ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

RILEVAMENTO DI MASCHERINE FACCIALI SANITARIE: SPERIMENTAZIONE E PORTING DI UNA SOLUZIONE ALLO STATO DELL'ARTE SU ANDROID CON TENSORFLOW LITE

Elaborato in
Programmazione di applicazioni data intensive

Relatore
Prof. Gianluca Moro

Presentata da
Mattia Achilli

Seconda Sessione di Laurea
Anno Accademico 2019 – 2020

PAROLE CHIAVE

Machine Learning

Deep Learning

Deep Neural Networks

Convolutional Neural Network

Python

A chiunque mi abbia sostenuto,
e mi abbia aiutato a raggiungere questo traguardo.

Introduzione

L'intelligenza artificiale (AI) è recentemente diventata uno dei principali trend tecnologici strategici grazie ad avanzamenti importanti nella ricerca scientifica del settore e nell'enorme sviluppo delle capacità di calcolo dei sistemi hardware.

E' una disciplina vastissima che risale all'epoca del Prof. Alan Turing, il quale ideò nel 1950 il famoso test, che prese poi il suo nome, per stabilire se una macchina è intelligente.

Negli ultimi dieci anni la ricerca scientifica in ambito AI ha fatto progressi che gli esperti ritengono superiori alle conoscenze accumulate in precedenza. Questi avanzamenti si sono tradotti in tecnologie software di pubblico dominio che hanno portato allo sviluppo di applicazioni eclatanti in quasi ogni ambito delle discipline scientifiche.

Una delle differenze più importanti tra i metodi di AI classici e quelli recenti è che questi ultimi sono fortemente basati sull'estrazione di conoscenza da masse di dati e maggiore è la disponibilità di dati, più la conoscenza estratta è accurata.

Perciò la svolta è avvenuta grazie anche alla disponibilità di masse di dati, la cui generazione continua con ritmi sempre maggiori e impensabili solo fino a qualche anno fa. Infatti, assistiamo all'introduzione in sempre più settori e ambiti, di dispositivi elettronici e sensori di ogni genere. Ciò ha reso necessaria la ricerca di nuovi metodi per la gestione e l'elaborazione dei dati, le cui dimensioni hanno largamente superato le capacità umane di trattarli e analizzarli.

In particolare nell'ultimo periodo, ha iniziato a guadagnare molta popolarità una macro area di tecniche nota con il nome di **machine learning**.

I recenti successi del machine learning sono trasversali a numerosi settori tra cui *speech recognition* e *computer vision*, tanto per citarne due tra i più famosi.

Uno dei punti di forza del machine learning è che le grandi quantità di dati da gestire non sono un problema, tutta la potenza si basa su grandi insiemi di informazioni. Anche per questo motivo si sta diffondendo molto velocemente in tantissimi campi grazie alla disponibilità di grandi masse di dati

L'utilizzo del machine learning può però portare a risultati non soddisfacenti quando si ha ad esempio la necessità di gestire immagini o testi poiché l'attività di preparazione dei dati e di selezione delle variabili decisive per l'estrapolazione del modello di conoscenza richiede l'intervento di esperti ed è molto dispendiosa. In questi casi, infatti, si ottengono risultati generalmente più accurati e precisi attraverso tecniche diverse, appartenenti alla macro area di ricerca del **deep learning** che ha tra le sue prerogative quella di riuscire ad estrarre autonomamente le variabili migliori direttamente dai dati grezzi.

Il deep learning ha permesso di **migliorare drasticamente** i risultati raggiunti in passato in tantissimi settori, consentendo ad esempio lo sviluppo di auto a guida autonoma, assistenti virtuali in grado di comprendere una conversazione e di fornire risposte alle nostre domande o macchinari medicali capaci di identificare masse tumorali con una precisione maggiore rispetto a quella umana.

All'interno di questo elaborato di tesi verranno analizzati e sperimentati approcci recenti in ambito delle *convolutional neural network* (CNN) e deep learning (DL), allo scopo di identificare se le persone indossano le mascherine mediche.

Il lavoro di tesi è stato suddiviso nei seguenti capitoli:

- **Capitolo 1** - Introduzione al contesto applicativo e alle tecnologie utilizzate;
- **Capitolo 2** - Panoramica delle possibili soluzioni al problema;
- **Capitolo 3** - Analisi del modello di conoscenza utilizzato per la soluzione definitiva;
- **Capitolo 4** - Introduzione al deep learning su Android e all'applicazione sviluppata;
- **Capitolo 5** - Il lavoro realizzato e il codice impiegato.

Indice

Elenco delle figure

Capitolo 1

Analisi preliminare del problema

In questo primo capitolo viene analizzato il contesto applicativo dell'elaborato di tesi e verranno descritte più precisamente le tecnologie accennate nell'introduzione, verranno poi definiti gli obiettivi finali.

1.1 Contesto applicativo

La pandemia COVID-19 attualmente in corso denominata anche "coronavirus" ha destabilizzato la nostra società e il nostro modo di vivere, l'utilizzo di mascherine mediche e il distanziamento sociale sono diventati essenziali per evitare la diffusione del virus.

Questo elaborato di tesi ha lo scopo di individuare se le persone indossano le mascherine mediche o meno grazie al supporto del machine learning e in particolare del deep learning, con lo scopo di limitare i contagi tra le persone.

Da internet è possibile reperire un gran numero di immagini contenenti persone che indossano una mascherina medica e altrettante immagini di persone che non indossano alcuna mascherina medica. Raccogliere un grande insieme di immagini con il più alto e ugual numero di immagini per tipo (mascherina e non) possibile aiuta un modello di conoscenza nel suo processo di addestramento ad apprendere meglio le differenze tra questi due tipi di immagini. Un modello di conoscenza è capace di imparare autonomamente grazie ad un elevato numero di esempi e dati, e questo grazie all'utilizzo del machine learning.

1.2 Machine Learning

Il machine learning è una tecnica nata recentemente che fornisce un nuovo approccio a problemi in cui l'obiettivo è quello di predire, stimare o ipotizzare "qualcosa" sulla base di dati.

Una delle definizioni più citate quando si parla di questa tecnologia, è quella proposta dal professor Tom Mitchell [1]:

> *Si dice che un programma apprende dall'esperienza E con riferimento ad alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E.*

Riuscire a far eseguire un determinato compito a una macchina, che riesce in maniera autonoma e indipendente a migliorare il modo con cui esegue quel determinato compito, usando l'esperienza acquisita dai dati, imparando in maniera sempre più precisa e senza dover essere stata programmata in alcun modo per farlo. Quindi con questo approccio la macchina impara sulla base di esempi e dati senza la necessità di dover sviluppare un algoritmo per eseguire il compito richiesto.

Per citare qualche esempio di pratico di utilizzo del machine learning:

- Predizione di consumi di elettricità;
- Categorizzazione di animali, macchine, oggetti di vario tipo;
- Previsione indici di borsa;

Al giorno d'oggi questa tecnologia è impiegata in diversi ambienti e settori: sanità, economia, trading di borsa, management aziendale e previsioni di diverso genere. Soprattutto utilizzata nel settore tecnologico come quello degli smartphone, computer e dispositivi vari per migliorare ad esempio la qualità delle foto scattate oppure riconoscere il tipo di attività che si sta svolgendo tramite i sensori di un smartwatch o di un telefono.

L'arrivo di questi metodi è stato sostenuto anche dalla crescente disponibilità di dati da poter analizzare e utilizzare, questi sono facilmente trovabili su internet e utilizzabili per eseguire determinati compiti, non solo, l'aumentare delle risorse computazionali degli elaboratori ha dato modo di creare le condizioni ideali per lo sviluppo di queste tecnologie sempre più sofisticate.

Un ulteriore esempio che ci si presenta tutti i giorni sono i cosiddetti metodi di *recommendation*, questi metodi sono utilizzati prevalentemente su siti di e-commerce per suggerire ad esempio articoli secondo i propri gusti o ricerche fatte in passato.

Figura 1.1: Esempio di un sistema di suggerimenti che propone prodotti simili a quelli già visti.

1.2.1 Sviluppo di un modello

Nella pratica, alla base del machine learning c'è un **modello** che è la parte fondamentale del sistema che si vuole costruire. Il modello deve riuscire ad approssimare nel modo migliore possibile i dati, trovando correlazioni tra loro.

In base all'organizzazione dei dati, i problemi che si possono riscontrare durante lo sviluppo di un sistema di machine learning possono rientrare in una delle seguenti tipologie:

1. Apprendimento supervisionato;
2. Apprendimento non supervisionato;

1.2.2 Apprendimento supervisionato

Nell'apprendimento supervisionato i dati disponibili sono composti da due parti parti principali: una parte di input da dare al modello che contiene uno o più campi descrittivi che descrivono l'oggetto, e una parte in cui è specificato il valore reale dell'output.

Prendiamo come esempio il seguente dataset supervisionato, utilizzato per identificare la specie di iris partendo dai dati generici del fiore:

Lung. sepalo	**Larg. sepalo**	**Lung. petalo**	**Larg. petalo**	**Specie**
5.7	4.4	1.5	0.4	*Setosa*
7.7	3.8	6.7	2.2	*Virginica*
5.4	3.7	1.5	0.2	*Setosa*
7.2	3.6	6.1	2.5	*Virginica*
6.0	3.4	4.5	1.6	*Versicolor*
6.2	3.4	5.4	2.3	*Virginica*

Tabella 1.1: Esempio di dati con le caratteristiche di diversi fiori.

Le prime 4 colonne rappresentano le caratteristiche di ogni fiore da passare in input al modello per far si che comprenda le correlazioni che ci sono tra input e l'output, mentre l'ultima colonna rappresenta il tipo di fiore e il valore che ci si aspetta in output.

Il modello, avendo a disposizione il risultato corretto (ultima colonna) può effettuare una predizione, e in base alla differenza tra risultato esatto e risultato predetto comportarsi di conseguenza.

Sostanzialmente un algoritmo di machine learning supervisionato segue il seguente schema:

- Predizione in base all'input;
- Calcolo dell'errore del modello (differenza tra risultati attesi e predetti);
- Miglioramento del modello sulla base degli errori commessi.

Questo processo è definito **addestramento** del modello, e viene eseguito finché l'errore ottenuto non è abbastanza piccolo da poter considerare il modello valido.

I modelli supervisionati sono generalmente utilizzati per due diversi tipi di problemi:

- **Problemi di regressione** - Predire un valore numerico, come ad esempio la predizione del consumo energetico sulla base della temperatura.
- **Problemi di classificazione** - Come nell'esempio della tabella vista sopra relativa alla specie corretta dell'iris, abbinare l'elemento alla categoria corretta;

1.2.3 Apprendimento non supervisionato

A differenza dell'apprendimento supervisionato in cui è presente l'output previsto, nei modelli di apprendimento non supervisionato non c'è alcuna colonna dedicata all'output.

Ovviamente i dati per gli algoritmi di apprendimento non supervisionato sono molto più facili da reperire, poiché non è necessaria l'analisi di una persona umana (il dataset d'esempio sui fiori visto precedentemente ha richiesto l'identificazione della specie corretta per ogni fiore presente).

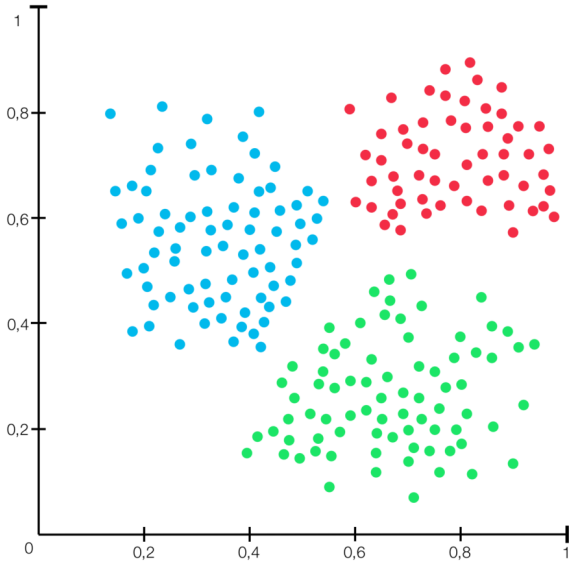


Figura 1.2: Rappresentazione grafica di un modello non supervisionato.

Non avendo alcun tipo di output atteso non esiste un metodo generico per controllare l'errore sviluppato dal modello.

Generalmente questo approccio viene utilizzato nei problemi di **clustering**, quando cioè è necessario classificare un insieme di elementi senza sapere a priori a che categoria appartengono.

Il fatto di non conoscere in partenza le diverse categorie, risulta però essere in molti casi vantaggioso, poiché il modello riesce a trovare legami spesso invisibili all'uomo.

1.2.4 Discesa del gradiente

Il metodo della discesa del gradiente è la base del machine learning, ciò che permette al modello di migliorarsi iterazione dopo iterazione.

In matematica viene utilizzato per trovare i punti di minimo e di massimo all'interno di una funzione, procedendo a step graduali.

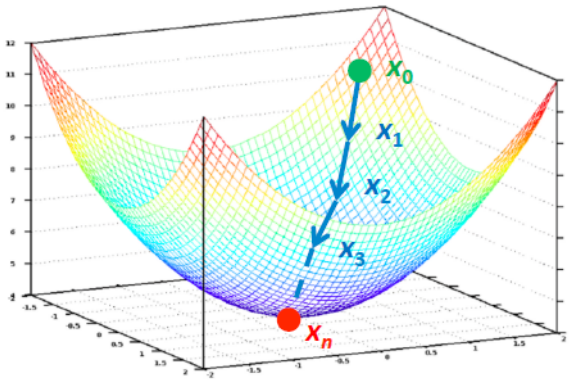


Figura 1.3: Discesa del gradiente dal punto x0 al punto xn

Quando nel processo di addestramento del modello vengono generate le predizioni, è possibile sviluppare una funzione che associa ad ognuna l'errore generato.

Si aggiunge quindi una dimensione alla funzione del modello, così da poter applicare la discesa del gradiente e minimizzare l'errore generato dalle predizioni future.

Nel grafico sopra, le dimensioni x e y che compongono il piano sono quelle relative ai dati di input, mentre la terza dimensione, la z, viene aggiunta per associare l'errore prodotto dal modello.

Per ridurre l'errore è quindi necessario:

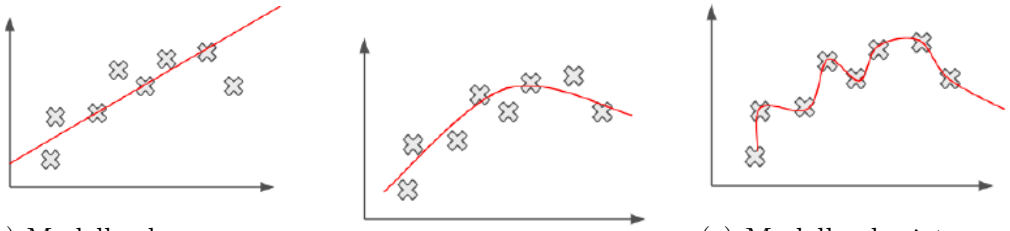
- Calcolare la funzione d'errore e il suo valore attuale ;
- Calcolare il gradiente della funzione nel punto, quindi *la direzione* da seguire per raggiungere il punto di minimo;
- Sottrarre al punto iniziale un vettore proporzionale al gradiente;
- Ricominciare da capo finché non viene raggiunto il punto obiettivo.

Questo approccio permette di analizzare un insieme di dati che possono essere rappresentati da una funzione di vario tipo e cercare di approssimare al meglio le variabili.

1.2.5 Validazione del modello

Quando dobbiamo addestrare un modello, abbiamo un insieme di dati su cui addestrare il modello, non tutti i dati vengono utilizzati per la fase di addestramento, ma si suddivide in due o tre sottogruppi differenti e non sovrapposti.

Questo modo di approcciare i problemi è detto **hold-out**, e prevede la divisione dei dati in due insiemi secondo una proporzione nota (generalmente 30%-70%). Sull'insieme più grande verrà addestrato il modello, mentre il restante più piccolo verrà utilizzato per validare il modello.



(a) Modello che non generalizza bene i dati, caso di underfitting

(b) Modello ottimale, generalizza bene i dati

(c) Modello che è troppo attaccato ai dati, caso di overfitting

Figura 1.4: Esempio di diversi modelli.

Nelle 3 immagini sopra vengono mostrati i diversi casi:

- Nel primo caso il modello è troppo semplice per generalizzare bene i dati, caso di **underfitting**;
- Nel secondo caso il modello generalizza correttamente i dati e le predizioni sui dati di addestramento e di validazione sono molto buone, modello ottimale;
- Nell'ultimo caso il modello è troppo "attaccato" ai dati e quindi avremo risultati molto buoni sui dati di addestramento ma non su quelli di validazione, caso di **overfitting**;

Quando si addestra un modello, il rischio infatti è quello di addestrare troppo il modello, in grado di produrre risultati molto accurati sui dati cui viene addestrato e risultati meno accurati su dati che non conosce.

Un buon sistema di machine learning dovrebbe essere il più generale possibile sui dati, per questo nella suddivisione hold-out viene utilizzata la parte più piccola per valutare il comportamento del modello dopo l'addestramento.

1.2.6 Deep Learning

Il machine learning, però, come già introdotto all'inizio presenta delle limitazioni ad esempio nell'elaborazione di immagini o testo oppure quando la quantità di dati disponibili per l'addestramento non è così abbondante.

Inoltre per poter utilizzare il machine learning è necessario selezionare le feature (variabili) manualmente, e ciò richiede competenze.

Al fine di affrontare queste limitazioni poste dal machine learning, negli ultimi anni una branca del machine learning ha guadagnato sempre più popolarità e interesse nel campo dell'intelligenza artificiale: il **deep learning**.

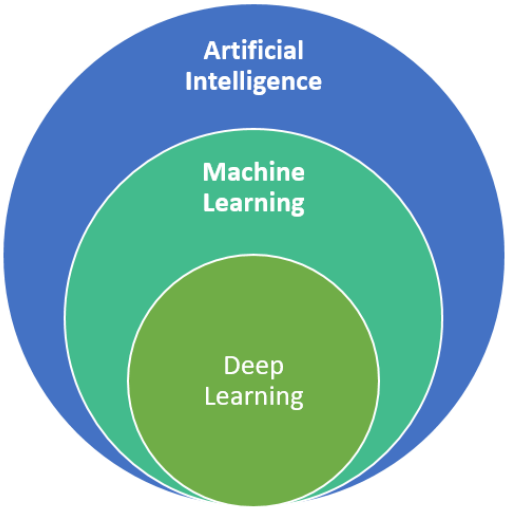


Figura 1.5: Gerarchia dell'intelligenza artificiale

Il deep learning trova fondamento e ispirazione nella struttura dei neuroni all'interno del cervello umano, dove non esiste un unico punto di ingresso nel quale elaborare i dati in entrata, ma al contrario esistono molti più nodi, suddivisi in diversi strati che collaborano tra loro per raggiungere risultati estremamente accurati.

All'interno delle **reti neurali** sviluppate tramite deep learning non si ritrova un solo modello, ma un grande numero di neuroni raggruppati all'interno di strati.

Dopo aver addestrato la rete neurale sui dati di training, questa riesce ad eseguire i compiti più complessi, con una precisione migliore rispetto a quella raggiungibile tramite machine learning.

La divisione in strati della rete neurale in neuroni consente di gestire in maniera progressiva l'astrazione dei problemi da affrontare: nei livelli più bassi si riconoscono aspetti semplici, mentre negli strati più alti aspetti più complessi.

Un esempio reale di quanto una rete neurale sia molto più accurata rispetto a metodi tradizionali, è l'applicazione a problemi riguardanti le immagini come il riconoscimento di persone o nel caso di questo elaborato di tesi riconoscere se le persone indossano mascherine mediche. Le reti neurali utilizzate prevalentemente per le immagini sono quelle convoluzionali.

1.2.7 Struttura delle reti neurali

Come già detto, un sistema di deep learning si sviluppa in una struttura molto più complessa rispetto a un modello di machine learning. Si creano degli strati denominati **layer** contenenti neuroni che utilizzano determinati algoritmi per svolgere il compito richiesto.

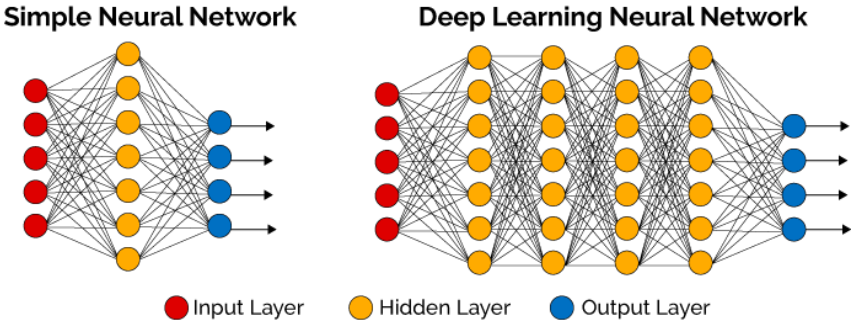


Figura 1.6: Struttura tipica di una rete neurale

Gli strati sono collegati tra loro, e ogni informazione in un'uscita da uno strato sono quelle in entrata nel successivo. Ogni neurone di uno strato è collegato ad ogni neurone dello strato successivo. Solitamente il primo layer viene utilizzato per l'input della rete neurale, mentre l'ultimo layer per l'output, gli strati intermedi vengono chiamati strati nascosti o più comunemente **hidden layers**.

In parole povere, un neurone in una rete neurale artificiale è: un insieme di valori in input **Xi** e pesi associati **Wi**, funzione che somma i pesi e mappa i risultati su un output. Il neurone non fa altro che eseguire la regressione per minimizzare l'errore e raggiungere risultati sempre più precisi.

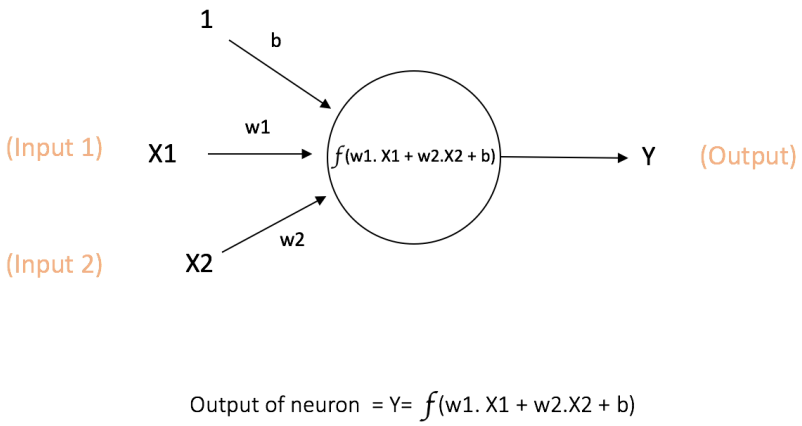


Figura 1.7: Neurone di una rete neurale

L'aggregazione di molti neuroni nel formare una rete provoca un'esplosione di complessità, in quanto si possono formare topologie complesse che possono prevedere anche cicli tra i neuroni della rete (reti neurali ricorrenti). I modelli più semplici non prevedono cicli e vengono chiamate reti feed-forward.

Una volta che una rete è stata strutturata per una particolare tipo di problema, quella rete è pronta per essere addestrata.

1.2.8 Addestrare una rete neurale

Nella fase di addestramento, è noto l'output corretto per ogni istanza (addestramento supervisionato), e ai nodi di output possono quindi essere assegnati valori "1" per il nodo corrispondente alla classe corretta e "0" per gli altri. È quindi possibile confrontare i valori calcolati dalla rete per i nodi di output con questi valori "corretti" e calcolare un errore per ogni nodo. Questi errori vengono quindi utilizzati per regolare i pesi negli strati nascosti per raggiungere un output sempre più vicino ai valori "corretti".

Una caratteristica chiave nell'addestramento delle reti neurali è un processo di apprendimento iterativo in cui i dati vengono presentati alla rete e i pesi associati (inizialmente casuali) ai valori di input vengono regolati ogni volta. Dopo che tutti i dati sono stati presentati, il processo spesso ricomincia da capo. Durante questa fase di apprendimento, la rete apprende regolando i pesi in modo da poter prevedere la corretta classe dei campioni di input.

L'algoritmo di rete neurale più popolare è l'algoritmo di **back-propagation** [2].

1.2.9 Feedforward, Back-Propagation

Attualmente, l'architettura di back-propagation sviluppata sinergicamente è il modello più popolare, efficace e di facile apprendimento per reti complesse e multistrato. La tipica rete di back-propagation ha un livello di input, uno di output e almeno uno nascosto.

Durante il processo di addestramento, viene eseguita una scansione in avanti attraverso la rete e l'output di ogni elemento viene calcolato layer per layer. La differenza tra l'output del layer finale e l'output desiderato viene propagata all'indietro ai layer precedenti, e i pesi di connessione vengono normalmente regolati utilizzando l'errore.

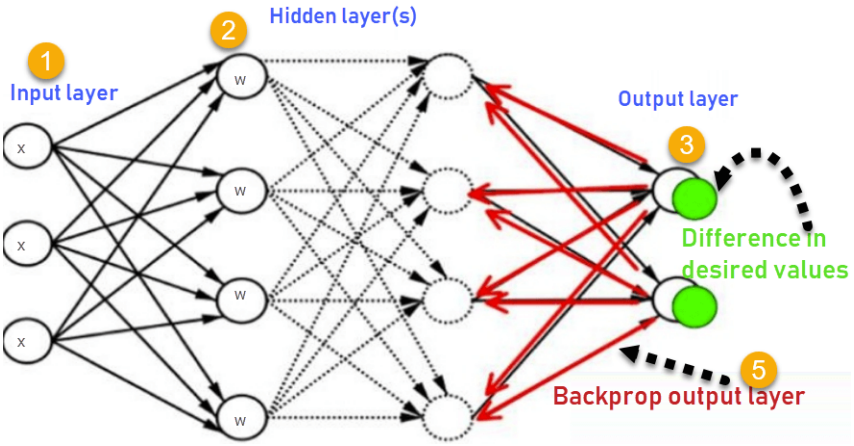


Figura 1.8: Algoritmo di back-propagation in una rete neurale

1.2.10 Funzioni di attivazione

Le reti neurali, costituite da layer di neuroni necessitano di **funzioni di attivazione** che permettono di ottenere l'output del neurone o della rete neurale da un input.

Esistono due principali tipi di funzioni di attivazione:

- **Funzioni di attivazione lineari o di identità**: l'output non viene limitato da nessun intervallo, l'output è proporzionale all'input;
- **Funzioni di attivazione non lineari**: permettono al modello di creare mappature complesse tra gli input e gli output;

Solitamente nelle reti neurali vengono utilizzate le funzioni di attivazione non lineari poiché le funzioni di attivazione lineari non permettono di creare mappature complesse e l'output finale corrisponderebbe alla funzione lineare dell'input.

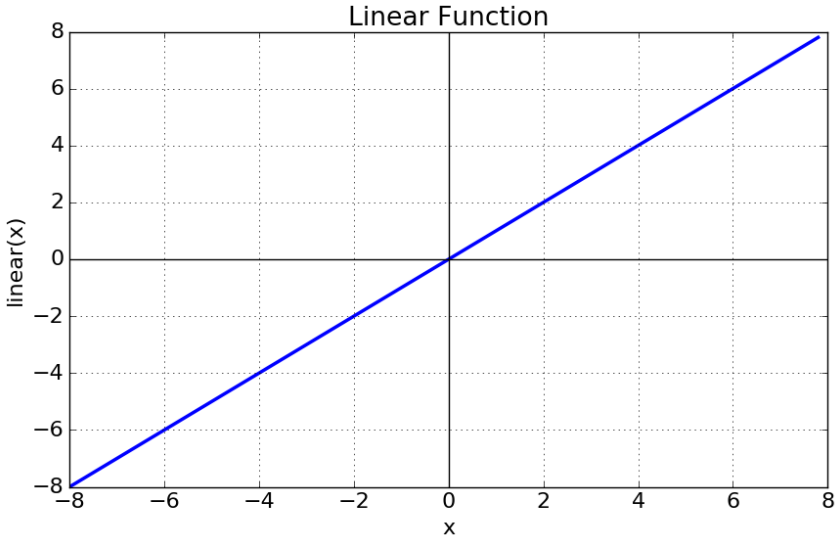


Figura 1.9: Esempio di funzione di attivazione lineare

Mentre le funzioni di attivazione non lineari consentono al modello l'apprendimento e la modellazione di dati complessi, come immagini, video, audio e set di dati che non sono lineari o hanno un'elevata dimensionalità.

Le funzioni di attivazione non lineari più utilizzate sono le seguenti:

- **Sigmoide**: la funzione sigmoide fornisce una curva a forma di "S". Utilizzata per mappare i valori in probabilità, la funzione mappa qualsiasi valore reale in un altro valore compreso tra 0 e 1.

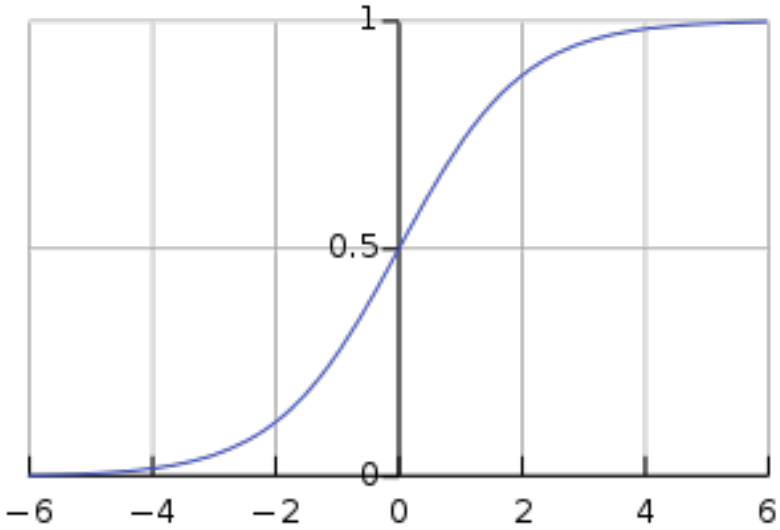


Figura 1.10: Funzione sigmoide

- **ReLU**: utilizzate principalmente in hidden layer, ampiamente utilizzata, è la scelta predefinita in quanto produce risultati migliori.

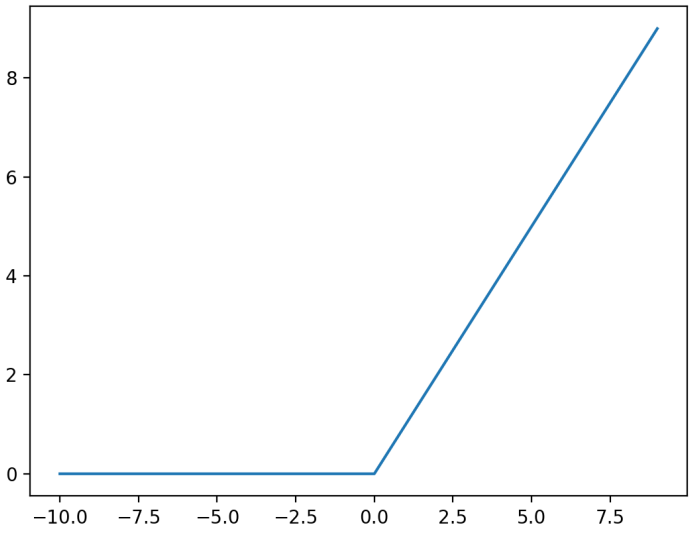


Figura 1.11: Funzione ReLu

- **Softmax**: la funzione Softmax calcola la distribuzione delle probabilità dell'evento su 'n' diversi eventi.

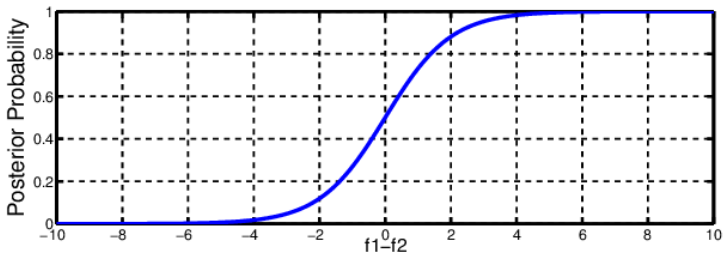


Figura 1.12: Funzione softmax

Tutte le funzioni di attivazione non lineari hanno caratteristiche differenti tra loro e ognuna viene utilizzata in base al tipo di problema da dover affrontare.

1.2.11 Convolutional Neural Network (CNN)

Come già accennato in precedenza, le convolutional neural network sono delle reti neurali utilizzate principalmente per l'elaborazione di immagini, utilizzate ad esempio nella classificazione di immagini di vari tipi di oggetti.

I computer però vedono le immagini come un insieme di pixel, un'immagine ha formato del tipo H x W x D (altezza, larghezza, dimensione), ad esempio un'immagine di dimensioni 260 x 260 x 3 avrà in totale 67.600 pixel ognuno con valore da 0 a 255.

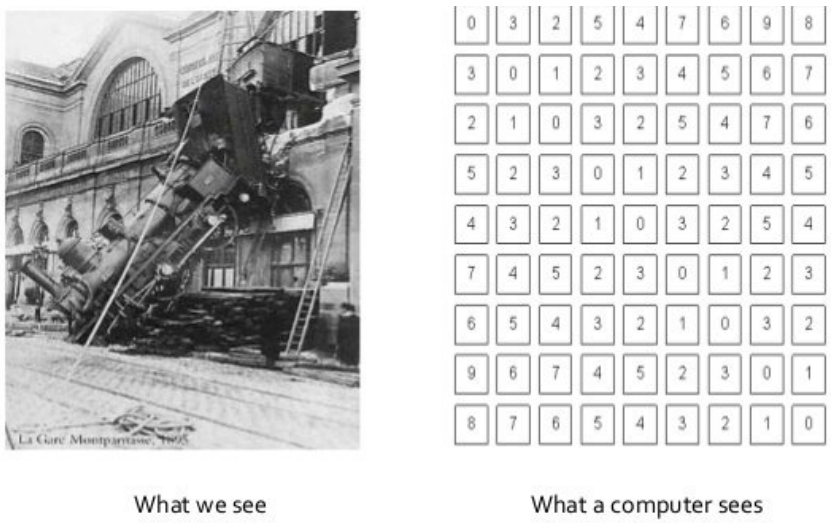


Figura 1.13: Esempio di come un computer vede le immagini

Le reti convoluzionali hanno due componenti principali:

- Apprendimento delle features: vengono estratti bordi, sfumature, linee, curve dall'immagine;
- Classificazione: assegnare una probabilità che l'oggetto sull'immagine sia ciò che l'algoritmo prevede;

1.2.12 Strati convoluzionali

Sono strati che vengono utilizzati per estrarre delle features da immagini, utilizzando piccoli quadrati di pixel chiamati **kernel**.

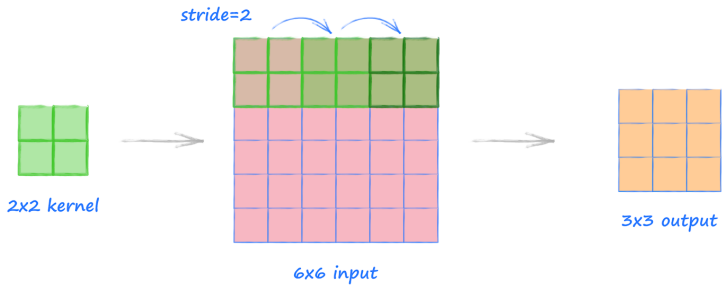


Figura 1.14: Immagine di 6x6 pixel con kernel 2x2 pixel e con passo 2

Nell'immagine sopra il passo chiamato anche **stride** permette di muovere il quadrato 2x2 di passo 2.

1.2.13 Sottocampionamento

A seguito di uno strato convoluzionale è comune utilizzare uno strato di sottocampionamento che riduce le dimensioni della matrice di output, le operazioni più comuni sono due:

- MaxPooling: ogni cella della matrice di output contiene il valore massimo della sottomatrice di partenza;
- AveragePooling: il valore di ciascuna cella è la media dei valori della sottomatrice di partenza;

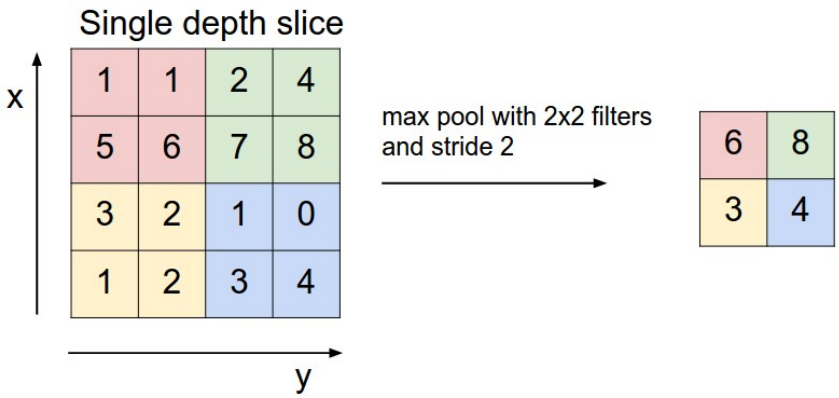


Figura 1.15: MaxPooling su una matrice 4x4 con kernel 2x2 e passo 2, la matrice di output contiene i valori massimi delle 4 sottomatrici

Esempio di una classica rete convoluzionale:

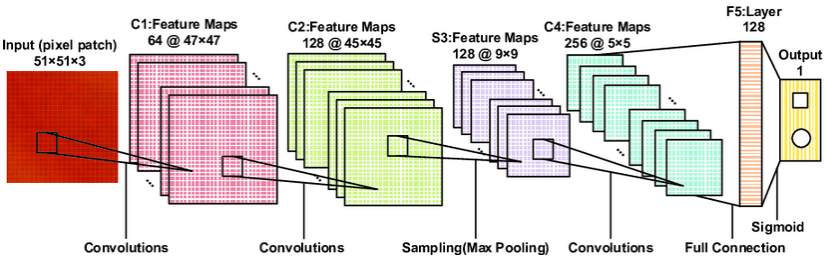


Figura 1.16: Struttura di una classica rete convoluzionale

1.3 Obiettivo

L'obiettivo di questa tesi è quello di sviluppare una applicazione Android in grado tramite la fotocamera del cellulare di classificare in tempo reale se le persone inquadrate indossano una mascherina medica o meno.

Figura 1.17: Esempio di immagine di una persona che indossa una mascherina medica.

Verranno utilizzati approcci esistenti al problema, usando metodi di **deep learning** in particolare utilizzando le **convolutional neural network** per l'elaborazione di immagini.

Capitolo 2

Panoramica delle possibili soluzioni al problema

In questo capitolo verranno descritte e analizzate le diverse soluzioni trovate e i diversi modelli sperimentati prima di giungere alla soluzione definitiva. Verranno analizzate le ulteriori tecnologie utilizzate, i problemi e limiti che le soluzioni imponevano.

2.1 Workspace e librerie utilizzate

Tutte le soluzioni e i diversi modelli di questo capitolo sono stati testati sulla piattaforma Google Colaboratory che permette di eseguire codice Python sul proprio browser eseguendo il codice in celle in cui ogni cella può essere eseguita in modo indipendente dalle altre. Utile per chi tratta soprattutto la data science e che offre: accesso gratuito alle GPU (schede video), condivisione semplificata e nessuna configurazione iniziale necessaria.

Per poter utilizzare i vari modelli di rete neurale è stato utilizzato il framework TensorFlow [3], che fornisce funzioni ottimizzate utili per il machine learning e il deep learning.

Inoltre sono state utilizzate maggiormente queste due librerie:

- **OpenCV**: per la lettura, l'elaborazione e la conversione delle immagini è stata utilizzata OpenCv [4], una libreria utilizzata nell'ambito della visione artificiale;

- **Numpy**: per poter utilizzare vettori e matrici è stata utilizzata la liberia Numpy [5], che aggiunge supporto a matrici e vettori multidimensionali insieme a funzioni da poter utilizzare su di essi;

2.2 Dataset RMFD

Inizialmente è stato reperito un dataset chiamato RMFD (Real-World Masked Face Dataset) contente immagini di persone con e senza mascherine, il dataset utilizzato è reperibile da GitHub [6].

Il Dataset al suo interno contiene solamente 406 immagini di persone con mascherina e circa 90 mila immagini di persone senza mascherina, questo comporta l'impossibilità di addestrare al meglio una rete neurale che data la grande differenza di classi di immagini non può generalizzare bene sui dati. Perciò è stato deciso di utilizzare una tecnica chiamata **data augmentation** in grado di aumentare il numero delle immagini di persone con mascherina elaborandole in diversi modi.

2.2.1 Data augmentation

La data augmentation è un processo che permette l'aumento della quantità e della diversità dei dati, non vengono collezionati nuovi dati bensì i dati già esistenti vengono trasformati in nuovi dati.

Nel nostro caso utilizzando la data augmentation sulle immagini sarà possibile aumentare le immagini di persone con mascherine.

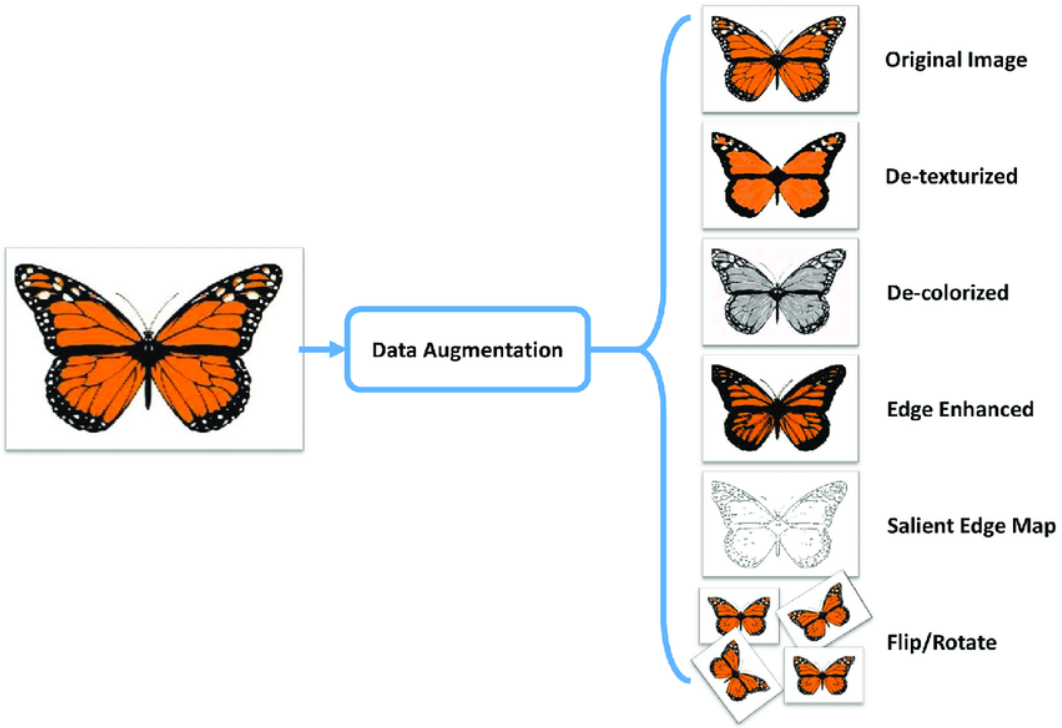


Figura 2.1: Esempio di data augmentation su immagini di farfalle.

Come si può notare dalla figura sopra, la data augmentation a partire da un'immagine permette di effettuare diverse elaborazioni su di essa, come ad esempio:

- Ruotare l'immagine di diversi gradi
- Capovolgere l'immagine in orizzontale e verticale
- Ritagliare l'immagine
- Trasformare l'immagine da RGB a scala di grigi
- Applicare un fattore di saturazione all'immagine
- Cambiare la luminosità all'immagine
- ...

Grazie alla data augmentation è stato possibile portare le immagini da 406 a svariate decine di migliaia, così facendo sarebbe stato possibile addestrare una rete neurale.

2.2.2 Xception

Inizialmente è stata utilizzata la rete **Xception** [7] che è una rete convoluzionale già addestrata su uno dei più popolari dataset che è **ImageNet** [8] contenente milioni di immagini su migliaia di categorie di oggetti.

La rete può classificare migliaia di categorie di oggetti come animali, tastiere, penne, macchine, ecc...

2.2.3 Architettura di Xception

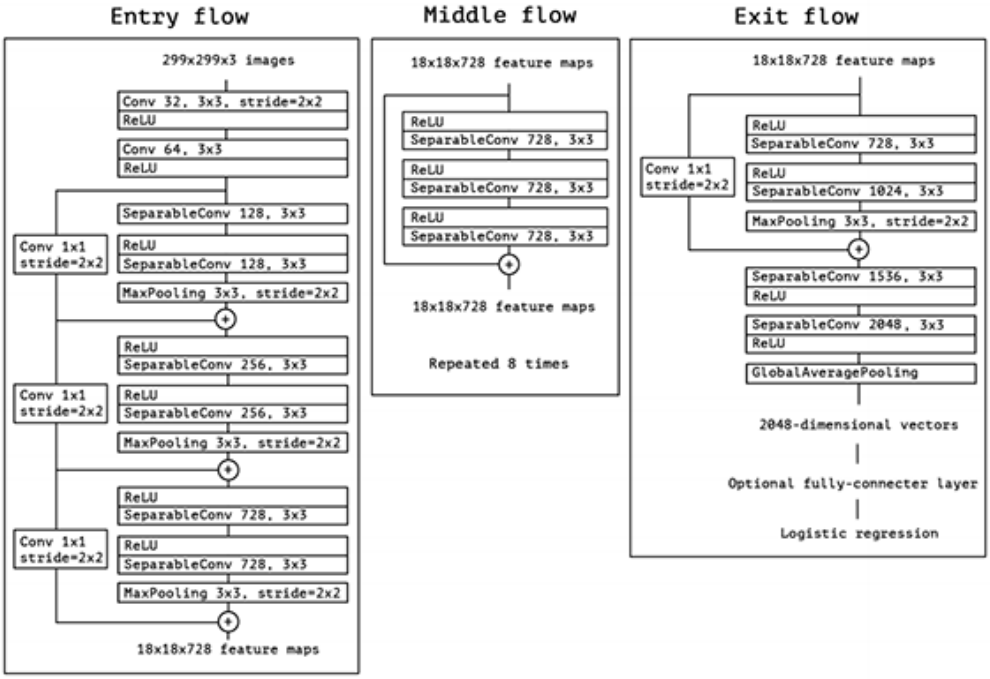


Figura 2.2: Architettura di Xception.

La rete è composta principalmente di layer convoluzionali per l'estrazione delle feature e layer di pooling per ridurre le dimensioni, vengono utilizzate anche delle connessioni residue (come scorciatoie) originalmente proposte dalla rete **ResNet** [9] che hanno permesso di raggiungere un'elevata accuratezza.

I layer denominati "SeparableConv" corrispondono alla convoluzione separabile in profondità modificata da **Inception-v3** [10], i layer "SeparableConv" sono trattati come moduli di Inception e inseriti nell'intera architettura della rete Xception.

Tra i modelli migliori addestrati su ImageNet Xception si è rilevata una delle migliori [11].

Xception è stata utilizzata come rete di partenza in Python utilizzando in particolare il transfer learning che verrà descritto a breve.

L'attuale rete che rappresenta lo **stato dell'arte** sul dataset ImageNet è **FixEfficientNet-L2** [12].

2.2.4 Transfer Learning

Il transfer learning permette di riutilizzare gran parte dei parametri di una rete neurale già addestrata in precedenza su un problema simile a quello che bisogna risolvere, soffermandoci solo sull'addestramento degli ultimi layer che sono solitamente quelli dedicati alla classificazione o regressione delle feature ottenute con i layer precedenti.

Questo consente di riutilizzare il comportamento di una rete già addestrata ad estrarre efficacemente feature dai dati di input e di limitare l'elaborazione ad un numero sensibilmente minore di parametri, riducendone il tempo.

Gli strati riutilizzati (i primi) verrebbero etichettati come "non addestrabili", così da addestrare solo gli ultimi layer, velocizzando di molto i tempi addestramento e in generale aumentando l'accuratezza della rete.

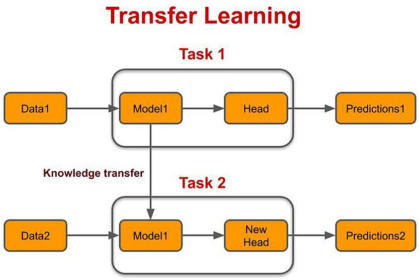


Figura 2.3: Esempio di transfer learning.

Come si può notare dalla figura sopra, della rete neurale pre-addestrata si mantengono intatti solo gli strati iniziali, andando a ridefinire solo gli ultimi strati di classificazione.

Un esempio potrebbe essere quello di dover classificare le varietà di mele da una immagine, si potrebbe utilizzare una rete neurale già addestrata a classificare immagini di aerei, cani, gatti, ecc.. Questo perché la maggiore varietà del dataset di addestramento garantisce una migliore capacità di estrazione feature di vario genere dalle immagini.

2.2.5 Addestramento di Xception

E' stato utilizzato il transfer learning per permettere alla rete Xception di classificare le immagini riguardanti persone che indossano una mascherina medica o meno. E' stato fatto ciò rendendo "non addestrabili" i primi layer che identificano feature di basso livello dalle immagini e addestrare quindi solo gli ultimi layer che permettono di ricavare le feature di livello più alto.

L'addestramento è stato fatto su gran parte del dataset RMFD contenenti le immagini cercando di avere lo stesso numero di immagini per classe (mascherina e non). Per l'addestramento sono stati utilizzati il 70% delle immagini, mentre per la parte di validation il restante 30%.

L'addestramento della rete ha richiesto svariati giorni sulla piattaforma Google Colaboratory.

2.2.6 Risultati di Xception su RMFD

I risultati della rete alla fine dell'addestramento sul dataset RMFD non sono stati soddisfacenti in quanto la rete ha raggiunto un'accuratezza finale del 50% circa, cioè la stessa accuratezza che avrebbe un modello che effettua predizioni casualmente (senza utilizzare un modello sulla base di dati).

Xception non sarebbe stata comunque molto efficace in quanto non è stata pensata per rilevare più oggetti (in questo caso persone) per ogni immagine e non sarebbe stata in grado di trovare le coordinate relative ai volti.

2.2.7 Possibili soluzioni

Per far fronte al problema legato all'accuratezza e all'identificazione di più persone in una sola immagine, bisognerebbe utilizzare una rete neurale che sia abbastanza accurata ma che possa anche rilevare più persone e i volti di queste dalle immagini.

Si è deciso quindi di optare per reti neurali già addestrate sul problema posto per risparmiare tempo e risorse di calcolo dovute all'addestramento.

2.3 Rete neurale di AIZOO

Dopo aver testato e notato che Xception non dava risultati soddisfacenti sui dati di validation è stato deciso di utilizzare una rete neurale già addestrata trovata su GitHub [13].

La seguente rete neurale è stata sviluppata appositamente con lo scopo di identificare se le persone indossano o meno una mascherina medica. La rete neurale è stata sviluppata per i 5 framework più utilizzati nel deep learning tra cui TensorFlow e TensorFlow Lite.

L'addestramento è stato fatto su un totale di 8000 immagini circa e il dataset utilizzato è scaricabile dal link GitHub. Le immagini utilizzate sono state divise in training e validation.

La rete neurale permette di identificare uno o più persone contemporaneamente trovando anche le coordinate relative ai volti.

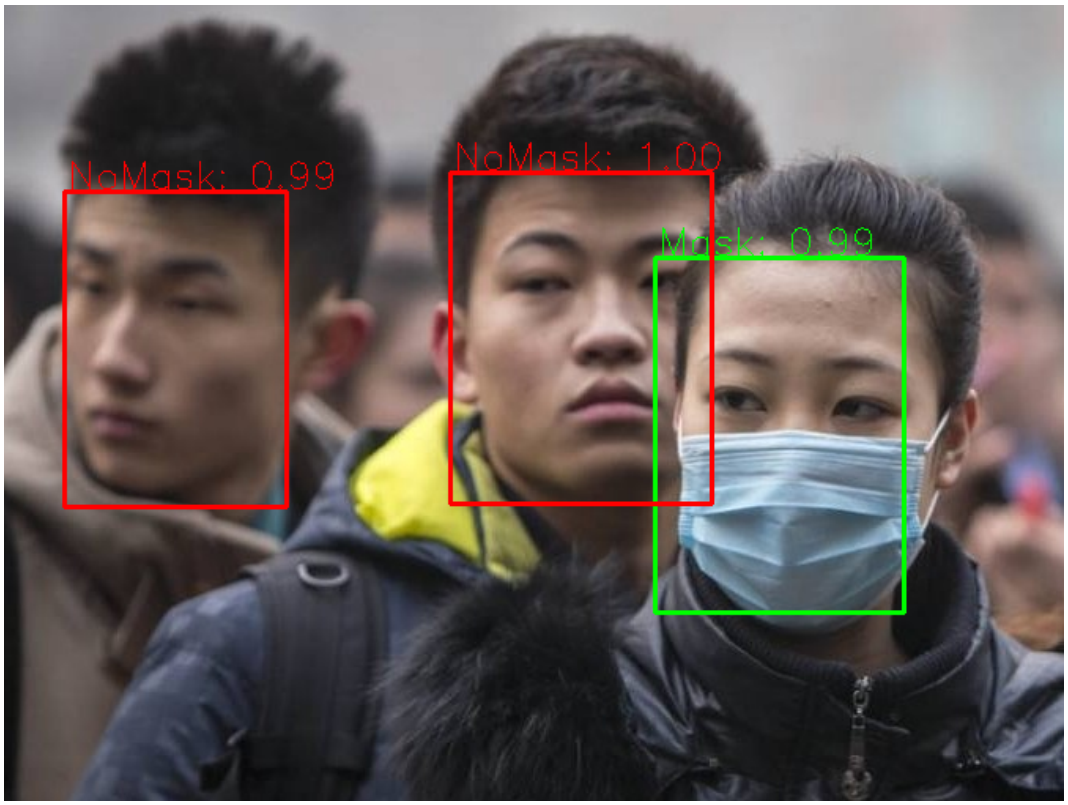


Figura 2.4: Demo della rete neurale.

L'esempio in figura sopra mostra il comportamento della rete data l'immagine, i volti delle persone che indossano la mascherina vengono contornati di verde mentre quelli che non la indossano di rosso, viene indicata anche la percentuale.

2.3.1 Struttura della rete neurale

La struttura di questa rete neurale è composta principalmente da layer convoluzionali per estrarre le feature dalle immagini e di layer di pooling per ridurre le dimensioni, questa rete a differenza di altre reti convoluzionali classiche utilizza più flussi separati per poter concatenare gli output dei singoli flussi tra loro.

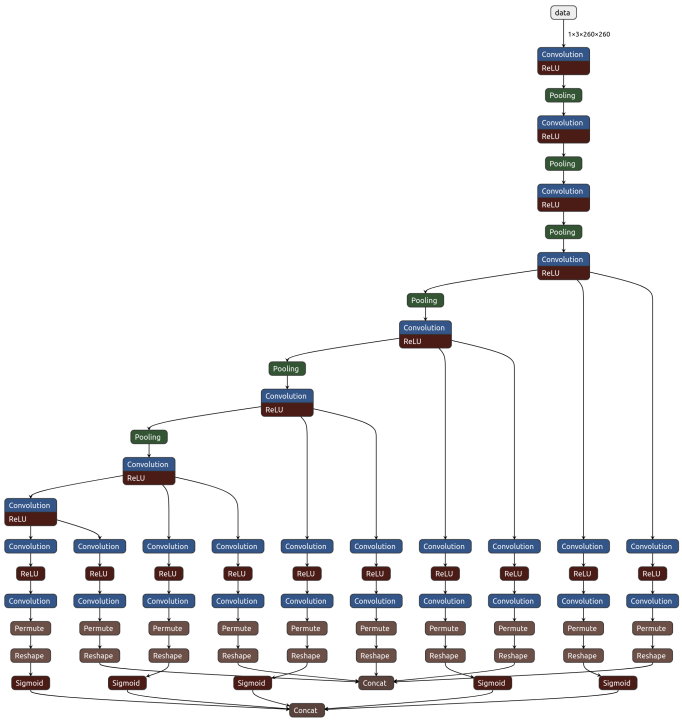


Figura 2.5: Architettura della rete di AIZOO.

La rete neurale accetta in input immagini di dimensioni di 260 x 260 pixel, convertendo le immagini da RGB in BGR è possibile avere una migliore accuratezza.

2.3.2 Workflow della rete neurale

Il workflow della rete dall'immagine in input alla predizione in output è il seguente:

- **Preprocessing dell'immagine**: il preprocessing dell'immagine viene eseguito prima di passare l'immagine alla rete neurale, consiste nel convertire l'immagine da RGB a BGR, ridimensionare l'immagine in 260 x 260 pixel e convertire i pixel da un range di 0-255 a un range di 0-1 per una migliore elaborazione;
- **Input**: l'immagine preprocessata viene passata in input alla rete neurale;
- **Output**: la rete in output restituisce due valori matriciali;

- **Postprocessing dell'output**: l'output non può essere utilizzato cosi com'è ma deve essere elaborato da diverse funzioni di computer vision che permettono di ottenere le coordinate relative ai volti delle persone e la predizione;

L'output finale contiene per ogni persona rilevata le coordinate del volto, se indossa la mascherina o meno e con quale percentuale.

2.3.3 Efficacia del modello e metriche

Per valutare il modello è necessario tenere conto di quali sono le label (etichette) reali, ovvero quante persone hanno la mascherina e quante non per ogni immagine. Le label insieme alle immagini sono reperibili dal link GitHub di AIZOO e in particolare le label sono all'interno di file XML.

Le immagini utilizzate durante la valutazione del modello sono immagini che il modello non ha mai visto.

Per valutare l'efficacia del modello sono state effettuate le predizioni su un totale di 1839 immagini, l'efficacia di un modello di classificazione viene misurata generalmente secondo queste metriche standard:

- **Accuratezza**: l'accuratezza indica quanto corretto è il modello a effettuare le predizioni, quindi il numero di volte che il modello effettua una predizione corretta su tutto il totale delle prove, si è raggiunta un'accuratezza intorno al 94%;

- **Precision**: la precision nei problemi di classificazione indica le istanze rilevanti sul totale delle istanze. La precision può essere calcolata sulle due classi separatamente, la precision relativa alla classe **no mask** è circa il 96% mentre quella della classe **mask** è del 93%. Quindi il modello generalmente è leggermente più preciso a riconoscere le persone senza mascherina;

- **Recall**: la recall al contrario della precision indica il numero di risultati corretti restituiti sul totale del numero dei risultati corretti che avrebbero dovuto essere restituiti. Anche qui come per la precision la recall può essere calcolata sulle due classi, per la classe **no mask** è del 92% mentre per la classe **mask** il 93%;

- **F1 score**: la f1 score è una misura di accuratezza che tiene conto sia di precision che recall, viene calcolata attraverso media armonica dei due. Come nelle altre due misure può essere calcolata separatamente su entrambe le classi, per la classe **no mask** è circa del 94% mentre per la classe **mask** 93%;

Le metriche di misura di un modello dipendono anche dal dataset che si è utilizzato, in questo caso è stato utilizzato un dataset contenente immagini abbastanza semplici, con altri dataset di immagini le metriche di accuratezza potrebbero essere migliori o peggiori.

Il seguente modello è stato testato anche sul dataset RMFD e i risultati ottenuti sono stati molto simili a quelli sul dataset utilizzato da AIZOO.

2.3.4 Limiti della soluzione

La rete neurale di AIZOO è risultata efficace nel suo utilizzo ma presenta dei limiti:

- Output complesso: l'output di questa rete è molto complesso e come già citato precedentemente necessita di non poche funzioni di computer vision per essere decodificato, questo comporta un utilizzo maggiore di risorse per eseguire diversi calcoli e un incremento dei tempi prima di ottenere la soluzione finale. Questo influisce negativamente sulle prestazioni e sui tempi di calcolo dato che la rete neurale deve essere portata su dispositivi Android;

- Efficace su immagini semplici: il modello è efficace ma su immagini relativamente semplici, utilizzando immagini relative a persone di profilo o persone in lontananza spesso queste non vengono identificate;

2.3.5 Possibili soluzioni

Per migliorare la seguente rete neurale si potrebbe modificare la sua struttura disponibile su GitHub in maniera tale da ottenere un output più semplice. In seguito si potrebbe riaddestrare la rete su molte immagini anche complesse, come di persone viste di profilo o in lontananza. Questo però richiederebbe alte capacità di progettazione di una rete neurale, ma non solo, richiederebbe molto tempo in giorni per addestrare il modello su migliaia di immagini.

La rete di AIZOO sarà analizzata anche sull'applicazione Android.

Capitolo 3

Analisi della soluzione definitiva

In questo capitolo verrà analizzata la rete neurale definitiva utilizzata in questo elaborato di tesi per il problema posto. La rete neurale trovata è disponibile su GitHub [14] e permette di riconoscere se le persone indossano una mascherina medica o meno attraverso immagini relativi a volti di persone.

3.1 MobileNetV2

La rete neurale utilizzata è stata addestrata utilizzando il transfer learning dalla rete **MobileNetV2** [15].

Nell'aprile 2017 un gruppo di ricercatori di **Google** ha pubblicato un documento in cui ha introdotto un'architettura di rete neurale ottimizzata per i dispositivi mobili.

L'architettura soprannominata **MobileNet** [16] ruota attorno all'idea di utilizzare convoluzioni separabili in profondità, che consistono in una convoluzione in profondità e una in senso uno dopo l'altra.

3.1.1 Blocchi residui

I blocchi residui collegano l'inizio e la fine di un blocco convoluzionale con una connessione di salto.

Aggiungendo questi due stati la rete ha la possibilità di accedere ad attivazioni precedenti che non erano state modificate nel blocco convoluzionale. Questo approccio si è rivelato essenziale per costruire reti di grande profondità.

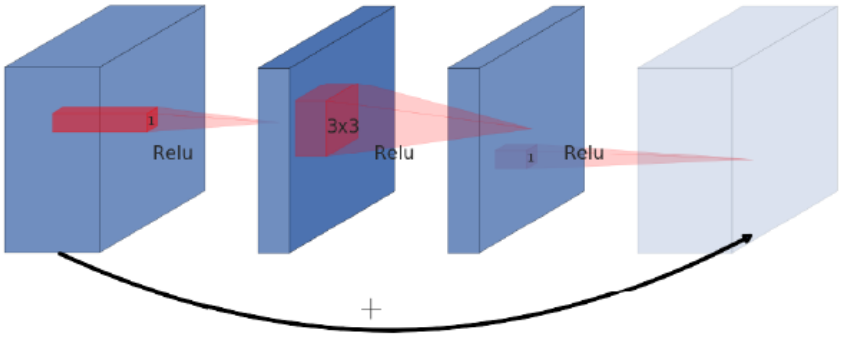


Figura 3.1: Un blocco residuo collega strati larghi con una connessione di salto mentre gli strati intermedi sono stretti.

Guardando un pò più da vicino la connessione di salto si nota che un blocco residuo **originale** segue un approccio **ampio -> stretto -> ampio** riguardo al numero di canali.

Il primo blocco ha un alto numero di canali, i quali vengono compressi con una convoluzione 1x1. In questo modo la seguente convoluzione 3x3 ha molti meno parametri. Per aggiungere input e output alla fine, il numero di canali viene nuovamente aumentato utilizzando un'altra convoluzione 1x1.

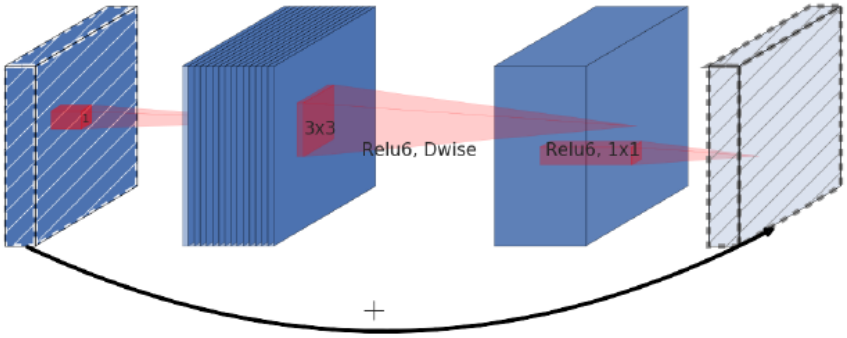


Figura 3.2: Un blocco residuo invertito collega strati stretti con una connessione di salto mentre gli strati intermedi sono larghi.

D'altra parte, MobileNetV2 segue un approccio **stretto -> ampio-> stretto** utilizzando dei blocchi residui **invertiti** .Il primo passaggio amplia la rete utilizzando una convoluzione 1x1 perché la successiva convoluzione 3x3 in profondità riduce già notevolmente il numero di parametri. Successivamente un'altra convoluzione 1x1 comprime la rete per far corrispondere il numero iniziale di canali.

Questa idea viene descritta come un blocco residuo invertito perché esistono connessioni di salto tra parti strette della rete che è l'opposto di come funziona

un blocco residuo originale. Utilizzando il blocco residuo invertito si hanno molti meno parametri.

3.1.2 Linear Bottlenecks

Il motivo per cui vengono utilizzate funzioni di attivazione non lineare nelle reti neurali è che le moltiplicazioni di matrici multiple non possono essere ridotte a una singola operazione numerica. Permettono di costruire reti neurali che hanno più livelli, allo stesso tempo, la funzione di attivazione ReLU, comunemente utilizzata nelle reti neurali, scarta valori inferiori a 0. Questa perdita di informazioni può essere contrastata aumentando il numero di canali per aumentare la capacità della rete.

Con i blocchi residui invertiti viene fatto l'opposto e vengono compressi i livelli in cui sono collegate le connessioni di salto. Questo danneggia le prestazioni della rete.

Gli autori hanno introdotto l'idea di un collo di bottiglia lineare (linear bottlenecks) in cui l'ultima convoluzione di un blocco residuo ha un'uscita lineare prima di essere aggiunta alle attivazioni iniziali. Basta fare ciò togliendo la funzione di attivazione non linare **ReLU** nell'ultimo blocco di convoluzione.

Il frammento di codice sotto mostra la struttura di un blocco convoluzionale che incorpora residui invertiti e linear bottlenecks.

```
''' expand = numero di features da apprendere per il primo "Conv2D"
    squeeze = numero di features da apprendere per il secondo
       "Conv2D" '''
def inverted_linear_residual_block(x, expand=64, squeeze=16):
  m = Conv2D(expand, (1,1), activation='relu')(x)
  m = DepthwiseConv2D((3,3), activation='relu')(m)
  m = Conv2D(squeeze, (1,1))(m)
  return Add()([m, x])
```

3.1.3 Batch Normalization e ReLU6

Per ottenere l'intera struttura di MobileNetV2 occorrono due ulteriori pezzi principali.

Il primo aspetto aggiunge semplicemente **la batch normalization** [17] dietro ogni livello convoluzionale. La normalizzazione batch è un metodo spesso utilizzato per rendere le reti neurali più veloci e più stabili attraverso la normalizzazione del livello di input.

La seconda aggiunta non è così comune infatti gli autori hanno utilizzato **ReLU6** invece di ReLU, che limita il valore delle attivazioni a un massimo di 6. L'attivazione è lineare fintanto che è compresa tra 0 e 6.

```
''' Funzione che implementa la ReLU '''
def relu(x):
  return max(0, x)

''' Funzione che implementa la ReLU6 '''
def relu6(x):
  return min(max(0, x), 6)
```

3.1.4 Architettura di MobileNetV2

Ora che sono stati descritti gli elementi principali di MobileNetV2, si può dare un'occhiata all'intera architettura.

Nella tabella si possono vedere come sono disposti i vari blocchi della rete. La colonna **t** rappresenta la velocità di espansione dei canali. La **c** rappresenta il numero di canali di ingresso e **n** quanto spesso il blocco viene ripetuto. Infine **s** ci dice se la prima ripetizione di un blocco ha utilizzato un passo di 2.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figura 3.3: Architettura di MobileNetV2.

Alla fine della struttura convoluzionale si trova una classica rete neurale.

3.2 Workflow della rete neurale

La rete neurale è disponibile in TensorFlow Lite, un framework open source che deriva da TensorFlow utilizzato principalmente per il deep learning su dispositivi mobile come Android e iOS ma utilizzabile anche su Google Colaboratory.

La rete neurale accetta immagini di dimensioni 224 x 224 pixel.

Questo modello di rete neurale a differenza di quella di AIZOO non ha bisogno dell'utilizzo di funzioni avanzate di computer vision per estrapolare l'output finale permettendo un risparmio delle risorse e del tempo di inferenza.

Per poter utilizzare questa rete neurale correttamente è necessario seguire una serie di step:

- **Ottenere l'immagine del volto delle persone**: per prima cosa dall'immagine contenente una o più persone vanno presi i volti, questi possono essere ottenuti tramite un'altra rete neurale che riconosce e ottiene i volti delle persone dalle immagini oppure può essere utilizzate direttamente l'immagine contenente il volto di una persona;

- **Input della rete**: l'immagine di input data alla rete deve essere ridimensionata in 224 x 224, i pixel vengono convertiti da un range di 0-255 a 0-1 per una migliore elaborazione e infine viene aggiunta una dimensione all'immagine;

- **Predizione**: la predizione viene fatta passando al modello l'immagine ridimensionata precedentemente;

- **Output della rete**: l'ouput contiene due valori, il primo contiene la probabilità che la persona indossi la mascherina mentre il secondo la probabilità che non la indossi. Il valore più alto tra i due indica la predizione fatta dalla rete.

Quindi una volta ottenuto il volto della persona la rete ne calcolerà l'output e il valore più alto tra i due rappresenta quello che la rete ha predetto.

3.3 Efficacia della rete

Dopo aver visto il workflow della rete neurale e la sua struttura è il momento di misurare la sua efficacia. Per questa analisi sono state utilizzate 400 immagini di volti con mascherina (prese da RMFD visto precedentemente) e 400 immagini di volti senza mascherina [18].

Le metriche di questo modello sono risultate le seguenti:

- **Accuratezza**: effettuando le predizioni su queste 800 immagini totali l'accuratezza è risultata del 91% circa;
- **Precision**: la precision relativa alla classe **no mask** è circa il 92% mentre quella della classe **mask** è del 89%. Quindi il modello generalmente è leggermente più preciso a riconoscere le persone senza mascherina;
- **Recall**: la recall per la classe **no mask** è del 89% circa mentre per la classe **mask** è del 93%;
- **F1 score**: in questo caso la f1 score per la classe **no mask** è circa del 90% mentre per la classe **mask** 91%;

Le immagini utilizzate raffigurano volti non sempre frontali alla fotocamera e immagini a volte piccole.

Il modello come già detto è molto efficiente in termini di risorse utilizzate e di tempo di inferenza, infatti per una singola inferenza su Google Colaboratory in Python il modello impiega dai 80ms ai 100ms.

Nota: l'efficienza varia in base al tipo di sistema su cui viene fatto eseguire il modello, su uno smartphone o un computer meno performante il tempo di inferenza del modello sarà maggiore.

3.4 Identificazione dei volti

Il modello nonostante riesca ad avere ottime prestazioni in termini di efficacia ed efficienza necessita in input del volto di una persona, questo comporta l'utilizzo di un'ulteriore rete neurale che sia in grado di trovare le coordinate relative ai volti delle persone e ottenere tutti i volti all'interno dell'immagine.

Qui sotto quel che sarà dell'architettura finale tra le due reti neurali:

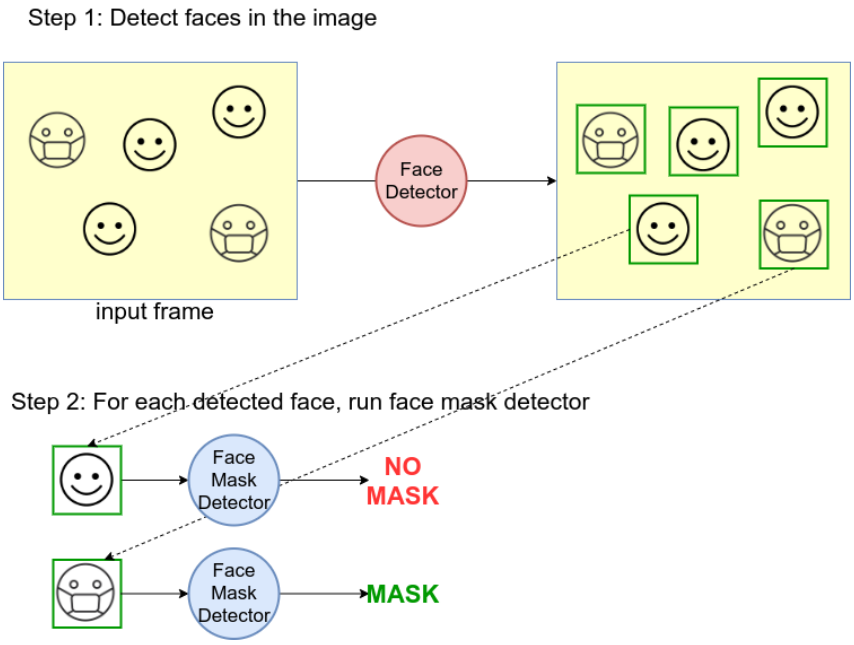


Figura 3.4: Struttura completa del progetto.

La rete neurale utilizzata per l'identificazione dei volti sarà illustrata nel successivo capitolo ed è utilizzabile solo su dispositivi mobile come Android e iOS.

Capitolo 4

Deep learning su Android

In questo capitolo verrà introdotto TensorFlow Lite [19], framework open-source utilizzato principalmente per il machine learning e deep learning su dispositivi mobile come Android e iOS ma utilizzabile anche su microcontrollori o dispositivi IoT. In seguito verrà analizzata la struttura dell'applicazione e il suo funzionamento.

4.1 TensorFlow Lite

TensorFlow Lite è un framework open source utilizzato per il machine learning e il deep learning, può essere utilizzato sia su dispositivi mobile come Android/iOS ma anche su dispositivi IoT.

TensorFlow Lite deriva da TensorFlow utilizzato per la progettazione e lo sviluppo di reti neurali principalmente su computer, il termine **lite** indica che il framework è stato sviluppato per dispositivi meno performanti come i dispositivi mobile.

4.1.1 Componenti

TensorFlow Lite si divide in due principali componenti:

- **Interprete**: il compito principale dell'interprete è quello di eseguire modelli ottimizzati su diversi tipi di Hardware, come smartphone Android e iOS, dispositivi embedded;

- **Convertitore**: il convertitore permette di convertire modelli TensorFlow in una forma più efficiente da utilizzare per l'interprete, può introdurre ottimizzazioni per ridurre le dimensioni e aumentare le performance;

4.1.2 Vantaggi di TensorFlow Lite

TensorFlow Lite è progettato per semplificare l'esecuzione dell'apprendimento automatico sui dispositivi, utilizzando TensorFlow Lite si hanno i seguenti vantaggi:

- **Latenza**: non c'è alcun server esterno da contattare quindi nessuna latenza;
- **Privacy**: non vengono raccolti e inviati dati al di fuori del dispositivo;
- **Connettività**: non è richiesta alcuna connessione ad internet;
- **Consumo energetico**: non c'è un dispendioso consumo energetico dato che non viene utilizzata alcuna connessione ad internet;

4.1.3 Workflow in TensorFlow Lite

Per utilizzare TensorFlow Lite su dispositivi mobile vanno eseguiti i seguenti passi:

- Selezionare un modello TensorFlow o TensorFlow Lite;
- Convertire il modello in un formato TensorFlow Lite;
- Testare il modello sul dispositivo utilizzando l'interprete;
- Ottimizzare il modello se poco efficace o efficiente;

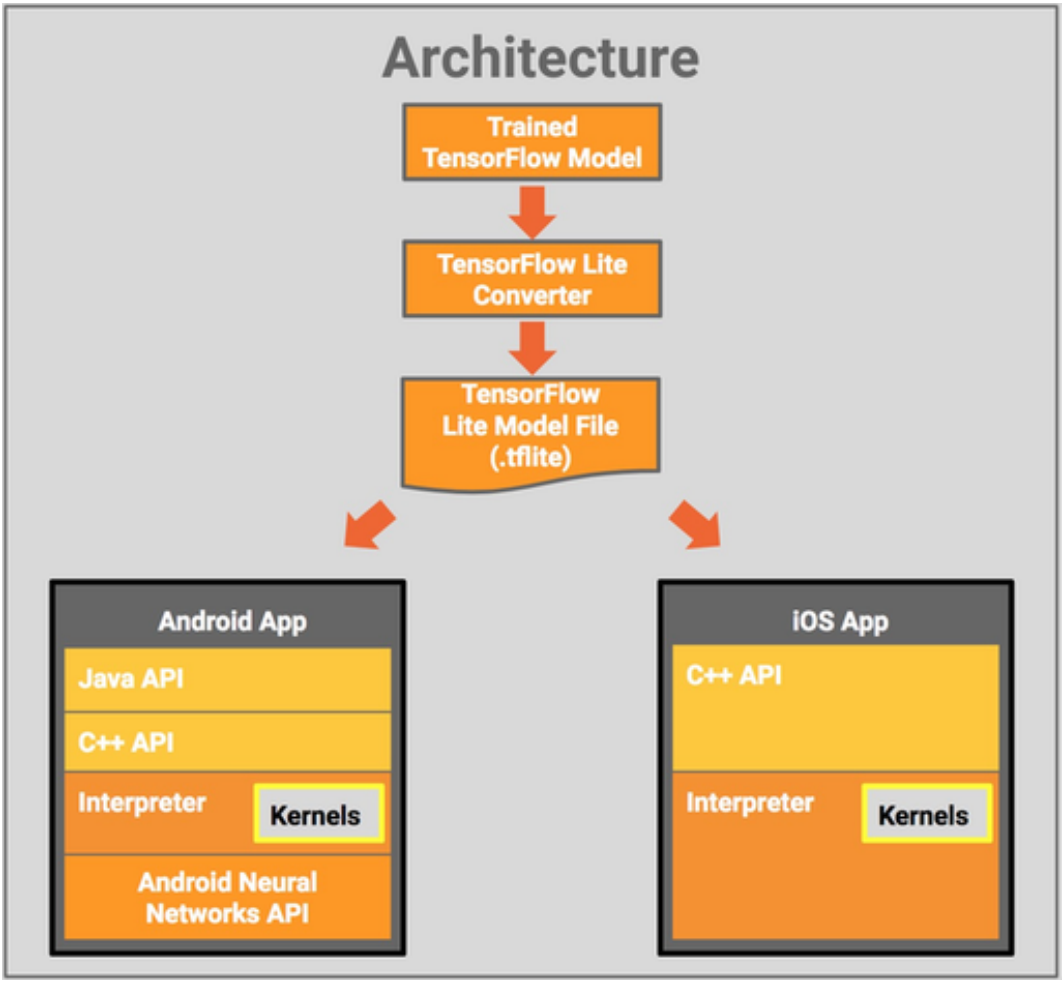


Figura 4.1: Workflow di un modello in TensorFlow Lite.

La rete neurale utilizzata per il rilevamento delle mascherine è già convertita in TensorFlow Lite e ottimizzata.

4.2 Applicazione Android

Ora che è stato introdotto brevemente TensorFlow Lite si può introdurre e analizzare l'applicazione finale, la sua struttura e il suo funzionamento.

Inizialmente è stata sviluppata una piccola applicazione da zero, ma sono stati riscontrati diversi problemi dovuti soprattutto alla grandezza dell'applicazione e all'utilizzo della fotocamera in tempo reale, per questo motivo la struttura principale dell'applicazione è stata presa in gran parte da uno dei

principali esempi di TensorFlow Lite per la **classificazione di oggetti** [20]. Il codice dell'applicazione è stato poi modificato in Java su Android Studio.

L'applicazione consente di rilevare e contornare i volti delle persone e indicare se indossano una mascherina medica, le persone che indossano una mascherina vengono contornate di verde mentre quelle che non indossano alcuna mascherina vengono contornate di rosso.

Il tutto viene eseguito in tempo reale, quindi è sufficiente utilizzare la fotocamera frontale o posteriore del proprio telefono e inquadrare una o più persone.

L'applicazione consente inoltre di ottenere alcune informazioni come: il numero di persone attualmente rilevate, quante indossano la mascherina e quante non e il tempo di inferenza.

Verrà analizzata infine l'applicazione utilizzando anche la rete neurale di AIZOO evidenziando le differenze trovate.

4.2.1 Google ML Kit

Come accennato già di recente l'applicazione per funzionare correttamente necessita di una rete neurale per l'identificazione dei volti, per i dispositivi Android e iOS esiste un SDK (Software Development Kit) chiamato **Google ML Kit** [21] in grado di eseguire diversi compiti di machine learning e deep learning, in questo caso l'identificazione e estrapolazione dei volti.

ML Kit può essere implementato facilmente nella propria applicazione configurandolo in base alle necessità senza l'utilizzo di connessione ad internet per funzionare.

Nel caso dell'identificazione dei volti, ML Kit non solo permette di identificare i volti delle persone ma permette anche di ottenere le coordinate relative a viso, occhi, naso, bocca ecc...

4.2.2 Funzionamento dell'applicazione

Installata l'applicazione, al primo avvio verrà chiesto di ottenere i permessi per poter utilizzare la fotocamera del proprio cellulare.

Il funzionamento dell'applicazione è molto semplice, è sufficiente aprire l'applicazione e inquadrare tramite la fotocamera frontale o posteriore del telefono delle persone inquadrandone i volti.

Possono essere inquadrate più persone contemporaneamente sia che siano ferme che in movimento. Di tutte le persone rilevate ne viene contornato il viso di verde in caso di mascherina indossata o rosso in caso la persona non indossi la mascherina.

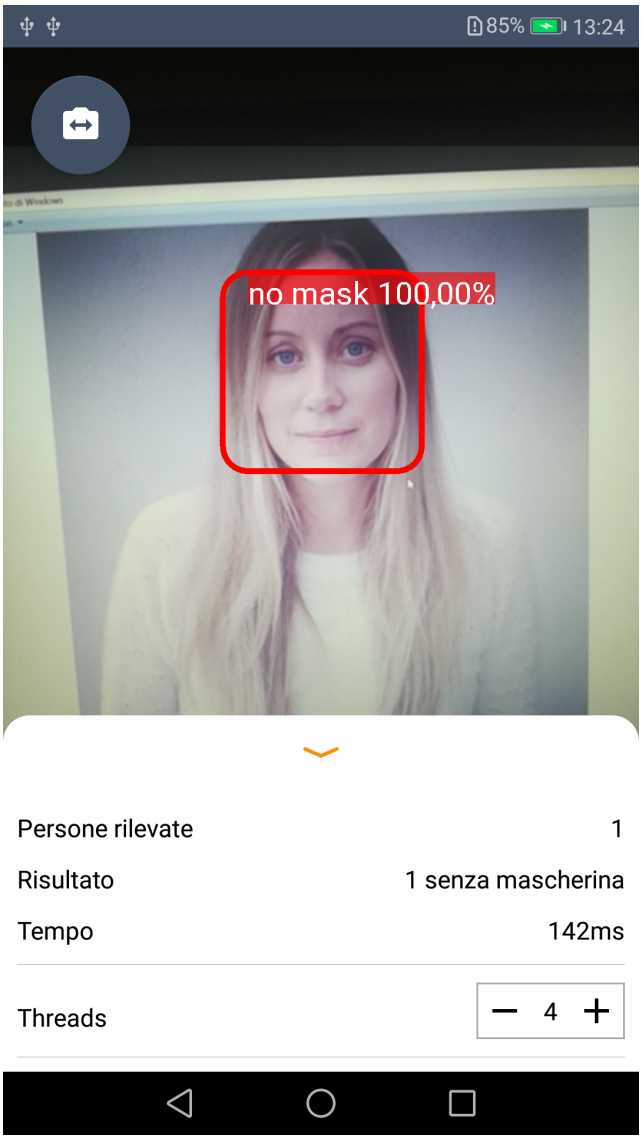


Figura 4.2: Persona rilevata senza mascherina.

Come si può vedere dall'immagine sopra il viso della ragazza viene contornato di rosso indicando che non indossa la mascherina al 100%.

Nella parte inferiore dell'applicazione viene indicato: il numero di persone rilevate, quante di queste indossano la mascherina o meno e infine viene indicato il tempo totale di inferenza (varia a seconda del tipo di dispositivo e dell'immagine).

Nella voce **threads** è possibile aumentare o diminuire i threads per l'interprete di TensorFlow Lite. I thread di default vengono impostati in base alle

caratteristiche del telefono, il numero di thread da impostare dipende dal tipo di cellulare che si sta utilizzando.

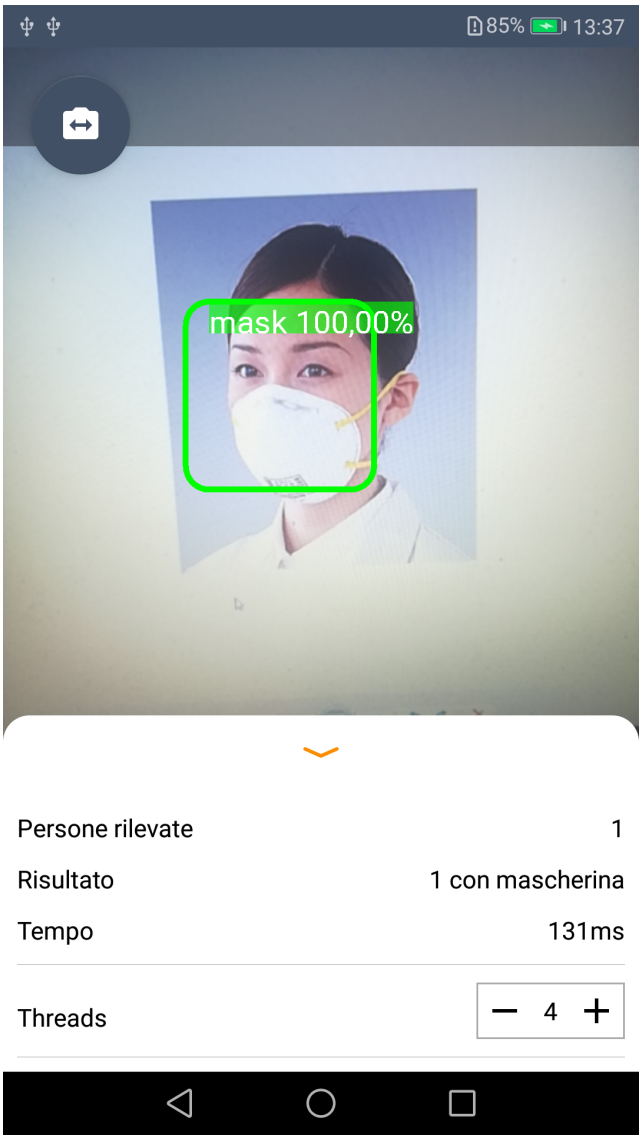


Figura 4.3: Persona rilevata con mascherina.

Inoltre l'applicazione è in grado di rilevare anche più persone contemporaneamente come mostra la figura sotto.

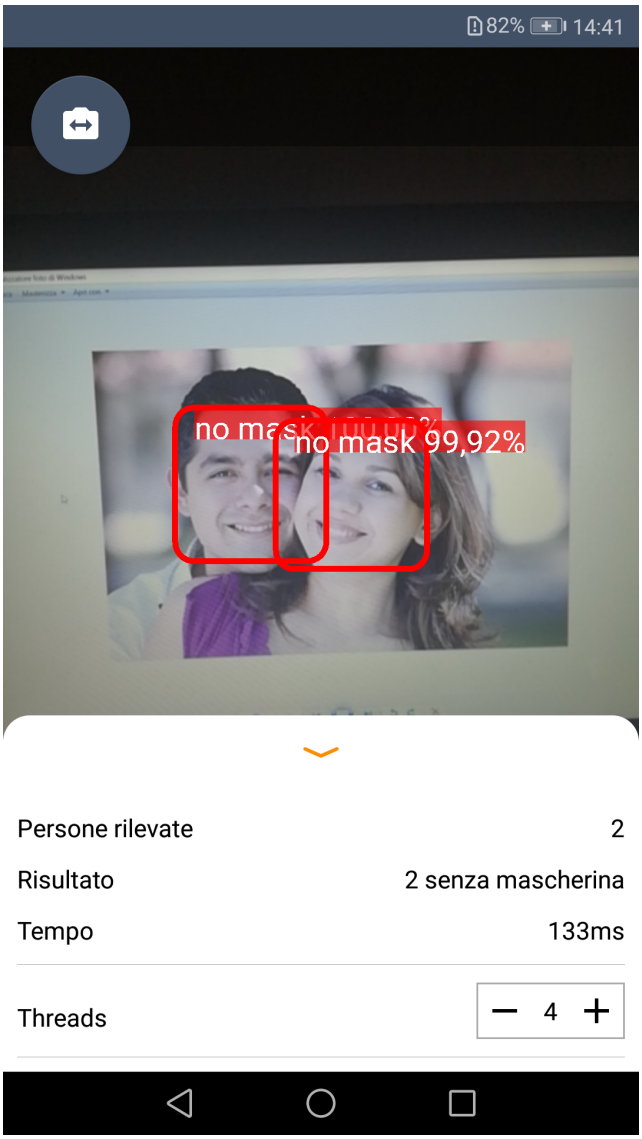


Figura 4.4: Due persone rilevate senza mascherina.

Non sempre l'applicazione riesce a rilevare tutti i volti delle persone presenti, in particolare quando: le persone sono abbastanza lontane dall'inquadratura, sono girate quasi completamente o il volto non è ben visibile.

4.2.3 Workflow dell'applicazione

L'applicazione Android sviluppata consiste in queste fasi principali:

- La prima fase si occupa di ottenere un frame cioè un'immagine dalla fotocamera del cellulare, il frame viene preso ogni volta che è disponibile

ovvero quando l'immagine precedentemente presa è già stata elaborata dalle due reti neurali. Il frame viene preso con una dimensione prefissata di **640 x 480**;

- Il frame viene passato alla rete neurale di Google ML Kit la quale rileva i volti delle persone (se presenti);
- Ogni volto identificato e ottenuto dalla rete viene passato alla seconda rete neurale che rileva se il volto indossa la mascherina o meno;
- Infine vengono contornati i volti delle persone secondo l'output;

Queste fasi vengono ripetute durante tutto il funzionamento dell'applicazione.

4.2.4 Prestazioni dell'applicazione

L'applicazione Android pur utilizzando due reti neurali non richiede molte risorse nel suo utilizzo né tanto meno lunghi tempi di calcolo.

Il dispositivo che è stato utilizzato per testare l'applicazione è un Huawei P9 Lite del 2017 con 3GB di RAM e un processore octa core.

Per visualizzare le risorse utilizzate dall'applicazione è stato utilizzato un profiler disponibile in Android Studio che permette di vedere quanta memoria RAM, processore e rete vengono utilizzati durante l'esecuzione. Per la RAM vengono utilizzati circa 100MB al massimo mentre il processore viene utilizzato al 30% circa, non vi è alcun traffico di rete dato che l'applicazione non necessita dell'utilizzo di internet.

Per una singola inferenza sul frame l'applicazione impiega sul dispositivo descritto poco più di 100 millisecondi.

4.2.5 Rete di AIZOO utilizzata nell'applicazione

Prima di utilizzare la soluzione finale è stata utilizzata la rete neurale di AIZOO, come detto precedentemente però la rete presenta dei limiti che sono stati riscontrati principalmente sull'applicazione Android. Utilizzando infatti questa rete neurale nell'applicazione è stato riscontrato un aumento del tempo di inferenza dovuto all'implementazione di funzioni di computer vision che richiedono diversi calcoli sull'output per ogni frame. L'applicazione che utilizza la rete neurale di AIZOO non necessita di ML Kit per l'identificazione dei volti.

Inoltre questa rete utilizzata dall'applicazione non permette spesso di identificare e rilevare correttamente i volti delle persone, in particolare quando, sono vicine tra loro, in lontananza e di profilo.

Qui sotto un esempio utilizzando la rete di AIZOO:

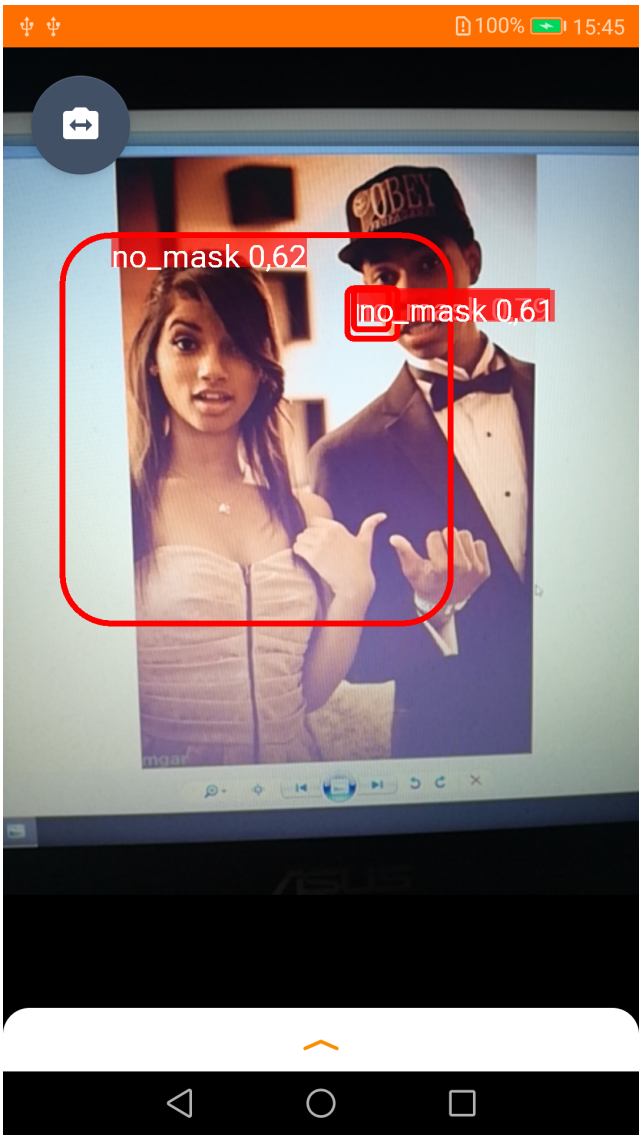


Figura 4.5: Rete di AIZOO utilizzata nell'applicazione.

Come si può ben vedere la rete fatica a trovare correttamente le coordinate relative ai volti delle due persone.

Capitolo 5

Codice prodotto

In questo quinto e ultimo capitolo vengono mostrate le principali porzioni di codice dell'elaborato di tesi. Il codice è stato scritto sia su Google Colaboratory utilizzando il linguaggio di programmazione Python e sia su Android Studio per lo sviluppo dell'applicazione utilizzando Java.

5.1 Codice su Google Colaboratory

In questa sezione vengono mostrate porzioni di codice sull'utilizzo della rete neurale per il riconoscimento delle mascherine in Python sulla piattaforma Google Colaboratory.

Per prima cosa vanno caricate le librerie necessarie.

```
''' tensorflow: utilizzata per l'interprete TensorFlow Lite
    numpy: utilizzata per la gestione di vettori e matrici
    cv2: utilizzata per la lettura e elaborazione delle immagini
'''
import tensorflow as tf
import numpy as np
import cv2
```

In seguito viene caricato il modello utilizzando l'interprete di TensorFlow Lite e allocati i tensori (array multidimensionali):

```
interpreter = tf.lite.Interpreter(model_path="drive/MyDrive
/Tesi_file_new/mask_detector.tflite") #Viene caricata la rete
    neurale in formato TensorFlow Lite
interpreter.allocate_tensors() #Allocazione dei tensori

''' Tensori di input ed output della rete '''
```

```
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

Funzione per il preprocessing dell'immagine:

```
''' Indica la dimensione dell'input della rete neurale '''
target_shape = (224, 224)

''' Funzione utilizzata per il preprocessing dell'immagine prima di
    essere passata in input alla rete neurale '''

def preprocess_image(path_img):
    ''' La funzione:
        - Legge l'immagine passata in input
        - Ridimensiona l'immagine in 224 x 224 pixel
        - Converte i pixel da un range di 0-255 a 0-1 '''

    img = cv2.imread(path_img)
    image_resized = cv2.resize(img, target_shape)
    image_np = image_resized / 255.0
    return np.expand_dims(image_np, axis=0)
```

Una volta che è stato caricato l'interprete e ridimensionata l'immagine viene fatta l'inferenza sull'immagine da parte del modello.

```
''' Vettore utilizzato per inserire la predizione da parte della
    rete '''
predicted = []

''' I pixel dell'immagine vengono convertiti in float a 32 bit '''
input_data = np.array(image, dtype=np.float32)

''' All'interprete viene impostato il tensore di input passando
    l'immagine come input '''
interpreter.set_tensor(input_details[0]["index"], input_data)

''' Viene fatta la predizione sull'immagine '''
interpreter.invoke()

''' Output della rete '''
output = interpreter.get_tensor(output_details[0]["index"])

''' Probabilita' che la persona indossi la mascherina e che non la
    indossi '''
```

```
mask = output[0][0]
no_mask = output[0][1]

''' Il valore piu' alto tra i due corrisponde all'output della rete
    '''
if mask > no_mask:
    predicted.append(0)
    print("Probabilita' che indossi la mascherina: " + str(mask))
else:
    predicted.append(1)
    #print("Probabilita' che non indossi la mascherina: " +
        str(no_mask))
```

5.2 Codice Android

In questa sezione vengono mostrate sezioni di codice Java scritte su Android Studio per la parte di sviluppo dell'applicazione.

Per l'identificazione dei volti utilizzando la rete di ML Kit il codice utilizzato è il seguente:

```
    InputImage image = InputImage.fromBitmap(croppedBitmap, 0);
    faceDetector
            .process(image) //L'immagine viene passata in input alla
               rete
            .addOnSuccessListener(new OnSuccessListener<List<Face>>()
               {
              @Override
              public void onSuccess(List<Face> faces) {
                /* Se sono stati trovati dei volti viene chiamata una
                    funzione che utilizzera' poi la seconda rete
                    neurale */
                if (faces.size() == 0) {
                  updateResults(currTimestamp, new LinkedList<>());
                  return;
                }
                runInBackground(
                        new Runnable() {
                          @Override
                          public void run() {
                            onFacesDetected(currTimestamp, faces);
                          }
                        });
```

```
            }
          });
```

Per prima cosa l'immagine ridimensionata viene passata al modello di ML Kit, se il modello ha rilevato dei volti, per ogni volto trovato viene fatta la predizione sull'immagine del volto nel seguente modo:

```
/* Output della rete */
Map<Integer, Object> outputMap = new HashMap<>();
output = new float[1][2];
outputMap.put(0, output);

Trace.beginSection("run");
/* Predizione del modello */
tfLite.runForMultipleInputsOutputs(inputArray, outputMap);
Trace.endSection();

/* Probabilita' che la persona indossi la mascherina e che non la
    indossi */
float mask = output[0][0];
float no_mask = output[0][1];

float confidence;
String id;
String label;

/* Il valore piu' alto tra i due corrisponde all'output della rete */
if (mask > no_mask) {
    label = "mask";
    confidence = mask;
    id = "0";
} else {
    label = "no mask";
    confidence = no_mask;
    id = "1";
}
```

Il codice utilizzato sopra è più articolato ma sostanzialmente uguale a quello mostrato in Python.

Conclusioni

Arrivato alla fine di questo elaborato di tesi che mi ha impegnato per diversi mesi dallo studio approfondito della disciplina sino ad oggi posso reputarmi soddisfatto del lavoro fatto, ci sono state delle difficoltà che sono state risolte grazie al relatore e a soluzioni già esistenti, l'obiettivo finale è stato raggiunto con successo e l'applicazione potrebbe essere utilizzata in diversi ambiti.

Ringraziamenti

Il primo ringraziamento va al relatore di questo elaborato di tesi, il Prof. Gianluca Moro, che ha reso possibile tutto ciò e ha stimolato il mio personale interesse nei confronti di questa disciplina.

Grazie alla mia famiglia che mi ha supportato in questi anni e a tutte le persone che hanno creduto in me.

Bibliografia

[1] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[2] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[6] X-zhangyang. Real-world-masked-face-dataset.

[7] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1800–1807. IEEE Computer Society, 2017.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[10] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition,*, 2016.

[11] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

[12] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy: Fixefficientnet. *CoRR*, abs/2003.08237, 2020.

[13] AIZOOTech. Facemaskdetection.

[14] estebanuri. facemaskdetector.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, June 2018.

[16] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. cite arxiv:1704.04861.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[18] kaggle. Dataset real and fake face detection.

[19] TensorFlow. Tensorflow lite.

[20] tensorflow lite. object detection android.

[21] google. ml kit.