



UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

*Corso di Laurea in Informatica per la
Comunicazione Digitale*

Dataset e riconoscimento automatico di volti
coperti da mascherina

Relatore: Dott. Giuliano Grossi
Correlatore: Dott. Sathya Bursic

Tesi di:
Vlad Florin Sodringa
Matricola: 910660

Anno Accademico 2019-2020

Indice

Introduzione	2
1 Reti neurali per la classificazione di immagini	3
1.1 Introduzione alle reti neurali artificiali	3
1.1.1 Training di una rete neurale artificiale	5
1.2 Convolutional Neural Networks	8
1.2.1 Un po' di storia	9
1.2.2 Layer presenti nelle CNN	10
1.2.3 Vantaggio di una CNN rispetto ad una rete fully connected .	14
1.3 Inception-ResNetV1	15
1.3.1 Residual blocks	19
1.3.2 Modulo Inception	21
2 Dataset utilizzati	23
2.1 MaskedFace-Net	23
2.2 MAFA	24
2.3 Medical Masks Dataset	25
2.4 FFHQ: Flickr Faces High Quality	26
2.5 Creazione del dataset utilizzato nel progetto	27
3 Classificazione di volti occlusi da mascherine	30
3.1 InceptionResNetV1 finetuning	30
3.2 Data augmentation	31
4 Risultati ottenuti	34
4.1 Metriche utilizzate	34
4.2 Training e test eseguiti	35
Conclusioni	47
Bibliografia e sitografia	48
Ringraziamenti	51

Introduzione

L'obiettivo di questa tesi è quello di documentare il lavoro di tirocinio interno svolto riguardo l'utilizzo delle reti neurali per la classificazione di volti occlusi da mascherine, in modo da riconoscere la presenza della mascherina oppure il corretto posizionamento della stessa sul volto. L'utilizzo di dispositivi per la protezione individuale, soprattutto mascherine per la protezione degli individui da malattie trasmissibili per via aerea, è un argomento attuale a causa della pandemia causata dal virus COVID-19.

Il lavoro di tesi svolto si sviluppa in due macrotemi principali: la creazione di un dataset composto da immagini che raffigurano volti occlusi da mascherine in modo corretto, errato e volti non occlusi da esse e l'utilizzo di un modello basato sulle reti neurali convoluzionali, in modo da creare un classificatore che riesca a classificare l'immagine in queste tre classi, con il tasso di errore più basso possibile.

Il dataset creato non è stato creato da zero, bensì sono stati utilizzati molteplici dataset pubblicati come MaskedFace-Net, MAFA, Medical Masks Dataset e FFHQ per la costruzione del dataset utilizzato. Il dataset di training utilizza, per le due classi con i volti occlusi da mascherina, immagini sintetiche, cioè volti occlusi da mascherine creati applicando tramite trasformazioni varie, la mascherina sul volto del soggetto. L'utilizzo di un dataset sintetico è dato dalla presenza ridotta di dataset contenenti immagini di volti che indossano mascherine in modo sbagliato ed errato. Il modello di classificazione utilizzato è una rete neurale convoluzionale chiamata Inception-ResNetV1, implementato e pre addestrato sul dataset VGG-Face2, un dataset creato per il riconoscimento facciale, in modo da riutilizzare l'apprendimento precedente per i nostri scopi tramite tecniche di finetuning.

La soluzione presentata dalla tesi permette di riconoscere ha un accuratezza del 98% , introducendo la terza classe dei volti che indossano mascherine in modo errato, l'accuratezza complessiva del modello si abbassa al 95%, con l'accuratezza della classe introdotta pari al 90%.

1 Reti neurali per la classificazione di immagini

1.1 Introduzione alle reti neurali artificiali

Le reti neurali artificiali (ANN, Artificial Neural Network), sono modelli matematici computazionali che si inspirano alle reti neurali biologiche create dalle cellule neuronali. Questi modelli sono molto versatili e vengono applicati ad una miriade di problemi: regressioni, clustering, classificazioni, riconoscimento vocale, predizioni di serie temporali etc.

Una rete neurale artificiale è composta da neuroni artificiali, interconnessi tra loro. Un neurone artificiale è una semplice funzione matematica che cerca di modellare il funzionamento di un neurone biologico, il quale riceve uno o più segnali dall'esterno, tramite i dendriti, processa ed integra il segnale nel corpo cellulare e, infine, il segnale elaborato viene propagato tramite i suoi assoni.

Il neurone artificiale riceve in input uno o più valori pesati con un certo valore, in seguito questi valori vengono sommati tra di loro, viene poi sommato un bias ed infine viene applicata una funzione di attivazione. Avendo in input n segnali $x_0 \dots x_{n-1}$ e $w_{k0} \dots w_{kn-1}$ pesi, b come bias ed $f(x)$ come funzione di attivazione, l'output del neurone k -esimo è dato da:

$$y_k = f\left(\sum_{j=0}^{n-1}(w_{kj} x_j)\right) + b$$

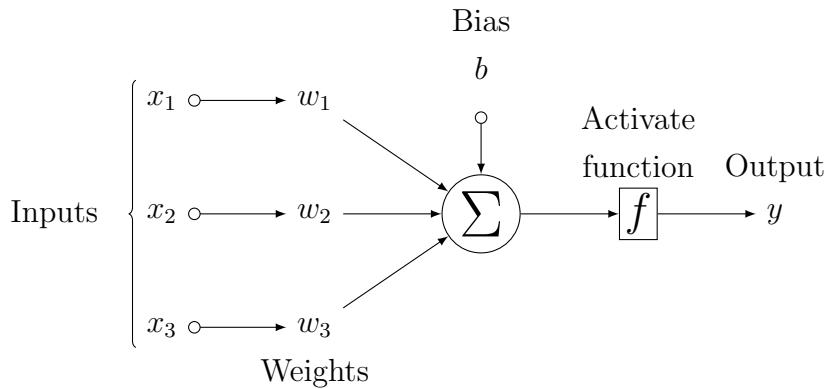


Figura 1: Struttura del neurone artificiale

La funzione di attivazione $f(x)$ permette di introdurre una componente di non linearità nel modello sopra descritto, rendendolo più complesso. I tipi di funzioni di attivazioni più utilizzate sono: lineari, rettificatrici (e.g. ReLU: Rectifier Linear Unit, leaky ReLU), e sigmoidi (e.g. logistica).

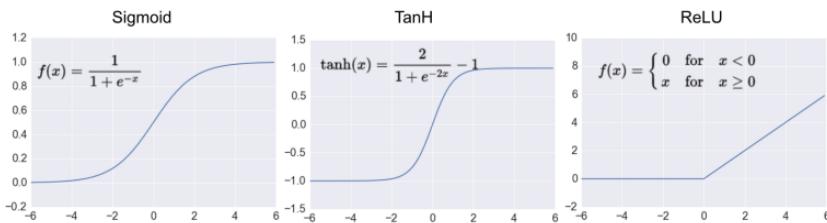


Figura 2: Grafici e formule delle tre funzioni di attivazione più utilizzate nel deep learning

Le Deep Neural Network (DNN) sono reti neurali artificiali, costruite a strati connessi tra di loro in modo da creare un grafo diretto aciclico (DAG), le quali contengono molti livelli tra il livello di input ed il livello di output. Questi livelli vengono chiamati livelli nascosti, e il numero di questi è un iperparametro del modello: più i livelli aumentano, più la complessità della DNN aumenta (diventando più "deep"). Questo comporta nella maggior parte dei casi una maggiore accuratezza del modello a discapito di un tempo di addestramento e di complessità computazionale maggiore, per questo motivo il parametro scelto deve essere

il giusto compromesso tra queste due caratteristiche e può dipendere dal tipo di applicazione e dalla disponibilità di potenza computazionale.

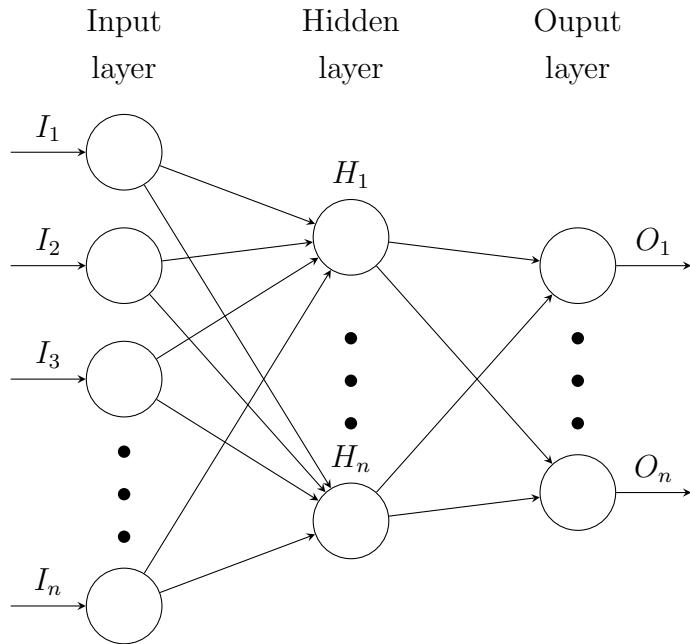


Figura 3: Struttura di una Deep Neural Network (DNN)

1.1.1 Training di una rete neurale artificiale

Per addestrare la rete neurale artificiale, in modo che possa svolgere il compito per cui la si vuole utilizzare (regressione, classificazione etc.), bisogna eseguire una fase di training, o allenamento, dove la rete "impara" a svolgere un determinato compito. In questa fase, la rete riceverà in input un insieme di dati (training set), che verranno elaborati da essa per ottenere l'output della rete (ad esempio dei numeri nel caso di regressione oppure una certa probabilità nel caso sia un lavoro di classificazione).

A questo punto, si confronta l'output della rete con il valore reale atteso e si calcola l'errore commesso utilizzando una funzione di costo o di errore, chiamate loss function. Le loss function più utilizzate sono la Mean Square Error, per i

problemi di regressione e la Cross Entropy Error, chiamata anche Log Loss, per i problemi di classificazione.

Una volta computato l'errore, bisogna modificare i vari pesi e bias dei neuroni della rete, in modo da diminuire l'errore e ottenere un modello più accurato ed efficiente. Questo problema è un problema di ottimizzazione matematica, lo scopo è trovare un minimo locale nel gradiente dell'errore della rete.

Per fare ciò si utilizza l'algoritmo di Backpropagation [**backprop**], che permette di computare i gradienti della loss function, per ogni parametro della rete, in modo efficiente. Questo algoritmo calcola quanto le connessioni dell'output abbiano contribuito sull'errore. Successivamente l'algoritmo applica la stessa logica per gli altri layer, computando quanto le connessioni neurali del layer in considerazione abbiano contribuito sull'errore, fino ad arrivare al layer di input.

In seguito si applica uno step di Gradient Descent [1], un algoritmo iterativo, che permette di trovare il minimo locale di una funzione, in questo caso la funzione di loss calcolata con Backpropagation. Viene eseguito aggiustando i pesi e i bias, secondo un certo learning rate, secondo il gradiente calcolato, in modo da affinare la rete. Questo processo viene ripetuto per un certo numero di volte, finché non si arriva alla convergenza del modello, cioè quando la loss diven. Ogni passaggio ha il nome di epoca.

Gradient Descent Gradient Descent è un algoritmo iterativo che permette di trovare un minimo locale in una funzione differenziabile. Avendo il gradiente calcolato per un certo punto, l'algoritmo fa un passo nella direzione opposta, andando verso un minimo, tanto quanto il suo parametro principale, cioè il learning rate.

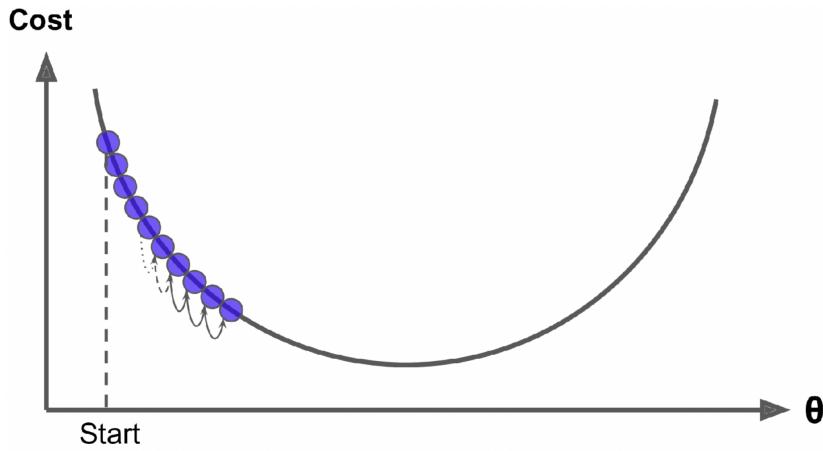


Figura 4: Gradient Descent

Se il learning rate è troppo basso, allora ci vorranno più iterazioni per trovare un minimo locale, mentre se è troppo grande, allora è probabile che l'algoritmo non riesca a raggiungere il minimo locale.

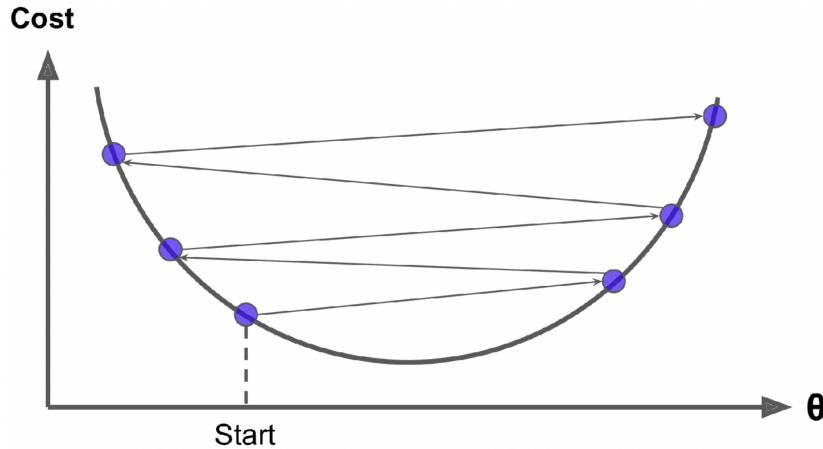


Figura 5: Gradient Descent con un learning rate troppo alto. Il minimo non è raggiungibile

L'algoritmo utilizzato nel training delle reti neurali non è Gradient Descent, ma Stochastic Gradient Descent oppure Mini-batch Stochastic Descent [1] nel caso si utilizzino più batch. La differenza sostanziale consiste nel computare i gradienti solamente per un dato elemento oppure una sola batch del training set rispetto

all’intero training set. Il percorso dell’algoritmo è più rumoroso e meno regolare, ma molto più veloce. Il rumore e la mancanza di regolarità permette all’algoritmo di sfuggire ai minimi locali, così da avere probabilità più alte nel trovare il minimo globale della loss function. Il risultato di questi due ultimi algoritmi non è ottimale.

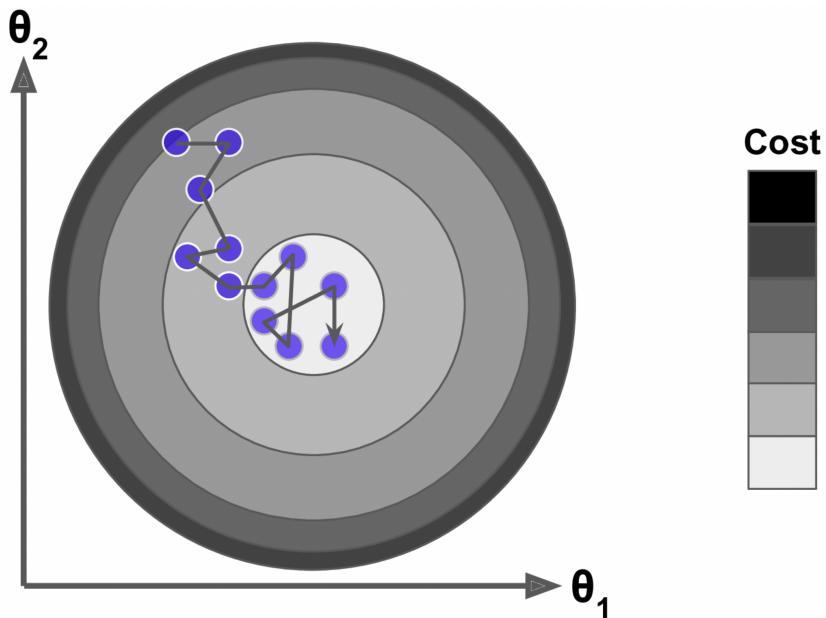


Figura 6: Stochastic Gradient Descent. Si nota come il percorso è rumoroso e non regolare

1.2 Convolutional Neural Networks

Le reti neurali che vengono utilizzate maggiormente per la classificazione di immagini si chiamano Convolutional Neural Networks (CNN o ConvNet).

Le CNN sono una classe di reti neurali artificiali che è diventata di riferimento per risolvere problemi negli ambiti della Computer Vision, nello specifico per fare detection, segmentation e classificazione di immagini e di video, ma possono essere utilizzate anche in campi come il riconoscimento vocale e il Natural Language Processing. In generale, sono efficaci per problemi ed elaborazioni che utilizzano dati di tipo matriciale.

Le CNN sono utilizzate anche per l'estrazione di feature o pattern da immagini, senza l'utilizzo manuale di algoritmi di feature extraction come HoG [2], SIFT [3], SURF [4].

Uno dei vantaggi principali dell'utilizzo di reti neurali è il possibile riutilizzo per altri scopi, simili a quello per cui sono state create ed allenate inizialmente. Questo può essere fatto tramite transfer-learning e finetuning in modo da riutilizzare i pattern appresi in precedenza, come nel progetto di cui tratta la tesi.

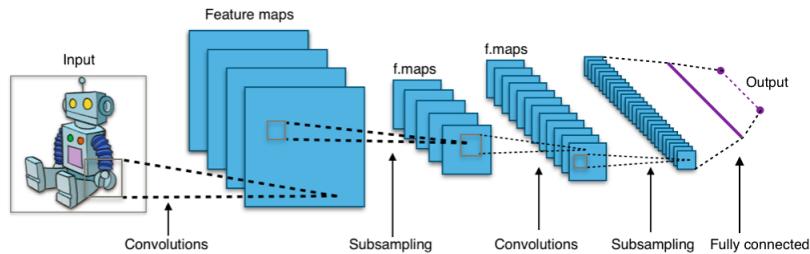


Figura 7: Architettura standard di una rete CNN

1.2.1 Un po' di storia

Gli studi sulla corteccia visiva degli anni '50 e '60, come gli esperimenti di Hubel e Wiesel [5] su gatti e sulle scimmie, mostrarono come alcuni neuroni rispondono alla visualizzazione di determinati pattern o feature, come ad esempio linee o bordi. Inoltre, le loro ricerche, hanno rilevato che alcuni neuroni vengono attivati per feature semplici, mentre altri invece reagiscono a pattern più complessi, combinando le risposte dei livelli inferiori grazie a campi recettivi più grandi.

Gli esperimenti di Hubel e Wiesel, hanno permesso a K. Fukushima di sviluppare il Neocognitron [6] negli anni ottanta e in seguito, nel 1998, Yann LeCun et al. [7] ad introdurre i convolutional layers (e perciò le CNN) con la rete LeNet-5. Lo scopo principale della rete LeNet-5 era quello di riconoscere i numeri scritti a mano (vedi MNIST [8]) e veniva, infatti, utilizzata da servizi postali e banche per la lettura automatica degli assegni.

1.2.2 Layer presenti nelle CNN

Convolutional layers Il layer più importante di una CNN (e il layer che differenzia una CNN da una rete neurale fully connected) è il convolutional layer. Come da nome, i convolutional layers utilizzano l'operazione di convoluzione tra due funzioni, che consiste nell'integrare il prodotto tra la prima e la seconda funzione traslata di un certo valore. Questa operazione è molto simile all'operazione di correlazione incrociata.

Questi layers utilizzano dei filtri, chiamati anche kernel, che non sono altro che matrici multidimensionali che contengono dei pesi. Queste matrici vengono fatte scorrere, pixel per pixel, su tutto il tensore dell'input e ad ogni scorrimento computano la somma pesata dei valori dei pixel. Praticamente, ogni neurone del livello superiore, è collegato solamente con i neuroni collocati nel rettangolo delle dimensioni del filtro e tutti i pesi ed i bias sono gli stessi, per ogni neurone.

La dimensione del filtro in altezza e in larghezza è da considerare come un iperparametro. I valori più utilizzati sono 3x3 oppure 5x5. La profondità di un filtro, invece, è sempre uguale al numero dei canali del layer di input (oppure i canali dell'immagine).

Ogni convolutional layer ha diversi filtri e ogni filtro produce una feature map, che corrisponde all'output della convoluzione tra il filtro e i neuroni dell'input. Anche questo è un iperparametro da configurare. Ciascun neurone di una feature map condivide gli stessi pesi e gli stessi bias degli altri.

Un convolutional layer permette di configurare altri parametri oltre alla dimensione del filtro ed il numero di filtri: lo *zero-padding* e lo *stride*. Il primo risolve il vincolo di dover avere input ed output delle stesse dimensioni, applicando nella matrice di input una cornice di zeri, in modo da allargare la matrice e mantenere, in questo modo, le stesse dimensioni. Lo stride, invece, è la velocità di traslazione del filtro, e permette di diminuire la complessità del modello permettendo al filtro di scorrere più velocemente. Ad esempio, con lo stride impostato a 2, per ogni passo, il filtro si sposterà di due neuroni in orizzontale e di 2 neuroni in verticale, in modo da avere un output ancora più ridotto. Il valore dello stride può essere diverso per

ogni dimensione.

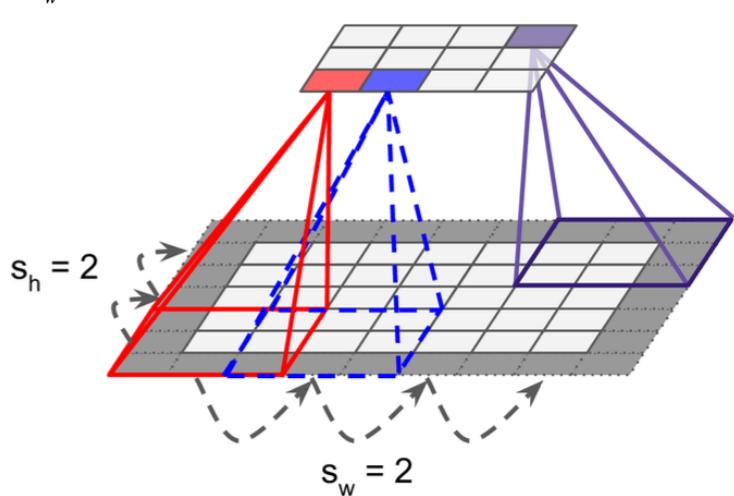


Figura 8: Esempio di layer convolutivo 3×3 con zero padding e stride 2. Dalla figura si può notare come lo stride riduca di molto la dimensionalità di output.

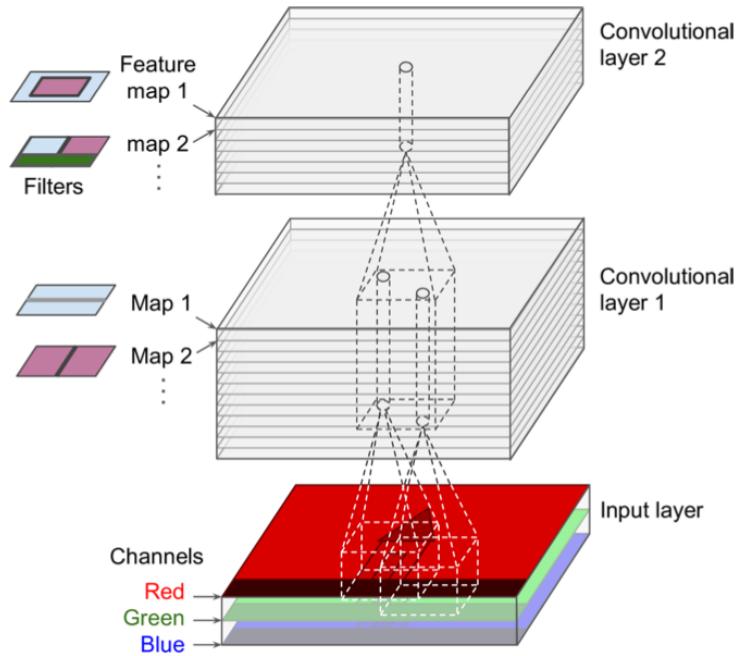


Figura 9: Figura riassuntiva di come funzionano i layer convolutivi. Si può vedere come due layer convolutivi, con molteplici filtri, agiscono su un’immagine RGB, creando feature map. Da notare come ogni kernel ha come profondità il numero di feature maps (o canali) di input.

Pooling layer Il pooling layer permette di fare un sub-sampling, cioè un’estrazione di un sottoinsieme di informazioni dall’input, in modo da ridurre ancora una volta il carico computazionale della rete neurale, anche se la perdita di informazione non è banale. Il pooling layer funziona analogamente al convolutional layer: una finestra viene fatta scorrere sulla matrice in input e, ad ogni scorrimento, applica una funzione in modo da estrarre un valore unico dai neuroni nella finestra. Alcune funzioni applicabili sono la media, prendere il valore massimo della finestra oppure il minore.

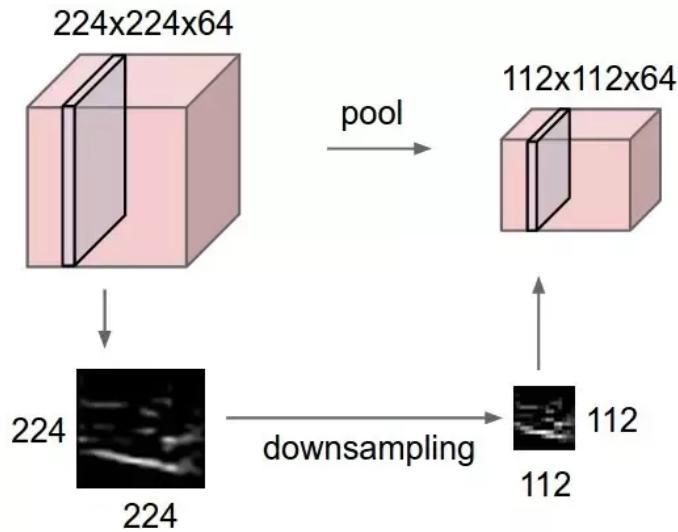


Figura 10: Esempio di come funziona un layer di pooling, in questo caso max pooling

Il pooling layer più utilizzato è il max-pooling, che restituisce il valore massimo presente nella finestra. La sua caratteristica più importante è quella di avere una proprietà invariante su piccole trasformazioni (e.g. traslazioni, piccole rotazioni etc.), dato che la CNN per se è invariante su traslazioni, ma non su altri tipi di trasformazioni. Un altro motivo per il suo utilizzo è scartare le feature più deboli e mantenere le feature più importanti, le quali probabilmente avranno un valore più, anche se secondo Andrew Ng: "Il motivo principale per il quale le persone utilizzano max-pooling è dato dal fatto che è stato utilizzato in molteplici esperimenti, [...] non conosco nessuno che sappia se è davvero quello il motivo per cui il max-pooling funziona bene sulle convnets" [9].

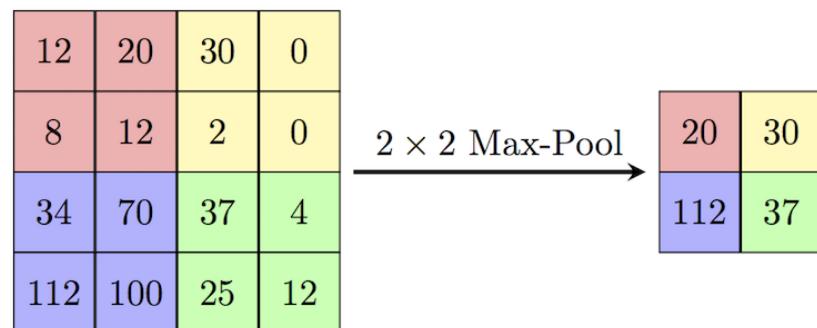


Figura 11: Esempio di max pool 2x2

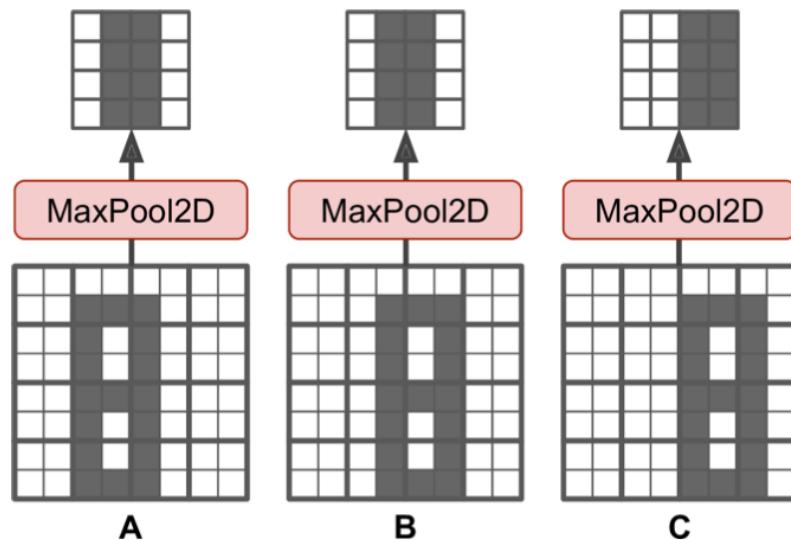


Figura 12: Proprietà inviarante del layer max pooling su piccole traslazioni

1.2.3 Vantaggio di una CNN rispetto ad una rete fully connected

Un layer fully connected permette di processare solamente un vettore unidimensionale. È possibile utilizzare un layer fully connected per problemi di classificazione di immagini, benchè siano piccole, utilizzando un layer di flattening, in modo da convertire un'immagine in un vettore unidimensionale.

Per immagini molto grandi, è computazionalmente pesante utilizzare reti fully connected. Se consideriamo, ad esempio, un'immagine a colori con tre canali R, G e B, di dimensioni 128x128, comporta ad avere 49152 pesi per ogni singolo neurone. Se le dimensioni aumentano e la rete è abbastanza profonda, cioè con tanti layer nascosti, si può notare come la soluzione è boriosa.

Dato che la struttura delle CNN permette di lavorare facilmente con dati di tipo matriciale, è possibile sfruttare tre proprietà:

- Equivarianza sulle traslazioni: la CNN riesce a riconoscere le feature che ha imparato anche se la feature è stata sottoposta a traslazioni o anche piccole trasformazioni affini/proiettive (mancante nei layer fully connected)
- Condivisione dei pesi: i pesi di ogni campo recettivo (i.e. kernel, filtro) sono gli stessi per ogni neurone di output
- Connessioni sparse: i layer non sono fully connected, ma ogni neurone di output è collegato con i neuroni del suo campo recettivo

1.3 Inception-ResNetV1

Inception-ResNetV1 [10] è una architettura progettata da Google Research, da Szegedy et al. L'architettura combina le residual connection [11], introdotte da He et al., ed i moduli Inception, introdotti per la prima volta nella rete GoogLeNet [12]. Il motivo principale dell'unione di questi due moduli è quella di avere sia l'efficienza e l'efficacia computazionale del modulo Inception, sia la capacità del residual block di ottenere buoni risultati da reti con molti layer.

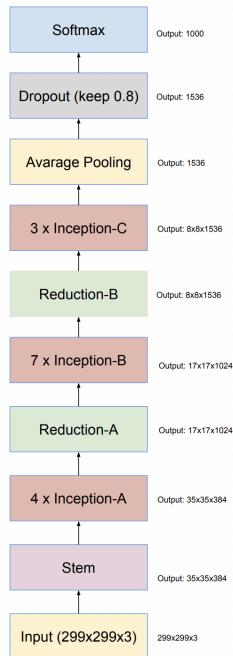


Figura 13: Architettura di Inception-ResNetV1

L'architettura di Inception-ResNetV1 è composta da una prima parte chiamata *stem*, il corpo centrale, composto da layer convolutivi e di max-pooling e da una successione di layer Inception e Reduction. I layer Reduction, vengono utilizzati per diminuire la dimensionalità, con filtri convoluzionali che applicano lo stride.

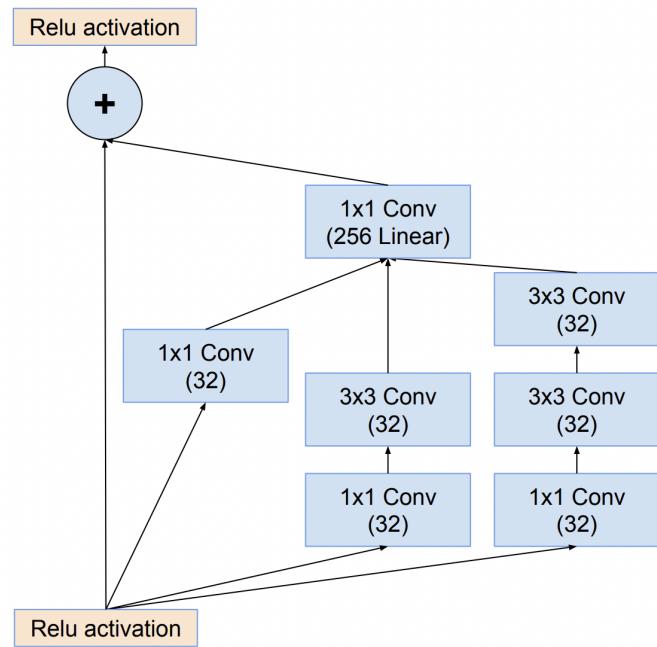


Figura 14: Esempio di un Inception layer, in questo caso Inception-A

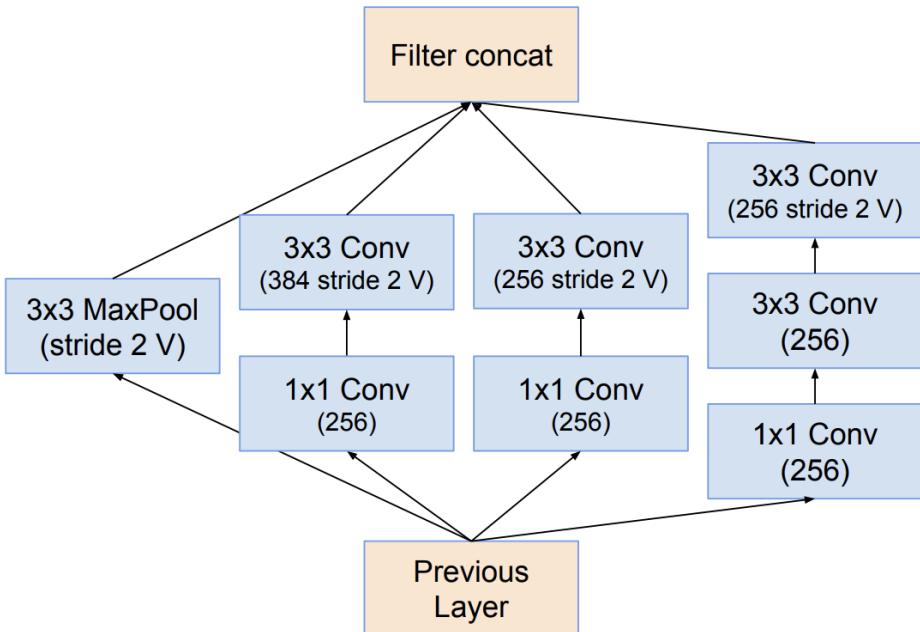


Figura 15: Esempio di un reduction layer, in questo caso Reduction-B. Si nota l'applicazione di filtri convoluzionali con stride in modo da ridurre la dimensionalità.

Infine viene applicato l'average pooling, un dropout del 20% per diminuire l'overfitting e la funzione softmax per la classificazione. Sia i layer Inception che Reduction sono layer Inception, la differenza tra i due sta nel fatto che Reduction utilizza lo stride in modo da ridurre la dimensionalità dell'output, mentre tutti i moduli Inception utilizzano skip connection. La differenza tra le versioni A, B e C dei layer è spiegata in [10]. La funzione di attivazione utilizzata è la ReLU.

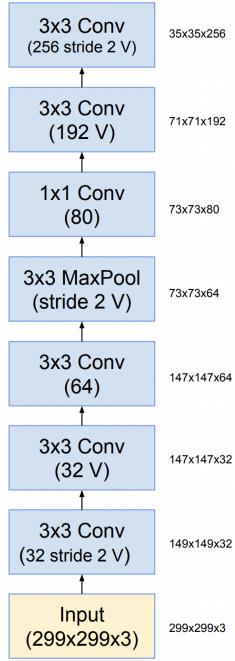


Figura 16: Stem o corpo centrale di InceptionResNetV1. I layer indicati con V (valid) non utilizzano zero-padding, mentre quelli non indicati con V (same) utilizzano zero-padding se necessario

1.3.1 Residual blocks

In teoria, una rete neurale più profonda permetterebbe di ottenere risultati migliori, avendo la possibilità di generalizzare maggiormente. Purtroppo però, non sempre, aumentare il numero di layer in una rete deep, porta a risultati e ad un'accuratezza migliore. Secondo [13], questo fenomeno non è dato dall'overfitting, che una rete con più livelli può avere, ma è stato dimostrato empiricamente che una rete neurale con più layer può avere ripercussioni negative sull'accuratezza del modello, anche riducendo la dimensione dei filtri.

He et al. con l'introduzione di ResNet e del residual block, hanno proposto una nuova architettura in modo da ovviare a questo problema. L'idea è quella di aggiungere comunque più layer ad una rete neurale ma questi nuovi layer aggiunti devono riuscire a ricreare o avvicinarsi alla funzione identità. In una rete normale

ciò non accade, dato che molto spesso i pesi vengono inizializzati con valori intorno allo zero, perciò un layer riesce facilmente a replicare la funzione zero, ma non la funzione identità.

Il residual block utilizza le skip connection: il segnale in input x viene propagato e sommato all'output dei layer. Avendo come funzione obiettivo $H(x) = \mathcal{F}(x) + x$, allora il layer dovrà modellare la funzione $\mathcal{F}(x) = H(x) - x$, cioè la funzione residuo. I layer dovranno quindi modellare solamente ciò che differenzia l'input dall'output: la funzione residuo. Se l'inizializzazione dei pesi è intorno allo zero, allora il blocco modellerà la funzione identità e durante il training, la funzione imparerà a discostarsi dalla funzione identità.

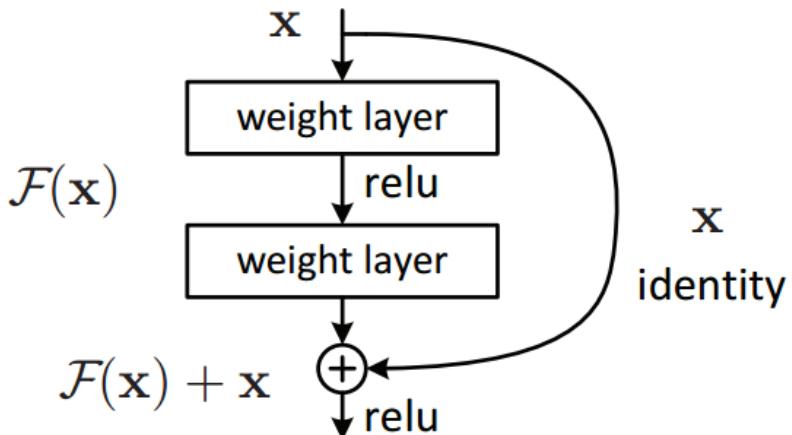


Figura 17: Residual block di ResNet

Nel caso la dimensionalità (numero di feature maps) dell'output e dell'input, dei layer di un residual block, sia diversa, è necessario utilizzare un convolutional layer 1×1 con lo stesso numero di feature maps dell'output dei layer, in modo da avere lo stesso numero di feature maps e poter sommare x e $\mathcal{F}(x)$.

He et al. con questa architettura ha permesso l'utilizzo di reti ancora più profonde, in modo da aumentare l'accuratezza dei modelli.

1.3.2 Modulo Inception

Il modulo Inception, introdotto nell’architettura GoogLeNet, permette di ottenere una computazione più efficiente utilizzando più layer convolutivi e di max-pooling sullo stesso livello, facendoli lavorare in parallelo. L’output finale del modulo Inception è la concatenazione di tutte le feature maps prodotte dai layer convolutivi e di max pooling presenti.

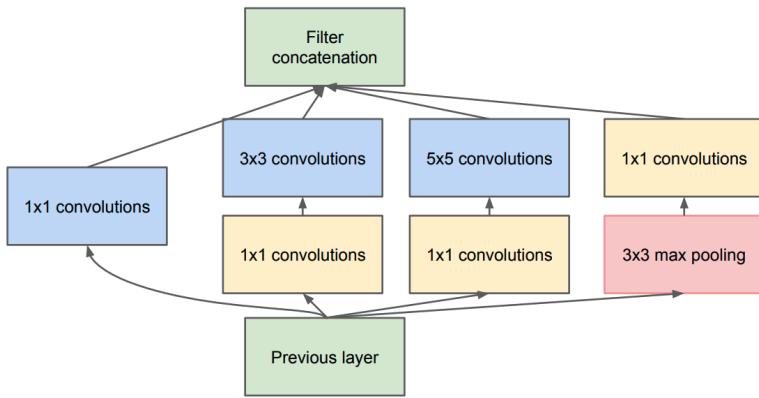


Figura 18: Modulo Inception

Come possiamo vedere dallo schema, le componenti principali di un modulo Inception sono tre: convoluzioni 1x1, convoluzioni 3x3 e 5x5 ed infine il layer di max pooling. Illustriamo brevemente lo scopo di ogni tipo di layer:

- **Convoluzioni 1x1:** questi filtri permettono di ridurre la dimensionalità dell’input, riducendo il numero di feature maps, producendo un effetto ”collo di bottiglia”, così da ridurre il carico computazionale. I filtri convoluzionali 1x1 inoltre riescono comunque a dare un contributo nella rilevazione dei pattern attraverso i feature maps in input, aumentando le prestazioni della rete.
- **Convoluzioni 3x3 e 5x5:** permettono di imparare e rilevare pattern spaziali su tutte e tre le dimensioni, variando anche la scala grazie all’utilizzo di filtri con misure diverse

- **Max pooling:** i benefici sono gli stessi spiegati nell'introduzione alle CNN, invariazione alle piccole traslazioni e l'estrazione delle feature più forti.

Ricapitolando, la filosofia del modulo Inception consiste nell'implementazione di un'architettura sia profonda che larga, utilizzando più filtri e tipi di blocchi con dimensioni diverse sullo stesso layer, in modo da ottenere variazioni di scala diverse e sfruttare tutte le informazioni che possono essere generate da molteplici tipi di layer, invece che scegliere solamente un tipo di layer con una certa dimensione.

2 Dataset utilizzati

Il primo passo nello svolgimento del progetto di tesi, è stata la ricerca di dataset che raffigurano volti di vario genere, specialmente volti occlusi da mascherine. La costruzione del dataset è stata una fase delicata, dato che per avere un buon classificatore CNN, è importante partire da un dataset di qualità e ricco di immagini. A scopo del progetto, si è dovuto mettere insieme ed utilizzare molteplici dataset, dato che, in rete, non esiste un unico dataset che può soddisfare i requisiti prestabiliti.

Quasi la totalità dei dataset utilizzati sono reali, mentre MaskedFace-Net [14], è sintetico. Si è scelto di utilizzare un dataset sintetico dato che non esiste in rete, almeno fino a questo momento, un dataset reale che presenti volti mascherati in modo errato, con un volume non banale. Alcuni dataset descritti nel capitolo annotano questa classe, ma la quantità delle immagini con volti mascherati reali utilizzabili è molto limitata. Integrare i dataset con nuove immagini, in modo da avere una quantità di volti mascherati è possibile, ma è molto dispendioso in termini di tempo.

Il dataset sintetico è stato utilizzato prevalentemente nelle fasi di training e di validation, per le classi con volti occlusi, e FFHQ (dataset reale) per la classe con volti non occlusi da mascherine. I restanti dataset sono stati impiegati solamente per il test del modello.

L'utilizzo di un dataset sintetico comporta una maggiore difficoltà nel training del modello, dato che il dataset sintetico utilizzato è molto uniforme e non riesce a simulare in modo adeguato le condizioni reali.

2.1 MaskedFace-Net

MaskedFace-Net è un dataset contenente circa 137000 immagini di volti occlusi da mascherine chirurgiche creato partendo dal dataset FFHQ, che contiene volti semplici e non occlusi. Questo dataset è un dataset sintetico, creato utilizzando FFHQ, applicando delle mascherine chirurgiche sul viso partendo dai landmark facciali presenti nelle immagini. Tale dataset è molto omogeneo: la posa dei volti è sempre frontale e le mascherine chirurgiche sono tutte dello stesso colore.

MaskedFace-Net è suddiviso in due classi principali: CMFD (Correctly Masked Face Dataset, volti con mascherine indossate correttamente), contenente circa 67000 immagini e IMFD (Incorrectly Masked Face Dataset, volti con mascherine indossate erratamente), contenente circa 69000. IMFD viene suddiviso in altre tre sottoclassi: mento scoperto, naso scoperto e naso e bocca scoperto. Per la classificazione della classe "mascherine errate", abbiamo deciso di considerare solamente i volti mascherati con il naso scoperto, considerando i volti con la mascherina abbassata fino al mento, come negativi.

Data la sua numerosità, soprattutto sulla classe delle mascherine indossate erratamente, il dataset è stato utilizzato nelle fasi di training e validation della rete neurale.

2.2 MAFA

MAFA: MAsked FAces [15], è un dataset che è stato molto utile al fine del progetto di tesi. Il dataset è composto da circa 30000 immagini che raffigurano persone con volti occlusi da mascherine, sciarpe, mani etc. Le immagini sono state raccolte facendo web scraping, in particolare raccogliendole su piattaforme social e motori di ricerca come Flickr, Google e Bing, utilizzando parole chiave come "face, mask, occlusion e cover".

Ogni immagine di MAFA ha un descrittore con le seguenti annotazioni:

- posizione del volto: è presente una bounding box su ogni faccia presente nell'immagine; alcune immagini che sono troppo sfocate oppure con volti troppo piccoli e non riconoscibili facilmente, hanno un flag "Ignore"
- posizione degli occhi: per ogni faccia, sono presenti due punti che descrivono la posizione degli occhi
- posizione della maschera: bounding box che descrive la posizione della maschera oppure di altri oggetti che occludono il viso come ad esempio occhiali, sciarpe etc.
- orientazione del viso: vi sono cinque orientazioni: sinistra, destra, frontale, sinistra-frontale e destra-frontale

- livello di occlusione facciale: definisce tre livelli di occlusione facciale, in base al numero delle regioni facciali occluse: occlusione bassa (una o due regioni), occlusione media (tre regioni) ed occlusione massima (quattro regioni). Le regioni del volto interessate sono: occhi, naso, bocca e mento.

MAFA è stato utilizzato come dataset di test, per la classe contenente volti mascherati correttamente, sia per la classe dei volti mascherati in modo errato, dato che permette di identificare il livello di occlusione del viso. Infatti come ultima classe sono state utilizzate le immagini con un’occlusione bassa.

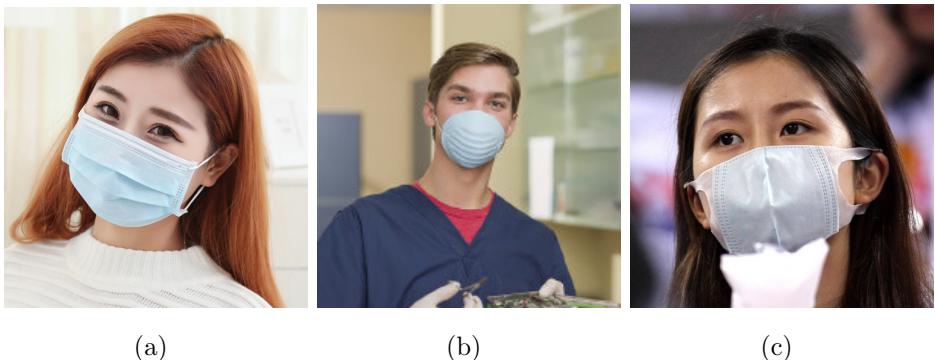


Figura 19: Immagini presenti nel dataset MAFA

2.3 Medical Masks Dataset

Medical Masks Dataset [16], pubblicato da Humans in the Loop, contiene 6000 immagini che raffigurano persone che indossano mascherine. Il dataset tiene traccia di 20 classi diverse, cioè accessori come occhiali, cappelli, hijab e la presenza della mascherina e nel caso fosse presente, se è stata indossata correttamente oppure no. Inoltre, il dataset è molto attento alle diversità etniche e di età.

Avendo sia volti non occlusi da mascherine, sia volti mascherati, con la possibilità di sapere se la mascherina è stata posizionata correttamente sul viso, il dataset è stato utilizzato, nella fase di test, in tutte e tre le classi.



Figura 20: Immagini presenti nel dataset Medical Masks Dataset

2.4 FFHQ: Flickr Faces High Quality

Un ulteriore dataset utilizzato, è FFHQ: Flickr Faces High Quality [17]. Il dataset contiene 70000 immagini ad alta qualità di volti senza nessun tipo di occlusione facciale, con una variazione incredibile di etnie, età, sfondo, accessori come cappelli, occhiali da sole etc.

Il dataset è stato utilizzato come classe negativa, invece di utilizzare il dataset "High Quality", abbiamo utilizzato solamente i thumbnails di dimensioni 128x128 scalandole a 160x160, in modo da alleggerire il carico computazionale sulle trasformazioni, dato che la rete richiede immagini di dimensioni 160x160.

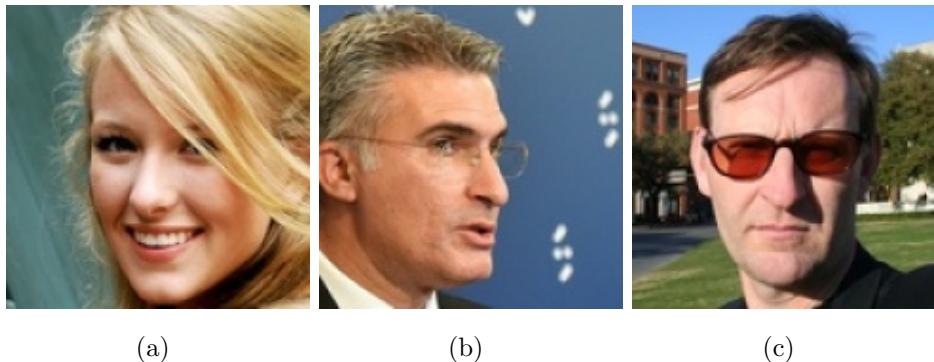


Figura 21: Immagini presenti nel dataset FFHQ

2.5 Creazione del dataset utilizzato nel progetto

Come specificato in precedenza, i dataset descritti sono stati utilizzati per la creazione del dataset su cui il modello ha eseguito la fase di training, di validazione e di test.

Il dataset principale, utilizzato nel progetto di tesi, contiene tutte e tre le classi (volti mascherati correttamente, volti mascherati in modo errato e volti senza occlusioni) ed è stato costruito con tutti i dataset descritti in precedenza. Ogni immagine è stata ridimensionata, con dimensioni 160x160, dato che il modello pre-trained è stato allenato su un dataset con la stessa dimensione. Il dataset di training e validation è composto utilizzando MaskedFace-Net e FFHQ, mentre per il dataset di test si è scelto di utilizzare MAFA, Medical Masks Dataset e FFHQ. Per estrarre i volti dalle immagini presenti in MAFA e Medical Masks Dataset, si è fatto affidamento ai loro descrittori.

Dataset	Corrette	Errate	Negativi
training	52980	45124	57738
validation	13246	11281	15033
test	779	395	748

Tabella 1: Numero di immagini presente nel dataset

Dalla tabella, si nota in modo particolare come il numero ridotto delle immagini di volti mascherati in modo errato sia molto piccolo. Il problema sta nel fatto che, le immagini annotate come volti mascherati in modo errato, non sono adatte per il training, dato che possono racchiudere occlusioni generiche e non solo da mascherina. Inoltre la maggior parte delle immagini presenta sfocature, rumori e angolazioni inaccettabili per le finalità del progetto.

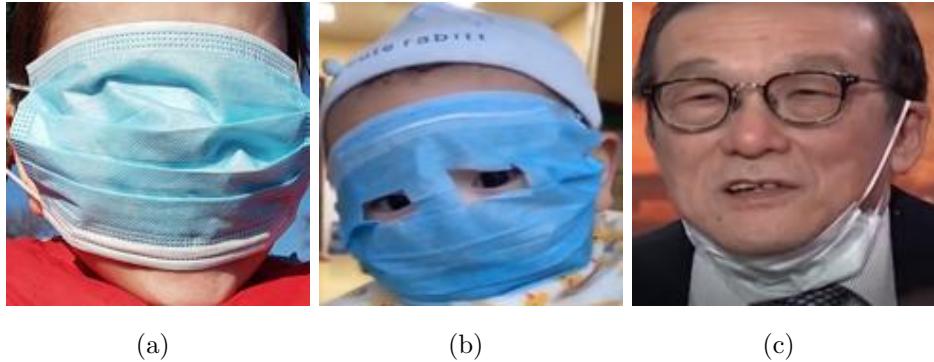


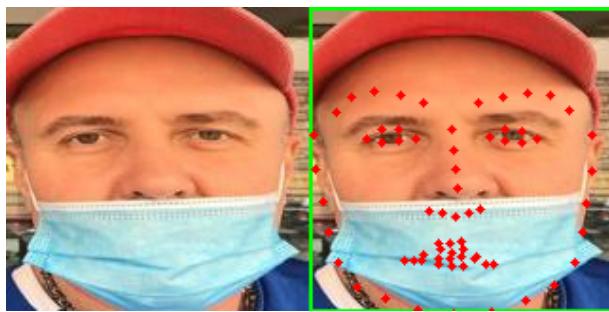
Figura 22: Immagini occluse in modo errato non utili allo scopo del progetto in quanto le prime due sono dei casi limite, mentre (c) da considerare come negativo

Per ovviare al problema, si è dovuto filtrare le immagini che non erano adatte al nostro scopo. Il numero di volti con mascherine errate reali nel dataset è circa 1100, mentre filtrandole a mano restano solamente 395.

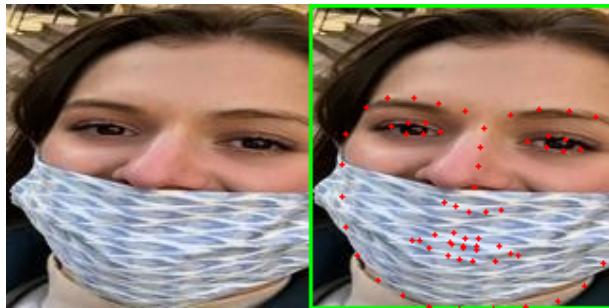
L’idea all’inizio consisteva nel filtrare i volti in modo automatico utilizzando un landmark detector ed uno skin detector, in modo da filtrare solamente le immagini che potevano avere un’angolazione eccessiva, mascherine non indossate (bocca non occlusa), molto sfocate, occlusioni non pertinenti etc.

Avendo landmark facciali, si può stimare la posizione del naso e dei due angoli della bocca, mentre lo skin detector valuta se il pixel appartiene al viso dell’individuo presente nell’immagine. Calcolando la percentuale dei pixel ”skin” nella region-of-interest, una bounding box che ha come base la distanza tra i due angoli della bocca e come altezza la distanza tra questa base e il naso, moltiplicati per un fattore di padding (nel caso dove tutti i landmark fossero presenti o rilevati dal detector).

I problemi della soluzione sono principalmente i seguenti: i due landmark detector utilizzati, MTCNN [18] e dlib [19] hanno problemi con il riconoscimento del volto e dei landmark sulla maggior parte delle immagini: MTCNN non riesce a stimare i landmark anche se l’immagine è chiara e con pochi rumori, invece dlib ha un severo problema nella stima dei landmark facciali con i volti inclinati, anche se di poco.



(a)



(b)

Figura 23: Esempio di stima landmark facciali di dlib con volti inclinati e non, utilizzando dlib. Si può vedere come in (b) dlib abbia problemi a stimare in modo accurato i landmark della bocca e del naso.

Il risultato dato dalla soluzione di filtering del dataset non è utilizzabile ai fini del progetto, dato che le immagini filtrate sono circa 80, utilizzando MTCNN, contro le 395 filtrate a mano.

3 Classificazione di volti occlusi da mascherine

3.1 InceptionResNetV1 finetuning

Come definito in precedenza, il modello deep utilizzato per il progetto di tesi è Inception-ResNetV1, il quale implementa sia i moduli inception sia le skip connection presenti nelle reti residuali (ResNet). L’implementazione di Inception-ResNetV1 utilizzata è presente nel package pip facenet-pytorch [20], sviluppata con il framework PyTorch [21]. La rete è stata già addestrata in precedenza per eseguire Face Recognition su determinati volti. Il dataset di addestramento della rete pre-trained è VGGFace2 [22], il quale contiene numerosi volti, con una pluralità di espressioni facciali, pose, età, etnie ed accessori.

Per utilizzare un modello già allenato e allenarlo nuovamente su un dataset diverso, bisogna utilizzare una tecnica di transfer learning chiamata finetuning. Il finetuning di un modello pre-addestrato viene realizzato cambiando l’ultimo layer (fully-connected) della rete, cioè il layer di classificazione, con un altro layer fully-connected che ha un numero di neuroni di output pari al numero di classi di interesse (tre nel nostro caso) ed eseguire il training sul modello modificato utilizzando un altro dataset ed abbassando via via il learning rate. In alternativa è possibile anche ”congelare” alcuni layer convolutivi, in modo da bloccare l’addestramento su di essi, soprattutto nei layer iniziali. Ciò permette di utilizzare le feature più a basso livello della rete già addestrata, cosicché l’addestramento finetuning si possa concentrare sulle feature più ad alto livello. Questa ultima tecnica non è stata utilizzata dato che tutti i layer del modello sono stati allenati sul nuovo dataset.

I vantaggi nel fare finetuning sono i seguenti:

- **riduzione del tempo di addestramento:** utilizzando un modello già addestrato, i pesi dei primi livelli (feature di basso livello) sono già efficaci per problemi simili
- **performance migliorata:** un modello addestrato su un dataset molto grande (VGGFace2 ha 3.31 milioni di volti), permette di avere dei layer convolutivi più bassi più efficaci rispetto ad un modello che è stato allenato sul dataset utile al problema da risolvere, molto più piccolo

- **maggior generalizzazione:** permette di avere una proprietà di generalizzazione maggiore e di contrastare l’overfitting a causa di un training eseguito su dataset più piccoli

Ci sono anche degli svantaggi non banali, nel fare finetuning. Non c’è nessuna garanzia che un modello pre-trained possa raggiungere una accuracy migliore rispetto ad un modello allenato ex novo. Per di più, utilizzando un modello pre-trained, ci si limita alla sua architettura, limitando anche le dimensioni delle immagini processate. Difatti, le immagini utilizzate per il training della rete hanno una dimensione di 160x160, stessa dimensione utilizzata nel pre-training del modello.

3.2 Data augmentation

Uno degli ostacoli più difficili da superare, è l’utilizzo del dataset sintetico MaskedFaceNet, il quale non rappresenta casi reali di volti occlusi da mascherina. Il dataset è uniforme, le mascherine sono tutte di tipo chirurgico e dello stesso colore e il dataset non presenta nessun tipo di rumore, sfocatura o disturbo. Ciò compromette la capacità del modello di generalizzare sui volti presi da situazioni naturali e inevitabilmente rumorose.

Dato che il problema è l’uniformità e la non presenza di rumore nel dataset, si è deciso di utilizzare delle tecniche di trasformazione delle immagini in modo da rendere il dataset più rumoroso e meno uniforme cosicché il modello possa generalizzare in modo migliore, specialmente con immagini sfocate o con molto rumore.

Le tecniche di data augmentation, permettono artificialmente di aumentare il numero delle istanze del dataset oppure modificarle, in modo da ottenere un dataset più generalizzato e spingere il modello a discriminare le immagini anche in presenza di variazioni di posizione, scala, orientamento, contrasto etc.

Per fare data augmentation, sono state utilizzate molteplici trasformazioni applicabili alle immagini. Queste trasformazioni sono presenti nel package *torchvision* di PyTorch. Nella fase di training, quando il dataset viene caricato sfruttando il *DataLoader* di PyTorch, il dataset viene trasformato secondo un oggetto che viene poi passato nel DataLoader, il quale contiene tutte le trasformazioni da applicare

su ogni immagine. Le trasformazioni vengono eseguite ogni volta che si accede ad un’immagine presente nel DataLoader e possono essere applicate su tutte le immagini, oppure assegnare una probabilità di trasformazione. Le trasformazioni applicate sono:

- **Gaussian Blur**: semplice sfocatura gaussiana. Il kernel ha una dimensione di 5 con $2 < \sigma < 8$ e $p = 0.1$
- **Color jitter**: modifica la luminosità, contrasto e tonalità del colore dell’immagine. I parametri utilizzati sono: $(0.1, 0.15, 0.15, 0.15)$ con $p = 0.4$.
- **Random horizontal flip**: applica l’effetto specchio all’immagine con $p = 0.5$
- **Random affine**: applica una trasformazione affine, cioè una trasformazione che mantiene i parallelismi. Il centro dell’immagine viene mantenuto e non vengono applicate trasformazioni di shearing o di traslazione. La trasformazione è stata utilizzata solamente per la rotazione con intervallo dei gradi sessagesimali delle rotazioni a $[0, 10]$.
- **Grayscale**: converte un’immagine a colori (ad esempio canali RGB) in un’immagine a livelli di grigio. È possibile scegliere se l’output della trasformazione deve avere solamente un canale (livello di grigio) o utilizzare tutti e tre i canali RGB per rappresentare il livello di grigio. Nell’unico esperimento dove è stata utilizzata questa trasformazione si è dovuto utilizzare tutti i tre i canali dato che la rete accetta in input solo immagini con tre canali.



Figura 24: Esempio di immagini processate con le trasformazioni descritte

4 Risultati ottenuti

In questo capitolo verranno descritti i risultati ottenuti sul modello scelto, utilizzando il dataset creato e descritto nel precedente capitolo. Il training del modello è stato fatto in più fasi, utilizzando iperparametri e trasformazioni diverse, con un subset del dataset, in modo da trovare empiricamente buoni valori per gli iperparametri che e garantire cos' alla rete la migliore accuratezza possibile. Per testare il modello, si è utilizzato il test set del dataset, il quale viene dato in input alla rete in modo poi da confrontare le istanze classificate dalla rete con i valori reali, utilizzando delle metriche che verranno descritte in seguito.

Da ricordare che nella fase di training, è stato utilizzato il dataset sintetico per le classi con volti occlusi da mascherine (MaskedFace-Net), mentre per il test si sono utilizzate solamente immagini reali "in the wild" e non generate sinteticamente. Lo scopo della tesi è proprio la creazione di un modello capace di classificare volti reali partendo da un dataset creato sinteticamente.

4.1 Metriche utilizzate

Accuratezza L'accuratezza è la probabilità di classificare correttamente un'istanza usando un dato classificatore. Detto valore viene calcolato come:

$$acc = \frac{\text{Numero delle predizioni corrette}}{\text{Numero totale delle predizioni}}$$

Curva ROC La curva ROC (Recieve operating characteristic) è un grafico che permette di valutare la bontà del classificatore binario data una certa soglia. Sulle ascisse si trova il True Positive Rate, cioè la frazione dei veri positivi (sensibilità) e sulle ordinate abbiamo il False Positive Rate, cioè la frazione dei falsi positivi ($1 - specificità$). La curva del grafico ROC è utile per vedere quanto il classificatore si scosta da un classificatore casuale oppure un classificatore ottimale e soprattutto per confrontare i classificatori binari tra loro, con l'AUC (Area Under Curve). Più l'AUC si avvicina a 1, più il classificatore tende ad avvicinarsi al classificatore ottimale.

Matrice di confusione La matrice di confusione è una tabella che permette di descrivere l'accuratezza del classificatore visualizzando il tasso di classificazione e confusione tra le diverse classi. Ogni riga della matrice rappresenta le istanze delle classi reali, mentre ogni colonna descrive le istanze delle classi predette, permettendo di visualizzare i veri positivi, i falsi positivi, i veri negativi ed i falsi negativi.

		Predetti	
Reali		Positivi	Negativi
	Positivi	a	b
	Negativi	c	d

Tabella 2: Esempio di matrice di confusione; a rappresenta i veri positivi, b i falsi negativi, c i falsi positivi e d i veri negativi

4.2 Training e test eseguiti

Sono state condotte numerose sessioni di addestramento del modello utilizzato (Inception-ResNetV1) per la scelta degli iperparametri e per ogni sessione, sono stati salvati e testati tutti i checkpoint di ogni esperimento.

Tutti gli addestramenti fatti sulla rete utilizzano lo stesso algoritmo di ottimizzazione, Adam [23], il quale è l'algoritmo utilizzato di default per reti CNN. Questo algoritmo differisce rispetto al semplice Stochastic Gradient Descent, dato che implementa il concetto di momentum optimization e unisce i benefici di AdaGrad [24] (migliori prestazioni su gradienti sparsi) e di RMSprop [25]. Questo algoritmo introduce due nuovi parametri oltre al learning rate: $\beta_1, \beta_2 \in [0, 1]$ uguali rispettivamente a 0.9 e 0.999, i valori di default. Il learning rate che offre l'accuratezza migliore è $1e^{-7}$. Il motivo di un learning rate così basso è dato dalla velocità del modello pre-trained a convergere; il valore è stato trovato empiricamente facendo più prove con learning rate diversi.

Gli esperimenti sono stati eseguiti su una macchina Ubuntu 18.04, con CPU LTS Intel Core i7-4770K e GPU NVIDIA Quadro P6000, versione di CUDA 9.2, versione di Python 3.7.6 e versione di PyTorch 1.7.1+cu92.

E' stato utilizzato *MultiStepLR* di PyTorch per diminuire il learning rate ad ogni epoca. MultiStepLR fa decadere il learning rate dato un fattore $\gamma = 0.1$ ($lr = lr * \gamma$) dopo la quinta epoca e dopo la decima. L'utilizzo del decadimento del learning rate è dato dalla convergenza veloce che si ha nell'addestramento della rete. Dopo 5 epocha la rete riesce ad avere un'accuratezza abbastanza buona, ma senza diminuire il learning rate, l'accuratezza del modello non migliora.

Il primo esperimento eseguito consiste nell'addestrare il modello solamente utilizzando le classi positive (mascherina corretta) e negative (senza mascherina). L'addestramento è stato eseguito per 25 epocha, con un batch size di 64. L'accuratezza complessiva è 0.93 e la durata totale dell'addestramento è stata di circa 43 minuti.

		Predetti	
		Positivi	Negativi
Reali	Positivi	676	103
	Negativi	1	747

Tabella 3: Matrice di confusione del test con volti mascherati correttamente e senza mascherine

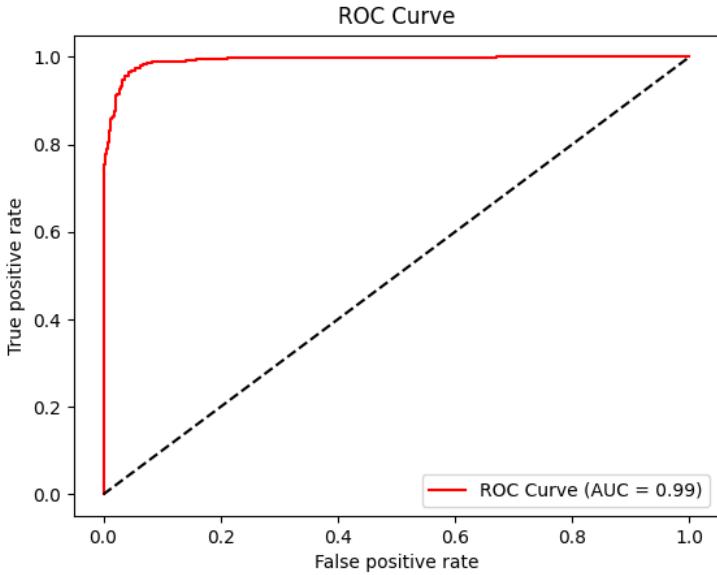


Figura 25: Curva ROC

Le curve dell'apprendimento del modello sono state ricavate utilizzando *Tensorboard* [26], un'applicazione Web che permette di visualizzare grafici e dati riguardanti esperimenti di Machine Learning e Deep Learning. Permette di avere una visualizzazione real time delle performance, in modo da capire come il modello sta eseguendo la fase di training e di validation. Il tool è stato sviluppato per TensorFlow ma può essere utilizzato facilmente anche con altri framework, ad esempio PyTorch. I grafici generati da Tensorboard hanno sulle ordinate il valore della funzione tracciata (accuratezza, costo), mentre sulle ascisse troviamo il tempo, in ore decimali. Il tracciamento delle funzioni sono state eseguite utilizzando uno smoothening adoperando la formula della media mobile esponenziale, con coefficiente $\alpha = 0.4$.

Per il grafico dell'accuratezza del modello, in grigio si ha l'accuratezza della fase di training, mentre la curva azzurra riferisce alla fase di validation; invece con il colore fucsia si ha l'accuratezza della fase di training, mentre la curva blu riferisce alla fase di validation. I grafici opachi rappresentano le versioni senza l'applicazione della media mobile.

Le due curve dell'apprendimento fanno notare come il modello riesce a convergere molto velocemente. Difatti, dopo solamente cinque epocha, si nota come l'accuratezza della validazione del modello raggiunge il valore di 0.94, mentre la funzione di costo della validazione è 0.25.

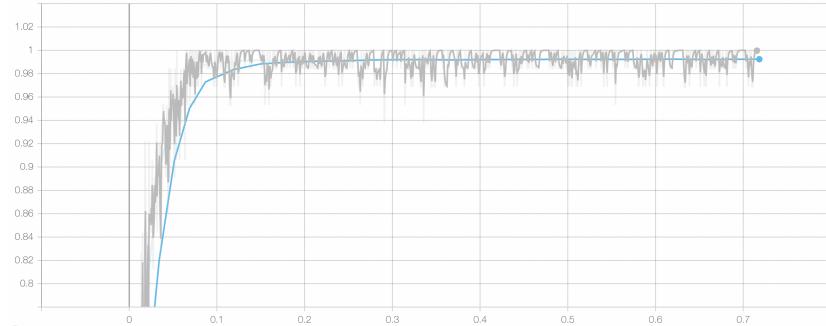


Figura 26: Grafico della accuratezza del primo esperimento

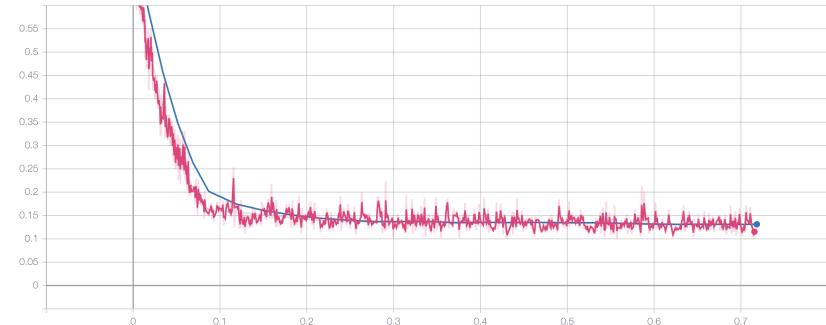


Figura 27: Grafico della funzione di costo del primo esperimento

Il risultato è molto buono, il classificatore riesce a distinguere in modo pertinente i volti occlusi da mascherine e volti senza occlusione. La matrice di confusione ci fa notare che la rete sbaglia soprattutto sui positivi. Questo problema verrà risolto in seguito applicando le trasformazioni descritte in precedenza.

Il passo successivo è stato introdurre la terza classe del dataset, cioè le mascherine errate. In totale, le classi diventano tre: volti con mascherine indossate correttamente, volti con mascherine indossate in modo errato e volti non occlusi da

mascherine. Il processo di addestramento è analogo al precedente, l'unico cambiamento è l'utilizzo dell'intero dataset, sia per la fase di test che per la fase di training e validazione del modello. L'addestramento è stato eseguito sempre per 25 epoch, con un batch size di 128 ed un learning rate uguale a $1e^{-7}$. L'accuratezza complessiva è di 0.91, mentre l'accuratezza sulla nuova classe è 0.68, un valore decisamente basso e la durata complessiva del training del modello è stata di circa *3h30m*.

Il principale problema riscontrato con la terza classe è che le immagini vengono spesso classificate come mascherine corrette. La rete ha difficoltà a discriminare tra le due classi, come si può notare dalla matrice di confusione, dato che alla fine la differenza visiva tra le due classi non è così grande come le immagini positive e negative (presenza del naso scoperto, posizione storta della mascherina etc.).

		Predetti		
		Corrette	Errate	Negativi
Reali	Corrette	740	35	4
	Errate	95	272	28
	Negativi	0	2	746

Tabella 4: Matrice di confusione del test con tutte e tre le classi

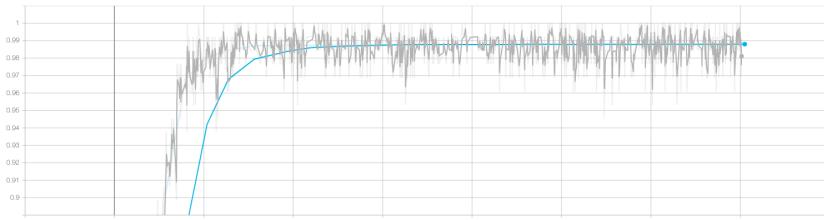


Figura 28: Grafico della accuratezza del secondo esperimento, utilizzando il dataset completo (tre classi)

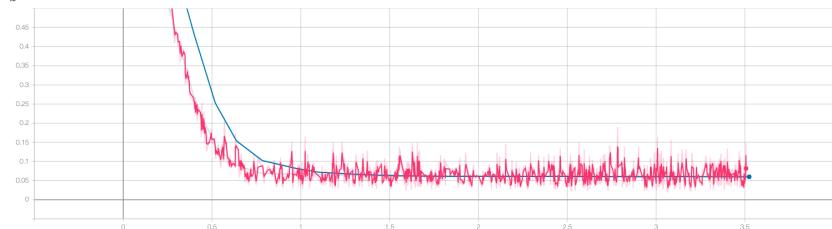


Figura 29: Grafico della funzione di costo del secondo esperimento, utilizzando il dataset completo (tre classi)

Le curve di apprendimento sono molto simili all'esperimento descritto in precedenza. Il modello riesce dopo cinque epocha ad avere un'accuratezza della validazione di 0.94, con una validation loss uguale a 0.09.

Per rimediare un'accuratezza così bassa sulla nuova classe , pari a 0.68 si introducono le trasformazioni affini di cui abbiamo discusso in precedenza. Si può notare dai grafici come l'accuratezza di tutto il modello, ma soprattutto l'accuratezza sulla nuova classe introdotta aumenti significativamente.

Questo probabilmente è dato dalla capacità della rete di apprendere casi limite e soprattutto estendere la sua capacità di generalizzazione, soprattutto nei casi reali (test set), nei quali si hanno diverse illuminazioni, colori delle mascherine diverse date dal cambio di tonalità e il riconoscimento di immagini sfocate grazie all'applicazione del kernel gaussiano. Ricordiamo che il dataset di training e validation è sintetico, molto omogeneo, poco rumoroso e con lo stesso tipo e tonalità di mascherine.

Infatti, allenando il modello utilizzando le trasformazioni descritte, le prestazioni di classificazione aumentano notevolmente, soprattutto sulla seconda classe. Utilizzando le trasformazioni sulle immagini del dataset di training, si è ottenuto il modello che riesce ad avere la maggiore accuratezza di classificazione. L'addestramento è stato eseguito su 128 epocha, con un batch size di 128. L'accuratezza totale del modello è di 0.92, mentre l'accuratezza della seconda classe è 0.87. Il miglioramento dell'accuratezza è netto con l'utilizzo delle trasformazioni; l'accuratezza della seconda classe è maggiore del 29% rispetto al modello allenato senza le trasformazioni applicate. Come si può vedere dalla matrice di confusione, che gli

errori commessi sono stati prevalentemente sulla distinzione tra le due classi con i volti occlusi da mascherine.

		Predetti		
		Corrette	Errate	Negativi
Reali	Corrette	748	28	3
	Errate	42	345	8
	Negativi	0	1	747

Tabella 5: Matrice di confusione del test con le tre classi, tutte le trasformazioni descritte in precedenza (no grayscale). Questo modello ha la maggiore accuratezza tra tutti i test effettuati su Inception-ResNetV1.

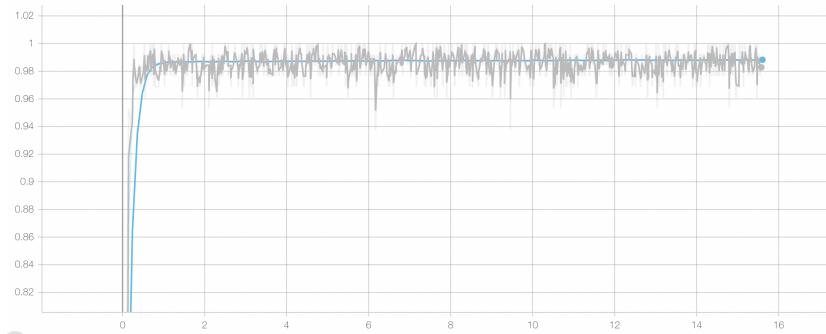


Figura 30: Grafico dell'accuracy dell'addestramento utilizzando il dataset completo (tre classi) e le trasformazioni descritte

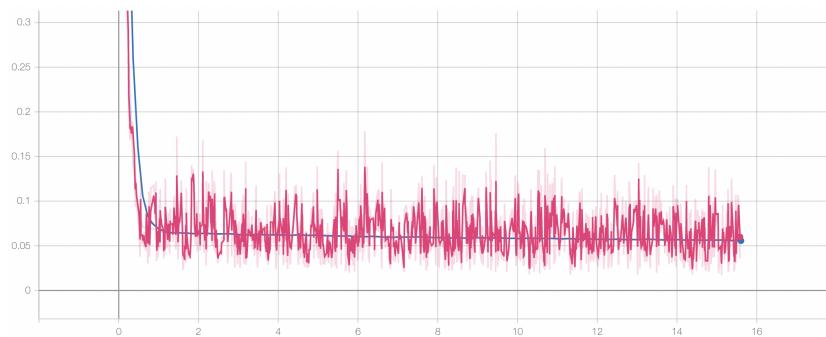


Figura 31: Grafico della funzione di costo dell'addestramento utilizzando il dataset completo (tre classi) e le trasformazioni descritte

Nella stessa sessione di addestramento, l'epoca numero 101 ha un'accuratezza maggiore sulla seconda classe rispetto al checkpoint illustrato precedentemente (numero 93), eppure l'accuratezza sulla prima classe decresce sensibilmente, classificando erroneamente i volti della prima classe. L'accuratezza complessiva di questo checkpoint è uguale a 0.90.

		Predetti		
		Corrette	Errate	Negativi
Reali	Corrette	718	53	8
	Errate	29	356	10
	Negativi	0	1	747

Tabella 6: Matrice di confusione del test con le tre classi, tutte le trasformazioni descritte in precedenza (no grayscale), checkpoint con l'accuratezza maggiore sulla seconda classe, utilizzando un batch size uguale a 128

Il risultato maggiore riguardante l'accuratezza della seconda classe è stato ottenuto facendo un'ulteriore esperimento, sempre sull'intero dataset e con gli stessi parametri dell'esperimento precedente, ma modificando il batch size da 128 a 256. Il checkpoint con l'accuratezza globale maggiore ha come accuratezza 0.90. Considerando il checkpoint con l'accuratezza maggiore sulla seconda classe, il valore è di 0.92, la migliore di tutti i test effettuati.

Con la macchina utilizzata per il training del modello non è possibile utilizzare batch size più grandi per questo esperimento, in modo da capire se davvero il batch size (e i layer di batch-normalization) influisce sull'accuratezza della seconda classe, a causa delle limitazioni di memoria GPU (casi di out of memory).

		Predetti		
		Corrette	Errate	Negativi
Reali	Corrette	718	53	8
	Errate	29	356	10
	Negativi	0	1	747

Tabella 7: Matrice di confusione del test con le tre classi, tutte le trasformazioni descritte in precedenza (no grayscale) con batch size 256. Questo modello ha la maggiore accuratezza sulla classe "errate" tra tutti i test effettuati su Inception-ResNetV1.

Dato che le trasformazioni applicate sono state così efficaci per avere un modello con un'accuratezza maggiore, sugli esperimenti utilizzando l'intero dataset con tutte e tre le classi, sono state utilizzate le stesse trasformazioni in modo da vedere il miglioramento sul dataset ridotto utilizzato nel primo esperimento con le due classi (volti che indossano mascherine in modo corretto e volti senza occlusioni). Gli iperparametri utilizzati sono gli stessi del primo esperimento, l'unica differenza è nel batch size, il quale è stato aumentato a 128. L'accuratezza complessiva del modello è 0.98 rispetto al primo esperimento dove l'accuratezza è 0.93, un miglioramento netto grazie all'utilizzo delle trasformazioni.

		Predetti	
		Positivi	Negativi
Reali	Positivi	756	23
	Negativi	2	746

Tabella 8: Matrice di confusione del test con volti mascherati correttamente e senza mascherine, applicando le trasformazioni descritte

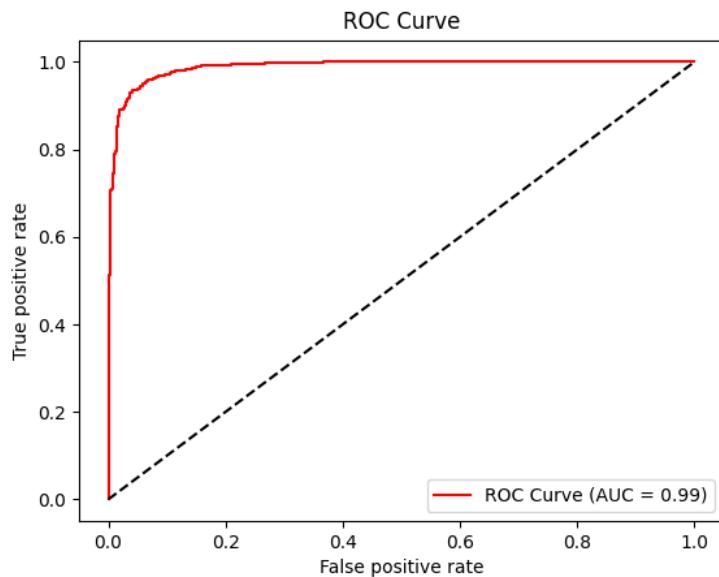


Figura 32: Curva ROC

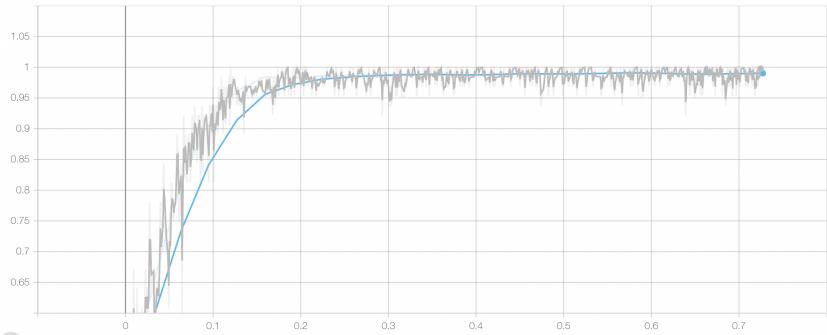


Figura 33: Grafico dell'accuratezza dell'addestramento con due classi e applicazione delle trasformazioni descritte)

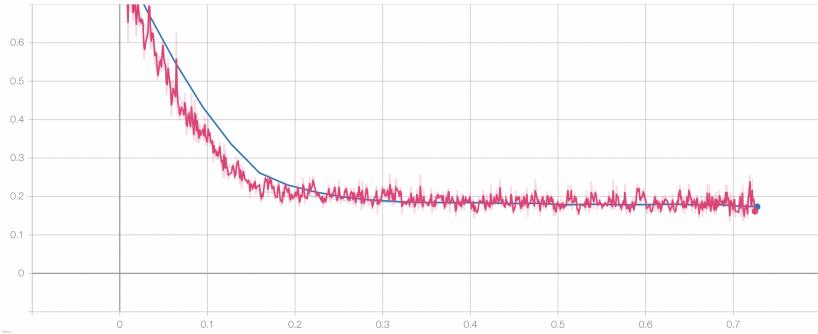


Figura 34: Grafico della funzione di costo dell'addestramento con due classi e applicazione delle trasformazioni descritte

Si è provato ad utilizzare anche la trasformazione *grayscale*, la quale converte l'immagine RGB in scala di grigi, utilizzando sempre tre canali dato che la rete necessita un'immagine che abbia tre canali colore (dato che è stata pre addestrata su un dataset RGB).

L'addestramento è stato sempre eseguito con gli stessi parametri del precedente esperimento. L'accuratezza dell'intero modello è di 0.88, mentre l'accuratezza della seconda classe è uguale a 0.55. Il modello non riesce ad avere una migliore prestazione utilizzando immagini in scale di grigio, questo è dato dal pre training della rete con VGGFace2, un dataset a colori (RGB). Ulteriormente, anche in questo caso la classe penalizzata è la seconda. Le curve di apprendimento del modello sono simili agli esperimenti descritti in precedenza.

		Predetti		
		Corrette	Errate	Negativi
Reali	Corrette	740	35	4
	Errate	95	272	28
	Negativi	0	2	746

Tabella 9: Matrice di confusione del test con le tre classi, utilizzando le trasformazioni descritte e la trasformazione grayscale

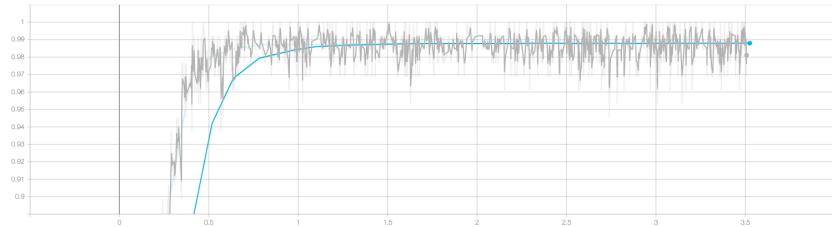


Figura 35: Grafico della accuratezza dell'esperimento con la trasformazione grayscale

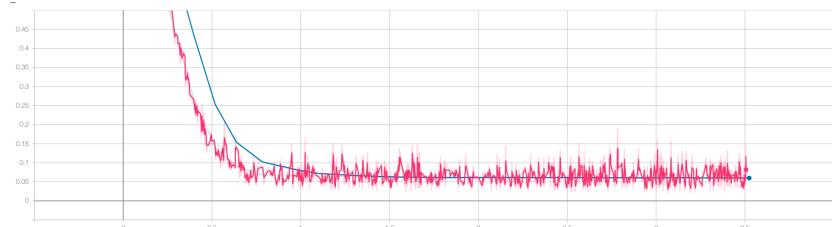


Figura 36: Grafico della funzione di costo dell'esperimento con la trasformazione grayscale

Sperimentando con le trasformazioni utilizzate, si è notato empiricamente che alcune trasformazioni hanno avuto un peso maggiore nell'aumentare l'accuracy del modello, soprattutto per la classificazione dei volti occlusi da mascherine indossate in modo errato. La seguente lista descrive in modo breve i test eseguiti, eliminando ad ogni addestramento una delle trasformazioni descritte. Tutti gli esperimenti sono stati eseguiti con un numero di epoch pari a 25 e batch size uguale a 128:

- **No applicazione kernel gaussiano:** senza l'applicazione della sfocatura gaussiana, il modello ottiene un'accuratezza globale di 0.92 e di 0.78 sulla seconda classe.
- **No trasformazioni affini e capovolgimenti orizzontali:** in questo caso, il modello ottiene 0.93 come accuratezza complessiva, mentre per la seconda classe otteniamo 0.78.
- **No color jitter:** senza il cambiamento di colore, tonalità e contrasto dell'immagine, il modello riesce ad ottenere 0.89 come accuratezza complessiva, mentre l'accuratezza della seconda classe cala notevolmente a 0.53.

Da questi esperimenti possiamo notare come l'ultima trasformazione abbia un ruolo chiave per ottenere una buona classificazione dei volti con mascherine indossate in modo errato. Il motivo, come spiegato in precedenza, è da imputare alla troppa omogeneità del dataset sintetico.

Conclusioni

Lo scopo dell’elaborato di tesi consiste nella creazione di un classificatore, utilizzando tecniche di Deep Learning, capace di distinguere se un volto di un dato soggetto è occluso da mascherine e specialmente, se la mascherina è stata indossata in modo appropriato.

Il modello scelto, la composizione del dataset e gli esperimenti eseguiti sono stati soddisfacenti dal punto di vista del risultato, ottenendo un’accuratezza ed una performance del modello elevata, sia per il problema di classificazione binaria, sia per la classificazione con tre classi.

Il principale ostacolo riscontrato è riconducibile all’uso di dataset sintetici e non reale, data la carenza di quest’ultimi, specie per mascherine indossate in modo errato. L’utilizzo delle trasformazioni descritte ha permesso tuttavia al modello di ottenere un’accuratezza buona nella classificazione dei volti occlusi da mascherine.

Sviluppi futuri Ci possono essere molteplici sviluppi futuri su questa architettura, ad esempio potrebbe essere d’aiuto utilizzare questo modello come backbone di un modello SSD (Single Shot Detector), in modo da creare un’architettura non solo per la classificazione, ma anche per fare riconoscimento di volti occlusi da mascherine.

Oltre a ciò si potrebbe aumentare le capacità del modello nella classificazione di casi limite, ad esempio volti messi di profilo, poca luce, sfocature e rumori maggiori, dato che la soluzione presentata è efficace su volti centrati e non molto ruotati, con una media tolleranza ai rumori. Sicuramente, un modo per ottenere un risultato migliore in questi casi, è l’utilizzo di un dataset non sintetico, ma reale ”in the wild”, con numerose immagini.

Un altro sviluppo futuro interessante potrebbe essere quello di esplorare altri tipi di modelli di classificazione basati su reti deep oppure sfruttare le feature prodotte dagli ultimi layer del modello per altri tipi di classificatori come ad esempio quelli bayesiani o incentrati sulla rappresentazione sparsa e learning di dizionari per detta rappresentazione.

Bibliografia e sitografia

Riferimenti bibliografici

- [1] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG].
- [2] N. Dalal e B. Triggs. «Histograms of oriented gradients for human detection». In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.
- [3] David G. Lowe. «Distinctive Image Features from Scale-Invariant Keypoints». In: *International Journal of Computer Vision* 60.2 (nov. 2004), pp. 91–110. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: <https://doi.org/10.1023/B:VISI.0000029664.99615.94>.
- [4] Herbert Bay, Tinne Tuytelaars e Luc Van Gool. «SURF: Speeded Up Robust Features». In: *Computer Vision – ECCV 2006*. A cura di Aleš Leonardis, Horst Bischof e Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.
- [5] D. H. Hubel e T. N. Wiesel. «Receptive fields of single neurones in the cat's striate cortex». eng. In: *The Journal of physiology* 148.3 (ott. 1959). PMC1363130[pmcid], pp. 574–591. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1959.sp006308. URL: <https://doi.org/10.1113/jphysiol.1959.sp006308>.
- [6] K. Fukushima. «Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position». eng. In: *Biological cybernetics* 36.4 (1980). 7370364[pmid], pp. 193–202. ISSN: 0340-1200. DOI: 10.1007/BF00344251. URL: <https://doi.org/10.1007/BF00344251>.
- [7] Y. LeCun et al. «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

- [8] Yann LeCun e Corinna Cortes. «MNIST handwritten digit database». In: (2010). URL: <http://yann.lecun.com/exdb/mnist/>.
- [9] Andrew Ng. *C4W1L09 Pooling Layers*. Youtube. 2017. URL: <https://www.youtube.com/watch?v=8o0gPU0-TBY>.
- [10] Christian Szegedy et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. eprint: [arXiv:1602.07261](https://arxiv.org/abs/1602.07261).
- [11] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: [1512.03385](https://arxiv.org/abs/1512.03385).
- [12] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: [1409.4842](https://arxiv.org/abs/1409.4842).
- [13] Kaiming He e Jian Sun. *Convolutional Neural Networks at Constrained Time Cost*. 2014. arXiv: [1412.1710](https://arxiv.org/abs/1412.1710).
- [14] Adnane Cabani et al. «MaskedFace-Net – A Dataset of Correctly/Incorrectly Masked Face Images in the Context of COVID-19». In: *Smart Health* (2020). ISSN: 2352-6483. DOI: <https://doi.org/10.1016/j.smhl.2020.100144>. URL: <http://www.sciencedirect.com/science/article/pii/S2352648320300362>.
- [15] Shiming Ge et al. «Detecting Masked Faces in the Wild with LLE-CNNs». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2682–2690.
- [16] Humans in the Loop. *Medical mask dataset*. 2020.
- [17] Tero Karras, Samuli Laine e Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2018. arXiv: [1812.04948](https://arxiv.org/abs/1812.04948).
- [18] Kaipeng Zhang et al. «Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks». In: (2016). DOI: [10.1109/LSP.2016.2603342](https://doi.org/10.1109/LSP.2016.2603342). arXiv: [1604.02878](https://arxiv.org/abs/1604.02878).
- [19] Davis E. King. «Dlib-ml: A Machine Learning Toolkit». In: *Journal of Machine Learning Research* 10 (2009), pp. 1755–1758.
- [20] Tim Esler. *Face Recognition Using PyTorch*. <https://github.com/timesler/facenet-pytorch>. 2019.

- [21] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». In: *Advances in Neural Information Processing Systems 32*. A cura di H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [22] Qiong Cao et al. *VGGFace2: A dataset for recognising faces across pose and age*. 2017. arXiv: 1710.08092.
- [23] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980.
- [24] John Duchi, Elad Hazan e Yoram Singer. «Adaptive Subgradient Methods for Online Learning and Stochastic Optimization». In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: <http://jmlr.org/papers/v12/duchi11a.html>.
- [25] T. Tieleman e G. Hinton. *Lecture 6.5—RMSprop: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.
- [26] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <http://tensorflow.org/>.

Ringraziamenti

A conclusione di questo percorso, desidero ringraziare tutte le persone senza le quali questo lavoro di tesi non esisterebbe nemmeno.

Un ringraziamento particolare va al professore Giuliano Grossi, mio relatore, e a Sathya Bursic che in questi mesi di lavoro mi hanno guidato con suggerimenti pratici e fondamentali per la ricerca e per la stesura di questo elaborato.

Ringrazio infinitamente i miei genitori per avermi permesso di arrivare fino a questo traguardo e per avermi sostenuto sia moralmente che economicamente durante tutto questo percorso.

Infine un grazie speciale a Rita, la persona che è stata sempre al mio fianco, che mi ha sostenuto e incoraggiato durante questi mesi di redazione della tesi. Grazie per tutto l'affetto e la forza che mi dai ogni giorno.