

# Report - DeepFake

Andrea Aceto

Mat:0522501861

a.aceto6@studenti.unisa.it

Mario Cicalese

Mat:0522501799

m.cicalese21@studenti.unisa.it

Vincenzo Maria Arnone

Mat:0522501818

v.arnone4@studenti.unisa.it

## SOMMARIO

Il rapido progresso dei modelli di *AI Generativa*, in particolare dei *Diffusion Models (DM)*, delle *Generative Adversarial Networks (GAN)* e dei *Large Language Models (LLM)*, ha segnato una nuova era in cui i contenuti generati dall'IA sono sempre più integrati in vari aspetti della vita quotidiana. Da un lato, la generazione di immagini sintetiche è utile in numerosi campi e ha aperto nuove opportunità. Dall'altro, ha creato minacce riguardanti la privacy, l'autenticità e la sicurezza. Rilevare immagini sintetiche è di fondamentale importanza per prevenire attività illegali. Di conseguenza, rilevare i contenuti multimediali sintetici generati da modelli di *AI Generativa* è diventato cruciale, ed è su questo aspetto che si è concentrato questo report.

L'obiettivo di questo progetto è stato quello di definire una *Rete Neurale Siamese in grado*, tramite una *classificazione binaria* basata sulla Distanza Euclide tra i vettori *embedding* generati dalla rete, di predire se un'immagine è sintetica o reale. Il modello è stato allenato utilizzando immagini RGB 200x200 convertite nel dominio delle frequenze tramite la *Trasformata di Fourier*, appartenenti al dataset *Artifact* (<https://github.com/awsaf49/artifact>); un dataset che raccoglie 2.496.738 immagini, di cui 964.989 reali e 1.531.749 sintetiche. È stato considerato il dominio delle frequenze poiché ricerche precedenti hanno dimostrato che i modelli generativi lasciano *impronte artificiali* (definite come *fingerprint*) nelle immagini da loro generate, che possono essere sfruttate per rilevarle. Tali impronte, però, non sono visibili nel dominio spaziale (RGB), ma in quello delle frequenze.

Per allenare il modello sono stati utilizzati: *Triplet Loss Function* come funzione di perdita, *Adam* come ottimizzatore e, in alcuni casi, *l'Offline Triplet Mining* per filtrare le triplette prima degli allenamenti. Il progetto è stato suddiviso in esperimenti, in cui si sono utilizzate differenti tecniche, modelli, dati di training, ecc., con l'obiettivo di ottimizzare sempre di più le performance del modello. L'implementazione e il modello con le migliori performance sono disponibili nella seguente *repository GitHub*:

<https://github.com/MarioCicalese/SiameseNN-for-Fake-images-detection>

## 1 INTRODUZIONE

Nell'ambito dell'elaborazione delle immagini, diverse tecnologie e algoritmi svolgono un ruolo cruciale nell'identificazione e autenticazione di entità basate su caratteristiche fisiche e comportamentali.

Di fondamentale importanza è il concetto di *rete neurale*, un modello computazionale composto da nodi interconnessi chiamati neuroni. Questi neuroni sono organizzati in strati: un livello di input, uno o più livelli nascosti e un livello di output. Le reti neurali svolgono compiti specifici, come il riconoscimento di immagini o la previsione di dati, attraverso un processo di addestramento che coinvolge l'ottimizzazione dei pesi delle connessioni tra i neuroni. Sulla base del concetto di rete neurale vengono create le *siamese*

*neural network*, un'architettura di rete neurale composta da due (o più) sottoreti identiche che condividono gli stessi pesi e parametri. Queste reti parallele prendono due diversi input e producono vettori di output (*vettori di embedding*) che vengono confrontati utilizzando una funzione di distanza. Le reti siamesi, grazie alla loro capacità di riuscire a distinguere tra coppie di input simili e dissimili, sono comunemente utilizzate per compiti di verifica e riconoscimento, come la determinazione della similarità tra due immagini o impronte digitali.

Per quanto riguarda gli algoritmi di elaborazione delle immagini, è fondamentale introdurre il concetto di *Generative Adversarial Networks (GAN)* e di *Diffusion Model (DM)*.

I GAN[6] sono una classe di algoritmi artificiali che sfruttano due reti: un generatore, che crea immagini sintetiche, e un discriminatore, che cerca di distinguere dalle immagini reali. Le due reti vengono addestrate congiuntamente in un gioco di min-max: man mano che il discriminatore diventa più efficace, lo diventa anche il generatore, creando nel tempo campioni sempre più realistici.

I DM[6], invece, sono utili per creare dati sintetici che possono migliorare l'addestramento e la robustezza dei sistemi di riconoscimento biometrico. Essi sfruttano due processi stocastici interconnessi: un primo processo che trasforma le immagini reali in rumore casuale aggiungendo, a piccoli step, rumore gaussiano, un tipo di disturbo casuale che segue una distribuzione normale. I campioni generati in una singola fase del processo vengono poi utilizzati per addestrare una rete neurale che inverte tale fase, rimuovendo parte del rumore dal campione in ingresso. Una catena di reti di questo tipo esegue il secondo processo, ovvero il processo a ritroso, convertendo gradualmente il rumore gaussiano in immagini sintetiche.

Nel contesto di questi algoritmi, è utile specificare che si possono perseguitare due obiettivi leggermente diversi: *detection*[6], che fornisce un punteggio globale che valuta la probabilità che l'immagine presa in esame sia sintetica, e *attribution*[6], che mira a rintracciare il modello generativo specifico utilizzato per sintetizzare l'immagine.

Nell'elaborazione delle immagini è molto significativo come esse vengono trattate e trasformate, per questo è utile definire i concetti di *Trasformata di Fourier* e di **RGB (Red, Green, Blue)**. La trasformata di Fourier è una tecnica matematica utilizzata per decomporre un'immagine in una somma di sinusoidi di diverse frequenze e ampiezze. Questo permette di analizzare e manipolare le componenti di frequenza dell'immagine, utili per il filtraggio, la compressione e l'elaborazione delle immagini biometriche. RGB è uno spazio di colori utilizzato per rappresentare le immagini tramite la combinazione di tre componenti di colore: rosso (Red), verde (Green) e blu (Blue). Ogni colore, in un'immagine RGB, è determinato dall'intensità dei tre colori primari. Le immagini RGB sono comunemente utilizzate per la cattura e l'analisi di caratteristiche biometriche come volti, impronte digitali e iridi, poiché forniscono informazioni dettagliate sui colori e le texture necessarie per un riconoscimento accurato.

## 1.1 ArtiFact Dataset

La generazione di immagini sintetiche ha dato il via a nuove opportunità, ma ha anche creato minacce in termini di privacy, sicurezza e autenticità. L'individuazione di immagini false è di fondamentale importanza per prevenire le attività illegali, e varie ricerche hanno dimostrato che i vari modelli generativi rilasciano schemi unici sulle loro immagini sintetiche in modo da poterle individuare. Tuttavia, il problema della generalizzazione, ovvero la difficoltà nel generalizzare generatori mai visti durante la fase di addestramento, persiste. A fronte di tale problematica, e per valutare quindi la generalizzabilità e la robustezza dei rilevatori di immagini, è stato creato "ArtiFact" [4], un dataset, che comprende diversi generatori, categorie di oggetti e sfide del mondo reale. Tale set di dati propone una schema di classificazione multiclass che rileva in maniera efficiente le immagini sintetiche da generatori sia visti che non visti. Il dataset include una collezione diversificata di immagini reali da più categorie, tra cui volti umani, animali, luoghi, veicoli, arte e molti altri oggetti reali. Esso sintetizza immagini provenienti da 25 metodi distinti, in particolare, include 13 GAN, 7 Diffusion e 5 altri generatori vari. Il set di dati contiene un totale di 2.496.738 immagini, di cui 964.989 reali suddivise in questo modo:

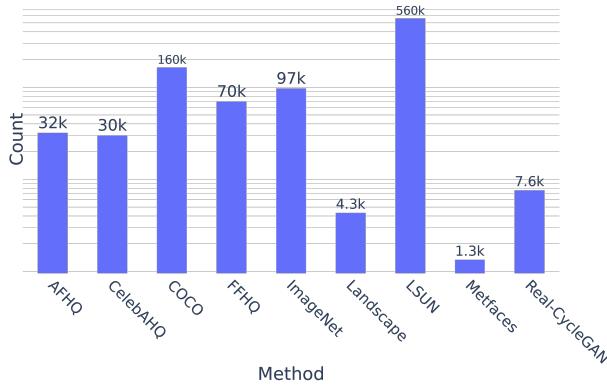


Figura 1: Distribuzione delle immagini reali nelle cartelle.

e 1.531.749 immagini false suddivise in questo modo:

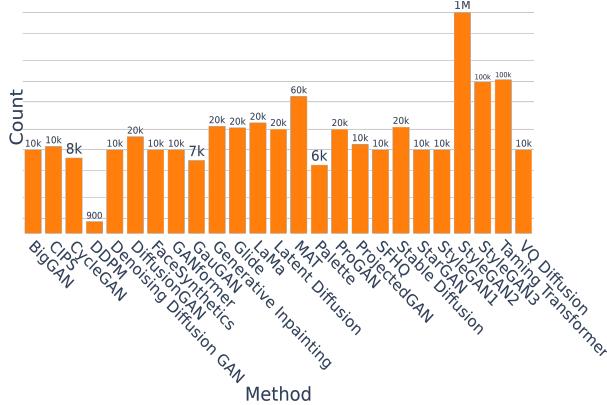


Figura 2: Distribuzione delle immagini fake nelle cartelle.

Le varie immagini sono suddivise in cartelle, ciascuna delle quali corrisponde a uno specifico generatore di immagini sintetiche o fonte di immagini reali. Ogni cartella contiene un file metadata.csv, che fornisce informazioni sulle immagini contenute nella cartella. Tale file presenta diverse colonne organizzate in questa maniera:

- filename: che indica il nome del file di immagine;
- image\_path: che rappresenta il percorso relativo al file di immagine;
- target: che rappresenta l'etichetta dell'immagine, e specifica se essa è un'immagine falsa, target 1 o maggiore, o un'immagine vera, target 0;
- category: che rappresenta la categoria dell'immagine (gatto, cane ecc.).

Inoltre per garantire una diversità significativa tra le diverse fonti, le immagini reali di ArtiFact sono campionate in modo casuale da set di dati di origine contenenti numerose categorie, e allo stesso tempo le immagini sintetiche sono generate dalle stesse categorie delle immagini reali. Il set di dati viene elaborato per riflettere gli scenari del mondo reale applicando random cropping, downscaling e JPEG compression.

ArtiFact è dunque utile come punto di riferimento per valutare le prestazioni dei rilevatori di immagini fake in condizioni reali.

## 1.2 Reti Neurali Siamesi

Nel mondo del deep learning le reti neurali, modello computazionale composto da nodi interconnessi chiamati neuroni, sono in grado di svolgere diversi compiti. Per ottenere prestazioni sempre più efficienti le reti neurali hanno bisogno di acquisire un gran numero di dati, ma per la realizzazione di alcuni problemi come quello del riconoscimento facciale o della verifica della firma, non possiamo fare affidamento su tale principio. Per tale motivazione vengono introdotte le reti neurali siamesi [1]. Queste altro non sono che delle architetture di rete neurale composte da due (o più) sottoreti identiche che condividono gli stessi pesi e parametri.

Nella maggior parte dei casi, viene addestrata solo una delle N (il numero di sottoreti scelte per risolvere il problema) e viene utilizzata la stessa configurazione (parametri e pesi) per le altre sottoreti [3].

Le reti di rete siamesi vengono utilizzate per trovare la somiglianza degli input confrontando i loro feature vectors.

Per spiegare nel dettaglio come funzionano tali reti immaginiamo di avere due immagini A e B, la prima sottorete prende un'immagine (A) come input e, attraverso strati convoluzionali e strati completamente connessi, ottieniamo una rappresentazione vettoriale dell'immagine. Successivamente in un'altra sottorete, che avrà la stessa configurazione della prima, verrà data in input la seconda immagine, ottenendone, anche in questo caso, la rappresentazione vettoriale. A questo punto avremo due codifiche E(A) e E(B) delle rispettive immagini, possiamo dunque confrontarle e, se le immagini sono simili allora anche le loro codifiche lo saranno. Se la distanza tra questi due vettori è piccola allora i vettori saranno simili se invece la distanza è grande i vettori sono diversi l'uno dall'altro [3].

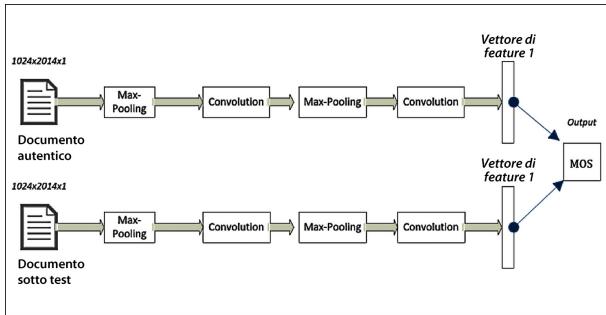


Figura 3: Esempio rete siamese.

Un aspetto importante da considerare quando si usano le reti siamesi sono le function loss. Esse si dividono in:

- *contrastive function*[3]: funzione che valuta semplicemente quanto bene la rete siamesa sia in grado di distinguere tra le coppie di immagini fornite. Tale valutazione viene effettuata mediante tale formula matematica:

$$(1 - Y) \frac{1}{2} (D_W)^2 + (Y) \frac{1}{2} \{ \max(0, m - D_W) \}^2$$

Dove **Dw** è definita come la distanza euclidea tra gli output delle varie sottoreti, **Y** assume valore 0 o 0, nel primo caso le immagini non appartengono alla stessa classe, mentre nel secondo caso si. **Max** è la funzione che denota il valore più alto tra **0** e **m-DW**, dove **m** è un valore di margine maggiore 0, se le coppie superano tale margine significa che non contribuiranno alla perdita;

- *Triplet loss function*[3]: tale funzione permette al modello di mappare due immagini simili una vicino all'altra e lontane da coppie di immagini dissimili. Questo approccio viene effettuato utilizzando la tripletta formata da: anchor, che rappresenta l'immagine campione, positive, che rappresenta l'immagine simile all'anchor, e negative, che rappresenta l'immagine dissimile. Inoltre per aumentare la distanza tra il vettore di output di coppie simili e dissimili e per mappare le immagini simili una vicina all'altra viene introdotto il margine, che, in questo caso, aumenta la separazione tra il vettore simile e dissimile e elimina anche l'output di qualsiasi soluzione ritenuta banale. In termini matematici tale funzione viene calcolata utilizzando la **distanza L2** e la **distanza coseno**. Considerando a come anchor, p come positive e n come negative tale funzione viene scritta in questo modo:

$$\text{loss}(a, p, n) = \max(0, d(a, p) - d(a, n) + \text{margin})$$

Le reti siamesi dunque presentano diversi vantaggi, di fatti sono necessarie poche classi d'immagini per effettuare una classificazione, inoltre esse si concentrano sull'apprendimento degli embeddings, che collocano le stesse classi vicine tra di loro, ciò permette di apprendere la semantic similarity. Tuttavia tali reti presentano anche degli svantaggi in quanto richiedono molto più tempo di addestramento rispetto alle reti normali e non generano delle probabilità ma dato che, come già specificato, l'addestramento prevede

l'apprendimento a coppie, esso restituirà la distanza da ciascuna classe[1].

## 2 OBIETTIVI DEL PROGETTO

I primi approcci di AI generativa potevano introdurre alcune incongruenze visive nelle immagini sintetiche, come le asimmetrie nelle ombre e nelle immagini riflesse. Ma, con la nascita delle tecnologie più recenti, come le reti generative avversarie (GAN) e i modelli di diffusione (DM), il realismo delle immagini sintetiche ha raggiunto un livello mai ottenuto in precedenza, lasciando delle impronte quasi invisibili. Inoltre, grazie alla recente diffusione degli *LLM*, ha reso la generazione di tali immagini sempre più semplice e flessibile. Infatti, anche utenti non esperti, sono in grado di generare o modificare immagini sintetiche a loro piacimento mediante del testo naturale dato in input a uno di questi modelli. Tuttavia, queste tecnologie potrebbero essere utilizzate per qualsiasi scopo illecito, come per esempio: campagne di disinformazione, propagande politiche, truffe, etc...

Di conseguenza, l'**obiettivo** di questo progetto sarà quello di **definire un modello di Machine Learning** (Rete Neurale Siamese) che **dovrà essere in grado**, date delle immagini in input, di **riconoscere** tramite una classificazione binaria, **le reali (Real)** e **le sintetiche (Fake)**. Per raggiungere tale obiettivo e allenare il nostro modello utilizzeremo il dataset "*ArtiFact*" [4], che contiene all'incirca 2.5 Milioni di immagini totali tra reali e sintetiche.

Durante la realizzazione del modello, verranno utilizzati **vari approcci e tecniche** divisi in esperimenti con l'obiettivo di **migliorare** sempre di più le **performance del modello**.

Tuttavia, l'**approccio generale** utilizzato alla base di tutti gli esperimenti condotti è l'utilizzo della trasformata di Fourier. Questo perché, ogni modello di AI generativa lascia all'interno delle immagini un sorta di "impronta artificiale", definita come **fingerprint** [6]. La presenza e la forma di tale fingerprint dipendono strettamente dal modello che ha generato l'immagine, più precisamente, esse possono dipendere da: architettura del modello, processo di creazione, e anche dal dataset di training. Tuttavia, **queste impronte non sono visibili nel dominio RGB**, di conseguenza, per un modello AI, è più difficile riconoscere un'immagine sintetica considerata nel dominio RGB. Invece, nel **dominio delle Frequenze**, questi artefatti **possono spesso essere individuati come forti picchi** negli spettri di potenza dei residui del rumore (vedi Figura 1, riga inferiore). Inoltre, è stato chiaramente dimostrato che i **generatori sintetici faticano a riprodurre perfettamente le distribuzioni spettrali** dei dati reali utilizzati per l'addestramento nelle frequenze medio-alte [6].

Di conseguenza, come approccio generale, utilizzeremo la trasformata di Fourier per trasformare le immagini di training (RGB) nel dominio delle Frequenze in modo che le fingerprint siano visibili e il modello possa utilizzarle per riconoscere un'immagine reale da una sintetica.

## 3 METODOLOGIA

In questa sezione verrà mostrata l'intera Pipeline di operazioni che sono state effettuate per creare il modello finale. Non tutti gli esperimenti seguiranno precisamente tale Pipeline, infatti, in alcuni degli esperimenti che verranno definiti successivamente

la Pipeline subirà alcune variazioni, ma di base, le operazioni che sono state impiegate sono quelle che verranno definite in questa sezione.

### 3.1 Preparazione dei dati

Come definito precedentemente nella Sezione 1.1, le immagini del dataset sono divise in cartelle, ognuna delle quali rappresenta un generatore differente. Per ciascuna cartella è definito un file .csv che specifica le informazioni, e soprattutto la tipologia, delle singole immagini. Tuttavia, questa struttura risulta essere scomoda quando si vuole lavorare su immagini di differenti cartelle, poiché bisognerebbe costantemente aprire cartelle e file .csv per accedere a queste ultime. Di conseguenza, per avere un accesso standard e più veloce, (1) sono stati definiti due file .csv generali, dove il primo contiene i metadati di tutte le immagini reali del dataset e il secondo quelli delle immagini sintetiche ("real.csv" e "fake.csv"). I due file .csv sono stati creati concatenando i file .csv delle singole cartelle, quindi la struttura non è cambiata. Tuttavia, durante la creazione dei due .csv abbiamo riscontrato due problemi/incosistenze presenti nel dataset. Infatti, nonostante nella documentazione del dataset viene riportato che l'attributo *target* assume due valori in base alla tipologia dell'immagine: **0 se è reale e 1 se è sintetica**. In realtà, ciò non viene rispettato, infatti, la variabile *target* assume dei valori compresi tra 0 e 6. Questo aspetto viene chiarito nelle issues della pagina *Github* del dataset<sup>1</sup>, dove uno dei creatori conferma la presenza di tale consistenza e afferma che tutte le immagini con un valore di **Target > 0** sono **sintetiche**. Di conseguenza, nel file .csv riferito alle immagini sintetiche ("fake.csv") (1.1) sono state inserite tutte le immagini con valore di **target > 0**. L'altra incosistenza invece, fa riferimento alla presenza di filename duplicati, infatti, ci sono più immagini di cartelle differenti che presentano lo stesso filename e ciò può portare a problemi nel momento in cui tali immagini verranno memorizzate nella stessa cartella, a seguito dell'applicazione dell'applicazione della Trasformata di Fourier su di esse. Quindi, per evitare problemi di duplicati e/o sovrascrizioni (1.2) abbiamo mappato il filename di tutte le immagini con il loro path (filename = image\_path per ogni immagine del dataset).

Tuttavia, vista la grandezza del dataset e il numero elevato dei differenti generatori, considerare tutte le immagini e/o tutti i generatori avrebbe richiesto una enorme disponibilità computazionale e temporale. Di conseguenza, (2) abbiamo definito una partizione che rappresentava un sottoinsieme dell'intero dataset, creando un file .csv contenente le informazioni solo delle immagini che facessero parte di tale partizione. La partizione può essere creata specificando in input il nome delle cartelle reali e sintetiche che contengono le immagini da includere, e successivamente verrà creato il file .csv ("dataset\_partition.csv"). NB: la partizione è logica, non fisica, infatti, la struttura, le immagini e le cartelle originali del dataset non vengono modificate.

Scelto il sottoinsieme di dataset da considerare per gli esperimenti, la fase successiva è stata (3) la creazione del Training e del Test Set. La partizione è stata effettuata utilizzando la classica divisione percentuale 80-20, dove il *Training Set* conteneva l'80% delle immagini della partizione, mentre il *Test Set* il restante 20%. Per

<sup>1</sup><https://github.com/awsa49/artifact/issues/4>

evitare problemi di **sbilanciamento**, entrambi gli insiemi sono stati creati in modo che la frequenza delle immagini reali e sintetiche fosse la stessa e, inoltre, che anche la frequenza delle immagini per cartella fosse uguale. Ad esempio, se il *Test Set/Training Set* deve contenere 2000 immagini e la partizione fa riferimento a *N* cartelle *Real* e *k* cartelle *Fake*, 1000 devono essere *Real* e 1000 *Fake*; però, ogni cartella *Real* deve avere una frequenza di *1000/N*, mentre ogni cartella *Fake* una frequenza di *1000/K*. In questo modo, tutto sarà **perfettamente bilanciato**.

Tuttavia, la struttura del *Training* e del *Test Set* è diversa; di conseguenza, il passo successivo è stato quello di (4) **definire la struttura del Training e del Test Set**. Avendo utilizzato come *loss function* la *Triplet loss function*, è necessario fornire in input al modello delle *triplette* tali che:

- Se l'**anchor** è un'immagine reale, allora anche la **Positive** deve essere reale, mentre la **Negative** deve essere sintetica (*Fake*);
- viceversa, se l'**anchor** è un'immagine sintetica, anche la **Positive** deve essere sintetica, mentre la **Negative** deve essere reale;
- **Anchor** e **Positive** devono essere differenti.

Il file .csv che fa riferimento al *Training Set* è quindi formato da tre colonne: anchor, positive e negative, dove ogni riga rappresenta una tripletta. Il file è stato definito in modo che ogni immagine del *Training Set* sia utilizzata una volta come *anchor*; quindi, il file .csv contiene un numero di righe uguale alla cardinalità dell'insieme di *Training*. Per ogni tripletta, le immagini *Positive* e *Negative* sono state definite scegliendo casualmente tra le immagini che rispettavano le condizioni definite nella lista precedente. Il file .csv che fa riferimento al *Test Set* invece, è formato da due colonne: la prima fa riferimento al percorso delle immagini reali, l'altra a quelle sintetiche. In questo modo possiamo utilizzare un unico file, essendo in grado di risalire alla natura delle immagini quando vengono calcolate le metriche di performance del modello.

### 3.2 Applicazione Della Trasformata di Fourier

Come specificato nella Sezione 2, i modelli di AI Generativa lasciano all'interno delle immagini sintetiche delle *impronte artificiali*, definite come *fingerprint*. Tuttavia, le *fingerprint* non sono visibili nel dominio RGB, mentre, nel dominio delle frequenze si manifestano come forti pichi negli spettri di potenza del rumore. Di conseguenza, nonostante le immagini del dataset considerato siano RGB, abbiamo deciso, come approccio generale, di trasformarle nel dominio delle frequenze utilizzando la Trasformata di Fourier in modo da ottimizzare i risultati del nostro modello.

NB: sono state convertite solo le immagini appartenenti al *Training* e *Test Set* (partizione) in quanto applicare Fourier richiede un'elevata quantità di risorse computazionali e di archiviazione. Per applicare la Trasformata di Fourier abbiamo eseguito i seguenti passaggi:

- (1) **Creazione della cartella Fourier e del metadata.csv:** abbiamo definito la cartella "fourier" dentro la quale verranno memorizzate le immagini convertite nel dominio delle frequenze e il *metadata.csv* per tenere traccia delle informazioni delle singole immagini;

- (2) **Conversione delle immagini in scala di grigio:** prima dell'applicazione di Fourier ogni immagine è stata convertita in scala di grigio. La trasformata di Fourier viene spesso applicata a immagini in scala di grigi piuttosto che a immagini RGB per diverse motivazioni: riduzione della complessità e della memoria necessaria. Inoltre la trasformata di Fourier di un'immagine in scala di grigi è più semplice da interpretare e visualizzare ;
- (3) **Applicazione di Fourier e archiviazione:** successivamente abbiamo applicato la Trasformata di Fourier sulle immagini in scala di grigio. L'immagine risultante è stata archiviata nella cartella *fourier* del passo 1 e le sue informazioni sono state memorizzate nel *metadata.csv*;
- (4) **Mapping tra Fourier e RGB:** Infine, abbiamo definito un nuovo file: *path\_mapping.csv*, che rappresenta una sorta di dizionario utilizzato per mappare il path RGB e di Fourier di una stessa immagine. Questo perché una volta applicato Fourier, una stessa immagine è memorizzata due volte nel dataset, quindi può essere utile definire un file nel quale si può velocemente recuperare il path RGB avendo quello di Fourier e viceversa.
- (5) **Cambiare il path delle immagini nel Training e Test Set:** Visto che il Training e il Test Set sono stati definiti basandosi sulle immagini originali, il path farà riferimento alle immagini RGB, ma, il modello deve ricevere in input le immagini nel dominio delle frequenze. Quindi, è stato cambiato il path RGB delle immagini nel Training e Test Set con il nuovo path di Fourier.

NB: quest'operazione va fatta solo se il Training e Test set siano stati definiti prima della trasformazione di Fourier, e quindi, se contengono path di immagini RGB.

Al termine di questi passaggi sono presenti due versioni della partizione iniziale all'interno del dataset: la prima memorizzata in RGB, mentre la seconda nel dominio delle Frequenze. In Figura 4 è possibile osservare un esempio di immagine reale visualizzata sia nel suo dominio d'origine (RGB) che nel dominio delle frequenze in seguito all'applicazione della trasformata di Fourier.



**Figura 4: Immagine reale visualizzata nel dominio delle frequenze**

In Figura 5 è possibile osservare un esempio di immagine sintetica prodotta da una AI generativa (*BigGAN*) visualizzata sia in RGB che nel dominio delle frequenze.



**Figura 5: Immagine sintetica visualizzata nel dominio delle frequenze**

Una volta convertite le immagini di Training e di Test nel dominio delle frequenze, tramite la trasformata di Fourier, abbiamo a disposizione tutti gli elementi per allenare e testare il modello (a meno che non si vogliano fare altre operazioni pre allenamento come fatto in alcuni esperimenti).

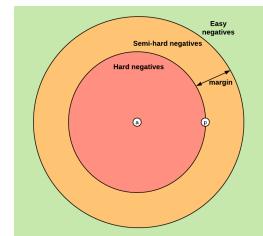
### 3.3 Offline Triplet Mining

In questa fase andiamo a definire su quali triplette verrà addestrato il modello, una fase cruciale visto l'impatto che ha sulle metriche finali.

Basandoci sulla definizione di *"Triplet Loss function"*, già definita nella Sezione 1.2 insieme ai concetti di tripletta e rappresentazione vettoriale (o *embedding*), possiamo distinguere tre categorie di triplette:

- **easy triplets:** triplette che hanno una *loss* pari a 0, perché  $d(a, p) + \text{margin} < d(a, n)$ ;
- **hard triplets:** triplette dove il *negative* è più vicino all'*anchor* rispetto al *positive*:  $d(a, n) < d(a, p)$ ;
- **semi-hard triplets:** triplette dove il *negative* non è più vicino all'*anchor* del *positive*, ma che presentano comunque una *positive loss*:  $d(a, p) < d(a, n) < d(a, p) + \text{margin}$ .

Le definizioni date dipendono sostanzialmente dalla posizione del negativo rispetto all'immagine di riferimento e al positivo nello spazio degli *embeddings*. Possiamo quindi estendere queste tre categorie ai negativi, individuando le tre regioni nello spazio degli *embeddings* mostrate nella figura seguente:



**Figura 6: I tre tipi di negativi, dati un *anchor* e un *positive*.**

Le ultime sono spesso quelle preferite, questo perché permettono di migliorare gradualmente il modello, evitando sia di concentrarsi su esempi troppo facili (che non forniscono molta informazione) sia di rendere l'allenamento troppo instabile con esempi troppo difficili.

Dopo aver capito che alcune triplette sono più utili di altre, l'attenzione si sposta ora sul come selezionarle, o estrarre ("mine"). L'***Offline Triplet Mining*** è la tecnica che consiste nel computare gli *embeddings* sul *training set*, e solo a quel punto selezionare le triplette in base alle condizioni dettate in precedenza [2].

Nel nostro caso è stata utilizzata una rete pre addestrata da libreria, la stessa che viene utilizzata per l'addestramento ma con il parametro "*pretrained*" impostato a *True*, per generare gli *embeddings*. Successivamente abbiamo selezionato solo le *semi-hard triplets*, calcolando le distanze tra gli *embeddings* con la formula della Distanza Euclidea e impostando un valore di margine uguale a 0.2.

Questa strategia, come vedremo in seguito, ci ha permesso di aumentare notevolmente le prestazioni, eliminando le informazioni fuorvianti che risiedevano nelle triplette non selezionate.

### 3.4 Addestramento

Per raggiungere gli obiettivi descritti nella Sezione 2, è stata utilizzata una SNN (*Siamese Neural Network*) composta da tre CNN (*Convolutional Neural Network*). In particolare il modello di CNN scelto è l'**EfficientNetV2\_B0**, il più "leggero" della famiglia EfficientNetV2 [5].

La CNN viene privata dell'ultimo *layer*, quello di classificazione, in modo da produrre in output gli *embeddings* (o *encoding vectors*). Un passaggio fondamentale, in quanto la *Loss Function*, in questo caso la **TripletMarginLoss** messa a disposizione da PyTorch, calcola le distanze tra le rappresentazioni vettoriali, come mostrato nella Sezione 1.2. L'*optimizer* scelto è, invece, **Adam**.

Per quanto riguarda gli iperparametri, troviamo:

- **Batch-Size**: il numero di triplette propagate contemporaneamente nella rete per ogni iterazione durante il processo di addestramento;
- **Learning Rate (LR)**: controlla quanto i pesi del modello vengono aggiornati in risposta al calcolo del gradiente;
- **Epochs**: il numero di volte in cui l'intero *training set* viene passato attraverso la SNN durante l'addestramento.

Per gestire eventuali plateau nella curva di apprendimento è stato anche introdotto un *Learning Rate Scheduler* che va a ridurre il LR di un fattore  $k$  ogni  $v$  epoch senza miglioramenti nella *loss*.

In preparazione a questa fase, dal *Training Set* viene estratto un sottoinsieme di triplette, che costituisce il *Validation Set*. Fatto ciò, vengono applicate delle trasformazioni alle immagini presenti nei due insiemi, imprescindibili per rispettare l'input atteso dal modello, e passate a due **DataLoader**, cruciale per la gestione e l'organizzazione dei dati in *batch*.

Durante la fase di addestramento vero e proprio, come si può vedere nella Fig. 7, la SNN apprende le similarità tra *anchor* e *positive* e le differenze tra *anchor* e *negative*. Per farlo, come abbiamo già visto, sfrutta gli *encoding vectors* generati in output dalle CNNs, calcolando le distanze secondo la formula della Distanza Euclidea. Il prodotto di questa fase è il modello addestrato.

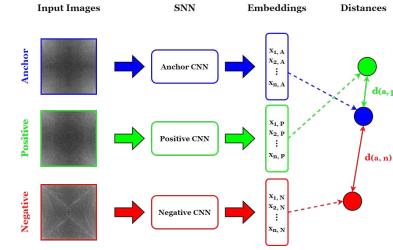


Figura 7: Fase di addestramento della SNN.

### 3.5 Testing e valutazione

La fase di Testing inizia sfruttando il modello addestrato per creare un Database di embeddings delle *anchors* presenti nel Training Set. Aiutandoci con la Fig. 8, osserviamo come il Testing faccia affidamento su questo Database per predire la classe di appartenenza delle immagini di test. Le immagini presenti nel *Test Set*, i cui

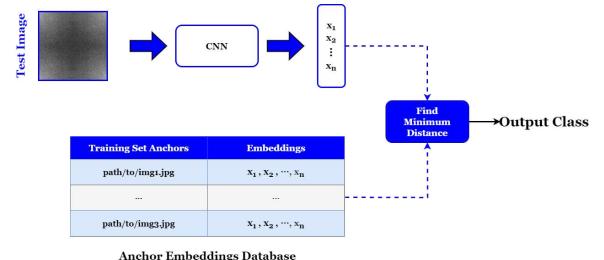


Figura 8: Fase di testing della SNN.

detttagli sono stati esposti nella Sezione 3.1, vengono, innanzitutto, passate attraverso il modello per ricavarne gli *embeddings*. Ogni *embedding* viene confrontato con quelli presenti nel Database per trovare quello più vicino, basandoci sempre sulla formula della Distanza Euclidea. A quel punto, la classe dell'immagine di test sarà la stessa dell'immagine *anchor* il cui relativo *embedding* è risultato il più vicino.

Per valutare il modello costruiamo la matrice di confusione con le predizioni fatte sulle immagini del Test Set, di cui conosciamo le classi reali.

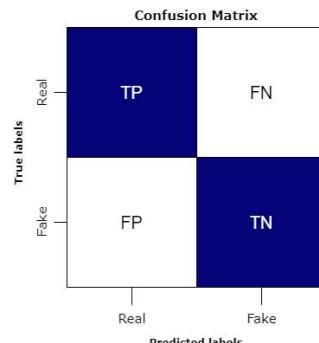


Figura 9: Matrice di confusione.

Sulla base di essa, e del fatto che i Veri Positivi sono riportati con la sigla TP, i Veri Negativi con TN, i Falsi Positivi con FP e i Falsi Negativi con FN, calcoliamo le seguenti metriche:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Esse forniscono una valutazione completa delle prestazioni di un modello di classificazione consentendo, al contempo, un confronto tra modelli diversi. In questo lavoro, vengono calcolati e utilizzati per il confronto i punteggi di *Accuracy*, *Precision*, *Recall*, *Specificity* e *F1*.

### 3.6 Evoluzione della Project Pipeline

Definiti i moduli che racchiudono le varie fasi di questo lavoro nei precedenti paragrafi di questa Sezione, passiamo ora a vedere come essi sono stati incastriati, modificati e adattati per andare a definire quella che è la *Project Pipeline*.

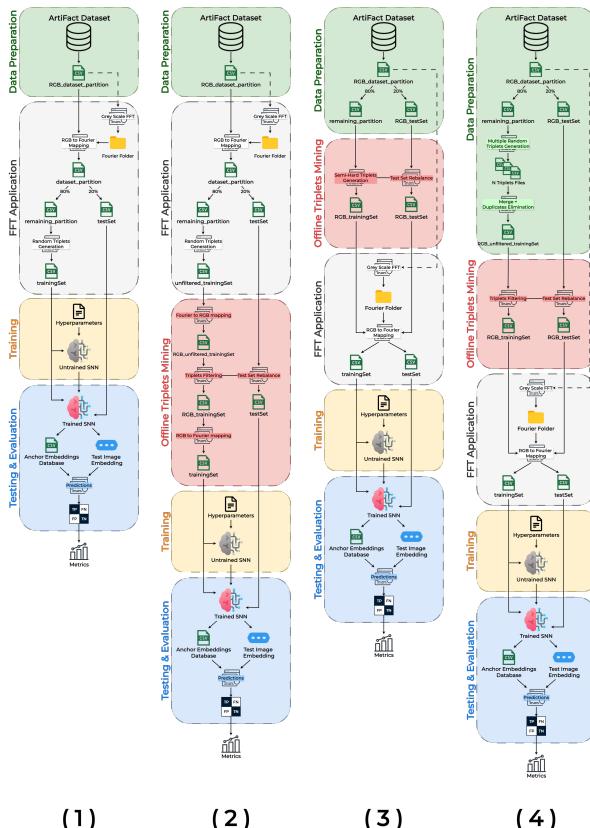


Figura 10: Evoluzione della Pipeline di Progetto.

Inizialmente progettata con l'approccio (1), la Fig. 10 mostra i cambiamenti che la Pipeline ha affrontato man mano che il lavoro andava avanti e il "cosa" fare e "come" farlo diventavano più chiari. Il primo approccio, abbastanza conoscitivo, non presentava, per esempio, la parte di *Offline Triplets Mining*, e ciò veniva rispecchiato nelle prestazioni del modello.

La Pipeline (2) ha visto l'introduzione del modulo mancante, con una struttura che, però, si presentava abbastanza confusionaria. Ciò era causato dal fatto che il nuovo modulo era stato innestato nel sistema senza alcun adattamento. Difatti uno dei principali contrasti visivi è proprio quello tra le Pipeline (2) e (3). In quest'ultima è stato deciso di rivedere l'iter delle operazioni per alleggerire il processo complessivo di preparazione alle fasi finali. Questa è anche l'unica versione della Pipeline che prevedeva un sistema alternativo di generazione delle triplette, sistema poi abbandonato perché giudicato inefficiente.

L'ultima Pipeline corrisponde a quella definitiva, un ibrido tra le due precedenti. Presenta la struttura ripulita della (3) e l'approccio alla generazione del *Training Set* della (2), seppur con delle modifiche che si sono rivelate vincenti nella corsa all'aumento delle performance.

## 4 ESPERIMENTI

### 4.1 Test #1

#### 4.1.1 Partizione del Dataset.

- Taglia: 24'000 immagini (12'000 real, 12'000 fake);
- Generatori:
  - Real: [Coco, Imagenet, LSun] (4'000 x generatore);
  - Fake: [BigGan, LatentDiffusion, TamingTransformer] (4'000 x generatore).

#### 4.1.2 Training Set e Test Set.

- **Training Set** ['anchor', 'positive', 'negative']: 19'200 triplette;
- **Test Set** ['real', 'fake']: 4'800 immagini (2'400 real, 2'400 fake).

In questo Test il Training Set è stato generato randomicamente, e non è stata applicata alcuna tecnica di Offline Mining.

Il rapporto tra i due Set, considerando per quello di addestramento le anchors uniche, è di **80:20**.

Di seguito sono riportate alcune statistiche utili sulla composizione del Training Set.

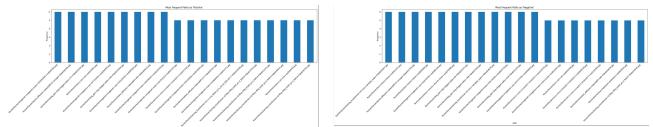


Figura 11: Path più frequenti come *Positive* (sx) e *Negative* (dx).

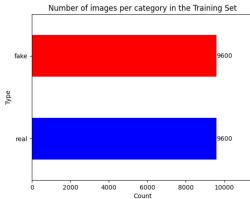


Figura 12: Natura delle Anchors.

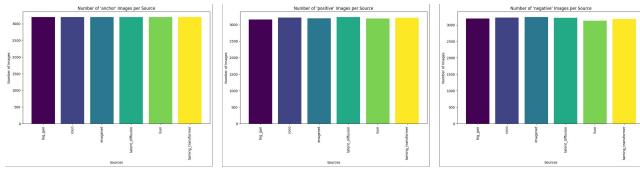


Figura 13: Generatore di appartenenza per Anchor, Positive e Negative.

#### 4.1.3 Addestramento.

- Iperparametri:
  - Batch-size:** 16;
  - Learning Rate:** 0.001;
  - Epoche:** 30.
- Miglior **Validation Loss:** 0.464.

#### 4.1.4 Performance.

Metriche	Performance Generali	BigGan	LatentDiffusion	TamingTransformer
Accuracy	74.58	86.19	68.56	71.31
Precision	74.52	80.06	71.81	73.52
Recall	74.71	96.38	61.12	66.62
Specificity	74.46	76.0	76.0	76.0
F1_score	74.61	87.47	66.04	69.9

**N.B.:** Le Metriche individuali sono relative solo ai generatori di immagini false, e sono state calcolate campionando un egual numero di immagini reali prese sempre dallo stesso generatore.

## 4.2 Test #2

### 4.2.1 Partizione del Dataset.

- Taglia: **60'000** immagini (**30'000 real, 30'000 fake**);
- Generatori:
  - Real: [Coco, Imagenet, LSUN] (10'000 x generatore);
  - Fake: [BigGan, LatentDiffusion, TamingTransformer] (10'000 x generatore).

### 4.2.2 Training Set e Test Set.

- Pre-Filtraggio:
  - Training Set** ['anchor', 'positive', 'negative']: 48'000 triplette;
  - Test Set** ['real', 'fake']: 12'000 immagini (**6'000 real, 6'000 fake**).

### • Post-Filtraggio

- Training Set** ['anchor', 'positive', 'negative']: 24'891 triplette;
- Test Set** ['real', 'fake']: 6'222 immagini (**3'111 real, 3'111 fake**).

In questo Test abbiamo utilizzato il modello EfficientNetV2\_B0, preaddestrato, per generare gli embeddings delle immagine contenute all'interno del TrainigSet, con lo scopo di filtrare solo le **Semi-Hard Triplets**. Avendo utilizzato un modello pretrainato su immagini RGB per generare gli embeddings di immagini che si trovavano nello spettro delle frequenze, le distanze calcolatrici tra le coppie di membri costituenti le triplets sono stati falsati. Di conseguenza i risultati non sono stati quelli attesi.

Il rapporto tra i due Set, considerando per quello di addestramento le anchors uniche, è di **80:20**. Per rispettare questo rapporto, dopo la fase di filtraggio è necessario operare un ribilanciamento del Test Set.

Di seguito sono riportate alcune statistiche utili sulla composizione del Training Set.

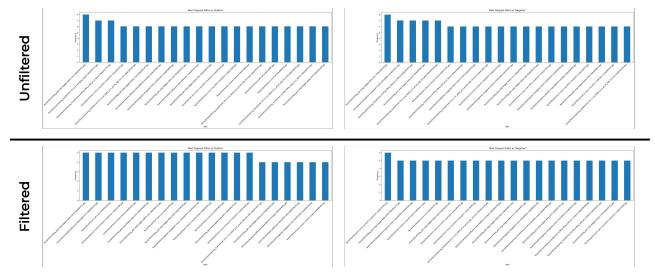


Figura 14: Path più frequenti come Positive (sx) e Negative (dx).

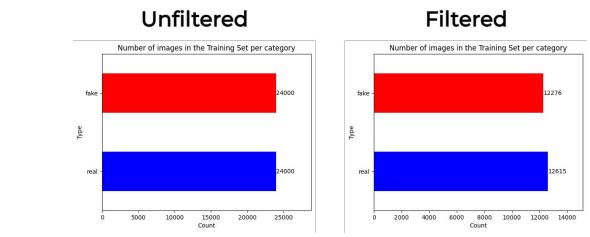


Figura 15: Natura delle Anchors.

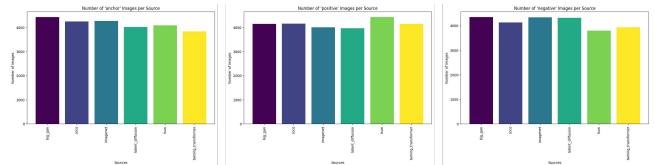


Figura 16: Generatore di appartenenza per Anchor, Positive e Negative.

#### 4.2.3 Addestramento.

- Iperparametri:
  - Batch-size:** 16;
  - Learning Rate:** 0.001;
  - Epoche:** 10
- Miglior **Validation Loss:** 0.249.

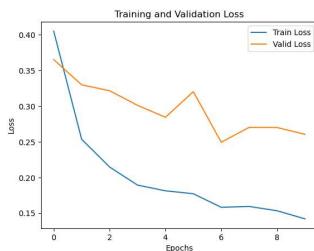


Figura 17: Curva di apprendimento

#### 4.2.4 Performance.

Metriche	Performance Generali	BigGan	LatentDiffusion	TamingTransformer
Accuracy	65.78	81.15	62.78	59.4
Precision	66.11	75.8	65.21	62.17
Recall	64.77	91.51	54.77	48.02
Specificity	66.7952	70.7811	70.7811	70.7811
F1_score	65.43	82.92	59.54	54.19

N.B.: Le Metriche individuali sono relative solo ai generatori di immagini false, e sono state calcolate campionando un egual numero di immagini reali prese sempre dallo stesso generatore.

### 4.3 Test #3

#### 4.3.1 Partizione del Dataset.

- Taglia: **60'000** immagini (**30'000 real, 30'000 fake**);
- Generatori:
  - Real: [Coco, Imagenet, LSUN] (10'000 x generatore);
  - Fake: [BigGan, LatentDiffusion, TamingTransformer] (10'000 x generatore).

#### 4.3.2 Training Set e Test Set.

- Pre-Filtraggio:
  - Training Set** ['anchor', 'positive', 'negative']: 48'000 triplette;
  - Test Set** ['real', 'fake']: 12'000 immagini (**6'000 real, 6'000 fake**).
- Post-Filtraggio
  - Training Set** ['anchor', 'positive', 'negative']: 6'594 triplette;
  - Test Set** ['real', 'fake']: 1'650 immagini (**825 real, 825 fake**).

In questo Test abbiamo utilizzato il modello EfficientNetV2\_B0, preaddestrato, per generare gli embeddings delle immagine contenute all'interno del TrainigSet, con lo scopo di filtrare solo le **Semi-Hard Triplets** (senza commettere lo stesso errore del Test precedente). Il rapporto tra i due Set, considerando per quello di addestramento le anchors uniche, è di **80:20**. Per rispettare questo rapporto, dopo la fase di filtraggio è necessario operare un ribilanciamento del Test Set.

Di seguito sono riportate alcune statistiche utili sulla composizione del Training Set.

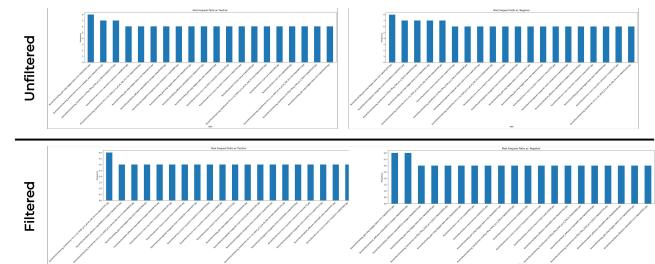


Figura 18: Path più frequenti come **Positive** (sx) e **Negative** (dx).

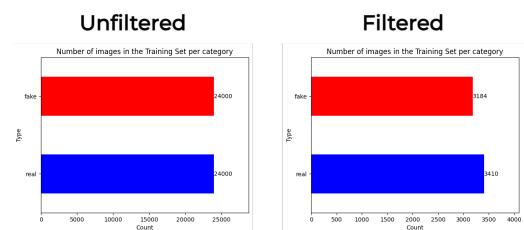
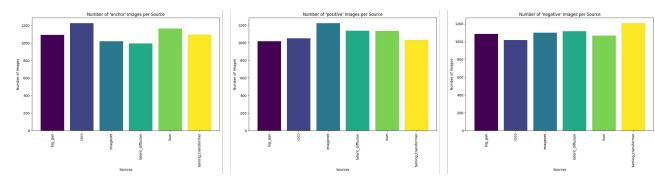


Figura 19: Natura delle Anchors.



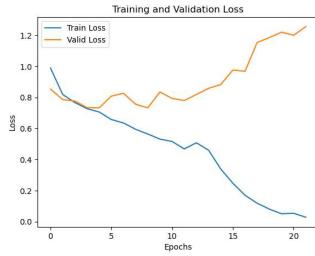


Figura 21: Curva di apprendimento

#### 4.3.4 Performance.

Metriche	Performance Generali	BigGan	LatentDiffusion	TamingTransformer
Accuracy	68.67	84.18	65.09	63.09
Precision	69.74	79.19	69.12	67.47
Recall	65.94	92.73	54.55	50.55
Specificity	71.3939	75.6364	75.6364	75.6364
F1_score	67.79	85.43	60.98	57.8

N.B.: Le Metriche individuali sono relative solo ai generatori di immagini false, e sono state calcolate campionando un egual numero di immagini reali prese sempre dallo stesso generatore.

### 4.4 Test #4

#### 4.4.1 Partizione del Dataset.

- Taglia: **60'000** immagini (*30'000 real, 30'000 fake*);
- Generatori:
  - Real: [Coco, Imagenet, LSun] (10'000 x generatore);
  - Fake: [BigGan, LatentDiffusion, TamingTransformer] (10'000 x generatore).

#### 4.4.2 Training Set e Test Set.

- **Training Set** ['anchor', 'positive', 'negative']: 21'445 triplete;
- **Test Set** ['real', 'fake']: 5'364 immagini (*2'682 real, 2'682 fake*).

In questo Test abbiamo sperimentato una tecnica differente rispetto sia ai precedenti che ai successivi. Piuttosto che filtrare le *Semi-Hard Triplets* generate randomicamente, abbiamo definito un algoritmo che le generasse apriori.

La complessità computazionale, i tempi di esecuzione lunghi e la difficoltà nell'ottenere un Training Set bilanciato ci hanno spinto ad abbandonare questa situazione.

Il rapporto tra i due Set, considerando per quello di addestramento le anchors uniche, è di **80:20**.

Di seguito sono riportate alcune statistiche utili sulla composizione del Training Set.

#### 4.4.3 Addestramento.

- Iperparametri:
  - **Batch-size:** 16;
  - **Learning Rate:** 0.001;
  - **Epoche:** 28

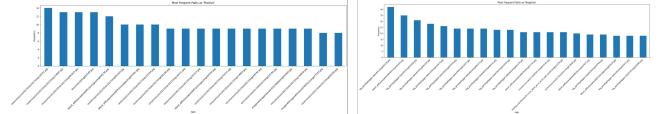


Figura 22: Path più frequenti come Positive (sx) e Negative (dx).

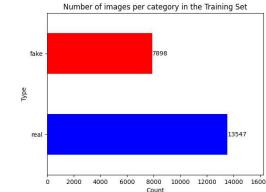


Figura 23: Natura delle Anchors.

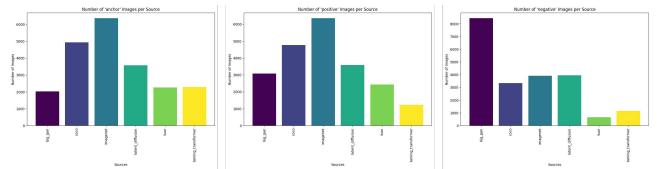


Figura 24: Generatore di appartenenza per Anchor, Positive e Negative.

- Miglior **Validation Loss:** 0.543.

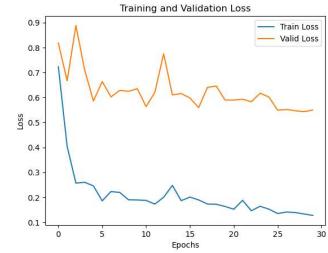


Figura 25: Curva di apprendimento

#### 4.4.4 Performance.

Metriche	Performance Generali	BigGan	LatentDiffusion	TamingTransformer
Accuracy	68.81	88.37	63.09	60.96
Precision	72.08	82.98	69.89	67.82
Recall	61.41	96.53	45.97	41.72
Specificity	76.2118	80.2013	80.2013	80.2013
F1_score	66.32	89.24	55.46	51.66

N.B.: Le Metriche individuali sono relative solo ai generatori di immagini false, e sono state calcolate campionando un egual numero di immagini reali prese sempre dallo stesso generatore.

## 4.5 Test #5

### 4.5.1 Partizione del Dataset.

- Taglia: **60'000** immagini (*30'000 real, 30'000 fake*);
- Generatori:
  - Real: [Coco, Imagenet, LSun] (10'000 x generatore);
  - Fake: [BigGan, LatentDiffusion, TamingTransformer] (10'000 x generatore).

### 4.5.2 Training Set e Test Set.

- Pre-Filtraggio:
  - **Training Set** ['anchor', 'positive', 'negative']: 144'000 triplette;
  - **Test Set** ['real', 'fake']: 12'000 immagini (*6'000 real, 6'000 fake*).
- Post-Filtraggio
  - **Training Set** ['anchor', 'positive', 'negative']: 19'600 triplette;
  - **Test Set** ['real', 'fake']: 4'904 immagini (*2'452 real, 2'452 fake*).

In questo Test abbiamo generato randomicamente molteplici file con struttura a triplette, per poi unirli (eliminando i duplicati). Successivamente abbiamo utilizzato il modello EfficientNetV2\_B0, pre-addestrato, per generare gli embeddings delle immagine contenute all'interno del TrainigSet, con lo scopo di filtrare solo le ***Semi-Hard Triplets***.

Il rapporto tra i due Set, considerando per quello di addestramento le anchors uniche, è di **80:20**. Per rispettare questo rapporto, dopo la fase di filtraggio è necessario operare un ribilanciamento del Test Set.

Di seguito sono riportate alcune statistiche utili sulla composizione del Training Set.

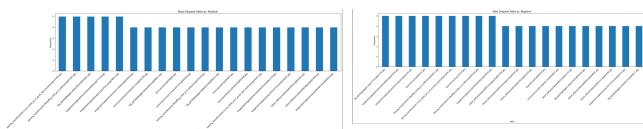


Figura 26: Path più frequenti come **Positive** (sx) e **Negative** (dx).

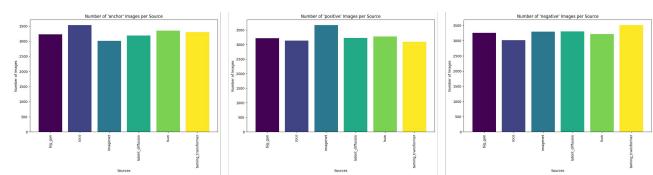


Figura 28: Generatore di appartenenza per **Anchor**, **Positive** e **Negative**.

### 4.5.3 Addestramento.

- Iperparametri:
  - **Batch-size**: 16;
  - **Learning Rate**: 0.001;
  - **Epoche**: 50
- Miglior **Validation Loss**: 0.579.



Figura 29: Curva di apprendimento

### 4.5.4 Performance.

Metriche	Performance Generali	BigGan	LatentDiffusion	TamingTransformer
Accuracy	75.62	87.82	69.65	74.17
Precision	75.94	82.39	74.43	77.02
Recall	74.99	96.21	59.85	68.91
Specificity	76.25	79.437	79.437	79.437
F1_score	75.47	88.77	66.35	72.74

**N.B.:** Le Metriche individuali sono relative solo ai generatori di immagini false, e sono state calcolate campionando un egual numero di immagini reali prese sempre dallo stesso generatore.

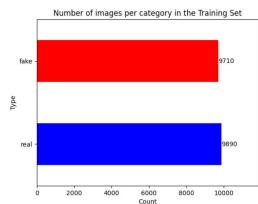


Figura 27: Natura delle Anchors.

## 5 ANALISI DEI RISULTATI

In questa sezione verrà effettuata l'analisi dei risultati di tutti gli esperimenti, evidenziando, in particolar modo, l'esperimento che ha ottenuto i migliori risultati e quali approcci hanno contribuito a migliorare e peggiorare le performance generali dei modelli.

Esperimenti	Accuracy	Precision	Recall	Specificity	F1_score
TEST#1	74.58	74.52	74.71	74.46	74.61
TEST#2	65.78	66.11	64.77	66.7952	65.43
TEST#3	68.67	69.74	65.94	71.3939	67.79
TEST#4	68.81	72.08	61.41	76.2118	66.32
TEST#5	75.62	75.94	74.99	76.25	75.47

Tabella 1: Confronto delle Performance dei 5 esperimenti

In Tabella 1 vengono visualizzate le Performance generali dei modelli definiti nei 5 Test, precedentemente descritti. Si può notare che il modello con le migliori performance è stato ottenuto dal Test 5. In tale esperimento piuttosto che limitarsi a definire un unico File di Training, contenente esattamente un sola volta, come anchor, ogni immagine, sono stati definiti **tre File randomici di Training**. Successivamente essi sono stati uniti in un unico grande File, andando poi a rimuovere le triplets duplicate. Così facendo, un'immagine non è più limitata ad apparire come anchor per una singola tripletta, ma può ripetersi in più istanze. Infine, il Training Set è stato filtrato in modo che contenesse solo *Triplets Semi-Hard*. L'efficienza di tale tecnica viene confermata anche confrontando le performance del Test 5 con quelle del Test 3. Infatti, i test sono praticamente uguali, con l'unica differenza che nel terzo è stato utilizzato un unico File di Training. Di conseguenza, **far apparire più volte una stessa immagine come anchor, sembrerebbe essere una tecnica che migliori le performance del modello**. Un altro aspetto che si evince, analizzando i risultati, è che le performance del Test 1 sono molto simili a quelle del test 5, nonostante gli approcci utilizzati siano diversi tra loro. Infatti, nel primo Test, non è stata applicata alcuna tecnica di filtraggio delle triplets e quindi i passaggi relativi all'utilizzo di un modello pre-trainato per calcolare i vettori di embedding e le loro distanze, al fine di filtrarli, non sono state applicate, cosa che invece nel Test 5, è stato effettuato. Di conseguenza, **le tecniche di filtraggio** applicate nell'ultimo esperimento(*semi-hard triplets*) portano a **un piccolo miglioramento ma a fronte di un costo computazionale ed a un numero di operazioni maggiori**; dunque, ciò dimostra che **utilizzare delle triplets completamente random**, senza alcun tipo di filtraggio, **produce delle Performance molto simili rispetto ad un Training Set filtrato** (considerando solo le *semi-hard triplets*). Il Test 2 è stato l'esperimento che ha prodotto il modello con le peggiori performance, ciò è dovuto all'utilizzo del modello pre-trainato su immagini RGB *EfficientNetV2\_B0* per generare gli embeddings delle triplets che però si trovavano nel dominio delle frequenze, e per tale motivazione le distanze risultano non corrette. Infatti, nel Test 3, sono state fornite in input al modello pre-trainato le immagini nel **dominio spaziale (RGB)**, **producendo delle performance migliori rispetto al secondo**. Infine, nel Test 4 è stato provato un approccio leggermente diverso rispetto al 3. Infatti, piuttosto che generare il file di Training randomicamente, e filtrarlo conservando solo le *semi-hard triplet*, è

stato creato in modo che contenesse già le *semi-hard triplet*, senza la necessità di filtrarlo. Tuttavia, le performance sono praticamente simili, nonostante la creazione del Training Set sia stata "guidata". Tale approccio non è stato utilizzato nel Test 5 in modo **da preservare la componente casuale, non influenzando direttamente la generazione dei dati di Training**.

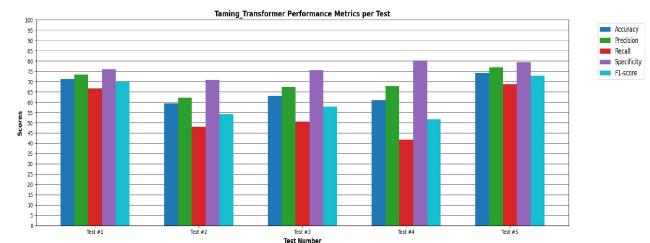


Figura 30: Performance dei 5 modelli per il generatore Taming Transformer

La Figura 30 mostra le performance dei 5 modelli sul generatore *Taming Transformer*. Osservandolo e analizzandolo è possibile notare che i risultati ottenuti dalle performance generali si confermano anche nei singoli generatori (in questo caso è stato utilizzato *Taming Transformer* come esempio).

## 6 CONCLUSIONI

In conclusione, il nostro progetto si è posto l'obiettivo di sviluppare una Rete Neurale Siamese, capace di distinguere tra immagini reali e sintetiche. Questo è stato reso possibile attraverso una classificazione binaria che utilizza la distanza euclidea tra gli embedding generati dalla rete. L'implementazione ha sfruttato il dataset ArtiFact[4], contenente quasi 2,5 milioni di immagini, e la Trasformata di Fourier per analizzare le immagini nel dominio delle frequenze, dove sono più evidenti le impronte artificiali lasciate dai modelli generativi. Nel corso del progetto, abbiamo condotto una serie di esperimenti per ottimizzare le performance del modello. Ogni esperimento è stato progettato per testare e migliorare specifici aspetti della rete neurale. L'ultimo esperimento, in particolare, ha rappresentato una svolta significativa. In questo test, abbiamo implementato tutte le ottimizzazioni precedenti, ma utilizzando più File di Training generati randomicamente e successivamente uniti e filtrati. Questo approccio ci ha permesso di affinare il modello in modo più efficace, migliorando la sua capacità di discriminare tra immagini reali e sintetiche. I risultati finali, come specificato nella precedente sezione, dimostrano che il nostro modello è in grado in modo abbastanza preciso di rilevare le immagini sintetiche, contribuendo così alla lotta contro l'uso illegittimo delle AI Generative. Di fondamentale importanza è, quindi, continuare a migliorare e adattare questi strumenti per rispondere alle crescenti sfide poste dall'evoluzione delle tecnologie generative.

## RIFERIMENTI BIBLIOGRAFICI

- [1] Sainath Krishnan. 2020. *A Friendly Introduction to Siamese Networks*. <https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942>
- [2] Olivier Moindrot. 2018. *Triplet Loss and Online Triplet Mining in TensorFlow*. <https://omoindrot.github.io/triplet-loss>
- [3] Rinki Nag. 2021. *A Comprehensive Guide to Siamese Neural Networks*. <https://medium.com/@rinkingnag24/a-comprehensive-guide-to-siamese-neural-networks-3358658c0513>
- [4] Md Awsafur Rahman, Bishnoy Paul, Najibul Haque Sarker, Zaber Ibn Abdul Hakim, and Shaikh Anowarul Fattah. 2023. Artifact: A Large-Scale Dataset With Artificial And Factual Images For Generalizable And Robust Synthetic Image Detection. In *2023 IEEE International Conference on Image Processing (ICIP)*. 2200–2204. <https://doi.org/10.1109/ICIP49359.2023.10222083>
- [5] Mingxing Tan and Quoc Le. 2021. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*. PMLR, 10096–10106.
- [6] Diangarti Tariang, Riccardo Corvi, Davide Cozzolino, Giovanni Poggi, Koki Nagano, and Luisa Verdoliva. 2024. Synthetic Image Verification in the Era of Generative Artificial Intelligence: What Works and What Isn't There yet. *IEEE Security & Privacy* (2024).