A Neural Network for Face Recognition task

Simone Ragusa, Lorenzo Pennisi

1. IL PROBLEMA

1.1 Scelta del problema

Il task che abbiamo scelto di affrontare è quello del riconoscimento facciale che rappresenta una delle sfide più interessanti nel campo dell'elaborazione delle immagini e dell'intelligenza artificiale. L'obiettivo principale del task è l'identificazione di persone all'interno di un database di immagini, attraverso l'individuazione delle loro caratteristiche facciali.

Nel caso particolare, ci siamo occupati della realizzazione di un modello in grado di realizzare delle rappresentazioni (o *embeddings*) delle immagini di ogni identità. L'idea, infatti, era quella di implementare un sistema di face verification che fosse in grado di calcolare la somiglianza dell'input con tutte le immagini della base di dati, basandosi su considerazioni geometriche applicate alle loro rappresentazioni.

1.2 Scenari applicativi

Il riconoscimento facciale ha una vasta gamma di scenari applicativi. Tra questi troviamo l'autenticazione biometrica per il controllo degli accessi fisici o digitali, il monitoraggio della presenza di persone specifiche all'interno di una stanza o l'applicazione in scenari in cui la sicurezza è prioritaria, come aeroporti o banche, sostituendo o integrando i tradizionali metodi di autenticazione basati su password o badge. In conclusione, quindi, un efficace modello di riconoscimento facciale potrebbe influenzare positivamente gran parte degli aspetti della vita quotidiana e dei settori industriali.

1.3 Scelta del dataset

Contrariamente alla scelta dell'approccio operativo, la scelta del dataset è risultata abbastanza immediata e semplice. Abbiamo deciso infatti di basarci su uno dei dataset maggiormente utilizzati per questo genere di task, il *Labeled Faces in the Wild dataset (LFW)*, di cui parleremo in modo più approfondito nella sezione dedicata.

2. IL DATASET

2.1 *Ricerca dei dati*

La nostra idea era quella di trovare un dataset ben organizzato e che avesse una buona varietà di immagini. Dopo una breve ricerca, abbiamo individuato il *Labeled Faces in the Wild (LFW),* un database di 13233 immagini JPEG rappresentanti 5749 identità diverse raccolte dal web. I principali punti di forza del dataset erano i seguenti:

- **Disponibilità in PyTorch**: anche se possibile scaricare il dataset dal web, vista la sua popolarità, è stato integrato tra i dataset presenti in PyTorch e questo è utile per poter istanziare "al volo" il dataset senza doverlo costruire manualmente.
- **Split adattati**: la suddivisione dei dati all'interno del dataset è fondamentale per ottenere set di train e di test bilanciati e che abbiamo al loro interno immagini uniformi (caratteristiche di illuminazione, posa ed espressioni facciali). Nel nostro caso il dataset poteva essere scaricato specificando un flag di 'train' o di 'test', in modo da ottenere split adeguatamente bilanciati in partenza.
- **Standardizzazione delle immagini**: le immagini sono 3 x 250 x 250 (RGB), in cui ogni pixel, per ogni canale, è codificato con un float tra 0.0 e 1.0. Questo riduce il lavoro di preprocessamento dei dati, che si riduce solo alla trasformazione delle immagini in tensori e alla normalizzazione rispetto alle medie e alle deviazioni standard per ogni canale
- **Diversità di approcci**: un altro grosso vantaggio che abbiamo riscontrato è la possibilità di approcciare il dataset in modi diversi, tutti forniti nella documentazione associata.

2.2 *Le caratteristiche del dataset*

Nonostante le buone qualità, il dataset presenta comunque dei piccoli difetti, che ci hanno posto di fronte a diversi ostacoli nel corso dello sviluppo del progetto. Infatti, come anticipato in precedenza, all'interno del dataset è stato possibile riscontrare una grossa varietà di dati, pur con qualche limite. Sulle 5749 identità, solo 1680 contano più di un'immagine nel database, e alcune tipologie di immagini non raggiungono un numero adeguato. Ad esempio, sono presenti poche immagini di bambini, di

anziani e di minoranze etniche. Inoltre, rispetto alla quantità di uomini, le immagini di donne sono meno numerose.

Nonostante questo, abbiamo iniziato a fare le prime sperimentazioni, studiando i dati e come erano organizzati. Gli approcci suggeriti dalla documentazione sono due:

1. *Image Restricted Configuration*

   Le informazioni di training e di test sono ristrette a coppie di immagini etichettate come simili o dissimili, in modo da allenare la rete a riconoscere elementi di similitudine (e non) nelle identità. Questi dati sono disponibili in due file, *pairsDevTrain.txt* e *pairsDevTest.txt,* contenenti, distribuite per riga, un certo numero di coppie di match e altrettante coppie di mismatch. Le prime associano l'ID di due foto diverse a una singola identità (foto simili), mentre le ultime associano a due foto diverse, due persone diverse fra loro (foto dissimili):

Figura 1: Esempio di match

Figura 2: Esempio di mismatch

2. *Unrestricted Configuration*

   Nell'approccio unrestricted le uniche informazioni disponibili si trovano nel file *people.txt*, nel quale in ogni riga è presente il nome di una certa identità associato al numero di immagini di quella persona presenti nel database. Così facendo è possibile generare le coppie simili e dissimili a proprio piacimento, senza dover essere necessariamente vincolati ai dati del database.

Alla luce di ciò la nostra scelta è ricaduta sull'approccio restricted, in quanto più semplice da gestire. Infatti, avendo trovato il dataset disponibile in *PyTorch*, la costruzione delle coppie a partire dai file di testo risultava essere già implementata, risparmiandoci così parte del lavoro.

Dal punto di vista pratico, quindi, non è stato necessario lavorare il dataset in fase iniziale, anche se, come vedremo nella sezione relativa al codice, abbiamo deciso di impacchettare i dati in una classe dedicata che integrasse non solo il loro scaricamento, ma anche una serie di metodi, utili per la loro gestione o manipolazione

3. METODI

In questa sezione verranno specificati i principali metodi di attacco al problema con un approccio top-down. Partiremo quindi dall'esterno, motivando la scelta della rete che abbiamo deciso di costruire, per poi passare al cuore del nostro algoritmo, con un focus sul modello di elaborazione delle immagini e sul metodo di training utilizzato.

3.1 La Siamese Network

Avendo a che fare con un dataset che forniva i dati nel formato di cui abbiamo parlato nella sezione Dataset, abbiamo capito fin da subito che non potevano trattare il problema come un classico problema di classificazione. I motivi erano principalmente due:

1. Quando si lavora con immagini di facce umane, l'estrema variabilità dei lineamenti e delle espressioni facciali rende molto difficile l'individuazione di pattern ricorrenti relativi alla singola persona, soprattutto se si hanno a disposizione pochi esempi per addestrare la rete. Nel nostro caso, circa il 70% delle identità contava solo un'immagine nel database, rendendo critica l'estrazione delle feature caratteristiche.

2. Anche se era possibile trattare il problema come una tipica classificazione, l'elevato numero di identità avrebbe reso troppo pesante gli ultimi layer del modello, incrementando non solo i tempi di addestramento ma anche le risorse necessarie alla computazione

L'unico modo per poter ovviare a questi problemi è il cosiddetto *Similarity Learning*, una tecnica di addestramento in cui si ricerca una funzione che riesca a predire quanto due entità siano realmente simili. Si potrebbe vedere il concetto anche da un punto di vista geometrico, considerando le due immagini come vettori in uno spazio multidimensionale. In quel caso l'obiettivo della rete dovrebbe essere quello di massimizzare la distanza euclidea tra immagini dissimili e minimizzare quella tra immagini simili, compito che in ambito di addestramento viene efficientemente svolto da funzioni di Loss quali le contrastive loss o la triplet loss, di cui discuteremo più avanti.

Dal punto di vista dell'architettura, una rete siamese è composta da due reti neurali separate ma identiche, che condividono i parametri e i pesi.

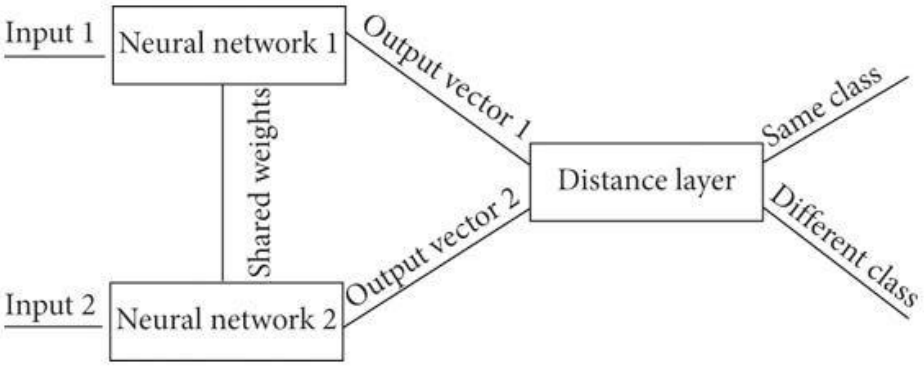

Figura 3: Architettura della Siamese Network

Come si può vedere in figura 3, la prima parte della rete, costituita da due CNN gemelle, si occupa di prendere in ingresso due input separati di cui si vuole conoscere la similarità. Una volta ottenuti gli output vector (embeddings) dei due input, entrambi effettuano un passaggio nel cosiddetto distance layer, in grado di calcolare la distanza euclidea tra i punti rappresentati dai vettori e di determinare con una certa accuratezza se i due input sono simili o no.

3.2 Il modello

Una volta definita l'architettura esterna della rete, possiamo scendere più in profondità per capire come abbiamo realizzato il modello di rappresentazione. Quando si parla di elaborazione di immagini, la scelta della rete neurale da utilizzare ricade sempre nelle *CNN (Convolutional Neural Network)*, per gli svariati vantaggi che queste reti portano con sé. Nella sua forma più semplice, una CNN è composta da una serie di layer di convoluzione, che riducono sensibilmente i parametri per ogni livello, e da una serie di layer di pooling, che attraverso diverse tecniche si occupano di ridurre progressivamente la dimensionalità degli input pur preservandone le caratteristiche principali.

Per il nostro progetto, abbiamo inizialmente pensato di adottare una rete come VGG16, al fine di ritrovarci con un modello pre-allenato ed estremamente adattato all'elaborazione di immagini per il riconoscimento facciale. Tuttavia, tramite alcuni primi esperimenti, ci siamo resi conto che VGG16 andasse troppo al di là delle risorse che avevamo a disposizione, dandoci quindi anche l'idea che potesse bastare qualcosa di molto più leggero per iniziare a sperimentare qualche ciclo di training. Spinti da questa realizzazione, abbiamo deciso di implementare una CNN completamente "from scratch", con l'idea di dare priorità alle conoscenze e all'esperienza che potevamo acquisire pur sacrificando, allo stesso tempo, le prestazioni ottenibili dalla rete.

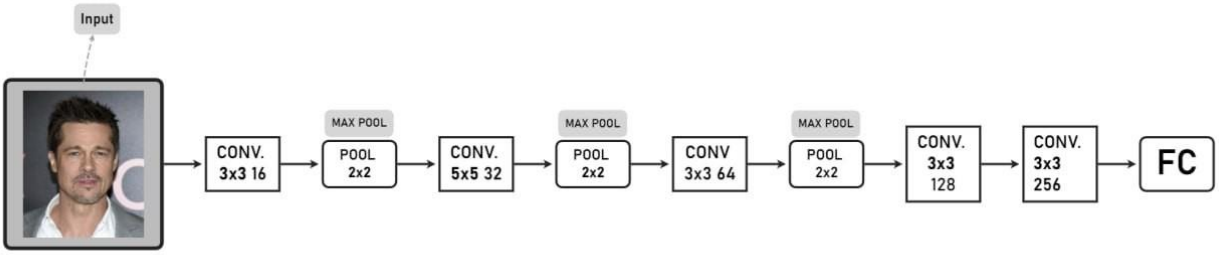

Figura 4: Architettura del modello

Come si può vedere in figura 4, il nostro modello presenta 5 layer di convoluzione con un numero crescente di kernel di dimensioni 3 x 3 o 5 x 5. Dopo ogni layer di convoluzione (tranne gli ultimi) abbiamo inserito un layer di pooling, mirato alla riduzione della dimensionalità delle mappe di feature, tramite la tecnica del MaxPooling. Nonostante non siano presenti in figura, abbiamo deciso di inserire dei layer di batch normalization in ingresso ai layer di convoluzione, avendo riscontrato degli effetti positivi sulla convergenza della loss in fase di training e validation.

Le immagini che vengono passate come input, come anticipato in precedenza, non subiscono particolari passaggi di preprocessing se non:

- un resize a 100 x 100 pixel, per facilitare l'elaborazione delle immagini
- una trasformazione in tensori
- una normalizzazione rispetto alla media e alla deviazione standard delle immagini

Abbiamo deciso di non applicare alcun approccio di data augmentation perché a seguito di alcuni esperimenti abbiamo notato che i risultati ottenuti non differivano particolarmente dalle immagini non processate.

3.3 Metodi di training

L'ultimo step che resta da definire è la scelta del criterio di loss (*il criterion*) e dell'*optimizer*, responsabile della ottimizzazione dei parametri e quindi della discesa del gradiente.

Quando si lavora nell'ambito delle reti siamesi, la loss più adeguata da scegliere è la *contrastive loss*, una funzione che prende in input gli embeddings delle due immagini e un etichetta di ground truth che stabilisce se gli input sono uguali (label = 1) o diversi (label = 0). La funzione è definita nel seguente modo:

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

Figura 5: Contrastive Loss Function

La distanza D_W che vediamo nell'equazione è la distanza euclidea calcolata tra gli embeddings delle immagini, che nel nostro caso viene ricavata appena prima del calcolo della loss. Per rendere questa funzione compatibile con i nostri dati, ci siamo resi conto che doveva essere leggermente modificata. Infatti, nella forma vista in figura 5, la funzione si applica nel caso in cui la label per input simili è 0 e per quelli dissimili 1. Nel primo caso, il secondo termine dell'addizione si annulla, lasciando la quantità $\frac{1}{2}(D_w)^2$ che risulta minimizzata quando la distanza euclidea è piccola, rispecchiando l'idea che va minimizzata la distanza tra rappresentazioni di immagini simili. Un ragionamento analogo può essere fatto per il caso di non similarità, in cui il fattore rimanente è $\frac{1}{2}\{\max(0, m - D_w)\}^2$, che porta al verificarsi di due casi:

1. Se $D_w > m$ allora la distanza tra le rappresentazioni è sufficientemente maggiore del margine impostato e la loss viene posta a 0

2. Se $D_w < m$ allora le due immagini sono troppo vicine rispetto al margine definito e il valore assunto dalla loss tende ad allontanarle il più possibile

Nel nostro caso, avendo le label invertite, abbiamo dovuto apportare una leggera modificata alla funzione, invertendo i coefficienti dei due termini dell'addizione.

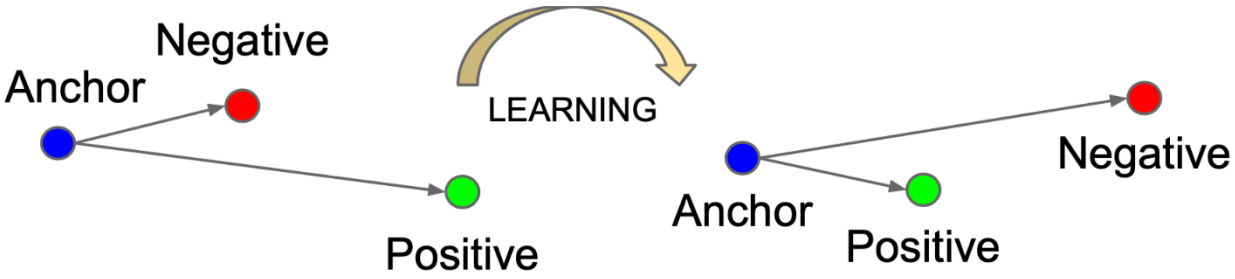

Figura 6: Obiettivo del ciclo di learning: minimizzare la distanza tra coppie simili, massimizzare la distanza tra coppie dissimili

Per quanto riguarda l'optimizer, la scelta è ricaduta *sull'SGD (Stochastic Gradient Descent)*, che velocizza in modo molto efficiente le operazioni di discesa del gradiente

4. VALUTAZIONE

Il principale metro di valutazione scelto è l'*accuracy*, in generale esprimibile come il rapporto tra il numero di predizioni positive e il numero totale di predizioni:

$$Accuracy = \frac{\#\hat{y}_{positive}}{\#y}$$

Con questo tipo di approccio, durante il training, utilizzando coppie di immagini e la rete siamese descritta prima, l'accuracy era legata al numero di coppie correttamente etichettate come simili o dissimili, in rapporto alle etichette di ground truth. Quindi, durante l'elaborazione sul singolo batch, tramite un metodo implementato ad-hoc, abbiamo calcolato le etichette predette sulla base delle distanze tra le immagini e le abbiamo confrontate con il vettore di etichette di ground truth.

Nonostante non rientri nella descrizione delle misure di valutazione, sfruttiamo questa sezione per sottolineare un problema riscontrato in fase di inferenza. Durante quest'ultima fase di progettazione, l'idea era di utilizzare un classificatore *knn* che fosse in grado di decidere la più probabile classe di appartenenza di un'immagine presa come input, dove la classe corrispondeva a una certa identità. Tuttavia, per poter eseguire questo task di classificazione, è necessario che il knn associ preventivamente vettori di feature e label di un set di training, in modo da realizzare

una mappatura completa delle classi che in fase di inferenza sul test set gli consenta di classificare le immagini. Applicando questa visione ai nostri dati ci siamo resi conto che, avendo poche immagini per ogni classe e avendo allo stesso tempo molte classi, il knn non sarebbe mai riuscito a realizzare una mappatura completa delle classi, a prescindere dallo split di train adottato, rendendo impossibile la classificazione di alcune identità.

Il problema così definito non influenza l'efficienza del modello ma non ci ha permesso di ottenere risultati coerenti in fase di inferenza.

Per completezza, nella sezione demo, spieghiamo il funzionamento desiderato, pur non avendo realizzato un vero e proprio testing del modello rispetto al task di face recognition.

5. ESPERIMENTI

Potremmo riassumere tutti gli esperimenti fatti in 3 macro-esperimenti, all'interno dei quali i risultati erano pressoché simili anche al variare dei parametri. Purtroppo, durante le prime due sperimentazioni, non eravamo ancora riusciti a implementare la misura di accuracy durante la fase di training e validation, e per questo mancano i grafici relativi a questo dato.

1. La prima categoria di esperimenti comprendeva una serie di prove fatte per testare diverse dimensioni dei batch e per iniziare a prendere confidenza con gli iper-parametri. Il miglior risultato ottenuto non era per niente soddisfacente ma il profilo della loss di validation ci ha aiutato a capire il problema.

Batch size	Epochs	Learning rate
32	25	0.01

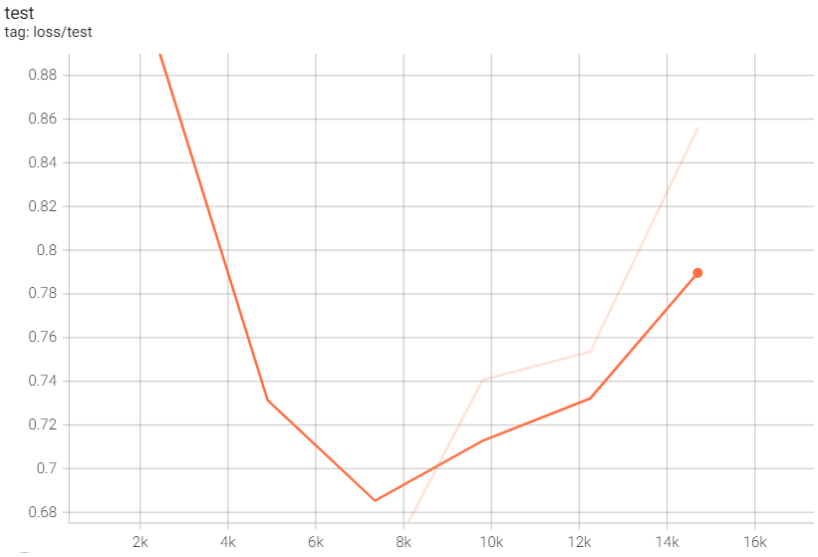

Figura 7: Validation loss sul primo esperimento

Come si può notare dalla figura 6, la loss sul validation set presenta chiari sintomi di un learning rate troppo alto. Abbiamo voluto inserire tra i dati relativi all'esperimento anche la dimensione dei batch in quanto è durante questa prima fase che abbiamo valutato le risposte del sistema a seguito di variazioni di questo parametro, ottenendo il miglior risultato con un batch size di 32. Nel resto della trattazione ometterò, quindi, questo dato.

2. In una seconda fase di sperimentazione abbiamo deciso di diminuire leggermente il valore di learning rate e il numero di epoche, modificando inoltre anche il modello, inserendo prima di ogni layer un layer di batch normalization.

Epochs	Learning rate
50	0.005

Figura 8: Validation loss sul secondo esperimento

In termini numerici, il risultato sembrava essere migliorato, anche se un valore di learning rate ancora più basso avrebbe sicuramente migliorato la convergenza della loss.

3. L'ultima sperimentazione, che ha portato al risultato finale, consisteva in una ulteriore riduzione del learning rate e nell'aggiunta del parametro di regolarizzazione weight decay ai parametri dell'optimizer. Inoltre, in quest'ultimo esperimento, siamo riusciti a riportare anche il grafico relativo all'accuracy misurata in fase di training e validation.

Epochs	Learning rate	Weight decay
50	0.001	0.1

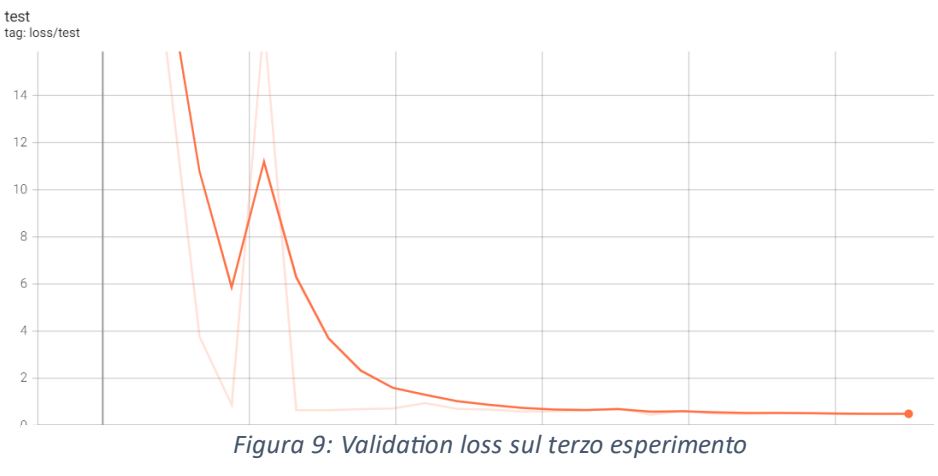

Figura 9: Validation loss sul terzo esperimento

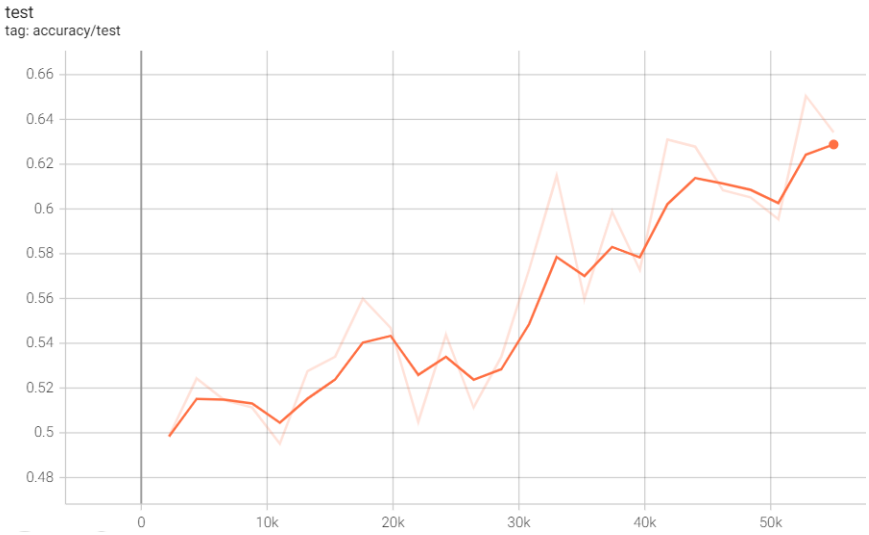

Figura 10: Validation accuracy sul terzo esperimento

Validation loss	Accuracy value
0.49	0.62

Successivi esperimenti consistevano in leggere modifiche degli iper-parametri, ma ottenendo solo risultati peggiori rispetto a quest'ultimo test, abbiamo deciso di concludere così la fase di sperimentazioni.

6. DEMO

Nonostante i problemi già discussi nella sezione Valutazione, descriviamo brevemente il funzionamento desiderato della demo del sistema di riconoscimento

Per poter verificare l'efficienza del modello in fase di inferenza, abbiamo realizzato una demo molto semplice, contenuta nel file demo.py. Dopo aver importato tutte le dipendenze necessarie, viene caricato il modello di rappresentazione allenato e viene istanziato un oggetto di tipo dataset, necessario per due motivi:

1. Ricavare le immagini relative a ogni identità per passarle al modello, il quale si occuperà di realizzarne gli embeddings. Così facendo saremo in grado di costruire una base di conoscenza composta da vettori di feature associati alle rispettive label (le identità).
2. Estrarre casualmente immagini da passare al modello per effettuare il testing.

Successivamente si definisce un knn classifier, che sarà responsabile della classificazione delle immagini di test. Dal punto di vista pratico la demo presenta un semplice menù testuale che fornisce all'utente le opzioni di test del modello su un'identità randomica o di uscita dalla demo. Se si sceglie di testare il modello, verrà stampata a schermo l'immagine caricata in modo casuale e l'etichetta di ground truth associata a quella persona, per poi ottenere in pochi secondi anche la predizione effettuata dal classificatore knn.

7. CODICE

Il codice è organizzato in tre file python che separano le responsabilità all'interno del progetto:

7.1 MyDatasets.py

Il file *MyDatasets.py* è responsabile del caricamento e della gestione del dataset LFW. Al suo interno, infatti, è stata definita la classe PairsDataset, il cui costruttore si occupa di scaricare il dataset da internet (se non presente nella directory di lavoro) e di applicare eventuali trasformazioni ai dati. Oltre a istanziare gli oggetti train_pairs e test_pairs (a partire dalla classe LFWPairs), che contengono i metodi per estrarre le coppie dai file pairsDevTrain.txt e pairsDevTest.txt, questo file assume un ruolo molto importante nel codice perché responsabile anche dell'estrazione di un set di validazione, oltre che della creazione dei dataloader per ogni set di dati. Infatti, per comodità, abbiamo deciso di implementare un metodo specifico, get_loaders, che prende in ingresso la batch_size desiderata e restituisce un dizionario contenente i diversi dataloader (train, validation e test). Ai fini del test di inferenza finale, all'interno della classe istanziamo anche un oggetto dataset di tipo LFWPeople, che ha implementati al suo interno metodi utili per ottenere la label associata a ogni identità presente nel database.

7.2 MyModel.py

Questo file si occupa di definire l'intera rete siamese, in ogni suo componente. Al suo interno abbiamo definito, infatti, una classe EmbeddingNet, che definisce la backbone su cui il nostro modello si basa, la classe ContrastiveLoss, necessaria per l'ottimizzazione dei parametri e la classe SiameseNetworkTask. Quest'ultima raccoglie tutti gli elementi citati in precedenza, creando un'unica unità di elaborazione in grado di effettuare il training e il test del modello. La prima operazione viene svolta da un metodo specifico (training_step) che restituisce, alla fine del training, il modello allenato. Quest'ultimo viene infine salvato su file se l'utente lo ritiene adeguato. La seconda operazione, invece, è affidata al metodo test_accuracy, il quale si occupa di effettuare il test del modello sui dati del test split, mai visti in precedenza, per poi restituire una misura di accuratezza del sistema.

7.3 main.py

In *main.py*, viene assemblato tutto l'occorrente per effettuare un ciclo di training o il test del modello. Infatti, al suo avvio, oltre a definire le dipendenze del codice, viene subito instanziato un oggetto PairsDataset, consentendo il download o il caricamento dei dati. Inoltre è qui che vengono definite le trasformazioni da applicare alle immagini, le quali vengono passate al costruttore di PairsDataset. Successivamente viene creata l'intera rete siamese e viene immediatamente chiesto all'utente, tramite un piccolo menu testuale, se si vuole addestrare o testare il modello. Nel primo caso è possibile scegliere manualmente il learning rate da impostare e il nome che si vuole dare al file di log, per poi passare questi parametri al metodo training_step della rete che avvia il ciclo di training. Nel secondo caso, viene caricato un modello pre-allenato in precedenza (se esistente) per poi invocare il metodo test_accuracy della rete sui dati di test. Alla fine del testing viene visualizzato un valore di accuracy medio, calcolato tenendo conto dell'accuracy su ogni batch.

8. CONCLUSIONE

Il lavoro svolto ci ha permesso di implementare un modello in grado di ricavare l'identità di una persona a partire da una sua immagine, avendo anche solo un'altra immagine di quella persona come punto di riferimento. L'accuratezza raggiunta in fase di test (pari circa al 62%) su immagini che il modello non aveva mai visto ci ha dato un'idea di quanto il mondo della face verification, e più in generale della face recognition, sia vasto e con tanto margine di miglioramento.

Nonostante le difficoltà riscontrate, abbiamo potuto toccare con mano i problemi più rilevanti nell'ambito del machine learning, quali:

- Una corretta scelta dei dati e della gestione del dataset
- L'importanza critica della scelta del modello e dei parametri di ottimizzazione
- La rilevanza dell'organizzazione del codice e della gestione delle limitate risorse hardware

Inoltre, i problemi che abbiamo avuto con il test finale del modello, lasciano aperta la sfida che ci siamo imposti, permettendoci di rivedere l'approccio utilizzato nella valutazione del sistema basandoci su uno studio più dettagliato delle tecnologie esistenti. Inoltre, partendo da ciò che abbiamo costruito, riteniamo che l'ottimizzazione del modello non abbia limiti. A partire da approcci di data augmentation fino a più avanzate tecniche di transfer learning, il nostro lavoro potrebbe vedere risultati di gran lunga più soddisfacenti.