Università Politecnica delle Marche

FACOLTÀ DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Analisi Sistematica di Algoritmi di Riconoscimento Facciale Basati su Deep Learning per Immagini Derivanti da Fotosegnalamento

Systematic Analysis of Deep Learning Based Face Recognition Algorithms for Law Enforcement Identification Images

Relatore:
Chiar.mo Prof. Aldo Franco Dragoni

Correlatore:
Dott. Paolo Sernani

Candidato:
Fabio Raffaeli
Matricola 1083300

Anno Accademico 2019–2020

Sommario

Il riconoscimento facciale automatico è una tecnica biometrica per l'identificazione ormai ampiamente diffusa e utilizzata nei più svariati ambiti applicativi. Seppur argomento di ricerca da lungo tempo, i maggiori sviluppi nel settore del riconoscimento facciale si sono avuti nel corso dell'ultimo decennio grazie all'affermazione del deep learning come base per la risoluzione di tali problemi, che ha consentito di raggiungere prestazioni eccellenti, impensabili fino a poco prima. Di particolare interesse sono gli algoritmi di riconoscimento facciale per l'uso da parte delle forze dell'ordine in quelle operazioni di riconoscimento in cui tecniche tradizionali siano difficoltose o del tutto impossibili da applicare. In questa tesi analizzo alcuni dei migliori sistemi di riconoscimento facciale finora sviluppati, come VGGFace [32] [7], FaceNet [36] e OpenFace [4], con lo scopo di delineare la situazione odierna in tale settore e consentire migliorie nelle tecniche di fotosegnalamento adottate dalla Polizia di Stato, in virtù di una collaborazione tra questa e l'università Politecnica delle Marche. Dapprima mi concentro su una trattazione teorica dei dettagli di tali modelli e traccio l'evoluzione che ha consentito di raggiungere lo stato di avanzamento tecnologico attuale. Poi conduco esperimenti su tali modelli volti a valutarne le prestazioni in termini di accuratezza e tempi di esecuzione in compiti di identificazione (riconoscimento 1:N). Sulla base dei risultati ottenuti in questi esperimenti, concludo che il sistema che garantisce il miglior livello di accuratezza tra quelli testati è VGGFace2, nella versione che poggia sull'architettura SENet50, mentre quello con tempi di esecuzione minori è OpenFace.

Indice

1 Introduzione

Il riconoscimento facciale è una delle tecniche biometriche maggiormente utilizzate per l'autenticazione di persone e trova applicazione in un vasto raggio di settori quali quello militare, quello della finanza e quello della sicurezza pubblica, oltre ad essere presente in numerose situazioni di vita quotidiana. Ricopre una particolare importanza proprio nell'ambito della sicurezza pubblica, grazie all'elevato grado di attendibilità ormai raggiunto, che consente di facilitare operazioni di riconoscimento altrimenti difficoltose. Il suo utilizzo da parte di forze dell'ordine in tutto il mondo è piuttosto consolidato e, con riferimento ad una collaborazione in essere tra l'Università Politecnica delle Marche e la Polizia di Stato, è di interesse affinare le tecniche di fotosegnalamento da essa adottate così da migliorare i risultati ottenuti in operazioni di riconoscimento facciale. A tal proposito, è obiettivo di questa tesi analizzare i più moderni sistemi di riconoscimento facciale pubblicamente disponibili e testarli al fine di fornire un resoconto dettagliato delle loro prestazioni, in modo da stabilire quale di essi si presta maggiormente all'uso suddetto. Esternamente a questa tesi, sul modello così individuato verranno condotti test facendo uso di immagini provenienti da fotosegnalamento con lo scopo di individuare possibili modifiche da applicare alle tecniche di fotosegnalamento stesse per migliorare le prestazioni nelle operazioni di riconoscimento.

Il riconoscimento facciale include due tipi di operazione: la verifica e l'identificazione. Si parla di verifica facciale quando si vuole stabilire se un volto dato corrisponde o meno ad una certa identità (riconoscimento 1:1) e di identificazione facciale quando si vuole individuare l'identità corrispondente al volto dato tra più possibili identità (riconoscimento 1:N). Entrambe le operazioni possono essere utili in svariate circostanze e questo vale anche per i possibili utilizzi da parte delle forze dell'ordine. Un esempio di una potenziale applicazione della prima tecnica è la verifica dell'identità di un soggetto in ambienti come un aeroporto o simili tramite il confronto tra l'immagine riportata nel suo documento di identità e una catturata in tempo reale del soggetto stesso. Per quanto riguarda la seconda tecnica, un possibile utilizzo consiste nell'analisi dell'immagine di un individuo segnalato dalle forze dell'ordine volta a stabilire se esso appartiene o meno ad un archivio fotografico, in modo da risalire alla sua identità qualora non altrimenti possibile. Un ulteriore esempio di applicazione è quello del riconoscimento dell'identità di cadaveri in quei casi in cui sia problematica l'identificazione tramite tecniche tradizionali. In questa tesi mi concentro sui problemi di identificazione, ma le considerazioni fatte e le tecnologie descritte hanno portata più generale e possono essere usate con gli opportuni accorgimenti anche per problemi di identificazione.

Inizialmente fornisco una descrizione dei progressi nel settore del riconoscimento facciale che hanno consentito di raggiungere lo stato attuale di

avanzamento tecnologico. Mi concentro, poi, sulle più moderne tecniche in uso in tale settore, con particolare riferimento alle reti neurali convoluzionali. A tal proposito, tratto in maniera dettagliata i sistemi di cui mi sono servito nella conduzione dei test. Infine spiego come è avvenuta la fase di predisposizione degli esperimenti e presento i risultati che ho ottenuto, fornendo indicazioni utili ad interpretarli in modo corretto.

2 Evoluzione storica

Il riconoscimento facciale è stato oggetto di studio nella comunità della *Computer Vision* da lungo tempo e ha acquisito particolare popolarità negli anni '90 a seguito della pubblicazione del modello Eigenface [48]. Questo modello si proponeva di estrarre le informazioni rilevanti dall'immagine di un volto, codificarle nel modo più efficiente possibile, e comparare la codifica ottenuta con un database di modelli generati nello stesso modo. Per l'estrazione delle informazioni si basava sulle variazioni rilevate in una collezione di immagini, senza fare alcun tipo di giudizio sulle specifiche caratteristiche di un volto. Un approccio del genere è denominato olistico, poiché concentrato su aspetti globali delle immagini piuttosto che sui singoli dettagli, e ha accomunato numerosi altri studi condotti nel periodo a seguire, nati proprio sulla scia di questa pubblicazione.

Un problema noto che affligge metodi del genere è che spesso essi non sono in grado di far fronte a variazioni facciali incontrollate che deviano rispetto alle assunzioni su cui tali modelli sono basati [51]. Proprio per questo motivo agli inizi degli anni 2000 ha preso piede un nuovo tipo di approccio basato sull'uso di descrittori di *features* (caratteristiche) definite localmente per le immagini. Alcune delle tecniche di rappresentazione adottate in tale contesto sono state Gabor [27] e LPB (*Local Binary Patterns*) [2], le quali si sono contraddistinte per le buone prestazioni garantite. Sfortunatamente, le feature delle immagini venivano ottenute manualmente, e questo determinava una mancanza di compattezza e di capacità di generalizzazione del sistema.

Il passo successivo, quindi, è stato introdurre le tecniche di apprendimento automatico al problema del riconoscimento facciale. Così facendo, intorno al 2010 si è passati da feature codificate manualmente a descrittori di volti che venissero appresi da parte di un modello [8], ottenendo una migliore capacità di distinzione e una maggiore compattezza nella rappresentazione. Tuttavia queste rappresentazioni, che possono essere definite "superficiali" (in contrapposizione con i successivi metodi sviluppatosi, definiti "profondi"), mostravano ancora forti limiti di robustezza rispetto alle variazioni non lineari nell'aspetto dei volti. Inoltre, in generale, i metodi tradizionali tentavano di far fronte ad un solo aspetto alla volta delle possibili variazioni tra le immagini, come ad esempio le condizioni di luce, la postura o l'espressione del viso, e non si aveva una tecnica integrata che tenesse conto di tutti

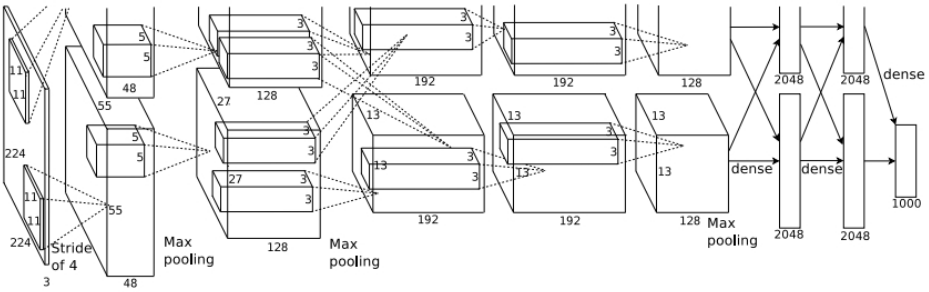


Figura 1: Architettura della rete AlexNet che ha contribuito alla rivoluzione nel settore del riconoscimento facciale. Essa è composta da otto strati, di cui i primi cinque sono di convoluzione mentre i rimanenti tre sono completamente connessi. La rete è suddivisa in due rami perché addestrata facendo uso di due GPU distinte: ogni ramo è di competenza di una delle GPU. (Fonte: [24])

questi aspetti contemporaneamente. Il risultato è stato che tutti gli sforzi compiuti fino a quel momento hanno portato a miglioramenti molto contenuti e che hanno richiesto parecchi anni di lavoro per essere raggiunti. Come conseguenza di tale insufficienza tecnica, i sistemi di riconoscimento facciale disponibili all'epoca venivano considerati molto poco stabili e davano luogo non di rado a interpretazioni errate in applicazioni nel mondo reale.

La situazione è cambiata profondamente nel 2012 a seguito della pubblicazione di AlexNet [24], un sistema per la classificazione di oggetti che ha vinto la competizione ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) [34] surclassando gli altri concorrenti grazie ad un tasso di errore nelle migliori 5 predizioni (la probabilità di non trovare la corretta classificazione di un'immagine tra le migliori 5 predizioni del modello) del 15,3 %, contro un secondo miglior risultato del 26,2 %. La particolarità di questo sistema era l'essere basato su tecniche di deep learning, fino a quel momento mai applicate al campo del riconoscimento facciale. Il sistema era costituito da una rete neurale convoluzionale profonda ben 8 strati (si veda la figura 1), dei quali ognuno apprendeva livelli di rappresentazione differenti corrispondenti a livelli di astrazione differenti, e garantiva una grande stabilità alle variazioni presenti tra le immagini. Questo grande risultato ha segnato l'inizio dell'adozione di tecniche di deep learning come standard per i problemi di estrazione di caratteristiche nel campo del riconoscimento visuale.

Le grandi innovazioni introdotte da AlexNet sono state applicate in breve tempo anche al campo del riconoscimento facciale da parte di un gruppo di ricerca di Facebook che, nel 2014, ha proposto DeepFace [45], un modello di rete neurale convoluzionale costituito da 9 strati e addestrato su oltre 4 milioni di immagini, che ha raggiunto il miglior livello di accuratezza per l'epoca sul dataset LFW (*Labeled Faces in the Wild*) [16], avvicinandosi per

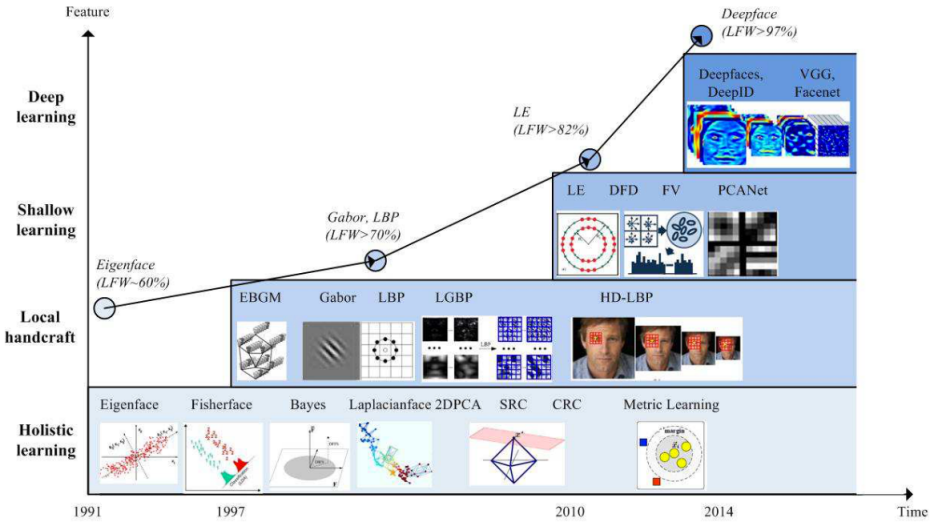


Figura 2: Evoluzione delle tecniche di rappresentazione di volti per il riconoscimento. I metodi olistici hanno dominato il panorama negli anni '90. Nei primi anni 2000 i descrittori realizzati "a mano" hanno acquisito popolarità, mentre l'apprendimento automatico di caratteristiche locali è stato introdotto negli ultimi anni 2000. Dal 2014 in poi si è avuta l'affermazione di tecniche basate sul deep learning. (Fonte: [51])

la prima volta alle prestazioni umane (DeepFace: 97,35 % - Umano: 97,53 %). Da questo momento in poi, il deep learning si è consolidato come standard anche nel settore del riconoscimento facciale (si veda la figura 2 per una panoramica dei momenti salienti nel progresso).

Sull'onda del successo ottenuto da DeepFace, numerosi altri gruppi di ricerca hanno proposto negli anni successivi nuovi modelli di reti neurali pensati per il riconoscimento facciale e l'affermazione del deep learning in tale settore ha consentito di raggiungere rapidamente livelli di accuratezza su LFW superiori al $99,80\,\%$, risultato impensabile solo fino a pochi anni prima. Tra i lavori più rilevanti troviamo, oltre al già nominato DeepFace, la serie DeepID [40] [39] [41] [38], VGGFace [32], VGGFace2 [7] e FaceNet [36].

3 Reti neurali convoluzionali

Le reti neurali convoluzionali (CNN, *Convolutional Neural Network*) sono particolari tipi di reti neurali pensate per elaborare immagini. Come le reti neurali classiche, esse sono costituite di più strati composti da neuroni artificiali (detti anche unità). Neuroni di strati differenti sono collegati tra loro opportunamente e ad essi sono associati pesi e bias, appresi a seguito di un processo di addestramento, mediante i quali è possibile ottenere il

comportamento desiderato da parte della rete. Una delle differenze tra le CNN e le reti neurali classiche è che per le prime viene fatta l'assunzione esplicita che l'input sia un'immagine e, sulla base di ciò, è possibile semplificare lo schema di collegamento tra gli strati rispetto a quello che si avrebbe altrimenti. In questo modo si riduce il numero di parametri della rete e, conseguentemente, la complessità computazionale e il rischio di *overfitting* (sovradattamento) che ne derivano. La fonte di ispirazione per l'ideazione di queste reti è stata la corteccia visiva animale, di cui esse richiamano vagamente la struttura nello schema di interconnessione dei neuroni.

In una rete convoluzionale gli input e gli output di ogni strato sono distribuiti su tre dimensioni: larghezza (L), altezza (H) e profondità (D), motivo per cui si usa, talvolta, il termine *volume* per riferirsi ad essi. Lo stesso vale anche per la distribuzione dei neuroni all'interno degli strati stessi. È bene osservare come il termine profondità, in questo contesto, assuma un significato diverso da quello che si usa quando si parla di profondità di un'intera rete, intesa come numero di strati che la compongono. Sulla base di quanto detto, anche le immagini che vengono fornite come input ad una CNN vanno interpretate come volumi e non come oggetti bidimensionali. In particolare, le dimensioni di larghezza e altezza del volume sono date dal numero di pixel che l'immagine presenta in queste due dimensioni, mentre la dimensione della profondità è data dal numero di canali: tre per un'immagine RGB, che è il caso più comune. Tale volume può essere visto come la sovrapposizione di più matrici, una per ogni canale, i cui elementi sono i valori di intensità dei pixel dell'immagine nel canale corrispondente. Ci si riferisce spesso alle dimensioni di larghezza e altezza come dimensioni spaziali, trattandole in modo separato dalla dimensione della profondità.

Per comprendere la necessità che ha portato allo sviluppo di questo tipo di rete per la gestione di input visuali, si prenda in considerazione il caso in cui si abbia un'immagine RGB di dimensione 224×224, affatto grande per gli standard di dimensione delle immagini. Utilizzando una rete neurale classica si avrebbe, per un solo neurone del primo strato nascosto, una connessione per ogni valore del volume di input, per un totale di $224 \cdot 224 \cdot 3 = 150\,528$ connessioni e altrettanti pesi. Questa considerazione andrebbe, poi, estesa ad ogni neurone dello strato nascosto e ripetuta per ognuno degli strati della rete, dando luogo a un numero estremamente elevato di parametri. Inoltre, con una rete tradizionale non si sfrutterebbero in alcuno modo le relazioni spaziali che intrinsecamente i pixel di un'immagine hanno tra loro. Le CNN pongono rimedio a questo problema traendo vantaggio proprio da tali relazioni.

Nel seguito vengono presentati gli strati più importanti che compongono una rete convoluzionale, che sono individuabili anche nelle architetture più innovative ed articolate. Bisogna tenere presente che di tali strati solo alcuni introducono nuovi parametri e che nel conteggio del numero di strati di una rete, cioè della sua profondità, si è soliti non considerare quelli privi di parametri. Di qui in avanti, se non altrimenti specificato, anche io seguo

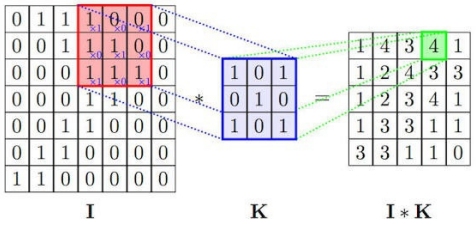


Figura 3: Operazione di convoluzione tra un input I e un kernel K. Per facilità di visualizzazione I e K sono volumi con profondità unitaria, cioè matrici. In generale non è così. (Fonte: [49])

questa convenzione.

Strato convoluzionale

Lo strato convoluzionale rappresenta l'elemento più importante di una CNN ed è caratterizzato da un certo numero di filtri (o *kernel*), che devono essere applicati al volume di input in un'operazione di convoluzione. Un filtro è anch'esso un volume, di dimensioni spaziali solitamente diverse (e inferiori) da quelle dell'input ma con uguale profondità. Esso rappresenta un elemento in grado di rilevare particolari caratteristiche nell'immagine su cui è applicato, come bordi o forme, e viene appreso dalla rete a seguito del processo di addestramento. In termini intuitivi, la convoluzione tra il volume di input e un kernel consiste nel far scorrere quest'ultimo lungo la superficie spaziale del primo, con un certo passo, e per ogni regione coperta effettuare il prodotto scalare tra i valori del kernel e quelli nelle posizioni corrispondenti della regione dell'input, per ogni livello di profondità (si veda la figura 3). Così facendo si ottiene una matrice di valori che prende il nome di *feature map* o *activation map*. Tale operazione viene ripetuta per ciascuno dei filtri e le feature map risultanti vengono impilate le une sulle altre per ottenere il volume di output, che, di conseguenza, ha profondità pari al numero di filtri applicati.

In modo equivalente si può dire che ogni neurone di uno strato di convoluzione è connesso solo ad una porzione dei neuroni dello strato precedente, chiamata campo ricettivo locale (*local receptive field*), e interpretare il filtro come un insieme di pesi associati proprio agli archi di connessione tra ogni neurone dello strato di convoluzione e i neuroni del rispettivo campo ricettivo locale dell'input. La peculiarità di una struttura del genere è, quindi, che l'insieme dei pesi - e cioè il filtro - è condiviso da tutti i neuroni di uno strato, con notevole risparmio sul numero di parametri rispetto ad una rete neurale tradizionale. Questo aspetto è in accordo con il fatto che una determinata caratteristica può essere presente in ogni punto di un'immagine e applicare uno stesso filtro consente di rilevarla indipendentemente dalla sua posizione. Ad un filtro è anche associato un bias.

In base a quanto detto finora, la dimensione spaziale dell'output di uno strato convoluzionale risulterebbe determinata completamente dalla dimensione spaziale dell'input e da quella del filtro. In realtà, è possibile introdurre un controllo tramite l'uso del *padding*, cioè un riempimento di valori posti a 0 che viene applicato al bordo di un volume di input per regolare la dimensione spaziale dell'output. Nella maggior parte dei casi esso è utilizzato per fare in modo che l'output abbia altezza e larghezza pari a quelle dell'input.

Uno strato convoluzionale si può trovare a qualsiasi livello di profondità in una rete neurale e, in generale, quelli che si trovano più in superficie rilevano caratteristiche più astratte e generiche, come semplici bordi tra gli elementi di un'immagine, mentre quelli in profondità rilevano caratteristiche più articolate, come ad esempio delle particolari forme.

In sintesi, per uno strato convoluzionale devono essere stabiliti:

- il numero di filtri K;
- la dimensione dei filtri F;
- il passo S (*stride*);
- il padding P.

Dato un generico volume di input di dimensione $W_1 \times H_1 \times D_1$, tale strato produce un output di dimensione $W_2 \times H_2 \times D_2$, dove

$$\begin{aligned} W_2 &= \frac{W_1 - F + 2P}{S} + 1 \\ H_2 &= \frac{H_1 - F + 2P}{S} + 1 \\ D_2 &= K \end{aligned}$$

Strato ReLU

ReLU (*Rectified Linear Unit*) è il nome con cui si indica un'unità che applica la funzione di attivazione $f(z) = \max(0, z)$. Essa porta a 0 i valori negativi presenti nelle activation map del volume su cui viene applicata e lascia invariati gli altri, come mostrato in figura 4. In tale procedura le dimensioni del volume rimangono invariate. Il compito dello strato ReLU è quello di introdurre una non-linearità nella rete, allo scopo di migliorarne le capacità rappresentative. Dato che, tipicamente, ad ogni strato di convoluzione è sempre applicata una funzione di attivazione di questo genere, talvolta lo strato ReLU viene considerato come parte integrante del primo e non lo si specifica in modo esplicito.

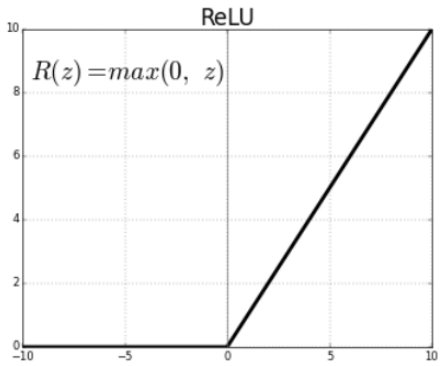


Figura 4: Funzione di attivazione ReLU. (Fonte: [35])

Strato di pooling

Uno strato di pooling ha il compito di ridurre le dimensioni spaziali delle feature map ricevute come input, mantenendo inalterata la profondità del volume, e segue, tipicamente, uno o più strati di convoluzione. Gli strati di pooling sono caratterizzati da una finestra bidimensionale, che scorre su una feature map e produce, per ogni porzione che copre su di essa, un unico valore, che dipende dal tipo di pooling applicato (si veda la figura 5). Il risultato è una matrice di dimensione più piccola rispetto a quella di partenza. Inoltre questa operazione viene fatta in modo indipendente su ciascuna feature map dell'input. Tra le funzioni di pooling più comuni troviamo:

- max pooling, che seleziona in ogni regione della feature map il valore massimo;
- average pooling, che fornisce, per ogni regione della feature map, il valore medio;
- L2 pooling, che fornisce, per ogni regione della feature map, la radice della somma dei quadrati dei valori.

Come per uno strato di convoluzione, lo strato di pooling è associato ad un passo, che determina l'ampiezza dello spostamento della finestra di pooling, mentre è rara l'applicazione del padding. Dato che uno strato di pooling applica una funzione prefissata sull'input, esso non introduce alcun parametro aggiuntivo. Il vantaggio dell'uso di strati del genere è la diminuzione del numero di parametri della rete e, conseguentemente, della sua complessità computazionale.

In sintesi, per uno strato di pooling devono essere stabiliti:

- la dimensione dei filtri F;
- il passo S (*stride*).

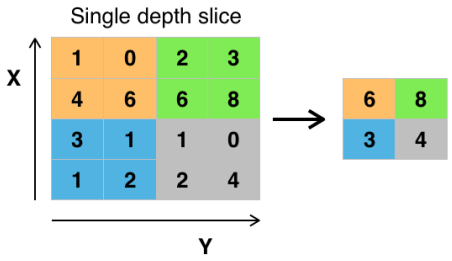


Figura 5: Strato di max pooling con finestra di dimensione 2×2 e passo 2. Per ogni porzione della matrice di input viene selezionato il valore massimo e inserito nella matrice di output. (Fonte: [5])

Dato un generico volume di input di dimensione $W_1 \times H_1 \times D_1$, tale strato produce un output di dimensione $W_2 \times H_2 \times D_2$, dove

$$W_2 = \frac{W_1 - F}{S} + 1$$
$$H_2 = \frac{H_1 - F}{S} + 1$$
$$D_2 = D_1$$

Strato completamente connesso

Dopo un certo numero di strati di convoluzione e strati di pooling si trovano, generalmente, uno o più strati completamente connessi (FC, *fully connected*), cioè strati costituiti da neuroni collegati a tutte le attivazioni dello strato precedente, come accade nelle reti neurali tradizionali. Ogni neurone in uno strato del genere, quindi, è associato ad un numero di pesi pari al numero di neuroni dello strato precedente e ad un termine bias. Gli strati completamente connessi sono deputati al "ragionamento" di più alto livello della rete.

Strato di loss

Lo strato di *loss* è l'ultimo strato di una CNN ed è quello in cui viene applicata la funzione di loss (detta anche funzione di errore), che specifica in che modo l'addestramento della rete debba penalizzare la deviazione dell'output prodotto rispetto al valore atteso. Tra le funzioni più utilizzate in questo contesto c'è la funzione Softmax, in grado di predire una classe tra K classi mutuamente esclusive:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e_j^z} \quad \text{per } i = 1, \ldots, K \text{ e } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

dove $\mathbf{z}$ è il vettore di input. Come si può intuire analizzando la formulazione matematica della funzione, ciò che essa fa è normalizzare i valori di input in

una distribuzione di probabilità. Infatti, il risultato dell'applicazione di tale funzione all'input $\mathbf{z}$ è un vettore i cui elementi sono positivi e la cui somma è pari ad 1, tra i quali quello con il valore massimo rappresenta la classe predetta.

Non sempre in una rete convoluzionale sono presenti gli strati finali completamente connessi. In tal caso la rete non è pensata per funzionare come classificatore essa stessa, bensì come estrattore di caratteristiche, le quali verranno sottoposte ad un sistema di classificazione distinto.

4 Pipeline

Nel moderno riconoscimento facciale si considera generalmente la *pipeline* da seguire come composta delle fasi di: rilevazione, pre-processamento, rappresentazione, classificazione (*detect*, *pre-process*, *represent*, *classify*). La fase di rilevazione è quella in cui, data un'immagine arbitraria, si determina se essa include o meno dei volti e, in caso affermativo, vengono fornite le posizioni e le estensioni degli stessi. La successiva fase di pre-processamento prevede operazioni di trasformazione dell'immagine volte a uniformare le immagini della collezione e renderle utilizzabili dal sistema di estrazione delle caratteristiche. Tipiche funzioni svolte in questa fase sono il ritaglio di un volto all'interno dell'immagine, il suo ridimensionamento e l'allineamento. La fase di rappresentazione è quella in cui vengono estratte le caratteristiche di un volto e codificate in modo compatto sotto forma di *embedding*, cioè un vettore descrittore. Infine, nella fase di classificazione viene effettuata una predizione sull'identità corrispondente al volto.

Spesso, quando si parla di riconoscimento facciale, si fa riferimento specificamente alla fase di estrazione delle caratteristiche ma, come appena mostrato, la sequenza di operazioni coinvolte è ben più articolata. In questa tesi prenderò in considerazione l'intera procedura, a partire dalla fase di rilevazione per finire con quella di classificazione. A dire il vero, nella maggior parte dei casi le due operazioni iniziali di rilevazione e pre-processamento vengono portate a termine tramite un unico strumento e il confine che le separa è piuttosto labile. Pertanto esse verranno discusse in maniera integrata nel seguito della trattazione. Inoltre, considerando che le dimensioni e il tipo di allineamento delle immagini da sottoporre ad una rete neurale dipendono dall'architettura di quest'ultima e dal tipo di immagini con cui essa è stata addestrata, e visto che per i miei esperimenti ho fatto uso di modelli preaddestrati, discuterò le fasi di rilevazione e pre-processamento caso per caso durante la trattazione dei singoli modelli utilizzati. La fase di classificazione, invece, è identica per tutti i modelli.

5 Modelli usati

Per la conduzione degli esperimenti ho considerato solo modelli di reti neurali di cui fossero pubblicamente disponibili delle versioni pre-addestrate. Questa decisione è stata mossa dal fatto che addestrare una rete da zero avrebbe richiesto una quantità di tempo e risorse computazionali fuori dalla mia disponibilità.

Nel seguito affronto la trattazione dei modelli che ho utilizzato, fornendo una spiegazione approfondita delle architetture di reti neurali su cui poggiano e delle loro peculiarità. Si tratta dei modelli VGGFace [32] [7], FaceNet [36] e OpenFace [4], la cui scelta è dovuta alla loro ampia diffusione e all'importanza del ruolo che hanno ricoperto negli sviluppi del riconoscimento facciale.

5.1 FaceNet

FaceNet [36] è un sistema che genera rappresentazioni vettoriali delle immagini di volti mappate nello spazio euclideo, in modo tale che la distanza rappresenti una misura della somiglianza di più volti. Fino al momento della sua pubblicazione, risalente al 2015, la prassi più comune seguita nelle CNN per il riconoscimento facciale era quella di inserire nella parte finale della rete degli strati classificatori, in grado di classificare identità su cui la rete era stata precedentemente addestrata. In questo modo la rete non è in grado di classificare identità diverse da quelle su cui è stata addestrata, ma l'output fornito dallo strato che precede quello di classificazione è comunque una buona rappresentazione di un volto, seppur ignoto. Perciò, per apprendere nuove identità, si può applicare la tecnica del *transfer learning*, con cui nuovi strati classificatori vengono sostituiti a quelli della rete originale o, in alternativa, si può sottoporre la rappresentazione ad un classificatore esterno. Questo approccio risulta poco diretto e inefficiente in quanto non garantisce che le rappresentazioni ottenute attuino una buona generalizzazione e, inoltre, la dimensione delle rappresentazioni risulta spesso molto elevata. FaceNet, al contrario, è sprovvisto di strati di classificazione ed è pensato specificamente per ottimizzare la costruzione di tali rappresentazioni, denominate *embedding* o descrittori. Si tratta di vettori monodimensionali di 128 elementi che possono essere usati in maniera semplice ed efficiente in operazioni di verifica facciale, identificazione e clustering. Per la prima è sufficiente calcolare la distanza tra due embedding e confrontarla con un valore di soglia, per la seconda si possono utilizzare degli algoritmi di classificazione come k-NN o SVM e per l'ultima algoritmi come il k-means.

FaceNet fa uso di due tipi di reti neurali: quella di Zeiler e Fergus [55] e la Inception di Szegedy *et al.* [43]. A partire dalla prima gli autori sviluppano il modello NN1, dalla seconda, invece, derivano i modelli NN2, NN3, NN4, NNS1 e NNS2. Tra di essi, quello che fa registrare prestazioni migliori è NN2. In realtà, in questo contesto i dettagli di tali architetture sono di

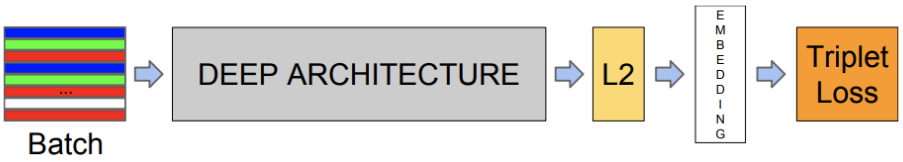


Figura 6: Struttura del modello FaceNet. Un batch di input viene inviato alla rete neurale adottata. All'output di quest'ultima viene applicata una normalizzazione L2, che determina l'ottenimento di un embedding. Questa sequenza è seguita dall'applicazione della funzione di triplet loss in fase di addestramento. (Fonte: [36])

secondaria importanza perché esse vengono trattate come scatole nere (si veda la figura 6), concentrandosi sull'apprendimento integrale dell'intero sistema.

A tal proposito viene utilizzata come funzione di errore la funzione triplet loss (si veda la figura 7), con cui si fa in modo che le immagini corrispondenti ad una stessa identità, indipendentemente dalle condizioni delle singole immagini, vengano rappresentate con embedding caratterizzati da una piccola distanza che li separa, mentre le coppie di immagini appartenenti a identità differenti vengano rappresentate con embedding separati da una maggiore distanza. Inoltre, la funzione triplet loss impone un margine tra le immagini di una stessa identità, in modo da evitare che esse vengano mappate in uno stesso punto dello spazio euclideo. Formalmente, si indichino con x una generica immagine di input e con $f(x) \in \mathbb{R}^d$ l'embedding che incorpora l'immagine in uno spazio euclideo d-dimensionale. Tale embedding è forzato a risiedere nella ipersfera d-dimensionale, cioè deve valere $\| f(x) \| = 1$. Quello che si vuole ottenere è che, per ogni identità, un'immagine àncora x_i^a sia più vicina a tutte le immagini positive x_i^p (cioè appartenenti alla stessa identità) di quanto lo è a quelle negative x_i^n (cioè appartenenti ad altre identità). In termini matematici, si vuole che valga:

$$\| f(x_i^a) - f(x_i^p) \|_2^2 + \alpha < \| f(x_i^a) - f(x_i^n) \|_2^2 \quad \forall f(x_i^a), f(x_i^p), f(x_i^n) \in T$$

dove α è il margine e T è l'insieme di tutti i possibili terzetti di immagini nel set di addestramento, con cardinalità N. Ne segue che la funzione di errore da minimizzare è:

$$L = \sum_i^N \Big[\| f(x_i^a) - f(x_i^p) \|_2^2 - \| f(x_i^a) - f(x_i^n) \|_2^2 + \alpha \Big]_+$$

In realtà, tra tutti i possibili terzetti di immagini ottenibili da un dataset ve ne sono molti che già in partenza soddisfano la condizione descritta dalla disequazione precedentemente mostrata e che, di conseguenza, contribuiscono

Figura 7: La funzione triplet loss minimizza la distanza tra un'immagine àncora e un'immagine positiva, appartenenti ad una stessa identità, e massimizza la distanza tra l'ancora e un'immagine negativa, appartenenti a identità differenti. (Fonte: [36])

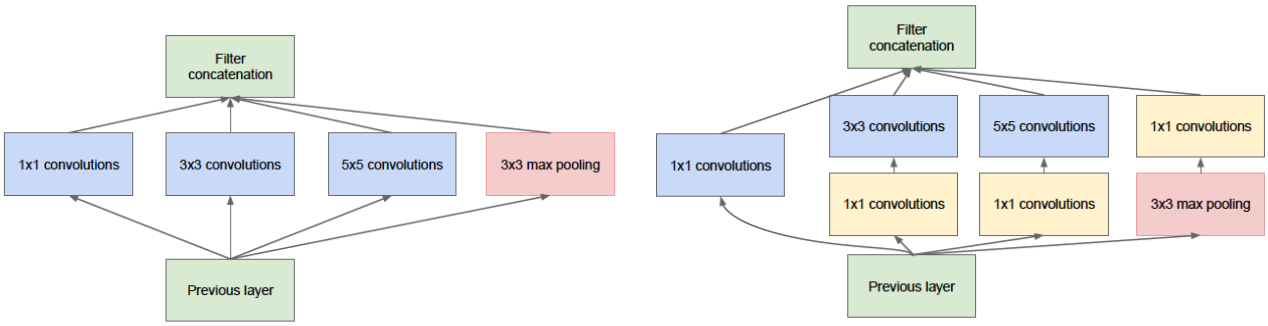
poco all'apprendimento da parte della rete, rallentandone la convergenza. Per questa ragione è necessario, al fine di addestrare velocemente la rete, selezionare i terzetti di immagini che violano la condizione in maniera più netta. Idealmente, fissata un'ancora x_i^a, bisognerebbe individuare l'immagine positiva x_i^p che massimizzi $\| f(x_i^a) - f(x_i^p) \|_2^2$ (*hard positive*) e l'immagine negativa x_i^n che minimizzi $\| f(x_i^a) - f(x_i^n) \|_2^2$ (*hard negative*). Nella pratica, però, applicare questa tecnica all'intero dataset di addestramento non è computazionalmente realizzabile, oltre al fatto che l'eventuale presenza di immagini erroneamente etichettate o di scarsa qualità influenzerebbe eccessivamente la composizione dei terzetti. La soluzione individuata dagli autori di FaceNet è stata quella di restringere la ricerca all'interno dei singoli mini-batch del dataset. Inoltre, in questa versione "allentata" della tecnica, essi non selezionano l'immagine hard positive per ogni ancora, bensì considerano tutte le coppie ancora-immagine positiva e, per ciascuna, individuano la hard negative, riducendo la complessità computazionale.

Nel prossimo paragrafo viene descritto il modello Inception, utilizzato come base in FaceNet per ottenere il modello NN2, mettendo in mostra le lievi differenze presenti tra i due.

5.1.1 Inception

L'architettura Inception [43] si basa sull'idea di utilizzare, sull'output di un certo strato, più filtri di dimensioni diverse simultaneamente e concatenare i loro risultati in un unico volume di output. Secondo questo principio viene assemblato il modulo Inception, costituito di 4 rami paralleli, di cui tre presentano filtri di dimensione, rispettivamente, 1×1, 3×3 e 5×5, e il rimanente presenta uno strato di max pooling, come mostrato in figura 8a.

Il problema che il modulo appena formulato presenta è il suo grande costo dal punto di vista computazionale, per risolvere il quale vengono introdotte delle convoluzioni 1×1 (ispirate da [26]) prima degli strati di convoluzione 3×3 e 5×5 al fine di ottenere una riduzione dimensionale, come mostrato



(a) *Versione base* **(b)** *Versione con riduzioni dimensionali*

Figura 8: Modulo Inception. (Fonte: [43])

nella figura 8b. L'idea dietro le convoluzioni 1×1 è di mantenere l'altezza e la larghezza dell'input inalterate e ridurre il numero di canali attraverso l'applicazione di un numero appropriato di filtri. Viene anche inserito uno strato di convoluzione 1×1 dopo lo strato di max pooling con compiti, stavolta, di proiezione. Le reti Inception sono composte di più blocchi come quello appena definito, impilati uno sopra l'altro ed intervallati occasionalmente da strati di max pooling.

La versione principale proposta da Szegedy *et al.* [43] è quella che prende il nome di GoogLeNet (in omaggio alla celebre rete LeNet [25]), costituita dalla sequenza di nove moduli Inception per un totale di 22 strati, come mostrato nella figura 9 a pagina seguente e nella tabella 1 a pagina 18. In aggiunta, questa architettura presenta due rami laterali terminati con una attivazione softmax che costituiscono dei classificatori ausiliari, introdotti al fine di migliorare le capacità di discriminazione degli strati sottostanti e di fornire una regolarizzazione ulteriore. In fase di addestramento le loro funzioni di errore vengono sommate, pesandole con un fattore di $0,3$, a quella principale per favorire l'apprendimento. Il modello NN2 di FaceNet è quasi del tutto identico al modello GoogleNet appena descritto, con differenze minimali che sono visibili nella tabella 2 a pagina 18.

Versioni successive del modello, chiamate Inception-v2 e Inception-v3, sono state proposte in [44], apportando miglioramenti nell'accuratezza e riducendo la complessità computazionale del modello originario. Ulteriori affinamenti si sono avuti in [42], che propone i modelli Inception-v4 e Inception-ResNet-v1. L'ultimo dei due è quello che viene utilizzato nella maggioranza delle implementazioni disponibili in rete del modello FaceNet, compresa quella di cui io mi servirò.

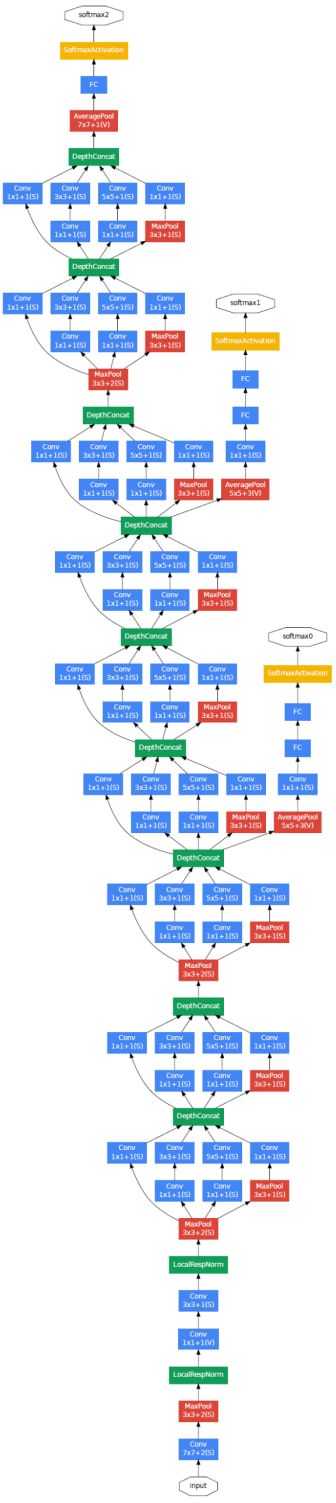


Figura 9: Rete GoogLeNet. (Fonte: [43])

Tabella 1: Architettura di GoogLeNet. "#3×3 reduce" e "#5×5 reduce" stanno a indicare il numero di filtri 1×1 presenti prima degli strati di convoluzione 3×3 e 5×5 rispettivamente. Il numero di filtri 1×1 presenti dopo gli strati di pooling è indicato, invece, nella colonna "pool proj". (Fonte: [43])

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Tabella 2: Architettura NN2 di FaceNet. Questo modello è quasi identico al modello GoogleNet. La differenza principale è data dall'uso del pooling L2 in luogo del max pooling, laddove specificato. (Fonte: [36])

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

5.1.2 Inception-ResNet-v1

L'architettura Inception-ResNet-v1 [42] nasce dall'idea degli autori di combinare il modello Inception, già di per sé piuttosto performante, con le connessioni residuali introdotte in [14] (spiegate nel paragrafo 5.2.2), con l'obiettivo di valutarne i benefici. L'architettura si compone della combinazione di vari moduli. Tre di questi sono Inception-ResNet-A (figura 10), Inception-ResNet-B (figura 11) e Inception-ResNet-C (figura 12). Essi sono costituiti, a loro volta, da un insieme di strati convoluzionali (quelli dei rami di destra delle rispettive figure) ispirati al modulo Inception originario, combinati con una connessione *shortcut*. La concatenazione dei risultati dei rami di convoluzione viene sottoposta ad un ulteriore strato di convoluzione 1×1 che ne aumenta la profondità, rendendola uguale a quella dell'input trasportato dalla connessione shortcut, operazione necessaria per poter eseguire la successiva somma dei due. Gli altri due moduli utilizzati sono chiamati Reduction-A (figura 13) e Reduction-B (figura 14), addetti ad eseguire le riduzioni dimensionali, rispettivamente, da 35×35 a 17×17 e da 17×17 a 8×8. La parte iniziale della rete, denominata "stelo" (*stem*), è visibile in figura 15, mentre la figura 16 offre una vista complessiva della rete.

5.1.3 Pre-processamento delle immagini

Nella pubblicazione di FaceNet non vengono forniti dettagli sulle tecniche adottate dagli autori per pre-processare le immagini da sottoporre alla rete convoluzionale. L'unico riferimento che viene fatto a tal proposito indica che la rete necessita di un allineamento minimale delle immagini, che consiste semplicemente in selezioni dei volti che siano molto aderenti ai bordi degli stessi. Vista la mancanza di indicazioni più specifiche, nei miei esperimenti mi sono servito della stessa MTCNN utilizzata anche per VGGFace, come descritto nel paragrafo 5.2.4, poiché questa rappresenta la scelta più comunemente seguita nelle fonti reperibili in rete.

5.2 VGGFace

VGGFace è il nome utilizzato per riferirsi a una serie di modelli per il riconoscimento facciale sviluppati dal Visual Geometry Group dell'Università di Oxford. Tra i maggiori contributi della prima pubblicazione [32], del 2015, si registra un'indicazione dettagliata di come il gruppo di ricerca è stato in grado di costruire un dataset molto esteso (2,6 milioni di immagini, 2600 identità), poi utilizzato in fase di addestramento, combinando tecniche automatizzate e intervento umano e ottenendo un valido compromesso tra purezza dei dati e tempo impiegato. Questa spiegazione risulta particolarmente rilevante poiché mostra alla comunità accademica come sia possibile costruire dataset massicci senza disporre della enorme mole di fotografie cui hanno accesso giganti del settore come Facebook e Google. Il secondo contributo rilevante è l'aver

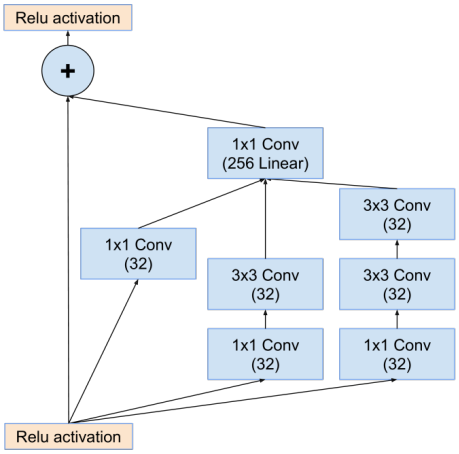


Figura 10: Inception-ResNet-A. (Fonte: [42])

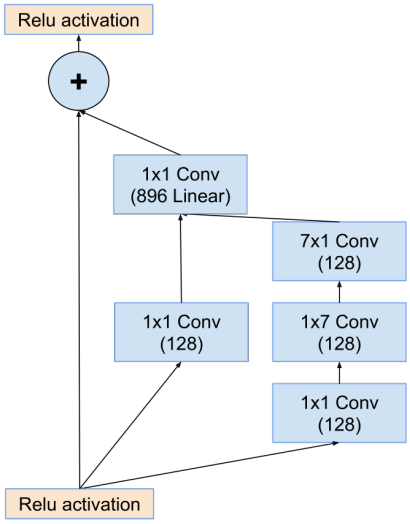


Figura 11: Inception-ResNet-B. (Fonte: [42])

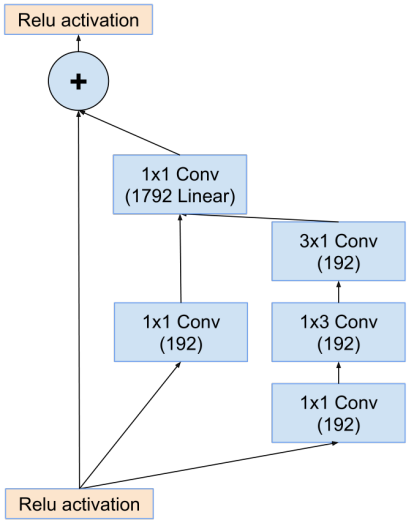


Figura 12: Inception-ResNet-C. (Fonte: [42])

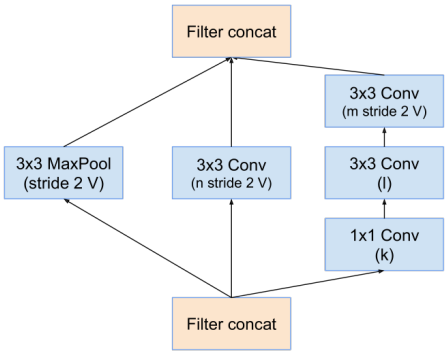


Figura 13: Reduction-A. (Fonte: [42])

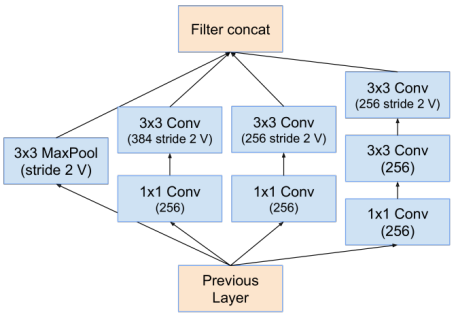


Figura 14: Reduction-B. (Fonte: [42])

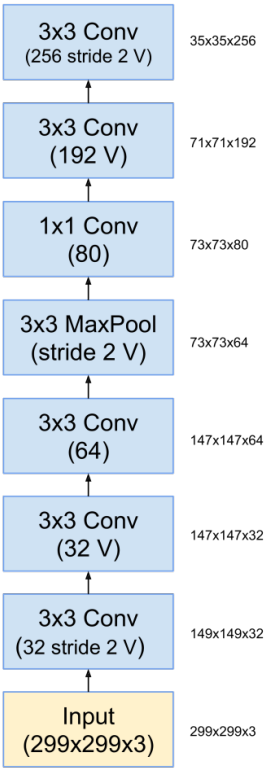


Figura 15: Stelo della rete Inception-ResNet-v1. (Fonte: [42])

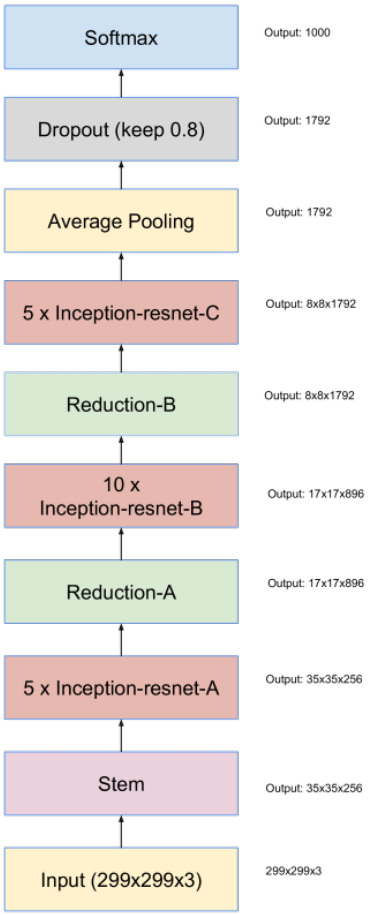


Figura 16: Struttura globale della rete Inception-ResNet-v1. (Fonte: [42])

Tabella 3: Architetture proposte in [37], di cui VGGFace [7] riprende le versioni A, B e D. I nomi degli strati convoluzionali seguono la sintassi "conv⟨dimensione del filtro⟩-⟨numero di filtri⟩". Ogni strato di convoluzione e FC è seguito da una funzione ReLU, omessa per brevità. (Fonte: [37])

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

riportato risultati di accuratezza comparabili a quelli migliori per l'epoca senza utilizzare architetture particolarmente articolate ma semplicemente conducendo una fase di addestramento appropriata. Nello specifico, le CNN da loro utilizzate sono basate sulle architetture A, B, e D di [37].

VGGFace2, del 2017 [7], introduce un dataset più ampio rispetto al predecessore (3,31 milioni di immagini, 9131 identità) e maggiormente variegato in termini di posa, età, illuminazione, etnia e professione dei soggetti. Su questo dataset vengono addestrate una rete ResNet-50 [14] e una SE-ResNet-50 [15], con cui si mettono in luce le migliorie derivate dall'uso di questo dataset rispetto ad altri..

5.2.1 VGG16

VGG16 è il nome colloquialmente utilizzato per riferirsi all'architettura che nella prima pubblicazione di VGGFace [32] è indicata come architettura D, mutuata da [37] (si veda la tabella 3). Per ottenerla, gli autori di VGGFace [32] sono partiti dalla A, la quale è addestrata inizialmente come classificatore, cioè mantenendo l'intera struttura presentata in tabella 3. Dopo questa

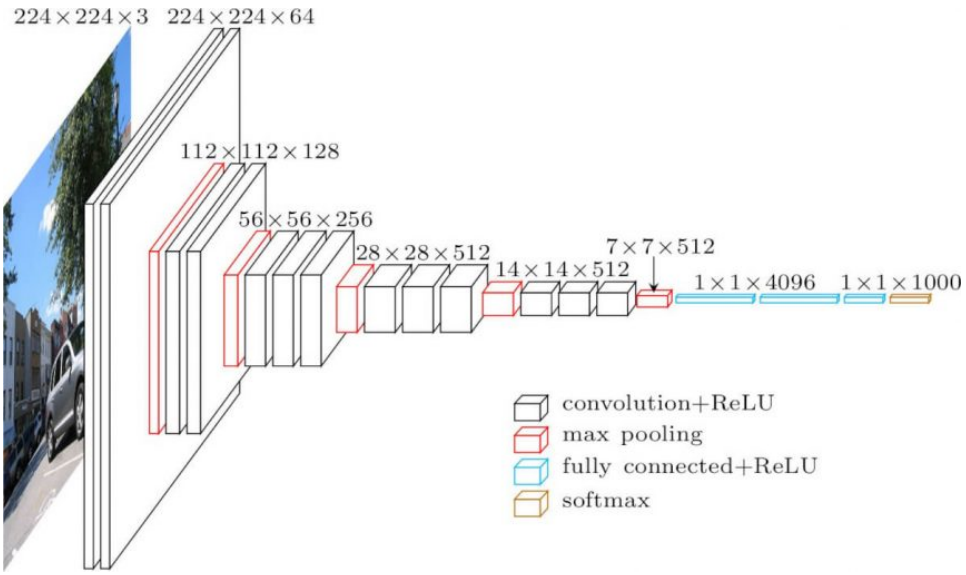


Figura 17: Architettura VGG16. (Fonte: [47])

fase iniziale l'ultimo strato della rete, deputato alla classificazione, viene rimosso, in modo tale che l'output rappresenti il descrittore di un volto, che può essere usato in compiti di verifica facendo ricorso alla distanza euclidea per compararlo ad altri. In realtà, viene applicata una seconda fase di addestramento, stavolta basata su una funzione di triplet loss, per ottimizzare la rappresentazione nello spazio euclideo di tali vettori. Il motivo per cui quest'ultima non viene applicata fin dall'inizio è che farle precedere la fase di addestramento come classificatore è risultato rendere più semplice e veloce l'intero processo di addestramento. A partire dalla versione A, si ottiene la D tramite l'aggiunta di strati completamente connessi e il fine-tuning della rete.

La fase di addestramento tramite triplet loss è del tutto simile a quella condotta in FaceNet e, per essere precisi, trae ispirazione proprio da quest'ultima. Una differenza risiede, però, nella scelta dei terzetti di immagini usati: mentre in FaceNet, per ogni coppia àncora-immagine positiva, si sceglie la hard negative, cioè l'immagine di una identità differente che ha la minore distanza dall'ancora, in VGGFace si procede selezionando un campione negativo in modo casuale tra quelli che violano la condizione di vicinanza. In questo modo, pur garantendo dei terzetti che contribuiscano all'apprendimento della rete, si riduce la complessità computazionale delle operazioni di ricerca.

Come mostrato in figura 17, l'input della rete è un'immagine di dimensione $224 \times 224 \times 3$, che attraversa una serie di strati di convoluzione i cui filtri hanno dimensione 3×3 e con un passo pari a 1. Ad essi è applicato un padding tale per cui venga preservata la dimensione spaziale del volume di input durante la convoluzione. Ad ogni strato di convoluzione è applicata una funzione ReLU. Tali strati di convoluzione sono occasionalmente intervallati

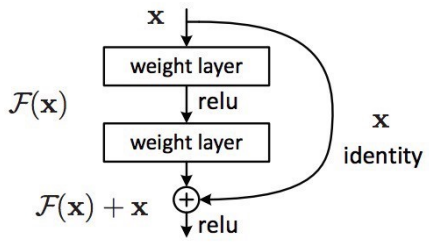


Figura 18: Blocco residuale. (Fonte: [14])

con strati di max-pooling (per un totale di cinque) caratterizzati da una finestra di dimensione 2×2 e un passo di 2. La rete è terminata da tre strati completamente connessi e un'attivazione softmax.

5.2.2 ResNet50

La peculiarità delle architetture ResNet [14] è la loro grande profondità, cosa affatto comune per l'epoca di pubblicazione, a causa delle difficoltà che si incontravano nell'addestramento di reti del genere per via dei fenomeni di decadimento/esplosione del gradiente. All'aumentare della profondità di una rete neurale, l'operazione di retropropagazione rendeva spesso il gradiente talmente ridotto da impedire l'apprendimento da parte della rete, le cui prestazioni saturavano per poi decrementare rapidamente.

La soluzione proposta dagli sviluppatori di RESNet è stata l'introduzione di una tecnica di apprendimento residuale: anziché tentare di far apprendere a un certo insieme di strati una funzione di interesse $H(\mathbf{x})$, si fa in modo, piuttosto, che esso riproduca la funzione residuale corrispondente, $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$. La funzione originale, contestualmente, diventa $F(\mathbf{x}) + \mathbf{x}$. Concretamente, questa formulazione viene ottenuta attraverso l'uso delle connessioni *shortcut* (scorciatoia), che, nel caso più semplice, sono banali collegamenti con cui si preleva l'input del blocco di strati preso in esame e lo si fornisce ad uno strato successivo dopo averlo sommato all'output del blocco in questione. Il blocco così ottenuto prende il nome di blocco residuale (si veda la figura 18).

Nelle considerazioni precedenti si è implicitamente assunto che $F(\mathbf{x})$ e $\mathbf{x}$ siano della stessa dimensione, consentendo, quindi, di essere sommati senza ulteriori elaborazioni. In realtà, non sempre è così e, in questi casi, risulta necessario operare una proiezione lineare nello shortcut, del tipo $\mathbf{y} = F(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$. Inoltre, è opportuno precisare che la formulazione residuale può essere applicata ad un qualsiasi numero di strati e anche a tipi diversi di strati, sia completamente connessi che di convoluzione.

L'utilizzo di connessioni residue risulta essere determinante perché consente di facilitare di molto il processo di apprendimento da parte di una rete, peraltro senza aggiungere ulteriori parametri né aumentare la complessità

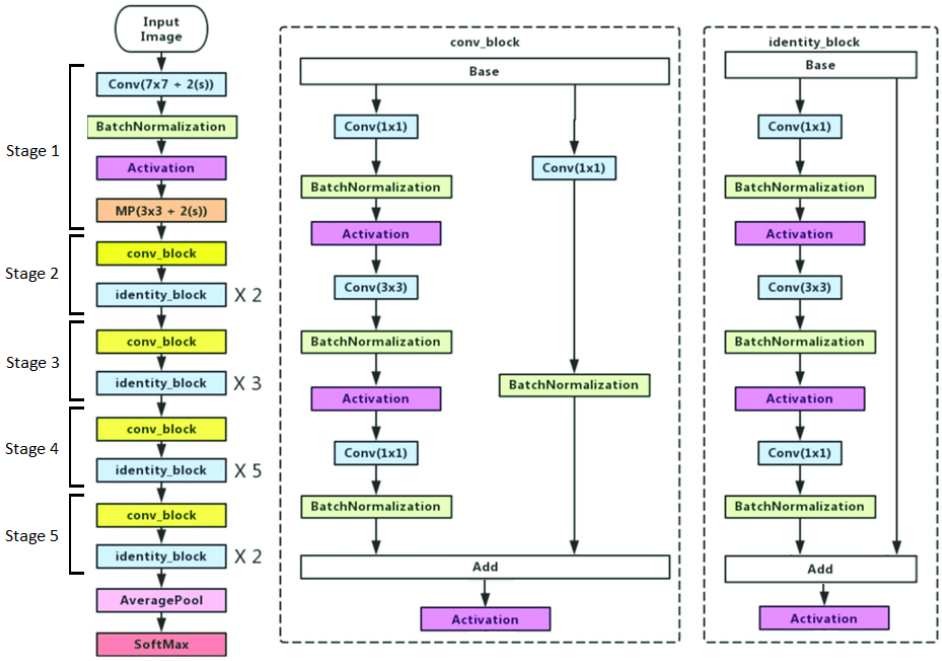


Figura 19: Architettura della rete RESNet50 con struttura globale (sinistra) e dettaglio del blocco convoluzionale (centro) e del blocco identità (destra). (Fonte:[18])

computazionale. In [14] molte sono le architetture proposte seguendo i principi finora descritti, tra le quali spicca quella costituita da 152 strati, ma per i miei esperimenti utilizzerò la versione da 50 strati, che è schematizzata in figura 19.

L'architettura RESNet50 è costituita di cinque stadi (figura 19, sinistra), i quali, fatta eccezione per il primo, sono costituiti da un primo blocco convoluzionale (figura 19, centro) seguito da più blocchi identità (figura 19, destra), in numero variabile rispetto allo specifico stadio considerato. Un blocco identità è strutturato come il generico blocco residuale presentato in precedenza e si compone di tre strati. Questo tipo di blocco viene utilizzato quando la dimensione dell'input coincide con quella dell'output e, di conseguenza, i due possono essere sommati senza ulteriori accorgimenti. Un blocco convoluzionale si distingue dal precedente per il fatto che nel ramo scorciatoia è presente uno strato di convoluzione (con un filtro di dimensione 1×1) necessario a ridimensionare l'input, cosicché esso sia coerente dimensionalmente con l'output cui deve essere sommato. Dopo il quinto stadio vengono applicati uno strato di average pooling e uno strato fully connected che, utilizzando l'attivazione softmax, esegue la classificazione.

Una schematizzazione di quanto detto è riportata nella tabella 4 di pagina 28, in cui la rete ResNet50 viene messa a confronto con la rete SE-

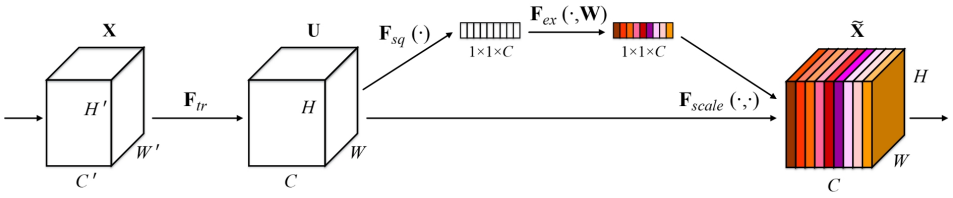


Figura 20: Blocco Squeeze-and-Excitation. (Fonte: [15])

ResNet50, descritta nel prossimo paragrafo. Si noti che in tale tabella gli stadi 2, 3, 4 e 5 vengono indicati come costituiti, rispettivamente, di 3, 4, 6, 3 strati, anziché 2, 3, 5, 2, come accade in figura 19 nella pagina precedente. Questa differenza è dovuta al fatto che nella tabella non viene fatta distinzione tra blocchi convoluzionali e blocchi identità e i due tipi vengono accorpati.

5.2.3 SE-ResNet50

L'architettura SE-ResNet50 [15], per semplicità SENet50 talvolta nel seguito, si basa sulla precedente ma è caratterizzata dall'aggiunta di blocchi SE (Squeeze-and-Excitation). Un blocco SE (si veda la figura 20) è un'unità architetturale che ricalibra in maniera adattiva le feature map prodotte in un certo strato convoluzionale tramite la modellazione esplicita di interdipendenze tra i canali di un filtro. Deriva dall'intuizione di analizzare l'influenza dei canali di un filtro sulla feature map che esso produce, in controtendenza con l'abitudine di focalizzare l'attenzione della ricerca sull'aspetto spaziale della rappresentazione dell'informazione. La prima fase prevista in un blocco SE è quella di Squeeze, che consiste nel comprimere ("spremere", letteralmente) le feature map prodotte da una convoluzione riducendole a scalari. In questo modo si ottiene un vettore monodimensionale con un numero di elementi pari al numero di filtri dello strato di convoluzione considerato e contenente valori interpretabili come descrittori di canale, cioè indicatori del grado di rilevanza dei canali corrispondenti. Si applica, poi, nella fase di Excitation, tale vettore a due strati completamente connessi, ottenendo come risultato un vettore di uguale dimensione, il quale viene utilizzato come collezione di pesi da applicare alla feature map originaria.

Formalmente, un blocco SE può essere costruito su di una trasformazione $\mathbf{F}_{\text{tr}}$ che mappa l'input $\mathbf{X} \in \mathbb{R}^{H' \times W' \times C'}$ nella feature map $\mathbf{U} \in \mathbb{R}^{H \times W \times C}$. Si consideri come trasformazione $\mathbf{F}_{\text{tr}}$ un'operazione di convoluzione e sia $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_C]$ il set di filtri. Allora l'output si può scrivere come $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_C]$, dove il generico c-esimo elemento è calcolato come:

$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s$$

Qui, $*$ denota l'operazione di convoluzione, $\mathbf{v}_c = [\mathbf{v}_c^1, \mathbf{v}_c^2, \ldots, \mathbf{v}_c^{C'}]$, $\mathbf{X} = [\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{C'}]$ e $\mathbf{u}_c \in \mathbb{R}^{H \times W}$. Per semplificare la notazione sono stati omessi i termini dei bias. Nella fase di Squeeze si comprimono le informazioni spaziali in un descrittore di canale, per ognuno dei C canali. Questo viene fatto utilizzando uno strato di global average pooling, il quale si è dimostrato fornire risultati migliori rispetto a quello di max pooling. In termini matematici, il vettore di descrittori $\mathbf{z} \in \mathbb{R}^C$ è generato contraendo $\mathbf{U}$ nelle sue dimensioni spaziali H e W in modo tale che il suo c-esimo elemento sia calcolato come:

$$z_c = \mathbf{F}_{\text{sq}}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{K} u_c(i, j)$$

Per fare uso delle informazioni aggregate, nella successiva fase di Excitation, si usa un meccanismo di gating basato su una funzione di attivazione Sigmoide:

$$s = \mathbf{F}_{\text{ex}}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z}))$$

dove δ rappresenta la funzione ReLU, $\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C}$ e $\mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}$. Per ridurre la complessità del modello e favorire la generalizzazione vengono inseriti due strati fully connected attorno alla non-linearità. Più precisamente, il primo di questi realizza una riduzione dimensionale basata sul fattore di riduzione r, mentre il secondo opera un incremento dimensionale, ritornando alla dimensione precedente di $1 \times 1 \times C$. L'output finale dell'intero blocco, $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{v}}_C]$, è calcolato scalando $\mathbf{U}$ con il vettore s, in modo tale che il generico elemento c-esimo sia:

$$\tilde{\mathbf{x}}_c = \mathbf{F}_{\text{scale}}(\mathbf{u}_c, \mathbf{s}_c) = \mathbf{s}_c \mathbf{u}_c$$

Il valore del fattore di riduzione più appropriato è stato stabilito come $r = 16$ a seguito di analisi empiriche che hanno mostrato come questa scelta rappresenti un valido compromesso tra accuratezza ottenuta e complessità del modello.

La grande flessibilità del blocco SE lo rende integrabile facilmente in tutte le architetture già esistenti. Nel caso di una rete residuale, per fare ciò si considera come trasformazione $\mathbf{F}_{\text{tr}}$ direttamente il ramo principale di un blocco residuo. In altri termini, un blocco SE-ResNet si ottiene a partire da un blocco residuale tradizionale inserendo un blocco SE prima dell'operazione di somma con lo shortcut, come mostrato in figura 21. Applicando tale modifica a tutti i blocchi residuali presenti nella rete ResNet50 è stata ottenuta la rete SE-ResNet50, come evidenziato nella tabella 4.

Il vantaggio dei blocchi SE è che, a fronte di un aumento contenuto del carico computazionale e del numero di parametri, garantisce significativi miglioramenti nelle prestazioni. Prendendo come esempio l'architettura ResNet e la sua controparte ottenuta con blocchi SE, la seconda richiede circa $3,87$ GFLOPs, che rappresenta un incremento relativo dello $0,26\,\%$

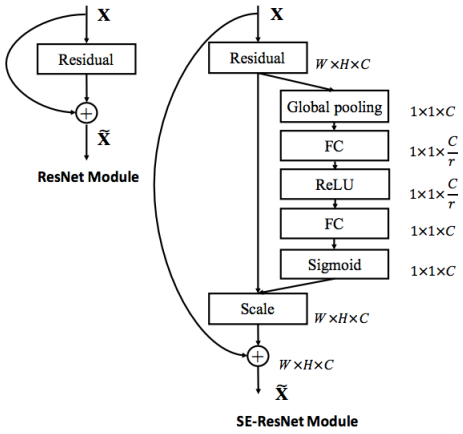


Figura 21: Lo schema del blocco residuale originale (sinistra) e del blocco SE-ResNet (destra). (Fonte: [15])

Tabella 4: Architetture ResNet-50 (sinistra) e SE-ResNet-50 (centro). Per i quattro stadi successivi a quello iniziale, tra parentesi quadre è riportata la composizione dei blocchi residuali e fuori dalle parentesi il numero di blocchi di quel tipo posti in sequenza. Per la Se-ResNet50 sono riportate anche le dimensioni dell'output dei due strati FC del modulo SE. (Fonte: [15])

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
112 × 112	conv, 7 × 7, 64, stride 2		
56 × 56	max pool, 3 × 3, stride 2		
	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \quad C=32 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28 × 28	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \quad C=32 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14 × 14	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \quad C=32 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 1024 \\ \text{conv}, 3\times3, 1024 \quad C=32 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1 × 1	global average pool, 1000-d fc, softmax		

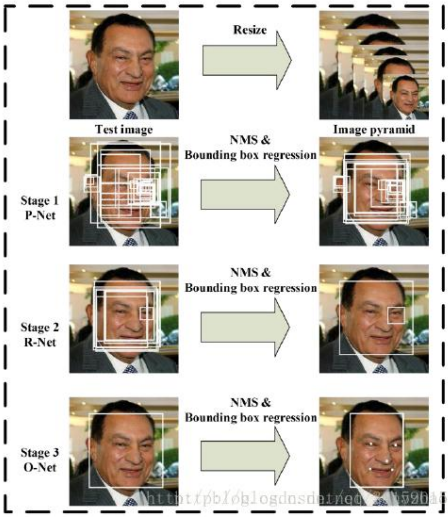


Figura 22: Funzionamento della MTCNN. (Fonte: [56])

rispetto alla prima, ma ottiene un'accuratezza ben maggiore, che addirittura si avvicina a quella della architettura ResNet101, ben più profonda.

5.2.4 Pre-processamento delle immagini

In VGGFace2 gli autori specificano che per preparare le immagini fornite alle reti, sia in fase di addestramento che in fase di test, si sono serviti della MTCNN (*Multi-task Cascaded Neural Network*) proposta in [56]. Per questa ragione anche io, nei miei esperimenti, mi sono servito di tale sistema e, ad essere precisi, non mi sono limitato ad utilizzarlo solo per VGGFace2 ma lo ho esteso anche a VGGFace e FaceNet.

Una MTCNN è un sistema multi-task costituito di più reti neurali collegate in cascata. Quello proposto in [56] si compone di 3 stadi. Data un'immagine, la prima operazione eseguita è costruire una piramide d'immagine (si veda la figura 22), cioè un insieme di copie di varie dimensioni di quella stessa immagine, al fine di poter rilevare volti di dimensioni diverse. Questa operazione è necessaria perché in seguito viene utilizzato un filtro di dimensione fissa per cercare i volti, il quale potrebbe risultare troppo piccolo o troppo grande rispetto alla dimensione del volto nell'immagine originale per poterlo contenere in modo accurato.

Ottenuta la piramide d'immagine, ognuna delle copie viene scansionata tramite un filtro di dimensione 12×12 e passo 2 che le scorre interamente, dall'angolo superiore sinistro a quello inferiore destro. Ciascuna delle porzioni di dimensione 12×12 di una copia viene inviata al primo stadio del sistema, una CNN chiamata P-Net (*Proposal Network*), mostrata in figura 23. Quest'ultima è addestrata per produrre come risultato un rettangolo di selezione

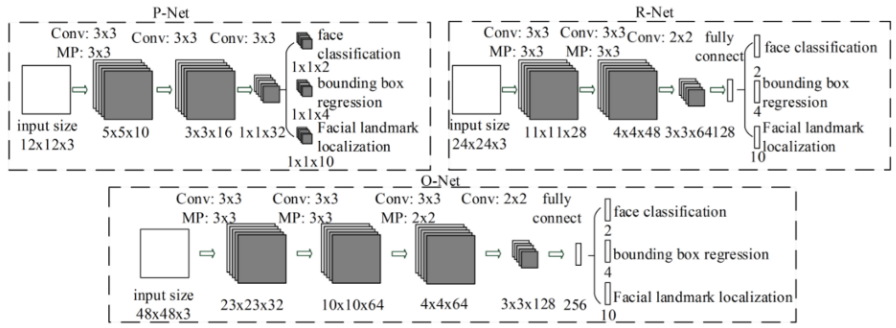


Figura 23: Architettura delle reti P-Net, R-Net e O-Net. "MP" sta per Max Pooling. Il passo negli strati di convoluzione e di pooling è 1 e 2, rispettivamente. (Fonte: [56])

attorno ad un presunto volto per ogni input e un livello di confidenza ad esso associato. In questa fase vengono eliminati i rettangoli di selezione caratterizzati da uno scarso livello di confidenza. Inoltre si riportano le coordinate dei delimitatori superstiti nel sistema di riferimento dell'immagine originale, non ridimensionata.

A questo punto, ancora molti delimitatori sono sopravvissuti e molti di essi sono tra loro parzialmente sovrapposti. Applicando la tecnica di *Non-Maximum Suppression* (NMS) si limita questo fenomeno: per ogni rettangolo si calcola l'area in comune tra esso e quello associato al massimo livello di confidenza e, se questa supera un certo valore di soglia, lo si elimina. Ciò è fatto in accordo con l'idea che se molteplici delimitatori hanno un'area in comune piuttosto estesa, presumibilmente fanno riferimento ad uno stesso volto e sono, quindi, ridondanti. Non si procede direttamente a selezionare come delimitatore definitivo quello con la confidenza maggiore perché così facendo, se l'immagine presentasse più volti, andrebbero eliminati tutti eccetto uno. In presenza di volti non interamente contenuti nell'immagine, l'array che rappresenta la regione interna al delimitatore viene reso tramite zeri nelle posizioni esterne al bordo dell'immagine. Gli array vengono poi ridimensionati ad una dimensione di 24×24, normalizzati a valori tra -1 e 1 e forniti al secondo stadio del sistema.

Qui, la rete R-Net (*Refine Network*) ripete la procedura dello stadio precedente, operando un raffinamento dei risultati e fornendo dei delimitatori più accurati unitamente ai livelli di confidenza. Di nuovo, vengono eliminati i delimitatori associati ad uno scarso livello di confidenza e soppressi quelli ridondati. Quelli rimanenti vengono portati ad una dimensione di 48×48 e inviati all'ultimo stadio.

La O-Net (*Output Network*) screma ulteriormente i risultati dello stadio precedente, fornendo rettangoli di selezione ancora più accurati e i rispettivi livelli di confidenza. Inoltre, a differenza delle prime due reti, produce come

risultato anche cinque punti chiave di un volto (gli occhi, il naso e i due angoli della bocca), utilizzabili in operazione di allineamento. Ancora, si scartano i delimitatori con minore confidenza e si applica la NMS sui rimanenti. A questo punto, a meno di errori da parte della MTCNN, si è ottenuto un unico delimitatore per ogni volto contenuto nell'immagine di partenza.

Fino al momento della pubblicazione di questo sistema, uno dei metodi più utilizzati per il rilevamento di volti era il metodo Viola-Jones [50]. Esso, però, ha mostrato numerosi limiti se utilizzato in applicazioni reali con immagini che presentassero forti variazioni. Successivi studi hanno tentato di applicare le reti neurali a questo compito, ottenendo risultati validi ma onerosi dal punto di vista computazionale e dei tempi. MTCNN, al contrario, fornisce risposte in tempo reale e ha ottenuto i risultati migliori sui più comuni benchmark per il rilevamento di volti, come FDDB [17] e WIDER FACE [53], affermandosi come uno dei modelli più performanti e ancora oggi utilizzati.

5.3 OpenFace

OpenFace [4] è un progetto nato con l'obiettivo di ridurre il divario tra i livelli di accuratezza raggiunti dai sistemi di riconoscimento facciale disponibili pubblicamente e quelli privati, intento raggiunto con successo visti i risultati su LFW [16] vicini a quelli umani. Esso si presta molto bene anche all'uso in sistemi mobili, considerati i tempi di esecuzione molto ridotti che lo contraddistinguono. OpenFace si ispira fortemente a FaceNet, e come questo è pensato specificamente come sistema per l'estrazione di caratteristiche da immagini di volti, che possono, poi, essere utilizzate in operazioni di classificazione, verifica e clustering. Anch'esso fa uso della funzione di triplet loss, nella stessa formulazione proposta da FaceNet.

La specifica architettura di rete neurale che viene riportata nella pubblicazione è indicata con il nome NN4.small2 ed è ottenuta dalla rete NN4 di FaceNet [36] tramite la rimozione degli strati 4b, 4c e 4d e la riduzione della dimensione degli strati 5a e 5b (si veda la tabella 5). Queste modifiche sono apportate dagli autori al fine di diminuire il numero di parametri, in accordo con la dimensione ridotta del dataset da loro utilizzato per l'addestramento rispetto a quello utilizzato da FaceNet. Per l'esattezza, il dataset da loro scelto per l'addestramento è dato dalla combinazione dei due più grandi dataset etichettati destinati alla ricerca, CASIA-WebFace [54] e FaceScrub [31], per una dimensione complessiva di circa 500 000 immagini, contro gli oltre 100 milioni utilizzati da FaceNet [36] e i 4,4 milioni utilizzati da DeepFace [45]. Nel repository GitHub del progetto [3] sono disponibili anche le reti NN2 e NN4, implementazioni delle omonime architetture descritte in FaceNet, e NN4.small1, un'altra versione ridotta della NN4. Per i miei esperimenti utilizzerò la versione più diffusa nonché performante, cioè la NN4.small2.

La procedura di addestramento seguita in OpenFace prevede la suddivisione del dataset in mini-batch, ognuno dei quali costituito da al più $P = 20$

Tabella 5: Architettura di NN4.small2. (Fonte: [4])

type	output size	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj
conv1 ($7 \times 7 \times 3, 2$)	$48 \times 48 \times 64$						
max pool + norm	$24 \times 24 \times 64$						m 3×3, 2
inception (2)	$24 \times 24 \times 192$		64	192			
norm + max pool	$12 \times 12 \times 192$						m 3×3, 2
inception (3a)	$12 \times 12 \times 256$	64	96	128	16	32	m, 32p
inception (3b)	$12 \times 12 \times 320$	64	96	128	32	64	ℓ_2, 64p
inception (3c)	$6 \times 6 \times 640$		128	256,2	32	64,2	m 3×3, 2
inception (4a)	$6 \times 6 \times 640$	256	96	192	32	64	ℓ_2, 128p
inception (4e)	$3 \times 3 \times 1024$		160	256,2	64	128,2	m 3×3, 2
inception (5a)	$3 \times 3 \times 736$	256	96	384			ℓ_2, 96p
inception (5b)	$3 \times 3 \times 736$	256	96	384			m, 96p
avg pool	736						
linear	128						
ℓ_2 normalization	128						

immagini a persona per $Q = 15$ persone. Le $M \simeq PQ$ immagini di ogni mini-batch vengono inviate alla rete neurale, così da ottenere M embedding. Si prendono, poi, tutte le coppie àncora-positiva in modo da ottenere $N = Q\binom{P}{2}$ terzetti. Su di questi si calcola la triplet loss e, utilizzando lo SGD (Stochastic Gradient Descent), si retropropaga il gradiente di ogni immagine nella rete. Se, data una coppia ancora-positiva, non è possibile individuare un campione negativo all'interno del margine α, semplicemente non si usa alcuna immagine negativa.

5.3.1 Pre-processamento delle immagini

Le immagini utilizzate dagli sviluppatori per l'addestramento della rete sono state preliminarmente allineate in modo da rendere quanto più possibile uniforme la posizione dei punti chiave dei volti tra le varie fotografie. Questa operazione è stata necessaria per via della dimensione relativamente esigua del dataset da loro utilizzato e con lo scopo di ridurre la dimensione dello spazio degli input. La tecnica adottata per l'allineamento è una trasformazione affine bidimensionale volta a far apparire gli occhi e il naso dei volti in posizioni simili in tutte le immagini. Per mettere in pratica tale operazione fa uso del rilevatore di *landmarks* (punti di riferimento) offerto nella libreria dlib [23], che è un'implementazione del metodo descritto in [21]. Data l'immagine di un volto, il rilevatore individua 68 punti di riferimento e successivamente viene operata una trasformazione affine dell'immagine sulla base dei tre punti corrispondenti agli angoli esterni degli occhi e alla punta del naso. In altre parole, viene manipolata l'immagine in modo tale che questi punti di riferimento siano il più possibile vicini alle posizioni medie calcolate per questi punti chiave (si veda la figura 24). Contestualmente l'immagine viene ridimensionata e tagliata in accordo ai bordi più esterni dei punti di

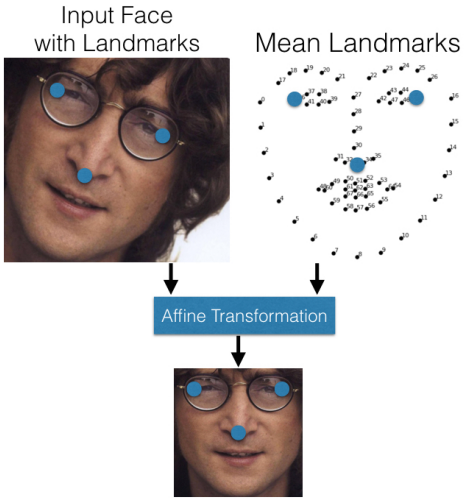


Figura 24: Trasformazione affine utilizzata da OpenFace. La trasformazione è basata sui punti di riferimento contrassegnati in blu e l'immagine finale è tagliata in accordo ai punti di riferimento più esterni. L'immagine è anche ridimensionata ad una dimensione di 96×96 pixel. (Fonte: [4])

riferimento, ottenendo una dimensione di 96×96. Proprio perché la rete è stata addestrata su immagini pre-processate in questo modo, è necessario applicare lo stesso tipo di elaborazione anche sulle immagini su cui si vogliono condurre i test.

6 Classificazione

Dopo aver ottenuto i vettori descrittori per le immagini del dataset, questi possono essere utilizzati per compiti di verifica, identificazione (o classificazione) o clustering. Nel mio caso, essendo l'identificazione l'oggetto dell'interesse, è stato necessario individuare il tipo di classificatore da utilizzare. Normalmente, in situazioni del genere si hanno due strade percorribili: è possibile utilizzare degli strati classificatori appositamente pensati posti al termine di una rete neurale o alternativamente un classificatore esterno, ottenuto, ad esempio, con tecniche di machine learning come SVM o k-NN.

Ho deciso di non affrontare la prima strada perché due dei modelli da me utilizzati, FaceNet e OpenFace, sono costruiti appositamente per fungere da estrattori di caratteristiche piuttosto che come strumenti per la classificazione e, quindi, sono sprovvisti di tali strati classificatori. Il modello VGGFace, inoltre, seppur dotato di questi strati, avrebbe richiesto l'applicazione della tecnica del *transfer learning* al fine di adattare la rete al dataset da me utilizzato rispetto a quello su cui è stato addestrato, il che rappresenta un'operazione poco agevole e richiede delle competenze piuttosto avanzate. Al contrario, utilizzare anche VGGFace come estrattore di caratteristiche non

è un problema, perché è sufficiente prelevare l'output che esso produce appena prima degli strati addetti alla classificazione. Tutti e tre i modelli sono quindi utilizzabili come estrattori di caratteristiche, che poi si prestano bene all'uso con k-NN e SVM. Peraltro, gli stessi autori di FaceNet [36] menzionano come possibile strumento di classificazione un k-NN, mentre quelli di OpenFace [4] testano il loro stesso modello con una SVM. Questa scelta consente anche di condurre i test in condizioni omogenee per tutti e tre i modelli, semplicemente predisponendo una stessa SVM o k-NN e sottoponendo loro, di volta in volta, gli embedding ottenuti dai tre. Questo aspetto è particolarmente rilevante se si considera che era mia intenzione fare una valutazione comparata delle reti sulla base della qualità degli embedding da loro prodotti, e questo richiedeva che la fase di classificazione venisse condotta in contesti analoghi per tutti e tre i modelli.

6.1 SVM

Le SVM (*Support-Vector Machines*, macchine a vettori di supporto) sono modelli di apprendimento supervisionato utilizzabili in problemi di classificazione e di analisi della regressione. Nella formulazione originaria rappresentano un classificatore lineare binario non probabilistico: dato un insieme di campioni di addestramento, ognuno etichettato come appartenente ad una di due categorie (più propriamente dette classi), una SVM realizza un algoritmo in grado di predire la categoria di appartenenza di un nuovo campione. Siano i campioni rappresentati come vettori p-dimensionali, la SVM tenta di posizionare un iperpiano $(p-1)$-dimensionale in modo tale che esso abbia il maggior margine di separazione possibile rispetto al campione più vicino di ognuna delle due classi, come mostrato in figura 25. I campioni più vicini all'iperpiano prendono il nome di vettori di supporto (*support vectors*) ed è da essi che dipende la posizione dell'iperpiano stesso. In fase di test, quando un nuovo campione deve essere classificato, ad esso viene assegnata una classe in base alla posizione in cui ricade rispetto all'iperpiano.

Non sempre i campioni sono linearmente separabili nello spazio in cui sono definiti, in questi casi è possibile usare il "trucco" del kernel, che consiste nel passare ad uno spazio dimensionalmente superiore per eseguire la suddivisione. Per fare ciò si deve definire una funzione kernel che mappi i vettori dallo spazio originale a quello aumentato. Altrimenti si parla semplicemente di kernel lineare.

Ad una SVM è associato un parametro detto di regolarizzazione, C, che regola il livello di compromesso tra la classificazione errata dei campioni e il margine minimo tra l'iperpiano e i campioni stessi. Usando un kernel lineare in contesti in cui i campioni non sono linearmente separabili, aumentare C determina una migliore classificazione nei campioni di addestramento ma al costo di un minor margine minimo tra iperpiano e campioni. Viceversa,

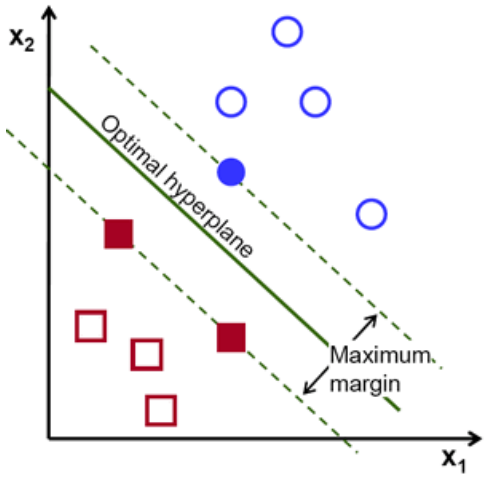


Figura 25: Funzionamento dell'algoritmo SVM in un problema di classificazione binario in cui è possibile una separazione lineare. L'iperpiano ottimo viene posizionato in modo da avere il massimo margine di separazione con i campioni più vicini delle due classi, denominati vettori di supporto. (Fonte: [11])

ridurre C favorisce un margine minimo più elevato ignorando, di contro, eventuali campioni mal classificati.

Questo algoritmo è applicabile anche a problemi multi-classe, semplicemente decomponendo questi ultimi in più problemi di classificazione binaria. L'approccio da me seguito è quello "uno-contro-uno", che prevede di suddividere un problema c-classe in $c(c-1)/2$ problemi binari. Più precisamente, si considerano una alla volta tutte le coppie di classi e per ciascuna di esse viene costruito l'iperpiano ottimo e assegnata una delle due classi al campione di test, seguendo il criterio esposto per il caso binario. Terminate le coppie di classi, la classe che viene assegnata in modo definitivo al campione è quella che più volte è risultata "vittoriosa" nelle prove precedenti.

6.2 k-NN

Il k-NN (*k-Nearest-Neighbors*) è un algoritmo non parametrico utilizzabile, come le SVM, in problemi di classificazione e regressione. Dato un insieme di campioni rappresentati come vettori in uno spazio multi-dimensionale, l'algoritmo prevede, nella fase iniziale, la sola memorizzazione di tali vettori unitamente alle rispettive etichette di classe. Un nuovo vettore viene classificato semplicemente assegnandolo alla classe che presenta il maggior numero di occorrenze tra i k campioni di addestramento più vicini ad esso (si veda la figura 26). La distanza può essere una qualsiasi misura metrica, sebbene la distanza euclidea standard sia la scelta più comune. k è una costante scelta dall'utilizzatore e il suo valore ottimale dipende fortemente dagli specifici dati

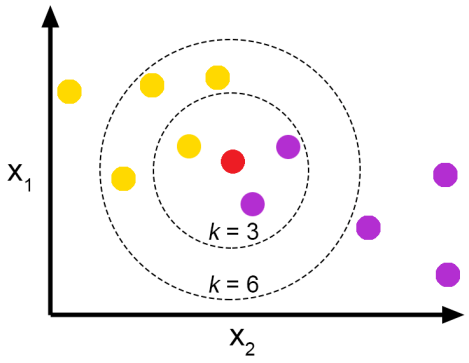


Figura 26: Funzionamento dell'algoritmo k-NN in un problema di classificazione binario. Scegliendo $k = 3$ il campione da classificare, in rosso, viene assegnato alla classe in viola, mentre scegliendo $k = 6$ esso viene assegnato alla classe in giallo. (Fonte: [20])

su cui si opera. In generale, aumentare questo valore diminuisce gli effetti del rumore ma, di contro, rende meno distinti i confini di classificazione, e viceversa. Questo algoritmo si distingue per essere non generalizzante, cioè non fa alcuna ipotesi sulla distribuzione dei dati che analizza. Inoltre esso può essere applicato senza alcuna differenza in problemi binari e multi-classe.

6.3 Validazione incrociata

Negli esperimenti che ho condotto ho reputato opportuno applicare la tecnica della validazione incrociata (*cross-validation*) al fine di ottenere risultati con un maggior grado di generalizzazione. Infatti, seguendo un metodo classico di validazione, in cui il dataset viene suddiviso in modo statico in un sottoinsieme di addestramento e uno di test, avrei ottenuto risultati poco indicativi della reale capacità del modello di predire nuovi campioni, poiché fortemente dipendenti dalla specifica suddivisione realizzata. In altre parole, il modello sarebbe potuto incorrere in *overfitting*. Tramite la convalida incrociata si argina il problema eseguendo più prove su suddivisioni sempre diverse del dataset tra sezione di addestramento e di test. Questo consente di ottenere risultati che diano una migliore indicazione dell'abilità del modello di generalizzare rispetto ad un nuovo dateset indipendente.

La versione da me adottata di questo metodo è la k-fold cross-validation stratificata. Essa consiste nel suddividere il dataset in k sottoinsiemi, dei quali $k-1$ vengono destinati all'addestramento e il rimanente al test. Questa operazione è ripetuta k volte, utilizzando ognuno dei sottoinsiemi esattamente una volta come set di test. Per ottenere un'unica stima si può, poi, calcolare la media dei k risultati ottenuti. La versione stratificata prevede che venga mantenuta la proporzione tra insieme di addestramento e insieme di test anche internamente alle singole classi. Questo significa che il numero di

campioni inseriti nel dataset di addestramento e quello dei campioni inseriti nel dataset di test sono uguali da classe a classe.

7 Esperimenti

In questo paragrafo vengono descritte da un punto di vista pratico le procedure che ho seguito per predisporre il necessario alla conduzione degli esperimenti, a partire dalla preparazione del dataset fino ai metodi di classificazione adottati. Vengono poi presentati i risultati prodotti da tali esperimenti e fatte considerazioni su di essi.

Tutti gli esperimenti sono stati condotti sulla piattaforma cloud Google Colaboratory [12] su una CPU da 2,20 GHz e 13 GB di RAM.

7.1 Costruzione del dataset

Negli ultimi decenni numerosi dataset per il riconoscimento facciale sono stati pubblicati, con una chiara tendenza ad aumentare sempre più di dimensione. Inoltre, si è via via passati da dataset costruiti ad hoc in condizioni controllate a dataset ottenuti da situazioni di vita reale. I primi vengono detti vincolati (*constrained*) poiché costituiti da immagini accomunate da condizioni costanti e sono pubblicati al fine di facilitare lo studio di parametri specifici nel riconoscimento facciale. Ad esempio, un dataset del genere potrebbe comporsi di fotografie scattate con le stesse condizioni di illuminazione o con una stessa posizione del volto. I secondi, invece, vengono detti non vincolati (*unconstrained*) e sono costruiti con lo scopo di simulare varie situazioni di vita comune. Non appena le prestazioni garantite dall'uso di un dataset raggiungono la saturazione, ne vengono sviluppati di nuovi al fine di supportare la ricerca nel campo del riconoscimento facciale. Ad essere precisi, si può dire senza esagerare che è proprio la progettazione di nuovi dataset che in gran parte indica la via da seguire nella ricerca in questo settore e un fattore determinante della qualità di un modello di riconoscimento facciale è proprio il dataset su cui esso è stato addestrato. Similmente, la qualità di un dataset usato in fase di test è determinante per ottenere una prospettiva precisa delle possibili prestazioni di un modello in applicazioni concrete.

Tra i più diffusi dataset pensati per la fase di test troviamo senza dubbio LFW(*Labeled Faces in the Wild*) [16], risalente al 2007, che è stato uno dei primi non vincolati, ideato appositamente per fornire immagini che coprissero un ampio raggio di condizioni tipicamente incontrate dalle persone nelle loro vite di tutti i giorni. Sulla scia di LFW, molti altri dataset basati sul concetto *in the wild* sono stati sviluppati negli anni a seguire, come MS-Celeb-1M [13], Megaface [22] [30] e YTF [52].

Per la conduzione dei miei esperimenti ho deciso di utilizzare un sottoinsieme di immagini ricavato dal dataset LFW. Questa scelta è dipesa dal fatto che tale dataset è probabilmente il più diffuso e utilizzato dai gruppi

di ricerca per condurre prove di riconoscimento facciale. Pertanto, risultati ottenuti su di esso sono più facilmente interpretabili alla luce della grande disponibilità di pubblicazioni che lo riguardano. Inoltre, LFW è un dataset non vincolato e questo consente di simulare in maniera appropriata il tipo di immagini derivanti da fotosegnalamento su cui i modelli verranno in futuro testati.

LFW è pensato primariamente per compiti di verifica facciale piuttosto che di identificazione e, a tal proposito, propone un *benchmark* unificato, che consiste in un elenco di coppie positive e negative sulle quali effettuare l'operazione di verifica. I gruppi di ricerca, in questo modo, possono confrontare i propri risultati in modo diretto con quelli altrui perché ottenuti nelle stesse condizioni. Tuttavia, considerando la grande varietà delle immagini che lo caratterizza, LFW può essere utilizzato efficacemente anche per operazioni di classificazione. L'unica differenza è che non è disponibile un mezzo di valutazione unificato per la comparazione dei risultati.

Per la costruzione del mio dataset ho individuato la quantità di 500 immagini come una dimensione valida per ottenere risultati sufficienti a garantire una buona capacità di generalizzazione, senza essere eccessivamente estesa da risultare scomoda da gestire in fase di test. Di queste 500 immagini ne sono presenti 20 per ciascuna di 25 identità, scelta che garantisce un buon livello di assortimento nelle caratteristiche dei volti. A questo scopo, nella scelta delle identità ho anche seguito un criterio di bilanciamento di età, etnia e sesso, vincolato parzialmente dai limiti intrinsechi che, sotto questo punto di vista, LFW presenta.

7.2 Implementazioni di CNN adottate

VGGFace e OpenFace rendono disponibili, nei rispettivi repository GitHub, i modelli descritti nelle proprie pubblicazioni, ma essi sono realizzati con framework come Torch [10] o Caffe [19]. Considerando che il framework di cui io mi sono servito è Keras [9], per via della sua maggiore diffusione e del supporto fornito, è stato necessario reperire versioni di quei modelli convertite per l'uso con tale framework. Per VGGFace ho optato per l'implementazione proposta nel repository GitHub `keras-vggface` [29], che contiene le tre architetture VGG16, ResNet50 e SENet50, mentre per OpenFace ho utilizzato quella del repository `Keras-OpenFace` [1]. Per FaceNet, invece, nessuna implementazione ufficiale dei modelli è resa disponibile dagli autori ma sono presenti, in rete, molte versioni realizzate basandosi su di esso, delle quali `keras-facenet` [46] è stata la mia scelta. Nella tabella 6 sono riportate le caratteristiche principali delle implementazioni che ho usato.

Tra le osservazioni che vanno fatte prima di proseguire troviamo il fatto che l'implementazione di VGG16 di cui mi sono servito prevede che l'output venga prelevato prima dei tre strati completamente connessi posti al termine della rete (si veda la tabella 3 di pagina 22) e non dopo di essi, come previsto

Tabella 6: Caratteristiche delle implementazioni di CNN da me utilizzate. Le architetture di VGGFace sono accomunate da una stessa fase di pre-processamento e da un'uguale dimensione delle immagini di input.

Sistema	Architettura	Pre-processamento	Numero parametri	Dimensione input	Dimensione embedding
VGGFace	VGG16	MTCNN	14 714 688	$224 \times 224 \times 3$	512
	ResNet50		23 508 032		2048
	SENet50		26 039 024		2048
FaceNet	Inception-ResNet	MTCNN	22 779 312	$160 \times 160 \times 3$	128
OpenFace	NN4.small2	dlib + OpenCV	3 733 968	$96 \times 96 \times 3$	128

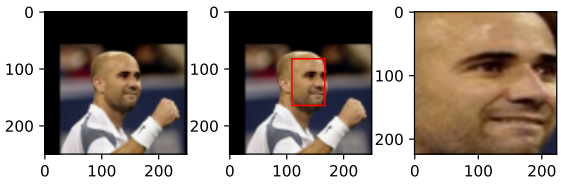
nell'articolo originale [32]. Da ciò segue che il descrittore ottenuto è un vettore di 128 elementi e non 1024. Inoltre l'implementazione di SENet50 presenta dei blocchi SE dalla struttura leggermente diversa rispetto a quella descritta nell'articolo originale [15] (visibile nella figura 21 a pagina 28), caratterizzata da strati di convoluzione in luogo dei due strati completamente connessi.

7.3 Conduzione degli esperimenti

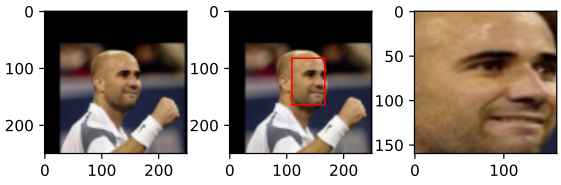
La prima fase nella conduzione degli esperimenti è stata quella della preparazione delle immagini, che combina la rilevazione dei volti al loro interno e il pre-processamento. Per le architetture di VGGFace (VGG16, ResNet50 e SENet50) ho fatto uso della MTCNN pubblicata nel repository GitHub `mtcnn` [33], che è una realizzazione del sistema descritto nel paragrafo 5.2.4. In questo modo ho ottenuto, per ogni immagine, le coordinate di un rettangolo di selezione, sulla base delle quali ho ritagliato i volti. Ho anche ridimensionato ogni immagine in accordo con le dimensioni spaziali richieste per l'input delle tre reti convoluzionali, pari a 224×224 (la dimensione di profondità, invece, è conforme già in partenza con quanto richiesto dalla rete, essendo tutte le immagini RGB). Ho condotto questa operazione un'unica volta in quanto tutte e tre le architetture di VGGFace necessitano di immagini pre-processate allo stesso modo.

Ho utilizzato lo stesso sistema anche per la preparazione delle immagini per FaceNet, con l'unica differenza che il ridimensionamento che ho applicato in questo caso è stato di 160×160, come richiesto dalla rete. Sia FaceNet che VGGFace non richiedono che le immagini siano allineate in modo esatto tra loro, quindi non ho applicato alcun tipo di elaborazione ulteriore. Per questa ragione non mi sono servito delle indicazioni fornite dalla MTCNN riguardo le posizioni dei punti chiave dei volti.

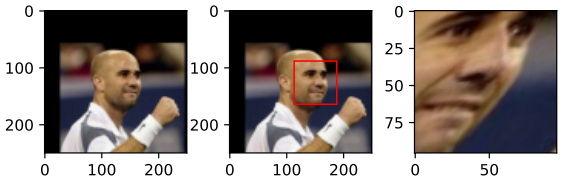
Per OpenFace, invece, ho seguito il metodo descritto dagli autori, utilizzando lo strumento che loro stessi hanno reso disponibile nel loro repository [3], il quale si basa su metodi forniti dalle librerie dlib [23] e OpenCV [6]. Ini-



(a) *VGGFace*



(b) *FaceNet*



(c) *OpenFace*

Figura 27: Immagini originale (sinistra), con applicazione di rettangolo di selezione (centro) e pre-processata (destra).

zialmente, tramite dlib vengono individuati, per un volto, un rettangolo di delimitazione e le coordinate di 68 punti di riferimento al suo interno contenuti. Di questi vengono considerati, poi, i tre relativi agli angoli esterni degli occhi e al naso e, tramite la libreria OpenCV, viene calcolata una trasformazione affine che consideri come coordinate di partenza quelle dei tre punti appena menzionati nell'immagine originale e come coordinate di destinazione quelle delle posizioni medie di tali punti di riferimento. Contestualmente l'immagine viene anche resa conforme alle dimensioni specificate, ossia 96×96.

Un confronto diretto tra le immagini risultanti dalle elaborazioni applicate nei tre casi è visibile in figura 27, che fa riferimento ad una stessa immagine di partenza. Le immagini pre-processate per l'uso con VGGFace e FaceNet sono tra loro identiche, in virtù della stessa tecnica adottata, ma differiscono nelle dimensioni. Quella destinata all'uso con OpenFace, invece, è sensibilmente diversa: la trasformazione affine che le è stata applicata ha fatto sì che i punti chiave considerati nel volto venissero spostati in posizioni prestabilite. In questo caso specifico, essendo il volto non orientato frontalmente nell'immagine originale, tale operazione ha prodotto un'immagine altamente deformata.

La fase successiva è stata quella di estrazione delle caratteristiche dalle

immagini dei volti. A tal proposito, mi sono limitato a sottoporre ad ognuna delle architetture testate le immagini preparate come appena descritto. Come output, ho ottenuto da parte dell'architettura VGG16 un vettore di 512 elementi per ogni immagine, dalle architetture ResNet50 e SENet50 vettori di 2048 elementi e dalle architetture Inception-ResNet (FaceNet) e NN4.small2 (OpenFace) vettori di 128 elementi.

Per l'ultima fase, quella di classificazione, ho predisposto un classificatore k-NN e un classificatore SVM. Per il primo, essendo le classi separate spazialmente in maniera piuttosto marcata, ed essendo quindi trascurabili gli occasionali effetti di rumore, ho optato per un valore di $k = 1$. In questa maniera, ad ogni nuovo campione viene assegnata come classe di appartenenza quella del campione ad esso più vicino. Per quanto riguarda, invece, la metrica, ho seguito la prassi comune e adottato la distanza euclidea. Per il secondo, invece, ho optato per una funzione di kernel lineare in quanto scelta più comune in casi del genere e per un valore del parametro $C = 1$ perché rappresenta una scelta ragionevole nel bilanciamento tra errori di classificazione e ampiezza del margine minimo, alla luce del fatto che, nel mio caso, ho campioni scarsamente affetti da rumore.

Per la validazione incrociata ho scelto una suddivisione stratificata per mantenere la proporzione nelle classi e un valore di $k = 5$ in funzione della numerosità del dataset e delle classi. Questa scelta determina, per ogni ripetizione, un dataset di addestramento di 400 immagini e uno di test di 100 di cui, rispettivamente, 16 e 4 per ogni classe. Ho, inoltre, applicato un rimescolamento dell'ordine dei campioni in ogni classe prima dell'applicazione della convalida incrociata, così da ottenere split diversi ad ogni esecuzione della procedura. Il parametro di valutazione che ho utilizzato è stato l'accuratezza, calcolata come rapporto tra il numero di volti correttamente identificati e il totale di quelli sottoposti come test, cioè 100.

Ho ripetuto l'intera sequenza di operazioni appena descritte, dalla preparazione delle immagini alla classificazione, dieci volte per ognuna delle cinque architetture testate, al fine di avere un campione sufficientemente numeroso per poter considerare i risultati ottenuti come attendibili. Ho registrato, separatamente, i tempi impiegati in ciascuna fase per avere un ulteriore metro di valutazione delle prestazioni degli algoritmi. Per i tempi relativi alle fasi di pre-processamento ed estrazione delle caratteristiche ho applicato una normalizzazione ad una singola immagine: quelli presentati in seguito, quindi, sono i tempi che ogni modello comporta, in media, per un'immagine. Per la fase di classificazione, invece, i tempi riportati sono quelli complessivi, cioè riferiti all'intero dataset, e comprensivi sia dell'operazione di addestramento che di quella di test.

Tabella 7: Accuratezza ottenuta nella classificazione con SVM e k-NN, espressa tramite valore medio e deviazione standard delle dieci misurazioni effettuate.

Modello	Architettura	SVM	k-NN
VGGFace	VGG16	$0,989 \pm 0,008$	$0,988 \pm 0,008$
	ResNet50	$0,992 \pm 0,007$	$0,992 \pm 0,008$
	SENet50	$0,993 \pm 0,006$	$0,994 \pm 0,006$
FaceNet	Inception-ResNet	$0,992 \pm 0,007$	$0,992 \pm 0,007$
OpenFace	NN4.small2	$0,978 \pm 0,010$	$0,974 \pm 0,014$

7.4 Risultati

7.4.1 Accuratezza

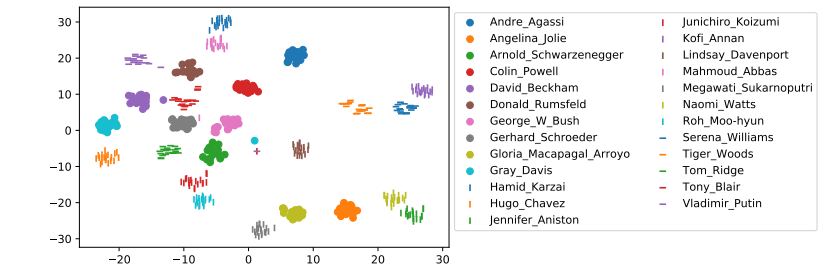
Come mostrato nella tabella 7, la rete che offre le migliori prestazioni nella classificazione con SVM è la SENet50, con un'accuratezza media di $0,993 \pm 0,006$. Questo significa che, in media, ad ogni iterazione della cross-validation, del set di 100 immagini destinate al test ne vengono predette correttamente oltre 99.

Dopo la SENet50, le reti che offrono migliori risultati sono la ResNet50 e la Inception-ResNet, con un'accuratezza media di $0,992 \pm 0,007$, valore non molto distante dal precedente. Anche in questo caso, oltre 99 volti tra quelli di test vengono classificati in modo corretto, su 100.

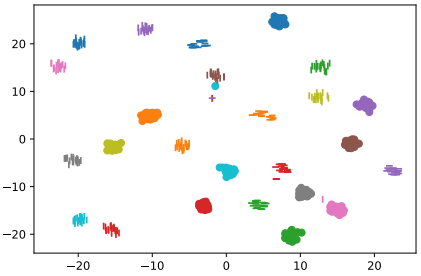
Più indietro troviamo VGG16 con un'accuratezza di $0,989 \pm 0,008$, inferiore alle precedenti ma ancora valida. Decisamente peggiori sono, invece, le prestazioni di OpenFace, con $0,978 \pm 0,010$.

La deviazione standard è, per tutti i modelli eccetto OpenFace, abbastanza ridotta - di due ordini di grandezza inferiore rispetto alla media - a indicazione del fatto che i risultati ottenuti sono discretamente costanti e non si discostano eccessivamente dal valore medio. Questo significa che il classificatore è in grado di effettuare delle classificazioni con risultati uniformi comunque si suddivida il dataset tra sezione di addestramento e sezione di test, a suggerimento di una qualità degli embedding costante per i volti.

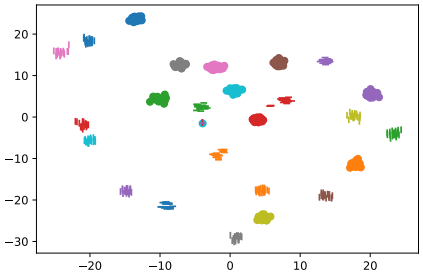
Nelle prove di classificazione con k-NN sono stati confermati i risultati ottenuti con la SVM: la classifica delle migliori prestazioni di accuratezza rimane inalterata e anche i valori assoluti sono pressoché invariati nei due casi. In particolare, SENet50 ottiene il miglior piazzamento con $0,994 \pm 0,006$, facendo registrare un leggero miglioramento rispetto al caso con SVM, pari a $0,001$. Il modello che offre risultati più differenti rispetto al caso con SVM è OpenFace, che fa registrare un peggioramento di $0,004$ assestandosi su $0,974 \pm 0,014$. Anche la deviazione standard, in questo caso, è più accentuata, ad indicare che le prestazioni risentono maggiormente della suddivisione attuata sul dataset.



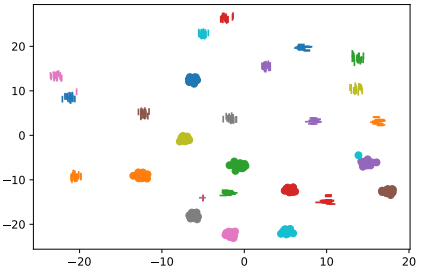
(a) *VGG16*



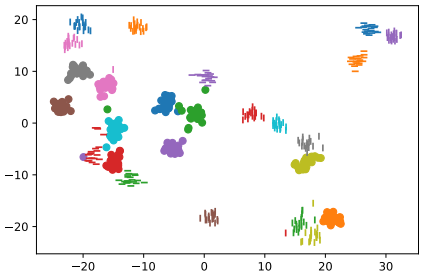
(b) *ResNet50*



(c) *SENet50*



(d) *FaceNet*



(e) *OpenFace*

Figura 28: Visualizzazione bidimensionale degli embedding tramite t-SNE. La legenda riportata in figura **a** è comune a tutti i grafici.

In sintesi, indipendentemente dal classificatore utilizzato, SENet50 è il modello che offre le migliori prestazioni, seguito a breve distanza da ResNet e Inception-ResNet, con cui risulta essere comparabile. Meno buoni sono i risultati prodotti da VGG16 e scadenti, se comparati agli altri, sono quelli di OpenFace.

Tali risultati trovano piena conferma nei grafici riportati in figura 28, che mostrano, per ogni architettura, una visualizzazione bidimensionale degli embedding da essa prodotti. Per ottenerla, è stato applicato lo t-SNE (*t-distributed Stochastic Neighbor Embedding*) [28], un algoritmo di machine learning che opera una riduzione dimensionale non lineare, utilizzato per comprimere dati altamente dimensionali in uno spazio di 2 o 3 dimensioni, in

modo da consentirne la visualizzazione. Esso opera in modo tale che oggetti simili vengano modellati sotto forma di punti tra loro vicini e oggetti differenti sotto forma di punti lontani. Una lettura di grafici così costruiti, quindi, consente di comprendere le capacità di distinzione di una CNN e la qualità degli embedding da essa prodotti.

In linea con i risultati di accuratezza ottenuti, i modelli OpenFace e VGG16 presentano dei grafici in cui i punti che mappano embedding di una stessa identità formano *cluster* (agglomerati) poco compatti, e i vari cluster sono tra loro molto ravvicinati, se paragonati ai grafici degli altri modelli. Ciò significa che tali sistemi non sono in grado di produrre embedding altamente discriminativi. Specialmente OpenFace è caratterizzato da agglomerati separati da distanze minime e talvolta quasi sovrapposti tra loro.

I modelli ResNet50, SENet50 e FaceNet presentano cluster nettamente più compatti e separati spazialmente tra loro, in accordo con gli elevati livelli di accuratezza che fanno registrare. In altre parole, tali modelli sono capaci di produrre embedding particolarmente concentrati per una stessa identità e separati da distanze marcate da quelli di altre identità. Alcune eccezioni, comunque, sono presenti per tutti i modelli: si tratta di vettori mappati in punti distanti dal cluster della classe di appartenenza e che corrispondono, in fase di classificazione, a fonti di errore.

7.4.2 Tempi

Come illustrato nella tabella 8, nella fase di preparazione delle immagini le architetture di VGGFace (che sono accomunate da una procedura analoga) necessitano di un tempo di oltre 1 s ad immagine, che sull'intero dataset è corrisposto ad un tempo complessivo di oltre 500 s. Questo tempo è piuttosto elevato e potrebbe risultare vincolante in situazioni pratiche in cui si abbia un numero maggiore di immagini. Inoltre, i tempi registrati sono affetti da una forte variabilità, come evidenziato dalla deviazione standard di $0,1609$ s che, sull'intero dataset, è corrisposta ad uno scostamento medio di circa 80 s rispetto al valore medio.

Non così diverso è il tempo associato al modello FaceNet, pari a $0,9833$ s per immagine. Questo non deve stupire, dato che la fase di preparazione delle immagini è condotta, per VGGFace e FaceNet, nella stessa maniera, ad eccezione della diversa dimensione delle immagini ottenute, la quale potrebbe essere la fonte della leggera differenza dei tempi nei due casi. Più ridotta è la deviazione standard, di $0,0703$ s, che denota una maggiore costanza nei tempi impiegati.

Un valore molto più basso è riportato da OpenFace e ammonta a $0,5963$ s per immagine, quasi la metà rispetto ai precedenti e derivato da un tempo sul dataset complessivo di circa 300 s. La differenza di tempo è legittima in quanto OpenFace fa uso di una tecnica di preparazione delle immagini differente dagli altri modelli fin qui citati. Di contro, tale modello risente

Tabella 8: Tempi (per immagine) di esecuzione delle fasi di pre-processamento ed estrazione delle caratteristiche, espressi tramite valore medio e deviazione standard delle dieci misurazioni effettuate. Nel caso della fase di estrazione delle caratteristiche si assumono i modelli di CNN come già caricati.

Modello	Architettura	Pre-processamento (s)	Estrazione caratteristiche (s)
VGGFace	VGG16	$1,0732 \pm 0,1609$	$0,5012 \pm 0,0038$
	ResNet50		$0,1903 \pm 0,0193$
	SENet50		$0,1994 \pm 0,0131$
FaceNet	Inception-ResNet	$0,9833 \pm 0,0703$	$0,0949 \pm 0,0045$
OpenFace	NN4.small2	$0,5963 \pm 0,1785$	$0,0194 \pm 0,0006$

di una variabilità maggiore rispetto a FaceNet e comparabile a quella di VGGFace.

Per quanto riguarda i tempi di estrazione delle caratteristiche si può osservare come VGG16 sia molto poco performante, impiegando $0,5012$ s per ogni immagine. Per rendere meglio l'idea, sull'intero dataset ha impiegato circa 250 s. Tempi più bassi di oltre la metà sono offerti da ResNet50 e SENet50, che richiedono circa $0,19$ s ad immagine, e ancora più veloce è il modello FaceNet, con $0,0949$ s. Per distacco, il modello con il tempo medio più basso è OpenFace, grazie ad un valore di $0,0194$ s. Detto altrimenti, sull'intero dataset il tempo di tale sistema è stato di appena $9,7207$ s. Tutti i modelli, nella fase di estrazione delle caratteristiche, si contraddistinguono per tempi stabili nell'arco delle prove, come dimostrato dai valori esigui delle deviazioni standard.

In generale, i risultati ottenuti non sono sorprendenti: i modelli VGGFace sono i più lenti sia nella procedura di preparazione delle immagini richiesta che nell'estrazione delle caratteristiche, con delle differenze, in quest'ultima, tra il modello della prima pubblicazione, VGG16, e i due più recenti, ResNet50 e SENet50, risultati più rapidi. FaceNet ha tempi migliori ma comunque non al livello di OpenFace, che risulta estremamente più veloce degli altri modelli.

Con riferimento alla fase di classificazione è opportuno notare che, in virtù del fatto che i classificatori usati sono gli stessi per tutti i modelli, le differenze di tempo riscontrabili dipendono prevalentemente dalla dimensione dei vettori che vengono forniti loro come campioni: vettori costituiti di più elementi comportano tempi maggiori e viceversa per quelli costituiti da meno elementi. In accordo con ciò, le prestazioni dei classificatori sugli embedding prodotti dai modelli FaceNet e Openface (di 128 elementi) risultano essere le più veloci, con differenze minime tra i due (si veda la tabella 9). Il modello che fornisce il secondo embedding più piccolo, dopo quelli citati, è VGG16

Tabella 9: Tempi di esecuzione della fase di classificazione con SVM e k-NN, espressi tramite valore medio e deviazione standard delle dieci misurazioni effettuate. Essi tengono conto in modo aggregato della fase di addestramento e di quella di test.

Modello	Architettura	SVM (s)	k-NN (s)
VGGFace	VGG16	$0,1864 \pm 0,0025$	$0,0581 \pm 0,0016$
	ResNet50	$0,7473 \pm 0,0220$	$0,2162 \pm 0,0053$
	SENet50	$0,5656 \pm 0,0030$	$0,1990 \pm 0,0012$
FaceNet	Inception-ResNet	$0,0384 \pm 0,0008$	$0,0190 \pm 0,0008$
OpenFace	NN4.small2	$0,0331 \pm 0,0012$	$0,0166 \pm 0,0008$

(512 elementi) e, in linea con questo, esso è quello che comporta tempi migliori nella classificazione proprio dopo FaceNet e OpenFace. I tempi peggiori sono quelli ottenuti utilizzando gli embedding prodotti da ResNet50 e SENet50, che, a tal proposito, sono composti di ben 2048 elementi. Le considerazioni fatte valgono sia per il classificatore SVM che per quello k-NN, con tempi sistematicamente migliori per il secondo.

In sintesi, OpenFace si attesta come il modello con tempi più bassi in tutte le fasi considerate. Innanzitutto, esso effettua l'operazione di produzione dei descrittori in modo molto veloce, aspetto prevedibile se si considera che uno degli scopi primari degli sviluppatori stessi è stato proprio quello di mantenere i tempi di addestramento e predizione ridotti. Anche le fasi di preparazione delle immagini e di classificazione vengono eseguite velocemente, la prima a causa dell'uso dello strumento di allineamento fornito dagli autori e la seconda per via della dimensione contenuta degli embedding.

7.4.3 Considerazioni globali

I risultati degli esperimenti che ho condotto non andrebbero interpretati in termini assoluti per diversi motivi: innanzitutto perché sono stati realizzati su un insieme di immagini non molto esteso, mentre in situazioni concrete si avrebbe, verosimilmente, una quantità di immagini superiore; in secondo luogo perché molte delle ottimizzazioni che si sarebbero potute mettere in pratica non sono state considerate perché necessitano di una profonda conoscenza della materia. Inoltre, i tempi riportati dipendono fortemente dalle specifiche tecnologie hardware e software che ho adoperato. Tuttavia, queste limitazioni sono state costanti in tutti gli esperimenti, pertanto i risultati ottenuti possono essere efficacemente analizzati in termini relativi per valutare quale sia il modello che meglio si presta alle proprie esigenze.

Tenendo a mente ciò, si può osservare che sulla base dei risultati ottenuti OpenFace è il modello di gran lunga più veloce ma anche quello che comporta

livelli di accuratezza più scadenti. SENet50 è, invece, quello più accurato ma, di contro, tra i più lenti. Se la priorità è abbattere i tempi, la scelta deve ricadere senza dubbio su OpenFace, con la consapevolezza che il prezzo da pagare è un'accuratezza non elevata; se, invece, l'interesse principale è l'accuratezza, un buon compromesso è dato dal modello Inception-Resnet che, a dispetto di tempi decisamente più ragionevoli rispetto a SENEt50, offre prestazioni del tutto simili ad esso.

Un'ulteriore considerazione merita la scelta del classificatore: seppur i risultati di accuratezza siano quasi sempre simili tra SVM e k-NN e quest'ultimo impieghi tempi sempre inferiori al primo, ne è sconsigliato l'uso quando si ha una collezione di immagini più estesa e sbilanciata di quella cui io faccio riferimento. Infatti, come spiegato nel paragrafo 6.2, i classificatori k-NN sono non-generalizzanti e in presenza di uno spazio di input più esteso le loro prestazioni potrebbero decrementare rapidamente.

8 Conclusioni

In questa tesi ho affrontato lo studio e la sperimentazione di algoritmi di riconoscimento facciale basati su tecniche di deep learning, con lo scopo di verificare lo stato dei progressi in tale settore al fine di fornire una base di partenza per migliorare le tecniche utilizzate nel fotosegnalamento da parte della Polizia di Stato. Nella prima parte di questo lavoro ho analizzato i più moderni e performanti sistemi adottati per il riconoscimento facciale e tracciato l'evoluzione in tale settore che ha portato all'affermazione del deep learning come standard. In questa fase ho individuato i tre modelli VGGFace, FaceNet e OpenFace come migliori candidati per approfondire lo studio. Su di essi, nella fase successiva, ho condotto esperimenti volti a stabilirne le prestazioni in operazioni di identificazione facciale, considerando come indici di valutazione le tempistiche di esecuzione e i livelli di accuratezza. In questo contesto ho fatto uso di strumenti di classificazione tipicamente utilizzati nel mondo del machine learning, come SVM e k-NN.

I maggiori contributi di tale lavoro sono riassumibili nei seguenti punti:

- aver mostrato la superiorità, in termini di accuratezza, del modello SENet50 (appartenente al progetto VGGFace2) sugli altri, seppur di poco rispetto a ResNet50 (anch'esso parte di VGGFace2) e FaceNet;
- avere mostrato la supremazia netta di OpenFace sugli altri modelli nei tempi di esecuzione;
- aver mostrato come i classificatori SVM e k-NN producano, per un dataset costruito come nel mio caso, risultati di accuratezza tra loro molto simili, con il secondo che ha tempi inferiori rispetto al primo.

Sebbene i risultati ottenuti siano stati piuttosto soddisfacenti, ritengo che ci sia ancora un ampio margine di miglioramento nell'analisi di tecniche di deep learning per il riconoscimento facciale. Un possibile tentativo è rappresentato dalla sperimentazione di alcuni dei modelli tra quelli che io non ho preso in considerazione: a tal proposito suggerisco di prendere in esame i modelli DeepID [40] [39] [41] [38] e DeepFace [45], dei quali sono reperibili implementazioni non ufficiali, e che rappresentano gli unici modelli di una certa rilevanza che non ho incluso nei test. Inoltre, sarebbe opportuno testare l'uso di un sistema di classificazione diverso rispetto a quelli da me adottati e basato su degli strati completamente connessi aggiunti al termine delle reti convoluzionali. In tal caso, l'accuratezza potrebbe risultarne incrementata, tuttavia ne sarebbe difficoltosa l'applicazione in presenza di un numero di identità elevato, in quanto a ciascuna di esse dovrebbe corrispondere esattamente un nodo nell'ultimo strato della rete. Suggerimenti ulteriori sono quelli di estendere le dimensioni del dataset per simulare in modo più realistico situazioni concrete e rendere più variegato il numero di immagini per ogni identità, anziché mantenerlo costante, come accade nel mio dataset. Inoltre, il sistema di classificazione che ho adottato è stato sottoposto a test su identità ad esso note, cioè identità sulle cui foto era stato precedentemente addestrato, e l'eventuale test su volti sconosciuti darebbe risultati fuorvianti; potrebbe, quindi, essere introdotto un sistema per cui le identità non note vengano correttamente riconosciute come tali.

Riferimenti bibliografici

[1] «Keras-OpenFace». `https://github.com/iwantooxxoox/Keras-OpenFace`. GitHub.

[2] T. Ahonen, A. Hadid e M. Pietikainen. «Face Description with Local Binary Patterns: Application to Face Recognition». In *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 28. 12. 2006, pp. 2037–2041.

[3] B. Amos, B. Ludwiczuk e M. Satyanarayanan. «OpenFace 0.2.1». `https://github.com/cmusatyalab/openface`. GitHub, 2016.

[4] B. Amos, B. Ludwiczuk e M. Satyanarayanan. «OpenFace: A general-purpose face recognition library with mobile applications». Rapp. tecn. CMU-CS-16-118, CMU School of Computer Science, 2016.

[5] Aphex34. «Max pooling with a 2x2 filter and stride = 2». `https://commons.wikimedia.org/w/index.php?curid=45673581`. Opera propria, CC BY-SA 4.0. Wikipedia, the free encyclopedia, 2015.

[6] G. Bradski. «The OpenCV Library». *Dr. Dobb's Journal of Software Tools*. 2000.

[7] Q. Cao, L. Shen, W. Xie, O. M. Parkhi e A. Zisserman. «VGGFace2: A dataset for recognising faces across pose and age». In *International Conference on Automatic Face and Gesture Recognition*. 2018.

[8] Z. Cao, Q. Yin, X. Tang e J. Sun. «Face recognition with learning-based descriptor». In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2707–2714.

[9] F. Chollet et al. «Keras». `https://keras.io`. 2015.

[10] R. Collobert, K. Kavukcuoglu e C. Farabet. «Torch7: A Matlab-like Environment for Machine Learning». In *NIPS 2011*. 2011.

[11] R. Gandhi. «Support Vector Machine — Introduction to Machine Learning Algorithms». `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`. Medium, towards data science, 2018.

[12] Google Colaboratory. `https://colab.research.google.com`.

[13] Y. Guo, L. Zhang, Y. Hu, X. He e J. Gao. «MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition». *Lecture Notes in Computer Science*. 2016, pp. 87–102.

[14] K. He, X. Zhang, S. Ren e J. Sun. «Deep Residual Learning for Image Recognition». In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[15] J. Hu, L. Shen e G. Sun. «Squeeze-and-Excitation Networks». In *IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[16] G. B. Huang, M. Ramesh, T. Berg e E. Learned-Miller. «Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments». Rapp. tecn. 07-49. University of Massachusetts, Amherst, 2007.

[17] V. Jain e E. Learned-Miller. «FDDB: A Benchmark for Face Detection in Unconstrained Settings». Rapp. tecn. UM-CS-2010-009. University of Massachusetts, Amherst, 2010.

[18] Q. Ji, J. Huang, W. He e Y. Sun. «Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images». *Algorithms* 12. 2019.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama e T. Darrell. «Caffe: Convolutional Architecture for Fast Feature Embedding». *arXiv preprint arXiv:1408.5093*. 2014.

[20] I. José. «KNN (K-Nearest Neighbors) #1». `https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d`. Medium, towards data science, 2017.

[21] V. Kazemi e J. Sullivan. «One Millisecond Face Alignment with an Ensemble of Regression Trees». In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2014, pp. 1867–1874.

[22] I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller e E. Brossard. «The megaface benchmark: 1 million faces for recognition at scale». In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4873–4882.

[23] D. E. King. «Dlib-ml: A Machine Learning Toolkit». *Journal of Machine Learning Research* 10. 2009, pp. 1755–1758.

[24] A. Krizhevsky, I. Sutskever e G. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». *Neural Information Processing Systems*. 2012.

[25] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard e L. D. Jackel. «Backpropagation Applied to Handwritten Zip Code Recognition». *Neural Computation*. 1989, pp. 541–551.

[26] M. Lin, Q. Chen e S. Yan. «Network In Network». *arXiv preprint arXiv:1312.4400*. 2013.

[27] C. Liu e H. Wechsler. «Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition». *IEEE Transactions on Image Processing* 11.4. 2002, pp. 467–476.

[28] L. van der Maaten e G. Hinton. «Viualizing data using t-SNE». *Journal of Machine Learning Research* 9. 2008, pp. 2579–2605.

[29] R. C. Malli. «keras-vggface». https://github.com/rcmalli/keras-vggface. GitHub, 2016.

[30] A. Nech e I. Kemelmacher-Shlizerman. «Level Playing Field For Million Scale Face Recognition». In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

[31] H. Ng e S. Winkler. «A data-driven approach to cleaning large face datasets». In *2014 IEEE International Conference on Image Processing (ICIP)*. 2014, pp. 343–347.

[32] O. M. Parkhi, A. Vedaldi e A. Zisserman. «Deep face recognition». In *BMVC*. 2015.

[33] I. de Paz Centeno. «mtcnn». https://github.com/ipazc/mtcnn. GitHub, 2018.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg e L. Fei-Fei. «ImageNet Large Scale Visual Recognition Challenge». *International Journal of Computer Vision (IJCV)* 115.3. 2015, pp. 211–252.

[35] K. Sarkar. «ReLU: Not a Differentiable Function: Why used in Gradient Based Optimization? and Other Generalizations of ReLU.» https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec. Medium, 2018.

[36] F. Schroff, D. Kalenichenko e J. Philbin. «FaceNet: A unified embedding for face recognition and clustering». In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

[37] K. Simonyan e A. Zisserman. «Very deep convolutional networks for large-scale image recognition». In *International Conference on Learning Representations*. 2015.

[38] Y. Sun, D. Liang, X. Wang e X. Tang. «DeepID3: Face Recognition with Very Deep Neural Networks». *arXiv preprint arXiv:1502.00873*. 2015.

[39] Y. Sun, X. Wang e X. Tang. «Deep Learning Face Representation by Joint Identification-Verification». *arXiv preprint arXiv:1406.4773*. 2014.

[40] Y. Sun, X. Wang e X. Tang. «Deep Learning Face Representation from Predicting 10,000 Classes». In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1891–1898.

[41] Y. Sun, X. Wang e X. Tang. «Deeply learned face representations are sparse, selective, and robust». In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

[42] C. Szegedy, S. Ioffe, V. Vanhoucke e A. Alemi. «Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning». *arXiv preprint arXiv:1602.07261*. 2016.

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke e A. Rabinovich. «Going deeper with convolutions». In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

[44] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens e Z. Wojna. «Rethinking the Inception Architecture for Computer Vision». In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[45] Y. Taigman, M. Yang, M. Ranzato e L. Wolf. «DeepFace: Closing the Gap to Human-Level Performance in Face Verification». In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1701–1708.

[46] H. Taniai. «keras-facenet». `https://github.com/nyoki-mtl/keras-facenet`. GitHub, 2017.

[47] R. Thakur. «Step by step VGG16 implementation in Keras for beginners». `https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c`. Medium, towards data science, 2019.

[48] M. Turk e A. Pentland. «Eigenfaces for Recognition». *Journal of Cognitive Neuroscience* 3.1. 1991, pp. 71–86.

[49] P. Veličković. «TikZ». `https://github.com/PetarV-/TikZ`. GitHub, 2016.

[50] P. Viola e M. Jones. «Robust Real-Time Face Detection». *International Journal of Computer Vision* 57. 2004, pp. 137–154.

[51] M. Wang e W. Deng. «Deep Face Recognition: A Survey». *arXiv preprint arXiv:1804.06655*. 2018.

[52] L. Wolf, T. Hassner e I. Maoz. «Face recognition in unconstrained videos with matched background similarity». In *CVPR 2011*. 2011, pp. 529–534.

[53] S. Yang, P. Luo, C. C. Loy e X. Tang. «WIDER FACE: A Face Detection Benchmark». In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[54] D. Yi, Z. Lei, S. Liao e S. Z. Li. «Learning Face Representation from Scratch». *arXiv preprint arXiv:1411.7923*. 2014.

[55] M. D. Zeiler e R. Fergus. «Visualizing and Understanding Convolutional Networks». *Lecture Notes in Computer Science*. 2014, pp. 818–833.

[56] K. Zhang, Z. Zhang, Z. Li e Y. Qiao. «Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks». *IEEE Signal Processing Letters* 23.10. 2016, pp. 1499–1503.