



UNIVERSITÀ DEGLI STUDI DI SALERNO  
DIPARTIMENTO DI INFORMATICA  
Software Engineering for AI

# Useless Deep Fake Detector

GIOVANNI DONISI  
FRANCESCO MADDALONI  
GIUSEPPE NAPOLITANO  
ANGELO NAZZARO

Anno Accademico 2024-2025



# Indice

<b>Indice</b>	<b>i</b>
<b>Elenco delle Figure</b>	<b>iv</b>
<b>Bibliografia</b>	<b>viii</b>
<b>1 Introduzione</b>	<b>1</b>
<b>2 Data Acquisition &amp; Understanding</b>	<b>3</b>
2.1 Dataset . . . . .	3
2.1.1 Subset utilizzato in questo lavoro . . . . .	3
2.2 Fairness Attraverso il Bilanciamento . . . . .	4
<b>3 Modeling</b>	<b>7</b>
3.1 Detector: Rilevazione di volti reali e sintetici . . . . .	7
3.2 Explainability: Full-GradCam . . . . .	7
3.3 Protector: Identificazione di attacchi di poisoning . . . . .	8
3.4 Il Modello UDFD . . . . .	10
<b>4 MLOps</b>	<b>11</b>
4.1 CI: Qualità del Codice . . . . .	11
4.2 Experiment & Data Tracking . . . . .	12
4.2.1 DVC (Data Version Control) . . . . .	12
4.2.2 Weights & Biases (W&B) . . . . .	12

4.3 AutoML . . . . .	15
<b>5 Sostenibilità in UDFD</b>	<b>17</b>
5.1 Efficienza del Modello e del Training . . . . .	17
5.2 Training intelligente e gestione delle risorse . . . . .	17
5.3 Ciclo di vita e riproducibilità . . . . .	18
<b>6 Deployment</b>	<b>19</b>
<b>Bibliografia</b>	<b>23</b>



# Elenco delle figure

F.2.1 Distribuzione del genere tra i volti. . . . .	4
F.2.2 Distribuzione congiunta di genere ed etnia nei volti reali. . . . .	5
F.2.3 Distribuzione congiunta di genere ed etnia nei volti fake. . . . .	6
F.2.4 Distribuzione congiunta di genere ed etnia nel dataset post-bilanciamento. Sono compresi sia i volti reali che sintetici. . . . .	6
F.3.1 Esempio Full-GradCAM. . . . .	8
F.3.2 Architettura del protector. . . . .	9
F.3.3 Architettura del modello UDFD. . . . .	10
F.4.1 Misure del Protector. . . . .	13
F.4.2 Prestazioni del protector. . . . .	13
F.4.3 Misure relative al Detector. . . . .	13
F.4.4 Prestazioni del detector. . . . .	14
F.4.5 Matrice di Confusione delle varie etnie. . . . .	14
F.6.1 Homepage web app. . . . .	20
F.6.2 Predizione web app. . . . .	20



# 1 | Introduzione

Useless Deep Fake Detector (**UDFD**) è un sistema progettato per sfruttare un modello di Deep Learning [Dim24] con l'obiettivo di distinguere immagini di volti reali (real) da quelle generate artificialmente (fake). Il progetto pone un'enfasi particolare sugli aspetti di ingegneria del software applicata all'intelligenza artificiale (IA), con l'intento di sviluppare un applicativo robusto, riproducibile e facilmente distribuibile. La repository del progetto è disponibile al seguente link: [Github](#).

Di seguito è riportata la pipeline di lavoro, strutturata secondo il **Team Data Science Process (TDSP)** [Mic24]:

- **Data Acquisition & Understanding:**

- *Raccolta dei dati*: reperimento del materiale utile all'addestramento del modello. L'obiettivo è quello di collezionare il maggior numero possibile di immagini real e fake. Nell'analisi preliminare verranno presi in considerazione formato, risoluzione e caratteristiche sensibili dei volti (es. etnia e genere), per valutare potenziali fonti di bias.
- *Preprocessing delle immagini*: rimozione dei duplicati, normalizzazione e ridimensionamento, al fine di ottenere un dataset coerente e ottimizzato per l'addestramento.

- **Modeling:**

- *Protector*: sviluppo e addestramento di un modello per l'individuazione di dati avvelenati.
- *Detector*: addestramento del modello sul dataset pre-processato, monitorando le metriche principali ed eventualmente intervenendo per mitigare fenomeni di overfitting, underfitting e, in particolare, bias predittivi nei confronti di specifici gruppi demografici.

- **Evaluation & Testing:**

- Valutazione delle predizioni mediante metriche sia quantitative (es. accuratezza, F1-score) sia qualitative, al fine di verificare la coerenza delle predizioni ed evidenziare eventuali bias predittivi.

- **Deployment:**

- Il modello addestrato verrà integrato in una *web application* interattiva, permettendo agli utenti di caricare immagini di volti e comprendere se si tratta di immagini reali o fake.
- Sarà inoltre incluso un modulo dedicato all’explainability del modello, tramite Full-GradCAM [SF19], con lo scopo di chiarire il perché di una determinata predizione .

L’applicativo sarà *containerizzato* attraverso l’utilizzo di Docker [Mer14].

Durante l’intero ciclo di vita del progetto verranno utilizzati i seguenti strumenti:

- **Weights and Biases (Wandb)** [Bie20]: impiegato per il monitoraggio degli esperimenti, la visualizzazione delle metriche di addestramento, la gestione degli iperparametri e il confronto tra modelli.
- **Continuous Integration (CI)**: strumenti di integrazione continua verranno utilizzati per il controllo della qualità del codice (linting, formattazione). In particolare sono stati utilizzati Black per la formattazione del codice e Autoflake per la rimozione di imports e variabili inutilizzate.
- **Continuous Deployment (CD)**: sarà implementata una pipeline di CD per consentire l’integrazione e la distribuzione automatica delle nuove versioni del modello e della webapp. Ogni modifica rilevante sarà testata, validata e pubblicata in ambiente di produzione in modo automatizzato.

Nel prosieguo del documento verranno analizzate nel dettaglio le varie fasi del progetto, con particolare attenzione agli aspetti di *fairness*, *explainability*, sicurezza e sostenibilità.

## 1.1 Ruoli e Contribuzioni

Di seguito sono riportati i ruoli e le contribuzioni di ogni membro del progetto.

Table T.1.1: Ruoli e contribuzioni.

Membro	Ruolo	Contribuzioni
Angelo Nazzaro	Project manager	Supervisione generale del progetto. Sviluppo, addestramento e valutazione del detector.
Francesco Maddaloni	Data engineer/Scientist	Individuazione, raccolta e analisi dei dati. Applicazione di tecniche di bilanciamento sui dati per la fairness.
Giuseppe Napolitano	Solution architect	Ricerca dell'architettura con la quale distribuire l'applicativo. Sviluppo del protector per il modulo di sicurezza.
Giovanni Donisi	Application Developer	Sviluppo, mantenimento e distribuzione dell'applicativo web.



# 2 | Data Acquisition & Understanding

## 2.1 Dataset

La raccolta dei dati si è concentrata sul dataset **ArtiFact** [Rah+23], un insieme di immagini su larga scala, progettato per includere una collezione eterogenea di immagini reali e sintetiche appartenenti a molteplici categorie: volti umani, animali, luoghi, veicoli, opere d'arte e altri oggetti del mondo reale. La raccolta si compone complessivamente di **2.496.738 immagini**, suddivise in **964.989 immagini reali** e **1.531.749 immagini false (sintetiche)**.

Per la generazione delle immagini sintetiche sono stati impiegati **25 metodi differenti**, tra cui **13 GANs**, **7 modelli basati su Diffusion** e **5 generatori di altra natura**.

### Statistiche principali

- **Numero totale di immagini:** 2.496.738
- **Immagini reali:** 964.989
- **Immagini fake:** 1.531.749
- **Metodi di generazione sintetica:** 25 (13 GANs, 7 Diffusion, 5 altri)
- **Sorgenti per immagini reali:** 8
- **Categorie:** volti umani, volti animali, luoghi, veicoli, arte, oggetti del mondo reale
- **Risoluzione delle immagini:**  $200 \times 200$  pixel

#### 2.1.1 Subset utilizzato in questo lavoro

Per i fini del presente progetto, è stato selezionato un sottoinsieme del dataset, focalizzato esclusivamente su immagini di volti umani:

- **CelebHQ:** 30.000 immagini reali di volti di celebrità.

- **FaceSync:** 10.000 immagini fake di volti generate artificialmente.

Questa scelta è stata guidata dalla necessità di addestrare e valutare modelli focalizzati sulla distinzione tra volti reali e generati. Al fine di analizzare e individuare caratteristiche sensibili, i volti sono stati classificati attraverso **DeepFace** [Ser23] che, per ogni immagine, ha estratto ed individuato feature relative all'etnia e al genere. Nello specifico, le immagini sono state classificate in:

- **Genere:**

- *Uomo*
- *Donna*

- **Etnia:**

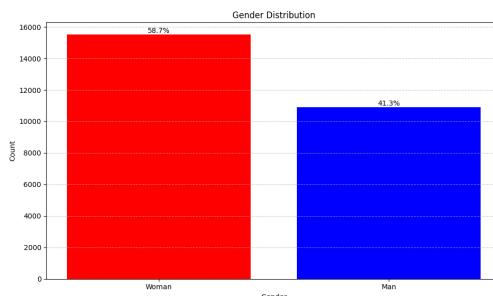
- **Bianco/Caucasica**
- **Asiatico**
- **Africana/Africano Americano/Nero**
- **Latino Ispanico**
- **Indiano**
- **Mediorientale**

Non approfondiremo lo studio su DeepFace in quanto non è stato il focus di questo lavoro.

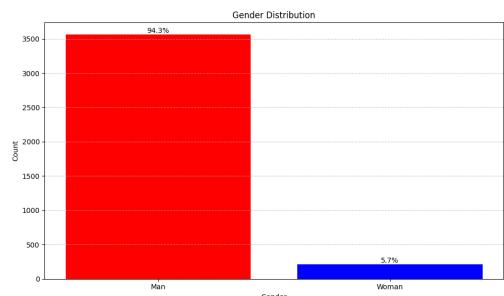
## 2.2 Fairness Attraverso il Bilanciamento

In seguito all'estrazione del genere e dell'etnia dei volti, è stata condotta un'analisi statistica sulla distribuzione dei dati, in maniera da prevenire/mitigare possibili bias predittivi a causa di sbilanciamenti.

Innanzitutto, è stato verificato il bilanciamento dei dati relativi alla presenza di uomini e donne; successivamente, è stato verificato il bilanciamento nei confronti delle varie etnie.



(a) Distribuzione del genere nei volti reali.



(b) Distribuzione del genere nei volti fake.

Figura F.2.1: Distribuzione del genere tra i volti.

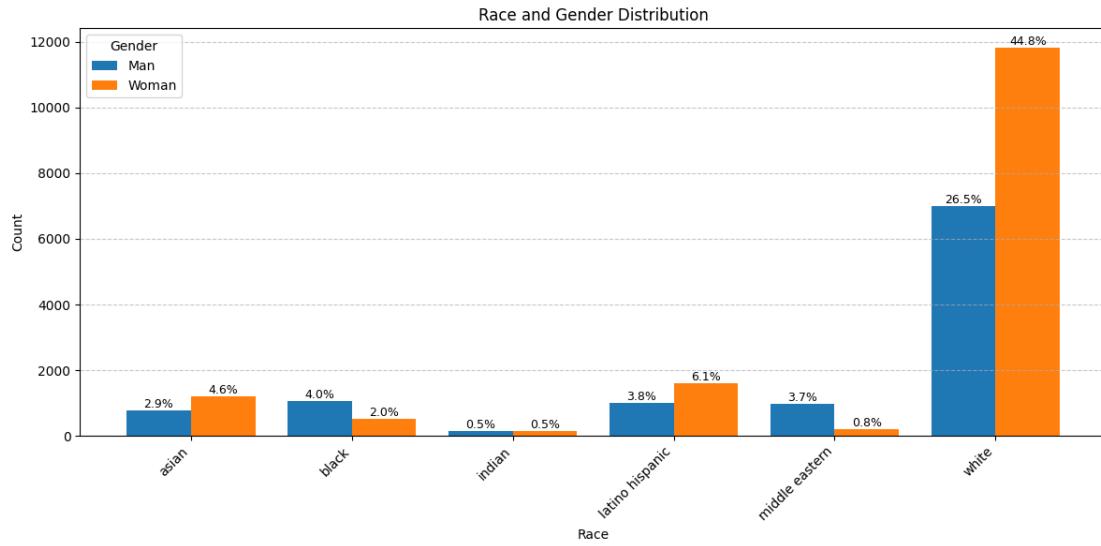


Figura F.2.2: Distribuzione congiunta di genere ed etnia nei volti reali.

Dai grafici riportati in Figura F.2.1, si osserva un leggero sbilanciamento a favore dei volti maschili nel dataset reale. Tale squilibrio risulta significativamente più accentuato nel dataset di immagini fake, evidenziando un bias introdotto dal generatore sintetico, che tende a prediligere la creazione di volti maschili.

Sulla base di questa prima analisi, si è deciso di approfondire ulteriormente la distribuzione delle caratteristiche demografiche, concentrandosi in particolare su etnia e sulla distribuzione congiunta di genere ed etnia. I risultati di tale analisi sono illustrati in Figura F.2.2 e Figura F.2.3, rispettivamente per i volti reali e per quelli sintetici.

Dall'analisi emerge una marcata sottorappresentazione delle etnie diverse da quella caucasica nei volti reali, che costituiscono circa il 71% del totale. Al contrario, la distribuzione per genere risulta più bilanciata all'interno di ciascuna etnia. La situazione si inverte nei volti sintetici: le etnie sono distribuite in maniera più uniforme, ma si registra una significativa predominanza di soggetti maschili.

Per mitigare possibili bias predittivi legati a genere ed etnia, entrambi i dataset sono stati sottoposti a un processo di bilanciamento. Le fasi adottate sono le seguenti:

- **Undersampling delle classi maggioritarie:** le classi più rappresentate sono state ridotte al fine di ottenere un bilanciamento uniforme tra etnie e generi.
- **Data augmentation delle classi minoritarie:** per le etnie sottorappresentate sono stati generati duplicati delle immagini originali, sui quali sono state applicate trasformazioni geometriche non distorsive: flip orizzontale, flip verticale e rotazioni multiple (tra  $0^\circ$  e  $270^\circ$ ). Queste tecniche sono state selezionate in modo da

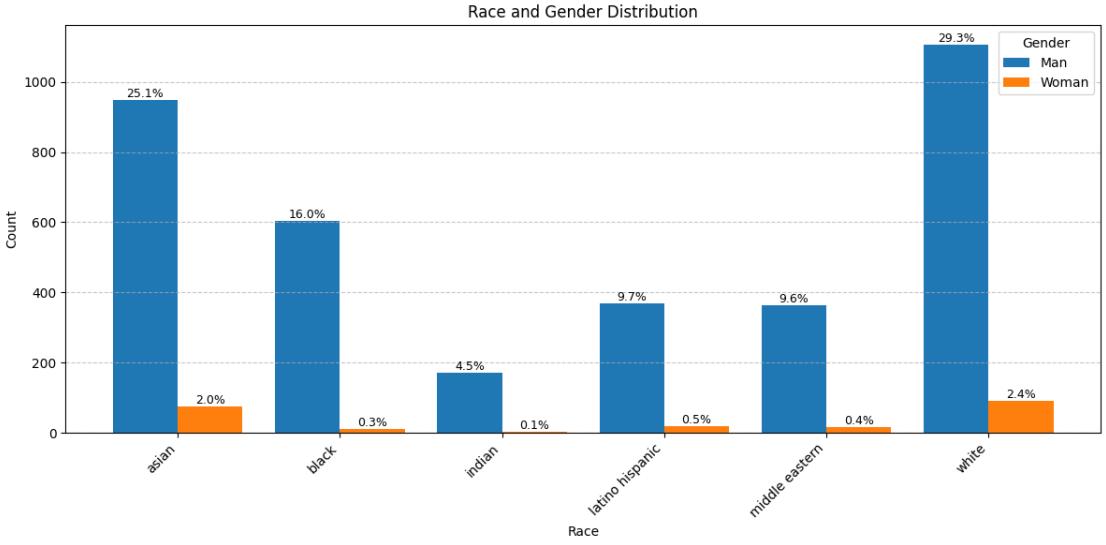


Figura F.2.3: Distribuzione congiunta di genere ed etnia nei volti fake.

preservare le caratteristiche semantiche del volto, evitando modifiche cromatiche o proporzionali che potessero introdurre ulteriori bias.

L’obiettivo finale è stato quello di portare ciascuna etnia a rappresentare almeno il 20% del dataset complessivo. Infine, per evitare una predominanza di immagini reali rispetto a quelle sintetiche (o viceversa), è stato eseguito un ulteriore *undersampling stratificato* tra le due classi, ottenendo un dataset finale composto da circa **4.000 immagini**, equamente suddivise tra volti reali e fake. In F.2.4 è mostrato il grafico del dataset comprendente sia volti reali che sintetici post-bilanciamento.

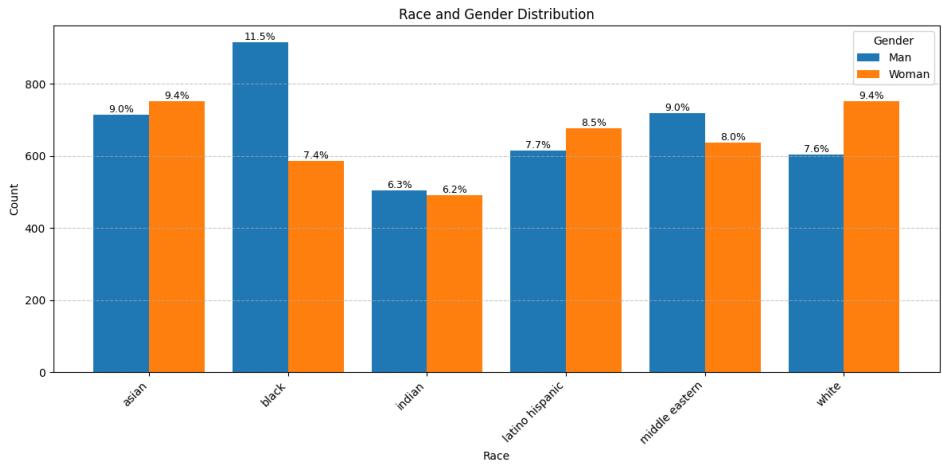


Figura F.2.4: Distribuzione congiunta di genere ed etnia nel dataset post-bilanciamento. Sono compresi sia i volti reali che sintetici.



# 3 | Modeling

## 3.1 Detector: Rilevazione di volti reali e sintetici

Il detector è stato implementato basandosi su un modello pre-addestrato chiamato **Vision Transformer**[Wu+20] (ViT), per maggiori informazioni prendere come riferimento il link alla documentazione.

## 3.2 Explainability: Full-GradCam

Al fine di migliorare l'explainability del modello utilizzato, è stato adottato il metodo **Gradient-weighted Class Activation Mapping** (**Grad-CAM**) [Sel+16]. Grad-CAM è una tecnica di visualizzazione che consente di interpretare in maniera qualitativa le decisioni di una rete neurale convoluzionale, evidenziando le regioni dell'immagine di input che hanno contribuito maggiormente alla predizione finale. La tecnica genera una **heat-map** che si sovrappone all'immagine originale, permettendo di identificare visivamente le aree di maggiore rilevanza per il modello. Nel caso specifico, è stato adottato Full-GradCAM [SF19], una variante di Grad-CAM che combina i gradienti da tutti i livelli della rete al fine di fornire una spiegazione quanto più comprensiva possibile.

Al fine di implementare il modulo di explainability è stato creato uno script python, sfruttando la libreria **pytorch-grad-cam** [Gc21]. Il processo di generazione si suddivide in diverse fasi:

- **Caricamento dell'immagine:** la funzione `prepare_image` gestisce il caricamento e la preparazione dell'immagine fornita in input. La funzione restituisce in output l'immagine nei formati necessari per le elaborazioni successive: **PIL** per eventuali operazioni di visualizzazione, e come array **NumPy** normalizzato in RGB per la sovrapposizione della heatmap finale.
- **Inizializzazione del modello e del processor:** il modello viene caricato tramite la funzione `load_model`, che recupera un modello pre-addestrato da Hugging Face o carica il modello da un checkpoint esistente. In parallelo, viene instanziato anche il **processor**, necessario per la corretta normalizzazione dell'immagine in input.

- **Selezione del layer target:** per poter applicare Grad-CAM, è necessario indicare un layer interno del modello da cui estrarre le attivazioni. Nel caso di architetture basate su **Vision Transformer (ViT)** [Dos20], è stato selezionato l'ultimo layer di normalizzazione all'interno del blocco **encoder**.
- **Applicazione del metodo di spiegazione:** attualmente è supportato il metodo **FullGrad**, una versione più sofisticata di Grad-CAM che tiene in considerazione sia le attivazioni, sia i gradienti del bias. Il modello è avvolto in una classe **HuggingfaceToTensorWrapper** al fine di renderlo compatibile con la libreria **pytorch-grad-cam**. L'immagine preprocessata viene poi passata al metodo, che restituisce una mappa di attivazione in scala di grigi.
- **Generazione e salvataggio:** la mappa risultante viene sovrapposta all'immagine originale passata in input, producendo una heatmap colorata tramite la funzione **show\_cam\_on\_image**. Il risultato viene salvato in un file JPEG, corredata da timestamp, all'interno della directory specificata.

Il modulo è progettato per essere estendibile anche ad altri metodi di explainability, basati su Grad-CAM e supportati da pytorch-grad-cam. In F.3.1 è mostrato un esempio di funzionamento di Full-GradCAM.

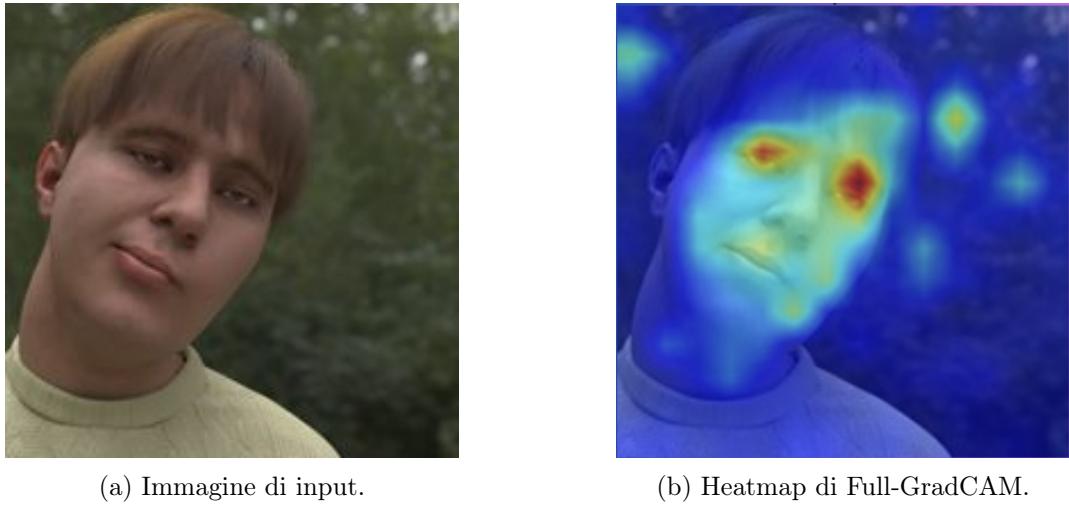


Figura F.3.1: Esempio Full-GradCAM.

### 3.3 Protector: Identificazione di attacchi di poisoning

Il **data poisoning** è una categoria di attacchi nel campo del machine learning in cui un agente malevolo compromette l'integrità ed il corretto funzionamento del modello, manipolando il dataset di addestramento. L'obiettivo principale degli attaccanti è quello di influenzare il comportamento del modello durante la fase di inferenza, inducendolo a fare previsioni errate, compromettendo l'**accuracy** o introducendo delle vulnerabilità

Nel contesto del nostro applicativo, è prevista una fase di raccolta di feedback sulle predizioni del modello durante la quale il sistema chiede all'utente se è d'accordo o meno sulle predizioni su immagini caricate manualmente. Questa fase è necessaria per rilevare potenziali prediction drifts. Per proteggere il modello da utenti disattenti o intenzionalmente malevoli, è stato sviluppato un modulo di sicurezza denominato **protector**, il cui scopo è rilevare e prevenire l'avvelenamento delle etichette durante questa fase interattiva.

Il **protector** è un modello composto da tre Multi-Layer Perceptron (MLP), ciascuno con un compito specifico:

- **Low-level MLP**: elabora le feature di basso livello proveniente dal primo hidden layer del detector;
- **High-level MLP**: elabora le feature di alto livello proveniente dall'ultimo hidden layer dal detector;
- **MLP Classificatore**: combina le rappresentazioni apprese dai due MLP precedenti per effettuare la predizione finale sull'etichetta: *avvelenata* o *non avvelenata*.

Il principio alla base è che, durante l'elaborazione di un'immagine, il detector attiva specifici “percorsi neuronali” e genera rappresentazioni discriminanti per quella tipologia di input, indipendentemente dalla corrispondenza o meno con l'etichetta assegnata dall'utente. Il **protector** sfrutta tali rappresentazioni per imparare a distinguere tra immagini reali e fake [Cut22]. In Figura F.3.2 è riportata l'architettura complessiva del modulo.

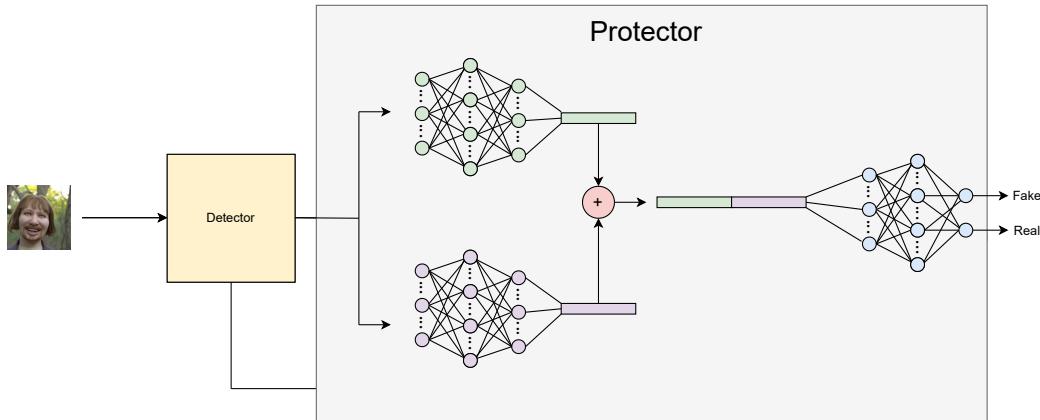


Figure F.3.2: Architettura del protector.

L'addestramento del protector avviene successivamente a quello del detector, utilizzando la stessa base dati impiegata per l'addestramento del modello principale.

### 3.4 Il Modello UDFD

Il modello **UDFD** è quindi costituito da due modelli che operano in cascata: il detector e il protector. Il detector riceve in input un'immagine, dalla quale estrae vettori di feature di basso e alto livello e fornisce una predizione binaria: *reale* o *falso*. Successivamente, il protector utilizza i vettori di feature prodotti dal detector come input per effettuare una seconda predizione sull'autenticità dell'immagine. Se la predizione del protector coincide con quella dell'utente, l'immagine viene caricata sul server per essere usata come futuro dato di addestramento con l'etichetta eventualmente corretta; altrimenti, viene segnalata come "sospetta" e viene ignorata.

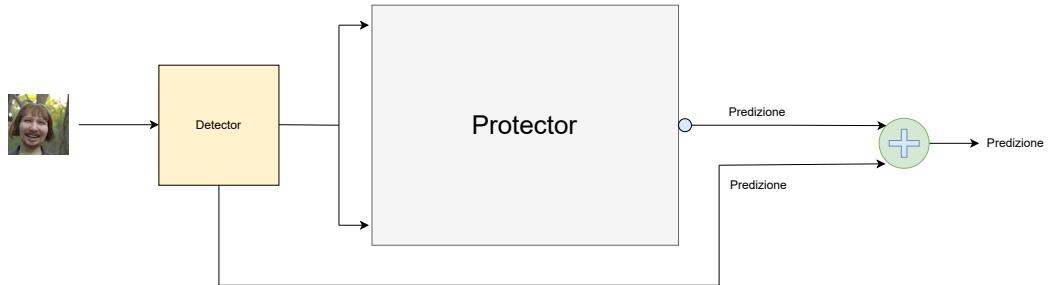


Figura F.3.3: Architettura del modello UDFD.

La Figura F.3.3 mostra una panoramica architetturale del sistema UDFD.



# 4 | MLOps

È stata progettata e sviluppata una pipeline completa per l'addestramento, il monitoraggio e la distribuzione del modello, in linea con le best practice del livello 2 di maturità MLOps. Questo livello prevede la gestione automatizzata di esperimenti, dati, versionamento e rilascio continuo, con particolare attenzione alla riproducibilità e alla scalabilità del processo.

La pipeline è suddivisa nei seguenti componenti principali:

- **CI (Continuous Integration)**: Controllo automatico della qualità del codice attraverso strumenti di linting e formattazione.
- **CD (Continuous Delivery)**: Automazione del processo di integrazione e rilascio di nuove versioni del modello e della web app.
- **Experiment & Data Tracking**: Tracciamento delle metriche di addestramento e versioning del dataset.
- **AutoML**: Retraining automatico del modello in risposta a cali di performance, in seguito ad una fase di activate learning basata sul feedback dell'utenza.

## 4.1 CI: Qualità del Codice

Per garantire una base di codice pulita, coerente e manutenibile, sono stati integrati strumenti di formattazione automatica e rimozione del codice inutilizzato.

- **Black** [LB25] è un formattatore di codice Python che analizza il codice sorgente e ne modifica la *forma*, rispettando una serie di regole prestabilite. Black, ad esempio, gestisce:
  - **Indentazione**: quattro spazi per livello (PEP8).
  - **Riformattazione delle espressioni**.
  - **Rimozione di spazi ridondanti**.
- **Autoflake** [PyC25] effettua la "pulizia" del codice, rimuovendo automaticamente codice non utilizzato, ad esempio:

- Import non referenziati.
- Variabili inutilizzate.
- Assegnazioni ridondanti.

## 4.2 Experiment & Data Tracking

### 4.2.1 DVC (Data Version Control)

DVC (Data Version Control) [Kup+21; BEA21] è stato adottato per la gestione e il versionamento dei dataset e dei modelli, grazie alla sua integrazione con Git senza appesantire la repository. Le principali funzionalità implementate sono:

- **Versionamento dei dati:** dataset e modelli vengono versionati tramite file `.dvc`, collegati a commit Git, con stoccaggio su remote storage esterni per evitare il sovraccarico del repository.
- **Definizione delle pipeline:** ogni fase del workflow (preprocessing, addestramento, ecc.) è descritta nel file `dvc.yaml`, garantendo tracciabilità e modularità.

### 4.2.2 Weights & Biases (W&B)

Weights & Biases [Bie20] è una piattaforma SaaS che consente il monitoraggio e la gestione interattiva degli esperimenti di Machine Learning. Le funzionalità chiave sfruttate nella pipeline includono:

- **Logging** in tempo reale delle metriche (e.g., loss, accuracy), degli iperparametri, delle immagini e delle curve di apprendimento.
- **Dashboard interattiva:** visualizzazione aggregata dei risultati e confronto tra esperimenti.
- **Gestione avanzata degli esperimenti:** consente di filtrare, raggruppare e analizzare esperimenti sulla base di parametri e performance.

Questa infrastruttura è stata essenziale per monitorare le prestazioni dei modelli *Protector* e *Detector*. Le relative metriche, curve e matrici di confusione sono riportate nelle Figure F.4.1–F.4.5.

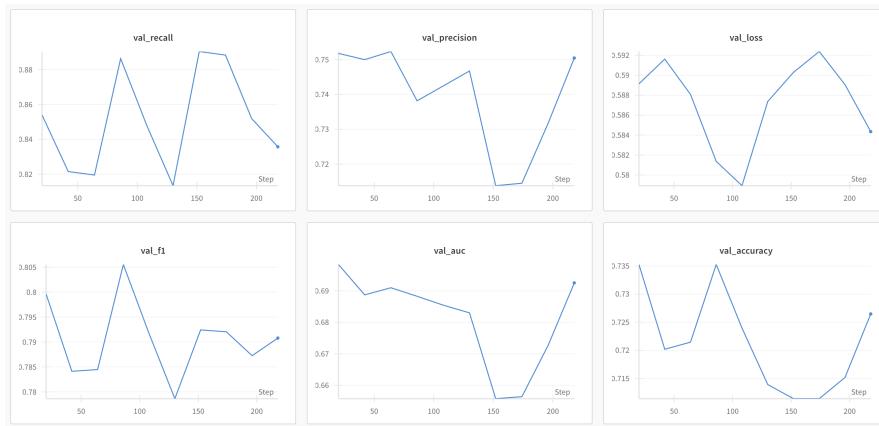


Figura F.4.1: Misure del Protector.

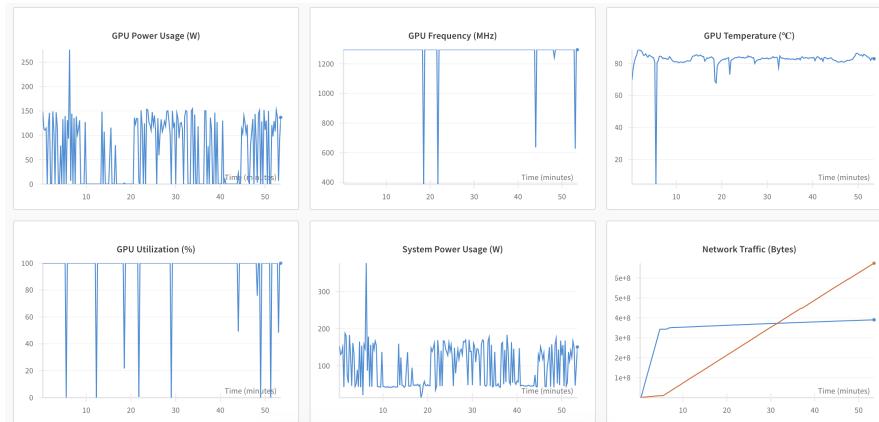


Figura F.4.2: Prestazioni del protector.

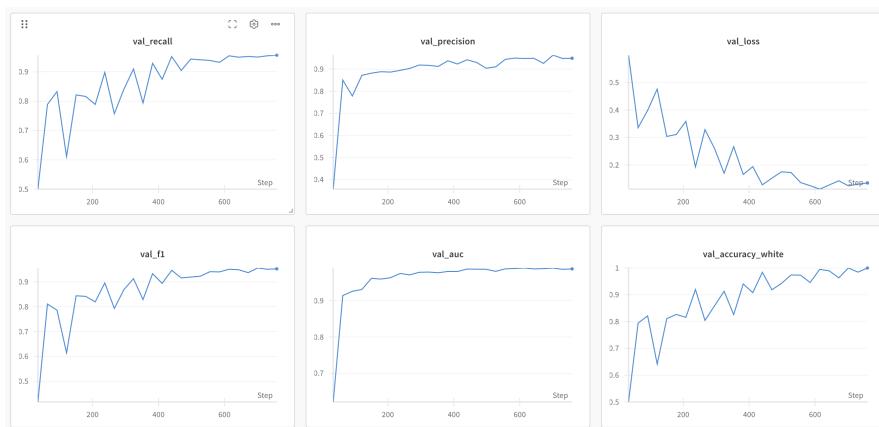


Figura F.4.3: Misure relative al Detector.

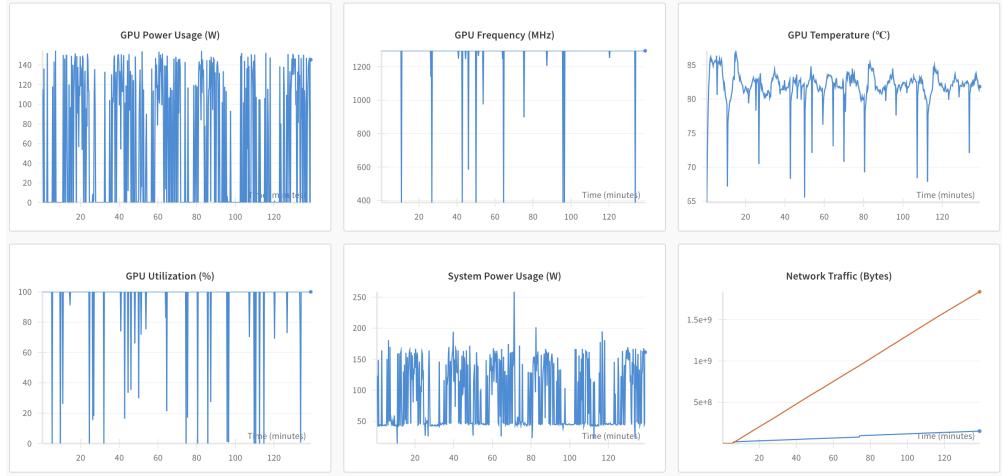


Figura F.4.4: Prestazioni del detector.

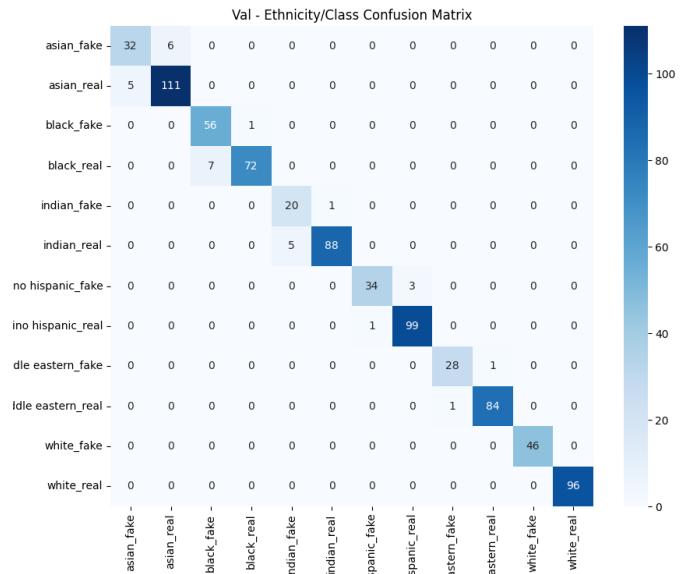


Figura F.4.5: Matrice di Confusione delle varie etnie.

### 4.3 AutoML

Al fine di garantire la capacità del modello di adattarsi a nuovi dati e prevenire il degrado delle prestazioni nel tempo, è stato implementato un modulo di **AutoML**, per l'addestramento automatico del modello.

Il sistema si basa su una strategia che monitora continuamente l'accuratezza del modello, confrontando le prestazioni attuali con la baseline salvata al termine dell'addestramento. Le fasi principali sono:

- **Memorizzazione della baseline:** al termine dell'addestramento iniziale, l'accuratezza del modello viene salvata come metrica di riferimento.
- **Raccolta del feedback utente:** durante l'interazione con la web app, gli utenti possono fornire un feedback esplicito sull'accuratezza delle predizioni ricevute.
- **Controllo delle prestazioni:** se l'accuratezza percepita dagli utenti si discosta dalla baseline di una certa soglia, attualmente impostata a 10 punti percentuali, il sistema rileva un potenziale degrado delle prestazioni.
- **Riaddestramento automatico:** quando viene rilevato un potenziale declino delle prestazioni, il modello viene riaddestrato automaticamente utilizzando il dataset originale arricchito con le nuove immagini caricate dagli utenti e annotate implicitamente tramite il loro feedback.



# 5 | Sostenibilità in UDFD

Durante la pianificazione del lavoro sono state prese scelte oculate per garantire la sostenibilità dell'applicativo sotto diversi aspetti, che verranno approfonditi nelle sezioni seguenti.

## 5.1 Efficienza del Modello e del Training

- **Architettura semplificata del modello:** Per il protector è stata proposta un'architettura semplice, basata su tre MLP. Questa scelta è stata dettata dalla necessità di ridurre i tempi di addestramento e i costi computazionali, pur mantenendo prestazioni adeguate rispetto alla complessità del detector.
- **Librerie Ottimizzate:** Sono state impiegate librerie altamente ottimizzate e testate per lo sviluppo e addestramento di modelli di Deep Learning, quali: PyTorch [Pas+17], PyTorch Lightning [FT19], HuggingFace [Wol+20], al fine di migliorare prestazioni ed efficienza.
- **Transfer Learning:** È stato effettuato il fine-tuning di un modello pre-addestrato. Ciò ha ridotto significativamente i tempi di training e l'energia richiesta rispetto alla costruzione e all'addestramento di un modello da zero.

## 5.2 Training intelligente e gestione delle risorse

- **Early Stopping:** Per evitare possibili sprechi di risorse quando quando il modello raggiunge un plateau nelle prestazioni o inizia a peggiorare, è stato adottato l'Early Stopping che interrompe automaticamente l'addestramento quando non si osservano miglioramenti nel valore della metrica di riferimento.
- **Checkpoint durante il training:** Per prenviare possibili situazioni disastore (es. l'interruzione dell'addestramento) e per facilitare un eventuale ripresa e/o miglioramento del modello in un secondo momento, sono stati salvati a intervalli regolari dei checkpoint del modello. È stata mantenuta solo la versione con le prestazioni migliori, riducendo il consumo richiesto di memoria.

- **Accelerazione Hardware:** Per ridurre ulteriormente il consumo energetico e i tempi di addestramento, gli addestramenti sono avvenuti su Apple Silicon, tramite Metal Performance Shaders (MPS) [App25], un backend ottimizzato per sfruttare al meglio le caratteristiche hardware delle GPU.

### 5.3 Ciclo di vita e riproducibilità

- **Retraining Automatizzato:** Nella pipeline è prevista una strategia di AutoML, che consente di aggiornare il modello con nuovi dati per contrastare il concept drift.
- **Riproducibilità Garantita:** Per garantire la riproducibilità dei nostri esperimenti e consentire operazioni di debugging, sono stati integrati nella pipeline W&B e DVC per gestire e tracciare le configurazioni degli esperimenti e il versioning del dataset. Inoltre, è stato impostato il seme di generazione a 42.



# 6 | Deployment

Come anticipato, il modello è reso disponibile tramite una web app interattiva, che consente agli utenti di caricare immagini di volti e ricevere una predizione sulla loro autenticità (reale o sintetica (fake)).

La web app include un form per l'upload delle immagini, composto da un input per il caricamento del file e da una checkbox opzionale che permette di attivare il modulo di explainability, qualora l'utente desideri ottenere una spiegazione della predizione. Dopo il caricamento, viene mostrata la probabilità associata a ciascuna delle due classi (reale vs. fake), fornendo una misura della confidenza del modello. Un ulteriore aspetto rilevante, come descritto in 3.4, è il coinvolgimento attivo dell'utente nel miglioramento del modello: viene infatti offerta la possibilità di fornire un feedback sulla predizione ricevuta, contribuendo così alla raccolta di dati utili per il fine-tuning futuro del modello. Per maggiori dettagli, riferirsi al Capitolo 3. Le Figure F.6.1 e F.6.2 mostrano l'interfaccia della web app.

Per garantire astrazione, automazione e riproducibilità, l'applicazione è distribuita sotto forma di container Docker [Mer14].

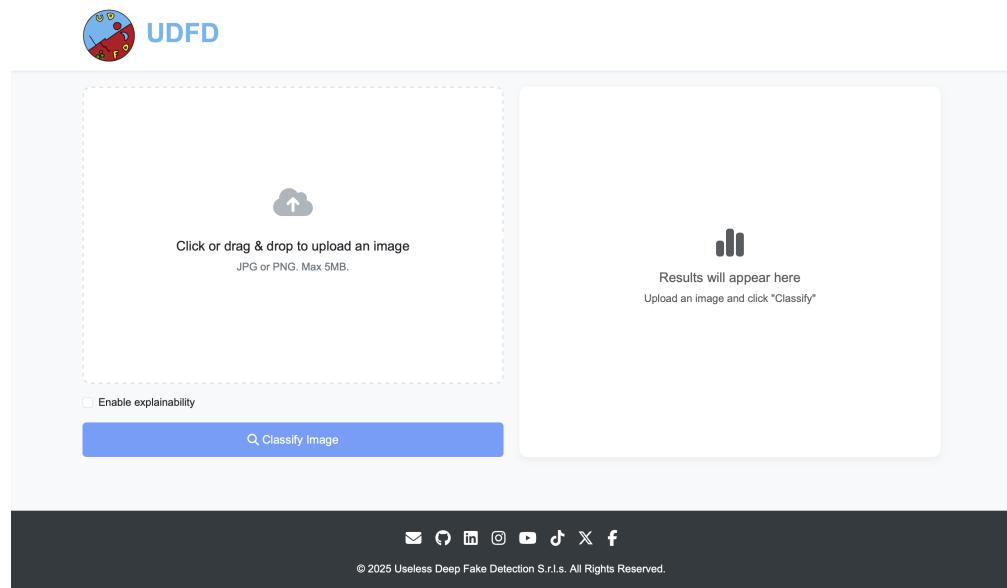


Figura F.6.1: Homepage web app.

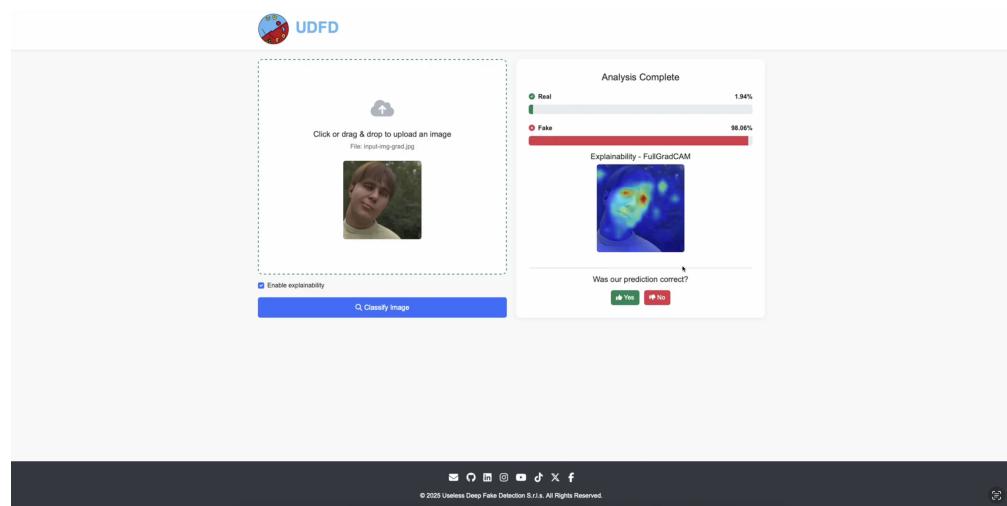


Figura F.6.2: Predizione web app.



# Bibliografia

- [App25] Apple Inc. *Metal Performance Shaders*. Retrieved July 15, 2025 from Apple Developer Documentation. Apple Inc. 2025. URL: <https://developer.apple.com/documentation/metalperformanceshaders>.
- [BEA21] Amine Barak, Ellis E. Egahn e Bram Adams. «On the Co-evolution of ML Pipelines and Source Code - Empirical Study of DVC Projects». In: *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2021, pp. 422–433. DOI: [10.1109/SANER50967.2021.00046](https://doi.org/10.1109/SANER50967.2021.00046).
- [Bie20] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: <https://www.wandb.com/>.
- [Cut22] Maya Cutkosky. «Defending deepfake detector against data poisoning attacks». In: *Mospace* (2022). URL: <https://mospace.umsystem.edu/xmlui/handle/10355/95196>.
- [Dim24] Dima. *deepfake\_vs\_real\_image\_detection*. 2024. URL: [https://huggingface.co/dima806/deepfake\\_vs\\_real\\_image\\_detection](https://huggingface.co/dima806/deepfake_vs_real_image_detection).
- [Dos20] Alexey Dosovitskiy. «An image is worth 16x16 words: Transformers for image recognition at scale». In: *arXiv preprint arXiv:2010.11929* (2020).
- [FT19] William Falcon e The PyTorch Lightning team. *PyTorch Lightning*. Ver. 1.4. Mar. 2019. DOI: [10.5281/zenodo.3828935](https://doi.org/10.5281/zenodo.3828935). URL: <https://github.com/Lightning-AI/lightning>.
- [Gc21] Jacob Gildenblat e contributors. *PyTorch library for CAM methods*. <https://github.com/jacobgil/pytorch-grad-cam>. 2021.
- [Kup+21] Ruslan Kuprieiev et al. *DVC: Data Version Control – Git for Data & Models*. Ver. 2.0.5. 2021. DOI: [10.5281/zenodo.4593044](https://doi.org/10.5281/zenodo.4593044). URL: <https://doi.org/10.5281/zenodo.4593044>.
- [LB25] Łukasz Langa e contributors to Black. *Black: The uncompromising Python code formatter*. <https://github.com/psf/black>. Software. 2025. URL: <https://black.readthedocs.io/en/stable/>.
- [Mer14] Dirk Merkel. «Docker: lightweight linux containers for consistent development and deployment». In: *Linux journal* 2014.239 (2014), p. 2.

- [Mic24] Microsoft Azure Architecture Center. *Team Data Science Process (TDSP)*. Web page. <https://learn.microsoft.com/it-it/azure/architecture/data-science-process/overview>. 2024. (Visitato il giorno 24/06/2025).
- [Pas+17] Adam Paszke et al. «Automatic differentiation in PyTorch». In: *NIPS-W*. 2017.
- [PyC25] PyCQA. *autoflake: Removes unused imports and unused variables*. <https://github.com/PyCQA/autoflake>. Accessed: 2025-07-15. 2025.
- [Rah+23] Md Awsafur Rahman et al. «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)*. 2023, pp. 2200–2204. DOI: [10.1109/ICIP49359.2023.10222083](https://doi.org/10.1109/ICIP49359.2023.10222083).
- [Sel+16] Ramprasaath R. Selvaraju et al. «Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization». In: *International Journal of Computer Vision* 128.2 (ott. 2016), pp. 336–359. ISSN: 1573-1405. DOI: [10.1007/s11263-019-01228-7](https://doi.org/10.1007/s11263-019-01228-7).
- [Ser23] Serengil. *deepface*. 2023. URL: <https://github.com/serengil/deepface?tab=readme-ov-file>.
- [SF19] Suraj Srinivas e François Fleuret. «Full-gradient representation for neural network visualization». In: *Advances in neural information processing systems* 32 (2019).
- [Wol+20] Thomas Wolf et al. «Transformers: State-of-the-Art Natural Language Processing». In: Association for Computational Linguistics, ott. 2020, pp. 38–45. URL: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.
- [Wu+20] Bichen Wu et al. *Visual Transformers: Token-based Image Representation and Processing for Computer Vision*. 2020. arXiv: [2006.03677 \[cs.CV\]](https://arxiv.org/abs/2006.03677).





# LaXidiaN

A L<sup>A</sup>T<sub>E</sub>Xtemplate inspired by Obsidian.

This document was written using LaXidiaN, a sophisticated L<sup>A</sup>T<sub>E</sub>X template inspired by Obsidian. If you have any feedback you'd like to share, be it a complaint or a new feature you'd like to see, please open an issue or a pull request on our [github repository](#).

[LaXidiaN repository](#)