

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA  
CAMPUS DI CESENA

---

Scuola di Scienze  
Corso di Laurea in Ingegneria e Scienze Informatiche

**STUDIO E Sperimentazione di Soluzioni allo Stato  
dell'Arte per il Rilevamento di Mascherina  
Sanitaria Facciale con Micro-Controllori**

*Elaborato in*  
Programmazione Di Applicazioni Data Intensive

*Relatore*  
Prof. Gianluca Moro

*Presentata da*  
Alessandro Marcantoni

---

Terza Sessione di Laurea  
Anno Accademico 2019 – 2020



# **PAROLE CHIAVE**

Machine Learning

Deep Learning

Convolutional Neural Network

Micro-Controllore

Python



*A chiunque mi sia stato vicino,  
e mi abbia aiutato a raggiungere questo traguardo.*



# Introduzione

L'intelligenza artificiale (AI) è una disciplina vastissima che trova origine nel 1950 quando Alan Turing ideò il famoso test (che prese successivamente il suo nome) per stabilire se una macchina fosse intelligente.

Nell'ambito del settore informatico, potremmo identificare l'AI - Artificial Intelligence - come la disciplina che si occupa di realizzare macchina (hardware e software) in grado di "agire" autonomamente (risolvere problemi, compiere azioni, ecc.).

Recentemente è diventata uno dei principali trend tecnologici strategici grazie ai progressi nella ricerca scientifica del settore ed all'enorme sviluppo della capacità di calcolo dei sistemi hardware. Invero, solo negli ultimi 10 anni i risultati della ricerca scientifica in questo campo hanno di gran lunga superato le conoscenze raggiunte in precedenza e si sono tradotti in tecnologie software di pubblico dominio. Ciò ne ha comportato l'applicazione in quasi ogni ambito delle discipline scientifiche.

I metodi di AI più recenti si distinguono da quelli classici precedenti in quanto basati sull'estrazione di conoscenza da masse di dati: maggiore è la disponibilità di dati, maggiore risulterà l'accuratezza della conoscenza estratta. L'introduzione di dispositivi elettronici e sensori in diversi ambienti accresce le dimensioni della massa di dati e di conseguenza cresce la necessità di ricercare nuovi metodi e tecniche per la loro gestione ed elaborazione.

Macchine intelligenti che "apprendono" e migliorano le proprie funzionalità: è questo il concetto alla base del **machine learning**, macro area di tecniche oggi molto popolari i cui recenti successi sono trasversali a numerosi settori (tra cui speech recognition e visione artificiale).

Il successo di queste tecniche è dovuto alla facilità che dimostrano nella gestione di ingenti quantità di dati, fondando la loro potenza su grandi insiemi di informazioni. Si stanno diffondendo molto velocemente in molti campi in quanto i modelli di sviluppo tradizionali non riescono più a garantire risultati veloci, precisi ed accurati.

L'utilizzo del machine learning presenta però dei limiti e può portare a risultati non soddisfacenti quanto si devono gestire immagini o testi in forma naturale poiché l'attività di preparazione dei dati e di selezione delle variabili

decisive per l'obiettivo finale richiede l'intervento di esperti e risulta dispendiosa. In questi casi si ottengono risultati generalmente più accurati e precisi con 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 è una tecnica relativamente giovane, le cui potenzialità sono ancora tutte da esplorare in grado di elaborare in maniera ancora più approfondita i dati e sarà affrontata con maggiore dettaglio successivamente all'interno del presente elaborato.

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

All'interno di questo elaborato verranno analizzati e sperimentati diversi approcci recenti in ambito *visione artificiale* (CV) e deep learning (DL), allo scopo di identificare persone all'interno di immagini e video e stabilire se queste indossano la mascherina oppure no.

Il lavoro di tesi è stato suddiviso nei seguenti capitoli:

- **Capitolo 1** - Analisi preliminare del problema;
- **Capitolo 2** - Panoramica sulle varie tecnologie esistenti in letteratura utilizzabili per il problema posto;
- **Capitolo 3** - Introduzione all'ambiente di lavoro e agli strumenti utilizzati;
- **Capitolo 4** - Descrizione delle soluzioni intermedie sperimentate;
- **Capitolo 5** - Descrizione della soluzione definitiva individuata;
- **Capitolo 6** - Descrizione dell'integrazione della soluzione;
- **Capitolo 7** - Il lavoro realizzato e il codice impiegato.

# Indice

<b>1</b>	<b>Analisi preliminare</b>	<b>1</b>
1.1	Contesto applicativo . . . . .	1
1.2	Immagini . . . . .	2
1.3	Dataset utilizzati . . . . .	3
1.3.1	RMFD . . . . .	3
1.3.2	FaceMaskDataset . . . . .	3
1.3.3	HardDataset . . . . .	4
1.4	Obiettivi . . . . .	4
<b>2</b>	<b>Le tecnologie disponibili</b>	<b>7</b>
2.1	Machine Learning . . . . .	7
2.1.1	Sviluppo di un modello . . . . .	9
2.1.2	Apprendimento supervisionato . . . . .	9
2.1.3	Apprendimento non supervisionato . . . . .	10
2.1.4	Apprendimento per rinforzo . . . . .	11
2.1.5	Discesa del gradiente . . . . .	12
2.1.6	Validazione del modello . . . . .	13
2.2	Deep Learning . . . . .	14
2.2.1	Struttura . . . . .	15
2.2.2	Funzione di attivazione . . . . .	17
2.2.3	Addestramento . . . . .	19
2.3	Visione Artificiale . . . . .	19
2.3.1	Reti Neurali Convoluzionali . . . . .	20
<b>3</b>	<b>Ambiente di lavoro e strumenti utilizzati</b>	<b>25</b>
3.1	Micro-controllori . . . . .	25
3.1.1	Nvidia Jetson Nano Developer Kit . . . . .	26
3.1.2	Raspberry Pi 4 Model B . . . . .	27
3.2	Python . . . . .	28
3.2.1	Librerie Python . . . . .	29
3.3	Google Colaboratory . . . . .	29

<b>4 Panoramica delle soluzioni testate</b>	<b>31</b>
4.1 Addestrare una nuova rete . . . . .	31
4.1.1 Transfer Learning . . . . .	31
4.1.2 Xception . . . . .	32
4.1.3 Data Augmentation . . . . .	33
4.1.4 Fase di addestramento . . . . .	34
4.1.5 Risultati ottenuti . . . . .	35
4.2 Modello di AIZOOTech . . . . .	35
4.2.1 Architettura della rete . . . . .	36
4.2.2 Funzionamento della soluzione . . . . .	37
4.2.3 Efficacia della soluzione . . . . .	37
4.2.4 Limiti della soluzione . . . . .	39
4.2.5 Possibili soluzioni . . . . .	39
<b>5 Analisi della soluzione definitiva</b>	<b>41</b>
5.1 Funzionamento della soluzione . . . . .	41
5.2 Rilevamento dei volti . . . . .	42
5.3 Classificazione dei volti . . . . .	43
5.4 Efficacia della soluzione . . . . .	44
5.4.1 Efficacia per numero di persone . . . . .	45
<b>6 Deep Learning su micro-controllori</b>	<b>49</b>
6.1 Tensorflow Lite . . . . .	49
6.1.1 Vantaggi di Tensorflow Lite . . . . .	49
6.1.2 Funzionamento di Tensorflow Lite . . . . .	50
6.2 Descrizione dell'applicazione . . . . .	50
6.2.1 Funzionamento dell'applicazione . . . . .	51
6.3 Test su micro-controllori . . . . .	51
6.3.1 Raspberry Pi 4 Model B 8GB . . . . .	51
6.3.2 Nvidia Jetson Nano . . . . .	52
6.3.3 Prestazioni . . . . .	53
6.3.4 Nvidia Face Mask Detection . . . . .	55
<b>7 Il codice prodotto</b>	<b>57</b>
7.1 Il codice della soluzione . . . . .	57
<b>Conclusioni e sviluppi futuri</b>	<b>61</b>
<b>Ringraziamenti</b>	<b>63</b>
<b>Bibliografia</b>	<b>65</b>

# Elenco delle figure

1.1	Come un computer vede le immagini . . . . .	2
1.2	Esempi di immagini contenute nel dataset <i>RMFD</i> . . . . .	3
1.3	Esempio di immagine contenuta nel dataset <i>FaceMaskDataset</i> . .	4
1.4	Esempio di immagine contenuta nel dataset <i>HardDataset</i> . . . . .	4
1.5	Esempio di soluzione da ricercare. . . . .	5
2.1	Esempio di un sistema di suggerimenti guidato dal machine learning. . . . .	8
2.2	Rappresentazione grafica di un problema di clustering. . . . .	11
2.3	Modello di base nell'apprendimento di rinforzo. . . . .	11
2.4	Discesa del gradiente dal punto A al punto B visualizzabile graficamente. . . . .	12
2.5	Esempi di <i>Underfitting</i> , <i>Apprendimento bilanciato</i> e <i>Overfitting</i> . .	13
2.6	Gerarchia delle tecnologie di intelligenza artificiale. . . . .	14
2.7	Diverse astrazioni trattate nelle reti neurali. . . . .	15
2.8	Struttura tipica di una rete neurale. . . . .	16
2.9	Struttura di un neurone. . . . .	16
2.10	Grafico della funzione sigmoide . . . . .	17
2.11	Grafico della funzione ReLU . . . . .	18
2.12	Grafico della funzione Softplus . . . . .	18
2.13	Campo recettivo di un neurone in una rete neurale semplice (sinistra) e in una rete neurale convoluzionale (destra). . . . .	20
2.14	Strati convoluzionali con campi recettivi locali rettangolari. . . .	21
2.15	Esempio di applicazione di filtri diversi e generazione delle mappe delle caratteristiche corrispondenti. . . . .	22
2.16	Rappresentazione in 3D di uno strato convoluzionale. . . . .	22
2.17	Rappresentazione di uno strato MaxPooling. . . . .	23
2.18	Architettura tipica di una rete neurale convoluzionale . . . . .	24
3.1	Schema Nvidia Jetson Nano Developer Kit . . . . .	27
3.2	Schema Raspberry Pi 4 Model B . . . . .	28
4.1	Architettura della rete Xception. . . . .	32

4.2	Differenza tra una normale rete (sinistra) e una rete con connessione residua (destra). . . . .	33
4.3	Rappresentazione di uno strato convoluzionale separabile. . . . .	33
4.4	Esempio di data augmentation. . . . .	34
4.5	Dimostrazione del funzionamento della rete AIZOO. . . . .	35
4.6	Architettura della rete AIZOO. . . . .	36
4.7	Applicazione della <i>non-maximum suppression</i> . . . . .	37
5.1	Struttura della soluzione finale. . . . .	42
5.2	Architettura della rete Dual Shot Face Detector. . . . .	42
5.3	Singolo blocco di strati della rete MobileNet. . . . .	43
6.1	Funzionamento del framework Tensorflow Lite. . . . .	50

# Capitolo 1

## Analisi preliminare

In questo capitolo vengono introdotti il contesto applicativo e i dati utilizzati durante il lavoro. Viene inoltre illustrato l'obiettivo del presente elaborato di tesi.

### 1.1 Contesto applicativo

Nel corso delle prime settimane di gennaio 2020, alcuni scienziati hanno riscontrato polmoniti anomale in alcuni lavoratori del mercato umido di Wuhan, la cui causa si è rivelata essere un nuovo *coronavirus*, identificato come *SARS-CoV-2*.

Alla fine del gennaio 2020, non si era ancora riusciti ad accettare le caratteristiche del virus, tanto da nutrire incertezze sulle esatte modalità di trasmissione e sulla patogenicità (la capacità di creare danno), sebbene si fosse riscontrata la facilità di trasmissione da persona a persona. La malattia associata è stata poi riconosciuta con il nome di **COVID-19**.

I pazienti accusavano sintomi simili all'influenza come dermatiti, febbre, tosse secca, stanchezza, difficoltà di respiro. Nei casi più gravi, spesso riscontrati in soggetti già gravati da precedenti patologie, si sviluppavano polmoniti ed insufficienza renale acuta talvolta così gravi da condurre al decesso.

Una volta individuate le principali modalità di diffusione del SARS-CoV-2 nelle goccioline di saliva derivanti da starnuti e tosse e nei contatti diretti personali e contatti tra le mani contaminate e le mucose, l'Organizzazione Mondiale della Sanità (OMS) emanava raccomandazioni dirette a limitare la diffusione del virus. Le linee guida prevedono anche oggi il lavaggio regolare delle mani per più di 20 secondi, la copertura di bocca e naso in caso di tosse o starnuti ed il distanziamento da chiunque mostri sintomi di malattie respiratorie (come appunto tosse e starnuti).

L'aumento esponenziale dei malati, con conseguente sovraffollamento delle strutture ospedaliere, ha indotto numerosi paesi in tutto il mondo, allo scopo di diminuire il più possibile i contatti interpersonali, a ricorrere a periodi di "lockdown" totale.

Altra misura adottata per contenere il numero di contagi è l'utilizzo delle mascherine. Da alcune ricerche [1] è emerso invero che le mascherine chirurgiche possono ridurre efficacemente l'emissione di particelle virali nell'ambiente e, di conseguenza, diminuire il rischio di contagio. Non sorprende quindi il fatto che, in vari stati tra cui l'Italia, l'utilizzo delle mascherine sia stato reso obbligatorio ovunque, tranne rare eccezioni. L'inadempimento a tale normativa, così come l'inesatto adempimento (vedi il non corretto utilizzo) comporta l'applicazione di sanzioni pecuniarie. Sarà sempre più importante, anche per il momento in cui stiamo vivendo, accertare l'utilizzo da parte delle persone e/o il corretto utilizzo di mascherine.

L'obiettivo principale di questa tesi è quello di trovare un metodo in grado di stabilire se un soggetto inquadrato da una fotocamera indossi correttamente la mascherina, con lo stesso grado di certezza con cui lo fa una persona.

## 1.2 Immagini

Le immagini, su cui opererà la soluzione, sono un tipo di dato destrutturato. Ciò significa che sono sprovviste di un modello in grado di spiegarne la semantica: aspetto che rende il processo di estrazione dell'informazione molto più complesso rispetto ai dati strutturati. Basti pensare che, di fatto, le immagini sono costituite da griglie (tre sovrapposte nel caso di foto a colori codificate in formato *RGB* o una soltanto per foto in bianco e nero) contenenti valori numerici (solitamente tra 0 e 255). Risulta perciò molto difficile per la macchina interpretare il loro contenuto e apprendere da esse.

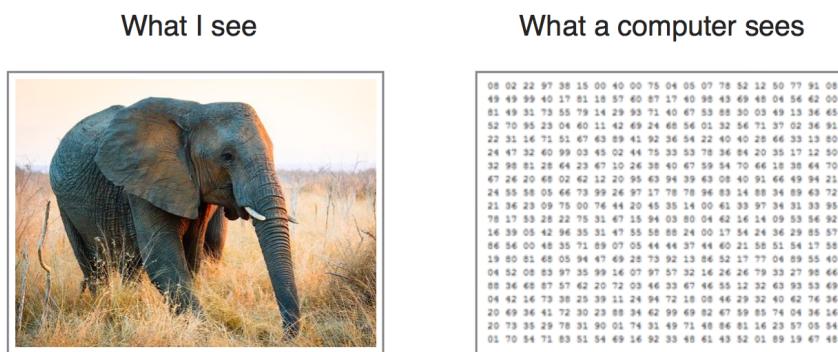


Figura 1.1: Come un computer vede le immagini.

## 1.3 Dataset utilizzati

È possibile reperire dal web un numero considerevole di immagini contenenti persone munite di mascherina facciale e, ovviamente, di altre che non lo sono. È particolarmente utile raccogliere molte immagini (esempi) in modo da favorire il processo di addestramento. Inoltre, è opportuno che il dataset risulti bilanciato nelle sue classi: il numero di immagini di persone con mascherina dovrebbe equivalere quello di immagini di persone senza mascherina. Questi accorgimenti permettono al modello di apprendere in modo più efficace. In particolare, sono stati utilizzati diversi dataset per raggiungere l'obiettivo prefissato.

### 1.3.1 RMFD

Questo dataset [2] contiene 95000 immagini, di cui 90000 raffigurano volti di persone senza mascherina e 5000 con mascherina. Le immagini non sono etichettate, perciò è stato necessario automatizzare il processo di assegnazione delle etichette.



Figura 1.2: Esempi di immagini contenute nel dataset *RMFD*.

### 1.3.2 FaceMaskDataset

Sono qui contenute circa 8000 immagini già etichettate di persone con e senza mascherina. Queste sono divise in *training set* e *validation set*, che corrispondono rispettivamente al 77% e al 23% del totale. L'aspetto interessante di questo dataset è che contiene immagini raffiguranti persone per intero (talvolta anche più di una) e non solamente il volto, perciò può essere un ottimo modo per testare la soluzione completa [3].



Figura 1.3: Esempio di immagine contenuta nel dataset *FaceMaskDataset*.

### 1.3.3 HardDataset

Il seguente è invece un dataset creato opportunamente per testare la rete in situazioni "difficili". Le circa 100 immagini al suo interno contengono infatti persone lontane dall'inquadratura o girate di profilo.



Figura 1.4: Esempio di immagine contenuta nel dataset *HardDataset*.

## 1.4 Obiettivi

L'obiettivo principale di questa tesi, come già sottolineato, sarà sviluppare un sistema capace di etichettare le persone inquadrate all'interno di un'immagine o un video in modo da stabilire se esse indossano la mascherina sanitaria facciale o meno.

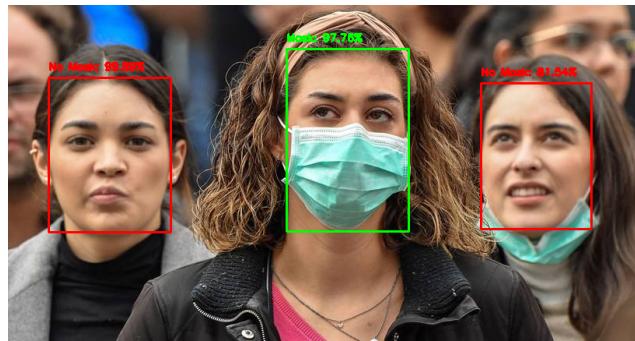


Figura 1.5: Esempio di soluzione da ricercare.

Verranno considerati e presi in considerazione diversi approcci esistenti al problema, usando metodi di **deep learning** differenti, per testare quale si adatti meglio al problema in causa.

Inoltre, la soluzione dovrà essere pensata per essere sfruttata da micro-controllori, i quali hanno risorse limitate.



# Capitolo 2

## Le tecnologie disponibili

In questa parte verranno descritte più nello specifico tutte le tecnologie delle quali si è già accennato nell'introduzione, per rendere più chiari gli obiettivi e le metodologie di lavoro impiegate di cui ai capitoli successivi.

### 2.1 Machine Learning

Il machine learning è una branca dell'intelligenza artificiale sviluppatasi recentemente (negli ultimi decenni del 1900 circa) che fornisce un nuovo approccio ai problemi il cui obiettivo è stimare, predire o ipotizzare "*qualcosa*".

Una delle definizioni più citate quando si parla di questa tecnologia, è quella proposta dal professor Tom Mitchell [4]:

*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  $E$ .*

Cioè, riuscire a far eseguire un'attività a una macchina, che riesce in maniera autonoma a migliorare il modo in cui la esegue, usando l'esperienza che acquisisce nel farlo. La macchina quindi *impara ad eseguire un compito* in maniera sempre più accurata e precisa, senza essere stata programmata in maniera specifica a farlo.

Questo approccio rappresenta una completa innovazione nel modo in cui gli algoritmi vengono utilizzati per risolvere problemi, in quanto generalmente l'approccio classico ne prevedeva lo sviluppo sfruttando la conoscenza plessa dell'autore o autori, mentre al contrario in questo modo la macchina impara da esempi e dati.

La circostanza che non esista un unico campo dove poterne sfruttare le potenzialità, consente di adattarsi benissimo ad un insieme molto vasto di problematiche. Per citarne qualcuna:

- Predizione di consumi;
- Categorizzazione di elementi;
- Previsione andamento meteorologico;
- Stima di costi.

E non solo. Al giorno d'oggi, infatti, questa tecnologia è impiegata negli ambiti e nei settori più disparati: sanità, sicurezza, management aziendale e previsione di fenomeni o eventi di carattere generale. Anche gli smartphone si affidano all'intelligenza artificiale per portare a termine i compiti più disparati (la capacità di predizione del testo durante la digitazione o le fotocamere in grado di applicare filtri personalizzati a ogni immagine).

Il progredire di questi metodi è stato sostenuto anche dalla crescente disponibilità di dati da analizzare che, come già accennato in precedenza, negli anni si sono letteralmente moltiplicati. Unitamente all'aumento della disponibilità di risorse computazionali si sono create le condizioni ideali per lo sviluppo di nuovi studi e ricerche.

Per fare un esempio di quanto questa tecnologia abbia avuto impieghi in casi del tutto reali e concreti, si prendano i **suggerimenti di prodotti simili** a quelli che si stanno visualizzando (ad esempio durante una sessione di shopping online). Non sono altro che dei sistemi che utilizzano il machine learning per trovare legami tra prodotti in maniera automatica, per poi successivamente consigliarne l'acquisto ai visitatori.



Figura 2.1: Esempio di un sistema di suggerimenti guidato dal machine learning.

Sono sistemi che ormai sono entrati nella quotidianità di tutti, e che silenziosamente garantiscono un'esperienza d'uso più gradevole e in grado di invogliare a passare più tempo sul sito o l'applicazione.

E per farsi un’idea di quanto siano importanti per le aziende questi meccanismi, basti pensare che nel caso di Amazon il 35% delle vendite viene generato proprio dai suggerimenti visualizzati all’interno delle pagine web.

### 2.1.1 Sviluppo di un modello

All’atto pratico, alla base del machine learning c’è un **modello**, il vero e proprio nucleo che compone il sistema che si vuole costruire.

Il modello deve riuscire ad approssimare nel modo migliore possibile la realtà descritta dai dati, trovando correlazioni non banali tra loro.

In base alla tipologia e all’organizzazione dei dati a disposizione, 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;
3. Apprendimento per rinforzo.

### 2.1.2 Apprendimento supervisionato

Nell’**apprendimento supervisionato** i dati disponibili sono composti da due parti principali: una parte di input da affidare al modello che contiene uno o più campi descrittivi, che definiscono le caratteristiche di ogni elemento, e una parte in cui è specificato il valore in output reale.

Si prenda come esempio il seguente dataset (di tipo 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	<i>Setosa</i>
7.7	3.8	6.7	2.2	<i>Virginica</i>
5.4	3.7	1.5	0.2	<i>Setosa</i>
7.2	3.6	6.1	2.5	<i>Virginica</i>
6.0	3.4	4.5	1.6	<i>Versicolor</i>
6.2	3.4	5.4	2.3	<i>Virginica</i>
5.0	3.3	1.4	0.2	<i>Setosa</i>
6.3	3.3	4.7	1.6	<i>Versicolor</i>
6.7	3.3	5.7	2.1	<i>Virginica</i>

Tabella 2.1: Esempio di dati in ingresso con le caratteristiche di diversi fiori.

Le prime 4 colonne compongono l'insieme di dati da passare al modello per far sì che riesca ad elaborare tutte le correlazioni esistenti, mentre l'ultima colonna contiene il valore che ci si aspetta in output dopo l'elaborazione.

Il modello, avendo disponibile il risultato corretto da predire in uscita, può effettuare una predizione, e comportarsi di conseguenza in base alla differenza tra risultato atteso e risultato predetto.

Quindi ciclicamente, un sistema di machine learning supervisionato segue questo schema:

- Predizione in base ai dati di input;
- Controllo della differenza tra risultati attesi e predetti (**errore del modello**);
- Miglioramento del modello in base agli errori commessi.

Questo processo è definito **addestramento** del modello, e viene eseguito finché l'errore ottenuto non raggiunge un livello abbastanza basso da poter considerare il modello valido. Verrà affrontato nella sezione 2.1.5 il modo in cui il modello riesce a migliorare le proprie predizioni.

I modelli supervisionati sono generalmente usati per 2 diversi tipi di problemi:

- **Classificazione** - Come nell'esempio della tabella 2.1 relativa alla specie corretta dell'iris, abbinare l'elemento alla categoria corretta;
- **Regressione** - Predire un valore numerico, al posto di una categoria (come ad esempio il prezzo di un immobile), partendo dalle caratteristiche dell'elemento.

### 2.1.3 Apprendimento non supervisionato

Nell'**apprendimento non supervisionato** i dati disponibili **non** contengono alcun tipo di indicazione sulla bontà della previsione. Questo tipo di informazione è detta anche *non etichettata*,

Ovviamente i dati con il valore richiesto in output sono molto più facili da reperire, poiché non è necessaria l'analisi di una persona umana (il dataset d'esempio precedente ha richiesto l'identificazione della specie corretta per ogni fiore presente).

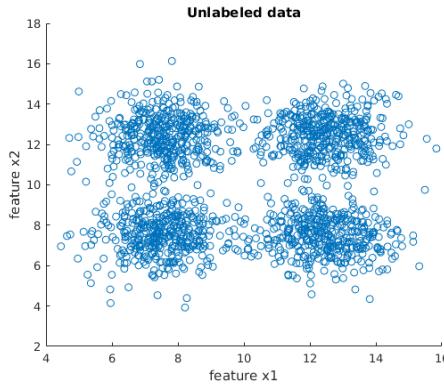


Figura 2.2: Rappresentazione grafica di un problema di clustering.

Fonte: [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/unsupervised\\_learning.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/unsupervised_learning.html)

Non avendo alcun tipo di *indicazione* sul valore che si aspetta in uscita, **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 però a priori avere le possibili categorie.

Il fatto di non sapere in partenza le diverse categorie, risulta però essere in molti casi vantaggioso, poiché il modello riesce ad analizzare e cogliere legami spesso *invisibili* alla mente umana.

#### 2.1.4 Apprendimento per rinforzo

**L'apprendimento per rinforzo** è una tipologia utilizzata quando è necessario che il nostro modello impari relazionandosi con l'ambiente in cui si trova: non sfrutta dataset pregressi, ma le mutazioni che riesce a captare.

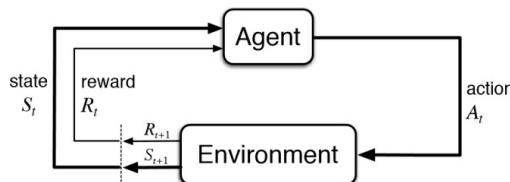


Figura 2.3: Modello di base nell'apprendimento di rinforzo.

Fonte: <https://www.kdnuggets.com/images/reinforcement-learning-fig1-700.jpg>

Viene utilizzato in casi molto particolari, generalmente facendo eseguire un compito a un modello in maniera ciclica. Dopo tantissimi tentativi (anche

svariati milioni) il sistema risulta esser in grado di svolgere operazioni complesse con un'accuratezza maggiore di quella umana.

Ad esempio, l'apprendimento per rinforzo è stato utilizzato in maniera soddisfacente per insegnare a un modello senza alcuna conoscenza di base a giocare a scacchi, affinando la propria tecnica partita dopo partita, arrivando a competere (e vincere) anche contro campioni mondiali.

### 2.1.5 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.

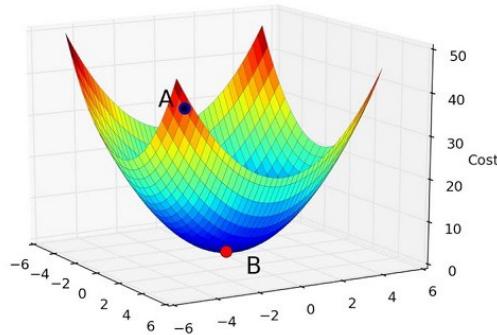


Figura 2.4: Discesa del gradiente dal punto A al punto B visualizzabile graficamente.

Fonte: <https://medium.com/abdullah-al-imran/intuition-of-gradient-descent-for-machine-learning-49e1b6b89c8b>

Nello stesso modo, 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 in figura 2.4 ad esempio, 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 quindi, è 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 affonda le proprie radici nel campo della statistica, dove viene chiamato **regressione**. Si occupa di analizzare un insieme di dati che possono essere rappresentati da una funzione (lineare, polinomiale, o di altri gradi) e cercare di approssimare al meglio le variabili dipendenti e indipendenti.

### 2.1.6 Validazione del modello

Avendo un insieme di dati su cui addestrare il modello, non vengono utilizzati tutti per la fase di training, ma al contrario 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 (*training set*) verrà addestrato il modello, mentre il restante più piccolo (*validation set*) verrà utilizzato per controllare la validità dell'addestramento.

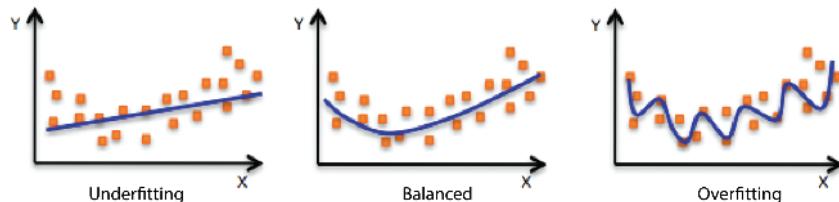


Figura 2.5: Esempi di *Underfitting*, *Apprendimento bilanciato* e *Overfitting*.

Il rischio, infatti, è quello di addestrare **troppo** un modello, in grado di produrre risultati accurati al 100% solo sui dati su cui viene sviluppato.

Al contrario, un buon sistema di machine learning dovrebbe essere il più generale possibile, ed è per questo che questa metodologia utilizza il gruppo di dati più piccolo, detto **validation set**, per controllare il comportamento del modello dopo l'addestramento:

- Se i risultati ottenuti nelle predizioni sono molto buoni solo sul training set ma non sul validation, allora c'è overfitting;
- Se su entrambi gli insiemi il modello si comporta *circa* nello stesso modo, allora l'addestramento risulta valido.
- Se il modello risulta poco accurato su entrambi gli insiemi, allora c'è underfitting.

## 2.2 Deep Learning

Il machine learning, però, mostra alcune limitazioni nella precisione delle previsioni sviluppate, in particolar modo se:

- Il tipo di dato che deve gestire è di tipo immagine o testuale (in particolar modo con testi naturali);
- La quantità di dati disponibili per il training non è così abbondante.

Inoltre, per poter utilizzare il machine learning è necessario **selezionare le feature manualmente**, e ciò richiede delle competenze non banali.

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

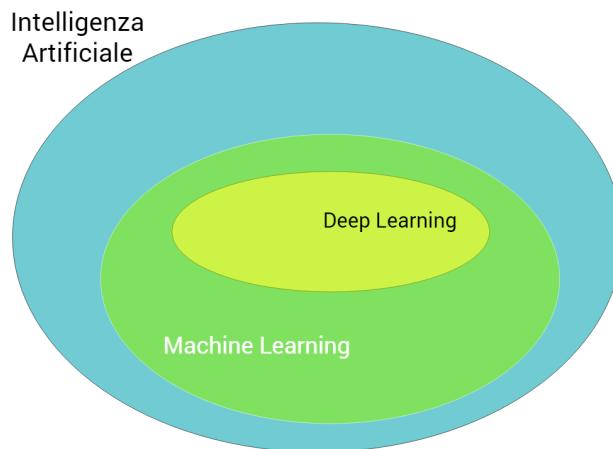


Figura 2.6: Gerarchia delle tecnologie di intelligenza artificiale.

Questa disciplina 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 (da qui il motivo della parola *deep*) che collaborano tra loro per raggiungere risultati estremamente accurati.

Nello stesso modo, all'interno delle **reti neurali** sviluppate tramite deep learning non si ritrova *un solo modello* da addestrare e da gestire, ma una fitta rete di *neuroni* raggruppati all'interno di strati.

Dopo aver eseguito l'addestramento sui dati di training, queste riescono ad eseguire i compiti più disparati, con una **precisione maggiore** di quella raggiungibile tramite machine learning: categorizzare immagini, riconoscere

lettere e numeri scritti a mano, e nei casi più avanzati anche a generare immagini realistiche o testi di senso compiuto.

La divisione in strati dei neuroni consente di gestire in maniera progressiva l'astrazione dei problemi da affrontare: nei livelli più bassi si riescono a riconoscere aspetti e pattern semplici, mentre negli strati più avanzati si gestiscono gli elementi di più alto livello, così da comprendere e individuare caratteristiche sempre più complesse.

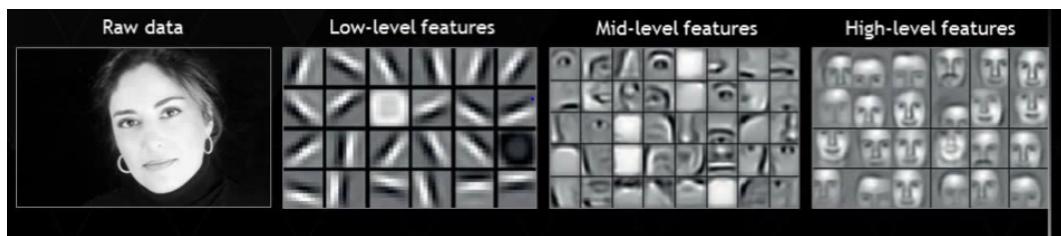


Figura 2.7: Diverse astrazioni trattate nelle reti neurali.

Fonte: <https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/>

Questo nuovo approccio ha consentito nel tempo di conseguire risultati impressionanti per precisione e tempi richiesti per il calcolo.

L'esempio più cristallino di quanto una rete neurale sia molto più accurata rispetto a metodi tradizionali, è l'applicazione a problemi riguardanti le immagini (uno su tutti riconoscimento e la categorizzazione dei soggetti) o il riconoscimento della grafia. Le reti neurali più efficaci per le immagini sono quelle convoluzionali e recentemente è stata dimostrata la loro efficacia anche per il trattamento di dati testuali [5].

Prendendo come esempio il popolare dataset MNIST (utilizzato generalmente come punto di riferimento in questo ambito), contenente svariate migliaia di immagini raffiguranti cifre, il tasso di errore migliore mai fatto registrare è attualmente dello **0.23%** [6], ed è stato raggiunto proprio da una rete neurale.

### 2.2.1 Struttura

Come detto, un sistema di deep learning si sviluppa in una struttura molto più complessa e articolata rispetto a un modello di machine learning.

Si creano degli strati di neuroni, elementi fondamentali che altro non sono che dei semplici algoritmi ai quali, dato in input un vettore di  $n$  elementi, generano un risultato in output generalmente compreso tra 0 e 1.

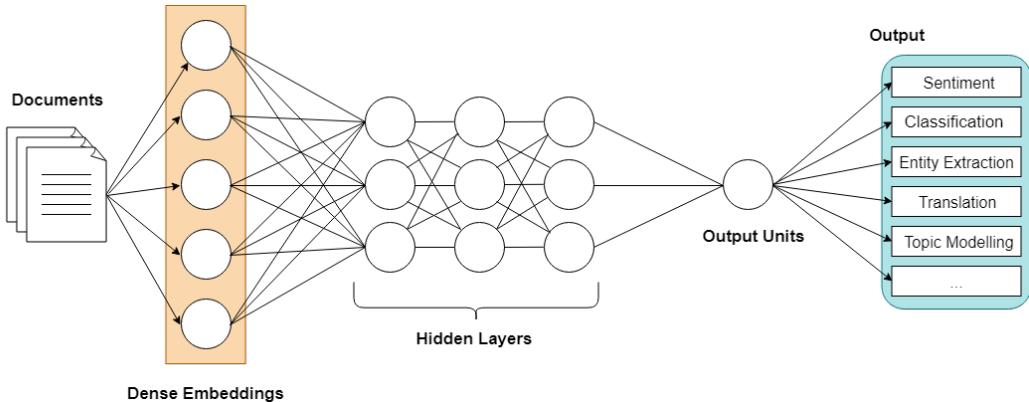


Figura 2.8: Struttura tipica di una rete neurale.

Gli strati sono tra loro collegati, e le informazioni in uscita di uno sono quelle in entrata del successivo. Ogni livello rappresenta un grado di astrazione differente, nel quale vengono gestiti aspetti differenti del dato in input, come in figura 2.7. Nel livello di output viene invece elaborata l'informazione di alto livello richiesta.

Ogni singolo neurone, riceve in input un insieme di valori all'interno di un vettore (detto in certi casi *tensore*) ed elabora internamente il valore  $y$  da ritornare in output tale che:

$$y = \sigma\left(\sum_{i=1}^d w_i x_i + b\right)$$

Viene quindi calcolata la combinazione lineare degli input  $x$  con i pesi  $w$  del singolo neurone, viene sommata l'intercetta  $b$  e successivamente viene applicata una funzione di attivazione  $\sigma$ .

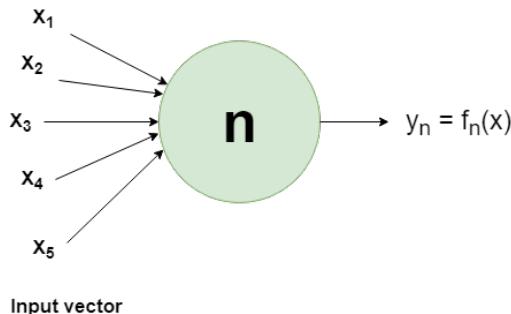


Figura 2.9: Struttura di un neurone.

### 2.2.2 Funzione di attivazione

Come accennato in precedenza, la **funzione di attivazione** di un neurone definisce l'output che quel nodo restituisce. Ne esistono di diversi tipi ma si possono classificare in:

- **Funzioni di attivazione lineari o di identità:** l'output è proporzionale all'input. Solitamente poco utilizzate nelle reti neurali poiché non permettono di creare mappature complesse dei dati.
- **Funzioni di attivazione non lineari:** sono le più diffuse all'interno delle reti neurali.

Le funzioni di attivazione non lineari permettono quindi di modellare dati complessi come immagini, video, audio e set di dati con correlazioni non lineari o con elevata dimensionalità. Ve ne sono svariate, tuttavia le più utilizzate sono le seguenti:

- **Sigmoide:** è un'approssimazione della funzione "gradino". È caratterizzata da una curva a forma di "S". Viene spesso utilizzata per mappare valori reali in probabilità.

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

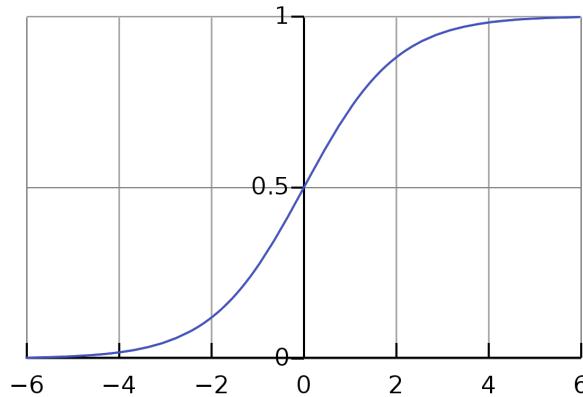


Figura 2.10: Grafico della funzione sigmoide

- **ReLU:** è stata dimostrata essere la migliore da utilizzare all'interno di reti neurali profonde poiché ne favorisce un migliore addestramento

rispetto alle altre funzioni di attivazione [7]. Di fatto, restituisce la parte positiva dell'argomento.

$$\sigma(x) = x^+ = \max(0, x)$$

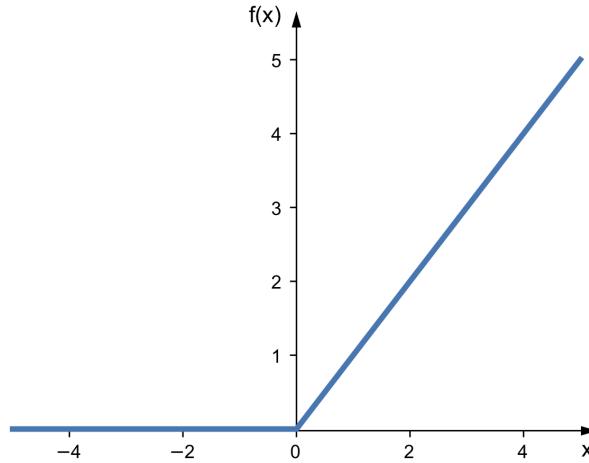


Figura 2.11: Grafico della funzione ReLU

- **Softplus:** è un'approssimazione della funzione  $\max$ .

$$\sigma(x) = \ln(1 + e^x)$$

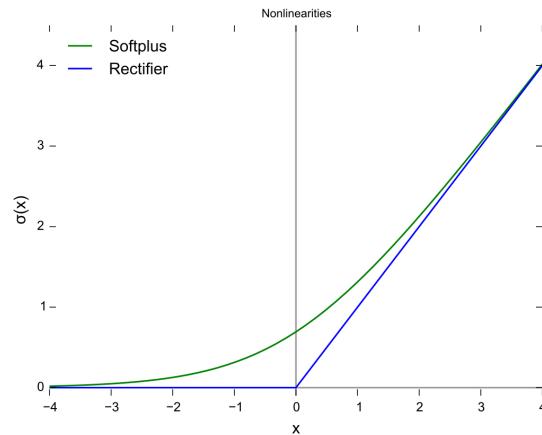


Figura 2.12: Grafico della funzione Softplus

### 2.2.3 Addestramento

Per anni i ricercatori hanno ricercato un metodo per addestrare reti neurali. Nella seconda metà degli anni '80, è stato introdotto l'algoritmo di *retro-propagazione* [8]. Quest'ultimo è in grado di calcolare il gradiente dell'errore commesso dalla rete considerando tutti i parametri della stessa. Ovvero, può scoprire come ogni peso della rete debba essere modificato in modo da ridurre l'errore: una volta calcolato il gradiente, semplicemente effettua un passo di discesa del gradiente. Il processo è ripetuto finché la rete converge alla soluzione.

L'algoritmo di retro-propagazione gestisce un gruppo di istanze (*batch*) per volta e scorre il training set diverse volte. Ogni batch attraversa quindi la rete, proprio come per ottenere una predizione: l'unica differenza è che i risultati degli strati intermedi vengono conservati. Successivamente, viene calcolato l'errore commesso dalla rete confrontando l'output con i risultati desiderati e quanto ogni connessione della rete (ogni peso) abbia contribuito all'errore. Quest'ultimo passaggio viene eseguito a ritroso, ovvero partendo dallo strato di output e arretrando fino ad arrivare allo strato di input. Infine, l'algoritmo effettua un passo di discesa del gradiente per modificare i pesi sfruttando il gradiente appena calcolato.

## 2.3 Visione Artificiale

La disciplina che si occupa dei problemi che hanno immagini come dati è detta **visione artificiale**. Il suo scopo principale è quello di riprodurre la vista umana. Vedere è inteso non solo come l'acquisizione di una fotografia bidimensionale di un'area ma soprattutto come l'interpretazione del contenuto di quell'area.

Alcuni classici problemi nella visione artificiale sono infatti:

- **Riconoscimento:** uno o più oggetti prespecificati o memorizzati possono essere ricondotti a classi generiche usualmente insieme alla loro posizione nella scena.
- **Identificazione:** viene individuata un'istanza specifica di una classe. Es. Identificazione di un volto.
- **Rilevamento:** l'immagine è scandita fino all'individuazione di una condizione specifica. Es. Individuazione di possibili cellule anormali o tessuti nelle immagini mediche.

Per svolgere questi compiti, l'immagine deve essere memorizzata in formato digitale, ovvero viene creata una rappresentazione numerica corrispondente

della stessa. L'immagine viene quindi trasformata da punti ottici (pixel) a valori digitali in memoria, con corrispondenza in matrice. A questo punto, l'immagine verrà analizzata da algoritmi in grado di estrarre da essa varie informazioni. Queste possono collocarsi, come già visto prima, su più livelli:

- basso livello: come le statistiche sulla presenza dei vari toni di grigio o colori, sui bruschi cambiamenti di luminosità, ecc.
- livello intermedio: caratteristiche relative a regioni dell'immagine e a relazioni tra regioni.
- alto livello: determinazione di oggetti con valenza semantica.

### 2.3.1 Reti Neurali Convoluzionali

Le **Reti Neurali Convoluzionali** (CNN) sono nate negli anni '80 in seguito a diversi studi della corteccia visiva del cervello [9] [10]. In particolare, è stato scoperto che i neuroni della corteccia visiva hanno un piccolo *campo recettivo locale*: ciò significa che reagiscono solo agli stimoli visivi che provengono da una regione limitata del campo visivo. I campi recettivi di neuroni diversi possono sovrapporsi, e insieme compongono l'intero campo visivo.

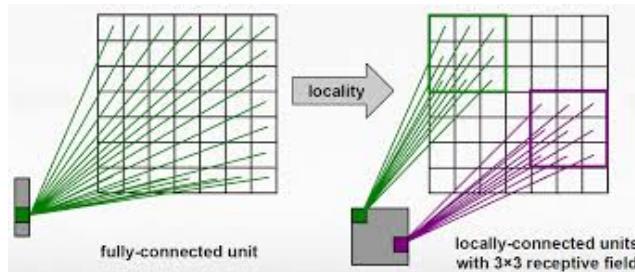


Figura 2.13: Campo recettivo di un neurone in una rete neurale semplice (sinistra) e in una rete neurale convoluzionale (destra).

È stato scoperto inoltre che alcuni neuroni reagiscono solo a immagini di linee orizzontali, mentre altri reagiscono solo a linee con diverse orientazioni (ciò significa che due neuroni potrebbero avere lo stesso campo recettivo ma reagiscono a linee con orientazioni diverse). Si è inoltre notato come alcuni neuroni abbiano campi recettivi più ampi e reagiscano a pattern più complessi (che sono combinazioni di pattern più semplici).

Queste osservazioni hanno suggerito che vi potesse essere una struttura gerarchica tra i neuroni: i neuroni di alto livello si basano sui risultati dei neuroni di livello inferiore a loro vicini. Questa complessa e potente architettura

è in grado di identificare anche i pattern più complessi in qualsiasi area del campo visivo.

Le Reti Neurali Convoluzionali presentano molti elementi in comune con le reti neurali più semplici; tuttavia, introducono anche alcune novità:

- Strato Convoluzionale
- Strato di Pooling

### Strato Convoluzionale

L'elemento costitutivo più importante per una Rete Neurale Convoluzionale, come si può intuire dalla denominazione, è sicuramente lo strato convoluzionale. I neuroni di questo strato non sono connessi ad ogni input dello strato precedente, ma solo agli input che fanno parte del loro campo recettivo. Questa architettura permette alla rete di concentrarsi su piccole caratteristiche di basso livello nei primi strati, per poi assemblarle in caratteristiche più grandi e di alto livello negli strati successivi. Questa struttura gerarchica è comune nelle immagini reali: questo è uno dei motivi per cui le CNN si comportano così bene con il riconoscimento di immagini.

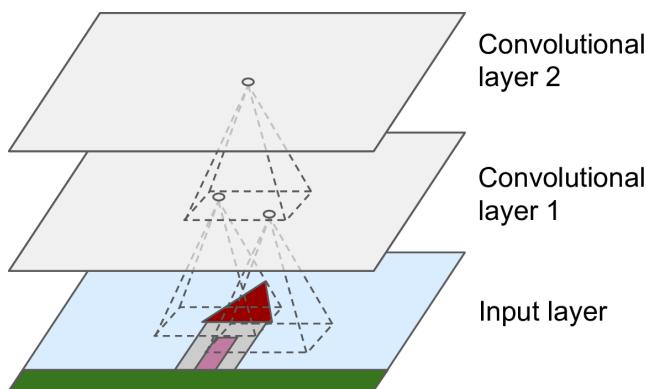


Figura 2.14: Strati convoluzionali con campi recettivi locali rettangolari.

Ogni neurone all'interno di uno strato presenta dei "pesi". Questi possono essere rappresentati come piccole immagini della dimensione del campo recettivo e prendono il nome di *filtri*. L'applicazione di un filtro ad un'immagine comporta la creazione di una *mappa delle caratteristiche*, che evidenzia le zone dell'immagine che attivano maggiormente il filtro.

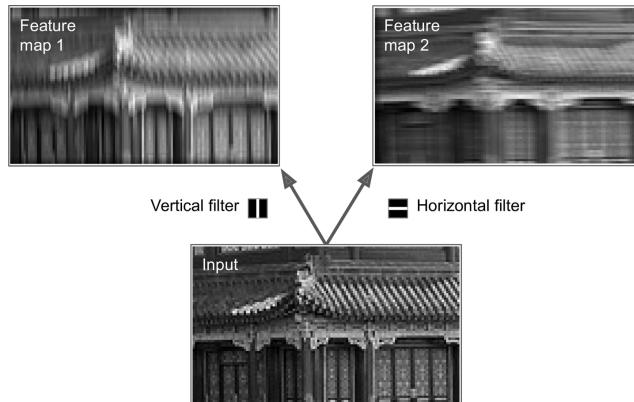


Figura 2.15: Esempio di applicazione di filtri diversi e generazione delle mappe delle caratteristiche corrispondenti.

Ovviamente, i filtri non devono essere definiti a mano ma la rete, durante l’addestramento, apprenderà da sola quali sono i filtri più utili per il compito da svolgere e come combinarli per ottenere pattern più complessi.

Ogni strato convoluzionale presenta più filtri e, di conseguenza, genererà altrettante mappe delle caratteristiche, perciò la rappresentazione più corretta dello strato sarebbe la seguente:

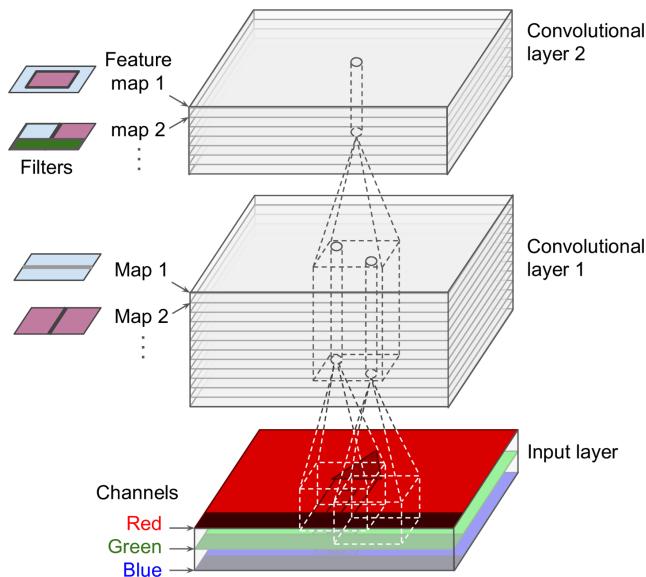


Figura 2.16: Rappresentazione in 3D di uno strato convoluzionale.

### Strato di Pooling

Lo scopo dello strato di pooling è quello di restringere l'immagine di input in modo da ridurre il costo computazionale, l'occupazione di memoria e il numero di parametri (permettendo di limitare l'overfitting).

Proprio come negli strati convoluzionali, ogni neurone è connesso solo ad un numero limitato di neuroni dello strato precedente, tipicamente situati in un piccolo campo recettivo rettangolare. Tuttavia, i neuroni di uno strato di pooling non hanno pesi; tutto ciò che fanno è aggregare gli input utilizzando una funzione di aggregazione. I più utilizzati sono:

- **MaxPooling:** l'output è rappresentato dal valore massimo della matrice di partenza.
- **AveragePooling:** l'output è costituito dalla media dei valori della matrice di partenza.

Perciò, solo il valore restituito dalla funzione di aggregazione viene mantenuto, gli altri input vengono scartati.

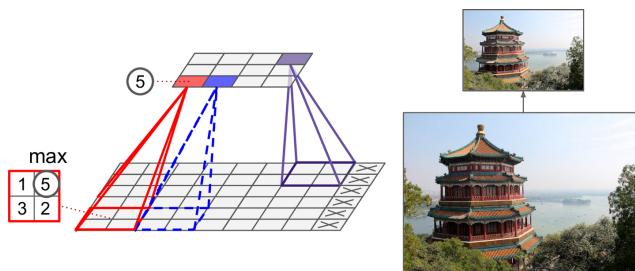


Figura 2.17: Rappresentazione di uno strato MaxPooling.

### Architettura delle Reti Neurali Convolutionali

Tipicamente, l'architettura di una rete neurale convoluzionale è costituita dall'alternarsi di strati convoluzionali e di pooling [11]. Il risultato di questa struttura è che l'immagine diventa sempre più piccola ma sempre più "profonda", cioè sarà costituita da numerose mappe di caratteristiche.

In cima alla pila di strati convoluzionali e di pooling viene poi aggiunta una rete neurale semplice composta da strati densamente connessi e lo strato finale restituirà il risultato della predizione.

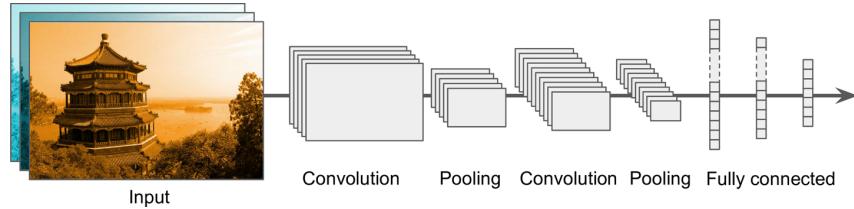


Figura 2.18: Architettura tipica di una rete neurale convoluzionale

L'introduzione degli strati convoluzionali e di pooling permette alla rete di lavorare anche con immagini di grandi dimensioni: una rete neurale profonda, con strati densamente connessi, richiederebbe infatti troppi parametri. Basti pensare che un'immagine di 100x100 pixel (che risulta comunque di piccole dimensioni) contiene 10000 pixel: se considerassimo un primo strato con solo 1000 neuroni, in una rete tradizionale otterremmo già 10 milioni di connessioni (soltanto per questo primo strato!). Le reti neurali convoluzionali risolvono questo problema usando strati parzialmente connessi e condivisione dei pesi.

# Capitolo 3

## Ambiente di lavoro e strumenti utilizzati

In questo capitolo verranno illustrati l'ambiente di lavoro e gli strumenti utilizzati per sviluppare le varie soluzioni testate.

### 3.1 Micro-controllori

Un micro-controllore è un dispositivo elettronico integrato su singolo circuito elettronico utilizzato generalmente in sistemi embedded, ovvero per applicazioni specifiche di controllo digitale.

Ad oggi, uno dei paradigmi di erogazione di servizi più utilizzati è il *cloud computing*. Questo si basa su diversi tipi di nodi:

- **nodi sensori**: sono sistemi che dispongono di risorse molto limitate e il loro compito è quello di raccogliere dati che verranno poi elaborati in cloud.
- **nodi edge**: si tratta di sistemi che hanno una disponibilità di risorse solitamente superiore a quella dei sensori e fungono da ponte di comunicazione tra questi ultimi e i nodi cloud. Fanno parte di questa categoria anche i micro-controllori.
- **nodi cloud**: sono sistemi con grande disponibilità di risorse ed eseguono computazioni sui dati ricevuti.

Questo paradigma presenta numerosi vantaggi, tra cui elevate prestazioni poiché i nodi cloud forniscono grande potenza di calcolo e spazio di archiviazione. Tuttavia, in contesti in cui si deve operare con bassa latenza, o addirittura in tempo reale, questa architettura presenta alcuni lati negativi. Infatti, il

passaggio di informazioni dai sensori, ai nodi edge ed infine ai nodi cloud richiede una grande capacità di rete e, soprattutto, è caratterizzato da una latenza introdotta dalla rete stessa.

Tutto ciò ha fatto sì che, in parallelo, si sviluppasse un nuovo paradigma di calcolo distribuito denominato **edge computing** caratterizzato dalla decentralizzazione delle risorse di calcolo. Questo dislocamento del calcolo dei dati prevede quindi l'utilizzo per la computazione dei nodi edge e si adatta perfettamente al contesto *Internet of Things*. Dato che la loro capacità computazionale è notevolmente aumentata e il loro prezzo è ad oggi molto accessibile, è possibile sfruttarli per eseguire anche modelli di intelligenza artificiale.

### 3.1.1 Nvidia Jetson Nano Developer Kit

La Nvidia Jetson Nano Developer Kit è una scheda di sviluppo di intelligenza artificiale che integra unità di calcolo volte all'accelerazione di algoritmi di machine learning e visione artificiale. Sebbene tale prodotto si rivolga prevalentemente al mercato della prototipazione, i suoi costi contenuti e la grande capacità computazionale, unite all'ottima efficienza energetica, la rendono appetibile anche in parecchie applicazioni commerciali che si basano su reti neurali profonde, specialmente nel settore dell'Internet of Things, come classificazione di immagini, riconoscimento di oggetti, processing multimediale, GPU computing e deep learning.

La Jetson Nano si basa su un modulo GPU Nvidia Maxwell a 128 core affiancato da una CPU quad-core ARM Cortex-A57 con parallelismo a 64 bit e frequenza di clock a 1,43 GHz, supportate da una RAM LPDDR4 da 4GB con bus a 64 bit a 1600 MHz. Il punto di forza di questa configurazione hardware, oltre al prezzo estremamente competitivo, è il consumo energetico stimato compreso tra i 5 e i 10 W grazie all'efficienza dei componenti scelti.

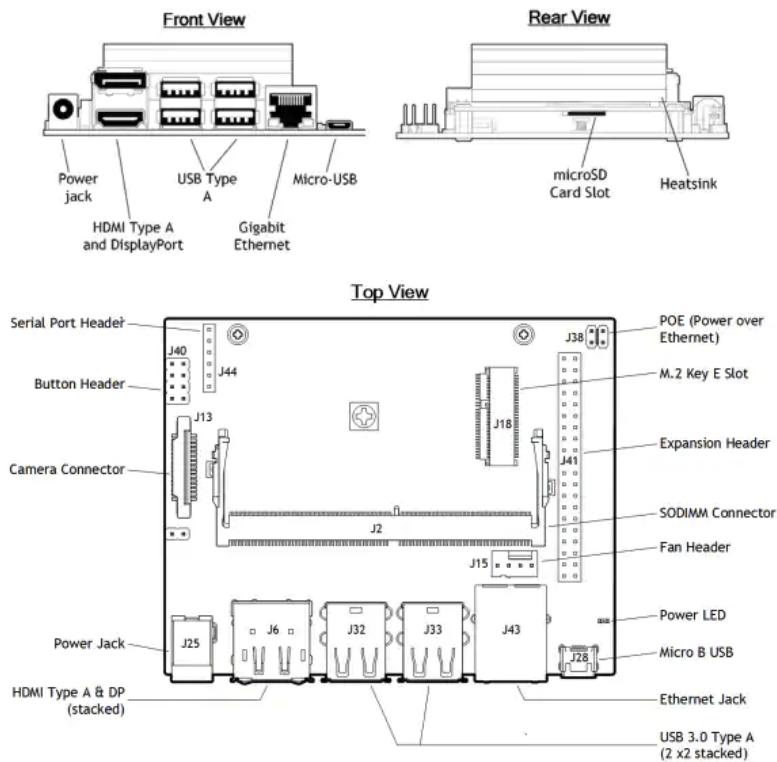


Figura 3.1: Schema Nvidia Jetson Nano Developer Kit

La Jetson Nano è supportata dal Nvidia JetPack SDK, un Software Development Kit sviluppato da Nvidia per tutti i suoi moduli della serie Jetson, che permette di gestire in maniera ottimale i prodotti dell'azienda, fornendo un software studiato per garantire le massime prestazioni nelle applicazioni di intelligenza artificiale sulla base dell'hardware adottato. Jetpack, oltre a fornire un sistema operativo, mette a disposizione librerie, esempi, strumenti di sviluppo e documentazione. Il sistema operativo è un derivato di Linux Ubuntu 18.04 in versione 64 bit con kernel 4.9, che fornisce automaticamente numerose librerie e driver, sufficienti per la maggior parte delle applicazioni.

### 3.1.2 Raspberry Pi 4 Model B

La Raspberry Pi 4 Model B è una scheda finalizzata all'esecuzione di sistemi operativi basati su kernel Linux e dal costo molto contenuto, risultando una delle soluzioni più diffuse nella prototipazione e nelle applicazioni a ridotto costo computazionale: caratteristiche che la rendono una scelta molto valida in svariati campi, incluso quello dell'Internet of Things. Nonostante non vi siano

acceleratori specificamente finalizzati all'incremento delle prestazioni di reti neurali profonde, l'impiego di questa scheda per svolgere compiti come la classificazione di immagini e il deep learning risulta essere comunque un'interessante ed economica soluzione.

Essendo la versione più aggiornata della scheda, la Raspberry Pi 4 Model B beneficia di un hardware attuale ed efficiente basato su una CPU Broadcom BCM2711 quad-core Cortex-A72 a 64 bit con una frequenza di clock di 1,5 GHz. La scheda è disponibile nei tagli di RAM da 1, 2, 4 e 8 GB LPDDR4 con bus a 64 bit e frequenza di 3200 MHz. Verrà utilizzata in questo caso la scheda con il massimo ammontare di memoria RAM pari a 8 GB. Pur non avendo una GPU dedicata, la scheda è provvista di acceleratori per la codifica e decodifica video (VPU) e supporto alle librerie grafiche OpenGL ES 3.0, garantendo una buona velocità ed efficienza nelle applicazioni multimediali.

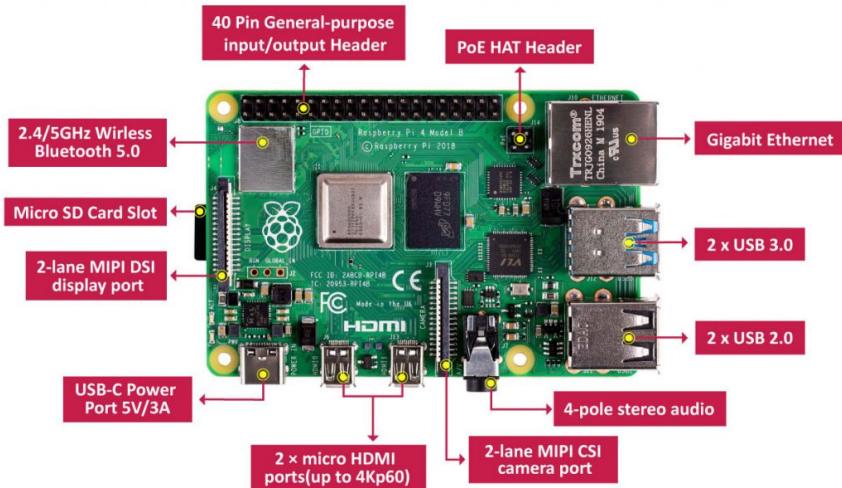


Figura 3.2: Schema Raspberry Pi 4 Model B

Grazie alla sua ampia diffusione e ai suoi diversi contesti di applicazione, la Raspberry Pi 4 dispone di diversi sistemi operativi di terze parti realizzati e ottimizzati appositamente per la scheda. Tuttavia il sistema operativo ufficiale e più conosciuto, nonché lo stesso diffuso da Raspberry Pi Foundation, è Raspberry Pi OS (chiamato precedentemente Raspbian). Questo deriva dalla distribuzione Linux Debian 10 Buster ottimizzato per architetture ARM.

## 3.2 Python

Il linguaggio utilizzato per lo sviluppo è Python. È un linguaggio interpretato ed estremamente flessibile. Inoltre, presenta numerose librerie che permettono

di gestire al meglio i dati e di interagire con i modelli di machine learning.

### 3.2.1 Librerie Python

Le librerie Python più utilizzate sono le seguenti:

- **Tensorflow**: è una libreria software open source per il machine learning che fornisce moduli sperimentali e ottimizzati, utili nella realizzazione di algoritmi per diversi tipi di compiti percettivi e di comprensione del linguaggio. La sua variante **Tensorflow Lite** permette di eseguire dei modelli di intelligenza artificiale estremamente leggeri e poco onerosi di risorse, mantenendo comunque una buona efficacia: questi sono perfetti per operare su dispositivi mobili e micro-controllori.
- **OpenCV**: è una libreria software multi-piattaforma nell'ambito della visione artificiale. Permette la lettura, elaborazione, conversione e visualizzazione delle immagini [12].
- **Numpy**: è una libreria open source che aggiunge supporto a grandi matrici e array multidimensionali insieme ad una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati [13].

## 3.3 Google Colaboratory

Colaboratory o, in breve, **Colab** è un prodotto di *Google Research*. Colab permette a chiunque di scrivere ed eseguire codice Python arbitrario tramite il browser ed è particolarmente adatto per machine learning, analisi dei dati e formazione. Più tecnicamente, è un servizio di blocchi note Jupyter ospitato, il cui utilizzo non richiede alcuna configurazione, che fornisce al contempo l'accesso gratuito alle risorse di calcolo, comprese le GPU e TPU.

Inizialmente, le soluzioni e i modelli sono stati testati su questa piattaforma per comodità e per facilità di condivisione.



# Capitolo 4

## Panoramica delle soluzioni testate

Di seguito sono descritte e discusse le varie soluzioni testate che hanno condotto infine alla soluzione finale del problema.

### 4.1 Addestrare una nuova rete

Il primo approccio al problema è stato quello di addestrare una rete neurale per svolgere il compito desiderato. Tuttavia, la fase di addestramento da zero di una rete neurale è un procedimento estremamente oneroso di risorse computazionali e di tempo. Inoltre, non è assolutamente garantito il raggiungimento di un risultato soddisfacente.

Per questo motivo si è scelto di iniziare l'addestramento basandosi su una rete neurale già esistente.

#### 4.1.1 Transfer Learning

Il transfer learning è una tecnica che permette di addestrare una rete neurale sfruttando le conoscenze già acquisite da una rete esistente. Questo è possibile poiché, come già detto, la conoscenza di una rete neurale è strutturata in modo gerarchico: i primi strati estraggono delle caratteristiche di basso livello e più generali mentre gli ultimi strati acquisiscono delle informazioni più specifiche e dettagliate. È quindi possibile e consigliato riutilizzare per una nuova rete neurale i pesi già acquisiti da una rete che svolge un compito simile.

Per eseguire questa tecnica è necessario recuperare i primi strati della rete già esistente e renderli *non addestrabili*. In questo modo durante la fase di addestramento verranno modificati solo i parametri degli ultimi strati. Successivamente, è utile eseguire un ulteriore addestramento della rete rendendo però questa volta modificabili tutti i parametri. Questo permetterà di rendere la rete ancora più precisa.

### 4.1.2 Xception

La rete scelta per eseguire il transfer learning è la **Xception** [14]: si tratta di una rete convoluzionale già addestrata su uno dei più popolari dataset che è **ImageNet** [15] il quale contiene milioni di immagini di migliaia di categorie di oggetti. La rete è infatti in grado di riconoscere persone, animali, automobili, edifici e migliaia di altre categorie.

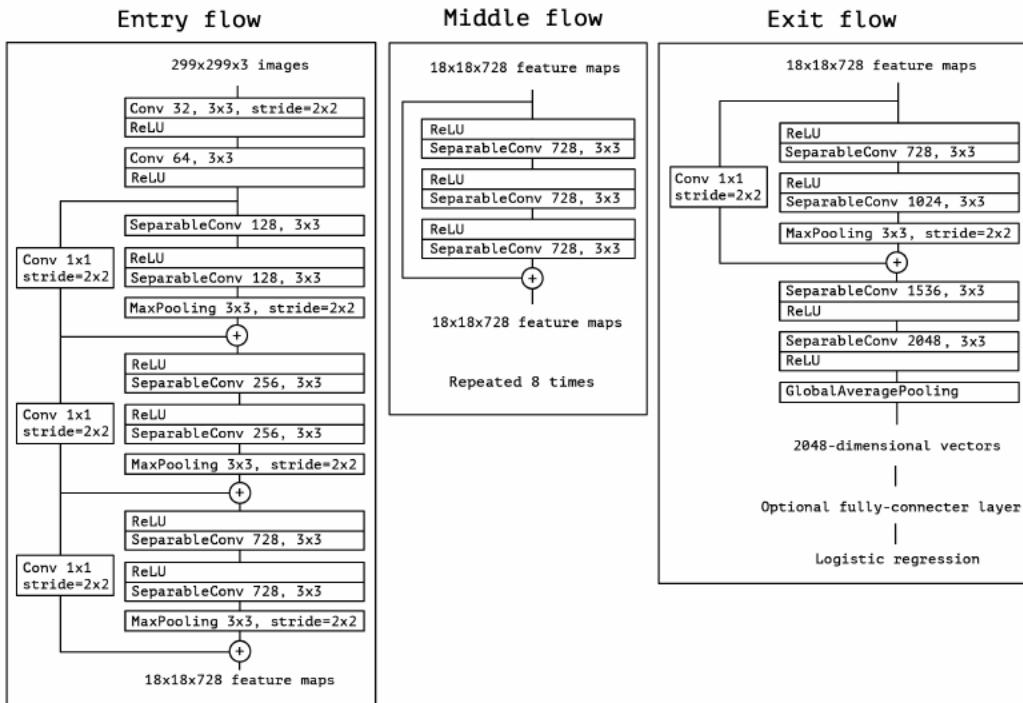


Figura 4.1: Architettura della rete Xception.

Come mostrato in figura 4.1, la sua architettura è caratterizzata prevalentemente da strati convoluzionali per l'estrazione di caratteristiche e strati di pooling per ridurre le dimensioni delle immagini. Sono presenti anche *connessioni residue* originariamente introdotte dalla rete **ResNet** [16]. Si tratta di una tecnica che consente di velocizzare notevolmente l'addestramento della rete e si esegue replicando il segnale in entrata di uno strato anche all'output di uno strato che si trova più avanti nella rete.

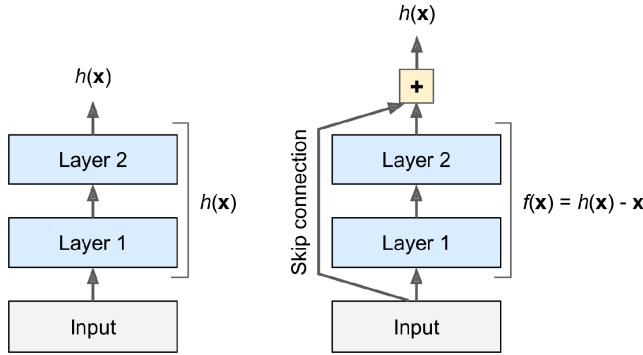


Figura 4.2: Differenza tra una normale rete (sinistra) e una rete con connessione residua (destra).

La rete Xception presenta inoltre un particolare tipo di strati convoluzionali: gli *strati convoluzionali separabili*. Mentre gli strati convoluzionali semplici utilizzano filtri che cercano di individuare allo stesso tempo dei pattern sia spaziali che su canali diversi, in quelli separabili si assume che questi due tipi di pattern possano essere modellati separatamente. Sono perciò costituiti da due parti: la prima applica un singolo filtro spaziale per ogni mappa delle caratteristiche in ingresso, poi la seconda cerca esclusivamente dei pattern tra canali diversi.

Questo tipo di strato richiede meno parametri, memoria e capacità computazionale ed è generalmente migliore dello strato tradizionale.

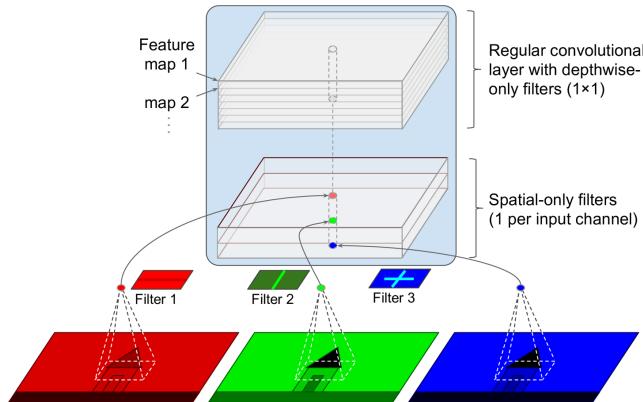


Figura 4.3: Rappresentazione di uno strato convoluzionale separabile.

### 4.1.3 Data Augmentation

Per l'addestramento della rete è stato utilizzato il dataset **RMFD** descritto in precedenza. Tuttavia, questo risultava fortemente sbilanciato poiché conteneva un numero di immagini di persone senza mascherina significativamente

maggiori di quelle di persone con mascherina. Tale caratteristica rende impossibile l'addestramento di una rete in grado di generalizzare al meglio i dati: basti pensare che la rete raggiungerebbe un'accuratezza molto alta semplicemente classificando tutte le immagini come appartenenti alla classe più numerosa.

Si è deciso quindi di eseguire un processo di **data augmentation**. Questo consiste nell'aumentare il numero di istanze della classe meno numerosa sfruttando quelle già a nostra disposizione: nel nostro caso sfrutteremo le immagini di persone con mascherina per crearne di nuove.

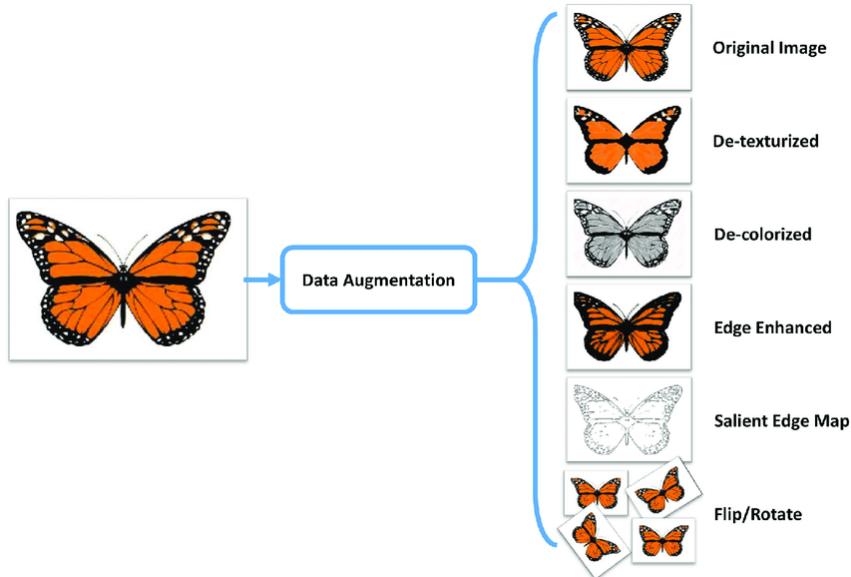


Figura 4.4: Esempio di data augmentation.

Come si può notare dalla figura 4.4, la data augmentation consiste nell'effettuare diverse elaborazioni sulla stessa immagine. Per citarne solo alcune:

- Rotazione dell'immagine
- Inversione dell'immagine sui piani orizzontale e/o verticale
- Ritaglio dell'immagine
- Applicazione di saturazione
- Modifica della luminosità

#### 4.1.4 Fase di addestramento

L'addestramento è stato quindi effettuato utilizzando la tecnica del transfer learning a partire dalla rete Xception sfruttando il 70% del dataset RMFD (il

restante 30% è stato utilizzato come validation set) su cui era stata eseguita la data augmentation. Sono state utilizzate anche funzioni che hanno permesso di individuare gli iperparametri di addestramento migliori, ovvero configurazioni particolari che rendono l'apprendimento più efficace.

In totale, questa fase ha richiesto svariati giorni sulla piattaforma Colab.

#### 4.1.5 Risultati ottenuti

I risultati ottenuti dalla rete al termine della fase di addestramento si sono rivelati insoddisfacenti poiché l'accuratezza raggiunta è stata intorno al 50%, ovvero la stessa che si otterrebbe da un classificatore casuale.

Questo tipo di approccio si è perciò dimostrato non efficace.

## 4.2 Modello di AIZOOTech

Avendo constatato che l'addestramento di una nuova rete neurale con transfer learning non ha prodotto i risultati sperati, si è deciso di cambiare totalmente approccio. È stata perciò individuata una rete preaddestrata [3] appositamente con lo scopo di individuare persone all'interno delle immagini e di stabilire se esse indossano o meno la mascherina medica.

La rete è stata implementata in diversi framework, tra cui Tensorflow e Tensorflow Lite. Il suo addestramento è stato effettuato sul dataset **FaceMask-Dataset** contenente un totale di circa 8000 immagini (il dataset è disponibile al link GitHub) già suddiviso in training set e validation set.

Rispetto alla soluzione precedente, è ora possibile individuare più persone all'interno dell'immagine e stabilirne le coordinate. Inoltre, sono già implementate alcune funzioni in grado di contornare i volti con colori diversi a seconda della classificazione eseguita.

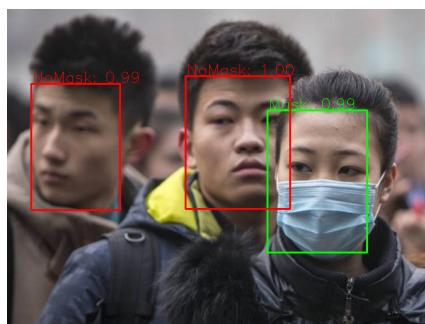


Figura 4.5: Dimostrazione del funzionamento della rete AIZOO.

#### 4.2.1 Architettura della rete

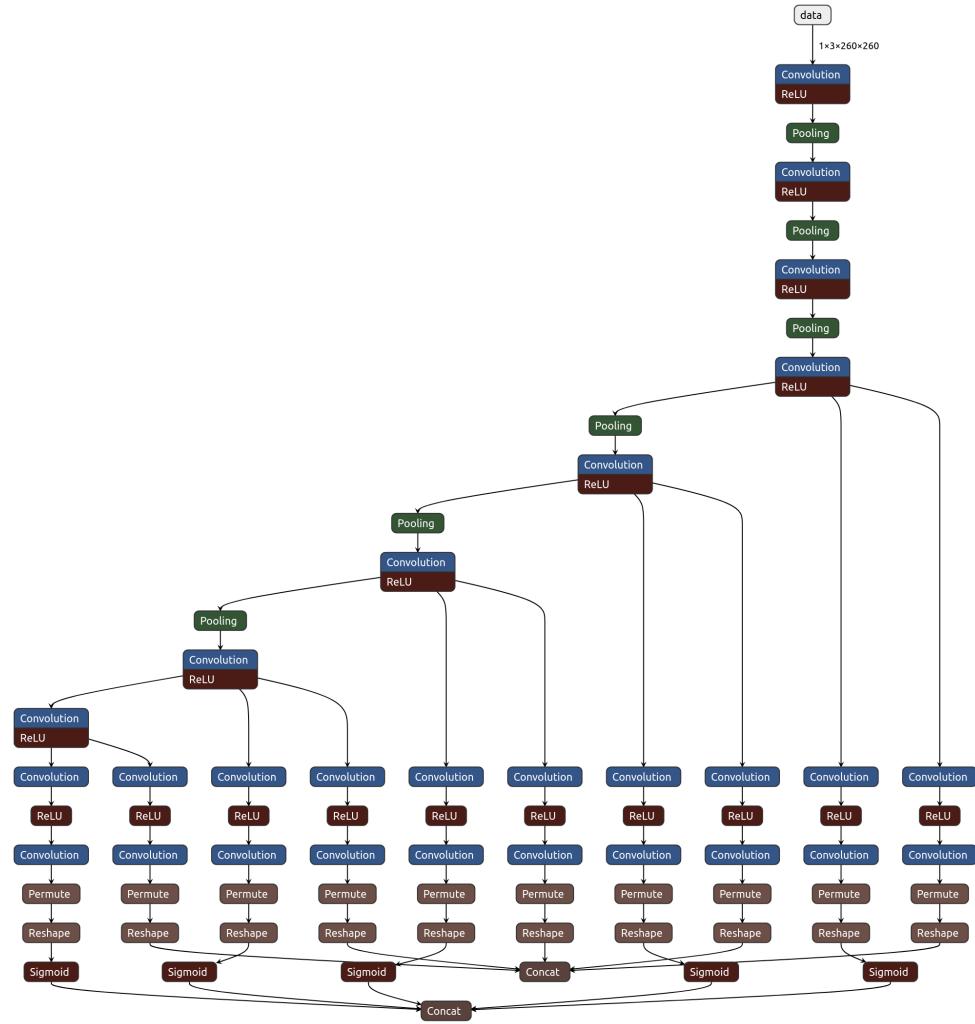


Figura 4.6: Architettura della rete AIZOO.

L'architettura della rete è costituita principalmente da strati convoluzionali per estrarre le caratteristiche dalle immagini e strati di pooling per ridurne le dimensioni. Come si può notare dalla figura 4.6 utilizza numerosi flussi separati per poi concatenarne nella parte finale gli output.

La rete lavora con immagini di dimensione 260x260 pixel e raggiunge un'accuratezza maggiore con codifica BGR al posto della classica RGB.

### 4.2.2 Funzionamento della soluzione

Le immagini che vogliamo classificare devono subire un pre-processamento per essere adattate all'input accettato dalla rete neurale. Questa fase consiste in:

- Conversione dell'immagine da codifica RGB a codifica BGR.
- Ridimensionamento dell'immagine a 260x260 pixel.
- Normalizzazione del valore dei pixel dal range 0-255 al range 0-1.

A questo punto l'immagine pre-processata sarà passata in input alla rete neurale che restituirà dei valori matriciali.

Infine, viene eseguito un post-processamento dei risultati ottenuti applicando alcune funzioni tipiche in visione artificiale tra le quali la **non-maximum suppression (NMS)** che permette di eliminare tutti i rilevamenti superflui mantenendo solo quelli più significativi.

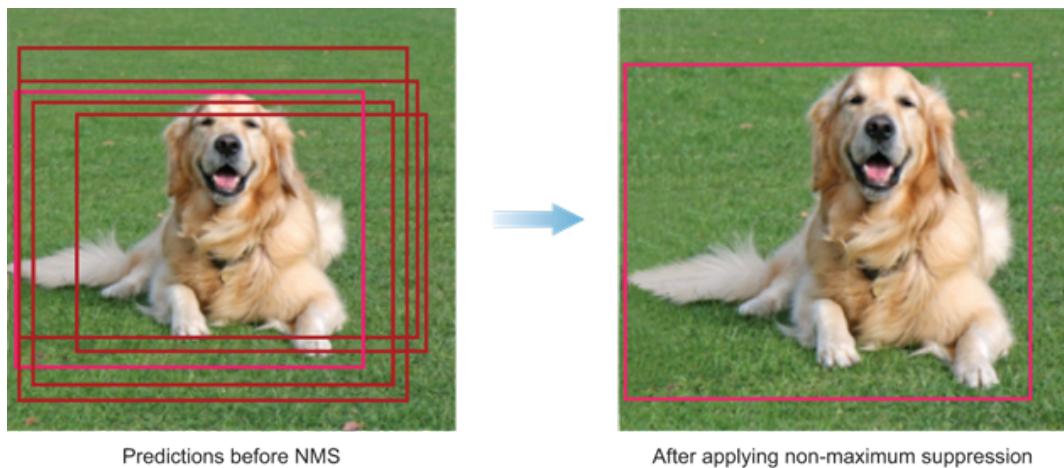


Figura 4.7: Applicazione della *non-maximum suppression*.

Il risultato finale contiene quindi le coordinate dei volti di tutte le persone individuate, la classe di appartenenza ("con mascherina" o "senza mascherina") e con quale probabilità è stata classificata tale (vedi figura 4.5).

### 4.2.3 Efficacia della soluzione

Il modello è stato testato sul validation set ottenuto dal dataset **FaceMaskDataset**, ovvero su immagini che la rete non aveva mai visto prima. Le metriche utilizzate per misurare l'efficacia sono:

- **Accuratezza:** indica la percentuale di predizioni corrette sul numero di predizioni totali. Il valore ottenuto è 94%.
- **Precisione:** indica la probabilità che l'avvenuta predizione di una classe eseguita dalla rete sia corretta. Sono stati ottenuti i seguenti risultati:

	Senza mascherina	Con mascherina
Precisione	96%	93%
Recall	92%	93%

Tabella 4.1: Precisione su FaceMaskDataset in Colab.

Si nota quindi che il modello è leggermente più preciso nel riconoscimento di persone senza mascherina.

- **Recall:** al contrario della precisione, indica la percentuale di elementi effettivamente appartenenti ad una classe che vengono riconosciuti come tali. I valori ottenuti sono:

	Senza mascherina	Con mascherina
Recall	92%	93%
F <sub>1</sub> Score	94%	93%

Tabella 4.2: Recall su FaceMaskDataset in Colab.

Notiamo qui che la differenza tra le due classi è esigua.

- **F<sub>1</sub> Score:** è una misura di accuratezza che viene calcolata come media armonica della precisione e della recall. Si ottengono quindi:

	Senza mascherina	Con mascherina
F <sub>1</sub> Score	94%	93%
Macro F <sub>1</sub> Average	93.5%	

Tabella 4.3: F<sub>1</sub> Score su FaceMaskDataset in Colab.

Il modello è stato inoltre testato sul dataset **RMFD** e i risultati sono stati molto simili a quelli appena descritti.

#### 4.2.4 Limiti della soluzione

La soluzione individuata risulta quindi efficace ma presenta dei limiti: il modello si comporta egregiamente su immagini relativamente semplici. Quando il sistema sarà utilizzato, dovrà essere in grado di riconoscere e classificare correttamente delle persone in situazioni più complesse: persone posizionate di profilo, lontane dall'inquadratura, condizioni di luce non ottimali e altre situazioni che non vengono correttamente gestite da questa soluzione.

#### 4.2.5 Possibili soluzioni

Per migliorare la rete neurale si potrebbe procedere modificandone la sua struttura e/o riaddestrandola utilizzando anche immagini che contengono situazioni difficili. Tuttavia, per fare ciò sono necessarie buone conoscenze nell'ambito della progettazione delle reti e, soprattutto, il procedimento richiederebbe una significativa quantità di tempo. Inoltre, il conseguimento di un risultato soddisfacente non è garantito.



# Capitolo 5

## Analisi della soluzione definitiva

Il questo capitolo verrà analizzata la soluzione definitiva individuata per questo elaborato di tesi. La soluzione è disponibile su GitHub ed è pronta per essere eseguita sia sui calcolatori ordinari che su micro-controllori sfruttando una videocamera.

### 5.1 Funzionamento della soluzione

Il funzionamento della soluzione si basa su due fasi:

- **Rilevamento dei volti:** una rete neurale addestrata per il rilevamento dei volti individua le coordinate di questi all'interno dell'immagine e poi invia i riquadri che li contengono al successivo modulo del progetto.
- **Classificazione dei volti:** una volta ottenuti i riquadri con i volti, una seconda rete neurale addestrata per la classificazione in "con mascherina" e "senza mascherina" eseguirà l'inferenza per ognuno di essi e restituirà la classe di appartenenza.

La soluzione utilizza quindi due reti neurali allo stato dell'arte: ciò permette sia un migliore rilevamento dei volti che una migliore accuratezza nella classificazione delle istanze rilevate.

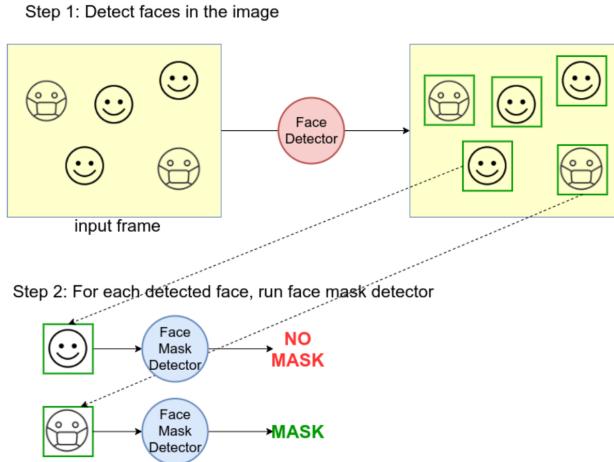


Figura 5.1: Struttura della soluzione finale.

## 5.2 Rilevamento dei volti

La rete utilizzata per il rilevamento dei volti è **Dual Shot Face Detector (DSFD)** [17] [18]. Si tratta di una rete allo stato dell’arte che presenta un’architettura particolare: è infatti costituita, come suggerisce il nome, da due flussi separati. In particolare, l’immagine viene all’inizio analizzata dalla prima catena di rilevamento costituita da strati convoluzionali; le mappe delle caratteristiche generate dagli strati vengono poi trasferite alla seconda catena mediante un modulo, denominato "intensificatore di caratteristiche" dove verranno nuovamente analizzate. Vi sono quindi una fase di analisi delle caratteristiche originali ed una di analisi delle caratteristiche intensificate come mostrato in figura 5.2.

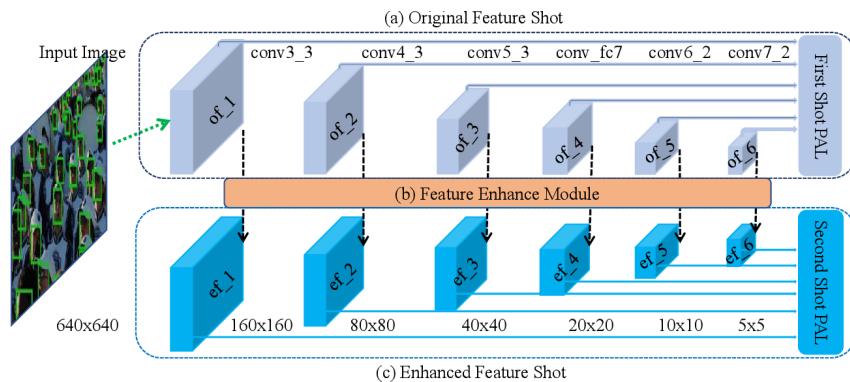


Figura 5.2: Architettura della rete Dual Shot Face Detector.

Tale rete è stata addestrata sfruttando un training set proveniente da **Wider Face** [19], ovvero un popolare dataset per il rilevamento dei volti, e successivamente testata su un test set ricavato dallo stesso dataset (ovviamente su immagini diverse da quelle utilizzate in fase di addestramento). I risultati ottenuti hanno confermato la validità della soluzione e la sua superiorità rispetto alle altre proposte allo stato dell'arte.

### 5.3 Classificazione dei volti

Per la classificazione dei volti nelle due classi "con mascherina" e "senza mascherina", è stata utilizzata una particolare rete [20] creata con transfer learning sfruttando l'architettura **MobileNet V2** [21]. Si tratta di un modello pensato per essere eseguito su dispositivi con limitate risorse computazionali grazie alla presenza di un numero ridotto di parametri.

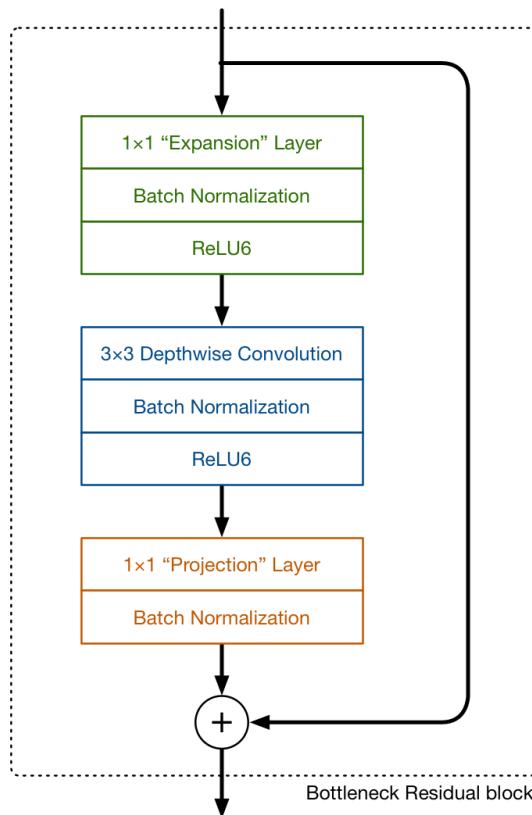


Figura 5.3: Singolo blocco di strati della rete MobileNet.

Come mostra la figura 5.3, i principali blocchi costitutivi della rete sono caratterizzati da tre strati convoluzionali:

- **Strato di espansione:** strato convoluzionale che permette di aumentare il numero di canali dei dati in ingresso prima che arrivino alla convoluzione separabile.
- **Strato di convoluzione separabile:** strato che, come già descritto in precedenza, ricerca prima dei pattern su ogni canale in ingresso e successivamente pattern tra canali diversi.
- **Strato di proiezione:** strato convoluzionale che riduce considerevolmente il numero di canali dei dati in ingresso.

Questa particolare struttura fa sì che i dati in ingresso e uscita di ogni blocco siano tensori con poche dimensioni mentre la fase di estrazione delle caratteristiche, che avviene al suo interno, sia eseguita su tensori ad alta dimensionalità.

Altri componenti del blocco sono:

- **Batch Normalization** [22]: esegue una normalizzazione dei dati in ingresso. Viene utilizzata per rendere più stabile l'apprendimento della rete ed evitare lo svanire e l'esplosione del gradiente durante la retro-propagazione.
- **ReLU6**: si tratta di una particolare versione della funzione di attivazione ReLu. Nel dettaglio, limita il valore dell'attivazione a 6.

$$\sigma(x) = \min(\max(0, x), 6)$$

Da notare inoltre l'utilizzo nel blocco di una connessione residua.

## 5.4 Efficacia della soluzione

La soluzione è stata testata sullo stesso dataset utilizzato per la precedente soluzione, ovvero **FaceMaskDataset**.

Sono state inoltre introdotte alcune misure di efficacia che aderiscono a standard internazionali:

- **Hamming Loss:** è la percentuale di istanze che vengono classificate in modo *non* corretto (in questo caso  $1 - Accuratezza$ ).
- **Macro  $F_1$  Average:** è la media aritmetica dell' $F_1$  Score per le singole classi.
- **Exact Match:** è la percentuale di immagini in cui tutte le persone vengono rilevate e correttamente classificate.

	<b>Senza mascherina</b>	<b>Con mascherina</b>
<b>Precisione</b>	95%	88%
<b>Recall</b>	93%	91%
<b><math>F_1</math> Score</b>	94%	90%
<b>Accuratezza</b>	92%	
<b>Hamming Loss</b>	8%	
<b>Macro <math>F_1</math> Average</b>	92%	
<b>Exact Match</b>	73%	

Tabella 5.1: Misure di accuratezza su FaceMaskDataset in Colab.

A primo impatto pare che la nuova soluzione non porti dei tangibili miglioramenti. Tuttavia, possiamo apprezzare le capacità di questo modello se applicato al dataset **HardDataset**, ovvero contenente immagini con condizioni non ottimali:

	<b>Senza mascherina</b>	<b>Con mascherina</b>
<b>Precisione</b>	77% (vs 47%)	90% (vs 89%)
<b>Recall</b>	83% (vs 88%)	86% (vs 49%)
<b><math>F_1</math> Score</b>	80% (vs 62%)	88% (vs 63%)
<b>Accuratezza</b>	85% (vs 62%)	
<b>Hamming Loss</b>	15%	
<b>Macro <math>F_1</math> Average</b>	84%	
<b>Exact Match</b>	23%	

Tabella 5.2: Misure di accuratezza su HardDaset in Colab.

### 5.4.1 Efficacia per numero di persone

Di seguito è riportata l'efficacia della soluzione all'aumentare del numero di persone all'interno delle immagini.

**1 persona**

	<b>Senza mascherina</b>	<b>Con mascherina</b>
<b>Precisione</b>	94%	92%
<b>Recall</b>	92%	94%
<b><math>F_1</math> Score</b>	93%	92%
<b>Accuratezza</b>	93%	
<b>Hamming Loss</b>		7%
<b>Macro <math>F_1</math> Average</b>		92.5%
<b>Exact Match</b>		84%

Tabella 5.3: Misure di accuratezza con una persona per immagine.

**2 persone**

	<b>Senza mascherina</b>	<b>Con mascherina</b>
<b>Precisione</b>	93%	83%
<b>Recall</b>	92%	84%
<b><math>F_1</math> Score</b>	92%	84%
<b>Accuratezza</b>	89%	
<b>Hamming Loss</b>		11%
<b>Macro <math>F_1</math> Average</b>		88%
<b>Exact Match</b>		56%

Tabella 5.4: Misure di accuratezza con due persone per immagine.

**3 persone**

	<b>Senza mascherina</b>	<b>Con mascherina</b>
<b>Precisione</b>	93%	76%
<b>Recall</b>	91%	81%
<b><math>F_1</math> Score</b>	92%	78%
<b>Accuratezza</b>	89%	
<b>Hamming Loss</b>	11%	
<b>Macro <math>F_1</math> Average</b>	85%	
<b>Exact Match</b>	28%	

Tabella 5.5: Misure di accuratezza con tre persone per immagine.

**4 persone**

	<b>Senza mascherina</b>	<b>Con mascherina</b>
<b>Precisione</b>	97%	85%
<b>Recall</b>	94%	93%
<b><math>F_1</math> Score</b>	96%	89%
<b>Accuratezza</b>	94%	
<b>Hamming Loss</b>	6%	
<b>Macro <math>F_1</math> Average</b>	92.5%	
<b>Exact Match</b>	33%	

Tabella 5.6: Misure di accuratezza con quattro persone per immagine.

Non sono qui riportate le misure ricavate da immagini contenenti più persone poiché le classi non venivano adeguatamente rappresentate: vi sono infatti troppe poche immagini contenenti più di 5 persone in cui ne sia presente almeno una con mascherina. Ciò renderebbe qualsiasi misura non significativa.



# Capitolo 6

## Deep Learning su micro-controllori

Si procede con la descrizione del framework utilizzato per l'implementazione della soluzione su micro-controllori. Difatti, con l'aumento di prestazioni di tali dispositivi e la riduzione dei costi, il loro impiego per progetti di deep learning risulta sempre più appetibile. Invero, sono andati sviluppandosi svariati framework e librerie con l'obiettivo di facilitare questi compiti.

### 6.1 Tensorflow Lite

Tensorflow Lite [23] è un framework per deep learning open-source e multi-piattaforma che converte modelli Tensorflow pre-addestrati in uno speciale formato (**.tflite**) ottimizzato in termini di prestazioni e spazio occupato. I modelli .tflite possono essere eseguiti su nodi edge come dispositivi mobili o micro-controllori.

#### 6.1.1 Vantaggi di Tensorflow Lite

- **Leggerezza:** i nodi edge dispongono di risorse limitate in termini di spazio di archiviazione e capacità computazionale perciò i modelli devono essere leggeri.
- **Bassa latenza:** dato che le inferenze vengono effettuate direttamente sul dispositivo, non è necessario contattare un server per cui si eliminano le latenze derivanti dalla rete.
- **Sicurezza:** di nuovo, le inferenze avvengono sul dispositivo perciò non vi sono dati che fuoriescono da esso. Ciò rende il framework sicuro.
- **Bassi consumi:** i nodi edge potrebbero avere un'autonomia limitata quindi l'utilizzo dei modelli deve richiedere poca energia.

### 6.1.2 Funzionamento di Tensorflow Lite

L'utilizzo del framework Tensorflow Lite prevede i seguenti passaggi:

- Scelta di un modello: è possibile utilizzare un modello preaddestrato tra quelli resi disponibili dalla piattaforma, riaddestrarne uno utilizzando transfer learning, addestrare un modello personalizzato o addirittura convertire una rete in formato .tflite.
- Esecuzione dell'inferenza: Tensorflow Lite utilizza una particolare libreria, denominata **interprete**, per eseguire le predizioni.

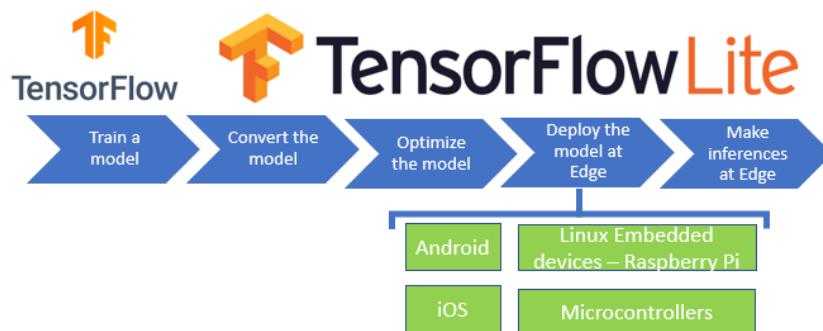


Figura 6.1: Funzionamento del framework Tensorflow Lite.

## 6.2 Descrizione dell'applicazione

L'applicazione, sviluppata in codice Python e adatta al funzionamento su micro-controllori (ma anche su macchine ordinarie), consente di rilevare e contornare i volti delle persone inquadrate dalla videocamera e stabilire se queste indossano la mascherina oppure no: in caso positivo, il riquadro sarà di colore verde, altrimenti rosso. La soluzione è ovviamente stata progettata per funzionare in tempo reale e ciò la rende adatta a situazioni di monitoraggio. Potrebbe essere sfruttata per automatizzare il controllo all'ingresso di un edificio o la situazione al suo interno e quindi tutelare chi prima svolgeva tale compito di persona. L'interfaccia consente inoltre di ottenere informazioni come il tempo di inferenza e il numero di fotogrammi analizzati al secondo.

### 6.2.1 Funzionamento dell'applicazione

Per il corretto funzionamento dell'applicazione è necessario disporre di una videocamera collegata alla propria macchina (micro-controllore o altro) e delle librerie Python necessarie: la lista è riportata in seguito. È sufficiente poi avviare l'applicazione utilizzando l'interprete Python nella versione 3 (*python3*) e inquadrare le persone di interesse. Possono ovviamente essere inquadrati più persone contemporaneamente, ferme o in movimento. Di ognuna sarà contornato il viso in verde nel caso questa indossi la mascherina, altrimenti in rosso.

Inoltre, viene eseguita anche una stima della distanza del volto dalla videocamera. Tale funzionalità è ottenuta considerando la dimensione della regione di interesse (il rettangolo che contiene il volto). Infatti, vi è un rapporto di proporzionalità inversa tra la distanza dall'inquadratura e l'area di tale regione: all'aumentare della distanza del volto dalla videocamera diminuisce la dimensione del rettangolo, e viceversa. Fornendo quindi alcuni valori di riferimento (coppie distanza-area), è possibile eseguire tale stima di distanza, non esatta ma comunque accettabile, a partire dall'area rilevata. Il valore ottenuto è mostrato al di sotto del contorno del volto.

## 6.3 Test su micro-controllori

La soluzione è quindi stata testata sui micro-controllori precedentemente descritti e di seguito sono riportati i valori di accuratezza e le prestazioni.

### 6.3.1 Raspberry Pi 4 Model B 8GB

I valori di accuratezza ottenuti risultano del tutto in linea con quelli già ricavati dai precedenti test su piattaforma Colab.

### Accuratezza FaceMaskDataset

	Senza mascherina	Con mascherina
Precisione	95%	88%
Recall	93%	91%
$F_1$ Score	94%	90%
Accuratezza		92%
Hamming Loss		8%
Macro $F_1$ Average		92%
Exact Match		73%

Tabella 6.1: Misure di accuratezza su FaceMaskDataset con Raspberry Pi.

### Accuratezza HardDataset

	Senza mascherina	Con mascherina
Precisione	76%	90%
Recall	83%	86%
$F_1$ Score	79%	88%
Accuratezza		85%
Hamming Loss		15%
Macro $F_1$ Average		83.5%
Exact Match		23%

Tabella 6.2: Misure di accuratezza su HardDataset con Raspberry Pi.

### 6.3.2 Nvidia Jetson Nano

Anche in questo caso i risultati ottenuti sono simili ai precedenti; possiamo apprezzare addirittura un lieve miglioramento.

### Accuratezza FaceMaskDataset

	Senza mascherina	Con mascherina
Precisione	94%	92%
Recall	92%	94%
$F_1$ Score	93%	93%
Accuratezza	93%	
Hamming Loss		7%
Macro $F_1$ Average		93%
Exact Match		73%

Tabella 6.3: Misure di accuratezza su FaceMaskDataset con Jetson Nano.

### Accuratezza HardDataset

	Senza mascherina	Con mascherina
Precisione	77%	90%
Recall	83%	86%
$F_1$ Score	80%	88%
Accuratezza	85%	
Hamming Loss		15%
Macro $F_1$ Average		84%
Exact Match		23%

Tabella 6.4: Misure di accuratezza su HardDataset con Jetson Nano.

### 6.3.3 Prestazioni

Di seguito sono riportate le prestazioni riscontrate per la soluzione:

	Jetson Nano	Raspberry Pi
Tempo inferenza	0.150	0.270
Frame al secondo	5.5	3

Tabella 6.5: Confronto prestazioni Jetson Nano e Raspberry Pi

Notiamo subito che Jetson Nano raggiunge delle prestazioni significativamente migliori: quasi il doppio rispetto alla sua concorrente, che comunque rimangono rispettabili dato l'utilizzo di due reti neurali. Tale risultato si può sicuramente ricondurre alle componenti migliori che la casa Nvidia ha riservato per il proprio micro-controllore.

Tuttavia, il framework Tensorflow Lite non è in grado di utilizzare la GPU Nvidia di cui è dotata la scheda Jetson Nano. Ciò significa che, nonostante la già affermata supremazia, non viene ancora sfruttato l'intero potenziale di tale dispositivo.

Si è perciò deciso di implementare la soluzione avvalendosi del framework Tensorflow. Quest'ultimo permetterà infatti di demandare del carico computazionale alla GPU in dotazione, a discapito però di una minore ottimizzazione dei modelli. Sono di seguito riportate le prestazioni ottenute da Jetson Nano con framework Tensorflow:

	<b>TF Lite</b>	<b>TF + GPU</b>	<b>TF No GPU</b>
<b>Tempo inferenza</b>	0.150	0.250	0.300
<b>Frame al secondo</b>	5.5	3.5	2

Tabella 6.6: Confronto prestazioni su Jetson Nano con framwork Tensorflow e Tensorflow Lite

Purtroppo l'utilizzo della GPU non permette di ottenere un miglioramento delle prestazioni. La spiegazione di questo fenomeno è da ricercare nella pesantezza e minore ottimizzazione dei modelli che il framework Tensorflow utilizza. Ciò fa sì che, sebbene si abbia a disposizione una maggiore capacità computazionale, le prestazioni comunque ne risentano. Di seguito sono riportate le dimensioni dei modelli utilizzati:

	<b>Tensorflow Lite</b>	<b>Tensorflow</b>
<b>Rilevamento volti</b>	580KB	6.1MB
<b>Classificatore (Face-Mask)</b>	10.2MB	11.5MB

Tabella 6.7: Confronto dimensioni modelli Tensorflow e Tensorflow Lite

È facile notare come, a parità di modelli utilizzati e numero di parametri dei modelli stessi, l'ottimizzazione del framework Tensorflow Lite permetta di ridurre la dimensione dei modelli (che diventa estremamente rilevante soprattutto per quanto riguarda la rete neurale per il rilevamento dei volti).

### 6.3.4 Nvidia Face Mask Detection

Una soluzione degna di nota è sicuramente quella fornita dall'azienda Nvidia stessa [24]. Durante la fase di test della soluzione definitiva, Nvidia ha infatti rilasciato la sua versione del sistema per il rilevamento della mascherina sanitaria facciale. Questa risulta particolarmente ottimizzata per i suoi micro-controllori, tra cui proprio Jetson Nano. Tale soluzione non fornisce un modello pre-addestrato. Tuttavia, fornisce le istruzioni per addestrare autonomamente una rete neurale *DetectNet V2* basata su *ResNet18*. Viene perciò applicata la tecnica del Transfer Learning già descritta in precedenza. In questo modo, vengono sfruttate le conoscenze già acquisite dalla rete *ResNet18* per addestrare il nuovo modello *DetectNet V2*. Per effettuare l'addestramento, Nvidia mette a disposizione diversi dataset.

I risultati riportati sono i seguenti:

Pruned	mAP (Mask/No-Mask) (%)	FPS (GPU)
No	86.12 (87.59, 84.65)	6.5
Yes (12%)	85.50 (86.72, 84.27)	21.25

Tabella 6.8: Risultati e prestazioni della soluzione Nvidia.



# Capitolo 7

## Il codice prodotto

In questo capitolo viene infine mostrato e discusso il codice dell'elaborato di tesi. Il codice è stato scritto in linguaggio Python.

### 7.1 Il codice della soluzione

Dapprima vengono importate le librerie necessarie:

```
import cv2
import tensorflow_runtime.interpreter as tflite
import numpy as np
import time
```

In particolare:

- **cv2**: libreria OpenCV per l'elaborazione delle immagini.
- **tflite\_runtime.interpreter**: interprete Tensorflow Lite per eseguire le inferenze.
- **numpy**: per gestire in modo più agevole vettori e matrici.
- **time**: per eseguire la misurazione delle prestazioni.

Successivamente, la soluzione è stata organizzata sotto forma di classe. Sono riportati di seguito la dichiarazione della stessa e il metodo costruttore:

```
class FaceMaskDetector:
    """
    Class representing the face mask detector.
    It contains all the fields and the methods that make it possible
        to make the detections.
    """

    def __init__(self):
        self.color_map = {
            0: (0, 255, 0), # Green
            1: (255, 0, 0) # Red
        }
        self.class_map = {
            0: "Mask",
            1: "No Mask"
        }

        # Definition of the Face Detection interpreter.
        self.face_target_shape = (320, 320)
        self.face_interpreter =
            tflite.Interpreter(model_path="models/detector.tflite")
        self.face_interpreter.allocate_tensors()
        self.face_input_details =
            self.face_interpreter.get_input_details()
        self.face_output_details =
            self.face_interpreter.get_output_details()

        # Definition of the Mask Detection Interpreter.
        self.mask_target_shape = (224, 224)
        self.mask_interpreter =
            tflite.Interpreter(model_path="models/mask_detector.tflite")
        self.mask_interpreter.allocate_tensors()
        self.mask_input_details =
            self.mask_interpreter.get_input_details()
        self.mask_output_details =
            self.mask_interpreter.get_output_details()
```

In questo modo vengono definite alcune proprietà della classe:

- **color\_map**: dizionario che associa alla classe predetta il colore corrispondente (verde o rosso).

- **class\_map**: dizionario che associa alla classe predetta l'etichetta corrispondente.
- Proprietà dei due interpreti (uno per il rilevamento dei volti e l'altro per la loro classificazione):
  - Dimensione dell'immagine in input.
  - Percorso del modello.
  - Definizione della struttura dei valori di ingresso e di uscita del modello.

Seguono poi alcune funzioni di utilità:

- **preprocess**: adatta le immagini ricavate dalla fotocamera in modo che possano essere utilizzate dal modello per il rilevamento dei volti.
- **py\_nms**: applica la non-maximum suppression in modo da eliminare i rilevamenti in eccesso.

Una volta applicate le trasformazioni sulle immagini in ingresso viene invocato l'interprete per eseguire le inferenze (rilevamento volti).

```
self.face_interpreter.set_tensor(self.face_input_details[0] ["index"],
    image_for_net)
self.face_interpreter.invoke()
bboxes = self.face_interpreter
    .get_tensor(self.face_output_details[0] ["index"])
```

In questo modo vengono restituite le coordinate dei riquadri contenenti i volti rilevati.

Successivamente, per ognuno di questi viene invocato in modo analogo l'interprete per eseguire la classificazione.

```
self.mask_interpreter.set_tensor(self.mask_input_details[0] ["index"],
    image_tensor)
self.mask_interpreter.invoke()
probabilities = self.mask_interpreter
    .get_tensor(self.mask_output_details[0] ["index"])
```

Vengono così restituite le classi predette con relative probabilità.

Infine, è riportato di seguito il corpo principale della soluzione: il ciclo che permette di ottenere immagini dalla fotocamera ed eseguire le inferenze.

```
while rval:  
    # Lettura di un frame dalla videocamera  
    rval, frame = vc.read()  
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  
  
    inference_start = time.time()  
    # Rilevamento volti  
    boxes = face_mask_detector.detect_faces(frame)  
    # Classificazione volti  
    frame = face_mask_detector.detect_masks(frame, boxes)  
    inference_stop = time.time()
```

# Conclusioni e sviluppi futuri

L'obiettivo iniziale consisteva nello sviluppo di un sistema in grado di determinare se le persone indossassero o meno una mascherina sanitaria facciale.

Dopo aver sperimentato diversi approcci e confrontato i rispettivi risultati, si è giunti alla conclusione che la soluzione costituita dalle reti *Dual Shot Face Detector* (per il rilevamento dei volti) e *MobileNet V2* (per la loro classificazione) si dimostra essere la migliore in termini di accuratezza. Questa soluzione garantisce delle prestazioni tali da poter essere sfruttata anche nel campo dei micro-controllori. Inoltre, l'analisi delle prestazioni, come prevedibile, ha evidenziato la superiorità del micro-controllore Jetson Nano, che tuttavia non ha soddisfatto al pieno le aspettative, posto che la presenza di una GPU avrebbe dovuto ulteriormente migliorarne le prestazioni.

La soluzione individuata è stata adattata al funzionamento su pc Desktop e su micro-controllori e permette, sfruttando una tradizionale webcam, di eseguire delle predizioni in tempo reale. Ciò la rende adatta ad essere utilizzata in vari contesti sostituendosi all'uomo.

Questo lavoro si presta a svariati sviluppi e tra i tanti ad un sistema per il controllo del distanziamento sociale. Ciò permetterebbe di stimare il rischio di contagio basandosi sulla distanza tra le persone e l'utilizzo della mascherina.

Un futuro in cui la macchina si potrà affiancare o addirittura sostituire all'uomo, diventando di fatto "i suoi occhi", rappresenta uno scenario suggestivo. E diventa tanto più stimolante contribuire, in un periodo come quello che ci occupa, ad addivenire a soluzioni con fini socialmente utili. Ciò rende questa disciplina ancor più affascinante di quanto lo sia già.



# Ringraziamenti

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

Grazie alla mia famiglia, che mi ha sostenuto, spinto a dare il massimo e mi è sempre stata vicina.

Ringrazio inoltre amici e colleghi che mi hanno accompagnato durante il percorso di studi e hanno reso questa esperienza unica.



# Bibliografia

- [1] Nancy Leung, Daniel Chu, Eunice Shiu, Kwok-Hung Chan, James Mcdevitt, Ben Hau, Hui-Ling Yen, Yuguo Li, Dennis Ip, Joseph S Peiris, Wing-Hong Seto, Gabriel Leung, Donald Milton, and Benjamin Cowling. Respiratory virus shedding in exhaled breath and efficacy of face masks. *Nature Medicine*, 26, 05 2020.
- [2] Real world masked face dataset. <https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset>.
- [3] Aizoootech - face mask detection. <https://github.com/AIZOOTech/FaceMaskDetection>.
- [4] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.
- [5] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657, 2015.
- [6] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3642–3649, 2012.
- [7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings.
- [8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations*

- in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.
- [9] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat’s striate cortex. *Journal of Physiology*, 148:574–591, 1959.
  - [10] David H. Hubel and Torsten N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195:215–243, 1968.
  - [11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
  - [12] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.
  - [13] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Comput. Sci. Eng.*, 13(2):22–30, 2011.
  - [14] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2016. cite arxiv:1610.02357.
  - [15] 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.
  - [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. cite arxiv:1512.03385Comment: Tech report.
  - [17] Jian Li, Yabiao Wang, Changan Wang, Ying Tai, Jianjun Qian, Jian Yang, Chengjie Wang, Ji-Lin Li, and Feiyue Huang. Dsfd: Dual shot face detector. *CoRR*, abs/1810.10220, 2018.
  - [18] Dual shot face detection - tensorflow. <https://github.com/610265158/DSFD-tensorflow/tree/tf2>.
  - [19] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. *CoRR*, abs/1511.06523, 2015.
  - [20] Face mask classifier. <https://github.com/estebanuri/facemaskdetector/tree/master/android>.

- [21] 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.
- [22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. cite arxiv:1502.03167.
- [23] Tensorflow lite. <https://www.tensorflow.org/lite>.
- [24] Nvidia-ai-iot - face mask detection. <https://github.com/NVIDIA-AI-IOT/face-mask-detection>.