UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Test sul riconoscimento facciale usando modelli pre-addestrati in Keras

Face Recognition test using pre-trained models in Keras

Relatore:
Prof. Aldo Franco Dragoni

Tesi di:
Ciocirlan Stefan

Correlatore:
Dott. Paolo Sernani

Anno accademico 2019/2020

Indice

Capitolo 1

1.Introduzione

Il *riconoscimento facciale* (in inglese *face recognition*) è una tecnica di intelligenza artificiale, utilizzata in biometria atta ad identificare in modo univoco una persona confrontando e analizzando modelli basati sui suoi "contorni facciali". La maggior parte dei sistemi di riconoscimento facciale si basano sulle reti neurali che apprendono che cosa nell'imagine di un volto per essere utile per identificarlo, ovvero ne estrae un descrittore. Con questa tecnica, le applicazioni possono utilizzare I dati acquisiti dai volti e possono identificare accuratamente e rapidamente gli individui interessati.

- La *fase di verifica* può essere vista come un problema di classificazione, dove, in poche parole, si prendono 2 immagini, si calcola la distanza (quadratica media) tra le 2, e se questa è più piccolo di un certa soglia, che verrà scelta in base a diversi test eseguiti da noi, allora sulle immagini c'è la stessa persona, persone diverse invece se la distanza assume un valore più grande di quella soglia. (Vedremo più in dettaglio come funziona questa fase nei capitoli successive)
- La *fase di identificazione* invece consiste nel comprendere se l'imagine contiene un volto noto, ovvero un soggetto appartenente ad un insieme di N soggetti noti.

1.1 Obiettivi

Gli obiettivi che ci siamo proposti da portare a termini all'interno di questo lavoro di tesi sono I seguenti:

- Cercare e confrontare modelli pre-addestrati per il riconoscimento facciale e valutare quale sia migliore (tra quelli scelti) in termini di accuratezza (identificazione delle persone) e di tempo.
- Confermare l'efficacia del "transfer learning" cioè la capacità di una rete neurale profonda pre-addestrata di estrarre features da un dataset diverso da quello di addestramento.
- Capire in che modo le prestazioni, in termini di velocità e accuratezza, dei vari modelli possono essere migliorate.

1.2 Struttura della Tesi

La prestazione del lavoro svolto si suddivide nel sequente modo:

- Nel primo abbiamo una piccolo introduzione al riconoscimento facciale e gli obiettivi della tesi.
- Nel secondo capitolo elencheremo gli strumenti utilizzati e i vari step eseguiti per configurare l'ambiente di lavoro.
- Nel terzo capitolo descriveremo il sistema realizzato in termini di codice e file.
- Nel quarto capitolo descriveremo I modelli confrontati.
- Nel quinto verrano esposte le conclusioni sui vari modelli usati.

Capitolo 2

Configurazione Software

Il linguaggio che useremo è Python in quanto è fornito da una libreria built-in estremamente ricca, che unitamente alla gestione automatica della memoria e a robusti costrutti per la gestione delle eccezzione lo fa uno dei linguaggi più ricchi e comodi da usare, comodo ma anche semplice in quanto la sua sintassi è pulita e snella cosi come I suoi costrutti, decisamente chiari e non ambigui [2]. Come ambiente di sviluppo invece useremo PyCharm, uno dei IDE più adatti per questo linguaggio e si può scaricare gratuitamente dal seguente link [3].Prima di iniziare il nostro lavoro, comunque abbiamo bisogno di installare qualche pacchetto tramite *pip,* il package manager di default, che si college a PyPI [4], cerca e scarica il pacchetto richiesto. Ogni pacchetto viene installato tramite la seguente istruzione nel terminale del IDE:

pip install nome_pacchetto

Inoltre per semplificare la distribuzione e la gestione dei pacchetti abbiamo bisogno di installare sulla nostra macchina la piattaforma Anaconda, che contiene al suo interno l'interprete Python, e dunque quando andiamo a creare il nostro progetto, dobbiamo scegliere come interprete Python contenuto in Anaconda:

Dopo aver creato il nostro progetto, andremo ad installare I suguenti pacchetti:

- **pip install numpy**

Numpy[5] è un'estensione del linguaggio Python che aggiunge il supporto per array di grandi dimensioni e multidimensionali, oltre a una vasta libreria di funzioni matematiche di alto livello per operare su questi array.

Richiamiamo questo pacchetto nel codice sorgente con la seguente istruzione:

import numpy as np

Di seguito tutti gli altri pacchetti verrano importati nel codice sorgente con l'istruzione:

import nome_pacchetto [as nome_abbreviato]

Il **nome_abbreviato** serve solamente per fare il nostro codice più facile da leggere.

Facendo riferimento al pacchetto **numpy**, un esempio pratico è il seguente:

Crea una matrice 3x3:

x=np.array(3,3)

- **pip install os**

Questo pacchetto[6] fornisce dozzine di funzioni per interagire con il Sistema Operativo. Lo andremo ad usare per navigare all'interno del nostro *dataset*.

- **pip install cv2**

Questo pacchetto, detto OpenCV[7], serve per risolvere problemi di Computer Vision. Dunque ci servirà per leggere le immagini. Vediamo un esempio di sintassi:

cv2.imread(path)

Questa istruzione andrà a leggere l'immagine contenuta nella cartella **path,** e ritorna come valore l'immagine stessa.

- **pip install matplotlib**

Questo pacchetto[8] ci servirà per la rappresentazione di certi risultati facendo uso di grafici.

Inoltre faremo uso della libreria **Keras** [9], in quanto I modelli pre-addestrati che analizzeremo la usano come riferimento. Questa libreria è stata fatta per lo sviluppo dei modelli in Deep Learning. Essa semplifica la vita a tutti coloro che vogliano approcciare al mondo delle reti neurali, ma non hanno interesse ad approfondire le tecniche matematiche connesse all'uso delle stesse, inoltre è stata progettata con l'obiettivo di consentire una rapida prototipazione di modelli neurali e lavora senza problemi sia su CPU che GPU. Per la manipolazione dei *tensori* (array multidimensionali), questa si basa su diversi framework che funzionano come «*Backend Engine*». Il framework di cui faremo uso è **Tensorflow**[10], scritto dal team Google Brain, che sfrutta il calcolo numerico utilizzando I grafici di flusso di dati, passando attraverso dei ''Nodi'' nei quali vengono elaborati tramiti formule matematiche per permettere l'apprendimento e lo sviluppo delle reti neurali.

Tensorflow

TensorFlow è uno dei più famosi framework open source per svolgere attività di machine learning e anche grazie alla sua ampia flessibilità ed adattabilità è stato adottato da molte aziende tra cui Airbnb, CocaCola, DeepMind, Airbus, Amd, Intel, Nvidia, Twitter, IBM, Google. Google nello specifico ne ha fatto uso per l'ampia capacità di comprendere e processare linguaggi naturali, classificazione di testi, immagini, riconoscimento di scrittura manuale ed `e l'azienda che porta avanti il suo sviluppo. TensorFlow viene catalogato come uno dei tools più semplici e di facile comprensione nel momento in cui ci si approccia per la prima volta al Deep Learning, grazie all'ampia documentazione fornita ma anche perchè usa di base Python come linguaggio e perchè, come detto in precedenza, `e supportato da Google e dispone di una ampia comunità di ricercatori che sviluppano continuamente Api aggiornate e migliorate in modo da mantenere sempre aggiornata la piattaforma allo stato dell'arte attuale nell'ambito del machine learning.

2.1 CNN

La Convolutional Neural Network (Rete Neurale Convoluzionale) è un rete neurale artificiale usata per l'analizzazione delle immagini. Alla base di questa rete sono i Convolutional Layers, che stanno nella parte degli Hidden Layers della rete, hanno la funzione di individuare certi "*features*" attraverso dei *filtri.* Quando parliamo di "*features*" ci riferiamo a diverse parti dell'immagine, ovvero presa un'immagine che contiene una persona, come esempi di patterns possono essere, gli occhi, il naso, la bocca, i capelli e cosi via. Dunque per ogni *feature* ci sarà un certo *filtro* che lo individua. In più i layer sono strutturati in 3 dimensioni: lunghezza, altezza e profondità (width, height, depth).

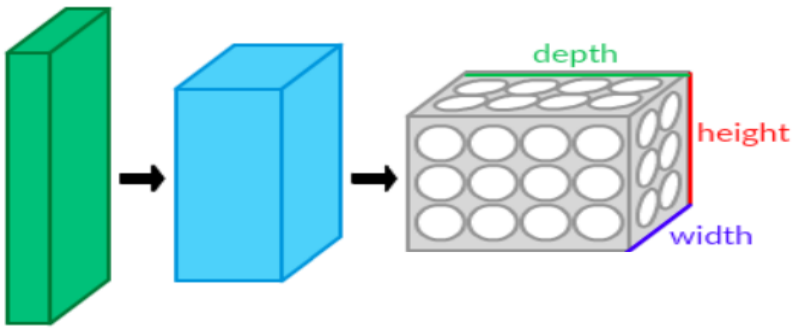


Fig. 1: Struttura CNN

In più i neuroni di un layer non sono connessi con tutti gli altri neuroni del layer successivo, ma solo con un piccola parte. Alla fine, l'ultimo output sarà ridotto ad un vettore con dei valori di probabilità organizzato lungo la dimensione della profondità (depth).

La CNN è composta da 2 parti:

1. **The Hidden Layers:** La parte dove la rete esegue una serie di operazioni di *convoluzioni* e di *pooling* per estrarre le features.
2. **The Classification part:** La parte dove si assegna una probabilità nell'identificare l'oggetto che c'è nell'immagine.

2.2 Architettura della CNN

Una *Convolutional Neural Network* usa principalmente 4 di tipi di layer:

- *Convolutional Layer*
- *ReLu Layer*
- *Pooling Layer*
- *Fully-Connected Layer e Softmax*

2.2.1 Convolutional Layer

Questo tipo di layer sta alla base di questa rete ed esegue le principali operazioni di training e consecutivamente accendono I neuroni della rete. Esso esegue l'operazione di convoluzione sull'input e consiste in un arrangiamento 3-dimensionale dei neuroni.

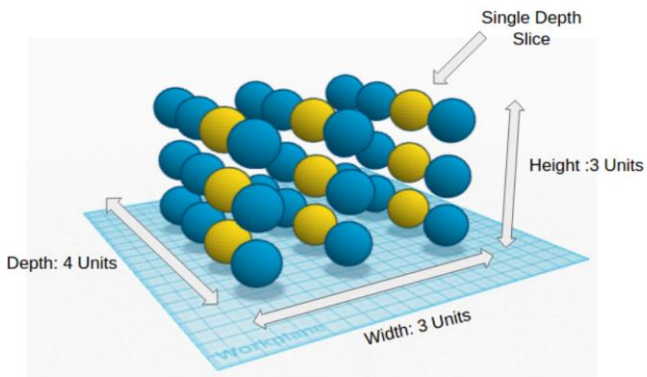


Fig. 2: Rappresentazione 3-dimensionale di un layer Convoluzionale con 3x3x4=36 neuroni

Ogni neurone è collegato solo ad una parte dell'input (*receptive field*), ottenendo cosi una diminuzione del numero dei pesi (weights) e quindi quando si va ad addestrare I pesi, la complessità computazionale si riduce.

Tutti I pesi dei layers Convoluzionali sono addestrati e aggiornati in ogni iterazione di "learning" usando un algoritmo "Back-Propagation" esteso e applicato ad arrangiamento 3-dimensionale dei neuroni, che computa il gradiente dello spazio dei pesi.

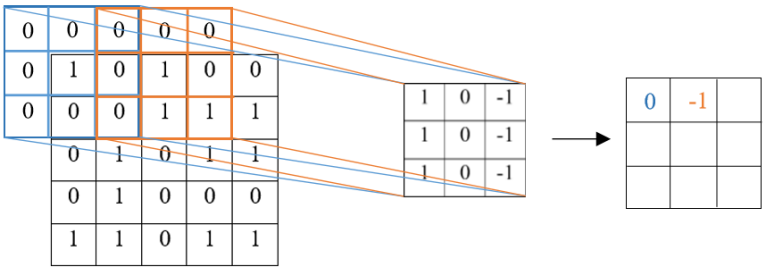
Supponiamo di avere come input un'immagine in bianco e nero per semplicità, ad esempio:

1	0	1	0	0
0	0	1	1	1
0	1	0	1	1
0	1	0	0	0
1	1	0	1	1

E supponiamo che il layer convoluzionale abbia definito un kernel di dimensione 3x3.

1	0	-1
1	0	-1
1	0	-1

La convoluzione viene calcolate come sommatoria della moltiplicazione tra le 2 matrici. Questo deve essere fatto per ogni pixel dell'immagine. Il risultato ottenuto è anche esso una matrice che identifica la feature estratta tramite il kernel definito. La matrice è la Feature Map:



Ovviamente se si utilizza un'immagine a colori, quindi che contiene la profondità (depth), I filtri applicati devono estendersi per tutta la profondità di tale dimensione.

I parametri del layer convoluzionale che determinano la dimensione della feature map sono:

Depth. Ogni filtro estrae un tipo di feature dall'immagine, visto che di solito abbiamo bisogno di estrarre più feature, dobbiamo introdurre più filtri, ogni filtro produrrà una feature map diversa. Il numero di questi filtri (e delle feature estratte) è rappresentato dal parametron Depth.

Stride. Questo rappresenta quant'è lo scorrimento applicato al kernel sull'elemento di input durante la convoluzione. Ovviamente più lo stride è largo e più le feature map saranno di dimensioni ridotte.

Zero-padding. Per poter catturare l'informazione ai lati dell'immagine può risultare utile applicare uno zero-padding ai bordi di essa

In aggiunta a questi parametri un'altra caratteristica da fissare è la dimensione dei filtri (filter size). Formalmente, il valore di un elemento in posizione (x,y) all'interno della feature map j-esima per il i-esimo livello è dato da [11]:

$$vij\ xyz = \tanh\left(bij + \sum \sum \sum wijm\ pq\ Qi-1\ q=0\ Pi-1\ m\ p=0\ v(i-1)m\ (x+p)(y+q)\right)$$

Dove w rappresenta il peso del kernel associato alla m-esima feature map, P e Q sono la dimensione dei kernel ed infine v(i-1) è l'input dello strato i-esimo, cioè il valore della feature map in output allo strato precedente. Inoltre, nella formula compaiono elementi non ancora definiti ma che il layer convoluzionale applica. Notiamo il bias b, che viene aggiunto al risultato dell'operazione di convoluzione e la funzione tangente iperbolica tanh(). Quest'ultima introduce la componente non lineare caratteristica delle reti neurali. La tangente è la funzione di attivazione del neurone e può essere sostituita da altre funzioni non lineari molto utilizzate, come la sigmoide o la funzione ReLU.

2.2.2 ReLu Layer

Questo layer serve per diminuire la non linearità delle immagini, visto che quando si guarda un'immagine ci sono tante features non lineari (bordi, colori, transazioni tra I pixel etc).

Dunque ReLU sta per Raddrizzatore (Rectified Linear Unit) ed è un tipo di funzione d'attivazione. Matematicalmente è definita in questo modo:

$$y = max(0, x)$$

2.2.3 Pooling Layer

L'obiettivo principale del layer di pooling è quello di ridurre la dimensione delle feature map, eliminando informazione irrilevante ma mantenendo quella utile, ovvero ridurre progressivamente la dimensione della rappresentazione dell'input. Questo viene fatto applicando una funzione alla feature map chiamata appunto operazione di pooling. Non esiste un solo tipo di operazione ma ne esistono differenti in base alla funzione applicata, tra le più note abbiamo il Max Pooling o l'Average Pooling. Data in input una feature map, il principio di funzionamento è quello di definire una finestra spaziale che scorre lungo tutta la feature map (come il kernel nel convolutional layer) e applicare l'operazione di pooling agli elementi dell'input che cadono all'interno della finestra. Ovviamente il risultato del layer di pooling è ancora una volta una feature map, che rappresenta la stessa quella partenza ma con dimensioni ridotte. Facciamo un esempio concreto per spiegare meglio il funzionamento. Ancora una volta dobbiamo andare a definire un parametro di stride e la dimensione dei filtri. Nel caso di un Max Pooling quello che si fa è prendere il massimo tra i valori compresi all'interno della finestra spaziale, producendo una matrice di dimensioni ridotte. Questo layer serve per diminuire il numero di parametri, limita l'overfitting, genera una rappresentazione ridotta e semplice da utilizzare.

2.2.4 Softmax e Fully-Connected Layer

Gli ultimi due layer vengono utilizzati per eseguire la classificazione. Essi in realtà non sono stati introdotti dal modello convoluzionale. Infatti, il layer fully-connected (FC) è rappresentato da una Multi Layer Perceptron tradizionale. Tale strato viene messo in combinazione con una funzione di attivazione softmax, che permette di restituire un grado di affinità tra le classi utilizzate nell'addestramento della rete. Di fatto eseguono la classificazione e perciò vengono inseriti in fondo alla rete. I due layer prendono in input le feature di alto livello estratte dagli strati convoluzionali ed eseguono la classificazione dell'input. Le classi potrebbero essere calcolate 28 direttamente dall'output convoluzionale (tante feature separate di alto livello), ma si è osservato che utilizzare i layer FC permette di combinare le feature ottenendo risultati migliori. In questi layer non dobbiamo specificare l'insieme di parametri che abbiamo visto precedentemente negli altri, ma basterà definire il numero di neuroni presenti nello strato. Come per le MLP l'output prodotto è un vettore trasformando l'input bi-dimensionale (o tri-dimensionale per un'immagine a colori) in un output mono-dimensionale. Questo implica che dopo dei layer fully-connected non possono essere inseriti altri layer convoluzionali.

2.2.5 Training

L'addestramento risulta simile a quello utilizzato per le reti neurali standard, con l'accortezza di modificare il normale algoritmo di apprendimento per I layer convolozuionali. L'algoritmo utilizzato è lo *stochastic gradient descent* con la *backward propagation*[12]. I passi del training sono:

1. Si inizializzano I parametri dei kernel e dei pesi in maniera randomica.
2. Si esegue la forward propagation. La rete prende in ingresso un input

etichettato con una classe che viene processato attraverso I pesi randomici e produce il grado di confidenza relativo alle classi da predire. Avendo I pesi random il risultato può essere totalmente errato.

3. Si calcola l'errore totale facendo riferimento all'output desiderato e quello ottenuto. Per fare questo è necessario definire la funzione che quantifica questo errore che viene denominata loss function. Tra le più note abbiamo l'errore quadratico medio, definito come:

$$E_{Tot} = \sum \frac{1}{2}(Target - Output)^2$$

4. A questo punto si utilizza la back propagation per calcolare il gradiente dell'errore totale a tutti I pesi della rete ($dETot/\ dW$) propagando all'indietro tra I vari strati. Viene chiamata back propagation appunto perchè il processo parte dallo strato di output e si ripercorre la rete all'indietro. Grazie al gradiente si aggiornano I pesi cercando di minimizzare la loss function. I pesi si aggiornano in opposizione alla direzione del gradiente:

$$w_{new} = w_{old} - \eta \frac{dE_{Tot}}{dW}$$

Dove η è definito learning rate e rappresenta un parametro da settare nella costruzione della rete. Il learning rate definisce la variazione del peso in funzione dell'errore. Esso è un parametro molto sensibile, se troppo alto si rischia di non riuscire a minimizzare la loss function, se invece è troppo basso si rischia di impiegare troppo tempo o addirittura non riuscire a variare I pesi.

2.3 Configurazioni Hardware

La macchina utilizzata per il testing del sistema di riconoscimento è un laptop Huawei Matebook X Pro. Le specifiche Hardware della macchina sono:

- CPU: i7-8550U 1.80 GHz
- GPU: GeForce MX 150 2GB
- RAM: 8 GB DDR
- SSD interno 512 GB

I modelli che useremo possono essere eseguiti sia sulla CPU che sulla GPU, ma in quanto lavoriamo con delle immagini, usando la CPU, le prestazioni in termini di tempo di esecuzione, riconoscimento delle identità è notevolmente maggiore rispetto a quando si usa la GPU. Dunque si è deciso di utilizzare I cuda core della scheda video, attraverso CUDA[13], per migliorare l'efficienza di elaborazione. CUDA è una piattaforma per l'elaborazione parallela creata da NVIDIA. Di fatto definisce una estensione ai linguaggi C, C++, Fortran, Python e MATLAB permettendo la scrittura di applicazioni capaci di eseguire calcoli in parallelo sulle GPU delle schede video NVIDIA. Questo permette di ottenere delle prestazioni di computing nettamente elevate. Per un ulteriore aumento delle performance si deciso di installare l'acceleratore grafico cuDNN e di compilare la libreria DLIB attraverso CUDA.

Grazie alla compilazione della libreria tramite CUDA si ha un'ottimizzazione nel processo di calcolo degli strumenti definiti dalla libreria. Le versioni di CUDA e cuDNN installate sono rispettivamente la 10.1 e la 7.6.4.

2.4 Dataset

I dataset di cui abbiamo fatto usato sono:

- Labeled Faces in the WILD (LFW)[14]
- Youtube Faces DB[15]

LFW

Questo database comprende più di 13.000 immagini prese dal web e consente il libero uso in diverse applicazioni, anche se di solito viene usato per il paragone tra 2 immagini per capire c'è la stessa persona oppure no "pair matching", può essere usato anche per le applicazione che comprendono il riconoscimento facciale. Ogni faccia è etichettata con il nome della persona che c'è nell'immagine. 1680 persone che hanno due o più immagini distinte.

Youtube Faces DB

Questo database di solito si usa per il riconoscimento facciale delle persone nei video, ma essendo I video una sequenza di frame, possiamo utilizzarlo anche per le applicazioni che riguardano il riconscimento delle persone nelle immagini. Contiene 3.425 video con 1595 persone diverse. Il video più corto contiene 48 frame, il più lungo invece 6070, e la media di un video è di 181 frame.

Capitolo 3

Il sistema realizzato

Il nostro progetto avrà la seguente struttura

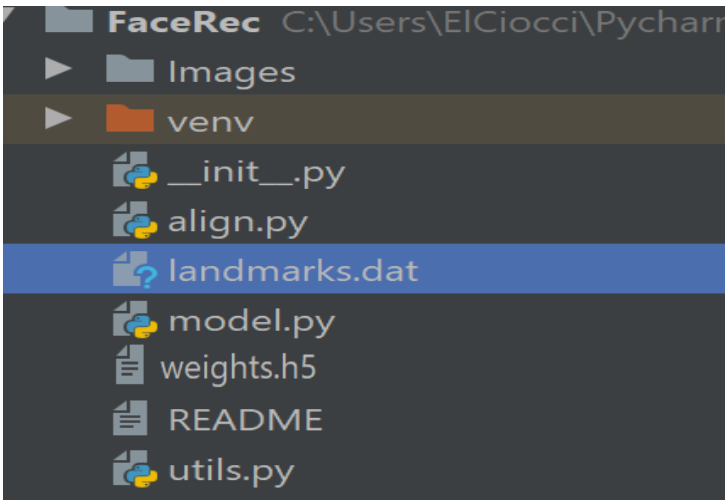


Fig. 3 Struttura del progetto

La cartella genitore è la FaceRec, qui vengono inseriti tutti I file le cartelle figlie relative al progetto, in più viene definite anche l'ambiente virtuale "venv", quest'ultimo ci permette di creare degli spazi indipendenti dal resto del Sistema in cui è possible testare e lavorare con Python e pip, concretamente in questa cartella ci sono I vari file necessary al funzionamento dell'ambiente e una copia dei binary di Python, e dentro questa possiamo installare tutti I moduli con la versione che vogliamo in base al progetto su cui stiamo lavorando, ma che richiederanno l'attivazione di questo ambiente per essere utilizzati. Per quanto riguarda la cartella Images, qua verrà inserito il dataset con il quale andremo a lavorare.

Il file che fa partire il nostro progetto è il __init_.py. Qua vengono create varie funzioni come il caricamento delle immagini, caricamento del modello e I suoi pesi, allineamento delle immagini, calcolo della distanza euclidea, addestramento e previsioni dei classificatori.

Per prima cosa si va a caricare il modello e I suoi pesi tramite :

```
nome_modello=create_model()
nome_modello.load_weights('pesi_modello.h5')
```

Con la prima istruzione si va a caricare il modello che sta nel file model.py, infatti qua abbiamo la funzione create_model() dove viene definita la struttura del modello, se invece il modello viene definite in un file di tipo .h5, allora si esegue la seguente istruzione:

```
nome_modello=load_model('nome_modello.h5')
```

Dopodicchè si va a caricare il dataset sul quale il modello andrà a fare le previsioni, ovvero se su una copia di immagini c'è la stessa persona oppure no.

```
def load_metadata(path):
    metadata = []
    for i in os.listdir(path):
        for f in os.listdir(os.path.join(path,
                                         i)):
                for s in os.listdir(os.path.join(path,i,f)):
                    metadata.append(IdentityMetadata(path, i,f,s))
    return np.array(metadata)


metadata = load_metadata('Images')
```

La funzione load_model è quella che si occupa della fase di caricamento delle immagini, ovvero non sono le immagini vere e proprie, ma I percorsi assoluti alle varie immagini, che verrano inseriti nell'array metadata.
Per il vero caricamento delle immagini si usa il modulo cv2 con la metodo imread, nel seguente modo:

```
def load_image(path):
    img = cv2.imread(path, 1)
    return img[..., ::-1]
```

Dopo il caricamento dell'immagine si va a cercare la faccia della persona e inquadrarla in una dimensione che prende in input il modello considerato. Le varie funzioni di allineamento vengono descritte nel file align.py

```
alignment = AlignLibrary('landmarks.dat')

def align_image(img):  # function to align the faces and put them into a small box 96x96px
    return alignment.align(96, img, alignment.getLargestFaceBoundingBox(img),
                           landmarkIndices=AlignLibrary.INNER_EYES_AND_BOTTOM_LIP)
```

Per ogni immagine si va a calcolare I vari vettori embedded vectors [X], con I quali si calcolerà la distanza euclidea e si faranno le previsioni.

```
img = (img / 255.).astype(np.float32)

embedded[i] = nome_modello.predict(np.expand_dims(img, axis=0))[0]
```

Per il riconoscimento vero e proprio delle persone abbiamo bisogno di un classificatore, che verrà addestrato sulla metà del dataset in questione, e farà il test del riconscimento dell'altra metà del dataset. Abbiamo usato 2 classificatori per vedere quale si comporta meglio, KNN e SVM.

```
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
svc = LinearSVC()

knn.fit(X_train, y_train)
svc.fit(X_train, y_train)

acc_knn = accuracy_score(y_test, knn.predict(X_test))
acc_svc = accuracy_score(y_test, svc.predict(X_test))
```

KNN

Questo classificatore (K-Nearest Neighbors[16]) trova le distanze tra l'immagine in questione e tutte le altre del dataset, e seleziona un certo numero di vicini K, che hanno la distanza minore. Nel nostro progetto abbiamo usato K=1, ovvero si sceglie il primo vicino, quello con cui si ha la distanze euclidea più piccola.

SVM

Le SVM[17] sono dei modelli supervisionati di Machine Learning usate per la classificazione o regressione. L'idea alla base delle SVM consiste nel ricercare un determinata funzione di decisione, a partire da un insieme di classi e di esempi etichettati. La funzione ha lo scopo di prevedere quale sia la classe di appartenenza di un ulteriore input non etichettato.

Dopo l'addestramento dei classificatori si passa al riconoscimento delle persone sulle singole immagini, usiamo un ciclo while per non far ripartire il progetto ogni volta quando abbiamo riconosciuto una persona, e fare tutto il procedimento da capo. Si prende dall'input l'immagine della persona della quale si vuole conoscere il nome, dopodicchè il classificatore avrà il compito di riconoscere la persona tramite il metodo predict.

```
answer = True

while answer:
    my_answer = int(input('Choose a number :'))
    example_idx = my_answer

    example_image = load_image(metadata[test_idx][example_idx].image_path())
    example_prediction = svc.predict([embedded[test_idx][example_idx]])
    example_identity = encoder.inverse_transform(example_prediction)[0]
    plt.imshow(example_image)
    plt.title(f'Recognized as {example_identity}')
    plt.show()
    my_answer2 = input("Would you like to recognize another person? [n/y]")
    if my_answer2 == 'y':
        continue
    else:
        answer = False
```

Capitolo 4

Modelli confrontati

I 3 modelli che abbiamo utilizzato per fare I nostri esperimenti sono:

1. **Open face nn4.small2.v1**
2. **Inception_resnet_v1**
3. **VGG_Face**

I 3 modelli sono già addestrati, quindi non abbiamo bisogno di fare il passo dell'addestramento, in quanto possiamo scaricare I pesi di ogni modello.

4.1 Open face nn4.small2.v1

Questo modello[18] è stato addestrato su 2 datasets, FaceScrub e CASIA-WebFace e prende in input immagini con dimensioni pari a 96x96 px. Per ogni immagine dei nostri dataset si fa un ridimensionamento e si cerca di inquadrare la faccia della persona usando la libreria AlignDlib e I punti di riferimento gli occhi e il naso, più precisamente «Outer eyes and nose», con le rispettive dimensioni. Tale modello è composto da 157 layers.

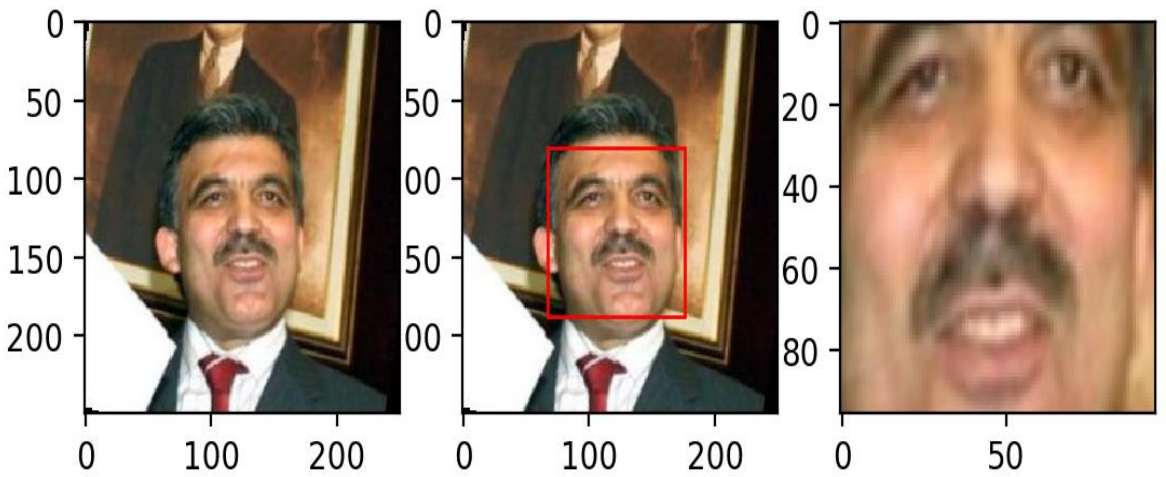


Fig. 4 Ridimensionamento dell'immagine e inquadratura della faccia.

Dopo aver fatto il ridimensionamento e inquadrato la faccia, l'immagine viene convertita in un vettore (embedded vector) per calcolare poi la Distanza Euclidea, con la quale si stabilisce se su 2 immagini c'è la stessa persona oppure no. Di seguito vediamo che questa distanza è molto minore quando sulla coppia c'è la stessa persona che quando ci sono persone diverse.

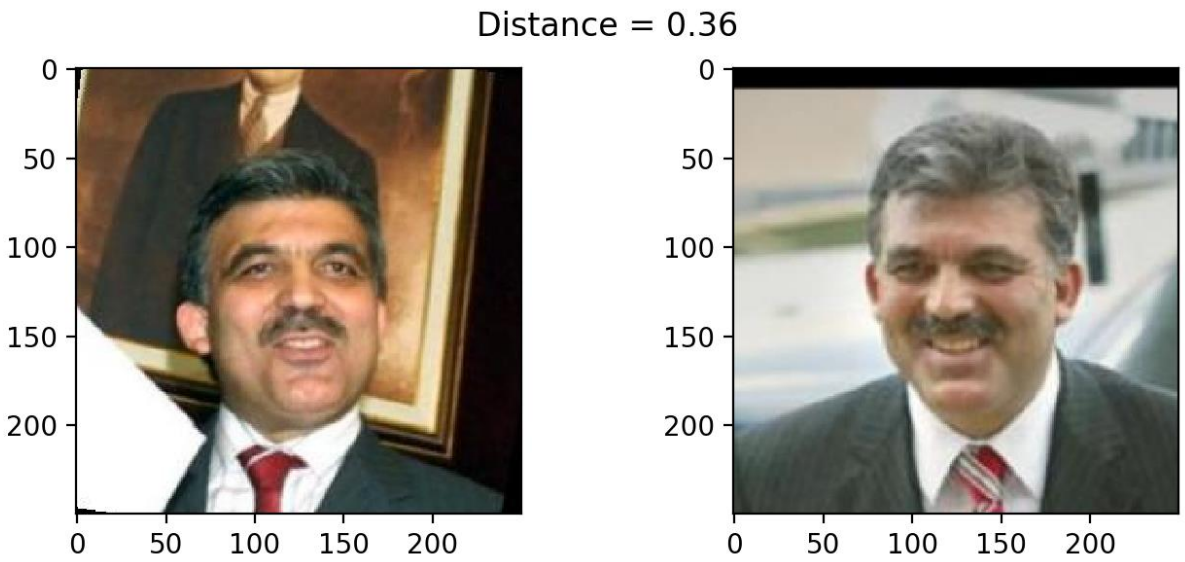


Fig. 5 Distanza Euclidea su una coppia di immagini con stessa persona

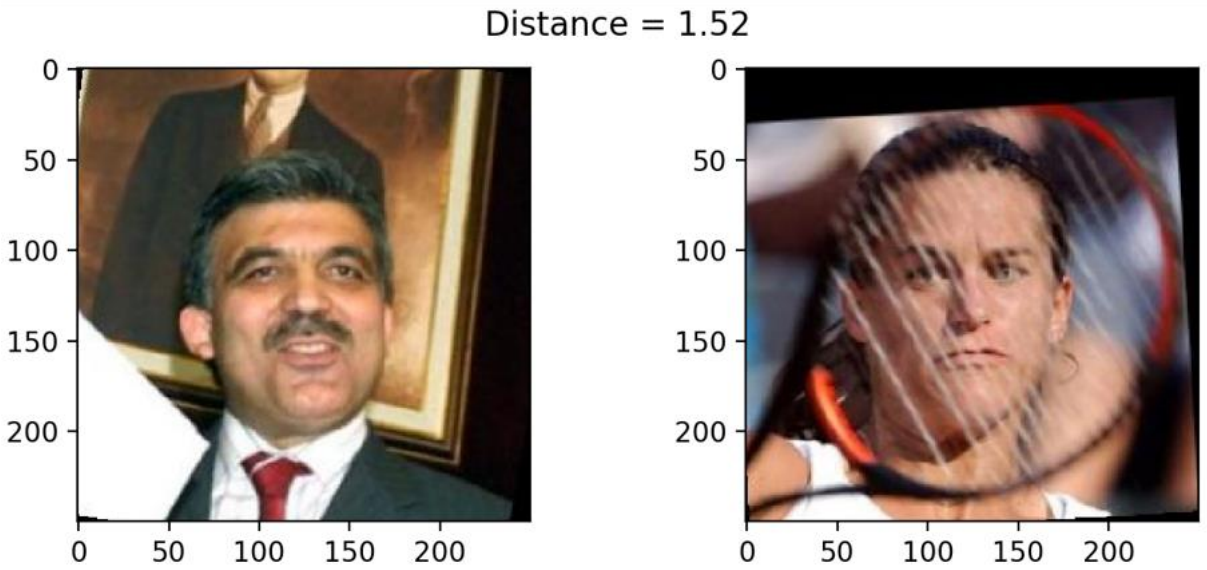


Fig. 6 Distanza Euclidea su una coppia di immagini con persone diverse.

Dopo aver calcolato la Distanza Euclidea tra tutte le coppie di immagini si è stabilito che l'accuratezza massima, ovvero di 97,9% si ha quando la distanza è pari a 0.54.

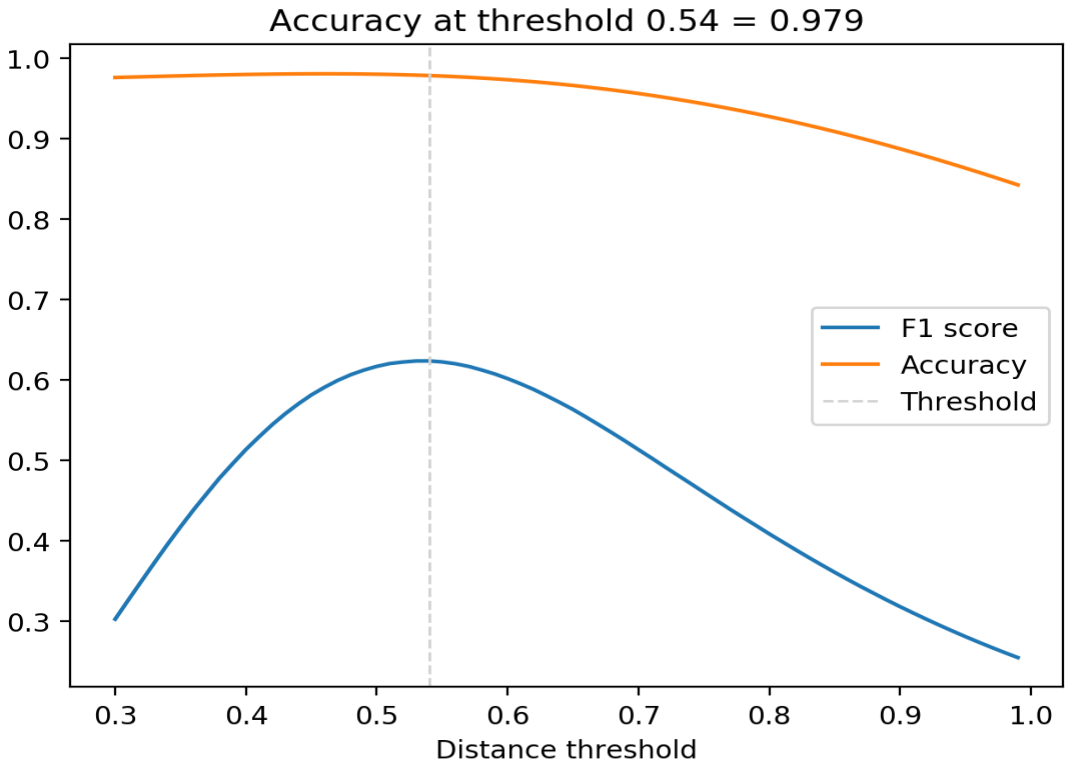


Fig. 7 Andamento delle curve di accuratezza e della misura F del modello nn4.small2.v1 con una suddivisione del datset LFW

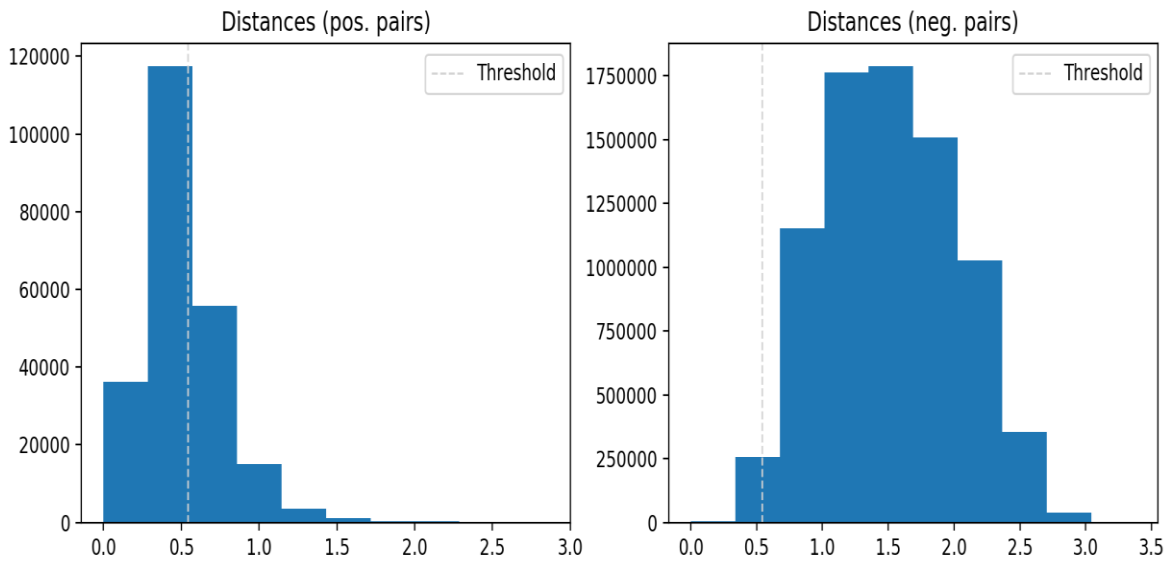


Fig. 8 Grafico del numero di coppie positive e negative del modello nn4.small2.v1 su una suddivisione del dataset LFW

Tutto il procedimento fatto fino adesso serve per capire se il modello funziona in modo corretto e riesce a stabilire se su una coppia di immagini c'è la stessa persona oppure no, per quanto riguarda il riconscimento vero è proprio si usa il classificatore.

Abbiamo usato entrambi I classificatori per vedere quale si comporta meglio e abbiamo ottenuto I seguenti risultati:

Usando come dataset una suddivisione del dataset LFW, 143 persone, 4031 immagini:

Accuratezza KNN: 83,6%

Accuratezza SVM:86,2%

Questo risultato è stato ottenuta al terzo tentative, al primo tentative abbiamo usato la seguente suddivisione, 50 persone, 5 immagini per ogni persona, I risultati in termini di accuratezza erano:

Accuratezza KNN: 67,5%

Accuratezza SVM: 71,6%

Al secondo tentative abbiamo usato I punti di riferimento Inner Eyes and Bottom Lip per l'identificazione del volto, con la suddivisione 143 persone, 4031 immagini, I risultati in termini di accuratezza erano:

Accuratezza KNN: 75,4%

Accuratezza SVM: 76,9%

Usando come dataset una suddivisone del dataset Youtube DB, 30 persone, 10000 immagini:

Accuratezza KNN: 98,1%

Accuratezza SVM:97,65%

Anche qua, abbiamo ottenuto questo risultato al terzo tentativo, al primo tentative abbiamo usato la seguente suddivisione, 50 persone, 5 immagini (le immagini appartengono a video diversi) per ogni persona, I risultati in termini di accuratezza erano:

Accuratezza KNN: 71,6%

Accuratezza SVM: 70,3%

Al secondo tentativo abbiamo usato I punti di riferimento Inner Eyes and Bottom Lip con la suddivisione 30 persone, 10000 immagini, e abbiamo ottenuto I seguenti risultati:

Accuratezza KNN: 90,3%

Accuratezza SVM: 89,9%

4.2 Inception_resnet_v1

Questo modello[19] è stato fatto da Hiroki Taniai e addestrato sul dataset MS-CELEB-1M e prende in input immagini con dimensioni pari a 160x160 px. Quindi anche in questo caso abbiamo bisogno di un ridimensionamento dell'imagine, inquadrando solo la faccia della persona, anche qua abbiamo usato la libreria per l'allineamento della faccia AlignDlib, però questo modello a confronto con nn4.small2.v1 funziona meglio con I punti di riferimento Inner Eyes and Bottom Lip. Tale modello è composto da 426 layers.

Prendiamo in considerazione il secondo dataset, YoutubeFaces DB, e facciamo una suddivisione con 30 persone, ogni persona ha almeno un video diviso in frame, per un totale di 10000 frame. Quando si calcola la Distanza Euclidea tra 2 frame consecutivi, questa è molto piccolo.

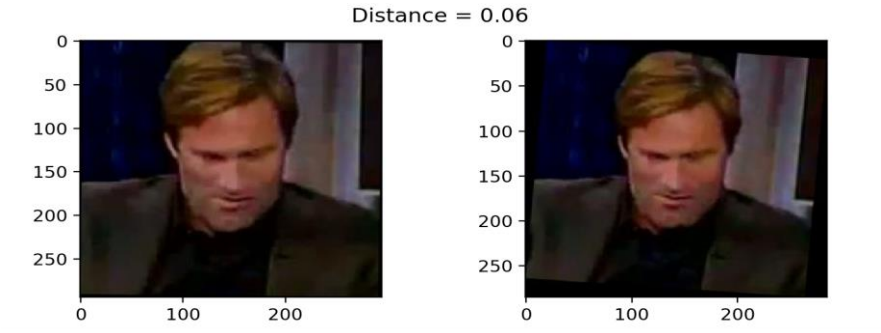


Fig. 9 Distanza Euclidea tra 2 frame consecutivi

Quando si ha 2 frame di 2 video diversi ma comunque della stessa persona, si ha un aumento della Distanza Euclidea.

Fig. X Distanza Euclidea stessa persona 2 frame arbitrari

Ma comunque se si prendono 2 frame di 2 persone diverse la Distanza Euclidea cresce ancora di più.

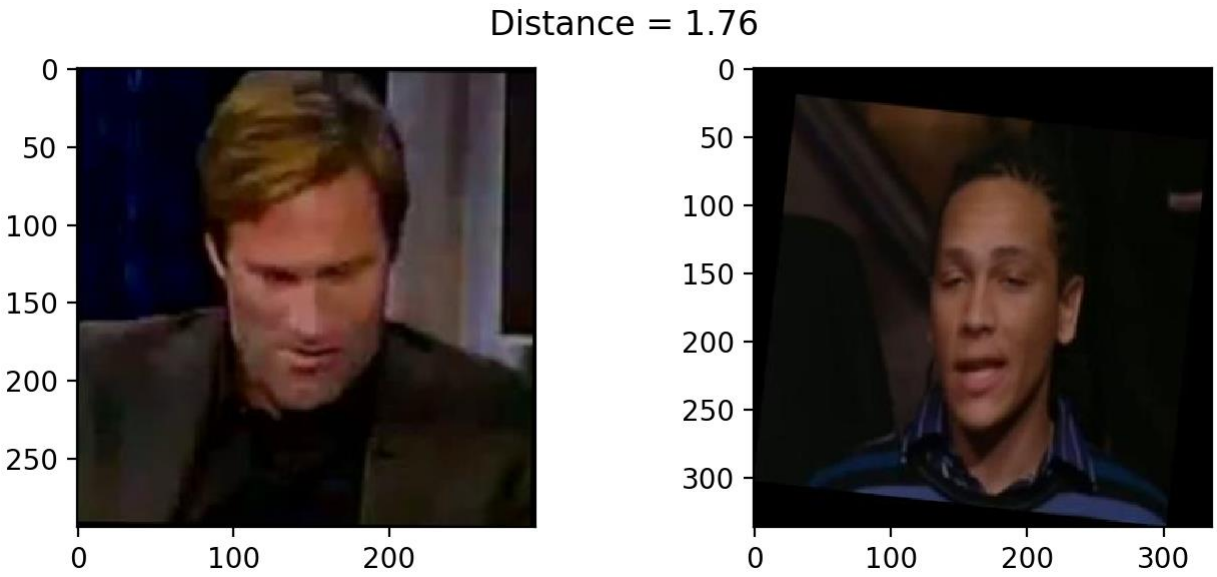


Fig. 10 Distanza Euclidea 2 persone diverse

Infine dopo aver fatto I calcoli per tutte le coppie presenti nella suddivisione del dataset si ha che il modello riesce a predire se su una coppia di frame c'è la stessa persona oppure no con un'accuratezza massima pari a 97,2% alla distanza 0.50. Di seguito vediamo I grafici

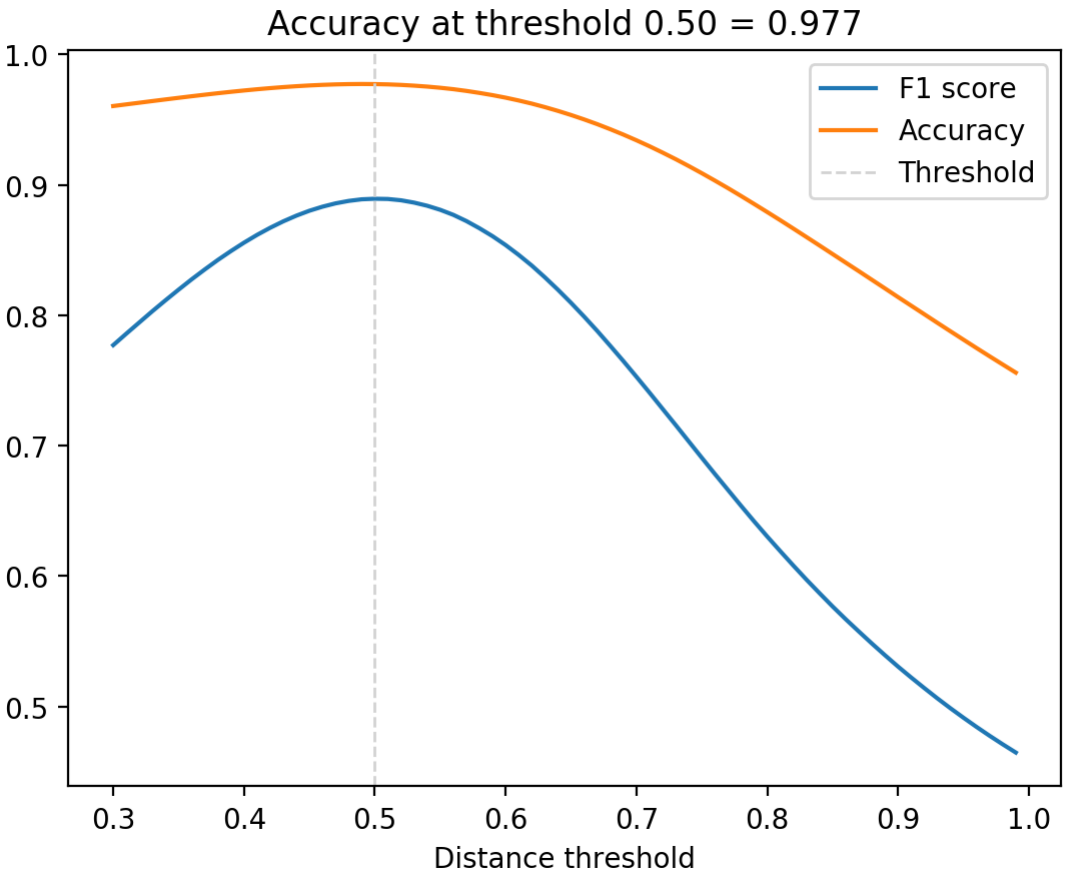


Fig. 11 Andamento delle curve di accuratezza e della misura F del modello inception_resnet_v1 su una suddivisione del dataset Youtube Faces DB

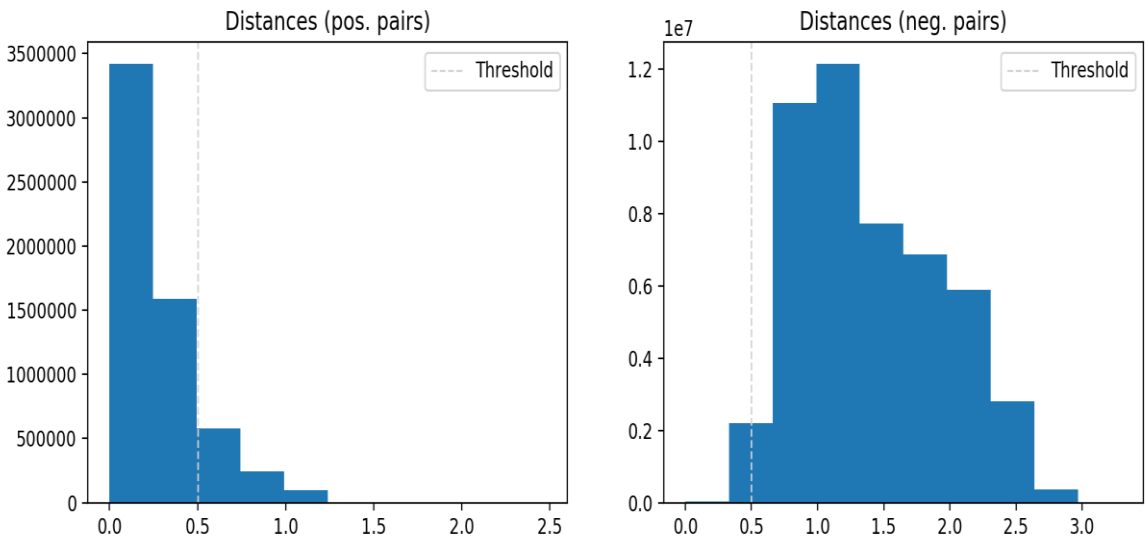


Fig. 12 Grafico del numero di coppie positive e negative del modello inception_resnet_v1 su una suddivisione del dataset YoutubeFaces DB.

I risultati in termini di accuratezza nel riconoscimento delle persone sono:

Usando come dataset una suddivisione del dataset LFW, 143 persone, 4031 immagini

Accuratezza KNN: 71,4%

Accuratezza SVM: 74,9%

Anche qua prima di ottenere questi risultati abbiamo fatto altri 2 tentativi, nel primo abbiamo usato una suddivisone, 200 persone, 3000 immagini, dunque più persone ma meno immagini per ogni persona, ottenendo questi risultati:

Accuratezza KNN: 51,9%

Accuratezza SVM: 55,3%

Al secondo tentative abbiamo usato la suddivisione del LFW, 143 persone, 4031 immagini, e abbiamo utilizzato I punti di riferimento con I quali l'nn4.small2.v1 funzioanva meglio, ovvero Outer Eyes and Nose, ottenendo I seguenti risultati:

Accuratezza KNN: 62,5%

Accuratezza SVM: 63,5%

Usando come dataset una suddivisone del dataset Youtube DB, 30 persone, 10000 immagini

Accuratezza KNN: 95,34%

Accuratezza SVM: 94,29%

Questo risultato è stato ottenuto al primo tentativo, gli altri tentativi, variando il dataset e cambiando I punti di riferimento davano risultati peggiori.

4.3 VGG Face

VGGFace[20] è l'ultimo modello che andremo ad esaminare. È basato sull'architettura del RESNET50 e codifica la faccia di una persona in una rappresentazione di 2048 numeri e prende in input immagini con dimensioni pari a 224x224 px. È composta da un totale di 37 layers. Si usano I punti di riferimento Inner Eyes and Bottom Lip della libreria AlignDlib per l'allineamento della faccia. Usiamo YoutubeFaces DB per illustrare il corretto funzionamento di questo modello. Vediamo che anche qua, quando si prendono 2 frame consecutivi, la distanza euclidea è molto minore rispetto a quando si prendono 2 frame arbitrari della stessa persona, ovviamante la distanza anche qui è molto maggiore quando sulla coppia ci sono 2 frame di 2 persone diverse.

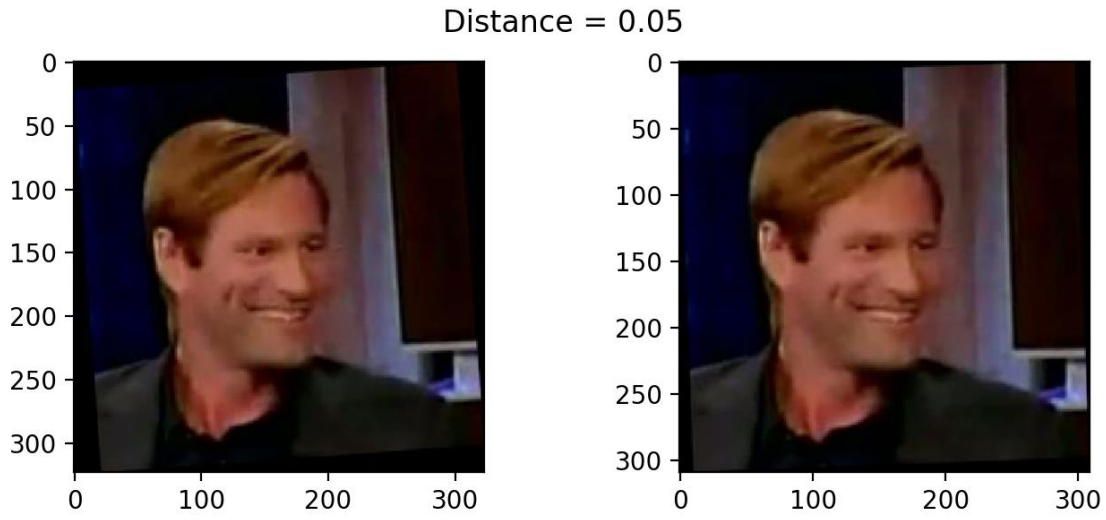


Fig. 13 Distanza euclidea 2 frame consecutivi

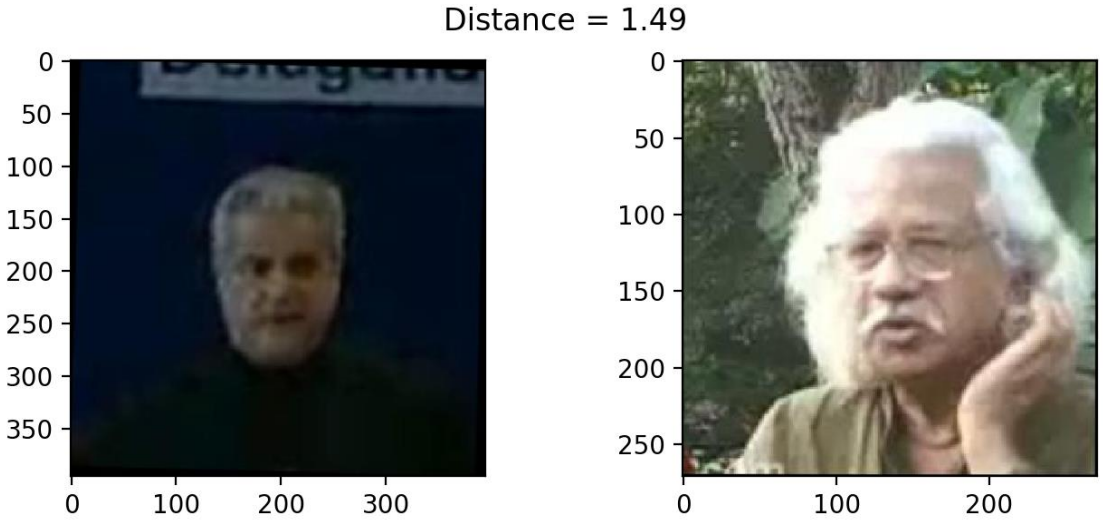


Fig. 14 Distanza euclidea 2 persone diverse

Infine dopo aver fatto I calcoli per tutte le coppie presenti nella suddivisione del dataset si ha che il modello riesce a predire se su una coppia di frame c'è la stessa persona oppure no con un'accuratezza massima pari a 93,4% alla distanza 0.63. Di seguito vediamo I grafici

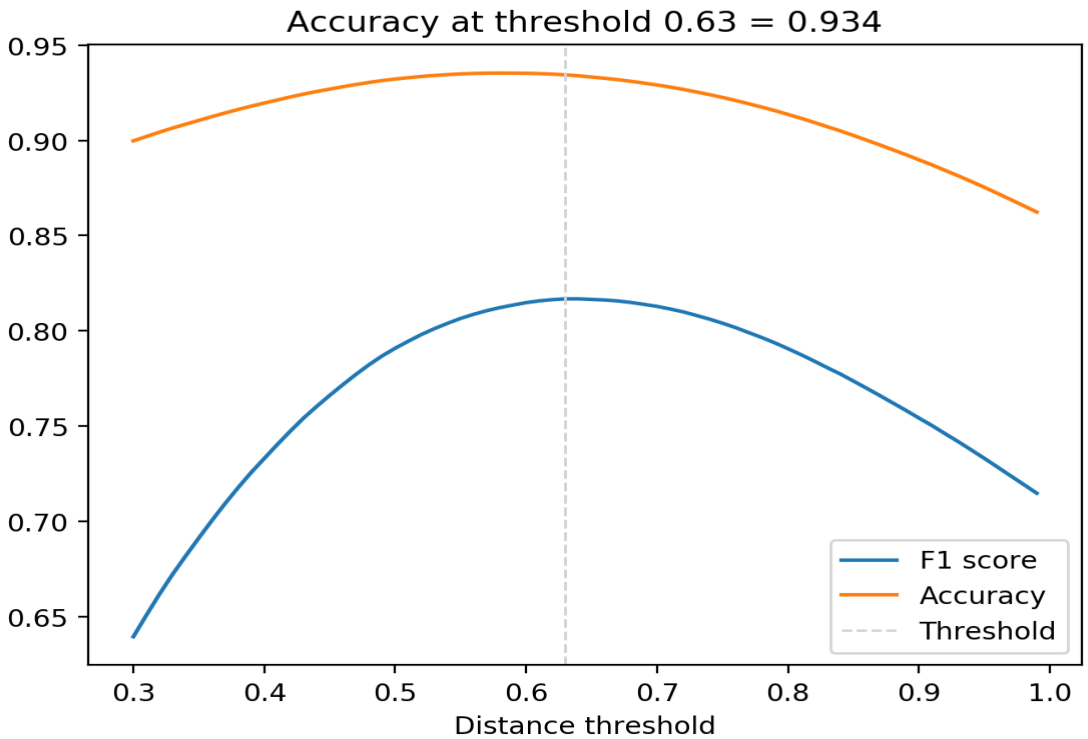


Fig. 15 Andamento delle curve di accuratezza e della misura F

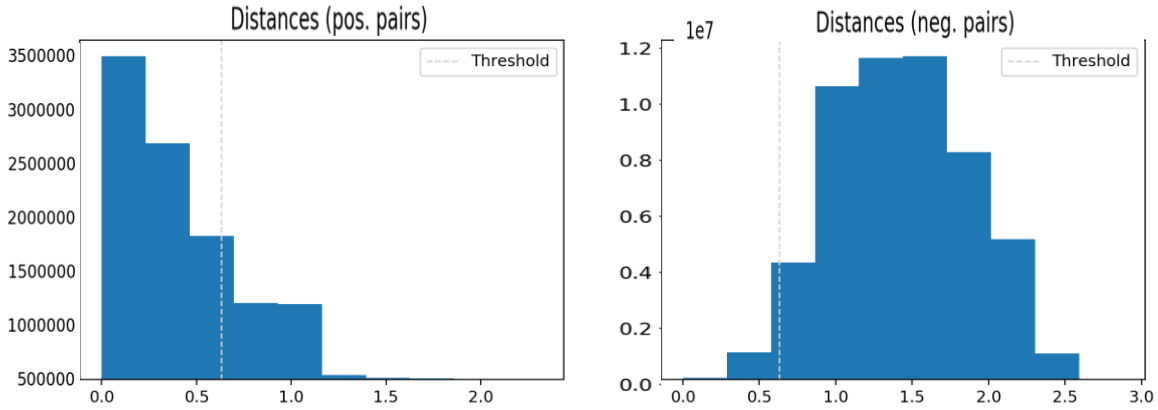


Fig. 17 Grafico del numero di coppie positive e negative

I risultati in termini di accuratezza nel riconoscimento delle persone sono:

Usando come dataset una suddivisione del dataset LFW, 143 persone, 4031 immagini

Accuratezza KNN: 78,3%

Accuratezza SVM: 85,2%

Questi risultati sono stati ottenuti al secondo tentativo, al primo abbiamo utilizzato I punti di riferimento Outer Eyes and Nose ed I risultati sono stati seguenti:

Accuratezza KNN: 70,9%

Accuratezza SVM: 79,2%

Al terzo tentativo abbiamo , usando I punti di riferimento Inner Eyes and Bottom Lip, come nel secondo, ma 250 persone e 5000 immagini, ottenendo I seguenti risultati:

Accuratezza KNN: 72,4%

Accuratezza SVM: 75,8%

Usando come dataset una suddivisone del dataset Youtube DB, 30 persone, 10000 immagini

Accuratezza KNN: 89,1%

Accuratezza SVM: 96,65%

Questo risultato è stato ottenuto al primo tentativo, usando I punti di riferimenti Inner Eyes and Bottom Lip.

Nei tentativi successivi abbiamo cambiato I punti di riferimento ed I risultati erano leggermente peggiori, aumentando invece il dataset a 50 persone e 20000 immagini (intorno a 400 immagini per persona), I risultati erano I seguenti:

Accuratezza KNN:91,5%

Accuratezza SVM: 97,2%

Ma il tempo complessivo nel calcolo della distanza euclidea tra tutte le coppie e addestramento dei classificatori era aumentato di 40 minuti, ed essendo I miglioramenti aumentati di poco, abbiamo ritenuto di tenere il primo tentativo come il migliore.

Capitolo 5

Conclusioni

Abbiamo visto che tutti e 3 modelli possono essere usati per il riconoscimento facciale delle persone, ma in base alle casistiche uno potrebbe essere più funzionante rispetto all'altro.

Per quanto riguarda **nn4.small2.v1**, il tempo complessivo nel calcolo della distanza euclidea di tutte le coppie, addestramento dei classificatori e riconoscimento delle persone è pari a 12 minuti e 31 secondi con la suddivisone scelta del LFW, e 24 minuti e 12 secondi con la suddivisione scelta del YoutubeFaceDB. L'accuratezza nel riconoscimento facciale dei classificatori è :

Usando LFW

- KNN: 83,6%
- SVM:86,2%

Usando YoutubeFaces DB

- KNN: 98,1%
- SVM:97,65%

Per quanto riguarda **inception_resnet_v1**, il tempo complessivo nel calcolo della distanza euclidea di tutte le coppie, addestramento dei classificatori e riconscoimento delle persone è pari 15 minuti e 14 secondi con la suddivsione scelta del LFW, e 19 minuti e 39 secondi con la suddivisone scelta del YoutubeFaceDB.

L'accurateza nel riconoscimento facciale dei classificatori è:

Usando LFW

- KNN: 71,6%
- SVM:74,2%

Usando YoutubeFaces DB

- KNN: 89,37%
- SVM:93,65%

L'ultimo modello scelto, il **VGGFace** ha avuto prestazioni peggiori rispetto agli altri 2, il tempo complessivo nel calcolo della distanza euclidea di tutte le coppie, addestramento dei classificatori e riconscimento delle persone è pari 45 minuti e 48 secondi con la suddivsione scelta del LFW, e 83 minuti e 41 secondi con la suddivisone scelta del YoutubeFaceDB.

L'accurateza nel riconoscimento facciale dei classificatori è:

Usando LFW

- KNN: 78,3%
- SVM:85,2%

Usando YoutubeFaces DB

- KNN: 89,1%
- SVM:96,65%

Ovviamente il tempo complessivo dipende dalle prestazioni della macchina sulla quale si lavora, con le configurazioni hardware descritte nel capitolo 2 I risultati sono molto scarsi, ovviamente anche la dimensione del dataset influisce, più il dataset è grande, ovvero tante persone e per ogni persona c'è un insieme grande di immagini, il tempo complessivo cresce notevolmente però in termini di accuratezza nel riconoscimento facciale aumenta.

Se consideriamo il controllo dei passaporti in aeroporto, il modello che si preste meglio è il **nn4.small2.v1**, infatti l'accuratezza nel riconoscimento facciale è la più altra tra l 3, in più si basa sugli occhi e il naso come punti di riferimento che è un vantaggio per le foto del passaporto.

Se consideriamo il monitoraggio di soggetti nuovi, l'**inception_resnet_v1** si potrebbe prestare meglio dei altri 2 modelli, infatti dai risultati ottenuti abbiamo visto che per quanto riguarda una successione di frame, inception_resnet_v1 è il più veloce ed anche l'accuratezza nel riconoscimento facciale usando il classificatore SVM è buona, circa 96%. **VGGFace** anche se ha risultati migliori in termini di accuratezza a confronto con l'**inception_resnet_v1** rimane comunque peggiore per il problema del tempo.

Bibliografia

- [1] David Stutz , “Understanding Convolutional Neural Networks”
- [11] S. Ji, W. Xu, M. Yang e K. Yu, «3D Convolutional Neural Networks for Human Action Recognition,» IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 35, n. 1, pp. 221-231, 2013
- [17] C. Corinna e V. Vapnik, «Support-vector networks,» Machine learning 20.3, pp. 273-297, 1995.

Sitografia

- [2] Tutto su Python : https://www.python.it/about/
- [3] Download link PyCharm: https://www.jetbrains.com/pycharm/download/#section=windows
- [4] https://pypi.org/
- [5] Documentazione NumPy https://numpy.org/
- [6] Documentazione os: https://docs.python.org/3/library/os.html
- [7] Documentazione OpenCV: https://www.geeksforgeeks.org/python-opencv-cv2-polylines-method/
- [8] Documentazione matplotlib: https://matplotlib.org/3.1.3/index.html
- [9]Documentazione Keras: https://keras.io/
- [10] Documentazione Tensorflow: https://www.tensorflow.org/
- [12] I. Goodfellow, Y. Bengio e A. Courville, Deep Learning, 2015.
- [13] NVIDIA, «Elaborazione in parallelo CUDA,» [Online]. Available: https://www.nvidia.it/object/cuda-parallel-computing-it.html
- [14] http://vis-www.cs.umass.edu/lfw/
- [15] https://www.cs.tau.ac.il/~wolf/ytfaces/
- [16] https://towardsdatascience.com/
- [18] https://cmusatyalab.github.io/openface/models-and-accuracies/
- [19] https://github.com/nyoki-mtl/keras-facenet
- [20] https://gist.github.com/EncodeTS/6bbe8cb8bebad7a672f0d872561782d9

Figure

Ringraziamenti

Ringrazio il Prof. Aldo Franco Dragoni per la disponibilità dedicata per lo sviluppo di questa tesi e per aver dato la possibilità di affrontare temi completamente nuovi.

Ringrazio il Dott. Sernani Paolo per la disponibilità dedicata e per essermi sempre a disposizione nel chiarimento di ogni dubbio, per tutto il materiale proposto nel affrontare questo argomento completamente nuovo per me, e soprattutto per la pazienza.