ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

SPERIMENTAZIONE E CONFRONTO DI SOLUZIONI PER IL RILEVAMENTO DELLA MASCHERINA SANITARIA FACCIALE

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Presentata da
Matteo Andreotti

Terza Sessione di Laurea
Anno Accademico 2019 – 2020

PAROLE CHIAVE

Machine Learning

Deep Learning

Convolutional Neural Networks

Image Classification

Python

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

Introduzione

Quando nel 1943 il neurofisiologo Warren S. McCulloch e il matematico Walter Pitts pubblicarono l'articolo "A Logical Calculus of the Ideas Immanent in Nervous Activity" [1] nel Bollettino della Biofisica Matematica, in cui discussero le reti di neuroni artificiali semplificati e di come esse potrebbero eseguire semplici funzioni logiche, probabilmente non si immaginavo quanto tale articolo sarebbe diventato l'ispirazione per lo sviluppo del **Deep Learning**.

Così come quando nel 1950 il famoso matematico Alan Turing creò il Test di Turing, che permetteva di determinare se una macchina fosse in grado di comportarsi intelligentemente, e tale scopo veniva raggiunto tramite un'interrogatorio in cui un umano doveva riconoscere se stesse parlando con un altro umano oppure con una macchina (spiegazione dettagliata pubblicata nell'articolo "Computing Machinery and Intelligence" [2]), non immaginava quanti passi in avanti avrebbe fatto da lì in poi l'Intelligenza Artificiale, dalla quale poi negli anni si svilupparono diversi sottoinsiemi, tra cui il già citato Deep Learning ed il **Machine Learning**.

Il Machine Learning è un insieme di tecniche che consento di creare sistemi di Intelligenza Artificiale, e per farlo si serve di algoritmi addestrati sui dati. Tali sistemi sono in grado di apprendere e migliorare le prestazioni in base ai dati che utilizzano. Una sempre più costante crescita della quantità e varietà di dati a disposizione, lo sviluppo di architetture sempre più potenti dal punto di vista computazionale e la presenza di spazi per l'archiviazione dei dati sempre più ampi e a buon mercato, ha favorito il progresso del machine learning. Esso permette ad esempio di classificare le recensioni relative a un film o a un prodotto in positive o negative, a stimare il prezzo di un'abitazione, oppure ancora a predire le propensione di acquisto di un utente.

Il Deep Learning è il cugino meno conosciuto delle tecnologie sopra citate, nonché una branca del machine learning. I modelli di deep learning imparano a eseguire attività di classificazione direttamente da immagini, testi o persino suoni. In questo caso non è necessario avere a disposizione dei dati strutturati, in quanto il sistema si basa su **reti neurali** artificiali organizzate in diversi strati, che combinano diversi algoritmi e sono modellate sul cervello umano. Ciò consente al sistema di elaborare anche dati non strutturati.

Il deep learning viene utilizzato ad esempio per le auto a guida autonoma, le quali devono essere in grado di riconoscere altri veicoli, segnali stradali e pedoni. Quindi è impiegato per il riconoscimento facciale ed è alla base dei dispositivi di assistenza vocale.

Il campo del Machine learning sta vivendo oggi una crescita esponenziale, soprattutto nel tema della visione artificiale. Rispetto al 2011 infatti, in cui il tasso di errore da parte degli esseri umani nella visione artificiale era del 26%, oggi è solo del 3%. Ciò significa che i computer sono già più efficaci nel riconoscere e analizzare le immagini rispetto agli esseri umani.

L'obiettivo di questa tesi va incontro a quelle che sono le esigenze sanitarie del momento storico che stiamo vivendo, ovvero quello segnato dalla pandemia di Covid-19. Per farlo si andranno ad utilizzare le tecnologie sopra nominate allo scopo di individuare mascherine sui volti delle persone, per stabilire se esse indossano correttamente o meno la mascherina protettiva come richiesto dalle misure di prevenzione anti-contagio.

Verranno confrontate diverse soluzioni che permettono di raggiungere questo traguardo, ciascuna costruita in modo diverso e con architetture differenti che saranno analizzate nel dettaglio.

Il lavoro di tesi è stato suddiviso nei seguenti capitoli:

- **Capitolo 1** - Analisi preliminare sulla struttura dei dati all'interno dei dataset contenenti;
- **Capitolo 2** - Panoramica sulle varie tecnologie esistenti in letteratura utilizzabili per il problema posto;
- **Capitolo 3** - Introduzione alla modellazione del problema con i primi approcci alle difficoltà riscontrate e alle relative soluzioni;
- **Capitolo 4** - Metodi di risoluzione dei problemi riscontrati e spiegazione dei diversi approcci utilizzati;
- **Capitolo 5** - Gli esperimenti realizzati e il codice impiegato.

Indice

Elenco delle figure

Capitolo 1

Analisi dei dati forniti

In questa parte vengono analizzati i dataset forniti e il modo in cui sono state strutturate le immagini all'interno di essi. In questo modo viene fornita una visione più specifica del lavoro che verrà svolto e verranno definiti gli obiettivi del progetto.

1.1 Contesto applicativo

A seguito di un aumento sempre più consistente del numero di contagiati in in un vasto numero di territori del globo, l'Organizzazione Mondiale della Sanità ha stabilito delle misure di prevenzione per limitare la diffusione del coronavirus. Tra le misure di prevenzione principali vi sono quelle di mantenere una distanza di sicurezza di almeno 1 metro dalle altre persone, igienizzare e lavare spesso le mani e indossare una mascherina di protezione. Per quanto riguarda il nostro paese è infatti obbligatorio sull'intero territorio nazionale avere sempre con sé dispositivi di protezione delle vie respiratorie. Proprio su questa restrizione si baserà la tesi.

Per poter addestrare un sistema a imparare a riconoscere se una persona sta indossando correttamente una mascherina è necessario reperire dei dataset utili allo scopo.

Un dataset è una collezione di dati, che possono essere strutturati (quindi organizzati in forma tabellare) e non strutturati. Poiché il sistema dovrà avere a che fare con immagini, i dataset desiderati dovranno contenere immagini di persone che indossano la mascherina, ed anche che non la indossano, dato che sarà importante che il sistema impari la differenza tra i 2 casi. Per reperire i dataset un valido supporto viene fornito da Kaggle.

Kaggle è una piattaforma online fondata nel 2010 che costituisce un punto di riferimento per gli amanti del Machine Learning. È possibile partecipare a delle competizioni messe a disposizione da Kaggle per confrontarsi con altri data scientist, e che consento anche di aggiudicarsi un premio in denaro. Queste competizioni coinvolgono vari tipologie di analisi: dalla regressione alla classificazione e altro ancora. Per poter dare inizio a ciascuna di queste competizioni però sono necessari dei dati da cui partire. Questi dati costituiscono i dataset, che possono di vario tipo: testuali, immagini, ecc.

Le immagini all'interno dei dataset cercati sono divise in due classi, rappresentate dalle cartelle, una per le immagini dei volti di persone che stanno indossando una mascherina protettiva, e una per i volti senza alcuna mascherina. Tali cartelle verranno utilizzate per assegnare delle etichette alle immagini, che serviranno alla rete che verrà costruita per capire quale tipo di immagine sta analizzando.

Oltre ai dataset essenziali per addestrare la rete, saranno necessari anche dei dataset che fungeranno da testset, ovvero che saranno impiegati per verificare quanto le soluzioni proposte abbiano imparato a distinguere tra un volto con la mascherina ed uno senza, e quanto sia invece l'errore commesso.

Nelle prossime sezioni verranno discussi nel dettaglio i contenuti dei dataset e testset.

1.2 Dataset per l'addestramento

Dopo aver cercato sulle piattaforme discusse le informazioni necessarie per poter raggiungere gli obiettivi preposti, sono stati individuati 3 dataset ottimali per l'addestramento delle reti neurali.

In ciascun dataset ogni immagine contenuta rappresenta il volto di una persona. A seconda che il volto nell'immagine indossi o meno una mascherina, essa viene archiviata in un'apposita cartella, che rappresenta sostanzialmente la classe a cui appartiene l'immagine.

Due dei dataset, inoltre, sono già correttamente divisi in immagini di training ed immagini di validation, ovvero che serviranno rispettivamente per addestrare la rete e per verificarne l'accuratezza, mentre il terzo racchiude tutte le immagini in un'unica cartella, sempre divise però tra con e senza mascherina. Per cui sarà necessario procedere manualmente alla loro suddivisione.

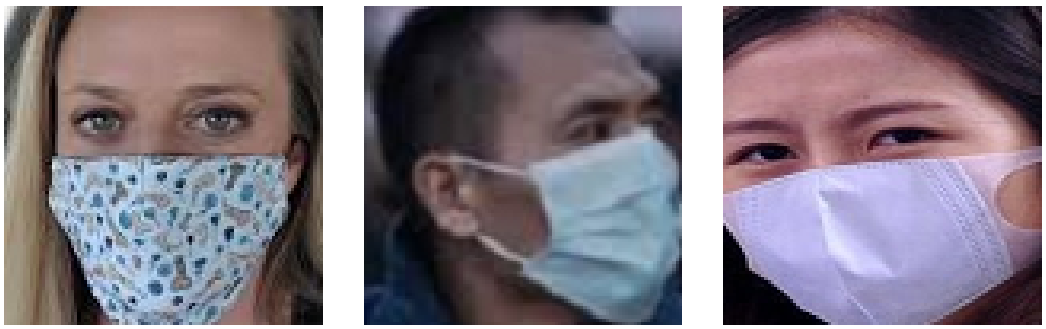
Mentre nei primi 2 dataset le immagini contengono principalmente volti appartenenti a persone occidentali, nel 3 ° dataset c'è la prevalenza di volti di persone asiatiche.

I 3 dataset contengono in totale 27.809 immagini, così suddivise:

- Nel 1° Dataset abbiamo un totale di 9456 immagini, suddivise in 8019 di training, di cui 4408 con mascherina e 3611 senza mascherina, e 1437 di validation, di cui 777 con mascherina e 660 senza mascherina.

- Nel 2° Dataset abbiamo un totale di 10800 immagini, suddivise in 10.000 immagini di training, di cui 5000 con mascherina e altre 5000 senza mascherina, e in 800 immagini di validation, di cui 400 con mascherina e altre 400 senza mascherina.

- Infine, nel 3° Dataset abbiamo un totale di 7553 immagini, suddivise in 5000 immagini di training, di cui 2500 con mascherina e 2500 senza mascherina, e 2553 di validation, di cui 1225 con mascherina e 1328 senza mascherina.



(a) 1° DS. (b) 2° DS. (c) 3° DS.

Figura 1.1: Campione di volti con mascherina dei dataset di addestramento.

(a) 1° DS. (b) 2° DS. (c) 3° DS.

Figura 1.2: Campione di volti senza mascherina dei dataset di addestramento.

1.3 Dataset per il testing

I testset sono strutturati in modo diverso rispetto ai dataset usati per l'addestramento. Essi infatti contengono immagini di persone complete, ovvero con anche busto e gambe rappresentate in foto. Benché vi sono comunque immagini che racchiudono anche persone tagliate, di cui ossia è raffigurata solamente la parte superiore del corpo, comprendendo sempre a prescindere il volto.

Inoltre, in ogni immagine posso essere presenti un numero variabile di persone, ciascuna con una propria distanza rispetto alla telecamera, che costituisce una difficoltà ulteriore nel classificare il volto.

All'interno dei testset sono presenti due cartelle, una contenente tutte le immagini e una contenente le etichette. Il nome del file che rappresenta un'etichetta è lo stesso del nome dell'immagine a cui si riferisce. Le etichette sono dei file in formato xml, ciascuno dei quali contiene un numero di elementi pari al numero di persone inquadrate nell'immagine che rappresenta il file. Ogni elemento descrive se sul volto è presente una mascherina o meno e i bounding box del volto della persona a cui si riferisce, ovvero le coordinate in termini di pixel in cui è possibile trovare il relativo volto nell'immagine.

Ad esempio nella cartella che include tutte le immagini, un campione di esse potrebbe avere nome "image.jpg" ed essere come segue:

Figura 1.3: Esempio immagine contenuta in un testset.

Il relativo file *etichetta* avrà lo stesso nome dell'immagine, ovvero "image.xml", e descriverà dove si trovano i volti delle persone nell'immagine in termini di coordinate, e se indossano o meno la mascherina.

```
<annotation>
  <folder>people_in_medical_masks</folder>
  <filename>test_image.jpg</filename>
  <source>
    <database>people_in_medical_masks</database>
  </source>
  <size>
    <width>1600</width>
    <height>900</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>mask</name>
    <bndbox>
      <xmin>1</xmin>
      <ymin>263</ymin>
      <xmax>104</xmax>
      <ymax>373</ymax>
    </bndbox>
  </object>
  <object>
  ...
  </object>
  <object>
    <name>none</name>
    <bndbox>
      <xmin>301</xmin>
      <ymin>151</ymin>
      <xmax>416</xmax>
      <ymax>243</ymax>
    </bndbox>
  </object>
```

Figura 1.4: Esempio etichetta di un immagine contenuta in un testset

In questo esempio, l'elemento *size* specifica la dimensione spaziale dell'immagine, ossia larghezza, altezza e profondità, quest'ultima intesa come scala di colori (RGB). Ogni elemento *object* indica una persona. Il sottoelemento *name* specifica se la persona indossa (mask) la mascherina o se non la indossa (none). Infine il sottoelemento *bndbox* specifica le coordinate in cui è racchiuso il volto della persona di riferimento. La stessa struttura la si ritroverà in tutti gli altri file di etichette.

I testset contengono in totale 2.001 immagini, da cui vengono estratti rispettivamente 5.086 e 4.072 volti. Nel 1° testset sono presenti 4.151 volti con mascherina e 935 volti senza mascherina. Nel 2° testset invece i volti con la mascherina sono 3.232 mentre i volti senza sono 840.

1.4 Obiettivi

L'obiettivo principale della corrente tesi, sarà quello di mettere a confronto soluzioni basate su tecnologie diverse e individuare la soluzione migliore che permetta di stabilire se le persone inquadrate in un'immagine indossano correttamente o meno la mascherina protettiva come previsto dalle norme anti-covid.

Per raggiungere l'obiettivo verranno confrontate diverse architetture finalizzate allo stesso scopo, le quali si basano sui principi di cui si è discusso, dandone una prima infarinatura, precedentemente, e che verranno invece approfonditi nel capitolo successivo.

Capitolo 2

Le tecnologie disponibili

In questa prima parte verranno descritte più nello specifico tutte le tecnologie che sono state solo accennate nell'introduzione, per rendere più chiari gli obiettivi e le metodologie di lavoro impiegate nei capitoli successivi.

2.1 Machine Learning

Il Machine Learning è un insieme di tecniche che consento di creare sistemi di Intelligenza Artificiale, e per farlo si serve di algoritmi addestrati sui dati. Tali sistemi sono in grado di apprendere e migliorare le prestazioni in base ai dati che utilizzano. Una sempre più costante crescita della quantità e varietà di dati a disposizione, lo sviluppo di architetture sempre più potenti dal punto di vista computazionale e la presenza di spazi per l'archiviazione dei dati sempre più ampi e a buon mercato, ha favorito il progresso del machine learning.

Esso permette ad esempio di:

- classificare le recensioni relative a un film o a un prodotto in positive o negative;
- stimare il prezzo di un'abitazione;
- predire le propensione di acquisto di un utente;
- ed altro ancora.

Il Machine Learning è una branca dell'Intelligenza Artificiale, e a differenza di quest'ultima che cominciò ad avere applicazione pratica già attorno agli anni '50 del secolo scorso, il machine learning iniziò ad avere una sempre maggiore risonanza a cavallo tra gli anni '80 e '90. Come già detto in precedenza, permette di creare modelli intelligenti in grado di prendere decisioni, riducendo l'intervento dell'essere umano al minimo.

La definizione che meglio descrive il funzionamento di un sistema di machine learning, è quella data da Tom M. Mitchell, informatico e professore noto per il suo contributo al progresso del machine learning, dell'intelligenza artificiale e delle neuroscienze cognitive, e riportata nel suo libro "Machine Learning" [3]:

> *"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."*

Quindi in sostanza possiamo dire che un algoritmo, un sistema o un programma, riesce ad apprendere se dopo aver svolto una certa attività, le sue performance migliorano, anche grazie all'esperienza acquisita in precedenza.

Ciò rappresenta pertanto un nuovo approccio per la risoluzione di problemi, dai più semplici ai più complessi, che si riscontano in una vasta gamma di settori, dalla quotidianità agli ambiti più professionali.

Infatti, per sviluppare un modello che permetta di fare previsioni, senza però fare ricorso al machine learning, occorre basarsi sull'esperienza dell'autore, il quale dovrà individuare personalmente i pattern che permettono di classificare una certa istanza in un certo modo oppure predire un certo valore. Dopodiché scriverà un algoritmo per ciascuno di questi pattern, verificherà la bontà di tali algoritmi e ripeterà il procedimento appena descritto finché non otterrà dei risultati soddisfacenti. Sembrerebbe una vera e propria odissea. Grazie al machine learning invece tutto ciò viene estremamente semplificato ed automatizzato. Infatti, un sistema basato sul machine learning imparerà automaticamente quali sono i pattern che permettono di classificare o predire nel migliore dei modi. Questi pattern verranno estratti dai dati. L'applicazione di tecniche di machine learning atte a esplorare in grandi quantità di dati può aiutare a scoprire pattern che non erano immediatamente evidenti. Questo insieme di tecniche e metodologie è noto come **data mining**.

Perciò come già accennato, la crescita costante della quantità di dati a disposizione e la presenza di spazi d'archiviazione dei dati sempre più ampi e a buon mercato, ha favorito il progresso del machine learning.

Tutte queste potenzialità vengono sfruttate oggi per compiere attività di ogni genere: ad esempio fare ricerche su internet avvalendosi di assistenti digitali, oppure ascoltare la nostra canzone preferita semplicemente pronunciandone il nome in presenza del lettore musicale, visualizzare prodotti, mentre navighiamo, che non abbiamo cercato ma che comunque rientrano nella nostra sfera di interessi e che ci vengono consigliati dai siti web sulla base di ciò che abbiamo

comprato, visto o cercato in precedenza. I robot aspirapolvere che puliscono la casa mentre noi dedichiamo il nostro tempo in altre attività. Il servizio di posta che filtra le email, selezionando quelle indesiderate, etichettandole come "spam" e archiviandole nell'apposita cartella. Anche dalla prospettiva sanitaria il machine learning trova la sua utilità, infatti permette di analizzare immagini per aiutare i medici ad individuare possibili tumori.

2.1.1 Sviluppo di un modello

Il Machine Learning può essere applicato a qualunque problema per il quale esistano dati sufficienti a rappresentarne il contesto, con l'obiettivo di estrarre da essi un modello di conoscenza. Per estrarre tale modello in ogni sistema di intelligenza artificiale basato su machine learning, è buona norma seguire uno specifico processo, costituito da diversi fasi.

Innanzitutto occorre reperire tutti i dati necessari e sufficienti per descrivere il problema posto, dopodiché si procede ad analizzarli. Quindi se ne verifica la completezza e la qualità e si stabilisce come comportarsi in presenza di eventuali valori mancanti. Nel caso in cui i dati del problema appartengano a scale di valori molto diverse tra loro, è utile procedere con la normalizzazione dei dati, in modo che tutti abbiano valori in un medesimo intervallo.

Poiché non tutti i dati avranno la stessa importanza nel raggiungere l'obiettivo finale, ma anzi è possibile vi siano dati che potrebbero confondere il modello, si procede selezionando solamente le variabili più rilevanti rispetto appunto agli obiettivi, e tali variabili verranno definite *features*.

Una volta che un modello di conoscenza è stato addestrato, è necessario esaminare la qualità dello stesso. Questo avviene testandolo su dei dati appositi.

I dati acquisiti in precedenza, utilizzati per addestrare il modello, vengono divisi in più porzioni, una appunto per l'addestramento e una per la validazione. Con la prima individuiamo i parametri che massimizzino l'accuratezza del modello sulla seconda porzione.

Dagli stessi dati è possibile estrarre più modelli di conoscenza, i quali vanno poi interpretati, individuando i migliori.

Infine si integra il modello di conoscenza in un sistema software o in un'applicazione.

A seconda dei dati che si hanno a disposizione, i tipi di algoritmi di apprendimento possono appartenere a diverse categorie:

- Apprendimento Supervisionato;
- Apprendimento Non Supervisionato;
- Apprendimento Rinforzato.

2.1.2 Apprendimento supervisionato

Nel learning supervisionato, le istanze dei dati di training che saranno utilizzati come input per addestrare un modello, sono dotate anche dell'output, ovvero del valore che dovrebbe essere predetto dall'algoritmo. I dati si dicono così *etichettati*. Il valore di output può essere un valore discreto oppure continuo.

Nel caso di valore discreto si avrà un compito basato su tecniche di *classificazione*, mentre nel caso di valore continuo si avrà un problema di *regressione*.

Un'esempio di un possibile problema di classificazione supervisionato può essere quello di distinguere tra email spam e non spam.

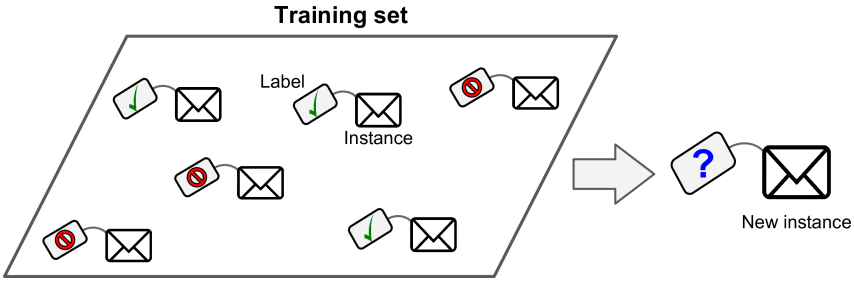


Figura 2.1: Apprendimento Supervisionato in un problema di Classificazione

In questo caso ogni istanza rappresenterà un'email diversa. Ad ognuna di esse saranno associati diversi attributi, come ad esempio il mittente, l'oggetto e il testo dell'email. Infine sarà presente un ultimo dato che esprimerà l'output da predire, ovvero se l'email è spam oppure no.

In un problema di regressione l'obiettivo è quello di predire un valore numerico. Perciò i dati avranno un attributo contenente un valore continuo che rappresenterà l'output da stimare.

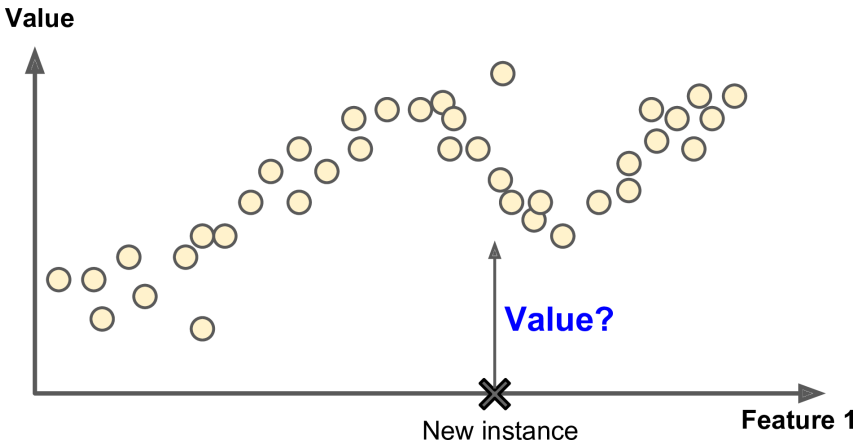


Figura 2.2: Apprendimento Supervisionato in un problema di Regressione

La figura mostra l'esempio di una regressione univariata, vale a dire un tipo di regressione in cui la variabile dipendente y da predire viene stimata sulla base di un'unica variabile dipendente x. Per cui si assume che y vari in modo direttamente proporzionale ad x.

La presenza nei dati del valore di output che dovrà essere predetto dal modello, permette a quest'ultimo di verificare l'errore commesso nella predizione, confrontando il valore predetto col valore reale. Tale errore permetterà al modello di migliorarsi.

Uno dei più importanti algoritmi di apprendimento supervisionato sono le **Reti Neurali**, che verranno discusse in seguito.

2.1.3 Apprendimento non supervisionato

Nel learning non supervisionato i dati non sono etichettati. Per cui non contengono alcun valore di output da predire, e questo rende difficile stabilire la bontà della predizione da parte dei modelli, poiché non esiste un metodo generale per verificare l'errore commesso dal modello.

Questo tipo di learning permette di estrarre dai dati informazioni e legami osservandone la struttura.

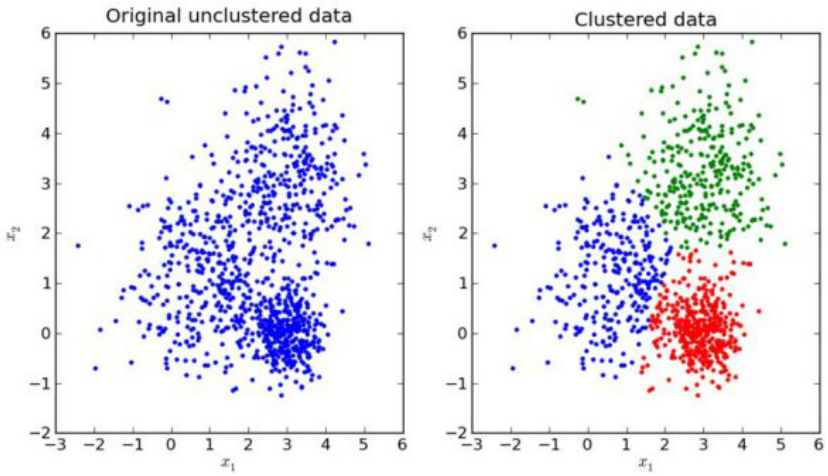


Figura 2.3: Suddivisione dei dati in cluster.
Fonte: https://www.kdnuggets.com/2017/05/must-know-most-useful-number-clusters.html

Quando ci si trova davanti a un problema di learning non supervisionato una delle possibili tecniche utilizzabili è quella del clustering, la quale permette di raggruppare i dati in gruppi detti cluster. All'interno di essi vi saranno elementi simili tra loro, con caratteristiche quindi molto affini. La tecnica del clustering consente così di scovare relazioni tra i dati.

2.1.4 Apprendimento per rinforzo

È un tipo di learning in cui vi è un sistema, detto *agente*, che interagisce con l'ambiente migliorando le proprie prestazioni. L'agente seleziona ed esegue delle azioni. A seconda della bontà delle azioni intraprese, esso riceverà delle ricompense, dette rinforzi, che possono essere positive ("*rewards*") o negative ("*penalties*"). Le ricompense permettono al sistema di migliorare le proprie performance. Quindi l'agente deve imparare da solo quale sia la miglior strategia, detta *policy*, per ottenere la ricompensa migliore. La *policy* definisce quale azione l'agente deve scegliere quando si trova in una determinata situazione.

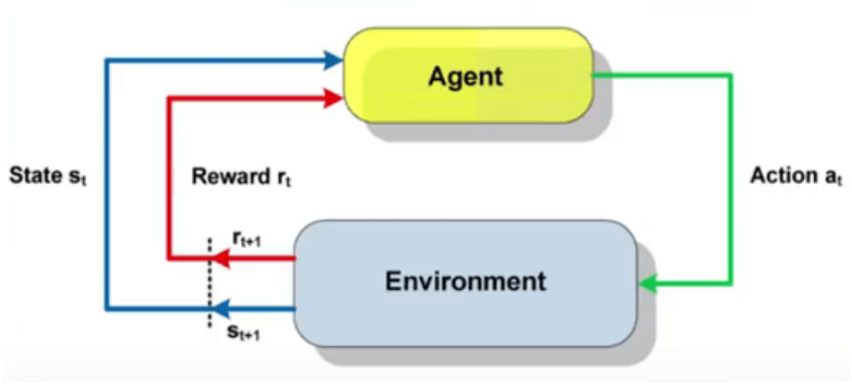


Figura 2.4: Scherma del funzionamento dell'apprendimento rinforzato.
Fonte: https://theaisummer.com/Reinforcement_learning

Un'esempio di sistema che ricorre al learning per rinforzo è quello del gioco degli scacchi. Qui il sistema deve decidere quali pedine muovere e in che direzione, e ciò rappresenta le azioni da intraprendere. Al principio le mosse effettuate non avranno dietro di esse una logica. In seguito alle decisioni intraprese però il sistema riceverà una ricompensa, in caso ad esempio in seguito all'azione abbia mangiato una pedina avversaria, mentre riceverà una penalità in caso di azione negativa. In questo modo il sistema darà maggior peso alle mosse che gli hanno portato maggiori benefici e tenderà a replicare lo stesso comportamento su nuove mosse future.

2.1.5 Discesa del gradiente

La discesa del gradiente è un algoritmo iterativo che permette di trovare i punti di massimo e di minimo locale di una funzione in più variabili, basandosi sul suo gradiente.

Il gradiente di una funzione è un vettore, le cui componenti non sono altro che le derivate parziali della funzione per ciascuna delle sue variabili.

La derivata parziale di una funzione è la derivata della funzione rispetto ad una delle sue variabili, considerando tutte le altre come costanti.

Il gradiente calcolato in un determinato punto indica l'inclinazione della curva in quello stesso punto. Per cui grazie al gradiente è possibile determinare la direzione in cui la curva sale o scende più rapidamente da un certo punto.

Partendo da un punto casuale, si valuta la funzione ed il suo gradiente. Si sottrae al punto un vettore proporzionale al gradiente calcolato, ottenendo così un nuovo punto. Dopodiché si ripete il procedimento fino a che non si converge ad un minimo.

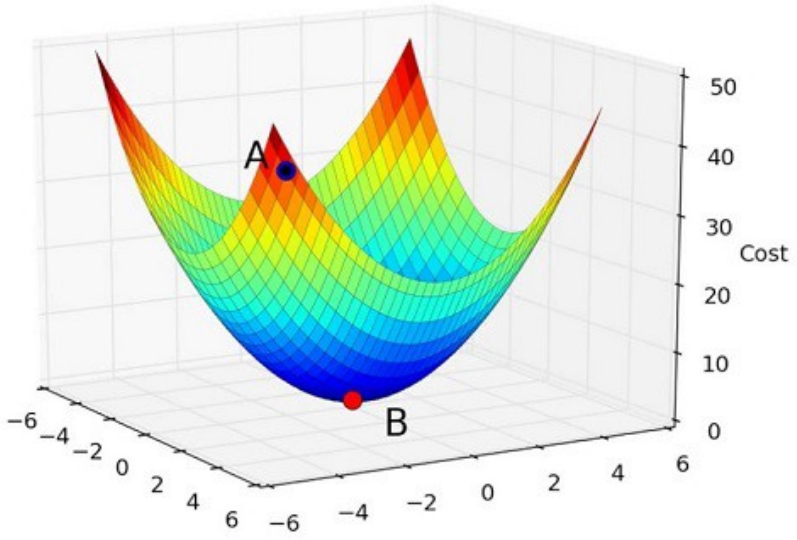


Figura 2.5: Discesa del Gradiente dal punto A al punto B di minimo.
Fonte: https://www.techjuice.pk/deep-dive-gradient-descent-in-machine-learning

Nel caso di un modello di machine learning si vuole trovare il punto in cui l'errore commesso dal modello sia minimo. Graficando la funzione di errore si ottiene un grafico ad n+1 dimensioni, dove n è il numero di parametri della funzione d'errore, e vi si aggiunge l'errore calcolato dalla predizione. Per trovare quindi il punto in cui tale errore sia minimo si utilizza l'appena descritto metodo di discesa del gradiente, che consiste nel:

- Fissare arbitrariamente dei valori dei parametri della funzione d'errore;
- Calcolare la funzione d'errore;
- Calcolare il gradiente della funzione;
- Sottrarre ai parametri della funzione d'errore un vettore proporzionale al gradiente;
- Ripete il procedimento fino a quando non si converge a un minimo locale.

È possibile modificare il procedimento affinché l'algoritmo si fermi dopo un numero prefissato di iterazioni oppure quando gli spostamenti da un'iterazione all'altra sono nulli o quasi.

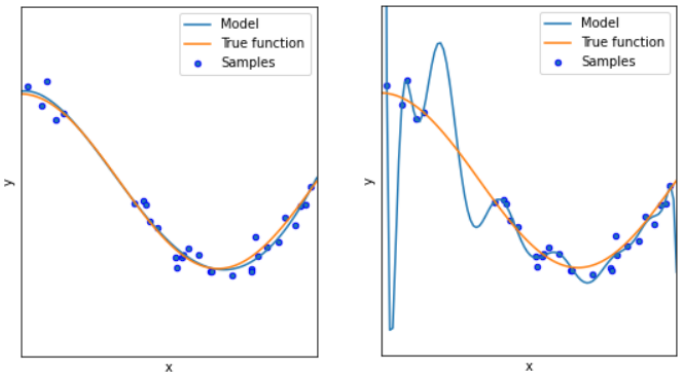
2.1.6 Validazione del modello

Addestrare un modello a fare previsioni non è sufficiente. Occorre sapere anche se ciò che è stato sviluppato funziona correttamente, e per farlo è necessario verificare l'accuratezza del modello su dei dati diversi rispetto a quelli su cui è stato addestrato.

Durante la fase di preparazione dei dati, essi vengono divisi in più porzioni: una porzione relativa ai dati di training, detta **training set** e che serve appunto per addestrare il modello, e una porzione detta **validation set** relativa ai dati che verranno utilizzati per verificare l'accuratezza del modello. Ci sarebbe anche un'ulteriore porzione, detta **test set**, che è utile per misurare l'accuratezza su nuovi dati ignoti a regime.

Per effettuare la suddivisone dei dati esistono diversi approcci. Un metodo abbastanza noto per tale compito è il metodo **hold-out**, il quale divide i dati a disposizione secondo una proporzione predefinita, ad esempio 70% di training e 30% di validation, 50%-50% (anche se è preferibile che l'insieme di training sia più corposo rispetto agli altri), ecc.

Dopo aver addestrato il modello se ne verifica dunque l'errore sul validation set. Se l'errore commesso sul validation set è simile a quello sul training set, allora significa che il modello ha generalizzato bene i dati. Se invece l'errore sul validation set è significativamente superiore rispetto a quello di training, ciò indica che il modello è affetto da **overfitting**, ovvero il modello perde di generalità in quanto si lega troppo ai dati su cui è stato addestrato, oppure significa che il training set non rappresenta adeguatamente il dataset.



(a) Modello addestrato correttamente.

(b) Modello affetto da overfitting.

Figura 2.6: Uno stesso modello addestrato in due modi differenti

L'overfitting può essere dovuto ad un'eccessiva quantità di dati di training, i quali fanno perdere di generalità al modello che in questo modo non rappresenta

adeguatamente dati ignoti. Per risolvere il problema è possibile utilizzare meno dati oppure considerare altre feature del dataset o escluderne delle altre.

2.2 Deep Learning

Come già detto nell'introduzione, il Deep Learning [4] è un sottoinsieme del Machine Learning e si basa sull'apprendimento non supervisionato. Il ricorso a questa tecnologia è specialmente indicato quando si ha a che fare con problemi particolarmente complessi. A differenza del Machine Learning che si serve di dati strutturati e in cui gli algoritmi vengono ottimizzati per mezzo dell'intervento umano che indica la qualità del lavoro svolto, un sistema di Deep Learning cattura le differenze significative tra i dati, ed è il sistema stesso a verificare la correttezza di quanto prodotto.

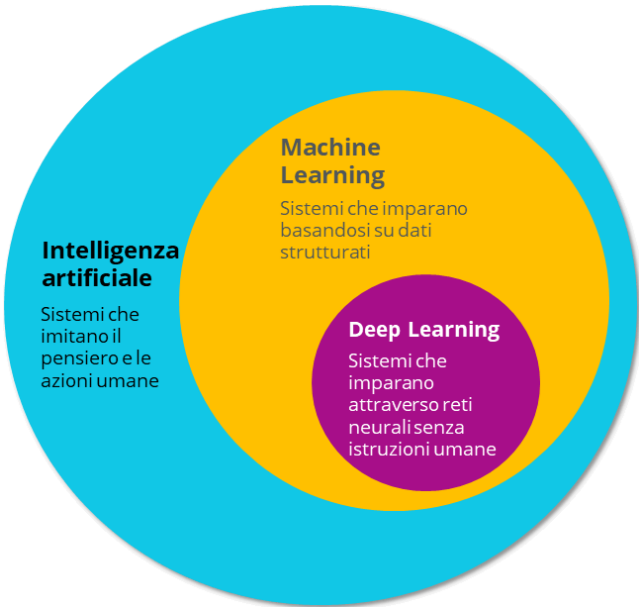


Figura 2.7: Gerarchia tecnologie di Intelligenza Artificiale.
Fonte:
https://www.ionos.it/digitalguide/online-marketing/marketing-sui-motori-di-ricerca/deep-learning-vs-machine-learning

Il Deep Learning ha guadagnato un'ampia popolarità soprattutto nell'ultimo decennio, poiché a differenza degli anni precedenti in cui si avevano a disposizione una quantità relativamente limitata di dati e le architetture a disposizione erano computazionalmente limitate, nei tempi più recenti sono aumentati considerevolmente i dati a disposizione e sono stati sviluppati sistemi di elaborazione con una maggiore potenza di calcolo e basati su **GPU**.

La GPU infatti, è diventata una parte integrante ora per eseguire qualsiasi algoritmo di Deep Learning. Inoltre domina in termini di precisione quando addestrato con enormi quantità di dati.

L'era dei cosiddetti "**Big Data**" della tecnologia fornirà enormi opportunità per nuove innovazioni nel Deep Learning.

Un altro motivo per cui i sistemi di apprendimento profondo stanno riscontrando molto successo è dovuto all'ottimizzazione dei metodi di addestramento delle **reti neurali**, le quali permettono oggi di risolvere problemi per i quali non erano mai stati ottenuti risultati soddisfacenti.

La seguente figura presenta un grafico che mostra l'attenzione che sta ricevendo il Deep Learning negl'ultimi anni, sulla base delle ricerche effettuate su Google:

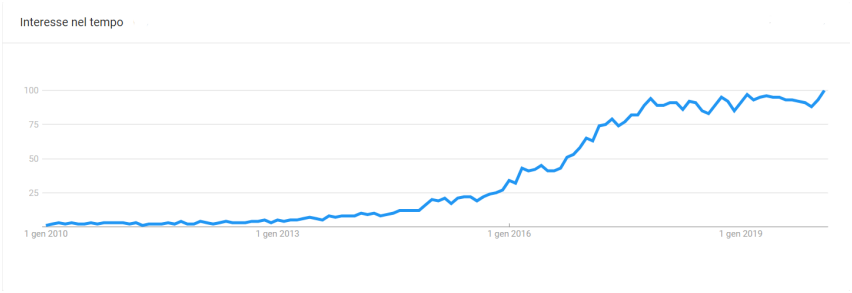


Figura 2.8: Trend del Deep Learning su Google.
Fonte: https://trends.google.com/trends/explore?date=2010-01-01%202020-02-29&q=deep%20learning

Oggi i sistemi basati su Deep Learning consentono ad esempio di:

- Distinguere e identificare gli oggetti rappresentati nelle immagini e nei video;
- Trascrivere il parlato in testo;
- individuare e interpretare gli interessi degli utenti online, mostrando i risultati più pertinenti per la loro ricerca.

In molti casi i sistemi di Deep Learning arrivano anche a superare le prestazioni degli esseri umani.

Il Deep Learning quindi in sostanza indica quella branca dell'Intelligenza Artificiale che fa riferimento agli algoritmi ispirati alla struttura e alla funzione del cervello chiamate reti neurali artificiali.

2.3 Reti Neurali

Nel corso della storia, uno svariato numero di invenzioni ha basato le sue fondamenta sulla natura: gli uccelli hanno ispirato la creazione degli aerei, lo studio delle dita dei gechi ha portato alla creazione di prodotti utili per l'arrampicata, il velcro usato quotidianamente deriva dallo studio del dicardio alpino, e tante altre.

Allo stesso modo la costruzione di macchine intelligenti ha tratto ispirazione dall'architettura del cervello umano. Da questa idea hanno preso poi vita le **Reti Neurali Artificiali** [5].

Le Reti Neurali Artificiali rappresentano il cuore del Deep Learning. Sono versatili, potenti e scalabili, rendendole ideali per affrontare compiti di Machine Learning molto complessi. Il sistema delle Reti Neurali può essere utilizzato per attività come la classificazione e la previsione.

Le Reti Neurali **si ispirano ai principi di funzionamento del cervello umano**. Sono composte da neuroni artificiali che si ispirano ai neuroni biologici interconnessi, i quali formano le reti neurali cerebrali.

Il *primo neurone artificiale* della storia venne proposto dai già citati McCulloch e Pitts in un famoso progetto del 1943 pubblicato nell'articolo "*A Logical Calculus of the Ideas Immanent in Nervous Activity*" [1], col quale cercarono di dimostrare che una macchina di Turing poteva essere realizzata con una rete finita di neuroni per dimostrare che il neurone era l'unità logica di base del cervello.

I neuroni di una rete neurale sono connessi tra loro all'interno di una rete molto ampia. Questi neuroni artificiali sono disposti in strati, detti *layers*. Mentre le prime Reti Neurali utilizzavano un ristretto numero di strati, a causa anche della complessità dell'addestramento dovuta alla potenza di calcolo delle architetture a disposizione, la maggior parte delle Reti Neurali moderne utilizza un ampio numero di strati.

Gli strati di neuroni sono raggruppabili in 3 categorie:

- **Strato di input** (*input layer*): è lo strato che contiene dei nodi fittizi, i quali ricevono i dati di input e hanno il compito di adattarli alle richieste dei neuroni della rete;
- **Strati nascosti** (*hidden layers*): in una rete neurale possono essere presenti più strati nascosti, i quali contengono i neuroni che effettuano il processo di elaborazione vero e proprio;
- **Strato di output** (*output layer*): in questo strato il numero di neuroni varia a seconda del tipo di compito che si sta svolgendo e di cosa si vuole

ottenere in output. Se ad esempio si desidera effettuare una classificazione binaria di un oggetto, si può scegliere di creare uno strato di output con 2 neuroni, uno per classe, dove ciascuno mostra in output la probabilità di appartenenza dell'oggetto alla relativa classe che rappresenta il neurone. Oppure si può inserire un solo neurone che mostra in output un valore compreso tra 0 e 1, e se tale valore è maggiore di 0.50 allora l'oggetto apparterrà ad una classe, altrimenti apparterrà all'altra.

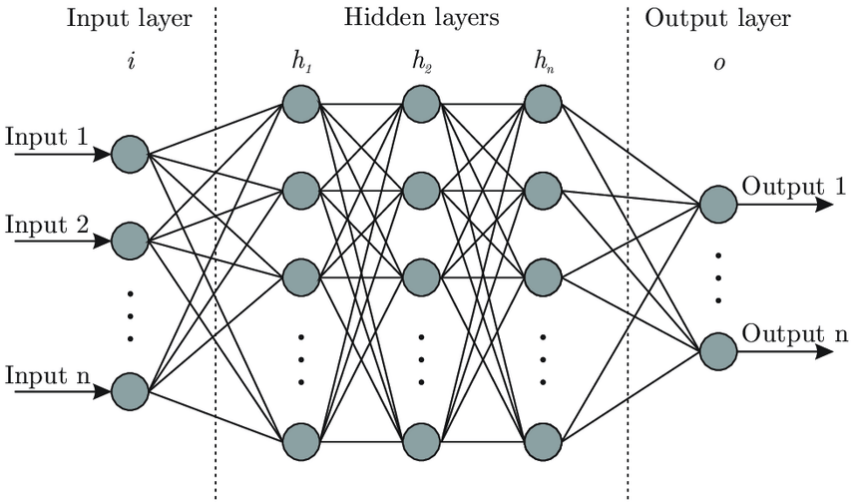


Figura 2.9: Architettura di una Rete Neurale Artificiale.
Fonte: https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051

Ogni neurone attiva il suo funzionamento ogni volta che si attivano un numero sufficiente di neuroni di input. Ad ogni instante, ciascun neurone emette un valore, il quale viene calcolato sulla base dei valori ricevuti dai nodi di input ad esso collegati. Per ogni neurone, un input è relativo ad un singolo altro neurone, mentre il suo output può essere inviato come input a tutti i neuroni a cui è collegato. Quindi gli input di un neurone in un certo strato, sono gli output di tutti i neuroni ad esso connessi dello strato precedente. Neuroni appartenenti ad uno stesso strato non possono essere connessi tra loro.

2.3.1 Funzione di Attivazione

Per produrre i risultati in output, ciascun neurone applica una *funzione di attivazione* ai dati di input ricevuti. I neuroni in ogni strato possono usare una diversa funzione di attivazione. Esistono diversi tipi di funzione di attivazione, tra le più utilizzate ci sono:

- **Sigmoide**: i valori che può restituire il neurone spaziano tra 0 e 1 in un intervallo continuo. La risposta in uscita della sigmoide non è più lineare. Quindi se un neurone successivo usa la risposta data dal neurone precedente che ha uscita sigmoidale, questo neurone si vede arrivare dei dati che non sono più lineari rispetto ai dati originali, ma sono stati trasformati in modo non lineare;
- **Rectified Linear Unit** ($ReLU$): è la funzione di attivazione maggiormente utilizzata. I possibili valori restituiti dal neurone variano da 0 a infinito. Restituisce 0 se la somma pesata dei dati in input è minore o uguale a 0, altrimenti restituisce proprio la somma pesata dell'input;
- **Softmax**: sigmoide normalizzata, viene spesso utilizzata nello strato di output. Restituisce la probabilità di appartenenza di un oggetto a ciascuna classe, per cui la somma delle probabilità dovrà quindi essere 1.

Grazie all'introduzione della non linearità nei dati da parte delle funzioni di attivazione, la Rete Neurale è in grado di capire autonomamente, senza che nessuno imponga un modello, come arrivare a separare i dati. Ovvero trova la migliore trasformazione da applicare ai dati. La rete deforma lo spazio, e vi sposta i dati in modo da trovarne una separazione lineare. La separazione lineare trovata nello spazio trasformato, corrisponde a una separazione non lineare nello spazio originale.

2.3.2 Costruzione di una Rete Neurale

Una Rete Neurale è inizialmente costruita impostando alcuni aspetti strutturali, i cosiddetti **iperparametri**. La flessibilità delle Reti Neurali è anche uno dei loro principali svantaggi, poiché ci sono molti iperparametri da modificare, come ad esempio il numero degli strati nascosti e dei neuroni in ogni strato, la funzione di attivazione da utilizzare in ogni strato, il numero di epoche, il learning rate, la batch size ed altri. Questi iperparametri devono essere scelti sulla base dello specifico compito che si dovrà andare ad eseguire.

Una volta che la rete è stata creata e che tutti gli iperparametri necessari sono stati scelti, i *parametri* della rete, ovvero i *pesi* associati agli input e i relativi *bias*, avranno dei valori iniziali di default. Questi parametri devono essere regolati addestrando la rete, al fine di ottenere da essa il comportamento previsto. **Le reti posso avere anche più di un milione di parametri da addestrare**.

Uno dei metodi maggiormente diffusi ed efficaci per l'addestramento delle reti neurali è il cosiddetto algoritmo di retropropagazione dell'errore, ovvero

l'**error backpropagation** [6]. Tale algoritmo modifica sistematicamente i pesi delle connessioni tra i neuroni, in modo che il risultato fornito dalla rete sia sempre più vicino a quello desiderato.

L'addestramento di una rete neurale avviene attraverso molteplici iterazioni sul training set. Il numero di iterazioni corrisponde al numero di *epoche* impostato precedentemente. Ad ogni iterazione, viene utilizzata per l'addestramento una *batch di dati*. Una batch è un gruppo di dati scelti casualmente dal training set. Occorre però definire quante istanze selezionare, e questo è determinato dalla *batch size*, altro iperparametro impostato in fase di costruzione della rete.

Ogni istanza di training include un vettore di input passato alla rete e l'output che ci si aspetta di ottenere da essa. Per ogni input la rete restituisce un output predetto. L'obiettivo dell'addestramento è quello di minimizzare la differenza tra output atteso e output predetto.

Per farlo occorre scegliere la combinazione di iperparametri migliore [7] in fase di costruzione della rete, come il numero di strati nascosti e di neuroni in ciascuno di essi, il learning rate e la batch size dei dati di addestramento.

Strati Nascosti

La maggior parte dei dati del mondo reale è strutturata in modo gerarchico e le Reti Neurali sfruttano automaticamente questo fatto: gli strati nascosti inferiori modellano strutture di basso livello (ad esempio, segmenti di linea di varie forme e orientamenti), gli strati nascosti intermedi combinano queste strutture di basso livello per modellare strutture di livello intermedio (ad esempio, quadrati, cerchi) e gli strati nascosti più alti e lo strato di output combinano queste strutture intermedie per modellare strutture di alto livello.

Per molti problemi è possibile iniziare con solamente uno o due livelli nascosti e si riusciranno ad ottenere risultati soddisfacenti. Quando si ha a che fare con problemi più complessi invece, è possibile aumentare gradualmente il numero di livelli nascosti, fino a quando il modello di rete non inizia a perdere di generalità e ad andare in overfitting con il training set.

Numero di Neuroni negli Strati Nascosti

Mentre il numero di neuroni negli strati di input e di output è determinato dal tipo di input e di output richiesto dall'attività che si sta svolgendo, per quanto riguarda gli strati nascosti, una volta era solito scegliere un numero di neuroni per strato in modo da formare una sorta di piramide, quindi con

sempre meno neuroni ad ogni strato. Questo perché molte caratteristiche di basso livello possono fondersi in molte meno caratteristiche di alto livello.

Tuttavia, questa pratica è stata in gran parte abbandonata ora, poiché sembra che utilizzare lo stesso numero di neuroni in tutti gli strati nascosti funzioni altrettanto bene nella maggior parte dei casi, o talvolta anche meglio. Inoltre a differenza dell'usanza precedente, vi è solo un iperparametro da regolare invece di uno per strato.

Ci sono però casi in cui può risultare conveniente scegliere per il primo strato nascosto un numero di neuroni superiore rispetto a tutti gli altri, ciò può dipendere ad esempio dal dataset.

Scegliere un numero di neuroni fisso all'interno degli strati nascosti però non è l'unica soluzione. Infatti, esattamente come per il numero di strati, è possibile provare ad aumentare gradualmente il numero di neuroni fino a quando la rete non va in overfitting. In generale, si aumenteranno il numero di strati rispetto al numero di neuroni per strato. Ciò nonostante, riuscire a trovare la quantità perfetta di neuroni da inserire in ogni strato non è così facile.

Un approccio più semplice consiste nello scegliere un modello con più strati e neuroni di quelli effettivamente necessari, quindi utilizzare l'early stopping per evitare che vada in overfitting.

Learning Rate

Il learning rate è sicuramente l'iperparametro più importante da impostare in una rete neurale. In generale, il learning rate ottimale è circa la metà del learning rate massimo (cioè, il learning rate al di sopra del quale l'algoritmo di addestramento diverge, ovvero si allontana dalla soluzione ottimale).

Un possibile approccio per regolare la velocità di addestramento può essere quello di impostare il learning rate con un valore grande che fa divergere l'algoritmo di addestramento, quindi dividere questo valore per 3 e riprovare e ripetere fino a quando l'algoritmo di addestramento smette di divergere. A quel punto, generalmente non si sarà troppo lontani dal learning rate ottimale.

Batch Size

La batch size è la quantità di istanze di training che l'algoritmo sceglierà casualmente dal training set ad ogni iterazione su di esso. La dimensione della batch può anche avere un impatto significativo sulle prestazioni del modello e

sul tempo di addestramento. In generale, la dimensione ottimale sarà inferiore a 32.

Batch di dimensioni più piccole garantiscono che ogni iterazione dell'addestramento sia molto veloce, mentre batch di dimensioni maggiori forniscono una stima più precisa dei gradienti. Tuttavia ciò non ha molta rilevanza poiché gli aspetti dell'ottimizzazione sono piuttosto complessi e inoltre la direzione dei gradienti non punta precisamente verso la direzione dell'ottimale.

Scegliere una dimensione della batch maggiore di 10 consente di trarre vantaggio dalle ottimizzazioni hardware e software, in particolare per le moltiplicazioni di matrici, quindi velocizzerà l'addestramento.

2.3.3 Data Augmentation

Quando si lavora con attività specializzate nella classificazione di immagini o video spesso i dati a disposizione sono in quantità limitata. Per addestrare un modello di deep learning efficiente occorre avere una discreta quantità di dati a disposizione. In alcuni settori tuttavia, risulta assai difficoltoso avere accesso ai dati, come ad esempio nell'industria medica dove l'accesso ai dati è fortemente protetto per questioni di privacy.

Per sopperire a questa mancanza di dati si fa ricorso alla *Data Augmentation* [8]. Essa consiste in un insieme di tecniche che consentono di **incrementare la quantità e la diversità di dati a disposizione**, aggiungendo copie leggermente modificate dei dati già esistenti. Quindi non si collezionano nuovi dati, piuttosto si trasformano i dati già presenti. Ad esempio ruotando l'immagine, cambiando il livello di zoom, ritagliandola o ancora cambiando il livello di luminosità dell'immagine e altro ancora.

Aumenta artificialmente le dimensioni del training set generando una serie di varianti realistiche di ogni istanza di training. Questo permette di ridurre l'overfitting, rendendolo una tecnica di regolarizzazione. Le istanze generate dovrebbero essere il più possibile realistiche: idealmente, data un'immagine dal training set aumentato, un essere umano non dovrebbe essere in grado di dire se è stata aumentata o meno.

Ruotare e ridimensionare l'immagine costringe il modello ad essere più tollerante alle variazioni di posizione, di orientamento e di dimensione degli oggetti nelle immagini. Se si desidera che il modello sia più tollerante a diversi livelli di illuminazione, è possibile generare in modo simile molte immagini con vari contrasti.

2.4 Reti Neurali Convoluzionali

Lo studio della corteggia visiva del cervello ha portato alla nascita di un nuovo tipo di rete neurale, le Reti Neurali Convoluzionali. Una rete di questo tipo è composta da diversi *stadi* e similarmente a quanto succede nella corteccia visiva, ognuno di questi stadi è specializzato nello svolgere attività diverse. Per permettere agli esseri umani di riconoscere gli oggetti, il cervello umano opera delle semplificazioni, e allo stesso modo opererà una Rete Neurale Convoluzionale. Uno degli stadi intermedi all'interno del cervello, è specializzato all'estrazione di forme o caratteristiche dell'immagine che si sta guardando. La stessa cosa si ritroverà nelle Reti Neurali Convoluzionali.
Esse vengono adoperate per il riconoscimento delle immagini fin dagli anni '80. Negli ultimi anni, grazie all'aumento della potenza di calcolo delle architetture e all'incremento della quantità di dati disponibili, le Reti Neurali Convoluzionali sono riuscite a ottenere prestazioni sovrumane [9] su alcuni compiti visivi complessi, come la classificazione di oggetti [10], volti [11] e scene [12].

Queste reti vengono impiegate per gli obiettivi più disparati come la ricerca di immagini, auto a guida autonoma, sistemi di classificazione video automatica e altro ancora. Inoltre, le Reti Neurali Convoluzionali non si limitano alla percezione visiva, ma vengono utilizzate anche in molti altri compiti, come il riconoscimento vocale o il Natural Language Processing.

L'uso principale di una Rete Neurale Convoluzionale resta però quello di identificare cosa un'immagine rappresenta, e con quale probabilità.

Gli studi sulla corteccia visiva hanno ispirato il neocognitron, introdotto nel 1980, [13] che si è gradualmente evoluto in ciò che oggi vengono chiamate reti neurali convoluzionali. Un traguardo importante è stato un documento del 1998 [14] di Yann LeCun ed altri, che ha introdotto la famosa architettura LeNet-5, ampiamente utilizzata per riconoscere i numeri di assegni scritti a mano. Questa architettura oltre ad avere alcuni elementi costitutivi già noti, come strati completamente connessi e funzioni di attivazione del sigmoide, introduce anche due nuovi elementi costitutivi: strati convoluzionali e strati di pooling.

Le Reti Neurali Convoluzionali permettono di sopperire ai problemi delle normali Reti Neurali Profonde per le attività di riconoscimento delle immagini. Quest'ultime infatti, nonostante funzionino bene con immagini di piccole dimensioni, non riescono a garantire le stesse prestazioni con immagini più grandi a causa dell'enorme numero di parametri che richiedono. Le Reti Neurali Convoluzionali risolvono questo problema utilizzando **strati parzialmente collegati** e **pesi condivisi**.

2.4.1 Struttura

Una Rete Neurale Convoluzionale generalmente funziona come tutte le altre. È infatti costituita da uno strato di input che prende i dati da passare alla rete, uno o più strati nascosti che effettuano le elaborazioni tramite funzioni di attivazione, e uno strato di output che effettua la classificazione vera e propria. Ma a differenza delle altre reti, una rete di questo tipo è caratterizzata dalla presenza degli strati di pooling ma soprattutto degli strati di convoluzione.

L'architettura standard di una normale Rete Neurale Convoluzionale infatti è formata da uno strato di input, una serie di strati convoluzionali intervallati da uno strato di pooling, e uno strato di output completamente connesso.

- Lo **strato di input** è costituito da una sequenza di neuroni, i quali ricevono le informazioni, ovvero l'insieme di numeri, che rappresentano l'immagine da trattare. A questo strato infatti verrà passato il vettore di dati che rappresentano i pixel dell'immagine in ingresso;

- Lo **strato convoluzionale** è il principale della rete. Il suo compito è quello di individuare ed estrarre, con elevata precisione, schemi all'interno dell'immagine, come ad esempio linee verticali, orizzontali o curve, angoli, figure quadrate o rettangolari, circonferenze. In una Rete Neurale Convoluzionale sono presenti diversi strati di questo tipo. Tutti con lo scopo di estratte caratteristiche dall'immagine iniziale. Man mano che si avanza nella rete, gli strati convoluzionali estrarranno caratteristiche sempre più di alto livello. Maggiore è il numero degli strati convoluzionali, maggiore è la complessità della caratteristica che riescono ad individuare;

- Lo **strato ReLU** ha il semplice compito di annullare eventuali valori negativi ottenuti negli strati precedenti, e solitamente è posto dopo gli strati convoluzionali;

- Lo **strato di pooling** permette di identificare se la caratteristica di studio è presente negli strati precedenti. Semplifica e rende più grezza l'immagine, mantenendo la caratteristica utilizzata dal livello convoluzionale. In questo modo riduce il carico computazionale, l'uso della memoria, e il numero dei parametri;

- Lo **strato di output**, anche detto completamente connesso, esegue di fatto la classificazione delle immagini. Connette tutti i neuroni dello strato precedente per stabilire le varie classi identificative visualizzate negli strati precedenti secondo una determinata probabilità.

Livello Convoluzionale

Come detto, è lo strato più importante di una Rete Neurale Convoluzionale. I neuroni nel primo strato convoluzionale non sono connessi ad ogni singolo pixel dell'immagine di input, ma solamente ai pixel che rientrano nel loro campo ricettivo. A sua volta, ogni neurone nel secondo strato convoluzionale è connesso solamente ai neuroni collocati all'interno di un piccolo rettangolo del primo strato. Questa architettura permette alla rete di concentrarsi su caratteristiche di basso livello nel primo strato nascosto, per poi assemblarle in caratteristiche di più alto livello nello strato successivo.

Per estratte le caratteristiche dalle immagini, lo strato convoluzionale si serve di **filtri**.

Un filtro non è altro che una matrice, composta da poche righe e colonne, che rappresenta una caratteristica che il livello convoluzionale vuole identificare, come ad esempio linee, curve, cerchi, ecc.

Nei primi strati il filtro rappresenta una caratteristica di basso livello in quanto indentifica oggetti più semplici, come appunto linee o curve. Mentre negli strati più avanzati il filtro rappresenta una caratteristica di alto livello poiché identifica oggetti più complessi, come una mano, uno specchietto, una coda, ecc.

In ogni strato convoluzionale è possibile specificare il numero di filtri da utilizzare e la dimensione degli stessi. La dimensione scelta per il filtro determina anche la dimensione del campo ricettivo dei neuroni di quello strato.

Per il primo strato, il filtro assume valori casuali. Esso viene fatto scorrere sulle diverse posizioni dell'immagine di input, e per ogni posizione viene prodotto un valore di output che corrisponde al prodotto scalare tra la matrice del filtro e i valori dei pixel della porzione di input coperta. Tale operazione va ripetuta per tutti i blocchi che l'immagine di input può contenere. Per cui il campo ricettivo viene fatto spostare di un determinato **passo**, detto *stride*, verso destra. Dopo aver fatto scorrere il filtro su tutte le posizioni dell'immagine, si ottiene una nuova matrice di pixel che può essere più piccola rispetto alla precedente, a seconda delle dimensioni del filtro e del passo scelto. Questa nuova matrice ottenuta è chiamata **mappa di attivazione**, o *feature map*.

La procedura appena descritta va ripetuta, nello stesso strato convoluzionale, per un numero di volte pari al numero di filtri che si è deciso di applicare, ottenendo così lo stesso numero di mappe di attivazione.

Oltre al numero di filtri, alla loro dimensione, e al passo, un ruolo importante lo gioca il cosiddetto **zero padding**. Esso consiste nell'aggiungere degli zeri attorno al volume di input originale, allo scopo di evitare di perdere delle

informazioni passando da uno strato a un altro. Infatti, come visto prima, la matrice ottenuta in seguito all'applicazione di un filtro, può essere più piccola dell'input originale, per cui usando filtri e passo relativamente grandi si otterrebbe un output decisamente più piccolo.

Man mano che si continuano ad applicare gli strati convoluzionali, la dimensione del volume di input diminuirà più velocemente di quanto si vorrebbe. Nei primi strati della rete sarebbe meglio conservare il maggior numero possibile di informazioni relative al volume di input, per poter estrarre le caratteristiche di basso livello che altrimenti andrebbero perse e che sarebbe poi impossibile recuperare negli strati successivi. Per fare ciò si applica lo zero padding, la cui dimensione dipende da quella dei filtri e dal passo.

Il numero di filtri, la dimensione dei filtri, il passo e lo zero padding, rappresentano tutti degli iperparametri caratteristici degli strati convoluzionali. Per scegliere i valori migliori di tali iperparametri non esiste uno standard unico, in quanto la rete dipende soprattutto dal tipo di dati che si hanno a disposizione.

Livello ReLU

L'output di ogni strato convoluzionale corrisponde all'input di uno strato ReLU, in quanto solitamente v'è n'è sempre uno collocato subito dopo lo strato convoluzionale.

Questo livello **rappresenta un livello non lineare**, in quanto il suo obiettivo è quello di introdurre la non linearità a un sistema che sta calcolando operazioni lineari durante gli strati convoluzionali, attraverso il prodotto scalare tra filtri e campi ricettivi.

Grazie agli strati ReLu, le Reti Neurali Convoluzionali riescono ad ottenere prestazioni migliori poiché la rete è in grado di allenarsi molto più velocemente senza impattare in modo significativo sull'accuratezza dei risultati.

Livello Pooling

L'obiettivo di uno strato di pooling è quello di stabilire se la caratteristica di studio è presente negli strati precedenti. Semplifica e rende più grezza l'immagine, mantenendo la caratteristica utilizzata dal livello convoluzionale, allo scopo di ridurre il carico computazionale, l'uso della memoria, e il numero dei parametri.

Ogni neurone in questo strato è connesso all'output di un limitato numero di neuroni dello strato precedente, i quali si trovano all'interno di un piccolo campo ricettivo rettangolare. Occorre definire la dimensione di tale campo ricettivo, il passo e il tipo di padding da applicare.

A differenza di altri strati, lo strato di pooling non ha pesi. L'unico compito di cui si occupa è quello di aggregare l'input usando una *funzione di aggregazione*, la quale determina anche il tipo di strato di pooling. Le funzioni di aggregazione più utilizzate per questo scopo sono il *massimo* e la *media*, con cui si ottengono rispettivamente un **MaxPoolingLayer**, che è anche il più popolare, e un **AveragePoolingLayer**.
Lo strato di pooling richiede un filtro e un passo della stessa lunghezza, e li applica al volume di input dello strato precedente. Se lo strato usa il massimo come funzione di aggregazione, viene passato allo strato successivo solamente il pixel col valore più grande all'interno del campo ricettivo, mentre se utilizza la media effettua la somma dei valori dei pixel e la divide per il numero di pixel all'interno del campo ricettivo, tale valore viene passato allo strato successivo.

2.4.2 MobileNet

Le Reti Neurali Convoluzionali trovano applicazione in un ampio numero di settori. Uno di questo tipo di reti, è la MobileNet [15]. Si tratta di una Rete Neurale Convoluzionale semplice ma efficiente che, come suggerisce il nome, è stata progettata per essere utilizzata nelle applicazioni mobile di visione artificiale.

Questa architettura è largamente impiegata in un vasto numero di applicazioni del mondo reale, come ad esempio il rilevamento di oggetti, la localizzazione, ecc. È stato reso open source da parte di Google, dando la possibilità di avere un ottimo punto di partenza per costruire i propri classificatori piccoli e veloci.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figura 2.10: Architettura di MobileNet.
Fonte: https://www.researchgate.net/figure/layers-of-MobileNet-architecture-4_fig2_331675538

MobileNet utilizza **convoluzioni separabili in profondità**, note come *depthwise separate convolutions* [16]. Ciò riduce notevolmente il numero di parametri rispetto alle reti con convoluzioni standard ma con la stessa profondità nelle reti. In questo modo si ottengono reti neurali profonde leggere.

Questa convoluzione ha avuto origine dall'idea che la profondità e la dimensione spaziale di un filtro possano essere separate, da qui il nome *separabile*.

La convoluzione separabile in profondità è composta da 2 strati, lo **strato di convoluzione in profondità**, detto *depthwise convolution layer*, e lo **strato di convoluzione in punti**, detto invece *pointwise convolution layer*. Il primo strato viene usato per filtrare i canali dell'input, mentre il secondo viene impiegato per combinarli in modo da creare una nuova funzionalità.

La depthwise convolution è usata per applicare un singolo filtro ad ogni canale di input. Ciò differisce da una normale convoluzione poiché quest'ultima applica i filtri a tutti i canali di input. Per una convoluzione normale, il costo computazionale dipende dal numero di canali di input e output e dalle dimensioni spaziali della feature map di input e del kernel di convoluzione.

Poiché la depthwise convolution si preoccupa solamente di applicare un filtro al canale di input, senza combinarli per creare nuove features, viene creato uno strato aggiuntivo, chiamato pointwise convolutions layer, ovvero strato di convoluzione in punti. Esso calcola una combinazione lineare dell'output della depthwise convolution.

2.4.3 MobileNetV2

Man mano che emergono nuove applicazioni che consentono agli utenti di interagire con il mondo reale in tempo reale, aumenta anche la necessità di reti neurali sempre più efficienti. Da questa esigenza è nata l'idea di progettare e sviluppare una **nuova versione di MobileNet** allo scopo di alimentare la prossima generazione di applicazioni mobile di visione artificiale, e che prende il nome di *MobileNetV2* [17].

Questa nuova versione rappresenta un miglioramento significativo rispetto a MobileNetV1 e si serve di tutte le conoscenze apprese fino ad ora nella sfera del riconoscimento visivo in ambito mobile, inclusa la classificazione, il rilevamento degli oggetti e la segmentazione semantica.

MobileNetV2 si basa sulle idee di MobileNetV1, utilizzando la convoluzione separabile in profondità come elementi fondamentali per la sua realizzazione. Tuttavia, MobileNetV2 introduce due nuove funzionalità nell'architettura: colli

di bottiglia lineari, detti *bottlenecks*, tra gli strati e collegamenti rapidi, detti *shortcuts*, tra questi bottlenecks.

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

Figura 2.11: Architettura di MobileNetV2.
Fonte: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

L'obiettivo dei bottlenecks è quello di codificare gli input e gli output intermedi del modello mentre lo strato interno incapsula la capacità del modello di trasformare da concetti di livello inferiore come i pixel a descrittori di livello superiore come le categorie di immagini. Infine gli shortcuts consentono un addestramento del modello più veloce e una migliore precisione.

Nel complesso, i modelli MobileNetV2 sono più veloci per ottenere la stessa accuratezza rispetto alla versione precedente, ovvero impiegano un tempo di elaborazione dell'input, necessario per produrre l'output, inferiore. In particolare, i nuovi modelli utilizzano il doppio delle operazioni, richiedono il 30% di parametri in meno e sono circa il 30-40% più veloci su un telefono Google Pixel rispetto ai modelli MobileNetV1, il tutto ottenendo una maggiore precisione.

MobileNetV2 inoltre è un estrattore di features molto efficace per il rilevamento e la segmentazione di oggetti.

2.4.4 ResNet50

La prima **Rete Residuale**, o **Residual Network** (da qui il nome ResNet), fu sviluppata nel 2015 da Kaiming He [18] e consisteva in una Rete Neurale Convoluzionale estremamente profonda composta da 152 strati. Alla base dell'addestramento di una Rete Residuale vi sono le cosiddette *skip connections*,

grazie alle quali l'input di uno strato viene anche aggiunto all'output di uno strato situato un po' più in alto nello stack.

A differenza di una normale Rete Neurale, in cui in fase di inizializzazione i suoi pesi sono vicini allo zero e che quindi restituisce solo valori vicini allo zero, una Rete Residuale, con l'aggiunta quindi di una skip connection, restituisce soltanto una copia dei suoi input. Se una Rete Residuale è composta da un elevato numero di skip connection, essa può iniziare a fare progressi anche se diversi strati non hanno ancora iniziato l'apprendimento. Una Rete Residuale può essere vista come una pila di *unità residuali* dove ogni unità residuale è una piccola Rete Neurale con una skip connection.

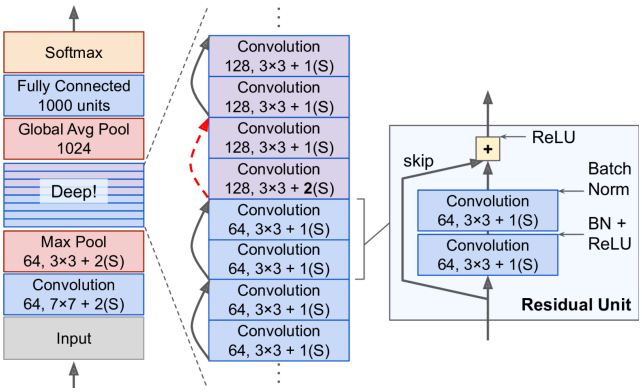


Figura 2.12: Architettura di ResNet.

L'architettura di una Rete Residuale è composta da uno strato di input, una serie di unità residuali, e lo strato di output.

I primi due strati dividono per 4 l'altezza e la larghezza dell'immagine, in modo da ridurre il carico computazione. Il primo strato convoluzionale utilizza dei filtri molto grandi, così da preservare molte delle informazioni.

Gli strati intermedi non sono altro che una pila di semplici unità residuali. Ogni unità residuale è composta da due strati convoluzionali, usando filtri 3x3 e preservando la dimensione spaziale, ricorrendo alla *Batch Normalization* [19] (una tecnica che aggiunge un'operazione nel modello subito prima o dopo la funzione di attivazione di ogni strato nascosto semplicemente normalizzando ogni input) e usando la funzione di attivazione ReLU. Non sono previsti invece, all'interno di questa pila, strati di pooling.

La tecnica consiste nell'aggiungere un'operazione nel modello subito prima o dopo l'attivazione della funzione di ogni strato nascosto, semplicemente centrando lo zero e normalizzando ogni input, quindi ridimensionando e spostando il risultato usando due nuovi vettori di parametri per strato: uno per il ridimensionamento, il altro per lo spostamento.

consente al modello di apprendere la scala e la media ottimali di ciascuno degli input del livello

In seguito, lo strato GlobaleAveragePooling restituisce la media di ciascuna feature map. Ciò elimina qualsiasi informazione spaziale rimanente, il che va bene poiché non c'erano molte informazioni spaziali rimaste in quel punto.

Gli ultimi strati sono uno strato completamente connesso con 1.000 neuroni, poiché ci sono 1.000 classi, e una funzione di attivazione sofrmax per mostrare la probabilità stimata di appartenenza a ciascuna classe.

Il numero di feature map viene raddoppiato ogni poche unità residuali, e allo stesso tempo la loro altezza e larghezza è dimezzata, a causa dell'utilizzo di un passo pari a 2 negli strati convoluzionali. Quando questo accade, gli input non possono essere aggiunti direttamente agli output dell'unità residuale finché non hanno la stessa forma. Per risolvere questo problema, gli input vengono passati attraverso uno strato convoluzionale con filtro 1x1, passo 2 e il giusto numero di feature maps di output.

Esistono diverse versioni di ResNet, che funzionano sostanzialmente nello stesso modo, e che sono chiamate in modo differente a seconda della loro profondità, ovvero del numero di strati convoluzionali di cui è composta.

ResNet50 è una variante del modello ResNet che ha 48 strati convoluzionali insieme ad uno strato MaxPool e uno strato AveragePool. A differenza di altre versioni, in ResNet50 le skip connections saltano tre strati invece di due.

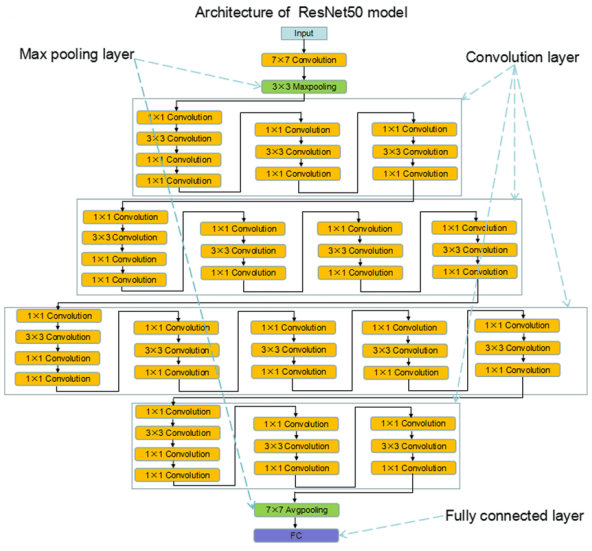


Figura 2.13: Architettura di ResNet50.

Fonte: https://www.researchgate.net/figure/The-architecture-of-ResNet50-and-deep-learning-model-flowchart-a-b-Architecture-of_fig1_334767096

Capitolo 3

Modellazione del progetto

Dopo aver descritto l'insieme delle tecnologie disponibili, si procede affrontando il modo in cui poterle utilizzare nella problematica in questione.

3.1 Analisi dei dataset

L'obiettivo della tesi è quello di individuare la soluzione migliore che permetta di stabilire se le persone inquadrate in un'immagine o in un video indossano o meno la mascherina protettiva nel rispetto delle norme anti-contagio come previsto dall'Organismo Mondiale della Sanità.

Affinché esista una soluzione in grado di raggiungere tale obiettivo, è necessario che impari a riconoscere quando su un volto è presenta una mascherina e quando invece no. Quindi sarà fondamentale avere a disposizione dei campioni d'esempio sui quali addestrare le soluzioni.

Come specificato nella sezione 1.1, sono stati reperiti tre dataset ottimali, utili per raggiungere questo scopo.

Analizzando questi dataset è possibile notare come due di essi siano già perfettamente predisposti per un lavoro di questo tipo. Questo perché le immagini al loro interno sono già organizzate nelle cartelle di training, di validation e di test. Per cui si suppone che le immagini contenute nella cartella di training saranno utilizzate per addestrare un eventuale modello di classificazione, mentre quelle nella cartella di validation serviranno per verificare se il modello addestrato ottiene dei buoni risultati e per migliorarne l'accuratezza durante l'addestramento stesso. Infine la cartella di testing può essere utilizzata per validare il modello, dopo l'addestramento, su dati ignoti a

regime, ovvero che non ha mai visto prima. Ma non sarà questo il caso, poiché per effettuare questa attività verranno utilizzati degli appositi testset.

I motivi di questa scelta sono molteplici:

- Si vuole verificare l'accuratezza delle soluzioni su un quantitativo di immagini superiore rispetto a quelle contenute all'interno della cartella di testing dei dataset che la includono;
- Le immagini di test dei dataset sono più o meno simili a quelle di training e validation, ovvero quelle su cui è stato addestrato il modello, per cui è preferibile verificarne l'accuratezza su immagini potenzialmente molto diverse per vari aspetti. Si vuole quindi che il modello generalizzi il più possibile;
- Uno dei dataset non prevede la suddivisione delle immagini di testing in cartelle con e senza mascherina, ciò è dovuto al fatto che esse rappresentano esclusivamente volti in primo piano in cui nessuno indossa la mascherina, eccetto qualche eccezione. In questo modo non è possibile avere una precisione affidabile del modello, soprattutto per quanto riguarda l'altra classe, ovvero quella coi volti che indossano la mascherina;
- Si vogliono testare i modelli in determinate condizioni, ad esempio per vedere come si comportano quando all'interno di un'immagine sono presenti molte persone rispetto a quando invece vi sono solo una o due persone inquadrate; oppure per vedere se trovino maggiore difficoltà nel classificare volti discretamente lontani dalla telecamera rispetto a volti in primo piano.

Il terzo dataset, come detto, non prevede le cartelle di training, validation e testing, ma divide direttamente tutte le immagini in cartelle dedicate rispettivamente alle immagini di volti con la mascherina e senza mascherina. Per cui sarà necessario provvedere personalmente alla suddivisione di tali informazioni in insieme di training ed insieme di validation.

Nel 1° dataset tutte le immagini hanno all'incirca la stessa dimensione in termini di pixel, tranne qualche eccezione. Tuttavia è facile vedere ad occhio come alcune di queste siano state precedentemente ridimensionate.

Nel 2° dataset invece le immagini hanno dimensioni piuttosto variabili, si va dalle più piccole di dimensione 25x25 alle più grandi di dimensione perlopiù 224x224.

Anche il 3° dataset ha immagini di dimensioni variabili, ma rispetto al 2° sono più distribuite, ovvero ci sono pressappoco lo stesso numero di immagini per le diverse dimensioni, e vi sono alcune immagini di dimensioni nettamente maggiori rispetto agli altri due dataset.

3.2 Analisi dei testset

Come specificato in precedenza, una volta addestrate tutte le soluzioni sui diversi dataset, occorre verificare la bontà, ovvero l'accuratezza, di ciascuna di esse, e per farlo si ricorrerà ad appositi dataset, che in questo caso avranno funzione di testing.

Sono stati reperiti 2 testset ottimali per raggiungere tale obiettivo. In tutti e due i testset i dati sono divisi in due cartelle: una per le immagini, e una per le etichette.

Le immagini, a differenza di quelle contenute nei dataset di addestramento, rappresentano istantanee in tempo reale del mondo reale. Inoltre ognuna di esse può contenere più persone inquadrate dalla telecamera. Occorrerà quindi poi estrarre il volto di ciascuna persona raffigurata nell'immagine e consegnarlo alla rete per verificare se indossa o meno una mascherina.

Per quanto riguarda le etichette, in entrambi i testset esse sono sotto forma di file xml, dove ciascuno di essi è formato da tag. Tali tag saranno essenziali per poter estrarre dai file le informazioni riguardo alla posizione dei volti delle persone in ciascun immagine, e per informare la soluzione se su ogni volto è presente o meno la mascherina.

3.3 Analisi delle Soluzioni

L'obiettivo finale di questa tesi è quello di avere una soluzione che permetta di stabilire se su un qualsiasi volto, rappresentato in un'immagine o in un frame di un video, sia presente o meno una mascherina protettiva. Per raggiungere questo scopo non si andrà a creare una soluzione da zero, piuttosto si andranno a cercare una serie di soluzioni già esistenti, ideate per compiti di questo tipo. Si cercherà più di una soluzione poiché l'intento è quello di trovare la soluzione migliore possibile.

Le soluzioni trovate sono composte da architetture, in parte simili e in parte diverse tra loro.

Nel passo successivo si andranno ad analizzare una per una tutte le soluzioni reperite, identificate dal nome dell'autore.

hritik

Questa soluzione **si basa sull'architettura MobileNetV2** che, come detto, è una rete neurale convoluzionale ideata per essere utilizzata per compiti di visione artificiale nelle applicazioni mobile, ed è una versione migliorata di MobileNetV1. In questo caso si è deciso di utilizzare dei pesi pre-addestrati, appunto per addestrare la rete. Tali pesi sono già stati addestrati su uno specifico dataset chiamato *ImageNet* [20], il quale è dotato di un vasto quantitativo di immagini che appartengono ad un elevato numero di classi, ed è comunemente utilizzato per valutare sistemi di visione artificiale. Questa soluzione carica quindi un modello MobileNetV2 pre-addestrato, ma i livelli di output completamente connessi del modello non vengono caricati, consentendo di poter aggiungere e addestrare un nuovo livello di output.
Per cui l'autore costruirà personalmente il nuovo livello di output del modello, che sarà costituito da:

- Uno strato *AvaragePooling2D* che utilizza la media come funzione di aggregazione per aggregare l'immagine ricevuta in input, semplificando e rendendo più grezza l'immagine;
- Uno strato *Flatten* che appiattisce l'input trasformandolo in un vettore ad una dimensione;
- Uno strato *Dense* che applica la funzione di attivazione $ReLU$ ai dati introducendo la non linearità;
- Uno strato *Dropout* [21] che azzera ad ogni batch alcuni input casuali, in modo che la rete si addestri su informazioni incomplete, con possibile riduzione dell'overfitting;
- Uno strato finale *Dense* con soli 2 neuroni, uno per classe, che utilizza la softmax come funzione di attivazione, cosicché l'output sia la percentuale di appartenenza dell'immagine a ciascuna delle 2 classi.

La rete così costruita prende come input, nel primo strato, immagini di dimensioni 128x128 pixel, quindi tutte le immagini dovranno essere ridimensionate in tal modo prima di poter essere passate alla rete.

Utilizza come algoritmo di ottimizzazione per trovare i pesi della rete l'algoritmo *Adam*, e ne imposta manualmente il learning rate. Inoltre specifica come misura d'errore che la rete deve minimizzare la *binary_ crossentropy* poiché vi sono solo 2 possibili classi.

Informa la rete di non addestrare i pesi degli strati di input e nascosti, poiché sono già addestrati su ImageNet. Per cui non verrà calcolato alcun gradiente per i livelli "congelati" e i pesi non verranno aggiornati.

Per l'addestramento utilizza:

- 10 epoche con batch di 32 immagini;
- *EarlyStopping* che consente di specificare un numero elevato arbitrario di epoche di addestramento e interrompere l'addestramento una volta che le prestazioni del modello smettono di migliorare sul validation set;
- *Learning Rate Reduction* che riduce il learning rate quando l'accuratezza non aumenta per 3 steps.

datarootsio

La seguente soluzione **si basa sull'architettura MobileNet**, ovvero la prima versione esistente di tale rete neurale, descritta nella sezione 2.4.2.
Come per la soluzione precedente, anche in questo caso per addestrare la rete vengono utilizzati dei pesi pre-addestrati sul dataset *ImageNet*, inoltre non vegnono caricati i livelli di output completamente connessi del modello.
Lo strato finale di output costruito dall'autore è composto nel seguente modo:

- Uno strato *GlobalAveragePooling2D* che usa la media come funzione di aggregazione per applicare il raggruppamento medio sulle dimensioni di altezza e larghezza per tutti i canali delle immagini fino a quando ogni dimensione spaziale è uno, lasciando le altre dimensioni invariate;
- Uno strato *Dense* che applica la funzione di attivazione *ReLU* ai dati introducendo la non linearità;
- Uno strato finale *Dense* con un solo neurone, che utilizza la sigmoide come funzione di attivazione, cosicché l'output sia un valore unico compreso tra 0 e 1. Se tale valore è maggiore di 0.50 allora l'immagine apparterrà ad una classe, altrimenti apparterrà all'altra, essendo appunto solo due le classi possibili.

La rete così costruita prende come input, nel primo strato, immagini di dimensioni 112x112 pixel, quindi tutte le immagini dovranno essere ridimensionate in tal modo prima di poter essere passate alla rete.

Utilizza come algoritmo di ottimizzazione per trovare i pesi della rete l'algoritmo *Adam*, e ne imposta manualmente il learning rate. Inoltre specifica come misura d'errore che la rete deve minimizzare la *binary_ crossentropy* poiché vi sono solo 2 possibili classi.

Informa la rete di non addestrare i pesi degli strati di input e nascosti, poiché sono già addestrati su ImageNet. Per cui non verrà calcolato alcun gradiente per i livelli "congelati" e i pesi non verranno aggiornati.

Per l'addestramento infine utilizza 2 epoche con batch di 32 immagini.

ritikbompilwar

Anche questa soluzione **si basa sull'architettura MobileNetV2**, ed è strutturata in modo pressoché identico alla soluzione di *hritik* descritta sopra.

Utilizza sempre pesi pre-addestrati su *ImageNet*, non carica i livelli di output completamente connessi del modello e costruisce manualmente lo strato finale nello stesso modo.

La motivazione che ha spinto a scegliere questa soluzione è dettata da due fattori, che sono anche le differenze con la soluzione di hritik, ovvero la dimensione delle immagini di input e le dimensioni del campo ricettivo nello strato AveragePooling2D.

La rete neurale prende infatti nello strato di input, immagini di dimensioni 224x224 pixel, che sono quindi nettamente più grandi rispetto all'altra soluzione, i cui input devo avere dimensione 128x128.
Inoltre la dimensione del campo ricettivo nello strato AveragePooling2D è pari a 7x7, mentre nel caso precedente era 2x2. Per cui si otterrà come output del suddetto strato, una matrice di pixel nettamente più piccola.

Il confronto tra le due soluzioni dunque permetterà di osservare se e quanta importanza avrà la dimensione delle immagini di input e la dimensione dell'immagine di output dell'ultimo strato di pooling, poiché il modello dovrà capire da tale immagine risultante, negli strati di output successivi. se ci sono caratteristiche di alto livello comuni con le classi possibili.

karankishanshetty

Questa soluzione **si basa sull'architettura ResNet50**, la quale consiste in una *Rete Residuale*, ovvero una Rete Neurale Convoluzionale estremamente profonda, caratterizzata dalla presenza di *skip connections* che consentono ad uno strato di inviare i risultati all'output di uno strato più in alto nello stack. Gli strati intermedi della rete sono detti *unità residuali* e ciascuna di esse è una piccola Rete Neurale con una skip connection.
Ogni unità residuale è composta da due strati convoluzionali, usando filtri 3x3 e preservando la dimensione spaziale, ricorrendo alla *Batch Normalization* e usando la funzione di attivazione ReLU. Non sono previsti invece, all'interno di questa pila, strati di pooling.
Tale architettura è quindi una variante del modello *ResNet*, ed è composta da 48 strati convoluzionali.
La soluzione carica quindi un modello ResNet50 pre-addestrato, utilizzando pesi pre-addestrati sul dataset *ImageNet*. Tuttavia i livelli di output completamente

connessi del modello non vengono caricati, consentendo di poter aggiungere e addestrare un nuovo livello di output.
Quindi l'autore si occuperà di realizzare personalmente lo strato finale della rete, che sarà così costituito:

- Uno strato *AveragePooling2D* che che dimezza la dimensione delle immagini utilizzando la media come funzione di aggregazione;
- Uno strato *Flatten* che appiattisce l'input trasformandolo in un vettore unidimensionale;
- Uno strato *Dense* che applica la funzione di attivazione $ReLU$ ai dati introducendo la non linearità;
- Uno strato *Dropout* che azzera ad ogni batch alcuni input casuali, in modo che la rete si addestri su informazioni incomplete, con possibile riduzione dell'overfitting;
- Un ultimo strato *Dense* costituito da 2 neuroni, con funzione di attivazione softmax, che produrrà in output la probabilità di appartenenza dell'immagine a ciascuna delle 2 classi possibili.

Il modello di rete finale prende come input immagini di dimensione 224x224 pixel, quindi tutte le immagini dovranno essere ridimensionate in tal modo prima di poter essere passate alla rete.

Utilizza *Adam* come algoritmo di ottimizzazione per trovare i pesi della rete e ne imposta manualmente il learning rate, mentre come misura d'errore che la rete deve minimizzare utilizza la *binary_ crossentropy*.

Informa la rete di non addestrare i pesi degli strati di input e nascosti, poiché sono già addestrati su *ImageNet*. Per cui non verrà calcolato alcun gradiente per i livelli "congelati" e i pesi non verranno aggiornati.

Per l'addestramento infine utilizza 2 epoche con batch di 32 immagini.

aieml

Questa soluzione, a differenza di tutte le altre proposte fino ad ora, **non si basa su nessuna architettura precostruita**, bensì è stata realizzata interamente dall'autore.
Egli crea prima di tutto un modello *sequenziale*. Questo è il tipo più semplice di modello di Keras, per le reti neurali che sono semplicemente composte da un singolo stack di livelli, collegati sequenzialmente.

Dopo aver creato la base del modello quindi, costruisce ciascuno strato della rete e lo aggiunge al modello:

- Il primo strato creato è quello di input, ed è uno strato *Conv2D*, ovvero uno strato convoluzionale, con 200 neuroni. Esso utilizza filtri 3x3 e prende in input immagini di dimensione 100x100 e profondità pari a 3;
- In seguito aggiunge uno strato *ReLU* che introduce non linearità nei i dati;
- Poi aggiunge uno strato *MaxPooling2D*, con dimensione del campo ricettivo pari a (2, 2). Questo strato riduce la dimensione dell'immagine, e per farlo utilizza il massimo come funzione di aggregazione;
- Successivamente aggiunge altri 3 strati che sono identici ai 3 appena creati;
- Dopodiché aggiunge uno strato *Flatten* che appiattisce l'input riducendo ad un vettore unidimensionale, e uno strato *Dropout* che azzera ad ogni batch alcuni input casuali;
- Infine aggiunge uno strato *Dense* con 50 neuroni e funzione di attivazione relu, e uno strato *Dense* di output con 2 neuroni, uno per classe, e funzione di attivazione softmax che mostra la probabilità di appartenenza dell'immagine alle 2 classi.

Dopo aver costruito la rete, compila il modello specificando *Adam* come algoritmo di ottimizzazione per addestrare i pesi della rete e la *categorical_ crossentropy* come misura d'errore che deve minimizzare la rete.

Capitolo 4

Sviluppo

Si procede con la descrizione del lavoro svolto sui dataset, e lo sviluppo dei modelli descritti nei capitoli precedenti.

4.1 Preprocessamento

Affinché le reti neurali di cui sono composte le soluzioni scelte possano essere addestrate correttamente, occorre che i dataset siano organizzati secondo una precisa disposizione, ovvero con immagini divise in set di training e di validation, i quali a loro volta dovranno essere suddivisi in immagini con la mascherina e senza la mascherina.

Tuttavia, come specificato nella sezione 1.2, uno dei tre dataset non è presuddiviso in immagini di training e di validation, per cui si procede prima di tutto a selezionare casualmente circa il 30% delle immagini dalla cartella contenente quelle con la mascherina e dalla cartella che comprende invece quelle senza mascherina, e con queste immagini si va a creare il set di validation del dataset.

Nel momento in cui tutti i dataset sono pronti e le soluzioni sono state trovate, si può procedere oltre. Prima di poter addestrare i modelli però, occorre preprocessare le immagini, affinché abbiano tutte le caratteristiche per essere utilizzate dalle reti neurali:

- Ogni immagine, dopo essere stata caricata in memoria, viene ridimensionata ad un'altezza e una larghezza necessarie per potere essere accettata dalla rete;

- Ciascun immagine viene poi convertita in un array NumPy;

- Dopodiché occorre adeguare ciascun immagine al formato che il modello richiede, e per farlo si utilizza una specifica funzione, chiamata *preprocess_input*, la quale aggiunge una dimensione alle immagini che consente di creare una batch di immagini, poiché appunto durante l'addestramento ad ogni iterazione il modello lavora su batch di immagini.

Oltre alle immagini occorre lavorare anche sulle y, ovvero sulle etichette delle immagini. Si crea quindi una lista di etichette, che conterrà due possibili valori corrispondenti al nome della cartella in cui sono contenute le immagini, quindi *WithMask* e *WithoutMask*. Successivamente si applica la codifica *one-hot* [22] alle etichette, per cui si avranno solo valori 0 (per le immagini con mascherina) e 1 (per le immagini senza mascherina).

A questo punto quasi tutti i modelli applicano la *data augmentation* alle immagini. Essa consiste in un insieme di tecniche che consento di incrementare la quantità e la diversità di dati, aggiungendo copie leggermente modificate dei dati già esistenti. Quindi non si collezionano nuovi dati, piuttosto si trasformano i dati già presenti. Ad esempio ruotando l'immagine, cambiando il livello di zoom, ritagliandola o ancora cambiando il livello di luminosità dell'immagine e altro ancora.

Ora si può procedere con l'addestramento delle soluzioni ripetendo il preprocessamento delle immagini per ogni dataset.

4.2 Valutazione dei Modelli

Una volta che tutte le soluzioni sono state addestrate su ciascun dataset, si procede alla valutazione di ognuna di esse, e per farlo si fa ricorso ai testset trovati in precedenza proprio per soddisfare questo requisito.

Dopo aver caricato in memoria i testset, si utilizzano i file delle etichette per:

- Estarre i volti dall'immagine a cui si riferisce ciascun file, tramite i bounding box del volto contenuti nel file stesso;

- Stabilire se ciascun volto indossa o meno una mascherina.

Quando tutti i volti sono stati estratti, prima di poter essere utilizzati dalle soluzioni per fare previsioni, occorre che siano preprocessati in modo adeguato. È necessario infatti che le immagini dei volti siano ridimensionate alle stesse

dimensioni delle immagini su cui è stata addestrata ciascuna soluzione, altrimenti esse verranno rifiutate dalla rete.
Successivamente al ridimensionamento, si converte lo spazio dei colori dell'immagine in RGB, la si trasforma in un array, e poiché le reti lavorano con batch di immagini si utilizza anche in questo caso la funzione *preprocess_input* per aggiungere un'ulteriore dimensione all'immagine.

A questo punto, per ogni modello addestrato su ciascun dataset, si effettuano le previsioni sui testset adeguatamente preprocessati, e si utilizza la funzione di libreria *classification_report* per calcolare e visualizzare le metriche di accuratezza del modello, ovvero *Precision*, *Recall* e *F1-measure*.

- La Precision mostra la percentuale di immagini classificate correttamente in ciascuna delle due classi (con e senza mascherina), tra quelle che il modello ha classificato come appartenenti a ciascuna di esse;

- La Recall mostra la percentuale di immagini classificate correttamente in ciascuna delle due classi, tra quelle che realmente appartengono a ciascuna di esse;

- L'F1-measure si limita a riassumere precision e recall in un unico valore.

Confrontando le metriche ottenute da ciascuna soluzione si determina la miglior soluzione generale.

Sono stati effettuati anche degli esperimenti per verificare il comportamento delle soluzioni in determinate condizioni, per testare quindi quali prevalgono in particolari situazioni con una difficoltà più elevata e quali invece prevalgono in condizioni più semplici. Tali esperimenti verranno trattati in maniera approfondita nel capitolo successivo.

4.3 Risultati

Di seguito vengono riportate in due tabelle riassuntive, le principali metriche di accuratezza ottenute dai vari modelli sui testset.
Si mostrano sia i risultati di ogni modello per ciascun dataset su cui è stato addestrato, sia la media dei risultati ottenuti.

	DATASET 1		**DATASET 2**		**DATASET 3**		**MEDIA**	
Soluzione\Metrica	PREC	REC	PREC	REC	PREC	REC	PREC	REC
hritik	0.89	0.75	0.71	0.79	0.71	0.80	0.77	0.78
datarootsio	0.67	0.67	0.70	0.69	0.66	0.74	0.68	0.70
ritikbompilwar	0.63	0.71	0.62	0.69	0.74	0.81	0.66	0.74
karankishanshetty	0.40	0.50	0.43	0.48	0.60	0.50	0.48	0.49
aieml	0.40	0.50	0.40	0.50	0.40	0.50	0.40	0.50

Tabella 4.1: Precision e Recall delle soluzioni ottenute sui testset.

	DATASET 1		**DATASET 2**		**DATASET 3**		**MEDIA**	
Soluzione\Metrica	ACC	F1	ACC	F1	ACC	F1	ACC	F1
hritik	0.89	0.79	0.78	0.72	0.77	0.72	0.81	0.74
datarootsio	0.80	0.66	0.82	0.69	0.64	0.61	0.75	0.65
ritikbompilwar	0.65	0.61	0.57	0.55	0.83	0.76	0.68	0.64
karankishanshetty	0.80	0.45	0.76	0.44	0.20	0.16	0.59	0.35
aieml	0.80	0.45	0.80	0.45	0.80	0.45	0.80	0.45

Tabella 4.2: Accuratezza e F1-Measure delle soluzioni ottenute sui testset.

La soluzione di *hritik* è quella che ottiene una maggiore precision, recall e f1-measure sui testset, sia pesate (cioè considerando il numero di istanze in ogni classe) che non.

La soluzione di *ritkbompilwar* è quella che classifica meglio le immagini di volti senza mascherine nella relativa classe, infatti ha una recall sulla classe WithoutMask maggiore rispetto alle altre soluzioni.

È facile vedere che la soluzione di *aieml* è la peggiore di tutte, seguita da vicino da quella di *karankishanshetty*. Infatti, entrambi trovano molta difficoltà nel classificare i volti senza mascherina.

Il modello *aieml* in particolare classifica tutte le immagini in un'unica classe, quella relativa ai volti con le mascherine. Quindi anche se sembra avere una buona precision media (0.80), questo non è un modello valido, così come non lo è quello di karankishanshetty, che anche se riesce a classificare alcune immagini senza mascherina, è comunque inadeguato ai fini dell'obiettivo.

Dai risultati ottenuti, la soluzione di *hritik* sembra essere la migliore del lotto. Un'analisi ancora più approfondita verrà condotta tramite lo svolgimento di alcuni esperimenti, che permetteranno di verificare se tale soluzione prevalga sulle altre anche in determinate condizioni.

4.4 Deployment miglior soluzione

Una volta determinata definitivamente quale sia la miglior soluzione, si è provveduto a creare una semplice **WebApp** che, attraverso l'utilizzo della webcam del dispositivo su cui è in esecuzione, effettua previsioni in tempo reale. Ovvero la webapp prende ciascun frame catturato dallo stream della webcam, determina i bounding box di ciascun volto raffigurato nell'immagine, estrare tutti i volti e ognuno di essi viene passato alla rete della miglior soluzione, la quale effettua la previsione e determina se su ciascun volto inquadrato dalla webcam è presente o meno una mascherina e con quale probabilità.
Dopodiché la webapp, su ogni volto disegna a video un riquadro di colore diverso a seconda del risultato della predizione, mostrando l'etichetta della classe predetta e la relativa probabilità.

Capitolo 5

Esperimenti

Come specificato nel capitolo precedente, oltre a verificare quali risultati ottengono le diverse soluzioni sui testset, si vuole anche esaminare il comportamento di queste soluzioni in svariate condizioni di difficoltà.
Per testare in questo modo le soluzioni, sono stati effettuati una serie di esperimenti. Questi esperimenti sono stati divisi in 3 categorie:

- Esperimento basato sul **quantitativo di persone** nelle immagini. Sono stati estratti dalle immagini due sottoinsiemi, un sottoinsieme contenente le immagini in cui sono raffigurate un numero ristretto di persone, e un sottoinsieme comprendente immagini con un elevato numero di persone;
- Un altro tipo di esperimento è basato sulla **region of interest** dei volti, ovvero sulla loro grandezza, che varia a seconda della distanza rispetto alla telecamera. Anche in questo caso sono stati estratti due sottoinsiemi, uno contenente solamente volti con una grande region of interest, quindi volti vicino alla telecamera, e uno contenente esclusivamente volti con una piccola region of interest, per cui volti più lontani dalla telecamera;
- Infine, per l'ultimo esperimento sono state **combinate le due categorie precedenti**, per crearne una terza. Ovvero si è sperimentato il comportamento delle soluzioni con volti vicino alla telecamera e in numerosità al massimo di poche persone, e con volti con distanza più ampia dalla telecamera e in numerosità più elevata. I due estremi in sostanza corrispondo alla difficoltà minima e massima della predizione.

Si elencano i passaggi svolti per raggiungere i risultati degli esperimenti appena descritti.

N.B.: A causa dei cattivi risultati ottenuti dalle soluzioni di *karankishanshetty* e *aieml*, vengono riportati solo i risultati degli esperimenti delle 3 migliori soluzioni, rispettivamente quelli di *hritik*, *datarootsio* e *ritikbompilwar*.

5.1 Preprocessamento dati

5.1.1 Caricamento dati e librerie

Vengono importate tutte le librerie necessarie, e vengono caricati in memoria i testset e le soluzioni addestrare sui diversi dataset.

```
In [ ]: from tensorflow.keras.applications.mobilenet_v2
             import preprocess_input
        from tensorflow.keras.preprocessing.image
             import img_to_array
        from sklearn.metrics import classification_report
        from tensorflow.keras.models import load_model
        import xml.etree.ElementTree as ET
        import pandas as pd
        import numpy as np
        import cv2
        import os
```

```
In [ ]: !gdown 'https://drive.google.com/
                 uc?id=1krSPHkMv8vuMLiLYkj6Za8Dwbdrz2Giq'
        !unzip "Testset.zip"
```

```
In [ ]: !gdown 'https://drive.google.com/
                 uc?id=1ByPsuirCAnpDzsikOiLwfsh7dGq8AWlT'
        !unzip "Soluzioni.zip"
```

```
In [ ]: # Gli esperimenti sono stati ripetuti per tutte le soluzioni,
        # quindi il processo di caricamento del modello è lo stesso
        # per ciascuna di esse, n è il nome della cartella
        # in cui si trova un modello, addestrato su ciascun dataset.
        model = []
        model.append(load_model('Soluzioni/n/model_ds1.h5'))
        model.append(load_model('Soluzioni/n/model_ds2.h5'))
        model.append(load_model('Soluzioni/n/model_ds3.h5'))
```

5.1.2 Estrazione dei volti

Per poter effettuare gli esperimenti occorre estrarre i volti da ciascun testset, e per farlo si ricorre al file delle etichette con i bounding box per ognuno di essi.

```
In [ ]: # Estrae i volti del testset passato come argomento.
        def extract_faces(imagePaths, labelPaths, mask_tag_text):
```

```
        faces = []
        labels = []
        rois = []
        num_faces = []
        i = 0
        q = 0
        for imagePath in imagePaths:
          tree = ET.parse(labelPaths[i])
          root = tree.getroot()
          img = cv2.imread(imagePath)
          (height, width) = img.shape[:2]
          for elem in root:
            if (elem.tag == "object"):
              for subelem in elem:
                if (subelem.tag == "name"):
                  q = q + 1
                  if (subelem.text == mask_tag_text):
                    labels.append(0)
                  else:
                    labels.append(1)
                if (subelem.tag == "bndbox"):
                  x1 = max(0, int(subelem[0].text))
                  x2 = min(width-1, int(subelem[2].text))
                  y1 = max(0, int(subelem[1].text))
                  y2 = min(height-1, int(subelem[3].text))
                  if (x1 > x2):
                    x1, x2 = x2, x1
                  faces.append(img[y1:y2, x1:x2])
                  rois.append((x2 - x1) * (y2 - y1))
          for m in np.arange(q):
            num_faces.append(q)
          q = 0
          i = i + 1
        return faces, labels, rois, num_faces
```

```
In [ ]: # Legge le immagini del 1° Testset e ne estrae i volti
        imagePaths_ts1 = []
        for file in os.listdir("Testset/1/images"):
            if file.endswith(".jpg"):
                imagePaths_ts1.append(os.path.join("Testset/1/images",
                                                   file))
        imagePaths_ts1.sort()
```

```
        labelPaths_ts1 = []
        for file in os.listdir("Testset/1/labels"):
            if file.endswith(".xml"):
                labelPaths_ts1.append(os.path.join("Testset/1/labels",
                                                   file))
        labelPaths_ts1.sort()
```

```
In [ ]: (faces1, labels1, rois1, num_faces1) =
            extract_faces(imagePaths_ts1, labelPaths_ts1, "mask")
```

```
In [ ]: # Legge le immagini del 2° Testset e ne estrae i volti
        imagePaths_ts2 = []
        for file in os.listdir("Testset/3/images"):
            if file.endswith(".png"):
                imagePaths_ts2.append(os.path.join("Testset/3/images",
                                                   file))
        imagePaths_ts2.sort()
        labelPaths_ts2 = []
        for file in os.listdir("Testset/3/labels"):
            if file.endswith(".xml"):
                labelPaths_ts2.append(os.path.join("Testset/3/labels",
                                                   file))
        labelPaths_ts2.sort()
```

```
In [ ]: (faces2, labels2, rois2, num_faces2) =
            extract_faces(imagePaths_ts2, labelPaths_ts2, "with_mask")
```

```
In [ ]: # Converte le etichette in array NumPy
        labels1_a = np.array(labels1)
        labels2_a = np.array(labels2)
```

5.1.3 Unione dei volti

```
In [ ]:# Raggruppa tutte le informazioni estratte
         faces = faces1 + faces2
        rois = rois1 + rois2
        nfaces = num_faces1 + num_faces2
        labels_a = np.concatenate([labels1_a, labels2_a])
```

5.1.4 Funzioni di Utility

```
In [ ]: # La seguente funzione calcola la media delle metriche
        # di accuratezza che verranno calcolate, ovvero precision,
```

```
# recall ed f1-measure, durante l'esperimento per un
# modello, su ogni dataset su cui è stato addestrato.
def compute_metrics(ds1_ts, ds2_ts, ds3_ts):
  prec_wm = np.mean([ds1_ts['WithMask']['precision'],
                     ds2_ts['WithMask']['precision'],
                     ds3_ts['WithMask']['precision']])
  rec_wm = np.mean([ds1_ts['WithMask']['recall'],
                    ds2_ts['WithMask']['recall'],
                    ds3_ts['WithMask']['recall']])
  f1_wm = np.mean([ds1_ts['WithMask']['f1-score'],
                   ds2_ts['WithMask']['f1-score'],
                   ds3_ts['WithMask']['f1-score']])

  prec_wom = np.mean([ds1_ts['WithoutMask']['precision'],
                      ds2_ts['WithoutMask']['precision'],
                      ds3_ts['WithoutMask']['precision']])
  rec_wom = np.mean([ds1_ts['WithoutMask']['recall'],
                     ds2_ts['WithoutMask']['recall'],
                     ds3_ts['WithoutMask']['recall']])
  f1_wom = np.mean([ds1_ts['WithoutMask']['f1-score'],
                    ds2_ts['WithoutMask']['f1-score'],
                    ds3_ts['WithoutMask']['f1-score']])

  acc_avg = np.mean([ds1_ts['accuracy'],
                     ds2_ts['accuracy'],
                     ds3_ts['accuracy']])

  prec_avg = np.mean([ds1_ts['macro avg']['precision'],
                      ds2_ts['macro avg']['precision'],
                      ds3_ts['macro avg']['precision']])
  rec_avg = np.mean([ds1_ts['macro avg']['recall'],
                     ds2_ts['macro avg']['recall'],
                     ds3_ts['macro avg']['recall']])
  f1_avg = np.mean([ds1_ts['macro avg']['f1-score'],
                    ds2_ts['macro avg']['f1-score'],
                    ds3_ts['macro avg']['f1-score']])

  prec_avg_w = np.mean([ds1_ts['weighted avg']['precision'],
                        ds2_ts['weighted avg']['precision'],
                        ds3_ts['weighted avg']['precision']])
  rec_avg_w = np.mean([ds1_ts['weighted avg']['recall'],
```

```
                                ds2_ts['weighted avg']['recall'],
                                ds3_ts['weighted avg']['recall']])
            f1_avg_w = np.mean([ds1_ts['weighted avg']['f1-score'],
                                ds2_ts['weighted avg']['f1-score'],
                                ds3_ts['weighted avg']['f1-score']])

            df_met = pd.DataFrame(columns=['precision', 'recall',
                                           'f1-score', 'support'],
                                  index=['WithMask', 'WithoutMask',
                                   'macro avg', 'weighted avg'],
                                  data=[[prec_wm, rec_wm, f1_wm,
                                         ds1_ts['WithMask']['support']],
                                        [prec_wom, rec_wom, f1_wom,
                                         ds1_ts['WithoutMask']['support']],
                                        [prec_avg, rec_avg, f1_avg,
                                         ds1_ts['macro avg']['support']],
                                        [prec_avg_w, rec_avg_w, f1_avg_w,
                                       ds1_ts['weighted avg']['support']]])
            print("{}\naccuracy: {}".format(df_met, acc_avg))
            return df_met, acc_avg
```

```
In [ ]: # Preprocessa le immagini da passare alla rete neurale.
        def preprocess_image(faces, height, width):
            X_test = []
            for face in faces:
                face_input = cv2.resize(face, dsize=(height, width))
                face_input = cv2.cvtColor(face_input, cv2.COLOR_BGR2RGB)
                face_input = img_to_array(face_input)
                face_input = preprocess_input(face_input)
                X_test.append(face_input)
            X_test = np.array(X_test, dtype="float32")
            return X_test
```

5.2 Esperimenti svolti

Ora che le librerie sono state importate, i dati sono stati caricati e le funzioni di utility sono state definite, si procede con gli esperimenti.

5.2.1 Esperimento Numero Volti

Il primo esperimento a cui si vogliono sottoporre le soluzioni è quello relativo al numero dei volti nelle immagini. Si procede col definire una funzione, la quale provvedere ad estrarre dall'insieme dei volti, solamente quelli che appartengono a immagini raffiguranti un numero di persone comprese in un certo intervallo. Dopodiché calcola le metriche di accuratezza.

```
In [ ]: def compute_metrics_by_faces(model, min_num_faces,
                                     max_num_faces):
            nfaces_idxs = [i for i,n in enumerate(nfaces)
                           if n >= min_num_faces
                           and n <= max_num_faces]
            X_test = preprocess_image([f for i,f in enumerate(faces)
                                     if i in nfaces_idxs], 128, 128)
            predIdxs = model.predict(X_test)
            predIdxs = np.argmax(predIdxs, axis=1)
            cr = classification_report(labels_a[nfaces_idxs],
                 predIdxs, target_names=["WithMask", "WithoutMask"],
                 output_dict=True)
            print(classification_report(labels_a[nfaces_idxs],
                  predIdxs, target_names=["WithMask", "WithoutMask"]))
            return cr
```

Caso Semplice

Nel caso più semplice saranno presenti poche persone nelle immagini, per cui tale intervallo sarà compreso tra 1 e 3.

```
In [ ]: cr_faces_ds1 = compute_metrics_by_faces(model[i], 1, 3)
```

Con questa chiamata sono state calcolate le diverse metriche di accuratezza di un modello addestrato su uno dei 3 dataset, solamente su immagini con poche persone. i corrisponde al dataset su cui è addestrato il modello corrente.

Si seguirà la medesima procedura per lo stesso modello addestrato sugl'altri dataset e per tutti gli altri modelli.

Una volta calcolate le metriche di un modello su ogni dataset, si effettua una media dei risultati per avere un risultato generale dell'accuratezza della soluzione. Si ricorre quindi alla funzione definita in precedenza, e si ripete la chiamata per ogni modello.

```
In [ ]: compute_metrics(cr_faces_ds1, cr_faces_ds2, cr_faces_ds3)
```

Risultati hritik

```
              precision    recall  f1-score  support
WithMask       0.935145  0.951726  0.942854     1526
WithoutMask    0.770755  0.665570  0.698329      304
macro avg      0.852950  0.808648  0.820592     1830
weighted avg   0.907836  0.904189  0.902234     1830
accuracy: 0.9041894353369763
```

Risultati datarootsio

```
              precision    recall  f1-score  support
WithMask       0.912841  0.947794  0.929542     1526
WithoutMask    0.689397  0.541667  0.596059      304
macro avg      0.801119  0.744730  0.762801     1830
weighted avg   0.875723  0.880328  0.874144     1830
accuracy: 0.8803278688524591
```

Risultati ritikbompilwar

```
              precision    recall  f1-score  support
WithMask       0.939676  0.674530  0.774237     1526
WithoutMask    0.380058  0.781798  0.487090      304
macro avg      0.659867  0.728164  0.630664     1830
weighted avg   0.846712  0.692350  0.726536     1830
accuracy: 0.6923497267759563
```

- L'accuratezza media delle 3 soluzioni è rispettivamente: 0,90, 0,88 e 0,69.
- L'F1-measure media invece è rispettivamente: 0,90, 0,87 e 0,73.
- La soluzione di hritik5102 è quella che classifica correttamente il maggior numero di immagini, anche se quella di datarootsio è molto vicina ad essa.

Caso Difficile

Nel caso più complesso ci saranno molte più persone, per cui si decide di settare un intervallo compreso tra 8 e 10.

```
In [ ]: cr_faces_ds1 = compute_metrics_by_faces(model[i], 8, 10)
```

Anche in questo caso si seguirà la medesima procedura per lo stesso modello addestrato sugl'altri dataset e per tutti gli altri modelli, per poi calcolare la media dei risultati ottenuti.

```
In [ ]: compute_metrics(cr_faces_ds1, cr_faces_ds2, cr_faces_ds3)
```

Risultati hritik

```
              precision    recall  f1-score  support
WithMask       0.931562  0.818926  0.865378     1099
WithoutMask    0.580015  0.733831  0.600706      268
macro avg      0.755788  0.776379  0.733042     1367
weighted avg   0.862641  0.802243  0.813489     1367
accuracy: 0.8022433552792002
```

Risultati datarootsio

```
              precision    recall  f1-score  support
WithMask       0.899641  0.791022  0.827499     1099
WithoutMask    0.465531  0.593284  0.481593      268
macro avg      0.682586  0.692153  0.654546     1367
weighted avg   0.814534  0.752256  0.759684     1367
accuracy: 0.7522555474274567
```

Risultati ritikbompilwar

```
              precision    recall  f1-score  support
WithMask       0.932618  0.683045  0.780967     1099
WithoutMask    0.411835  0.797264  0.530967      268
macro avg      0.672226  0.740154  0.655967     1367
weighted avg   0.830518  0.705438  0.731955     1367
accuracy: 0.7054376981224091
```

- L'accuratezza media delle 3 soluzioni è rispettivamente: 0,80, 0,75 e 0,71.
- L'F1-measure media invece è rispettivamente: 0,81, 0,76 e 0,73.
- Anche in questo caso la soluzione di hritik5102 è quella che classifica correttamente il maggior numero di immagini.

5.2.2 Esperimento ROI Volti

Non tutti i volti sono uguali. Ci possono essere volti più dettagliati e volti meno definiti. Questo può comportare una differenza nell'accuratezza della predizione da parte delle soluzioni. Il livello di dettaglio di un volto in questo caso è determinato dalla vicinanza dello stesso con la telecamera. È infatti estremamente intuibile che volti in primo piano rappresentino dei dati con cui è più facile lavorare per le reti. Per ciò si dividono le immagini in due sottoinsiemi, uno con volti vicini e uno con volti lontani dalla telecamera, per poi calcolarne le metriche di accuratezza dei modelli.

```
In [ ]: def compute_metrics_by_rois(model, roi_min, roi_max):
            rois_idxs = [i for i,r in enumerate(rois)
                         if r >= roi_min and r <= roi_max]
            X_test = preprocess_image([f for i,f in enumerate(faces)
                          if i in rois_idxs], 128, 128)
            predIdxs = model.predict(X_test)
            predIdxs = np.argmax(predIdxs, axis=1)
            cr = classification_report(labels_a[rois_idxs],
                 predIdxs, target_names=["WithMask", "WithoutMask"],
                 output_dict=True)
            print(classification_report(labels_a[rois_idxs],
                  predIdxs, target_names=["WithMask", "WithoutMask"]))
            return cr
```

Caso semplice

Nel caso migliore, si è deciso di considerare solamente volti con una region of interest compresa tra 10.000 e 25.000 pixel.

```
In [ ]: cr_rois_ds1 = compute_metrics_by_rois(model[i], 10_000,
                                              25_000)
```

Si esegue la medesima procedura per lo stesso modello addestrato sugl'altri dataset e per tutti gli altri modelli, per poi calcolare la media dei risultati ottenuti.

```
In [ ]: compute_metrics(cr_rois_ds1, cr_rois_ds2, cr_rois_ds3)
```

Risultati hritik

```
              precision    recall  f1-score   support
WithMask       0.947730  0.952164  0.949469       878
```

```
WithoutMask    0.817046  0.773984  0.786412      205
macro avg      0.882388  0.863074  0.867941     1083
weighted avg   0.922993  0.918436  0.918604     1083
accuracy: 0.9184364419821484
```

Risultati datarootsio

```
              precision    recall  f1-score  support
WithMask       0.938042  0.930524  0.934258      878
WithoutMask    0.712063  0.736585  0.723988      205
macro avg      0.825052  0.833555  0.829123     1083
weighted avg   0.895267  0.893813  0.894456     1083
accuracy: 0.8938134810710988
```

Risultati ritikbompilwar

```
              precision    recall  f1-score  support
WithMask       0.952567  0.621488  0.737835      878
WithoutMask    0.398003  0.868293  0.526318      205
macro avg      0.675285  0.744890  0.632076     1083
weighted avg   0.847594  0.668206  0.697797     1083
accuracy: 0.6682056017236073
```

- L'accuratezza media delle 3 soluzioni è rispettivamente: 0,92, 0,89 e 0,69.
- L'F1-measure media invece è rispettivamente: 0,92, 0,89 e 0,69.
- Le soluzioni di hritik5102 e datarootsio ottengono buoni risultati con volti vicini.

Caso Difficile

Nel caso più difficile, sono stati presi in considerazione solamente volti con una region of interest al massimo di 3.000 pixel.

```
In [ ]:cr_rois_ds1 = compute_metrics_by_rois(model[i], 0, 3_000)
```

Si ripete la medesima procedura per lo stesso modello addestrato sugl'altri dataset e per tutti gli altri modelli, per poi calcolare la media dei risultati ottenuti.

```
In [ ]:compute_metrics(cr_rois_ds1, cr_rois_ds2, cr_rois_ds3)
```

Risultati hritik

```
              precision    recall  f1-score  support
WithMask       0.916619  0.765733  0.822985     4068
WithoutMask    0.555505  0.719637  0.565715     1139
macro avg      0.736062  0.742685  0.694350     5207
weighted avg   0.837628  0.755649  0.766709     5207
accuracy: 0.7556494462582423
```

Risultati datarootsio

```
              precision    recall  f1-score  support
WithMask       0.876577  0.698460  0.741694     4068
WithoutMask    0.389443  0.565701  0.417648     1139
macro avg      0.633010  0.632080  0.579671     5207
weighted avg   0.770020  0.669419  0.670811     5207
accuracy: 0.6694193713590679
```

Risultati ritikbompilwar

```
              precision    recall  f1-score  support
WithMask       0.922589  0.662652  0.763734     4068
WithoutMask    0.420537  0.800117  0.543298     1139
macro avg      0.671563  0.731384  0.653516     5207
weighted avg   0.812768  0.692721  0.715515     5207
accuracy: 0.6927213366621855
```

- L'accuratezza media delle 3 soluzioni è rispettivamente: 0,76, 0,67 e 0,69.
- L'F1-measure media invece è rispettivamente: 0,77, 0,67 e 0,72.
- Le soluzioni di hritik5102 e datarootsio perdono entrambe molta precisione, mentre aumenta quella di ritkbompilwar, di cui aumenta la recall sui volti senza mascherina. Quindi quest'ultima soluzione con volti più distanti può capire meglio se non hanno una mascherina. Anche in questo caso, però, in generale prevale la soluzione di hritik5102.

5.2.3 Esperimento Numero e ROI Volti

Ora che è stato analizzato come si comportano le soluzioni quando devono lavorare con volti piccoli o grandi o con immagini con pochi volti o molti, si può

pensare di mettere insieme le due cose, aggiungendo così un'ulteriore livello di difficoltà per le soluzioni. Per cui si estraggono i volti con una region of interest compresa in un certo intervallo, si estraggono i volti appartenenti a immagini che contengono un certo numero di persone minimo e massimo. Dopodichè si intersecano i due sottoinsiemi e se ne ottiene un terzo, il quale contiene i volti che soddisfano entrambe le condizioni. A questo punto si calcolano le metriche di accuratezza sull'insieme dei volti ottenuto.

```
In [ ]: def compute_metrics_by_faces_and_rois(model, min_num_faces,
                                    max_num_faces, roi_min, roi_max):
          rois_idxs = [i for i,r in enumerate(rois)
                       if r >= roi_minand r <= roi_max]
          nfaces_idxs = [i for i,n in enumerate(nfaces)
                         if n >= min_num_faces
                         and n <= max_num_faces]
          idxs = list(set(rois_idxs) & set(nfaces_idxs))
          X_test = preprocess_image([f for i,f in enumerate(faces)
                                     if i in idxs], 128, 128)
          predIdxs = model.predict(X_test)
          predIdxs = np.argmax(predIdxs, axis=1)
          cr = classification_report(labels_a[idxs], predIdxs,
               target_names=["WithMask", "WithoutMask"],
               output_dict=True)
          print(classification_report(labels_a[idxs], predIdxs,
                target_names=["WithMask", "WithoutMask"]))
          return cr
```

Caso Semplice

Nel caso più semplice quindi sono stati esaminati esclusivamente volti con una region of interest compresa tra 15.000 e 80.000 pixel e un numero massimo di persone raffigurate nell'immagine uguale a 5.

```
In [ ]: cr_faces_rois_ds1 = compute_metrics_by_faces_and_rois(
                            model[i], 1, 5, 15_000, 80_000)
```

Si esegue la medesima procedura per lo stesso modello addestrato sugl'altri dataset e per tutti gli altri modelli, per poi calcolare la media dei risultati ottenuti.

```
In [ ]:  compute_metrics(cr_faces_rois_ds1,
                        cr_faces_rois_ds2,
                        cr_faces_rois_ds3)
```

Risultati hritik

```
              precision    recall  f1-score  support
WithMask       0.874435  0.907946  0.890648      688
WithoutMask    0.044936  0.035842  0.038955       93
macro avg      0.459685  0.471894  0.464801      781
weighted avg   0.775660  0.804097  0.789230      781
accuracy: 0.8040973111395647
```

Risultati datarootsio

```
              precision    recall  f1-score  support
WithMask       0.874238  0.909399  0.891442      688
WithoutMask    0.045282  0.032258  0.037530       93
macro avg      0.459760  0.470829  0.464486      781
weighted avg   0.775528  0.804951  0.789760      781
accuracy: 0.8049509176269739
```

Risultati ritikbompilwar

```
              precision    recall  f1-score  support
WithMask       0.871008  0.613372  0.706829      688
WithoutMask    0.088910  0.326165  0.134789       93
macro avg      0.479959  0.469768  0.420809      781
weighted avg   0.777878  0.579172  0.638711      781
accuracy: 0.579172001707213
```

- L'accuratezza media delle 3 soluzioni è rispettivamente: 0,80, 0,80 e 0,58.
- L'F1-measure media invece è rispettivamente: 0,79, 0,79 e 0,64.
- Le soluzioni di hritik5102 e datarootsio ottengono dei risultati accettabili quando lavorano con immagini con pochi volti e vicini alla telecamera.

Caso Difficile

Nel caso più complicato invece sono stati esaminati unicamente volti con una region of interest massima di 3.000 pixel e un numero di persone presenti nell'immagine compreso tra 10 e 15.

```
In [ ]: cr_faces_rois_ds1 = compute_metrics_by_faces_and_rois(
                            model[i], 10, 15, 0, 3_000)
```

Si esegue la medesima procedura per lo stesso modello addestrato sugl'altri dataset e per tutti gli altri modelli, per poi calcolare la media dei risultati ottenuti.

```
In [ ]: compute_metrics(cr_faces_rois_ds1,
                        cr_faces_rois_ds2,
                        cr_faces_rois_ds3)
```

Risultati hritik

```
              precision    recall  f1-score  support
WithMask       0.773778  0.667392  0.705844      919
WithoutMask    0.303331  0.412082  0.323367      309
macro avg      0.538555  0.539737  0.514605     1228
weighted avg   0.655400  0.603149  0.609602     1228
accuracy: 0.6031487513572205
```

Risultati datarootsio

```
              precision    recall  f1-score  support
WithMask       0.767525  0.633660  0.654511      919
WithoutMask    0.279948  0.407767  0.295869      309
macro avg      0.523736  0.520713  0.475190     1228
weighted avg   0.644836  0.576819  0.564266     1228
accuracy: 0.5768186753528773
```

Risultati ritikbompilwar

```
              precision    recall  f1-score  support
WithMask       0.762888  0.575626  0.650358      919
WithoutMask    0.269713  0.464941  0.336727      309
macro avg      0.516300  0.520283  0.493543     1228
weighted avg   0.638791  0.547774  0.571440     1228
accuracy: 0.5477741585233442
```

- L'accuratezza media delle 3 soluzioni è rispettivamente: 0,60, 0,57 e 0,55.
- L'F1-measure media invece è rispettivamente: 0,61, 0,56 e 0,57.
- La precisione delle soluzioni cala notevolmente, al punto da essere quasi equivalente in questo caso. Si nota quindi come sia difficile per tutte le soluzioni lavorare con volti lontani, per cui poco definiti, e presenti in immagini con molte persone.

Conclusioni

Il problema iniziale richiedeva l'utilizzo di un modello che fosse in grado di stabilire se una persona indossasse una mascherina protettiva sul volto.

Dopo aver cercato e trovato varie soluzioni costruite per questo scopo, e dopo aver testato ciascuna di esse su delle specifiche immagini di persone, si è giunti alla conclusione che **la soluzione di hritik, basata sull'architettura MobileNetV2, sia la migliore del lotto**.

Mentre invece le soluzioni peggiori sono quelle di karankishanshetty, basata su ResNet50, e di aieml, costruita direttamente dall'autore, le quali fanno molta fatica con volti che non indossano la mascherina, classificando quasi tutte le istanze come immagini con la mascherina.

Le restanti due soluzioni, ovvero quelle di datarootsio e ritikbompilwar, basate rispettivamente su MobileNet e MobileNetV2, costituiscono una via di mezzo, ottenendo risultati che non sono né eccezionali né pessimi.

Si nota infine come, nonostante sia la soluzione di hritik che quela di ritikbompilwar siano entrambe basate su MobileNetV2, ottengono risultati discretamente diversi. Questo permette di osservare come anche la dimensione delle immagini utilizzate per addestrare la rete, giochi un ruolo importante sulla qualità della rete costruita.

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 per avermi sempre sostenuto, e ai miei amici per la loro costante presenza.

Bibliografia

[1] Warren S. McCulloch and Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. In Margaret A. Boden, editor, *The Philosophy of Artificial Intelligence*, Oxford readings in philosophy, pages 22–39. Oxford University Press, 1990.

[2] Alan M. Turing. Computing machinery and intelligence. In Margaret A. Boden, editor, *The Philosophy of Artificial Intelligence*, Oxford readings in philosophy, pages 40–66. Oxford University Press, 1990.

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

[4] Lihi Shiloh-Perl and Raja Giryes. Introduction to deep learning. *CoRR*, abs/2003.03253, 2020.

[5] B. Mehlig. Artificial neural networks. *CoRR*, abs/1901.05639, 2019.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[7] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 437–478. Springer, 2012.

[8] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015.

[10] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vis.*, 111(1):98–136, 2015.

[11] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 3730–3738. IEEE Computer Society, 2015.

[12] Bolei Zhou, Àgata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 487–495, 2014.

[13] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[14] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, page 319, Berlin, Heidelberg, 1999. Springer-Verlag.

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

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

[17] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520. IEEE Computer Society, 2018.

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

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[20] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[21] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[22] Pau Rodríguez, Miguel Ángel Bautista, Jordi Gonzàlez, and Sergio Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image Vis. Comput.*, 75:21–31, 2018.