Match	Match%
Clean	-	2/248	0,81%	90,07%	0/2	0,00%
Binarized Clean	-	57/248	22,98%	80,76%	57/57	100,00%
APGD	1	21/140	15,00%	85,27%	0/21	0,00%
	2	119/210	56,67%	84,65%	94/119	78,99%
	3	108/160	67,50%	82,03%	75/108	69,44%
	4	104/235	44,26%	85,31%	81/104	77,88%
DeepFool	1	44/98	44,90%	81,95%	0/44	0,00%
	2	113/147	76,87%	84,67%	81/113	71,68%
	3	96/146	65,75%	85,56%	65/96	67,71%
	4	137/220	62,27%	87,18%	104/137	75,91%

In verde sono riportati i valori più alti ottenuti tra gli attacchi e, come si può notare, riguardano tutti la versione quattro di DeepFool.

La versione uno invece, risulta la peggiore per entrambi gli attacchi ed ha restituito valori nulli per il match, questo perché, come già detto in precedenza, la prima versione del processo di stampa ha introdotto uno “zoom in” non necessario alle immagini e ciò porta il matcher a sbagliare.

Bisogna anche precisare che i risultati sono riportati in funzione del numero di impronte *live* digitali, ma in realtà per ogni impronta si hanno a

disposizione dieci acquisizioni; quindi, si è scelto di considerare un'impronta *live* se anche una sola sua acquisizione è risultata *live*, in questo modo l'attacco si può considerare realizzato con successo.

In ugual modo per i valori del match, si sono considerate solo le acquisizioni *live* e se una di esse, relative alla stessa impronta, supera il match, allora l'impronta viene considerata positivamente.

Ultima considerazione degna di nota riguarda la mancanza della versione cinque stampata; è stato scelto di trascurarla poiché durante lo svolgimento del progetto il processo di stampa è migliorato gradualmente andando a realizzare artefatti sempre meno rumorosi. Quindi, si è deciso di focalizzarsi su altri studi che sono mostrati in seguito, ma la versione cinque potrebbe essere utile per un lavoro futuro per cercare di superare la soglia di 137 immagini live posta da DeepFool versione 4.

Di seguito è riportata una tabella ricapitolativa delle impronte digitali (D) e stampate (P):

Attacco	Versione	Live/Tot	Acc%	MLS	Match	Match%
Clean	D	2/250	0,80%	96.11%	2/2	100,00%
	P	2/248	0,81%	90.07%	0/2	0,00%
Clean Bin	D	40/248	16,13%	87.97%	40/40	100,00%
	P	57/248	22,98%	80.76%	57/57	100,00%
APGD	1 D	140/248	56,45%	88.98%	56/140	40,00%
	1 P	21/140	15,00%	85.27%	0/21	0,00%
	2 D	210/248	84,68%	90.28%	165/210	78,57%
	2 P	119/210	56,67%	84.65%	94/119	78,99%
	3 D	160/248	64,52%	94.23%	130/160	81,25%
	3 P	108/160	67,50%	82.03%	75/108	69,44%
	4 D	235/248	94,76%	95.78%	199/235	84,68%
	4 P	104/235	44,26%	85.31%	81/104	77,88%
	5 D	228/248	91,94%	98.44%	123/228	53,95%
	1 D	98/248	39,52%	89.62%	87/98	90,82%
DeepFool	1 P	44/98	44,90%	81.95%	0/44	0,00%
	2 D	147/248	59,27%	91.64%	109/147	74,15%
	2 P	113/147	76,87%	84.67%	81/113	71,68%
	3 D	146/248	58,87%	92.48%	112/146	76,71%
	3 P	96/146	65,75%	85.56%	65/96	67,71%
	4 D	220/248	88,71%	94.49%	162/220	73,64%
	4 P	137/220	62,27%	87.18%	104/137	75,91%
	1 D	136/248	54,84%	86.83%	71/136	52,21%
	2 D	177/248	71,37%	91.11%	127/177	71,75%
	3 D	169/248	68,15%	92.78%	126/169	74,56%
IFGSM	4 D	224/248	90,32%	94.52%	192/224	85,71%

4.3.1 Studio relativo: Binarized vs Perturbed

Come già accennato, il risultato delle *binarized clean* è abbastanza significativo (23%) e si vuole capire di quanto le impronte stampate per i vari attacchi migliorano o peggiorano i risultati. In particolare, nella tabella sottostante sono riportati i precedenti valori di accuracy per i set stampati e, in aggiunta, vengono mostrate per quante immagini digitali relative all'attacco si ha già a disposizione un corrispettivo stampato *live* con le sole *binarized* (#Clean) e quindi si confronta di quanto aumenta o diminuisce l'accuracy rispetto alle *live* appena accennate ($\Delta L\%$).

Attacco	Versione	Live/Tot	Acc%	#Clean	ΔL%
APDG	1 P	21/140	15,00%	40/140	-13,57%
	2 P	119/210	56,67%	51/210	32,38%
	3 P	108/160	67,50%	46/160	38,75%
	4 P	104/235	44,26%	55/235	20,85%
DeepFool	1 P	44/98	44,90%	36/98	8,16%
	2 P	113/147	76,87%	44/147	47%
	3 P	96/146	65,75%	43/146	36,30%
	4 P	137/220	62,27%	54/220	37,73%

Si nota che solo per la versione uno di APGD si ha un calo di accuracy, mentre per tutte le altre si ha un aumento e ciò serve a capire che è stato utile stampare i vari attacchi e non utilizzare solo le *binarized*.

In seguito, è riportato un ulteriore studio al riguardo.

4.3.2 Studio incrementale

In questo ultimo paragrafo si vogliono riportare dei risultati complessivi di tutti i set di immagini stampate per cercare di massimizzare l'accuracy sul test set in lattice.

Nel dettaglio si controlla quante immagini *live*, in aggiunta alle *binarized* e quindi diverse da esse, ogni attacco ha fornito (#Live/Rel), per poi indicare la somma dei risultati (Live/Tot). Lo stesso ragionamento viene effettuato per i match e come ultima riga è presente una somma di tutti gli attacchi per massimizzare l'accuracy.

Attacco	Versione	#Liv/Rel	Live/Tot	L%	ΔL%	MLS
Clean Bin	P	-	57/248	22,98%	-	80.76%
	1 P	8/191	65/248	26,21%	14,04%	78.82%
	2 P	84/191	141/248	56,85%	147,37%	84.03%
	3 P	72/191	129/248	52,02%	126,32%	81.81%
	4 P	68/191	125/248	50,40%	119,30%	85.26%
APGD	1 P	18/191	75/248	30,24%	31,58%	79.61%
	2 P	78/191	135/248	54,44%	136,84%	85.21%
	3 P	67/191	124/248	50,00%	117,54%	84.68%
	4 P	98/191	155/248	62,50%	171,93%	85.40%
	All	Any	171/191	228/248	91,94%	300,00%
All		Any	171/191	228/248	91,94%	84.35%

Attacco	Versione	Match/Rel	Match%Rel	Match	Match%
Clean Bin	P	57/57	100,00%	-	-
	1 P	0/8	0,00%	57/248	22,98%
	2 P	61/84	72,62%	118/248	47,58%
	3 P	43/72	59,72%	100/248	40,32%
	4 P	47/68	69,12%	104/248	41,94%
APGD	1 P	0/18	0	57/248	22,98%
	2 P	51/78	65,38%	108/248	43,55%
	3 P	38/67	56,72%	95/248	38,31%
	4 P	68/98	69,39%	125/248	50,40%
	All	Any	140/171	81,87%	197/248
All		Any	140/171	81,87%	79,44%

In definitiva si può affermare che combinando tutte le impronte realizzate si raggiunge un'accuracy dell'attacco sul *liveness detector* pari al 91.94%; mentre per il *matcher* si ha un'accuracy dell'attacco pari al 79.44% sul totale di immagini a disposizione.

Ovviamente una futura stampa di immagini relative all'attacco IFGSM o alla versione cinque degli attacchi presentati non può far altro che aumentare l'accuracy totale.

Ultimo risultato importante è anche l'aumento del 300% dell'accuracy ottenuto combinando tutte le immagini rispetto alle sole *binarized*.

CONCLUSIONI

La metodologia proposta ha portato allo sviluppo di un processo di attacco ai *liveness detectors*, il quale non solo permette di realizzare immagini perturbate nel mondo digitale, ma ne facilita anche la stampa per poi assicurare una percentuale di successo dell'attacco nel mondo fisico abbastanza elevata. In questo modo è stato possibile evidenziare le vulnerabilità dei LD realizzati con CNN: non presentano vulnerabilità solo nel mondo digitale, ma queste possono essere sfruttate anche nel mondo fisico.

Tramite l'utilizzo dei tre attacchi forniti da ART è stato possibile mostrare come perturbare le immagini nel mondo digitale per fuorviare *convolutional neural network*; ma, in seguito, il tutto è stato finalizzato alla stampa nel mondo fisico, per questo sono state realizzate cinque versioni di uno stesso processo di attacco ed ogni versione ha analizzato un problema differente per poi risolverlo.

Nel complesso l'aspetto più importante riguardante il processo, e che ha portato all'ottenimento di ottimi risultati, è risultato l'*enhancement*, il quale ha permesso di facilitare la stampa e di ottenere artefatti puliti e coerenti con le impronte reali.

Inoltre, è stato dimostrato come gli attacchi proposti non alterino le caratteristiche principali delle impronte: le minuzie, e ciò ha permesso di effettuare test con successo riguardanti il *matcher*.

Infine, sono stati effettuati vari studi sui risultati ottenuti per poter estrarre il maggior numero di informazioni utili. Il più importante tra essi è lo studio che combina le varie tipologie di attacco con le immagini che hanno subito solo l'*enhancement*, in questo modo si cerca di utilizzare al massimo tutte le risorse a disposizione per assicurare un livello di successo finale elevato.

Sviluppi futuri

Sulla base dei ragionamenti che hanno condotto allo sviluppo della metodologia proposta e dei risultati ottenuti, si propongono una serie di idee per future attività, sviluppi e modifiche:

- Integrazione dei risultati relativi all'attacco IFGSM e alla versione cinque di APGD e DeepFool per studiarne il comportamento.
- Utilizzo di attacchi differenti presenti nella letteratura per verificare quale si adatta meglio al dominio delle impronte digitali; per esempio, l'attacco Carlini&Wagner, il quale rappresenta lo stato dell'arte per gli *adversarial attacks*.
- Realizzazione di una nuova versione di attacco che combini le principali caratteristiche delle versioni illustrate durante il progetto, come una combinazione di *random ROI* con dilatazione e rumore artificiale per il test.
- Utilizzo della metodologia proposta con altri *scanner* per verificarne la capacità di generalizzazione.
- Utilizzo degli artefatti realizzati per testare *liveness detectors* differenti e quindi realizzare un attacco cross-scanner.
- Individuazione di nuovi materiali per la stampa delle immagini perturbate.
- Realizzazione di LD migliori basandosi sui risultati ottenuti nel corso della tesi.

BIBLIOGRAFIA

- [1]A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097-1105. (s.d.).
- [2]Aleksander Madry, A. M. (s.d.). Towards Deep Learning Models Resistant to Adversarial.
- [3]Alex Krizhevsky, I. S. (s.d.). ImageNet Classification with Deep Convolutional Neural Networks.
- [4]Biometria, Enciclopedie on line, Istituto dell'Enciclopedia italiana Treccani. (s.d.).
- [5]C, M. S. (s.d.). On the Transferability of Adversarial Perturbation Attacks Against Fingerprint Based.
- [6]Craig I. Watson, M. D. (s.d.). User's Guide to NIST Biometric Image Software.
- [7]Enciclopedie on-line, I. d. (s.d.). Biometria.
- [8]Francesco Croce, M. H. (s.d.). Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, arXiv:2003.01690.

-
- [9] Goodfellow, I.J., Shlens, J., Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 . (s.d.).
 - [10] Hong, L., Wan, Y., and Jain, A. K. 'Fingerprint image enhancement: Algorithm and performance evaluation'. IEEE Transactions on Pattern Analysis and Machine Intelligence 20, 8 (1998), pp 777-789. (s.d.).
 - [11] Mattiello, D. (s.d.). Leveraging Adversarial Perturbations in Fingerprint Liveness Detection.
 - [12] Nogueira, R.F., de Alencar Lotufo, R., Machado, R.C., 2016. Fingerprint liveness detection using convolutional neural networks. IEEE transactions on information forensics and security 11, 1206-1213. (s.d.).
 - [13] Seyed-Mohsen Moosavi-Dezfooli, A. F. (s.d.). DeepFool: a simple and accurate method to fool deep neural networks.
 - [14] Standard ISO IEC 30107-1:2016. (s.d.).
 - [15] Stefano Marrone, C. S. (s.d.). Adversarial Perturbations Against.
 - [16] Stefano Marrone, R. C. (s.d.). Fingerprint Adversarial Presentation Attack in the Physical Domain.
 - [17] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., 2013. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 . (s.d.).
 - [18] Utkarsh-Deshmukh. (s.d.). Fingerprint-Enhancement-Python. Tratto da <https://github.com/Utkarsh-Deshmukh/Fingerprint-Enhancement-Python>
 - [19] Valerio Mura, L. G. (s.d.). LivDet 2015 Fingerprint Liveness Detection Competition 2015.
 - [20] Valerio Mura, L. G. (s.d.). LivDet 2015 Fingerprint Liveness Detection Competition 2015.

