Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica
Laurea Magistrale

Analisi della sicurezza dei modelli per il riconoscimento facciale

Relatore:
Prof. Rebecca Montanari

Dottorando:
Nicolò Romandini

Presentata da:
Gabriele Tassinari

Anno Accademico 2022-2023

Indice

1 Introduzione

Negli ultimi anni, il campo dell'intelligenza artificiale e della sicurezza informatica ha registrato un notevole incremento di interesse nei confronti del riconoscimento facciale, suscitando crescente attenzione in ambito accademico. Le tecnologie di riconoscimento facciale si sono diffusamente consolidate, trovando applicazione in una vasta gamma di settori, tra cui quelli della sicurezza e dei social media, per citarne solo alcuni.

Per quanto concerne l'applicazione delle tecnologie di riconoscimento facciale nei sistemi di sicurezza, è possibile individuare diversi esempi significativi. Ad esempio, i controlli di accesso a edifici sensibili o aree restritte possono avvalersi del riconoscimento facciale per identificare e verificare l'identità degli individui che cercano di accedere a tali aree. Inoltre, le forze dell'ordine utilizzano il riconoscimento facciale per individuare potenziali criminali o individui ricercati, contribuendo così agli sforzi di mantenere l'ordine pubblico.

Parallelamente, i social media hanno adottato sempre più il riconoscimento facciale per scopi diversi. Ad esempio, alcune piattaforme utilizzano questa tecnologia per suggerire l'etichettatura automatica delle persone in foto, semplificando il processo di identificazione degli amici nelle immagini condivise. Allo stesso tempo, il riconoscimento facciale è impiegato per offrire funzionalità di filtraggio e personalizzazione, come la creazione di effetti speciali o filtri virtuali che si adattano ai volti degli utenti durante le videochiamate o i selfie.

Nonostante i numerosi vantaggi che il riconoscimento facciale offre in diversi ambiti, emergono legittime preoccupazioni legate alla sicurezza e alla privacy. Ad esempio, esistono rischi di abusi o utilizzi impropri delle informazioni biometriche raccolte attraverso il riconoscimento facciale. Ciò potrebbe includere la sorveglianza di massa non autorizzata, l'identificazione errata delle persone o la violazione della privacy delle informazioni personali. Inoltre, si sono verificati casi in cui i dati di riconoscimento facciale sono stati compromessi o sfruttati per scopi fraudolenti, sollevando interrogativi sulla sicurezza delle informazioni biometriche. Pertanto, è fondamentale che le tecnologie di riconoscimento facciale siano sviluppate e utilizzate con un approccio attento alla sicurezza e alla privacy.

La ricerca ha rivelato che i sofisticati modelli di riconoscimento facciale, ampiamente utilizzati per le operazioni di identificazione e autenticazione, possono presentare una varietà di vulnerabilità nei confronti di diversi tipi di attacco. Il processo di riconoscimento facciale si basa sulla cattura delle caratteristiche distintive del volto da un video o da un'immagine, le quali vengono successivamente elaborate per generare un vettore di embedding. Questo vettore di embedding è composto da un array di dimensioni N, ad esempio, nell'implementazione di FaceNet, un array di 512 dimensioni, che rappresentano in modo accurato i punti chiave e le caratteristiche peculiari del volto analizzato. Tale array viene memorizzato nel nostro sistema, pronto per essere utilizzato quando necessario, al fine di consentire o negare l'accesso. Pertanto, quando una persona desidera essere autorizzata tramite il riconoscimento facciale, viene generato un nuovo vettore di embedding, che viene quindi confrontato con quello precedentemente memorizzato nel sistema, al fine di determinare se autorizzare o rifiutare l'accesso al dispositivo o al servizio in questione.

Sorge, dunque, un'interessante area di ricerca che riguarda la possibilità di invertire il processo di riconoscimento facciale, cioè generare un'immagine realistica a partire dai dati che identificano i punti chiave del volto. Questo solleva gravi preoccupazioni per la sicurezza, poiché potrebbe consentire a potenziali attaccanti di creare volti falsi al fine di eludere i sistemi di autenticazione o commettere frodi.

L'inversione del processo di riconoscimento facciale rappresenta una sfida complessa e multidisciplinare. Richiede la combinazione di conoscenze approfondite nell'ambito dell'apprendimento automatico, della visione artificiale e della cybersicurezza. Gli attaccanti potrebbero sfruttare questa vulnerabilità per generare immagini artificiali di volti che appaiono realistici e che riescono a ingannare i sistemi di riconoscimento facciale. Ciò potrebbe mettere a rischio la sicurezza dei sistemi di autenticazione basati

sul riconoscimento facciale, che vengono sempre più utilizzati per garantire l'accesso a dispositivi mobili, servizi finanziari e altre applicazioni sensibili.

La consapevolezza di questa possibile minaccia è fondamentale per gli sviluppatori di sistemi di riconoscimento facciale. È necessario dunque investire nella ricerca per migliorare la robustezza e la sicurezza di tali sistemi, sviluppando tecniche di riconoscimento facciale avanzate e resilienti agli attacchi di inversione. Inoltre, è importante promuovere una maggiore consapevolezza tra gli utenti e le organizzazioni sulle potenziali vulnerabilità e sulle contromisure che possono essere adottate per mitigare i rischi associati all'utilizzo del riconoscimento facciale. Solo attraverso sforzi congiunti nel campo della ricerca, della tecnologia e della regolamentazione, sarà possibile garantire una maggiore sicurezza e protezione nell'implementazione delle tecnologie di riconoscimento facciale.

La presente relazione mira dunque a descrive il processo seguito per creare modelli in grado appunto di generare un volto a partire da un vettore di embedding. In primo luogo, utilizzando il modello FaceNet e il dataset LFW-DeepFunneled, noto per la sua qualità e diversità di volti, sono stati creati i vettori di embedding.

L'aspetto cruciale affrontato in profondità nel presente progetto riguarda l'analisi dettagliata della sicurezza dei modelli di riconoscimento facciale. Al fine di valutare le vulnerabilità e le potenziali conseguenze in termini di sicurezza e privacy, sono stati condotti test di Membership Inference Attack. Questi test mirano a determinare se un'immagine generata da un modello provenga o meno dal dataset utilizzato per l'addestramento. Attraverso tale analisi, vengono messi in evidenza gli scenari di vulnerabilità dei modelli e le possibili implicazioni per la sicurezza e la privacy dei soggetti coinvolti.

In aggiunta, il presente articolo offre una riflessione approfondita sulla ricerca degli iperparametri dei modelli di riconoscimento facciale e sui risultati connessi alla configurazione ottimale di tali parametri. L'ottimizzazione accurata degli iperparametri riveste un ruolo di fondamentale importanza per garantire la precisione, l'affidabilità e la robustezza dei modelli di riconoscimento facciale impiegati.

Attraverso un'analisi approfondita dei risultati ottenuti e delle riflessioni critiche emerse dal progetto, il presente articolo fornisce un significativo contributo al dibattito in corso riguardante la sicurezza, l'affidabilità e le implicazioni etiche delle tecnologie di riconoscimento facciale. L'obiettivo finale è quello di promuovere una maggiore consapevolezza e comprensione degli aspetti tecnici e delle sfide coinvolte nell'implementazione e nell'utilizzo responsabile del riconoscimento facciale. Soltanto attraverso un approccio oculato e una comprensione approfondita di tali questioni, sarà possibile sviluppare e adottare politiche e pratiche efficaci per garantire la sicurezza e la protezione dei dati personali nell'ambito del riconoscimento facciale.

2 Background Tecnologico

Per lo svolgimento di questo progetto sono state utilizzate le seguenti tecnologie.

2.1 Machine Learning

Con Machine Learning (ML) si intende la capacità di una intelligenza artificiale di apprendere un procedimento senza che questa sia specificatamente codificata da un essere umano per compierlo. In particolare, il ML si basa su algoritmi di apprendimento che utilizzano i dati di input per elaborare previsioni e prendere decisioni a valle di un processo di addestramento.
Vi sono diverse modalità con cui è possibile allenare gli algoritmi di ML. I principali sono:

- Supervised Learning:
  consiste nel fornire all'algoritmo dati etichettati, al fine di ottenere una funzione che mappi l'input a un output desiderato. Questo avviene rifinendo i parametri dell'algoritmo per iterazioni successive, minimizzando una funzione di costo che misura la correttezza delle previsioni. Questo è il paradigma adottato in questa relazione.

- Unsupervised Learning:
  al contrario della precedente, questa è caratterizzata dall'assenza di dati etichettati. L'algoritmo cerce di individuare strutture o pattern nei dati di input senza che venga fornita una risposta di riferimento.

- Semi-Supervised Learning:
  è una combinazione delle precedenti. Il dataset fornito, è dunque, in parte etichettato ed in parte sprovvisto di etichette.

- Reinforcement Learning:
  l'algoritmo interagisce con un ambiente dinamico e viene definita una funzione in grado di "premiarlo" o "punirlo" a seconda che progredisca o regredisca dall'obiettivo fissato. Questo permette un procedimento a tentativi tipico dell'apprendimento umano ed elimina la necessità di avere dataset estremamente dettagliati.

2.2 Deep Neural Network Learning

Le intelligenze artificiali (IA) sono un ambito di ricerca e sviluppo di grande rilevanza contemporanea. Utilizzano un "Dataset" per addestrare reti neurali o modelli di IA [3]. Il Dataset si suddivide in training set, validation set e testing set. I training set addestrano il modello, i validation set lo validano e i testing set lo testano. La distribuzione dei dati tra questi insiemi varia, ma comunemente si usa il 70-80% per il training set e il 10-15% per i restanti [6].
Nell'addestramento di reti neurali, sono importanti concetti come Epoche, Batch Size e Overfitting. Le Epoche rappresentano il numero di volte in cui il dataset viene presentato alla rete neurale. Con il tempo, il modello si affina e si adatta meglio ai dati di addestramento. Tuttavia, troppe Epoche possono causare Overfitting, dove il modello memorizza i dati di addestramento senza generalizzare su nuovi dati.
Il Batch Size indica quanti esempi di addestramento vengono usati per calcolare l'errore in un'iterazione. Un Batch Size grande produce pesi più accurati, ma rallenta l'apprendimento. Un Batch Size piccolo richiede meno memoria ma può essere instabile a causa delle fluttuazioni dei dati.
L'Overfitting si verifica quando il modello si adatta eccessivamente ai dati di addestramento, perdendo la capacità di generalizzare su nuovi dati. Le cause principali sono la complessità del modello, la mancanza di regolarizzazione, il dataset di addestramento piccolo e la mancanza di dati di validazione.

Le funzioni di attivazione introducono non linearità nei neuroni e consentono alle reti neurali di apprendere relazioni complesse tra input e output. Alcune comuni sono ReLU, sigmoid, Tanh e Leaky ReLU [11]. Le Reti Neurali Convoluzionali (CNN) sono un tipo di rete neurale artificiale [7]. Queste prevedono la creazione di un modello, il quale viene allenato attraverso due fasi principali:

- Forward propagation:
  Durante questa fase, l'input viene passato attraverso una serie di layer della rete per generare l'output finale.
- Backward propagation:
  Durante questa fase, viene calcolato il gradiente dell'errore rispetto ai parametri della rete. Ciò consente di aggiornare i pesi della rete in base per l'appunto all'errore commesso durante la fase di forward propagation nella generazione dell'output.

In generale, supponendo di avere una dataset bidimensionale e ponendo dunque tutti i dati su un piano cartesiano, le reti neurali cercano di identificare la "retta di best fit" in modo da diminuire il più possibile la loss e riuscire a prevedere nel miglior modo possibile.

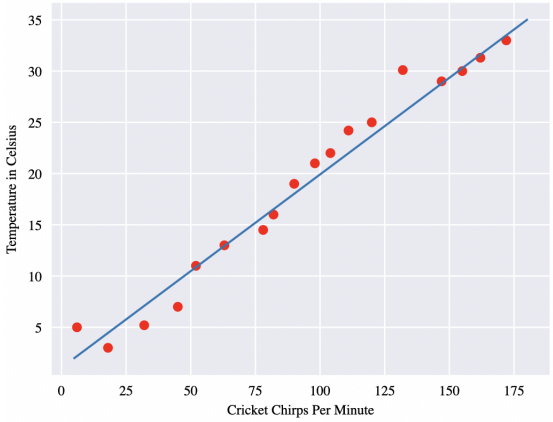

Figura 1: Retta di best fit

Si cerca dunque di trovare, tramite l'addestramento del modello, la retta che consenta di prevedere il risultato migliore e dunque di diminuire la loss.

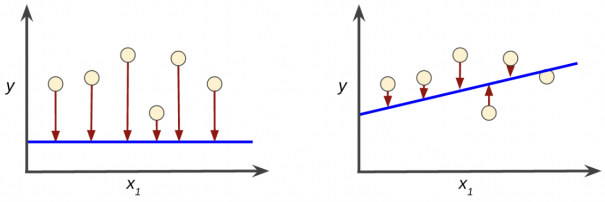

Figura 2: Diminuzione Loss

2.3 Generative Adversarial Networks

Le Generative Adversarial Networks (GAN) [2] sono un tipo di architettura di reti neurali artificiali composta da due componenti principali: un generatore e un discriminatore. L'obiettivo principale delle GAN è generare nuovi dati sintetici che assomiglino a quelli presenti nel dataset di addestramento.
Il generatore prende in input un vettore di rumore casuale e lo trasforma in un'immagine sintetica. Il suo compito è imparare a generare dati che possano ingannare il discriminatore, facendoli apparire simili ai dati reali. Il discriminatore, d'altra parte, è un classificatore che cerca di distinguere tra dati reali e

dati sintetici generati dal generatore. Il suo obiettivo è imparare a distinguere correttamente i dati reali da quelli sintetici.
Durante il processo di addestramento, il generatore e il discriminatore vengono addestrati simultaneamente in modo avversario. Il generatore cerca di migliorare sempre di più la qualità dei dati sintetici generati, mentre il discriminatore cerca di diventare sempre più abile nel distinguere i dati reali da quelli generati.
L'addestramento delle GAN è un processo iterativo in cui il generatore e il discriminatore si sfidano a vicenda, cercando di migliorarsi reciprocamente. Questa competizione crea una dinamica di apprendimento che spinge il generatore a generare dati sempre più realistici e il discriminatore a diventare sempre più efficace nel riconoscere i dati sintetici.
Le GAN sono state utilizzate con successo in diversi ambiti, come la generazione di immagini realistiche, la sintesi di suoni, la generazione di testo e molto altro. Tuttavia, l'addestramento delle GAN può risultare complesso e richiedere risorse computazionali considerevoli. Vari miglioramenti e varianti delle GAN sono state proposte per affrontare sfide specifiche e migliorare le prestazioni, come le Conditional GAN, le Progressive GAN e le CycleGAN.
In generale, le GAN hanno aperto nuove possibilità nel campo della generazione di dati sintetici e hanno contribuito notevolmente all'avanzamento dell'intelligenza artificiale e del deep learning.

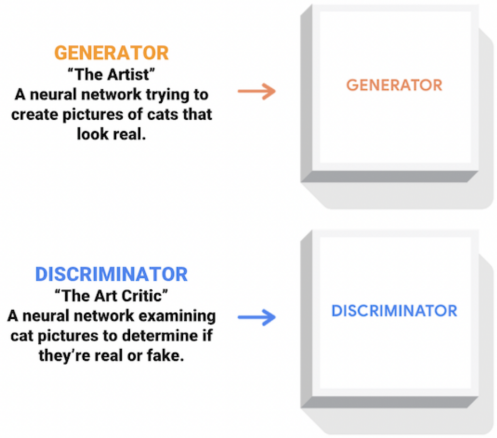

Figura 3: Funzionamento GAN

2.4 Riconoscimento facciale

Il riconoscimento facciale si basa su un processo di estrazione delle caratteristiche facciali e successivamente di confronto degli embedding risultanti. Nel dettaglio, vengono eseguiti questi passaggi:

1. Pre-elaborazione dell'immagine:
   L'immagine del volto viene pre-elaborata per garantire una rappresentazione coerente; viene dunque ridimensionata per adattarla alle dimensioni di input richieste dal modello e viene effettuata la normalizzazione dei valori dei pixel.

2. Estrazione delle caratteristiche:
   L'immagine pre-elaborata passa attraverso la rete neurale convoluzionale (CNN) e viene generato un vettore di embedding, che rappresenta il volto in uno spazio multidimensionale. Si noti che l'embedding è una rappresentazione numerica delle caratteristiche del volto

3. Confronto degli embedding:
   I vettori di embedding vengono utilizzati per confrontare i volti. A tale scopo viene calcolata la distanza tra i rispettivi vettori di embedding utilizzando metodi come la distanza euclidea o la

similarità del coseno. Una distanza bassa o una similarità alta possono indicare una maggiore somiglianza tra i due volti.

4. Classificazione e identificazione:
   Utilizzando le distanze o le similarità calcolate, è possibile eseguire diverse operazioni di riconoscimento facciale. Ad esempio, per la classificazione binaria, si può impostare una soglia, e considerare due volti come uguali se la distanza tra i loro embedding è inferiore a tale soglia. Per l'identificazione, si può confrontare l'embedding di un volto con un database di embedding noti per determinare la corrispondenza più vicina.

2.4.1 Facenet

Facenet [9] è un modello di riconoscimento facciale ad alta precisione sviluppato dagli ingegneri Florian Schroff, Dmitry Kalenichenko e James Philbin di Google. È stato introdotto nel 2015 con l'obiettivo di affrontare le sfide del riconoscimento facciale su larga scala, consentendo di identificare e confrontare volti umani all'interno di grandi dataset.
La caratteristica distintiva di Facenet è l'uso delle reti neurali convoluzionali (CNN) per estrarre le caratteristiche facciali distintive dai volti delle persone, indipendentemente da possibili variazioni, come l'illuminazione, lo sofndo, l'orientamento e le espressioni facciali.
Questo potente modello è inoltre stato addestrato su un vasto dataset di immagini di volti noto come "MS-Celeb-1M" che contiene milioni di immagini di celebrità. Questo addestramento intensivo ha permesso al modello di apprendere rappresentazioni facciali altamente discriminanti e di generare embedding coerenti per volti simili.
Per il progetto è stata utilizzata la versione tensorflow di Facenet presente su Github.

3 Progetto

Il progetto si focalizza sulla creazione di modelli di inversione del riconoscimento facciale che generano volti realistici utilizzando dati di embedding. Viene approfondita l'architettura dei modelli e le sfide riscontrate durante l'implementazione. Inoltre, vengono condotti test di Membership Inference Attack per valutare la sicurezza dei modelli e le possibili vulnerabilità legate alla loro generazione di volti.

Il primo passo nell'implementazione del progetto consiste nella creazione del dataset.
Inizialmente, sono stati scaricati il modello FaceNet [1] (responsabile del riconoscimento facciale) e il database di facce LFW-DeepFunneled [12].

```
[-0.123, -0.020, 0.123, -0.042, -0.122, -0.036,
-0.064...., 0.098]
```

Figura 4: Esempio di vettore di embedding facciale

3.1 LFW-DeepFunneled

LFW-DeepFunneled è un dataset ampiamente utilizzato per la valutazione dei sistemi di riconoscimento facciale. LFW è l'acronimo per "Labeled Faces in the Wild" e rappresenta un dataset di immagini di volti raccolte da Internet, che include una grande varietà di pose, illuminazioni e sfondi.
La versione DeepFunneled del dataset LFW è una versione pre-elaborata, ovvero sono applicate una serie di tecniche di pre-elaborazione per standardizzare le immagini dei volti e ridurre le variazioni indesiderate. Il dataset ad oggi contiene esattamente 13233 immagini di volti di 5749 persone diverse.
Questo dataset è diventato un punto di riferimento importante nella comunità del riconoscimento facciale per confrontare le prestazioni dei modelli e valutare i progressi nell'ambito del riconoscimento facciale su larga scala.

3.2 Creazione embeddings facciali e dataset

Nell'ambito del progetto, è stato avviato il processo di creazione del dataset. Per garantire la compatibilità con il modello Facenet, le immagini provenienti da LFW-DeepFunneled, inizialmente in formato 250x250, sono state ridimensionate alle dimensioni di 160x160, corrispondenti al formato utilizzato per l'addestramento di Facenet.
Successivamente, ogni immagine è stata passata al modello, una alla volta, per ottenere i rispettivi 13233 vettori di embedding facciale, costituiti da un array di 512 features. In questo modo, sono stati generati 13233 input e le relative etichette necessari per l'addestramento della rete neurale.
Dopo aver completato la creazione del dataset, è stato necessario suddividerlo in diverse porzioni. In generale, la prassi suggerisce di suddividere il dataset in tre insiemi distinti: il training set, il validation set e il testing set. Tuttavia, dato il carattere limitato del dataset in questione, si è scelto di creare solamente due set: il training set, utilizzato per addestrare il modello, e il testing set, utilizzato per valutarne le prestazioni [4].

3.3 Creazione della rete

Per avviare l'esplorazione del modello, è stata sviluppata una rete neurale sequenziale, composta da tre strati Dense seguiti da un livello di Reshape [13]. Tale configurazione è stata adottata per elaborare le immagini nel formato RGB, garantendo così la conservazione delle informazioni di colore.
Il primo strato Dense ha il compito di apprendere le caratteristiche iniziali dell'immagine, mentre i successivi strati Dense consentono una progressiva estrazione di feature più complesse e astratte. Infine,

il livello di Reshape viene utilizzato per modellare l'output finale in modo coerente con l'immagine a colori.
Questa configurazione di rete neurale fornisce un framework iniziale per l'elaborazione delle immagini e costituisce una base solida per ulteriori iterazioni e ottimizzazioni del modello.

```
model1 = tf.keras.Sequential([
    tf.keras.layers.Dense(512, activation='relu', input_shape=(512,)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(160 * 160 * 3, activation='sigmoid'),
    tf.keras.layers.Reshape((160, 160, 3))
])
```

Per visualizzare i risultati e monitorare l'andamento dell'addestramento, è stata aggiunta una funzione di callback che viene richiamata al termine di ogni epoca. Questa funzione ha permesso di osservare in modo grafico i progressi della rete neurale durante il processo di addestramento.
La callback ha fornito informazioni cruciali come l'andamento della funzione di perdita e l'accuratezza del modello, consentendo una valutazione visiva delle performance del modello nel corso delle epoche. Questa visualizzazione delle immagini ha permesso di identificare eventuali problemi, come l'overfitting o l'underfitting, e di apportare le opportune modifiche al modello.
In questo modo, al termine di ogni epoca, il modello (allenato con il training set) prende in input un vettore di embedding facciale estratto casualmente dal testing set.

```
class ShowImagesCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        # Scegli casualmente un'immagine dal set di test
        idx = np.random.randint(len(x_test))
        test_image = x_test[idx]
        test_label = y_test[idx]

        # Applica il modello all'immagine scelta
        decoded_image = model.predict(test_image[np.newaxis, ...])[0]

        # Visualizza l'immagine decodificata e la label
        fig, ax = plt.subplots(1, 2)
        ax[0].imshow(decoded_image)
        ax[0].set_title('Decoded Image')
        ax[1].imshow(test_label)
        ax[1].set_title('Label')
        plt.show() #bloccante
```

Successivamente avendo una rete neurale funzionante, si è intrapreso un processo di miglioramento attraverso la modifica dei suoi layer costituenti. Nell'ambito delle reti neurali dedicate alla generazione di immagini (GAN), è prassi comune utilizzare una sequenza di strati Conv2D seguiti da strati MaxPooling2D. Di conseguenza, è stato creato il seguente modello:

```
model2 = tf.keras.Sequential([
    tf.keras.layers.Reshape((16, 32, 1), input_shape=(512,)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
```

```
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(160 * 160 * 3, activation='sigmoid'),
    tf.keras.layers.Reshape((160, 160, 3))
])
```

Ho dunque testato questo secondo modello, ottenendo risultati più accurati rispetto al primo modello. Non ottenendo un'immagine accurata e non avendo a disposizione un vasto dataset, ho dunque provato a complicare ulteriormente la rete, in modo da ottenere output più accurati. Ho dunque creato un nuovo modello, formato da una architettura composta da 3 blocchi convoluzionali, ognuno dei quali comprende due strati convuluzionali, una normalizzazione di batch, un max pooling e un dropout.

Successivamente, è stato eseguito un test utilizzando il secondo modello, il quale ha fornito risultati più precisi rispetto al primo modello. Tuttavia, dato ancora l'insuccesso nell'ottenere immagini accurate a causa della limitata disponibilità di un ampio dataset, è stata intrapresa un'ulteriore complicazione della rete al fine di ottenere output più dettagliati.
È stato quindi creato un nuovo modello costituito da un'architettura composta da tre blocchi convoluzionali, ciascuno dei quali comprende due strati convoluzionali, una normalizzazione batch, un MaxPooling e un Dropout. Tale configurazione è stata adottata per consentire alla rete di apprendere caratteristiche sempre più complesse e significative dalle immagini.
L'inclusione dei blocchi convoluzionali, che combinano diversi strati convoluzionali con tecniche di normalizzazione e dropout, mira a migliorare la capacità della rete di rilevare pattern e strutture significative nelle immagini. Questo approccio permette di gestire la complessità del dataset, migliorando le prestazioni e la capacità di generalizzazione del modello.

```
model = tf.keras.Sequential([
    tf.keras.layers.Reshape((16, 32, 1), input_shape=(512,)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(160 * 160 * 3, activation='sigmoid'),
    tf.keras.layers.Reshape((160, 160, 3))
])
```

Sono stati dunque utilizzati i layer:

- **Reshape**:
  Vieni utilizzato per ridimensionare l'input in una forma tridimensionale.
- **Conv2D**:
  Questo layer applica una convoluzione sulla matrice di input utilizzando un kernel di dimensione 3x3.
- **BatchNormalization**:
  Questo layer normalizza le attivazioni del layer precedente in modo da mantenere la media vicina a 0 e la varianza vicina a 1. Ciò aiuta a prevenire il problema della covarianza tra i layer.

- **MaxPooling2D**:
  Questo layer esegue il downsampling dell'input utilizzando una finestra 2x2 e restituisce il valore massimo della finestra come output. Ciò aiuta a ridurre la dimensione dell'input e a prevenire l'overfitting.

- **Dropout**:
  Viene applicata una regola di regolarizzazione chiamata "dropout" per prevenire l'overfitting. Il layer seleziona casualmente alcuni neuroni e li disattiva durante la fase di addestramento, forzando la rete a imparare una rappresentazione più robusta.

- **Flatten**:
  Questo layer viene utilizzato per appiattire l'output del layer precedente in forma unidimensionale, in modo che possa essere passato a un layer completamente connesso.

- **Dense**:
  Un layer completamente connesso.

- **Reshape**:
  Questo layer viene utilizzato per ridimensionare l'output in una forma tridimensionale di dimensione 160x160x3.

3.4 Loss e Optimizer

Ogni modello creato è stato compilato usando come ottimizzatore 'adam' e come loss 'mse'.

```
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
```

La MSE (Mean Squared Error) è un tipo di funzione di loss comunemente usata in problemi di regressione, in cui l'obiettivo è di prevedere un valore continuo piuttosto che una classe. Essendo questo progetto una realizzazione di una CNN per la generazione di immagini e dunque non un problema di classificazione, la MSE rappresentava la tipologia di funzione di loss più adatta. Nel dettaglio, la funzione calcola la media dei quadrati della differenza tra il valore di output previsto e il valore di output atteso (o target) per ogni campione di addestramento. In altre parole, la funzione MSE misura la distanza tra la previsione del modello e il valore atteso e dunque la loss aumenta all'aumentare della differenza tra questi due valori. La formula per la MSE è:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - y_i)^2$$

dove y è il valore atteso (o target), y_i è il valore previsto dal modello e n è il numero di campioni di addestramento.
Inoltre in una rete, un ottimizzatore è un algoritmo che ha il compito di regolare i pesi dei neuroni durante la fase di addestramento della rete, con l'obiettivo di minimizzare la funzione di loss. All'interno del progetto, è stato usato l'ottimizzatore adam, ovvero uno degli ottimizzatori più comuni. In particolare, Adam ovvero Adaptive Moment Estimation è un algoritmo di ottimizzazione per la discesa del gradiente stocastico (SGD). Questo algoritmo combina le caratteristiche di altri due algoritmi di ottimizzazione:

- AdaGrad:
  Algoritmo che adatta i passi della discesa del gradiente per ciascun parametro in modo indipendente, utilizzando la storia dei gradienti passati in modo da calcolare un tasso di apprendimento separato per ogni parametro.

- RMSProp:
  Questo algoritmo cerca di risolvere un problema di AdaGrad che risulta inefficace davanti a pro-

blemi dove i gradienti cambiano rapidamente. Tuttavia RMSProp non tiene conto dei gradienti passati

L'ottimizzatore Adam oltre ad unire questi due algoritmi, utilizza anche una media mobile dei gradienti passati e una media mobile dei quadrati dei gradienti passati per calcolare le proporzioni di aggiornamento dei parametri e ciò lo rende robusto a gradienti rumorosi e agli sbalzi nella funzione di loss.

Per finire, il modello è stato allenato con il training set precedentemente creato, e valutato utilizzando il testing set, in modo da ottenere una accuracy e una loss veritiere.
Nel dettaglio, si è ottenuta così una accuracy di 0.6655849814414978 e una loss di 0.38326340913772583.

```
# Addestra il modello
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.0,
    callbacks=[ShowImagesCallback()])
# Valuta il modello sul testing set
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=1)
```

3.5 Risultati

Prima di analizzare i risultati occorre effettuare una premessa. Il modello di Facenet utilizzato per creare i dati, effettua le seguenti operazioni alle immagini:

- Rimozione dello sfondo
- Ritagliare il volto
- Rimozione degli oggetti (occhiali, capelli, baffi,...)
- Raddrizzare il volto in modo che questo sia rivolto frontalmente

Ovvero effettua operazioni del tutto conformi al suo utilizzo, ovvero a consentire un riconoscimento facciale accurato indipendentemente dagli oggetti, dalla barba o dall'orientamento del volto che gli viene presentato.

3.5.1 Invertire il vettore facciale di embedding

Richimando ora la funzione di callback on_epoch_end(), si mostrano i progressi della rete effettuati durante il training. Al termine delle prime due epoche, il modello risulta ancora lontano dalla label:

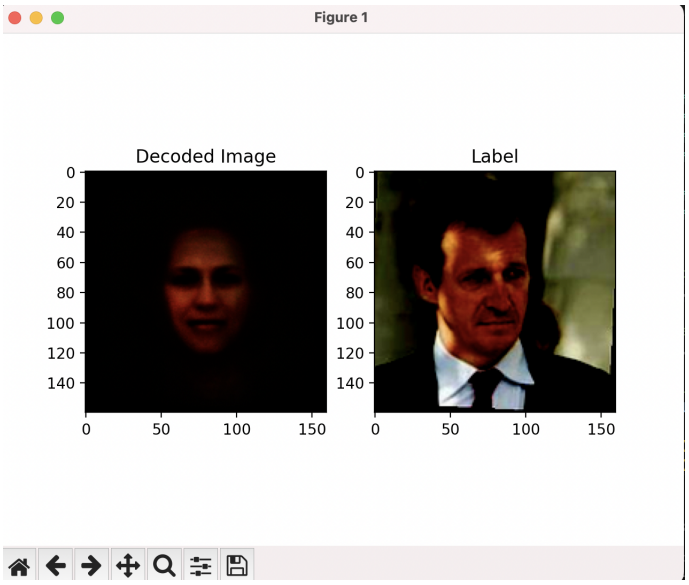

Figura 5: Termine epoca 2/10

Al termine della terza e quarta epoca,il modello inizia a comprendere la forma del volto ed in particolare la forma di alcuni lineamenti come quello del naso:

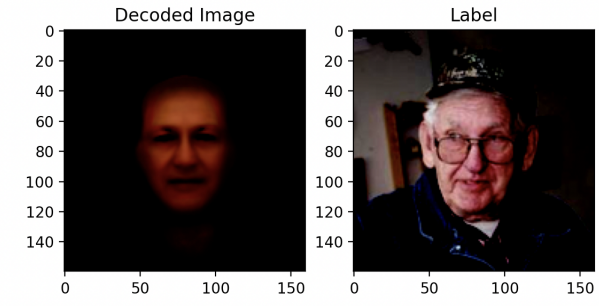

Figura 6: Termine epoca 4/10

Al termine della quinta e sesta epoca, la rete si avvicina alla rappresentazione di un volto accurato, mostrando infatti che la forma del volto diventa sempre più accurata:

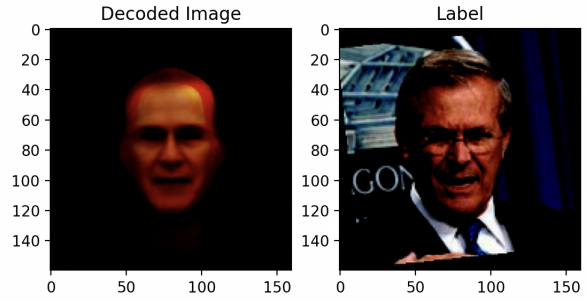

Figura 7: Termine epoca 6/10

Successivamente al termine della settima e ottava epoca, l'accuratezza dell'immagine inizia ad aumentare:

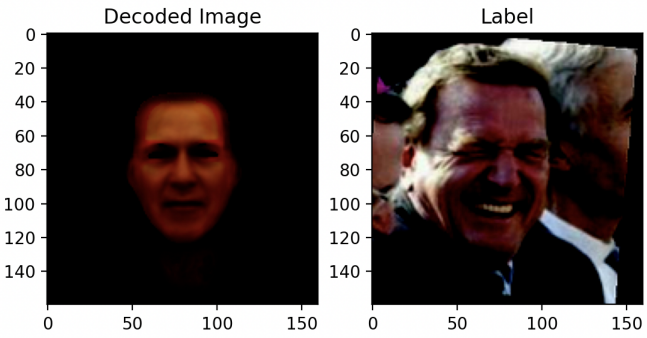

Figura 8: Termine epoca 8/10

Al termine delle ultime due epoche, si ottiene un volto sempre più somigliante alla label:

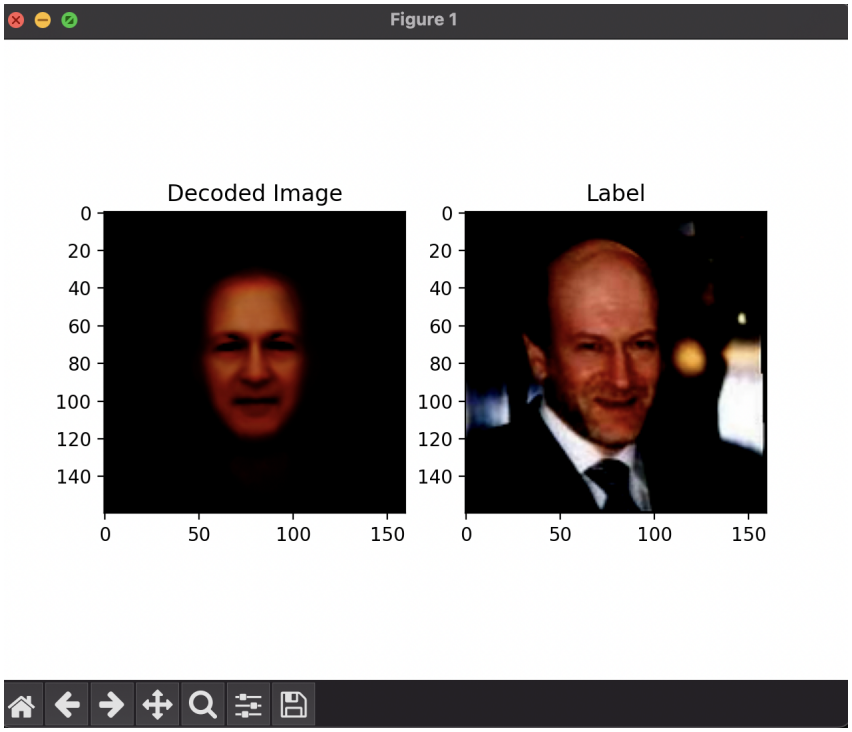

Figura 9: Termine epoca 10/10

Sono stati svolti anche dei test per verificare l'efficienza delle rete, modificando anche empiricamente i valori del numero delle epoche e del batch size. Al fine di ottenere un modello il più accurato possibile.

3.5.2 Membership Inference Attack

È stata eseguita un'analisi sulla possibilità di risalire ai dati utilizzati durante l'addestramento del modello dopo il suo completamento. Tale verifica è stata condotta attraverso l'applicazione di tecniche di Membership Inference, in cui sono stati presentati alla rete neurale tutti i possibili dati (o dati reperibili, come nel caso di immagini disponibili su Internet) come input, seguito dall'analisi dell'output risultante. È emerso che l'output risulta più accurato quando l'input fornito al modello corrisponde a quello incluso nel set di addestramento utilizzato per l'addestramento della rete [10].
È stata confermata questa dinamica nel modello in questione, in quanto il set di addestramento includeva

diverse immagini dell'ex presidente degli Stati Uniti d'America, George W. Bush. È stato scelto di presentare al modello addestrato una foto appartenente al set di addestramento dell'ex presidente, e si è potuto constatare una notevole riduzione della loss e un considerevole aumento dell'accuratezza. Va sottolineato che questo fenomeno non si è mai verificato con immagini non precedentemente fornite alla rete neurale.

4 Discussione

4.1 Ricerca nello spazio degli iperparametri

Gli iperparametri sono parametri che non vengono appresi dal modello durante il processo di addestramento, ma devono essere impostati manualmente prima dell'avvio del processo di addestramento stesso. Questi iperparametri includono il numero di strati nascosti in una rete neurale, il numero di unità in ciascuno di essi, la velocità di apprendimento, la dimensione del batch e molti altri. La scelta dei valori ottimali per questi iperparametri può influire significativamente sulle prestazioni e sulla capacità di generalizzazione del modello [8].
Keras Tuner è una libreria open-source sviluppata da TensorFlow che offre uno strumento per l'ottimizzazione automatica degli iperparametri nei modelli di machine learning creati con Keras [5].
Attraverso l'utilizzo di questa libreria, è possibile automatizzare il processo di ricerca e ottimizzazione degli iperparametri, riducendo così la necessità di esperimenti manuali e migliorando l'efficienza nel trovare le migliori configurazioni per i modelli di machine learning. Ciò consente di ottenere in modo completamente automatizzato la rete neurale ottimale per il compito in questione. Nel contesto di questo progetto, Keras Tuner è stato impiegato per determinare il numero di strati Dense nella rete neurale (3.3), al fine di minimizzare la funzione di perdita.
Per eseguire questa operazione con Keras Tuner, è necessario generare tutti i modelli e addestrarli utilizzando il proprio dataset, permettendo così al tuner di identificare quale modello minimizza la funzione di perdita. Successivamente, è possibile ottenere il modello ottimale nel modo seguente:

```
#Genero un insieme di modelli con iperparametri casuali e scelgo il modello
con la minima perdita (loss) come modello migliore
tuner = keras_tuner.RandomSearch(
    hypermodel=build_model,
    objective="loss",
    max_trials=2, #numero di volte di esecuzione dell'algoritmo di ricerca
    executions_per_trial=2, #numero di esecuzioni per ogni modello
    overwrite=True,
)
tuner.search(x_train, y_train, epochs=10, batch_size=32, validation_split=0.0)
#prendo il modello migliore
best_model = tuner.get_best_models(num_models=1)
```

4.2 Problemi Facenet

Durante questa ricerca, si è riscontrata un'importante problematica legata alla scarsa documentazione di Facenet, rendendo complessa l'individuazione delle informazioni necessarie.
Un ulteriore problema è emerso durante il ridimensionamento delle immagini, con la conversione da immagini 250x250 (prese da lfw-deepfunneled) a immagini di dimensione 160x160, corrispondenti alle dimensioni su cui Facenet è stato addestrato. Tale conversione ha inevitabilmente comportato una perdita di qualità delle immagini, rendendole a volte difficili da riconoscere anche per l'occhio umano. Questi errori hanno chiaramente influenzato negativamente la precisione del modello, che si è trovato a dover elaborare anche immagini di questo tipo.

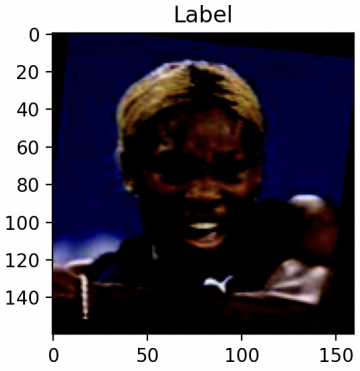

Figura 10: Perdità di qualità e definizione

5 Conclusione

Le reti neurali hanno dimostrato un potenziale rivoluzionario in diversi ambiti grazie alla loro capacità di apprendimento automatico e all'elaborazione efficiente di vasti quantitativi di dati. Esse costituiscono strumenti estremamente potenti e accessibili a tutti. Tuttavia, sorgono delle questioni di sicurezza che richiedono una considerazione attenta.
Una delle principali preoccupazioni riguarda l'ampio spettro di possibilità offerto da queste reti neurali e se siano sufficientemente sicure. È fondamentale esaminare attentamente le potenzialità offerte da tali reti, nonché valutare se i sistemi attualmente in uso siano adeguatamente protetti o se possano emergere nuove vulnerabilità in conseguenza di questa tecnologia innovativa.
In particolare, questa ricerca ha dimostrato la capacità di creare una rete neurale avanzata per la generazione di immagini da un vettore di embedding facciale che ne identificava i punti chiave corrispondenti. Tuttavia, è necessario condurre ulteriori studi e valutazioni approfondite sulla sicurezza per garantire un utilizzo responsabile e sicuro di tali reti neurali nel futuro accademico e oltre.

Riferimenti bibliografici

[1] Ryann Alimuin, Elmer Dadios, Jonathan Dayao, and Shearyl Arenas. Deep hypersphere embedding for real-time face recognition. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18(3):1671–1677, 2020.

[2] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[3] Google. Introduction to neural networks. *Sito*, 1:109, 2019.

[4] Prateek Joshi. *Artificial intelligence with python*. Packt Publishing Ltd, 2017.

[5] Sharmad Joshi, Jessie Ann Owens, Shlok Shah, and Thilanka Munasinghe. Analysis of preprocessing techniques, keras tuner, and transfer learning on cloud street image data. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 4165–4168. IEEE, 2021.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[7] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[8] Mohamad Zaim Awang Pon and Krishna Prakash KK. Hyperparameter tuning of deep learning models in keras. *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing (STAIQC)*, 1(1):36–40, 2021.

[9] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[10] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[11] tensorflow.org. Introduction to tensorflow. *Sito*, 1:45, 2019.

[12] vis www.cs.umass.edu. Lfw-deepfunneled. *Rivista*, 1:4, 2015.

[13] Jinming Zou, Yi Han, and Sung-Sau So. Overview of artificial neural networks. *Artificial neural networks: methods and applications*, pages 14–22, 2009.