Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Riconoscimento del colore nelle immagini di tessuti tramite tecniche di Deep Learning

Color recognition in fabrics images through deep learning techniques

Relatore

Prof. Domenico Ursino

Candidato

Simone Di Saverio

Correlatrice

Prof.ssa Alessia Amelio

Anno Accademico 2020-2021

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Introduzione

Nella vita quotidiana siamo fortemente circondati dai colori che giocano un ruolo centrale nella moda e nel settore tessile. Uno dei principali problemi del colore è il suo riconoscimento; immaginiamoci da un sarto per chiedere quale sia il tessuto più simile all'immagine del tessuto fotografato dal nostro smartphone. Rispondere a questo quesito è molto complicato, perché, probabilmente, chi sta vendendo la stoffa può utilizzare nomi differenti per indicare lo stesso colore, oppure ha sfumature diverse della stessa tipologia di tessuto, rendendo la scelta più complessa.

Un'ulteriore problematica è rappresentata dalle condizioni di luce con cui l'immagine viene acquisita, rendendo difficile il riconoscimento. Pensiamo a immagini scattate in ambiente particolarmente buio: il colore fotografato risulterà più scuro. Inoltre, chi sta acquisendo l'immagine potrebbe prendere il colore a partire da una vetrina, oppure in immagini scattate per strada, dove eventuali ombre rendono più difficile il riconoscimento.

L'obiettivo della tesi è quello di poter realizzare un sistema automatico di riconoscimento del colore per immagini di tessuti, in grado di aiutare le persone ad effettuare la scelta. A livello applicativo possiamo immaginare un software in grado di aiutare chi sta scegliendo il tessuto a riconoscere il colore fotografato.

Prima di realizzare l'applicazione software, però, abbiamo la necessità di individuare una tecnica in grado di riconoscere il colore in maniera efficace. In questo elaborato verranno valutate le possibili soluzioni che utilizzano metodi basati sul *Deep Learning (DL)*. Quest'ultimo è stato scelto per la efficacia dimostrata per applicazioni di *image processing*; infatti, il DL permette di definire delle reti neurali *feed-forward*, chiamate *Convolutional Neural Network (CNN)*, ampiamente utilizzate per l'elaborazione delle immagini. Una qualsiasi rete può prendere il nome di CNN se ha uno o più strati convoluzionali. Essi sono fondamentali perché eseguono un'operazione di convoluzione sul volume di input estraendo in maniera automatica le feature. Dunque, le reti CNN sono di tipo *featureless* e, quindi, in automatico, riescono ad estrarre le informazioni dai dati passati in input rendendo la fase di addestramento più semplice, non avendo il bisogno di pre-processing.

Inoltre, la particolarità delle CNN è data dalla loro struttura fortemente ispirata alla percezione visiva animale, offrendo i seguenti vantaggi: *collegamenti locali*, *condivisione dei pesi* e *riduzione dei parametri*.

Per lo sviluppo del modello che permette l'individuazione del colore, nel corso dell'elaborato verrà utilizzato il "Fabrics Dataset", il quale contiene immagini di tessuti fotografati in diverse condizioni di luce, ma è costruito per essere usato in task diversi. Infatti, in [21] esso è stato introdotto per il riconoscimento del materiale sfruttando la microgeometria e la riflettanza; dunque, prima di poterlo applicare per il riconoscimento del colore, si è effettuata una etichettatura manuale dei diversi tessuti che lo compongono, definendo Arancione, Bianco, Blu, Ciano, Giallo, Magenta, Nero, Rosso, Terra, Verde, Verde Smeraldo e Viola come colori di riferimento.

Prima di proseguire con una descrizione dei test, la tesi introduce il concetto di colore e descrive una serie di tecniche applicabili al riconoscimento dello stesso. Il Capitolo 2, infatti, introduce una serie di concetti fondamentali che verranno poi ripresi nel corso dei nostri test. Tra questi verrà illustrata l'estrazione del *colore dominante* attraverso l'utilizzo di algoritmi che fanno uso del *k-means clustering* e della libreria `ColorThief`, che utilizza l'algoritmo *Modified Median Cut Quantizaztion (MCCQ)*.

Il presente elaborato costituisce un vero e proprio primo studio rivolto al riconoscimento del colore nei tessuti. In letteratura, infatti, non sono presenti lavori analoghi per risolvere il problema. I nostri test, dunque, vogliono esplorare quale sia la possibile direzione da intraprendere per la risoluzione del problema. Essi sono stati eseguiti utilizzando il metodo di valutazione che sfrutta la *k-fold cross validation* e si suddividono in tre macro categorie:

- riconoscimento del colore mediante utilizzo di immagini RGB;
- riconoscimento mediante metodo delle differenze;
- miglioramento delle performance utilizzando tecniche di Ensemble Network.

Il riconoscimento del colore mediante l'utilizzo di immagini RGB è stato eseguito con la rete *Color Deep*, introdotta in letteratura da [42] per il riconoscimento del colore dei veicoli. I test sono stati effettuati con e senza colore dominante e sono visibili nel Capitolo 4. Mettendo a confronto i risultati si nota che il modello con immagini contenenti texture ottiene i migliori risultati con un valore di accuratezza superiore al 9% rispetto all'utilizzo del colore dominante. Ulteriore test è quello che fa utilizzo del *Transfer Learning (TL)*; in questo caso si è pre-addestrata la rete con il dataset per il riconoscimento del colore nei veicoli. I modelli pre-addestrati sono, poi, stati utilizzati come base per l'addestramento della *Color Deep* sul nostro dataset. In queto caso le performance non sono risultate migliori rispetto al caso senza TL.

Successivamente viene introdotto il metodo delle differenze. Esso ci permette di spostare il dominio dei colori in quello delle differenze. Queste ultime vengono calcolate rispetto ai colori di riferimento, utilizzati per etichettare il dataset. Tale soluzione ci permette di ottenere delle performance migliori rispetto al semplice utilizzo di immagini RGB. Insieme al trasferimento di dominio, viene presentata una rete più semplice: la *CNN-Differenze*, che, messa a confronto con le reti *ResNet*, *DenseNet* e *InceptionResNet*, ottiene i valori di accuratezza più elevati. Il confronto viene eseguito utilizzando il metodo delle differenze.

La *CNN-Differenze* ottiene risultati molto positivi perché il dataset utilizzato non contiene un numero di immagini elevato e, quindi, essendo una rete sequenziale

con pochi livelli ci consente una maggiore generalizzazione. Inoltre, come dimostrato in [42], le CNN, che hanno l'obiettivo di riconoscere il colore, lo fanno nei primi livelli. La rete, dunque, riesce in maniera più efficace a riconoscere il colore, dal momento che, avendo delle immagini contenenti un singolo colore, non hanno bisogno di individuare eventuali contorni o altre eventuali caratteristiche.

Come ultimo passo per migliorare le performance del modello, si è deciso di applicare delle tecniche di *Ensemble* utilizzando la *CNN-Differenze*. Dai test effettuati nel Capitolo 6, otteniamo un'accuratezza pari all'80% dimostrando che il metodo individuato è molto promettente.

La presente tesi è strutturata come di seguito specificato:

- Il Capitolo 1 introduce il concetto di colore spiegando come esso viene rappresentato in digitale.
- Il Capitolo 2 elenca una serie di soluzioni possibili per il riconoscimento del colore, spiegando il concetto di colore dominante.
- Il Capitolo 3 presenta il dataset utilizzato per i testi effettuati.
- Il Capitolo 4 descrive il primo approccio al problema utilizzando la rete *Color Deep*, spiegando come viene eseguita la fase di valutazione del modello.
- Il Capitolo 5 presenta il metodo delle differenze e la rete *CNN-Differenze*, mostrando il confronto tra le varie reti proposte come alternative.
- Il Capitolo 6 descrive e valuta l'implementazione dell'Ensemble Network.
- Il Capitolo 7 illustra la letteratura correlata.
- Il Capitolo 8 presenta le conclusioni del lavoro svolto e i possibili sviluppi futuri.

1

Il colore: concetti preliminari

In questo primo capitolo verrà descritto cos'è il colore e come esso viene rappresentato in digitale. In particolare sarà introdotto il concetto di spazio del colore.

1.1 Il colore proprietà fisiche

Spesso il *colore* viene visto come una caratteristica scontata propria degli oggetti; sappiamo invece che esso è un fenomeno particolarmente complicato che coinvolge la luce, i fenomeni di assorbimento, di riflessione e rifrazione.

Newton ha dimostrato che i colori sono contenuti nella luce e che il loro manifestarsi è dovuto all'interazione fra luce ed oggetti, infatti facendo passare un raggio di luce bianco all'interno di un prisma di vetro si ottiene uno spettro di colori (colori spettrali puri) che vanno dal rosso scuro al violetto. In Figura 1.1 è possibile vedere una riproduzione dell'esperimento di Newton. Esso dimostra come la luce bianca è tale perché contiene tutte le frequenze dello spettro del visibile.

In fisica la luce può essere vista come una radiazione elettromagnetica in grado di propagarsi alla velocità di 299 792 458 m/s (velocità della luce). Dunque, dal punto di vista fisico-ondulatorio, prendendo un'onda luminosa con una sola componente cromatica, nel vuoto, trascurando ampiezza e polarizzazione, è possibile descrivere l'onda attraverso un unico parametro, ovvero la lunghezza d'onda λ.

Al variare della lunghezza d'onda si possono identificare differenti zone dello spettro elettromagnetico. L'uomo attraverso l'occhio riesce a percepire solo lo spettro del visibile che va, indicativamente, dai 390 nm ai 700 nm. In Figura 1.2 è possibile vedere graficamente la posizione dello spettro del visibile. Dunque, la lunghezza d'onda è la responsabile della tonalità del colore percepito; ad esempio, un'onda elettromagnetica di lunghezza compresa tra i 650 nm e i 700 nm provoca in un soggetto normodotato la visione del colore rosso.

L'uomo vede grazie all'occhio e, attraverso il cervello, riesce a percepire il colore. In maniera schematica, la vista umana funziona in questo modo:

- la luce entra nell'occhio tramite la cornea che modifica la sua dimensione a seconda dell'intensità;

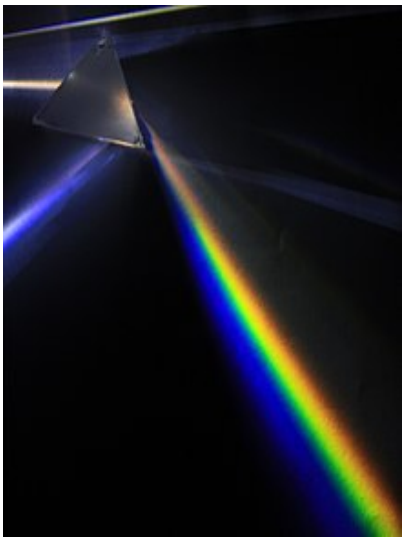


Figura 1.1. Prisma che separa la luce nei colori che compongono lo spettro visibile (fonte: *Wikipedia*)

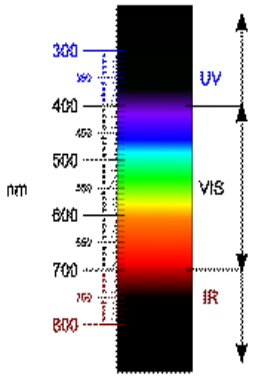


Figura 1.2. Posizione dello spettro del visibile (fonte: *Wikipedia*)

- Attraverso la pupilla essa arriva al cristallino, il quale si occupa della messa a fuoco;
- i *coni* e i *bastoncelli* della retina assorbono la luce e inviano messaggi al cervello attraverso il nervo ottico;
- il cervello trasforma gli impulsi in immagini a colori;

Dunque i recettori del nostro occhio (coni e bastoncelli) ci permettono di riconoscere i colori; anomalie nel funzionamento dei ricettori causano patologie come il daltonismo. Nello specifico i recettori si dividono i compiti nel seguente modo:

- I coni sono responsabili della visione diurna (fotopica), hanno la massima concentrazione — fino a 160.000 per millimetro quadrato — in una piccola zona

della retina, completamente priva di bastoncelli, detta fovea, e presiedono alla percezione del colore e alla nitidezza dei contrasti. Ciascun cono presente nella fovea è collegato ad una cellula nervosa. A questa via privilegiata di comunicazione con il cervello si deve la maggiore capacità di discriminazione dei dettagli, associata con la stimolazione dei coni della fovea [24].

- I bastoncelli, molto più sensibili dei coni alla stimolazione da parte della luce, permettono all'occhio di vedere anche in condizioni di scarsa luminosità. La visione resa possibile dai bastoncelli è una visione non cromatica; essa assume importanza primaria in condizioni di scarsa luminosità ed è detta scotopica [24].

La percezione del colore, dunque, viene lasciata al cervello umano; ciò implica che la differenza qualitativa tra i diversi colori dello spettro del visibile non trova un vero e proprio riscontro a livello fisico, mentre la differenza quantitativa può essere misurata attraverso l'utilizzo della lunghezza d'onda.

1.2 Rappresentazione del colore

Come rappresentare il colore? Per farlo solitamente ci si affida al concetto di modello di colore. Esso è un modello matematico astratto che descrive un modo per rappresentare i colori come combinazioni di numeri, tipicamente come tre o quattro valori detti "componenti colore". Tuttavia, questo modello è una rappresentazione astratta, per questo viene perfezionato con specifiche regole adatte all'utilizzo che se ne andrà a fare, creando uno *spazio dei colori* [9].

Esistono differenti modelli di rappresentazione dei colori. Il primo lo si deve proprio a Newton. Consiste in un cerchio che ha al proprio centro il bianco e lungo la circonferenza i colori scomposti dal prisma, ovvero i colori spettrali puri. Il numero di colori identificati in questo modello sono sette: rosso, arancione, giallo, verde, azzurro indaco e violetto. I colori estremi allo spettro del visibile sono posti in modo da creare una transizione continua [24]. In Figura 1.3 a sinistra viene mostrato il cerchio dei colori di Newton.

La mescolazione di colori spettrali e non genera i colori non spettrali, mentre mescolare i colori estremi dello spettro visibile (rosso e violetto) permette di realizzare una serie di colori chiamati *porpore*. Preso, dunque, il cerchio cromatico di Newton è possibile riformularlo in modo più completo aggiungendo anche i colori non spettrali. A destra della Figura 1.3 è possibile vedere il cerchio di Newton arricchito.

Una caratterista da non sottovalutare è che la percezione visiva è sintetica. Questo vuol dire che se abbiamo una luce rossa o una luce verde che colpiscono lo stesso punto della retina avremo come risultato la percezione del giallo; infatti, il nostro sistema visivo non è in grado di discriminare i due segnali luminosi [24].

Il modello realizzato da Newton non è l'unico. Nel tempo, infatti, molti studiosi hanno cercato di realizzare dei modelli in grado di spiegare una serie di fenomeni che riguardano la visione dei colori da parte dell'occhio umano; la teoria dei processi opposti è proprio uno di questi. Questa teoria, a livello pratico, è stata implementata dando origine al sistema *Natural Color System (NCS)* [24].

Le immagini digitali solitamente utilizzano un metodo di rappresentazione a *sintesi additiva*.

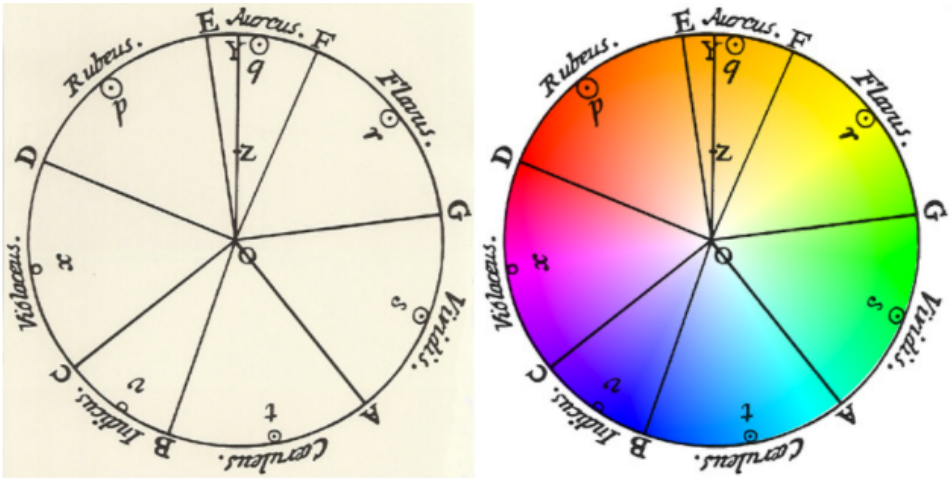


Figura 1.3. Cerchio cromatico di newton

1.2.1 La sintesi additiva

La sintesi additiva si basa sul concetto di mescolanza dei colori; Secondo quale concetto luci di differente lunghezza d'onda se sommate insieme, generano il colore bianco. Questo tipo di fenomeno viene, appunto, definito mescolanza additiva.

Questo modello realizza un sistema in grado di creare tutti i colori attraverso la mescolanza di colori definiti come primari. Questi ultimi sono il rosso, il verde ed il blu, divenuti standard per la riproduzione delle immagini nei dispositivi elettronici.

La scelta di questi colori primari è del tutto arbitraria, mentre utilizzare un numero di colori pari a tre è dovuto al fatto che esso è il numero minimo di elementi in grado di generare una gamma di colori paragonabile allo spettro del visibile. Inoltre, i colori scelti devono essere il più possibile indipendenti l'uno dall'altro, in modo tale da stimolare il più possibile i ricettori dell'occhio umano [24].

Esistono tre diversi tipi di sintesi additiva:

- *Spaziale*: la sintesi additiva spaziale è l'effetto prodotto dalla sovrapposizione di luci su una stessa porzione di spazio [24].
- *Media spaziale*: essa avviene quando luci di colore differente, molto ravvicinate tra loro, sono viste dall'occhio da una distanza tale per la quale non è più possibile individuare le singole componenti, ottenendo la mescolanza dei due colori [24].
- *Media temporale*: otteniamo questa soluzione nel momento in cui luci di colore differente colpiscono in successione; quando la velocità nell'alternarsi è molto elevata non distinguiamo più i singoli ed otteniamo un colore-somma (mescolanza tra i colori) [24].

1.3 Spazio dei colori

Lo *spazio colore (CS)* è la combinazione di un modello di colore e di una appropriata mappatura di questo modello [9].

Esso è fondamentale nel momento in cui si parla di immagini digitali, infatti la maggior parte dei colori che visualizziamo nei nostri laptop, tv o smartphone utilizzano un determinato tipo di CS per mostrare il colore.

I CS maggiormente utilizzati, come visto nella sezione precedente, fanno uso di terne di coordinate per rappresentare un dato colore. Esistono però anche spazi come quello CYMK che fanno uso di un modello di tipo sottrattivo. Quest'ultimo trova il suo impiego nella stampa digitale è, quindi, utilizzato da stampanti, plotter ecc...

Gli spazi colore più famosi sono: RGB, HSV, LAB, ma ne esistono molti altri che possono trovare diverse applicazioni in ambito di *image processing*, dove la scelta del sistema di rappresentazione del colore è fondamentale. Questo perchè spazi di colore differenti rappresentano in modo diverso la stessa informazione.

1.3.1 Spazio colore RGB

Lo spazio RGB è lo spazio più utilizzato per la rappresentazione del colore; esso è basato su un modello di sintesi additiva e, come si evince dal nome, utilizza, per la rappresentazione, una terna di valori che corrispondo ai colori primari Rosso, Verdee Blu.

Questo CS è largamente diffuso per rappresentare il colore nei dispositivi elettronici. Quando visualizziamo un'immagine digitale su uno schermo quello che stiamo guardando sono una serie di pixel che compongono una matrice; tale matrice rappresenta proprio l'immagine. I pixel, a loro volta, sono formati dalla terna R, G e B. Effettuando uno split dell'immagine in tre matrici ognuna delle quali rappresenta una coordinata, otteniamo i canali.

Nel momento in cui si utilizza uno spazio colore differente, le terne fanno riferimento a coordinate differenti, ma il concetto di immagine rappresentata come matrice è identico.

Il range di valori che possono assumere le tre coordinate varia tra 0 e 255; normalmente,per l'image processing, vengono usati anche dei valori normalizzati.

Banalmente potremmo pensare che due terne differenti RGB definiscono due colori differenti, questo, però, non è del tutto vero; la differenza tra due terne di coordinate non è sempre percepibile all'occhio umano. Questo accade perchè lo spazio colore RGB non è uno *spazio di tipo uniforme*.

Attraverso lo spazio colore appena definito possiamo iniziare ad introdurre la metrica di *distanza* per capire se due colori sono tra loro simili e, dunque, hanno la stessa *tonalità*. La soluzione più banale è quella di usare la distanza euclidea:

$$distanza = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2} \tag{1.1}$$

Questa soluzione non tiene conto di come l'occhio umano percepisce il colore. Dunque, nel tempo, sullo spazio RGB si è cercato di definire delle metriche più sofisticate; queste ultime, però, non hanno portato a risultati soddisfacenti [3].

1.3.2 Spazio colore Lab

Il CS *Lab*, anche spesso denominato *CIELAB*, è uno spazio definito dalla *Intrenational Commission on Illumination (CIE)*. Esso descrive tutti i colori visibili dall'occhio umano ed esprime il colore attraverso tre coordinate: L per la luminosità mentre a e b per le dimensioni di colore opponente.

Il range di valori utilizzato per rappresentare i canali è il seguente:

$$0 <= L <= 100, -127 <= a <= 127, -127 <= b <= 127$$

Inizialmente il CIE pensava di aver realizzato uno spazio colore uniforme dunque variazioni uniformi dei componenti nello spazio colore Lab corrispondono a cambiamenti uniformi del colore percepito, cosa che invece non accade in RGB come visto in precedenza. L'uniformità dello spazio ci permette di utilizzare in maniera soddisfacente la distanza euclidea come misura di differenza. Purtroppo, è stato successivamente scoperto che lo spazio non era uniforme e, per risolvere questo problema, la Commissione Internazionale sull'Illuminazione ha realizzato una misura di distanza più sofisticata chiamata ΔE [3]:

- Δ sta a significare che essa misura una differenza;
- E sta per "Empfindung" parola tedesca che significa "sensazione", per ribadire che questa misura tiene conto di come l'occhio umano vede i colori.

Il CIE ha definito diverse formule per calcolare il ΔE; questo, perchè, nel tempo, si è cercato di perfezionare il calcolo nel momento in cui si è scoperto che lo spazio non era percettivamente uniforme.

La Figura 1.4 mostra un'immagine in Lab con i canali separati.

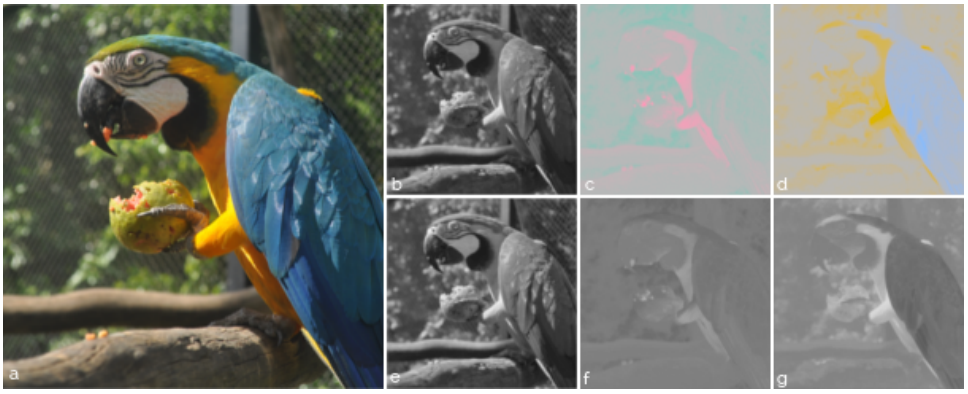


Figura 1.4. Immagine originale (a) e i suoi canali Lab: luminanza (b), coordinata a (c) e coordinata b (d). Sulla seconda riga mostriamo ogni canale in scala di grigi. (Fonte: https://rodrigoberriel.com/2014/11/opencv-color-spaces-splitting-channels/)

1.3.3 Spazio colore HSV

Lo spazio HSV, creato da Alfred H. Munsell, descrive il colore attraverso l'utilizzo di concetti come tonalità, saturazione e luminosità. Più specificatamente:

- *La tonalità* si riferisce al colore effettivo e come l'occhio umano lo percepisce; dunque, possiamo ricondurre in modo semplice questo concetto alla lunghezza d'onda.
- *La luminosità* specifica quanto il colore sia chiaro o scuro, quindi specifica la quantità di bianco e di nero presenti nel colore.
- *La saturazione* misura il grado di purezza e intensità di una tonalità; tinte molto sature hanno un colore acceso mentre con il diminuire della saturazione il colore è debole e tendente al grigio.

Tali concetti vengono rappresentati attraverso l'utilizzo di un cono rovesciato, dove la tonalità è rappresentata in gradi sul cerchio del cono, la luminosità è misurata verticalmente mentre la saturazione viene misurata radialmente a partire dal centro del cono verso l'esterno. La Figura 1.5 mostra in maniera più chiara quanto detto.

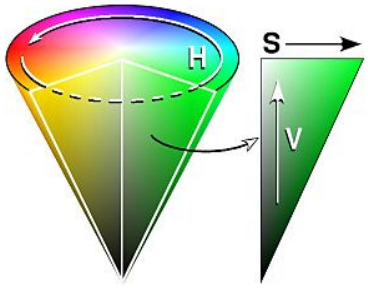


Figura 1.5. Rappresentazione dello spazio colore HSV (fonte: italiawiki.com)

Il range di valori che assumono le tre coordinate sono i seguenti:

$$0 <= H <= 360, 0 <= S <= 1, 0 <= V <= 1$$

Per avere lo spazio HSV quello che si fa è eseguire una conversione dallo spazio RGB. La libreria *OpenCv* esegue la conversione in questo modo:

$$V = max(R, G, B)$$

$$S = \begin{cases} \frac{V - min(R,G,B)}{V} & se\ V\ !=\ 0 \\ 0 & altrimenti \end{cases}$$

$$H = \begin{cases} 60(G-B)/V - min(R,G,B) & se\ V = R \\ 120 + 60(B-R)/V - min(R,G,B) & se\ V = G \\ 240 + 60(R-G)/V - min(R,G,B) & se\ V = B \\ 0 & se\ R = G = B \end{cases}$$

1.3.4 Spazio colore YCbCr

Esso consiste in una famiglia di spazi colore utilizzata nei sistemi a video componenti e di fotografia digitali. Si basa sul modello di tipo additivo, dove:

- Y è la componente di luminanza, che rappresenta l'intensità di luce complessiva nell'immagine;
- Cr indica i valori di differenza dal colore rosso;
- *C*b indica i valori di differenza dal colore blu.

La Figura 1.6 ci mostra in modo più esplicativo l'informazione trasportata tra i diversi canali. Come possiamo notare, se prendiamo solo il canale Y, otteniamo l'immagine in scala di grigi, mentre se prendiamo il canale Cb (terza foto dall'alto della Figura 1.6) e Cr (ultima foto in basso della Figura 1.6) notiamo come il blu del cielo ha un livello elevato in Cb mentre basso in Cr. Cr e Cb vengono definiti, anche, come componenti di crominanza [11].

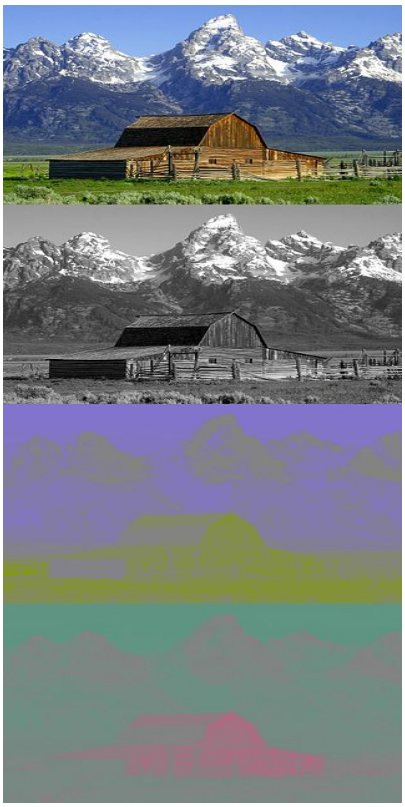


Figura 1.6. Immagine a colori e sue componenti nello spazio colore YCbCr (fonte:Wikipedia)

Per lavorare con questo spazio colore nell'image processing solitamente partiamo sempre dallo spazio RGB. Infatti, come detto in precedenza, esso è lo standard di

rappresentazione. Successivamente eseguiremo una conversione verso YCrCb. La conversione viene effettuata in questo modo:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

$$Cr = (R - Y) \cdot 0.713 + delta$$

$$Cb = (B - Y) \cdot 0.564 + delta$$

$$\text{dove } delta = \begin{cases} 128 & \textit{per immagini 8-bit} \\ 32768 & \textit{per immagini 16-bit} \\ 0.5 & \textit{per immagini a virgola mobile} \end{cases}$$

Le formule appena definite sono implementate esattamente in questo modo nella libreria *OpenCv* che realizza la conversione da RGB a YCrCb, tale libreria verrà presentata in dettaglio successivamente.

2

Tecniche di riconoscimento del colore

Nel capitolo corrente si fornirà una panoramica sulle principali tecniche di riconoscimento per il colore. In particolare, ci concentreremo su tecniche di Machine Learning e Deep Learning.

2.1 Riconoscimento mediante metrica di distanza

Come visto nel Capitolo 1 abbiamo introdotto la misura di distanza per individuare quanto sono differenti due tonalità di colore.

Potremmo, dunque, sfruttare questo tipo di metrica per riconoscere il colore. In questo caso, il problema di riconoscimento si trasforma in un vero e proprio problema di ricerca. Possiamo, infatti, pensare di utilizzare un database (DB) in cui sono presenti tutte le tonalità di colore conosciute con le sue coordinate RGB (oppure sfruttare altro spazio colore). In questo modo potremmo calcolare la distanza euclidea per ogni singola tonalità all'interno del DB dei colori e riconoscere il colore individuando la distanza minima.

La soluzione appena descritta è una delle più semplici da applicare ma non è sempre fattibile da percorrere. Il principale problema è che molto spesso vogliamo riconoscere il colore da immagini che hanno texture, sfumature differenti e sono state acquisite in ambienti poco luminosi o disturbati. In questi casi prima di applicare la soluzione proposta, dobbiamo individuare il *colore dominante* rappresentato dall'immagine (introdurremo il concetto di *dominant color* più avanti).

2.1.1 Esempio di implementazione pratica

Vediamo una semplice implementazione di quanto scritto in modo da spiegare meglio le problematiche relative alla soluzione. Per farlo utilizziamo tre librerie:

- *OpenCV*: è una libreria nell'ambito della visione artificiale in tempo reale. Essa mette a disposizione una serie di funzioni e metodi per lavorare attraverso le immagini.
- *Pandas*: è una libreria utile nell'analisi dei dati che ha la capacità di fornire una serie di strumenti per la loro manipolazione.

- *Numpy*: è una libreria open source per il linguaggio di programmazione Python, che fornisce supporto a grandi matrici e array multidimensionali, insieme a una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati.

Il primo obiettivo è quello di realizzare un dataset di dati contenenti tutte le tonalità di colore con i corrispondenti valori RGB. In aiuto ci viene l'algoritmo Color-Names [4]. Il suo funzionamento è molto semplice esso riceve in input i colori presenti nella pagina `https://en.wikipedia.org/wiki/Lists_of_colors` e va a generare in output un file con formato `.csv`.

Il risultato che otteniamo è dato da 865 colori definiti dal proprio nome completo, il loro valore in esadecimale e i loro valori RGB. Il `.csv` in maniera tabellare si presenta come mostrato in tabella 2.1.

	Nome colore	**Hex**	**R**	**G**	**B**
1	Air Force Blue (Raf)	♯5*d*8*aa*8	93	138	168
2	Air Force Blue (Usaf)	♯00308*f*	0	48	143
3	Air Superiority Blue	♯72*a*0*c*1	114	160	193
...					
864	Vegas Gold	♯*c*5*b*358	197	179	88
865	Violet	♯8*f*00*ff*	143	0	255

Tabella 2.1. Esempio di file `.csv` in cui sono presenti tutti i colori

Ottenuto il file `.csv` l'algoritmo risulta di immediata applicazione. Infatti, basta definire una semplice funzione che scorra l'intera lista di colori calcolando la distanza rispetto al colore da individuare. Come risultato, la funzione ritorna il colore che corrisponde alla distanza minore. Il Listato 2.1 presenta proprio il codice della funzione `getColor`, che esegue quanto descritto. Il codice è scritto in Python.

```
1  import numpy as np
2  import cv2
3  import pandas as pd
4
5  index=["color","color_name","hex","R","G","B"]
6  csv = pd.read_csv('colors.csv', names=index, header=None)
7
8  def getColorName(R,G,B):
9      minimum = 1000
```

```
    for i in range(len(csv)):
        color1 = np.array((R, G, B))
        color2 = np.array((int(csv.loc[i,"R"]), int(csv.loc[i,"G"]), int(csv.loc[i,"B"])))
        d = np.linalg.norm(color1 - color2)
        if(d<minimum):
            minimum = d
            color_name = csv.loc[i,"color_name"]
    return color_name
```

Listato 2.1. Codice della funzione `getColor`

Il problema della soluzione è dato dal fatto che può solo individuare colori a partire da valori RGB ciò vuol dire che abbiamo bisogno di una fase di preprocessing per estrapolare il colore dall'immagine. Una soluzione sarebbe quella di realizzare un color picker, dunque di sfruttare un utente che indichi al nostro algoritmo quale sia la terna RGB.

L'implementazione è molto semplice ed è possibile sfruttare *OpenCv*. Si realizza una funzione di `Callback` in grado di monitorare gli eventi del mouse; se l'evento accade, essa aggiorna le proprie variabili estraendo i valori RGB. Possiamo vedere la sua implementazione nel Listato 2.2.

```
clicked = False
r = g = b = xpos = ypos = 0

def draw_function(event, x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDBLCLK:
        global b,g,r,xpos,ypos, clicked
        clicked = True
        xpos = x
        ypos = y
        b,g,r = img[y,x]
        b = int(b)
        g = int(g)
        r = int(r)
```

Listato 2.2. Codice della funzione `Callback`

A questo punto basta realizzare una finestra in cui mostrare l'immagine dalla quale vogliamo estrarre il colore. Nel momento in cui l'utente seleziona con il mouse, la variabile `clicked` sarà `True`, il codice andrà dunque ad aggiornare il valore del nome del colore attraverso la `GetColor()`. Il Listato 2.3 mostra il codice implementato.

```
cv2.namedWindow('color detection')
cv2.setMouseCallback('color detection',draw_function)

while(1):

    cv2.imshow("color detection",img)
    if (clicked):
        #cv2.rectangle(image, startpoint, endpoint, color, thickness)-1 fills entire rectangle
        cv2.rectangle(img,(20,20), (750,60), (b,g,r), -1)
        color_name = getColorName(r,g,b) + ' R='+ str(r) + ' G='+ str(g) + ' B='+ str(b)

        #cv2.putText(img,text,start,font(0-7),fontScale,color,thickness,lineType )
        cv2.putText(img, color_name,(50,50),2,0.8,(255,255,255),2,cv2.LINE_AA)

        #For very light colours we will display text in black colour
        if(r+g+b>=600):
            cv2.putText(img, color_name,(50,50),2,0.8,(0,0,0),2,cv2.LINE_AA)
        clicked=False

    #when user press esc
    if cv2.waitKey(20) & 0xFF ==27:
        break
cv2.destroyAllWindows()
```

Listato 2.3. Codice implementazione riconoscimento colore.

Il risultato ottenuto ci permette di individuare il colore ma non è la soluzione più adatta per un riconoscimento automatico; inoltre, come ombre e colori acquisiti con condizioni di luce non ottimali, soffre particolarmente di problematiche che verranno trattate nelle sezioni successive.

2.2 Riconoscimento mediante tecniche di Machine Learning

Il *Machine Learning (ML)* è una branca dell'Intelligenza Artificiale che "impara" e si adatta nel tempo. Invece di usare regole statiche codificate in un programma, questo tipo di tecnologia identifica gli schemi di input ed evolve il proprio algoritmo nel tempo. Un algoritmo di Machine Learning presi dei dati riesce senza il controllo umano ad assumere delle decisioni. Questa tecnica cambia ampiamente il modo di lavorare di ricercatori e scienziati; infatti, invece di cercare di trovare una soluzione diretta ad un problema si cerca un approccio che troverà la soluzione partendo dai dati di esempio.

Possiamo considerare il ML come l'intersezione di tre discipline: la statistica, l'informatica e l'Intelligenza Artificiale. L'unione quindi di queste tre discipline ci permette di realizzare algoritmi in grado di fornirci in modo automatico risultati più efficienti e generalizzati. Il ML definisce una serie molto vasta di algoritmi che cercano di apprendere in modo automatico da un set di dati di partenza; questo insieme di tecniche e metodologie prendono il nome di *Data Mining*.

Tra le tecniche di *Data Mining* di particolare interesse è l'algoritmo *k-means*. Esso è il più semplice algoritmo di clustering non supervisionato. L'obiettivo del *k-means* è semplice: raggruppare punti di dati simili e scoprire modelli sottostanti. Per raggiungere questo obiettivo, esso cerca un numero fisso (k) di cluster in un set di dati. Successivamente individua un numero (k) di centroidi e alloca ogni punto dello spazio al cluster più vicino. I centroidi vengono di volta in volta aggiornati e sono sempre il baricentro del gruppo.

Un esempio molto semplice è riportato in Figura 2.1, essa rappresenta dei dati di input che passiamo all'algoritmo. A questo punto il funzionamento viene eseguito in step:

1. Scelta del numero k di cluster.
2. Inizializzazione dei k centroidi in maniera casuale generati all'interno del dominio dei dati che saranno il baricentro. Nel nostro esempio immaginiamo di avere $k = 3$ (seconda immagine a partire da sinistra Figura 2.1).
3. Vengono creati k cluster associando ogni osservazione alla media più vicina.
4. Il baricentro di ciascuno dei k cluster diventa la nuova media, Figura 2.1.
5. I passaggi 2 e 3 vengono ripetuti fino al raggiungimento della convergenza. La convergenza viene raggiunta quando le assegnazioni non cambiano più.

Questo tipologia di algoritmo viene molto utilizzata nel momento in cui abbiamo necessità di raggruppare dei dati. Esso viene spesso, utilizzato in ambito commerciale in cui abbiamo la necessità di segmentare la clientela, oppure può essere utilizzato per l'analisi di outlier per individuare ad esempio frodi bancarie. Nel nostro caso introduciamo l'algoritmo per applicare tecniche di *Color Quantization*.

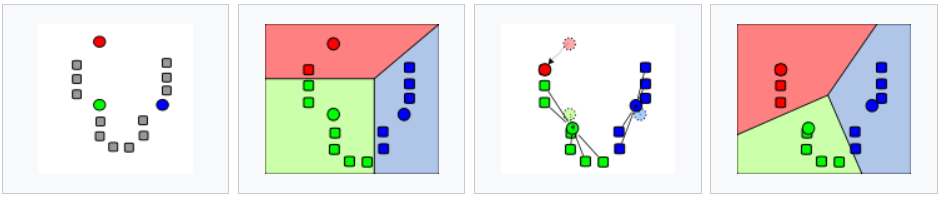


Figura 2.1. A partire da sinistra: la prima figura rappresenta la scelta dei centroidi, la seconda figura rappresenta la fase di assegnazione tra i diversi cluster, la terza figura rappresenta l'aggiornamento del baricentro, la quarta figura rappresenta la convergneza dell'algoritmo (fonte:Wikipedia).

2.2.1 Riconoscimento mediante quantizzazione

Lo spazio colore completo è composto da circa 16 milioni di pixel (codifica a 24 bit), quello che vogliamo fare attraverso la quantizzazione è dividere in un piccolo numero di regioni di un'immagine; per ogni regione viene utilizzato un unico colore rappresentativo di ogni pixel che cade nella regione. In questo modo possiamo passare da una codifica RGB a 24 bit ad una codifica a 16 bit oppure a 8 bit.

Per capire meglio cosa intendiamo con quantizzazione ci può venire in aiuto l'immagine in Figura 2.2. Essa rappresenta un'immagine con 96.615 colori. Applicando l'algoritmo di quantizzazione basato su k-means, con k pari a 64, otteniamo l'immagine in Figura 2.3. Come possiamo notare in questo caso l'immagine è costituita solo da 64 colori.

Figura 2.2. Immagine originale con 96.615 colori (fonte:ScikitLearn).

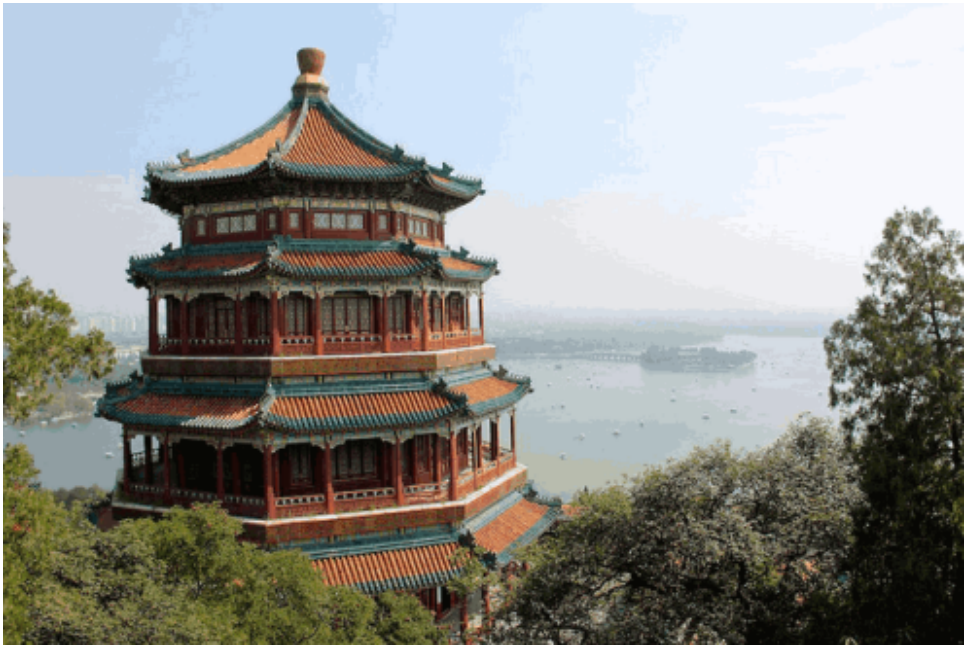


Figura 2.3. Immagine quantizzata con k-NN e k=64 colori (fonte:ScikitLearn).

Esistono anche tecniche euristiche molto efficaci per applicare la quantizzazione una tra queste è la *Modified Median Cut Quantizaztion (MCCQ)* [14]. Essa è implementata attraverso una libreria Python chiamata `ColorThief`.

La quantizzazione è fondamentale nel momento in cui vogliamo riconoscere il colore in immagini nelle quali sono presenti texture o diverse sfumature. Pensiamo di effettuare una ricerca del colore come visto nella sezione 2.1 nel caso di immagini come quelle in Figura 2.4. In questo caso la decisione di quale sia il colore da riconoscere non è banale. Per farlo potremmo pensare di ricondurre il tutto ad un solo colore che chiameremo *dominante*. Il colore dominante, dunque, è il colore più presente all'interno dell'immagine. Per farlo useremo la quantizzazione, possiamo individuare quale sia il cluster con il valore più alto di elementi e il colore dominante sarà il centro del cluster.

Figura 2.4. Esempio di immagine di fotografia di un tessuto.

Come esempio applicativo, utilizziamo la tecnica appena descritta all'immagine in Figura 2.4. Una possibile applicazione attraverso *k-means* è visualizzabile nel Listato 2.4. Come possiamo notare dal codice l'approccio prevede l'applicazione di un algoritmo *k-means* con $cluster = 1$ in modo da uniformare il colore.

```
def quantize(im_path=None, out=None, clusters=1):

    from sklearn.cluster import MiniBatchKMeans
    import cv2
    # load the image and grab its width and height

    if im_path is not None:
        image = cv2.imread(im_path)
    else:
        image = im_path.copy()
    (h, w) = image.shape[:2]

    # convert the image from the RGB color space to the L*a*b*
    # color space -- since we will be clustering using k-means
    # which is based on the euclidean distance, we'll use the
    # L*a*b* color space where the euclidean distance implies
    # perceptual meaning
    image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)

    # reshape the image into a feature vector so that k-means
    # can be applied
    image = image.reshape((image.shape[0] * image.shape[1], 3))

    # apply k-means using the specified number of clusters and
    # then create the quantized image based on the predictions
    clt = MiniBatchKMeans(n_clusters = clusters)
    labels = clt.fit_predict(image)
    quant = clt.cluster_centers_.astype("uint8")[labels]

    # reshape the feature vectors to images
    quant = quant.reshape((h, w, 3))
    image = image.reshape((h, w, 3))

    # convert from L*a*b* to RGB
    quant = cv2.cvtColor(quant, cv2.COLOR_LAB2BGR)
    image = cv2.cvtColor(image, cv2.COLOR_LAB2BGR)

    cv2.imwrite(out, quant)
    return quant
```

Listato 2.4. Codice relativo all'implementazione della quantizzazione ad un cluster per individuare il colore dominante.

Applicando l'algoritmo alla Figura 2.4 otteniamo l'immagine a sinistra della Figura 2.5.

Figura 2.5. Colore dominante dell'immagine del tessuto in Figura 2.4. A sinistra il colore ottenuto attraverso algoritmo *k-Means*; a destra colore ottenuto attraverso liberia `ColorThief`.

Potremmo applicare la stessa soluzione attraverso l'utilizzo della libreria `ColorThief` che fa uso dell'algoritmo MCCQ; il risultato è visibile a destra della Figura 2.5. Pos-

siamo notare come i due algoritmi si equivalgono nella scelta del colore dominante; infatti le tonalità, a primo impatto, possono sembrare identiche. Preso però il codice colore in RGB, le singole soluzioni differiscono, abbiamo infatti una terna pari a $(95, 106, 119)$ per soluzione ottenuta attraverso *ColorThief*, mentre per la soluzione con *k-Means* otteniamo $(100, 113, 126)$. Questo ci conferma come spostamenti nello spazio colore RGB non siano totalmente percettibili all'occhio umano.

Ottenuto il valore del colore dominante, potremmo applicare l'algoritmo di ricerca individuato nel Capitolo 1 e, dunque, andare a riconoscere il colore.

2.2.2 Riconoscimento del colore mediante SVM

Un ulteriore metodo presente in letteratura per il riconoscimento del colore utilizza la *Support Vector Machine (SVM)*.

SVM è un algoritmo usato per problemi di classificazione, nato negli anni '90 nei laboratori Bell. Questi classificatori sono chiamati, anche, classificatori a massimo margine. Essi individuano un iperpiano che divide il set di dati in due.

Per capire come funziona una SVM abbiamo bisogno dei seguenti concetti fondamentali:

- *Support Vector* - I vettori di supporto sono i punti che possiamo definire come i più vicini all'iperpiano (Figura 2.6).

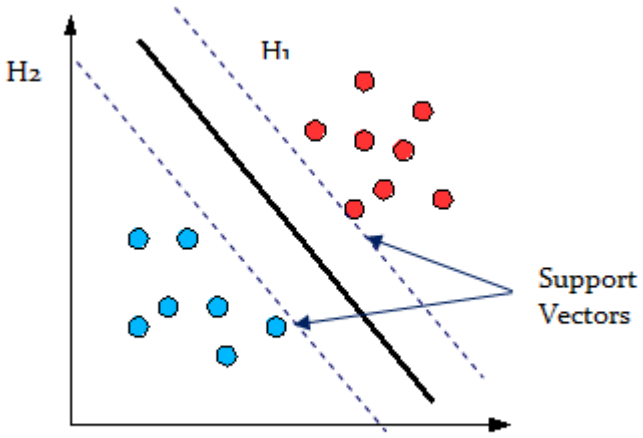


Figura 2.6. Esempio di vettori di supporto (Fonte: Algoritmo Support Vector Machine).

- *Margine* - esso viene definito come la distanza tra i vettori di supporto di due classi differenti più vicini all'iperpiano.

Il funzionamento della *Support Vector Machine* è quello di individuare l'iperpiano che meglio divide i vettori di supporto in classi. Dunque si esegue una ricerca

di un iperpiano linearmente separabile se ne esiste più di uno, sceglie quello che ha margine più alto con i vettori di supporto.

L'iperpiano ottimale può essere visto attraverso la seguente relazione:

$$w^t x + b = 0 \tag{2.1}$$

dove w rappresenta il vettore normale all'iperpiano separatore, x diventa il vettore delle caratteristiche di input e b rappresenta il bias. Dunque, è semplice intuire che i punti che si trovano sopra un determinato iperpiano devono soddisfare la seguente condizione:

$$w^t x_i + b > 0 \tag{2.2}$$

Quelli al disotto, invece, dovranno essere minori di 0. Così facendo, però, stiamo considerando solo l'iperpiano separatore, infatti, come obiettivo ci dobbiamo porre quello di massimizzare la distanza tra i vettori di supporto. Per farlo, dunque, includiamo anche i margini delle classi:

$$w^t x_i + b \geq 1, \ \ se \ y_i = 1 \tag{2.3}$$

$$w^t x_i + b \geq 1, \ \ se \ y_i = 1 \tag{2.4}$$

dove y_i è l'i-esima classe che può assumere soltanto valore $+1$ o -1. Abbiamo, dunque, definito i confini del margine (Figura 2.7). Sapendo che w è il vettore normale all'iperpiano separatore, possiamo calcolare la sua lunghezza attraverso il calcolo della norma $||w||$. A questo punto possiamo definire il margine m come :

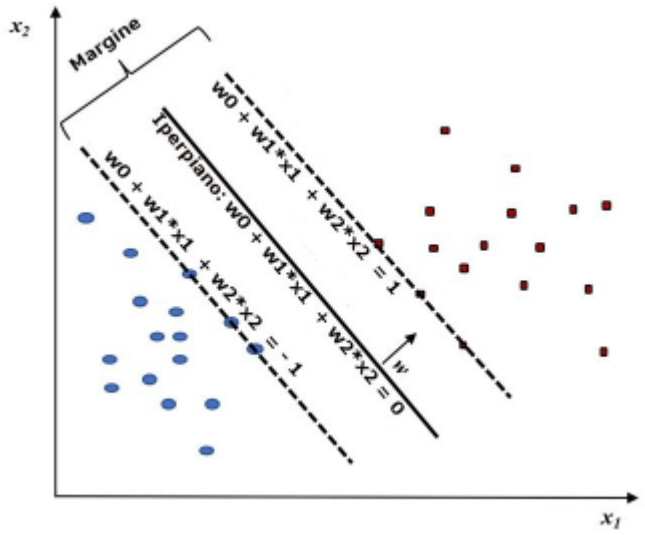


Figura 2.7. Definizione dei confini del margine (Fonte: Algoritmo Support Vector Machine)

$$m = \frac{1+1}{||w||} = \frac{2}{||w||} \tag{2.5}$$

L'equazione deriva dal fatto che le support vector si trovano a distanza 1 dall'iperpiano separatore. Minimizzando il vettore w otteniamo il margine massimo che determina l'iperpiano ottimale.

Il problema di individuazione dell'iperpiano ottimale rientra tra i problemi di ottimizzazione lineare, ed è possibile trovare una soluzione attraverso il metodo dei moltiplicatori di Lagrange e dalle condizioni di Karush-Kuhn-Tucker.

Come si può intuire, la SVM lavora con spazi linearmente separabili, questa però è una grande limitazione che, in realtà, è stata superata attraverso l'applicazione del metodo del kernel, più comunemente chiamato, *Kernel Trick*.

Data la Figura 2.8 possiamo notare che non esiste un limite di decisione lineare; quello che potremmo pensare di fare è aggiungere una terza dimensione stabilendo che venga calcolata nel seguente modo $z = x^2 + y^2$ (equazione del cerchio). Prendendo una fetta del nuovo spazio otteniamo la Figura 2.9 [2].

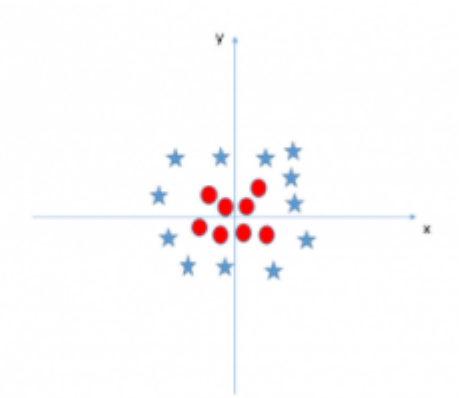


Figura 2.8. Esempio di spazio non linearmete divisibile (Fonte: Algoritmo Support Vector Machine)

Possiamo notare che aggiungendo la terza dimensione otteniamo che, nel nuovo spazio, si possono suddividere i dati da un iperpiano parallelo all'asse x ad una certa quota z.

Quello che abbiamo fatto è sfruttare il metodo del kernel per trasformare il nostro spazio e renderlo separabile. In generale un kernel è una funzione che mappa il nostro spazio di partenza in uno spazio a maggiori dimensioni. Matematicamente, questo può essere definito nel seguente modo:

$$K(x, y) =< f(x), f(y) > \tag{2.6}$$

dove K è la funzione del kernel, x e y sono vettori di input a dimensione n e $< x, y >$ indica il prodotto scalare [2]. Esistono diverse tipologie di kernel: lineare,

Figura 2.9. Spazio reso divisibile aggiungendo una terza dimensione (Fonte: Algoritmo Support Vector Machine)

polinomiale, RBF; esse non verranno trattate nella tesi si rimanda dunque a [56] per tutti i dettagli.

Quello che ci potremmo chiedere è come effettuare la classificazione nel momento in cui abbiamo problemi multiclasse. Le soluzioni proposte sono due:

- *One Versus All (OVA)* - OVA è una soluzione che prevede l'addestramento di C differenti classificatori a due classi. Dunque nel momento in cui devo scegliere la classe di un valore in input x applico:

$$arg(max_c f_c(x)) \tag{2.7}$$

  dove $f_c(x)$ sono i diversi classificatori che vanno da 1 a C, addestrati a due classi dunque c e $C-(c)$.
- *One Versus One (OVO)* - OVO prevede l'addestramento di $\binom{C}{2}$ classificatori a 2 classi e, dunque, tutte le possibili classificazioni. Nel momento in cui valutiamo a quale classe assegnare l'input x scegliamo la classe con le maggiori competizioni vinte. Il seguente metodo è poco utilizzato perché essendo combinatorio se C è molto grande non è efficiente.

In [20] viene spiegato come, attraverso una selezione di feature, è possibile raggiungere ottimi risultati nel riconoscimento del colore dei veicoli mediante SVM. L'estrazione di feature non viene effettuata sull'intera immagine ma utilizzando delle regioni di interesse (ROI), in modo da estrarre solo informazioni utili riuscendo a migliorare il riconoscimento.

2.3 Riconoscimento mediante tecniche di Deep Learning

Il *Deep Learning* è una branca del *Machine Learning* che ha l'obbiettivo di raggiungere la realizzazione di 'un'intelligenza umana'.

Quando facciamo riferimento ad una rete neurale, stiamo facendo riferimento ad una serie di classificatori che lavorano insieme basati sulla regressione lineare seguita da funzioni di attivazione. La differenza principale con i classici classificatori è che, nel caso di *Deep Learning (DL)* abbiamo molti nodi neurali. L'insieme dei nodi vengono definiti *Neural Network*, mentre il singolo nodo neurale viene definito *neurone* [25].

Esso prende il nome di rete profonda (deep) perché la rete è costruita da diversi strati. Infatti, in questo caso, diversamente da quanto accade nelle *Neural Network* la rete deep ha diversi livelli chiamati *hidden layer*, formati da diverse centinaia, se non migliaia, di neuroni, tra input e output (Figura 2.11).

Lo studio del DL ha portato a raggiungere svariati obiettivi, negli anni la sua evoluzione ha fatto si che questa tecnologia potesse essere applicata in diversi ambiti raggiungendo diversi successi. Uno degli ambiti maggiormente utilizzato è la classificazione di immagini dove si sono raggiunti i migliori risultati. Nella classificazione delle immagini il continuo miglioramento delle tecniche di deep learning ha portato il tasso di errore al $3,5\%$, che è inferiore a quello della gente comune [25].

La differenza principale tra tecniche che fanno utilizzo di deep learning e tecniche di analisi dati classica è data dal concetto di feature. Ad esempio, in un problema di classificazione con tecniche classiche, abbiamo la necessità di preprocessare i dati in modo da individuare degli attributi che ci permettano di effettuare la classificazione. Nel caso di tecniche che fanno utilizzo di reti *reti neurali artificiali* non dobbiamo porci questo tipo di problematica; infatti, sarà la rete stessa ad individuare le feature.

Quello che possiamo pensare di fare è sfruttare la deep neural network per la classificazione delle immagini volte a trasformare il problema di ricerca mendiate distanza del colore in un problema di classificazione. Per fare questo utilizziamo delle *Convolutional Neural Network (CNN).*

2.3.1 Reti neurali

Prima di parlare delle *Convolutional Neural Network* bisogna introdurre la definizione di *Rete Neurale Artificiale (ANN)* classica. Come scritto nella parte iniziale della Sezione 2.3 la rete neurale è costituita da singoli neuroni artificiali, essi sono basati su un semplice modello matematico. Tale modello ha tre semplici insieme di regole: moltiplicazione, somma e attivazione.

L'elemento base che poi costituisce le reti neurali è il *neurone artificiale.* Il suo design e le sue funzionalità derivano dall'osservazione e dallo studio dei neuroni biologici che è alla base del funzionamento del nostro cervello. Come possiamo vedere a sinistra della Figura 2.10 nel neurone biologico le informazioni entrano attraverso il dendrite, che passano poi alla soma che si occupa di processare l'informazione che verranno poi passate all'assone.

In caso di neurone artificiale, invece, ogni input viene pesato questo vuol dire che viene moltiplicato per un peso che è diverso per ogni singolo neurone. Nella sezione centrale invece il neurone esegue la funzione di somma e va a sommare tutti gli input pesati e il bias. All'uscita la somma di input e bias passa attraverso una *funzione di attivazione (funzione di trasferimento).* Successivamente il neurone artificiale passa

l'informazione in output. A destra della Figura 2.10 possiamo vedere in maniera schematica quello che abbiamo appena definito [52].

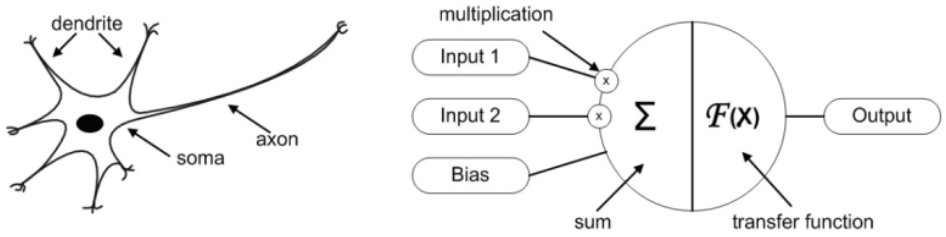


Figura 2.10. Neurone biologico a sinistra e neurone artificiale a destra (Fonte: Artificial Neural Networks - Methodological Advances and BiomedicalApplications).

Il vantaggio del neurone artificiale è che possiamo definire il suo modello matematico, che permette di capire in maniera più semplice il suo funzionamento:

$$y(k) = F(\sum_{i=0}^{m} w_i(k) \cdot x_i(k) + b) \tag{2.8}$$

Dove:

- $x_i(k)$ è l'input a tempo discreto;
- $w_i(k)$ è il valore del peso i a tempo discreto;
- b è il bias;
- F è la funzione di trasferimento;
- $y_i(k)$ è il valore di output a tempo discreto.

Come possiamo vedere dall'Equazione 2.8 il neurone fa utilizzo di una *funzione di trasferimento*. Essa definisce le sue proprietà e viene scelta in base al problema che dobbiamo risolvere, nella maggior parte dei casi la scelta ricade tra le seguenti categorie di funzioni: *Step Function*, *Linear Function*, *Non-Linear function* [52]. Le principali funzioni di attivazione verranno trattate successivamente.

La struttura di un singolo neurone artificiale è relativamente semplice; il potenziale di questi neuroni viene fuori nel momento in cui li colleghiamo (Figura 2.11) tra loro formando delle architetture anche molto complesse.

Il modo in cui i diversi neuroni artificiali vengono connessi prende il nome di *topologia* e, considerando il fatto che le connessioni possono essere effettuate in modi diversi; si traduce nella possibilità di avere numerose topologie.

Le topologie si possono dividere in due classi fondamentali:

- *Topologia feed-forward (FNN)* , in cui le informazioni fluiscono dagli input agli output senza che ci siano presenze di cicli nella rete. Possiamo vedere un esempio di rete feedforward a sinistra della Figura 2.12 [52].
- *Topologia recurrent (RNN)*, esse vengono comunemente chiamate reti ricorrenti e sono descritte attraverso un grafo semiciclico; dunque le informazioni non fluiscono soltanto dall'input all'output ma anche in senso opposto. A destra della Figura 2.12 possiamo vedere un esempio di rete ricorrente [52].

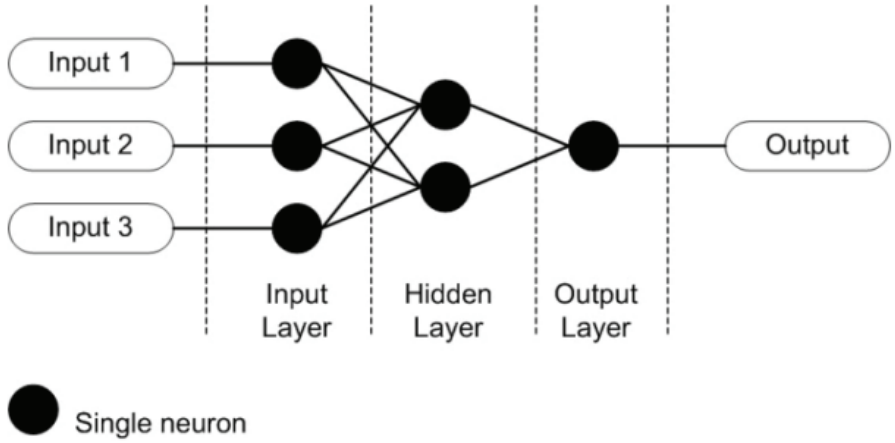


Figura 2.11. Esempio di semplice rete neurale artificiale (Fonte: Artificial Neural Networks - Methodological Advances and BiomedicalApplications).

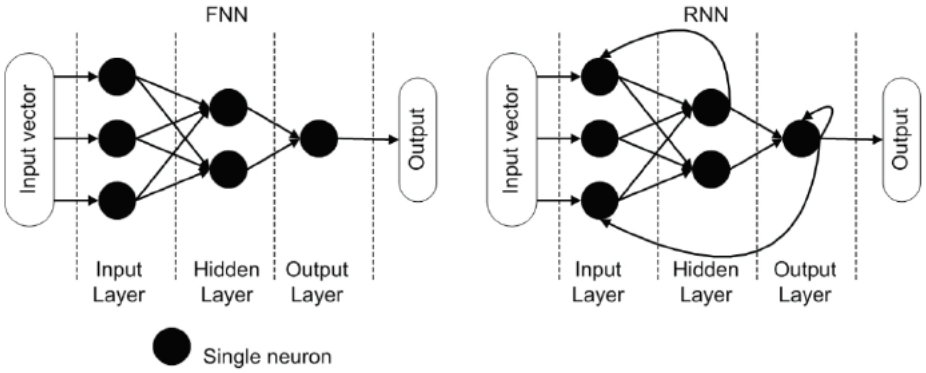


Figura 2.12. Esempio di reti neurali artificiali in topologia Feed-forward (FNN) e recurrent (RNN) (Fonte: Artificial Neural Networks - Methodological Advances and BiomedicalApplications).

Apprendimento delle reti neurali

Introdotto il concetto di rete neurale, vogliamo vedere come essa possa riuscire ad apprendere. Ogni rete neurale utilizza un algoritmo di calcolo del gradiente solitamente definito come algoritmo di *back-propagation (algoritmo di propagazione dell'errore)*. Considerando il fatto che nel proseguo della tesi faremo riferimento a reti di topologia FNN, spiegheremo brevemente, come funziona l'algoritmo di propagazione dell'errore solo per questa topologia di reti, tralasciando le reti ricorrenti.

L'algoritmo si basa sul calcolo dell'errore e aggiorna i pesi partendo dall'ultimo strato della rete fino a raggiungere lo strato di input. L'aggiornamento dei pesi viene fatto con l'obbiettivo di minimizzare l'errore della rete neurale.

L'algoritmo, dunque, funziona in questo modo:

1. Inizializza tutti i pesi con valori casuali.
2. Calcola il valore dell'errore, confrontando input e output della rete.
3. Per minimizzare l'errore calcola il gradiente della funzione di errore rispetto a ciascun peso. Il gradiente serve per trovare dunque il minimo della funzione di errore.
4. Ogni peso viene aggiornato in modo iterativo nella direzione opposta al gradiente. L'aggiornamento viene ripetuto fino a quando l'errore non converge.

La formula generale relativa all'aggiornamento è la seguente:

$$w \leftarrow w - n\frac{\partial E}{\partial w} \tag{2.9}$$

Come possiamo vedere il valore, del peso all'iterazione successiva è dato dal valore del peso all'iterazione precedente meno un valore proporzionale al gradiente. Il segno negativo è dato dal fatto che ci vogliamo spostare nella direzione opposta al gradiente. Il parametro n rappresenta il *learning rate* (tasso di apprendimento della rete); esso decide il "passo" di aggiornamento dei pesi [8].

Possiamo apportare alcune modifiche all'algoritmo di discesa del gradiente oltre alla scelta del parametro di *learning rate*; infatti, in fase di addestramento si può specificare *l'ottimizzatore*. Essi individuano dei metodi alternativi all'algoritmo di discesa del gradiente, descritto in precedenza, per addestrare la rete e modificare i loro parametri.

Esistono diversi tipi di ottimizzatori:

- *Stochastic Gradient Descent (SGD)*; esso viene utilizzato per grandi dataset che sono impossibili da caricare in memoria. SGD permette di lavorare in batch, in modo tale da operare con piccole quantità di dati per volta [45].
- *Momentum*; esso è un metodo che aiuta ad accelerare la discesa del gradiente verso una direzione corretta, smorzando eventuali effetti indotti da oscillazioni. Spesso è utilizzato insieme ad SGD, infatti, quest'ultimo soffre di alcune difficoltà in aree dove la superficie presenta una curvatura accentuata, rallentando la convergenza verso un minimo ottimale [45]. In Figura 2.13 possiamo vedere un'esempio di SGD con e senza *momentum*.

(a) SGD without momentum (b) SGD with momentum

Figura 2.13. Discesa del gradiente senza momentum (a) e con momentum (b) (Fonte: An overview of gradient descent optimization algorithms [45])

- *Adagrad*; esso è un algoritmo di ottimizzazione basato sul gradiente che adatta la velocità di apprendimento ai parametri, eseguendo aggiornamenti maggiori per parametri poco frequenti e aggiornamenti minori per parametri frequenti. Per questo motivo è adatto a gestire dati sparsi [45].
- *Adadelta*; è un estensione di *Adagrad* che cerca di ridurre il suo tasso di apprendimento aggresivo e monotono decrescente. Invece di accumulare tutti i gradienti passati, Adadelta limita la finestra dei gradienti passati ad una dimensione fissa w [45].
- *Adam*; è un estensione di due tipologie di ottimizzatori differenti: l'Adagrad e RMSProp. Invece di adattare i tassi di apprendimento del parametro in base al primo momento medio come in RMSProp, Adam utilizza anche la media dei secondi momenti dei gradienti [45].

La sinistra della Figura 2.14 ci mostra un confronto dei diversi algoritmi di ottimizzazione su una superficie di perdita (la funzione di Beale[1]). Tutti gli ottimizzatori sono inizializzati nello stesso punto, ma, come possiamo vedere prendono strade diverse per raggiungere il minimo. Possiamo notare che gli algoritmi Adagard e Adadelta riescono a trovare subito la direzione che li porta alla convergenza, mentre Momentum è stato portato inizialmente fuori pista per poi riprendere verso la convergenza. La Figura 2.14 presenta, anche, algoritmi come RMSprop e NAG [45] che non sono stati trattati. A destra della Figura 2.14 possiamo vedere il comportamento degli algoritmi in corrispondenza di un punto di sella [45].

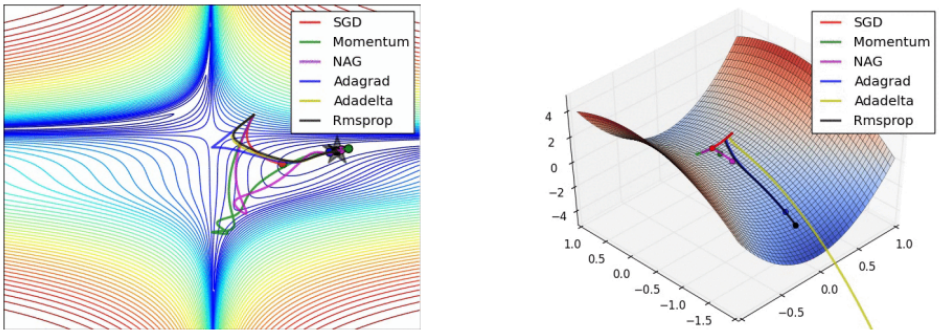


Figura 2.14. Performance degli algoritmi di ottimizzazione su una superficie di perdita (a sinistra) e su un punto di sella (a destra) (Fonte: An overview of gradient descent optimization algorithms [45])

Le funzioni di attivazione

Come detto in precedenza, ogni nodo ha una *funzione di attivazione*. Esse possono essere di diverse tipo. La ricerca ha da sempre come fondamentale obiettivo quello

[1] La funzione di Beale è una funzione di test utilizzata per testare il funzionamento e l'efficienza degli algoritmi di ottimizzazione. Per maggiori approfondimenti sulle funzioni di test si rimanda a [5].

di valutare ed individuare nuovi tipi di funzioni, in modo che si possano adattare alle diverse applicazioni. Di seguito elencheremo le principali funzioni di attivazione:

- *Sigmoid Function* - La funzione sigmoide è una funzione di attivazione non lineare, principalmente usata nelle reti neurali feedforward. È una funzione reale differenziabile limitata, definita per valori di input reali, con derivate positive ovunque e un certo grado di smoothness [40]. Essa è descritta dalla seguente espressione:

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{2.10}$$

  Questa funzione solitamente viene utilizzata nei layer di output delle reti neurali ed è applicata con successo in problemi di classificazione binaria. Essa è principalmente utilizzata in reti poco profonde.
- *Hyperbolic Tangent Function (Tanh)* - La funzione tangente iperbolica è una funzione centrata in zero con codominio $(-1, 1)$; essa è definita dalla seguente equazione:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.11}$$

  Questa funzione risolve notevoli problemi introdotti dalla funzione sigmoide; uno tra questi è il problema degli output non centrati in zero dovuti all'utilizzo della sigmoide. Inoltre, ha migliori prestazioni con reti multistrato [40].
- *Softmax Function)* - La funzione Softmax è un altro tipo di funzione di attivazione utilizzata per architetture deep neural network. Essa viene utilizzata per calcolare la distribuzione di probabilità da un vettore di numeri reali. La funzione Softmax produce un output che è un intervallo di valori compreso tra 0 e 1, con la somma delle probabilità pari a 1 [40]. Essa viene calcolata utilizzando la seguente relazione:

$$f(x_i) = \frac{e_i^{x}}{\sum_j e_j^{x}} \tag{2.12}$$

  Softmax viene utilizzata nei modelli multiclasse, dove restituisce le probabilità di ciascuna classe, la classe di destinazione ha la probabilità più alta. La funzione appare per lo più in quasi tutti i livelli di output delle architetture di deep learning [40].

2.3.2 Convolutional Neural Network

La CNN è stata per la prima volta proposta nel 1980. Tuttavia, solo con lo sviluppo delle *Graphics Processing Unit (GPU)* si è riusciti a risolvere problemi molto complessi come quello proposto da ImageNet[2] [23], risolto nel 2012 attraverso l'utilizzo

[2] *ImageNet* è un'ampia base di dati di immagini, realizzata per l'utilizzo in ambito di visione artificiale, nel campo del riconoscimento di oggetti. Il dataset contiene più di 14 milioni di immagini che sono state annotate manualmente con l'indicazione degli oggetti in esse rappresentati e della bounding box che li delimita. [6].

della rete AlexNet [37]. Queste reti sono utilizzate prevalentemente per applicazioni di *image processing*, come nell'automative, per applicazioni di guida autonoma, in campo medico per supporto alla diagnosi di patologie, nel miglioramento di video e immagini e in molti altri settori. Il nome *Convolutional Neural Network* deriva dal fatto che le reti utilizzano un'operazione chiamata convoluzione per estrarre in maniera del tutto automatica feature dai dati. Dunque, data una classica *rete neurale artificiale*, se inseriamo uno strato che esegue la convoluzione essa prende il nome di rete convoluzionale. Per realizzare modelli di *Convolutional Neural Network* abbiamo la necessità di utilizzare 3 componenti che si alternano definendo l'architettura: *Convolutional layer*, *Pooling layers*, *Fully connected layer*.

Se facciamo riferimento alle classiche *Reti Neurali Artificiali* sappiamo che ogni neurone è collegato sia allo strato precedente che a quello successivo, e quindi facciamo riferimento ad una struttura tipicamente definita *fully connected*. Questa struttura non è in realtà particolarmente adatta nel momento in cui facciamo utilizzo delle immagini e, quindi, di dati in input di tipo matriciale. Per questo abbiamo bisogno di una struttura più ispirata alla percezione visiva, in modo da ottenere una rete che possiede diversi vantaggi:

- *Collegamenti locali*: ogni neurone non è più connesso a tutti i neuroni dello strato precedente, ma solo a un piccolo numero di neuroni il che è efficace per ridurre i parametri e accelerare la convergenza [38].
- *Condivisione dei pesi*: un gruppo di connessioni può condividere gli stessi pesi, il che riduce ulteriormente i parametri [38].
- *Riduzione dimensionale*: uno strato di pooling (il quale vedremo successivamente cosa significa) sfrutta il principio della correlazione locale dell'immagine per effettuare un sottocampionamento, in modo da ridurre la quantità di dati pur mantenendo l'utilità dell'informazione [38].

La convoluzione

Prima di introdurre e spiegare come funzionano i principali layer che compongono una CNN, abbiamo la necessità di capire come funziona l'operazione di convoluzione.

Essa viene largamente usata in applicazioni di computer vision, ma anche di riconoscimento vocale, in cui la struttura classica delle reti neurali potrebbe rovinare la topologia dei dati. La *convoluzione* viene, infatti, utilizzata per far in modo che la rete possa operare con array multidimensionali in cui l'ordine degli assi è ben definito. L'esempio, come sempre, può essere un'immagine in cui larghezza ed altezza seguono un determinato ordine di pixel, e eventuali cambiamenti portano ad una variazione dell'immagine. Inoltre la convoluzione è fondamentale per lavorare con i canali che rappresentano diverse viste dello stesso dato (come sempre si può pensare ai canali RGB di un'immagine) [26].

Dunque, per applicazioni in cui è utile preservare l'informazione topologica dei dati, entra in gioco la convoluzione discreta, che riesce a conservare la nozione di ordinamento.

La Figura 2.15 ci mostra come avviene la convoluzione. La griglia azzurra viene chiamata mappa delle caratteristiche di input. Per semplificazione, in questo caso abbiamo una sola mappa con un solo canale, ma, solitamente, abbiamo diversi canali. L'area ombreggiata nella Figura 2.15 corrisponde al kernel che scorre lungo

la mappa delle caratteristiche. Possiamo, inoltre, notare nella foto che sono stati aggiunti dei valori 0 nella cornice della matrice; questa tecnica viene chiamata *padding* ed è utilizzata perché i pixel sul bordo altrimenti verrebbero "eliminati" perchè non si trovano mai al centro del kernel; per evitare questo viene effettuato uno zero padding [26].

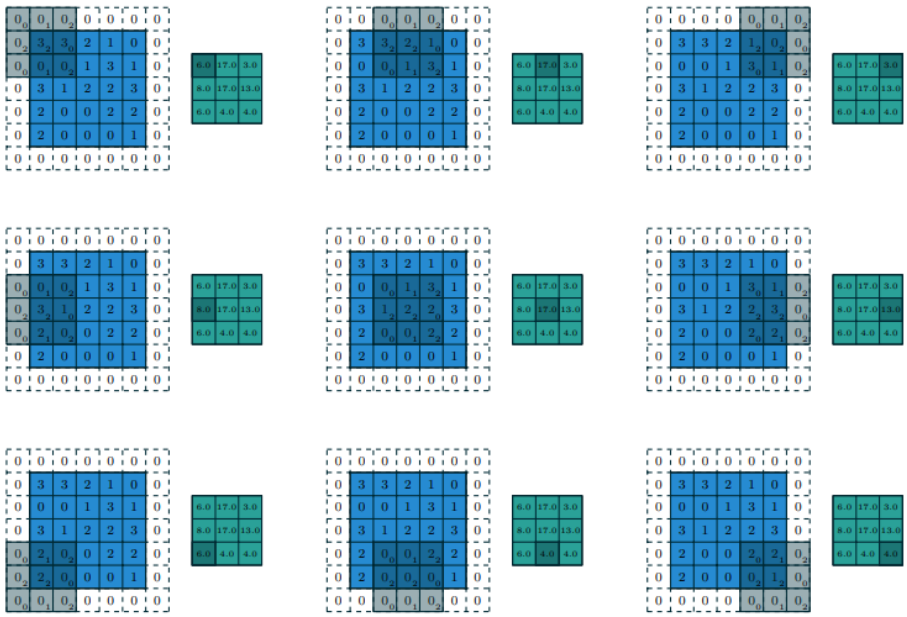


Figura 2.15. Esempio di operazione di convoluzione (Fonte: A guide to convolution arithmetic for deep learning [26]).

Oltre al padding, nel momento in cui effettuiamo una convoluzione, vogliamo in output una dimensione inferiore rispetto all'input; per farlo, oltre ad inserire un possibile *pooling layer*, potremmo definire uno *stride (passo)*, che esegue un vero e proprio sottocampionaemnto. L'idea è quella di saltare alcune delle posizioni nella mappa delle caratteristiche durante la convoluzione. Ad esempio, in Figura 2.16, è possibile vedere come avviene la convoluzione con stride pari a 2, che equivale a spostare il kernel di uno ma mantenendo solo gli elementi di output dispari.

Per calcolare la larghezza e altezza del volume di output o di una convoluzione possiamo utilizzare la seguente relazione:

$$o = (\frac{i + 2p - k}{s}) + 1 \tag{2.13}$$

dove i è la dimensione dell'input, k rappresenta la dimensione del kernel, s rappresenta lo stride applicato allo spostamento del filtro e p è la quantità di padding.

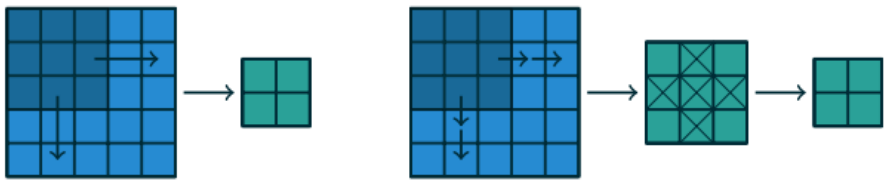


Figura 2.16. Esempio convoluzione con stride pari a 2 (Fonte: A guide to convolution arithmetic for deep learning [26])

Formalmente possiamo definire la transizione da uno strato di input ad un primo strato nascosto, nel momento in cui abbiamo una convoluzione, in questo modo:

$$y^{(2)}_{(i,j)} = F(w^{(1)}_0 + \sum_{a=1}^{G}\sum_{b=1}^{G} w^{(1)}_{a,b} x_{(i+a-1,j+b-1)}) \tag{2.14}$$

Dove l'apice indica il livello del layer, il pedice (i, j) identifica la posizione dell'elemento nella matrice e G rappresenta l'estensione spaziale del filtro.

Architettura CNN e colore

Come detto nell'introduzione alla sezione precedente, la rete neurale convoluzionale è composta da diversi strati; mostreremo, ora, brevemente come essi funzionano.

- *Convolutional Layer* - In precedenza abbiamo introdotto l'operazione di convoluzione. Nel momento in cui si realizza un livello convoluzionale in una rete neurale non stiamo individuando un singolo filtro ma definiamo N filtri tutti con stessa estensione spaziale. Dunque, questi filtri vengono fatti convolvere sul volume di input e produrranno N mappe di attivazione bidimensionali. Il convolutional layer estrae in modo automatico dai dati di input una serie di feature che poi saranno utilizzati dagli strati successivi.
- *Pooling Layer* - L'idea principale del pooling layer è di effettuare un downsampling al fine di ridurre la complessità per i livelli successivi. Uno dei metodi più comuni di pooling è il *max-pooling*. Esso opera sulle feature map e realizza delle sottoregioni restituendo il valore massimo all'interno della sottoregione [13]. Solitamente il max-pooling viene utilizzato con una dimensione 2×2 un esempio è visibile in Figura 2.17 .
- *Fully COnnected Layer* - Questo strato è molto simile a quello di una ANN. Pertanto, ogni nodo di un livello connesso è direttamente connesso ad ogni nodo sia nel livello precedente sia in quello successivo. La Figura 2.18 mostra un esempio. Esso è il livello con più parametri in una CNN e questo implica che rappresenta il livello al quale è richiesto il maggior tempo per essere addestrato.

Possiamo vedere un esempio di rete CNN AlexNet in Figura 2.19. Come possiamo notare, essa è caratterizzata da una prima serie di strati convoluzionali. Questo perchè, anche se non ci sono determinate indicazioni riguardanti le modalità con cui bisogna costruire la CNN e come deve essere definita la squenza di layer, è bene

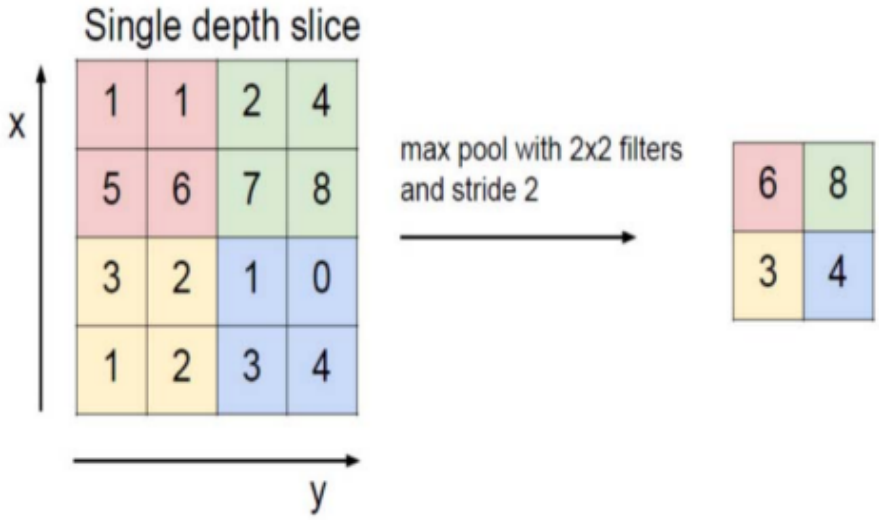


Figura 2.17. Esempio di max-pooling 2×2 e stride pari a 2 che porta ad un down-sampling di ciascun blocco 2×2 ad essere mappato in un solo blocco (Fonte: Understanding of a Convolutional Neural Network [13]).

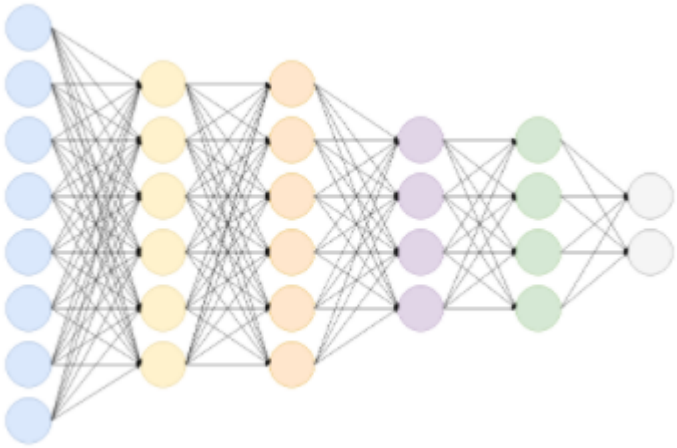


Figura 2.18. Esempio di fully-connected layer (Fonte: towardsdatascience.com).

definire diversi strati convoluzionali in modo tale da estrarre quante più feature possibili provenienti dai dati di input.

Dall'architettura della CNN possiamo intuire che le reti siano in grado a partire dai dati di apprendere come riconoscere determinati oggetti o colori. Essa, quindi, può anche essere applicata per il riconoscimento dei colori; quello che dobbiamo fare, in questo caso, è considerare un dataset etichettato con i diversi colori e addestrare una rete in modo da ottenere un modello che riesca a classificare il colore.

Una soluzione di questo tipo viene adottata nell'articolo [42], dove viene uti-

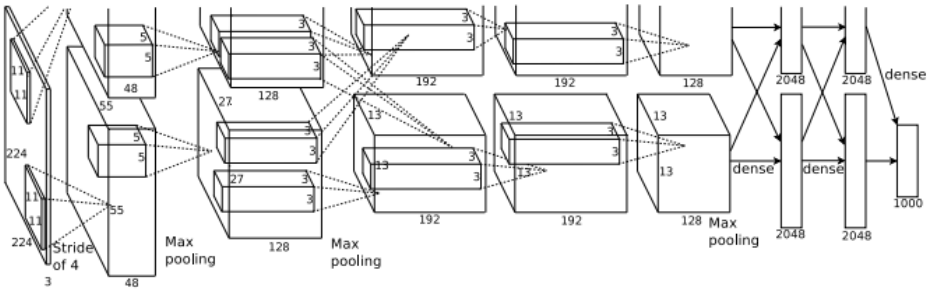
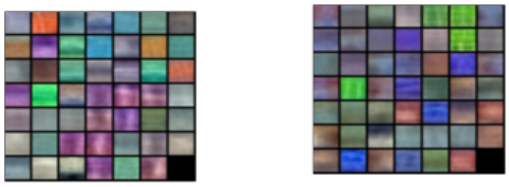


Figura 2.19. Architettura della rete AlexNet (Fonte: ImageNet Classification with Deep Convolutional Neural Networks [37]).

lizzata la rete convoluzionale per riconoscere il colore dei veicoli. In questo caso ci soffermeremo sulla capacità della rete di riconoscere il colore, mentre l'architettura utilizzata e la rete verranno trattate nei capitoli successivi.

Per vedere come la rete cattura informazioni sul colore bisogna visualizzare quali sono le feature estratte dai layer convoluzionali. [42] fa notare come sia il primo livello convoluzionale ad estrarre le principali caratteristiche sul colore dell'immagine di input. La Figura 2.20 mostra tutti i kernel di primo livello convoluzionale. Come si può notare tutte le variazioni di colore del veicolo nel set di dati sono presenti nel kernel.



(a) Kernel from network 1 (b) Kernel from network 2

Figura 2.20. Kernel in uscita dal primo livello convoluzionale sono due perchè la rete è costitutita da due sottoreti separate (Fonte: Vehicle Color Recognition using Convolutional Neural Network [42])

La CNN dunque può essere applicata in maniera efficace per il riconoscimento del colore.

3

Il Dataset

In questo capitolo introdurremo il dataset utilizzato per il riconoscimento del colore dei tessuti. In particolare, mostreremo come esso è stato acquisito e quali sono le principali problematiche ad esso relative. Infine, definiremo qual è l'obiettivo da raggiungere.

3.1 Introduzione al "The Fabrics Dataset"

Nei capitoli precedenti abbiamo introdotto cosa è il colore e le tecniche utilizzate per riconoscerlo. A partire da questo Capitolo descriveremo come riconoscere il colore partendo da immagini di tessuti. Questo problema di visione artificiale non è particolarmente studiato in letteratura e, dunque, possiamo vedere l'intero lavoro come un primo approccio al problema.

Non avendo un dataset dal quale partire si è cercato di trovare in letteratura un set di dati in grado di adattarsi all'obiettivo di riconoscimento del colore. Il focus si è concentrato sull'articolo "Fine-grained material classification using micro-geometry and reflectance" [21]. Esso introduce un dataset, chiamato *Fabrics Dataset*, contenente immagini di tessuti per il riconoscimento del materiale partendo dalla sua microgeometria e riflettanza. La Figura 3.1 mostra una serie di immagini presenti nel Fabrics Dataset.

Come possiamo vedere dagli esempi, le immagini sono a colori e contengono diverse tipologie di tessuto. Per acquisire le immagini è stato utilizzato un *photometric stereo sensor* in modo tale da raccogliere 2000 campioni di capi e tessuti. Le immagini sono state acquisite da tessuti di capi realmente in commercio e, per ogni capo, il dataset contiene informazioni riguardanti la composizione del tessuto [21].

Il photometric stereo sensor (Figura 3.2) è un metodo molto utilizzato nella computer vision che utilizza tre o più immagini di una superficie dallo stesso punto di vista, ma in condizioni di illuminazione differente, per stimare il vettore normale e l'albedo[1] in ciascun punto [21]. Questo tipo di sensore trova applicazione in vari ambiti come la medicina, la robotica e l'archeologia.

[1] L'albedo (dal latino albēdo, "bianchezza", da albus, "bianco") di una superficie e la frazione di luce o, più in generale, di radiazione solare incidente che è riflessa in tutte

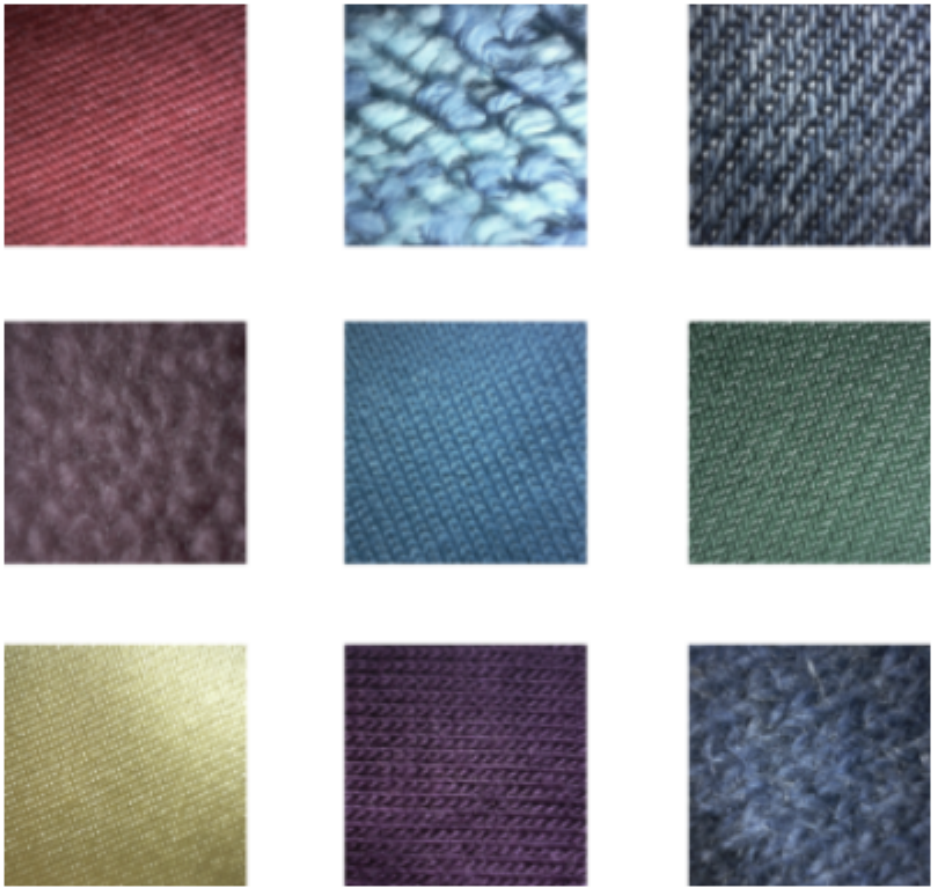


Figura 3.1. Esempio di immagini contenute nel Fabric Dataset (Fonte: The Fabrics Datasets [21])

Figura 3.2. Photometric stereo sensor su un tavolo (Fonte: The Fabrics Datasets [21])

Questo implica che, per i 2000 capi, non abbiamo una singola immagine acquisita ma ne abbiamo 3 o 4. Dunque, l'utilizzo del Fabrics Dataset per il riconoscimento del colore nei tessuti ci permette di avere un numero più ampio di immagini da considerare; infatti, esso è costituto da un numero totale di immagini pari a 7757.

La procedura eseguita in [21] per l'acquisizione delle immagini è la seguente:

- il photometric stereo sensor è posto su una regione piana del capo;
- inizialmente il sensore accende una sorgente luminosa per consentire alla fotocamera di mettere a fuoco la superficie del capo;
- successivamente vengono acquisite 4 immagini, una per ogni sorgente luminosa; questo perché il photmetric stereo sensor è costituito da 4 array di led;
- si normalizzano le immagini catturate con una superficie di albedo noto [21].

In Figura 3.3 viene mostrato come le immagini sono state acquisite.

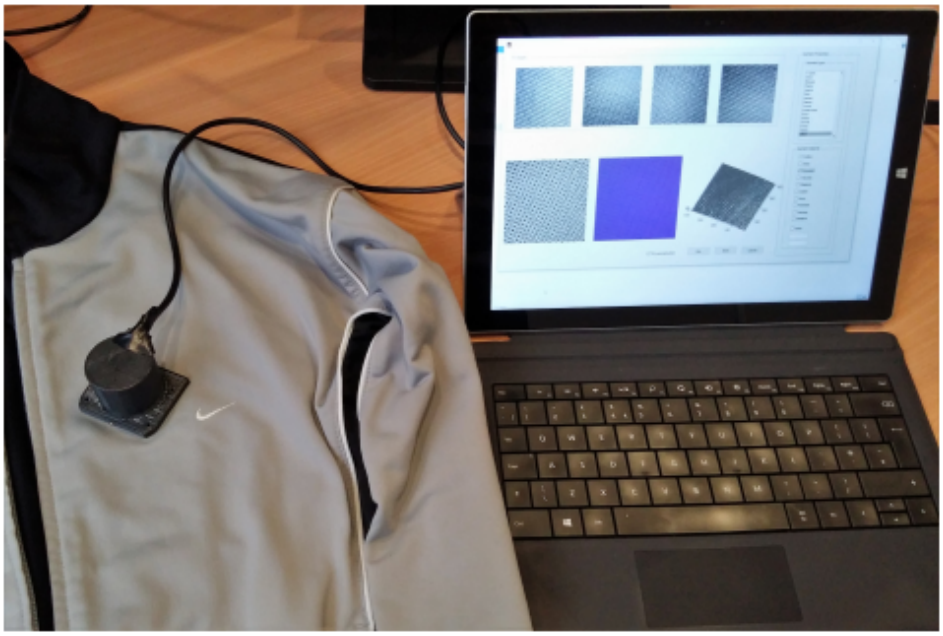


Figura 3.3. Setup di cattura delle immagini (Fonte: The Fabrics Datasets [21])

3.1.1 Classificazione del Dataset

Il dataset non contiene informazioni riguardanti il colore dei tessuti; dunque, prima di poter essere utilizzato ha bisogno di una fase classificazione.

---

le direzioni. Esso indica, dunque, il potere riflettente di una superficie. L'esatto valore della frazione dipende, per lo stesso materiale, dalla lunghezza d'onda della radiazione considerata. Si misura attraverso un albedometro [1].

L'etichettatura delle immagini è stata eseguita totalmente in maniera manuale; dunque si è proceduto, attraverso un riscontro visivo alla scelta del colore di ogni singola immagine. Le classi prese in considerazione sono 12: Arancione, Bianco, Blu, Ciano, Giallo, Magenta, Nero, Rosso, Terra, Verde, Verde Smeraldo e Viola. Durante questa fase si è riscontrato che il dataset contiene immagini che raffigurano colori non presenti nella lista appena citata; quindi, si è deciso di scartare le immagini non corrispondenti alle etichette scelte. Non si sono aggiunte ulteriori etichette perché il dataset non è sufficientemente grande da garantire che ci siano elementi a sufficienza per ogni etichetta.

Conclusa questa fase, ci siamo ritrovati con un dataset suddiviso come mostrato in Tabella 3.1. Come si può notare, non è stato possibile assegnare una label a 3166 immagini poiché non corrispondono a nessuno dei 12 colori scelti.

	Classe	**Numero elementi**
1	Arancione	21
2	Bianco	917
3	Blu	360
4	Ciano	307
5	Giallo	208
6	Magenta	52
7	Nero	1856
8	Rosso	230
9	Terra	157
10	Verde	100
11	Verde Smeraldo	155
12	Viola	228
	Totale	**4591**

Tabella 3.1. Tassonomia del dataset dopo l'etichettatura

3.2 Principali sfide per il riconoscimento del colore

In questa sezione analizzeremo meglio il Fabrics Dataset per capire quali sono le sfide da affrontare nel momento in cui si vuole riconoscere il colore dalle immagini.

Come detto nel Capitolo 2, il principale task della nostra soluzione dovrà essere quello di riconoscere un colore a partire da immagini con presenza di texture e in differenti condizioni di luce.

3.2.1 Come l'acquisizione delle immagini condiziona il colore

Per capire come l'acquisizione delle immagini condiziona il colore possiamo svolgere un piccolo esperimento. Presa un'immagine costituita da un solo colore come quella a sinistra in Figura 3.4, dove ogni pixel corrisponde ad un'unica terna RGB $(0, 134, 205)$, si estrae la sua distribuzione dei pixel visibile a sinistra della Figura 3.4.

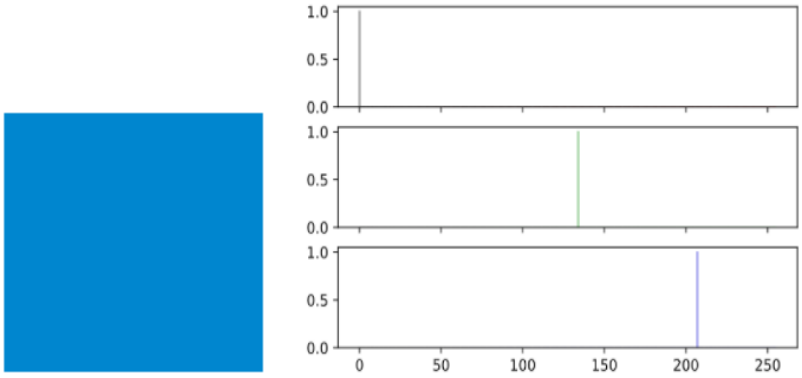


Figura 3.4. Distribuzione di immagine con singolo colore

Come possiamo notare dalla distribuzione questo tipo di immagine digitale rappresenta una singola tonalità di colore e tutti i pixel sono concentrati nei valori che definiscono la tonalità stessa. Prendendo il medesimo colore e acquisendolo attraverso uno smartphone otteniamo il colore e la distribuzione Figura 3.5.

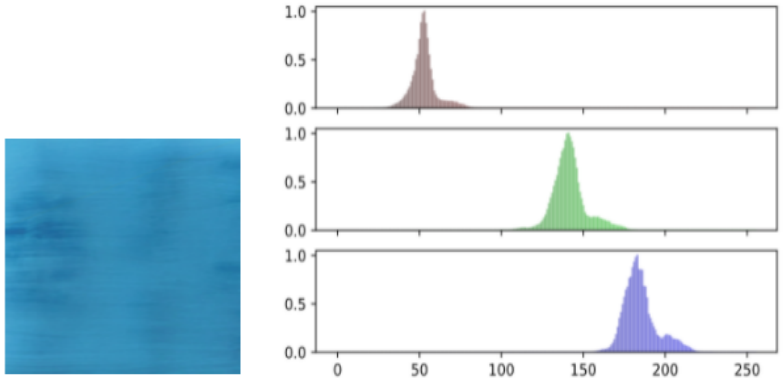


Figura 3.5. Distribuzione di immagine acquisita da uno smartphone

Possiamo notare come la distribuzione dei pixel non si concentra su un singolo punto; ciò è dovuto al semplice fatto che il dispositivo con il quale stiamo realizzando

la fotografia e l'ambiente di acquisizione tendono a modificare il colore visualizzato. Questo problema in campo fotografico viene risolto attraverso una calibrazione del colore effettuata prima dell'acquisizione delle immagini, utilizzando strumenti come quelli proposti dall'azienda X-Rite[2]. Essa propone una correzione del colore attraverso l'utilizzo di *Color Checker*[3] che, insieme all'utilizzo di programmi per l'elaborazione delle immagini e la collabarozione con aziende che producono i dispositivi di acquisizione permettono la correzione del colore. Inoltre, esistono molti studi in letteratura che individuano delle soluzioni per effettuare una correzione automatica del colore applicabile dopo la sua acquisizione, alcune di esse sono: "Color Correction Using Improved Linear Regression Algorithm" [48], "Color Correction Using Root-Polynomial Regression" [27], "Semantic White Balance: Semantic Color Constancy Using Convolutional Neural Network" [12], "Conditional GANs for Multi-Illuminant Color Constancy: Revolution or Yet Another Approach?" [50].

Il Fabrics Dataset, come descritto in precedenza, va ad acquisire delle immagini in diverse condizioni di luce. Dunque, nel nostro dataset, sono presenti sia immagini in ombra che immagini particolarmente esposte alla luce.

Per capire come l'ombra modifica la nostra immagine in termini di distribuzione del colore acquisiamo un'immagine, sempre, attraverso uno smartphone mettendo a confronto distribuzione senza ombra e la distribuzione con ombra. La Figura 3.6 presenta il confronto tra le due distribuzioni: la prima riga mostra l'acquisizione del colore privo di ombra, la seconda riga il colore in ombra.

Come si può notare, tra le due distribuzioni otteniamo uno spostamento verso sinistra dei pixel. Dunque, le immagini acquisite in condizioni di ombra appariranno più scure rispetto alle immagini in buone condizioni di illuminazione.

Il risultato ottenuto non considera immagini parzialmente in ombra; questo perché il dataset utilizzato non contiene questa tipologia di immagini. Poichè la tesi vuole dare una visione completa del problema, elencando tutte le possibili problematiche, non possiamo escludere, che in ambiente reale, le fotografie possano contenere un colore parzialmente in ombra. La Figura 3.7 ci mostra questo tipo di casistica utilizzando un colore descritto dalla terna RGB $(18, 10, 143)$.

Si può notare, dalla distribuzione dei colori che otteniamo in modo abbastanza netto che i pixel si concentrano in due zone differenti, una più chiara e una più scura, dovuto all'ombra.

3.2.2 Il flash glare

Per quanto invece riguarda l'elevata esposizione a fonti luminose il Fabrics Dataset contiene diverse immagini di tessuti altamente riflettenti che sono presenti maggiormente nella classe Giallo. Questo tipo di immagini contengono delle zone altamente illuminate generando il problema del *flash glare*.

[2] X-Rite, Inc. è un fornitore di prodotti per la misurazione e la gestione del colore, con sede a Grand Rapids, Michigan, Stati Uniti.

[3] Il ColorChecker (spesso indicato con il suo nome originale Macbeth ColorChecker o, semplicemente, Macbeth chart) è un target di calibrazione del colore costituito da una disposizione con cornice di cartone di 24 quadrati di campioni dipinti. Il ColorChecker è stato introdotto in un articolo del 1976 di McCamy, Marcus e Davidson nel Journal of Applied Photographic Engineering.

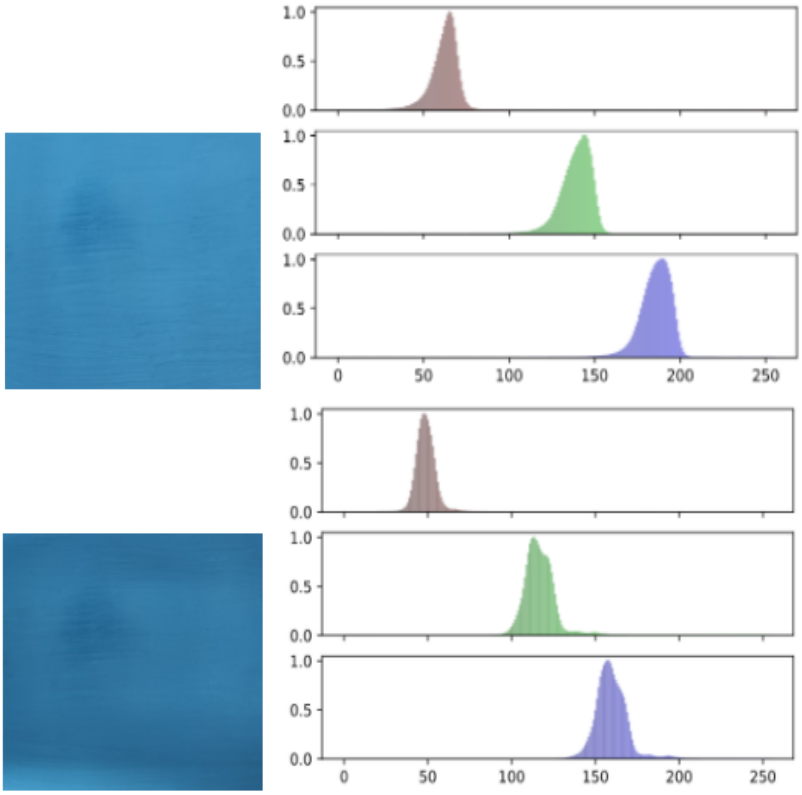


Figura 3.6. La prima riga mostra immagine e distribuzione acquisite in ambiente luminoso. La seconda riga mostra immagine e distribuzione acquisite in ambiente con ombra

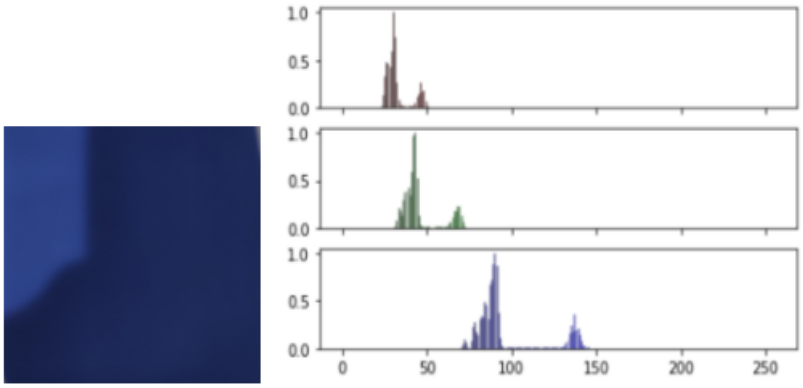


Figura 3.7. Immagine parzialmente in ombra

Il fenomeno del glare avviene nel momento in cui, in presenza di luce intensa, abbiamo difficoltà nel vedere. Per fare un banale esempio possiamo pensare a dei fari abbaglianti che colpiscono lo specchietto della nostra auto rendendoci la guida fastidiosa e difficoltosa. Questo tipo di problematica colpisce anche i dispositivi di acquisizione, sopratutto se dotati di flash. Infatti si potrebbe verificare questo effetto di abbagliamento con persone che portano occhiali da vista, come in Figura 3.8.

Figura 3.8. Esempio di flash glare negli occhiali (Fonte: Spectacle problem removal from facial images based on detail preserving filtering schemes [55])

Il problema è largamente studiato in letteratura dove vengono adottate diverse soluzioni per la sua rimozione, come la rimozione del glare da targhe di auto [54], la riduzione del glare nell'acquisizone delle immagini [31] o la definizione di algoritmi anti-glare negli occhiali [47] .

Nel nostro dataset, come detto in precedenza, il flash glare viene causato da tessuti notevolmente riflettenti, producendo risultati come quelli in Figura 3.9 e causando delle difficoltà nel riconoscimento del colore.

Estrarre il colore dominante, come fatto nella Sezione 2.1, utilizzando immagini con un fenomeno di glare molto grande potrebbe condizionare il colore estratto.

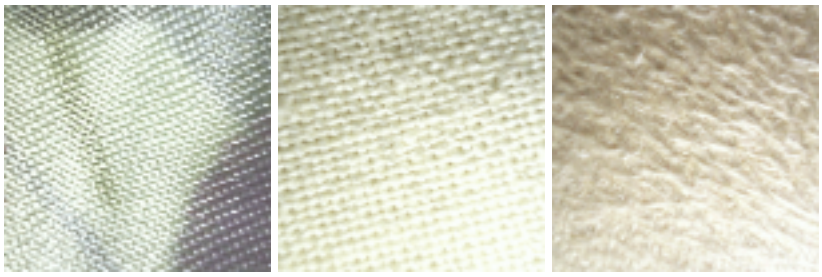


Figura 3.9. Esempi di immagini in Fabrics Dataset con fenomeno di flash glare

3.3 Obiettivi

La tesi non vuole presentare una singola soluzione per la sfida proposta, ma cerca di definire un primo approccio al problema investigando sulle possibili direzioni da intraprendere.

L'obiettivo principale è quello di individuare una soluzione in grado di risolvere le principali sfide proposte, ovvero:

- riconoscimento del colore con ombra;
- riconoscimento del colore anche in presenza di flash glare;
- riconoscimento del colore in presenza di diverse condizioni di luce.

Per farlo verranno valutate le soluzioni che fanno utilizzo di tecniche di deep learning con e senza l'utilizzo del colore dominante.

4

Modelli e analisi con rete Color Deep

Nel capitolo corrente verrano mostrati progettazione, implementazione e risultati del riconoscimento del colore mediante rete Color Deep. In particolare ci concentreremo sull'architettura della rete stessa fornendo una sua implementazione. Successivamente si valuteranno i modelli ottenuti dalla rete effettuando un confronto tra risultati con e senza colore dominante.

4.1 La rete Color Deep

La prima soluzione proposta sfrutta la rete *Color Deep* per effettuare il riconoscimento del colore, che viene appreso dalla rete a partire da immagini di tessuti etichettate.

4.1.1 Architettura della rete

Le rete *Color Deep*, come è possibile vedere in Figura 4.1, è costituita da due reti di base formate da 8 layer ciascuno con un totale di 16 strati convoluzionali, il primo dei quali usa un kernel 11×11, mentre il secondo livello sfrutta il kernel 3×3, con un totale di 128 kernel. Invece il terzo ricorre al kernel 3×3 ,con un totale 192 kernel. il quarto livello impiega il kernel 3×3 con un totale di 192 kernel. Infine, il quinto livello si serve di un kernel 3×3, con un totale di 128 kernel. Ogni layer convoluzionale è seguito da un livello di normalizzazione e un livello di pooling di dimensione 3×3 con passo pari a 2. Come visto nel Capitolo 2, il layer convoluzionale è fondamentale per estrarre feature dalle immagini ed esegue una convoluzione. La funzione di attivazione utilizzata per questi livelli convoluzionali è la *Rectified Linear Unit (ReLU)*.

La ReLu è una funzione di attivazione che non è stata trattata nella Sezione 2.3.1. Essa è stata proposta da Nair e Hinton nel 2010 e, da allora, è la funzione di attivazione più utilizzata nelle applicazioni di *Deep Learning*. Il suo elevato utilizzo è derivato dal fatto che offre migliori prestazioni rispetto alle funzioni di attivazione `tanh` e `Sigmoid`. La funzione ReLU, inoltre, è pressoché lineare, e questo le permette di risultare più semplice da ottimizzare attraverso l'utilizzo degli algoritmi di discesa del gradiente [40].

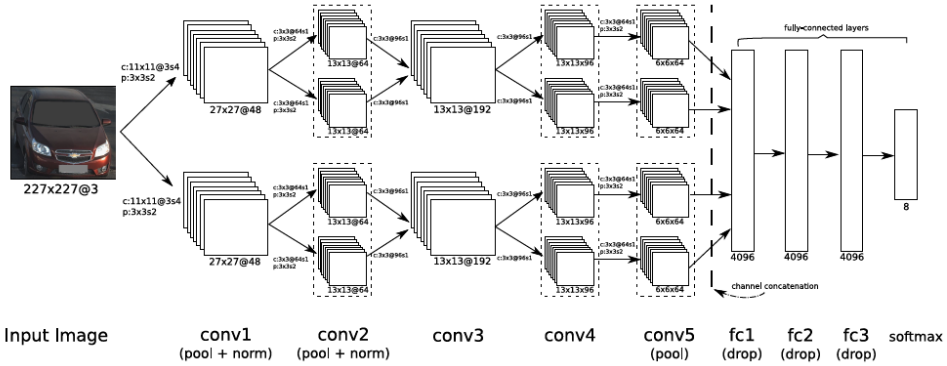


Figura 4.1. Architettura della rete Color Deep (Fonte: "Vehicle Color Recognition using Convolutional Neural Network" [40]).

La funzione ReLU viene descritta dalla seguente equazione:

$$f(x) = max(0, x) = \begin{cases} x_i & per\ x_i \geq 0 \\ 0 & per\ x_i < 0 \end{cases} \tag{4.1}$$

Tornando all'architettura della rete *Color Deep* possiamo vedere dalla figura che il sesto, il settimo e l'ottavo strato sono *fully connected layer*. L'ingresso della rete è un'immagine a 3 canali con risoluzione 227×227. In totale la rete contiene un numero di neuroni pari a 658.280.

4.1.2 Generazione del modello

La generazione del modello è stata effettuata sfruttando il dataset etichettato come visto nel Capitolo 3. In accordo con gli obiettivi della tesi, ci siamo chiesti se la rete proposta da [40] sia effettivamente adatta per il riconoscimento del colore nel *Fabrics Dataset*.

Il nostro dataset è fortemente sbilanciato e questo implica che, prima di effettuare una fase di addestramento per cercare di valutare meglio il comportamento della rete, si è eseguito un ricampionamento del dataset.

Ricampionamento del dataset

Quando parliamo di ricampionamento del dataset facciamo riferimento ad una serie di tecniche che ci permettono in fase di pre-processing, di alterare la distribuzione delle classi in modo da eliminare lo sbilanciamento [39]. Esistono diverse tecniche che possono essere applicate per effettuare un ricampionamento, tra queste citiamo:

- *OverSampling*: questa tecnica viene utilizzata per aumentare il numero di campioni all'interno del nostro dataset. Per farlo vengono presi campioni meno presenti e replicati all'interno del dataset aumentando, così, il numero dei membri della classe di minoranza. Il principale vantaggio di questa tecnica è che permette di non perdere informazioni sul set di dati di partenza.

- *UnderSampling*: questa tecnica va a ridurre la classe con la dimensione maggiore, mirando a ridurre il numero di campioni di maggioranza per ribilanciare la distribuzione delle classi.
- *Cost-Sensitive Learning (CSL)*: utilizza i costi di classificazione errata minimizzando il costo totale. L'obiettivo di questa tecnica è, principalmente, quello di perseguire un'elevata precisione di classificazione degli esempi in un insieme di classi conosciute [39].

Si è scelto di effettuare un under-sampling del dataset, per cui tutti i test effettuati con la rete *Color Deep* utilizzano un numero di elementi pari a 1200. Ogni classe nel dataset sottocampionato ha un numero di elementi pari a 100, e questo implica che per le classi con meno elementi, in fase di pre-processing abbiamo effettuato l'*oversempling*.

La scelta di utilizzare, per la generazione del modello, un dataset sottocampionato è nata dal fatto che esso ci permette di effettuare un migliore confronto tra modelli differenti.

Data augmentation

Durante i nostri test si fa uso di tecniche di *data augmentation* per evitare fenomeni di *overfitting*. Esso è un problema largamente presente nel *data mining* e si verifica quando un modello si adatta ai dati di addestramento. Questo implica che il modello non riesce più a generalizzare e a riconoscere dati al di fuori del training set. A sinistra della Figura 4.2 viene mostrato un modello in cui l'errore di training ha un punto di flesso nella settima epoca e inizia ad aumentare. Questo sta ad indicare che il modello si sta adattando eccessivamente ai dati di training. Invece, la Figura 4.2 mostra le condizioni in cui ciò non accade [49].

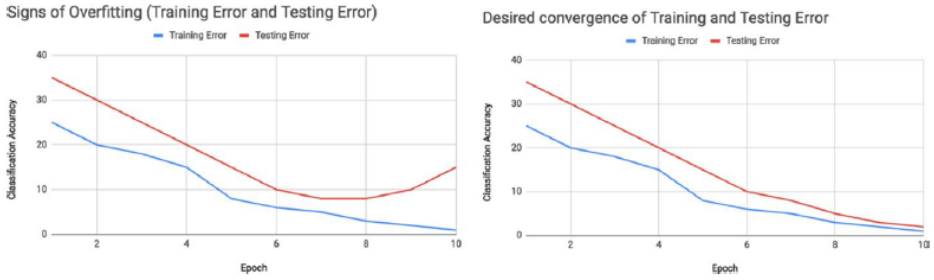


Figura 4.2. A sinistra un modello con overfitting. A destra un modello senza fenomeno di overfitting (Fonte: A survey on Image Data Augmentation for Deep Learning [49]).

Per evitare il problema di overfitting esistono diverse strategie; tuttavia, in questa sezione indicheremo solo quelle effettivamente adottate nei test effettuati con la Color Deep, esse sono:

- *Random flip augmentation*: inverte le righe o le colonne di pixel effettuando dunque un capovolgimento in orizzonatale o in verticale dell'immagine. Solitamente

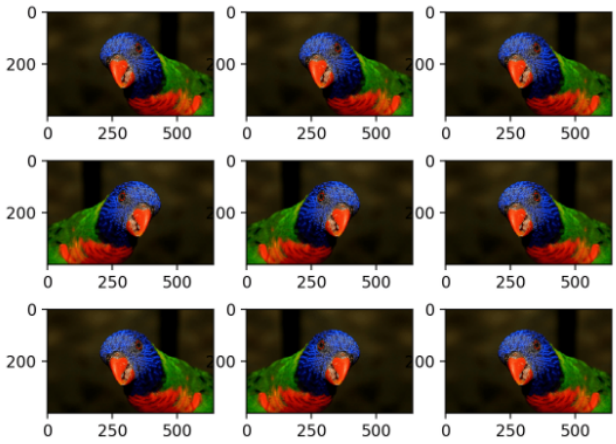


Figura 4.3. Esempio di random horizontal flip augmentation (Fonte:How to Configure Image Data Augmentation in Keras [17]).

viene utilizzato un flip in base al dataset che prendiamo in considerazione. La Figura 4.3 riporta un esempio di capovolgimento orizzontale.

- *Random zoom augmentation*: ingrandisce casualmente l'immagine e aggiunge nuovi valori di pixel attorno ad essa o interpola, rispettivamente, i valori di pixel. La Figura 4.4 ci mostra un esempio di immagine ingrandita casualmente [17].

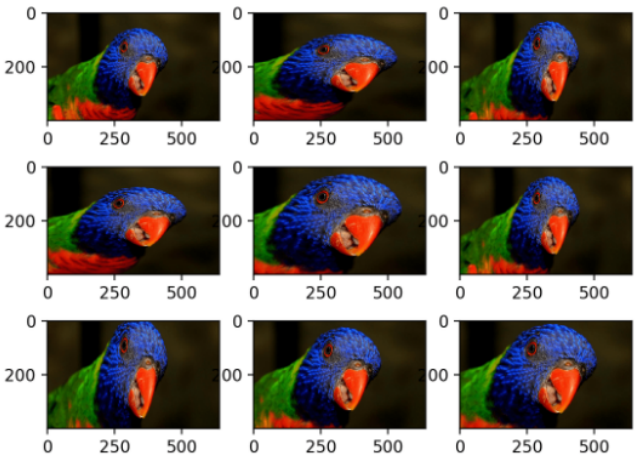


Figura 4.4. Esempio di random zoom augmentation (Fonte: How to Configure Image Data Augmentation in Keras [17]).

Valutazione del modello

La fase di addestramento è fondamentale per ottenere un risultato soddisfacente in termini di performance del modello stesso. Per poter dunque individuare la capacità della rete di riconoscere il colore nel dataset proposto utilizziamo la *K-cross fold validation (K-fold)*.

La K-fold è una procedura largamente utilizzata in data mining per la valutazione di modelli perché ci permette di avere una stima meno distorta e meno ottimistica dell'abilità del modello rispetto ad altri metodi di valutazione. Possiamo considerare questa tecnica come un ricampionamento del dataset. La procedura generale per eseguirla è la seguente:

1. Si mescola casualmente il set di dati.
2. Si divide il set in K gruppi.
3. Per ogni gruppo:
   a) Si sceglie un gruppo come test set.
   b) Si scelgono i restanti gruppi come training set.
   c) Si esegue l'addestramento sul training set.
   d) Si valuta il modello.
4. Si esegue una valutazione media tra i vari modelli addestrati.

La Figura 4.5 ci mostra in maniera grafica il comportamento della k-fold.

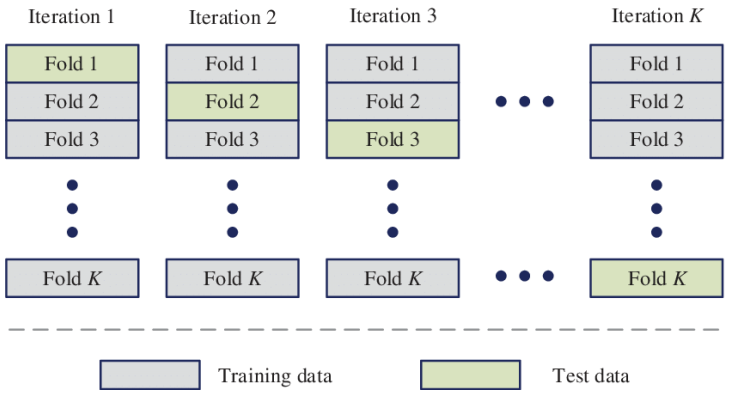


Figura 4.5. Architettura della rete Color Deep (Fonte: "Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives" [44].

In fase di addestramento si eseguirà una k-cross fold validation con k pari a 5. Attraverso questo metodo valuteremo i modelli addestrati con e senza colore dominante servendoci delle metriche di *Precision*, *Recall*, *F1-Score* e *Accuracy*. Per definire le metriche appena citate dobbiamo introdurre la *matrice di confusione*, in cui nelle colonne vengono rappresentate le predizioni mentre nelle righe le classi effettive. Righe e colonne possono anche essere invertite. La Figura 4.6 mostra la matrice di confusione binaria (a due classi).

A questo punto possiamo spiegare cosa misurano le varie metriche:

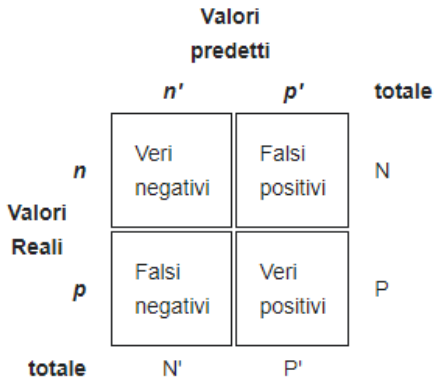


Figura 4.6. Matrice di confusione binaria (Fonte: Wikipedia).

- *Precision*: possiamo definirla come il rapporto tra il numero di previsioni corrette per una determinata classe sul totale degli elementi predetti di quella classe. La Precision misura il grado di precisione del modello per un determinato tipo di classe. Essa assume un range di valori che va da 0 a 1. La formula è la seguente:

$$Precision_i = \frac{TP_i}{TP_i + FN_i} \tag{4.2}$$

  Nell'equazione i definisce la classe i-esima.
- *Recall*: la recall, spesso qualificante come sensibilità di un modello, viene definito come il rapporto tra previsioni corrette per una classe e il numero effettivo di elementi che compongono la classe. Quindi la misura ci permette di capire quanto il modello è sensibile per una determinata classe. Esso assume un range di valori che va da 0 a 1.

$$Recall_i = \frac{TP_i}{TP_i + FP_i} \tag{4.3}$$

  Nell'equazione i definisce la classe i-esima.
- *F1-Score*: possiamo vedere la metrica F1-Score come la media armonica tra Precision e Recall. La formula è la seguente:

$$F1 - Score_i = 2 \times \frac{Precision_i \times Recall_i}{Precision_i + Recall_i} \tag{4.4}$$

  Nell'equazione, i definisce la classe i-esima. Il range di valori che essa può assumere varia tra 0 e 1, può essere pari a 1 nel momento in cui sia Precision sia Recall sono pari a 1 (caso ideale).
- *Accuracy*: l'accuracy è il parametro fondamentale per la valutazione di un modello e misura la percentuale di previsioni corrette. La sua formula è la seguente:

$$Accuracy_i = \frac{TP_i + TN_i}{TP_i + FP_i + TN_i + FN_i} \tag{4.5}$$

  Nell'equazione i definisce la classe i-esima.

4.2 Implementazione della rete

L'implementazione della rete è stata eseguita utilizzando il linguaggio Python nonchè le librerie `Keras` e `Tensorflow` (TF). Quest'ultima è una libreria open source sviluppata da Google e largamente impiegata per applicazioni di *Deep Learning*. Infatti, essa fornisce una serie di API di cui si servono diversi applicativi commerciali come Gmail, Google Foto e il Riconoscimento vocale. Anche *Keras* è una libreria open source scritta in Python, usata per l'apprendimento automatico e le reti neurali. Essa è progettata come un'interfaccia con un livello di astrazione superiore alle altre librerie simili di più basso livello. *Keras* supporta come back-end le librerie `TensorFlow`, `CNTK` (Microsoft Cognitive Toolkit) e `Theano`. Bisogna ricodare che, attualmente, TF con la sua Versione 2.0, ha completamente integrato la libreria Keras, definendo un unico modulo di riferimento per la gestione del *Deep Learning*.

Il listato 4.1 ci mostra l'implementazione della rete Color Deep presente su GitHub all'indirizzo: `https://github.com/jasonqhuang/Car_Color_CNN`.

```
1   def color_net(num_classes):
2       # placeholder for input image
3       input_image = Input(shape=(224,224,3))
4       # ========================================= TOP BRANCH ================================================
5       # first top convolution layer
6       top_conv1 = Conv2D(filters=48,kernel_size=(11,11),strides=(4,4),
7                          input_shape=(224,224,3),activation='relu')(input_image)
8       top_conv1 = BatchNormalization()(top_conv1)
9       top_conv1 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(top_conv1)
10
11      # second top convolution layer / split feature map by half
12      top_top_conv2 = Lambda(lambda x : x[:,:,:,:24])(top_conv1)
13      top_bot_conv2 = Lambda(lambda x : x[:,:,:,24:])(top_conv1)
14
15      top_top_conv2 = Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
            top_top_conv2)
16      top_top_conv2 = BatchNormalization()(top_top_conv2)
17      top_top_conv2 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(top_top_conv2)
18
19      top_bot_conv2 = Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
            top_bot_conv2)
20      top_bot_conv2 = BatchNormalization()(top_bot_conv2)
21      top_bot_conv2 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(top_bot_conv2)
22
23      # third top convolution layer
24      # concat 2 feature map
25      top_conv3 = Concatenate()([top_top_conv2,top_bot_conv2])
26      top_conv3 = Conv2D(filters=192,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(top_conv3)
27
28      # fourth top convolution layer / split feature map by half
29      top_top_conv4 = Lambda(lambda x : x[:,:,:,:96])(top_conv3)
30      top_bot_conv4 = Lambda(lambda x : x[:,:,:,96:])(top_conv3)
31
32      top_top_conv4 = Conv2D(filters=96,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
            top_top_conv4)
33      top_bot_conv4 = Conv2D(filters=96,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
            top_bot_conv4)
34
35      # fifth top convolution layer
36      top_top_conv5 = Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
            top_top_conv4)
37      top_top_conv5 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(top_top_conv5)
38
39      top_bot_conv5 = Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
            top_bot_conv4)
40      top_bot_conv5 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(top_bot_conv5)
41
42      # ========================================= TOP BOTTOM ================================================
43      # first bottom convolution layer
44      bottom_conv1 = Conv2D(filters=48,kernel_size=(11,11),strides=(4,4),
45                            input_shape=(227,227,3),activation='relu')(input_image)
46      bottom_conv1 = BatchNormalization()(bottom_conv1)
47      bottom_conv1 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(bottom_conv1)
48
49      # second bottom convolution layer / split feature map by half
50      bottom_top_conv2 = Lambda(lambda x : x[:,:,:,:24])(bottom_conv1)
51      bottom_bot_conv2 = Lambda(lambda x : x[:,:,:,24:])(bottom_conv1)
52
53      bottom_top_conv2 = Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
            bottom_top_conv2)
54      bottom_top_conv2 = BatchNormalization()(bottom_top_conv2)
55      bottom_top_conv2 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(bottom_top_conv2)
```

```
56
57      bottom_bot_conv2 = Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
                bottom_bot_conv2)
58      bottom_bot_conv2 = BatchNormalization()(bottom_bot_conv2)
59      bottom_bot_conv2 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(bottom_bot_conv2)
60
61      # third bottom convolution layer / concat 2 feature map
62      bottom_conv3 = Concatenate()([bottom_top_conv2,bottom_bot_conv2])
63      bottom_conv3 = Conv2D(filters=192,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
                bottom_conv3)
64
65      # fourth bottom convolution layer / split feature map by half
66      bottom_top_conv4 = Lambda(lambda x : x[:,:,:96])(bottom_conv3)
67      bottom_bot_conv4 = Lambda(lambda x : x[:,:,96:])(bottom_conv3)
68
69      bottom_top_conv4 = Conv2D(filters=96,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
                bottom_top_conv4)
70      bottom_bot_conv4 = Conv2D(filters=96,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
                bottom_bot_conv4)
71
72      # fifth bottom convolution layer
73      bottom_top_conv5 = Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
                bottom_top_conv4)
74      bottom_top_conv5 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(bottom_top_conv5)
75
76      bottom_bot_conv5 = Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),activation='relu',padding='same')(
                bottom_bot_conv4)
77      bottom_bot_conv5 = MaxPooling2D(pool_size=(3,3),strides=(2,2))(bottom_bot_conv5)
78
79      # ============================================ CONCATENATE TOP AND BOTTOM BRANCH ================================
80      conv_output = Concatenate()([top_top_conv5,top_bot_conv5,bottom_top_conv5,bottom_bot_conv5])
81
82      # Flatten
83      flatten = Flatten()(conv_output)
84
85      # Fully-connected layer
86      FC_1 = Dense(units=4096, activation='relu')(flatten)
87      FC_1 = Dropout(0.6)(FC_1)
88      FC_2 = Dense(units=4096, activation='relu')(FC_1)
89      FC_2 = Dropout(0.6)(FC_2)
90      output = Dense(units=num_classes, activation='softmax')(FC_2)
91
92      model = Model(inputs=input_image,outputs=output)
93      sgd = SGD(learning_rate=1e-3, decay=1e-6, momentum=0.9, nesterov=True)
94      model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
95
96      return model
```

Listato 4.1. Codice di implementazione in keras della rete Color Deep.

4.3 Test e analisi dei risultati

Vediamo ora, i risultati ottenuti con la rete appena presentata e addestrata sul dataset definito nella Sezione 4.1.2. I test sono stati effettuati utilizzando *Google Colab (*`Colab`*)*[1], il codice è stato implementato utilizzando Python e le librerie `OpenCv`, `Tensorflow` e `Scikit-learn`[2].

I parametri della rete sono definiti come di seguito specificato:

- Ottimizzatore SGD;
- Learning rate pari a 0.001;
- Decadimento pari a 0.000001;
- Momentum pari a 0.9.

[1] Google Colaboratory (noto, anche, come `Colab`) è un ambiente di notebook Jupyter gratuito che viene eseguito nel cloud e memorizza i suoi notebook su Google Drive.

[2] `Scikit-learn` (ex scikits.learn) è una libreria open source di apprendimento automatico per il linguaggio di programmazione Python.

La scelta di utilizzare un learning rate e un decadimento fisso che non varia con il numero di epoche è dovuta alla dimensione del dataset che ha un numero di elementi piccolo.

Invece, per quanto concerne la fase di training e valutazione, come detto nella Sezione 4.1.2, si utilizza la K-fold con $k = 5$ e si esegue la suddivisione del dataset nel seguente modo:

- Test set pari al 20% del set di dati di partenza; dunque, per ogni iterazione, il test set ha un numero di immagini pari a 240.
- Traning set pari all'80% del set di dati di partenza. Questo verrà, a sua volta, suddiviso in Covalidation set e Training set. questo, implica che si riesegua un'ulteriore suddivisone 80/20 del Training set ottenendo, dunque, 816 elementi per il Training set e 144 elementi per il Covalidation set.

La Figura 4.7 mostra, in maniera più esplicativa, la suddivisione appena descritta.

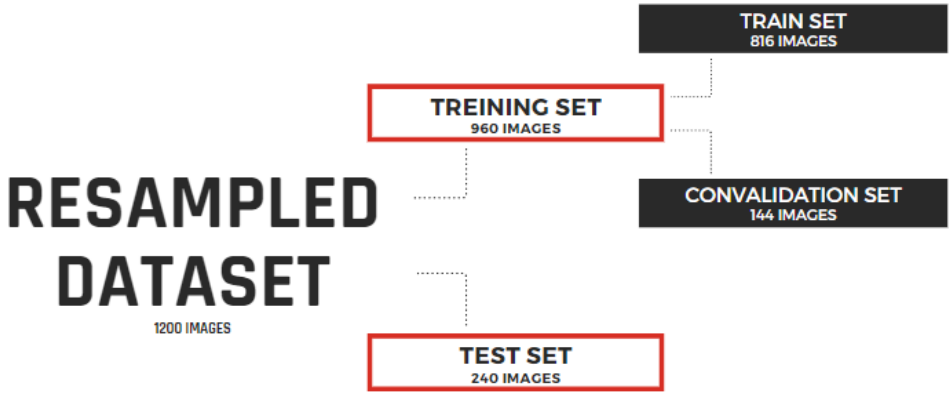


Figura 4.7. Suddivisione del dataset.

La stessa impostazione dei parametri è stata utilizzata sia per la rete con colore dominante che per la rete senza colore dominante.

4.3.1 Valutazione dei modelli

La valutazione metterà a confronto il risultato ottenuto con colore dominante e quello senza colore dominante. Inoltre per ogni tipologia di test, verrà mostrato il migliore modello (best model) in termini di accuratezza.

Best model senza colore dominante

La Tabella 4.1 presenta i risultati del miglior modello ottenuto senza l'utilizzo del colore dominante.

Classe	Precision	Recall	F1-Score
Arancione	0.74	1.00	0.85
Bianco	0.90	0.95	0.93
Blu	0.83	0.95	0.88
Ciano	0.47	0.40	0.43
Giallo	0.69	0.55	0.61
Magenta	0.67	1.00	0.80
Nero	0.75	0.75	0.75
Rosso	1.00	0.90	0.95
Terra	0.94	0.85	0.89
Verde	0.93	0.70	0.80
Verde Smeraldo	0.67	0.40	0.50
Viola	0.65	0.75	0.70
		Accuracy	**0.77**
Media	**0.77**	**0.77**	**0.76**

Tabella 4.1. Report del migliore modello addestrato (fold 4) senza colore dominante.

Best model con colore dominante

La Tabella 4.2 presenta i risultati del miglior modello con colore dominante, ottenuto attraverso l'utilizzo della libreria `ColorThief`, come visto nel Capitolo 2.

Confronto con e senza colore dominante

Quello che ci aspettiamo dai modelli è che, nel caso di utilizzo del colore dominante, la rete abbia performance maggiori rispetto al caso senza colore dominante. Come è possibile vedere in Tabella 4.3, che mostra un confronto tra color-deep con e senza colore dominante, in realtà, il comportamento migliore viene ottenuto senza colore dominante.

Il risultato ottenuto è dovuto al fatto che, per alcune immagini, l'estrazione del colore dominante non sempre individua il colore effettivo. Nel caso di immagini particolarmente scure, o che hanno trame particolari, il colore dominante può risultare diverso da quello che vogliamo riconoscere.

Classe	Precision	Recall	F1-Score
Arancione	0.95	1.00	0.98
Bianco	0.76	0.95	0.84
Blu	0.67	0.70	0.68
Ciano	0.00	0.00	0.00
Giallo	0.48	1.00	0.65
Magenta	0.89	0.80	0.84
Nero	0.37	0.35	0.36
Rosso	0.68	0.95	0.79
Terra	0.89	0.85	0.87
Verde	0.63	0.60	0.62
Verde Smeraldo	0.48	0.60	0.53
Viola	1.00	0.15	0.26
		Accuracy	**0.66**
Media	**0.65**	**0.66**	**0.62**

Tabella 4.2. Report del migliore modello addestrato (fold 1) con l'utilizzo del colore dominante.

Classe	Precision	Recall	F1-Score	Accuracy
Color deep senza colore dominante	0.72	0.716	0.696	0.72
Color Deep con colore dominante	0.5	0.62	0.594	0.628

Tabella 4.3. Confronto Color Deep tra utilizzo con colore dominante e senza colore dominante.

4.4 Il Transfer Learning

Il *Transfer Learning (TL)* è una tecnica largamente utilizzata nel Deep Learning perché permette di risolvere un determinato problema prendendo come punto di partenza un modello sviluppato per risolvere un problema differente o simile (problema di origine) a quello di cui stiamo cercando una soluzione (problema di riferimento). L'approccio è particolarmente efficace nel momento in cui le caratteristiche apprese dal primo modello siano particolarmente generali, permettendo, così, di poter sfruttare questa capacità per altri tipi di compiti.

Il TL viene largamente utilizzato per problemi di detection e segmentation di immagini; in questi casi, infatti, le reti sono particolarmente complesse e richiedono molto tempo per essere addestrate completamente. Solitamente si sfrutta un modello pre-addestrato su un dataset impegnativo (come quello proposto da *ImageNet*), in modo da partire con una rete già in grado di estrarre determinate feature dai dati. Le reti CNN risultano più generiche ai primi livelli mentre più specifiche per i livelli successivi [10]. Nel momento in cui vengono applicate tecniche di TL si va a

bloccare l'apprendimento per i primi livelli convoluzionali addestrando solo i livelli più profondi. Questo tipo di pratica è largamente utilizzata in letteratura.

Il Transfer Learning ci permette di risparmiare del tempo nel momento in cui stiamo generando un modello molto complesso; infatti, la rete, in questo modo, non ha bisogno di apprendere da zero come riconoscere un oggetto, ma potrebbe sfruttare ciò che è stato appreso dal modello di partenza raggiungendo, anche se con dataset più piccoli, dei risultati soddisfacenti in termini di performance.

4.4.1 Implementazione del Transfer Learning

Il test che ci proponiamo di effettuare in questo caso consiste nello sfruttare il dataset dei veicoli in [42] per addestrare un modello di riconoscimento.

In letteratura per sfruttare il TL si definiscono due metodologie: *approccio con sviluppo del modello* o *approccio che sfrutta pesi pre-addestrati*. I due metodi differiscono su come viene scelto il modello che verrà successivamente riusato per il nuovo dominio applicativo.

Nel nostro caso utilizzeremo l'approccio con sviluppo del modello; i passi per applicare il TL sono i seguenti:

- *Selezione del task di origine* - Si individua un problema di modellazione predittiva con abbondanza di dati, che ha caratteristiche simili al problema di interesse.
- *Sviluppo del modello di origine* - Si sviluppa un modello in grado di ottenere buone performance nel dominio di origine.
- *Riutilizzo del modello* - Il modello addestrato nel dominio di origine viene riutilizzato come base di partenza per il dominio di interesse. Ciò può comportare l'utilizzo di tutto o parte del modello, a seconda della tecnica di modellazione utilizzata [15].
  Per riconoscere il colore dei tessuti, dunque, si è fatto uso del modello addestrato dai veicoli.

Nel nostro caso verrà utilizzato il dataset sui veicoli "Cars Dataset" presente in `https://ai.stanford.edu/~jkrause/cars/car_dataset.html`. Esso contiene $16,185$ immagini (per maggiori dettagli al dataset si veda [36]).

Per etichettare il dataset si è scelto di utilizzare i colori proposti in [42], e, per farlo si sono utilizzate insieme le tecniche di individuazione del colore dominante e di ricerca del colore (rispettivamente, trattate nella Sezione 2.2.1 e nella Sezione 2.1). La Figura 4.8 mostra le immagini presenti nel "Cars Dataset".

Per addestrare il modello si è utilizzata la rete *Color Deep* che abbiamo trattato nel Capitolo 4 con gli stessi parametri definiti in Sezione 4.3.

Il modello addestrato con i veicoli è stato successivamente usato come modello di origine per il riconoscimento del colore nei tessuti. Il nostro modello di riferimento è stato generato utilizzando la stessa tecnica vista nella Sezione 4.3.

4.4.2 Analisi dei risultati con tecniche di Transfer Learning

I risultati ottenuti da questo test sono confrontati, in Tabella 4.4, con quelli ottenuti senza utilizzo di *transfer learning*.

Figura 4.8. Esempio di immagini presenti nel "Cars Dataset" [36] (Fonte: `https://ai.stanford.edu/~jkrause/cars/car_dataset.html`).

Classe	Precision	Recall	F1-Score	Accuracy
Color deep	0.72	0.716	0.696	0.72
Color deep TL	0.72	0.712	0.688	0.71

Tabella 4.4. Confronto con e senza tecnica di transfer learning.

La Tabella 4.4 mostra l'assenza di un vero e proprio miglioramento nel riconoscimento del colore. Inoltre, per quanto concerne l'accuratezza del modello, si può notare un decremento dell'1% dovuto, principalmente, alla differenza tra i due dataset. Nel caso dei veicoli la rete impara anche a riconoscere le forme del veicolo e a capire quali sono le zone di maggiore interesse per distinguere il colore. Invece, nel caso del Fabrics Dataset, questo avviene in maniera diversa; la rete non deve far riferimento a determinate forme ma deve cambiare totalmente il modo in cui sceglie la regione di interesse per individuare il colore.

5

Modelli e analisi con metodo delle differenze

In questo capitolo si discuterà il metodo delle differenze illustrando il suo funzionamento. Successivamente si introdurranno le varie reti utilizzate per effettuare i test. Infine, presenteremo le analisi dei modelli ottenuti attraverso un confronto dei risultati.

5.1 Il metodo delle differenze

Il metodo delle differenze può essere considerato come un'alternativa al semplice utilizzo della CNN. La soluzione proposta aggiunge una fase di pre-processing alle immagini in modo da aggiungere informazione utile al riconoscimento del colore.

La trasformazione che applichiamo alle immagini riguarda il calcolo delle differenze rispetto ai colori di riferimento. Questo vuol dire che calcoleremo 12 immagini che rappresentano le differenze del colore da classificare rispetto ai riferimenti che abbiamo scelto nella fase di etichettatura del dataset.

Il riconoscimento avviene sempre mediante l'utilizzo del *Deep Learning*; dunque la rete riceverà in input delle immagini costituite da 36 canali, cioè 12×3.

Questa procedura viene eseguita perché il nostro dataset non ha un numero di elementi elevato, ed effettuando questo pre-processing spostiamo il problema in un dominio differente, che è quello delle differenze. Il cambio di dominio è fondamentale per cercare di semplificare il problema di riconoscimento. La Figura 5.1 mostra il calcolo delle 12 differenze; in alto al centro troviamo il tessuto mentre sotto troviamo le differenze calcolate rispetto ai riferimenti.

5.1.1 Le rete CNN-Differenze

La rete utilizzata per applicare questa soluzione sarà diversa rispetto alla rete Color Deep presentata nel Capitolo 4. In questo caso, forti dell'esperienza e dei test effettuati nel caso precedente, abbiamo optato nel realizzare una rete CNN sequenziale con pochi livelli.

La scelta di utilizzare una rete poco profonda è dettata dal fatto che il nostro dataset ha poche immagini. Infatti, reti molto profonde hanno il problema di dover

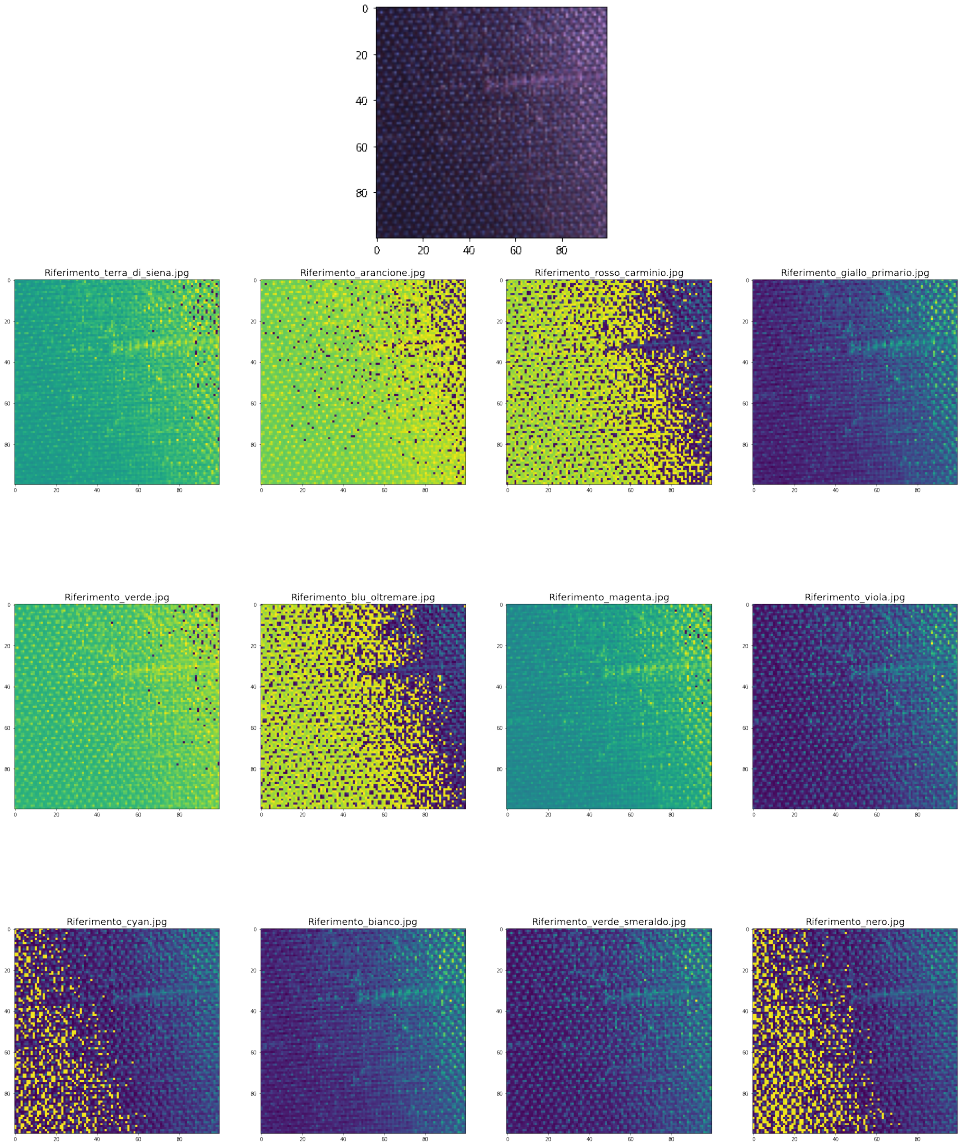


Figura 5.1. Esempio di calcolo delle 12 differenze.

allenare un numero di parametri particolarmente elevato. Nel caso in cui si hanno pochi dati, reti con un numero elevato di livelli tendono ad adattarsi maggiormente ai dati generando problemi di *overfitting*.

La rete alla differenze è costituita da uno strato di input che riceve immagini di dimensioni $100 \times 100 \times 36$, dove 36 sono i canali. Subito dopo lo strato di input, abbiamo un layer di tipo convoluzionale con kernel 3×3, seguito da un *Max Pooling layer* di dimensione 2×2. Il secondo livello della rete è costituito sempre da un secondo strato di tipo convoluzionale con la stessa dimensione del kernel definito in

precedenza, anch'esso seguito da uno strato di pooling di dimensione 2×2. La parte finale della rete è costituita da un *Dense Layer* con 128 neuroni collegato ad un layer di output con attivazione *softmax*. In totale il numero di parametri della rete è $175,476$. Da adesso in avanti chiameremo la rete con il nome di *CNN-differenze*.

Implementazione della rete

Il Listato 5.1 presenta l'implementazione della rete decritta in precedenza. Essa come fatto nel Capitolo 4, è stata implementata utilizzando il linguaggio *Python* e la libreria `TensorFlow`.

```
1   def model():
2     cnn_model = Sequential()
3
4     cnn_model.add(Conv2D(100, 3, 3, input_shape = (100, 100, 36), activation = 'relu'))
5
6     cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
7
8     cnn_model.add(Conv2D(100, 3, 3, activation = 'relu'))
9     cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
10
11    cnn_model.add(Flatten())
12
13    cnn_model.add(Dense(128, activation = 'relu'))
14    cnn_model.add(Dense(12, activation = 'softmax'))
15
16
17    cnn_model.compile(loss='categorical_crossentropy',
18                optimizer='adam',
19                metrics=['accuracy'])
20    return cnn_model
```

Listato 5.1. Codice di implementazione in `tensorflow/keras` della rete CNN sequenziale alle differenze.

Possiamo notare che, come ottimizzatore, si è utilizzato `Adam` perché fornisce maggiori prestazioni su dataset di piccole dimensioni.

5.1.2 ResNet, InceptionResnet e DenseNet

Oltre ad utilizzare la rete definita in precedenza, per effettuare i test si è cercato di capire se altre reti molto usate in letteratura riuscissero ad ottenere risultati migliori rispetto alla rete proposta.

Per farlo si sono scelte diverse reti: *ResNet50v2*, *ResNet152v2*, *InceptionResnet*, *DenseNet169* e *DenseNet201*. Queste sono largamente usate per diversi tipi di applicazioni e ambiti differenti; in [43] viene utilizzata la *ResNet* per diagnosticare in modo corretto l'infezione di malaria partendo da immagini cellulari; in [30] viene utilizzata la *DenseNet* per eseguire una segmentazione anatomica del cervello umano. Inoltre, queste reti sono spesso utilizzate come base per realizzare reti ulteriormente complesse specifiche per determinate attività.

ResNet

La *Deep Residual Network (ResNet)* è stata principalmente introdotta per risolvere il problema del degrado di una rete profonda. Infatti, quest'ultima ha un aumento dell'errore maggiore all'aumentare delle epoche rispetto a quello che si ottiene con una rete meno profonda. Per risolvere questo problema la *ResNet* aggiunge dei blocchi chiamati redisuali per migliorare le prestazioni del modello.

Come visto nella Sezione 2.3.1 le reti neurali calcolano il gradiente della funzione di costo per individuare l'aggiornamento da applicare ai parametri retropropagando l'errore calcolato.

Se il numero dei livelli cresce, il gradiente può subire due tipi di effetti:

- *exploding gradient*, generando parametri non gestibili dall'elaboratore e causando, dunque, overflow;
- *vanishing gradient* determinando un aggiornamento minimo dei pesi e causando un rallentamento nel training [41].

I blocchi residuali utilizzano il concetto di *skip connections*, che permette di diminuire il problema del *vanishing gradient* e dell'*exploding gradient*. Ciò garantisce che i livelli superiori del modello non abbiano prestazioni peggiori rispetto ai livelli inferiori, eliminando il problema della degradazione.

La rete ha avuto enorme successo vincendo la competizione ISLVRC[1] 2015 e la competizione di segmentazione COCO[2] 2015.

Esistono diversi tipi di *ResNet*; la prima introdotta è la ResNet34 [32] che costituisce la base per le diverse alternative. Per i test con le differenze sono state usate la ResNet50v2 e la ResNet152v2, dove 50 e 152 indicano il numero di blocchi residuali. La Figura 5.2 mostra la configurazione generale della rete ResNet; attraverso la legenda si specifica il numero di blocchi residuali per ottenere le diverse tipologie di rete.

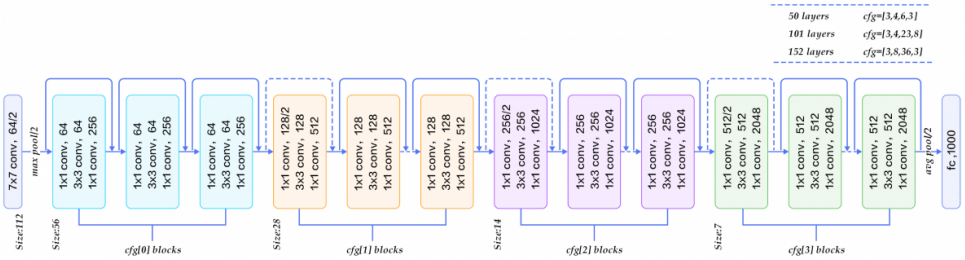


Figura 5.2. Esempio di architettura ResNet (Fonte: `https://andreaprovino.it/resnet/`).

InceptionResNet

Di *InceptionResNet* esistono versioni differenti; la prima versione, ovvero, l'*Inception-v1*, viene anche normalmente chiamata *GoogLeNet*. Successivamente, l'architettura della Inception si è evoluta portando all'introduzione della *batch normalization* e

[1] La ImageNet Large Scale Visual Recognition Challenge (ILSVRC) è una competizione indetta ogni anno dal 2015 dove programmi software vengono fatti competere per classificare e rilevare correttamente oggetti e scene contenuti nelle immagini.

[2] COCO è un set di dati utilizzato per la classificazione e segmentazione; esso contiene 1.5 milioni di istanze di oggetti.

prendendo nome di *Inception-v2*. Per i nostri test si è utilizzata la *InceptionResNetv2 (Inception-v2)*.

Inception-v2 è una rete neurale convoluzionale introdotta per la prima volta da [53] che propone una serie di miglioramenti volti a diminuire la complessità computazionale della prima versione della rete. Principalmente, per migliorare la rete, viene introdotta la *batch normalization*, che va a ridurre il fenomeno della dipendenza dei gradienti dalla scala dei parametri o dai loro valori iniziali, permettendo di utilizzare valori di *learning rate* più elevato.

L'*InceptionResNetv2*, nella sua architettura, amplia i banchi di filtri per evitare il collo di bottiglia; inoltre, trasforma le convoluzioni 5×5 in due convoluzioni 3×3. Questa scelta viene fatta perché, in questo modo, le convoluzioni risultano più efficienti facilitando la fase di training della rete. La Figura 5.3 ci mostra l'architettura dell'*Inception-v2* vista in maniera compressa.

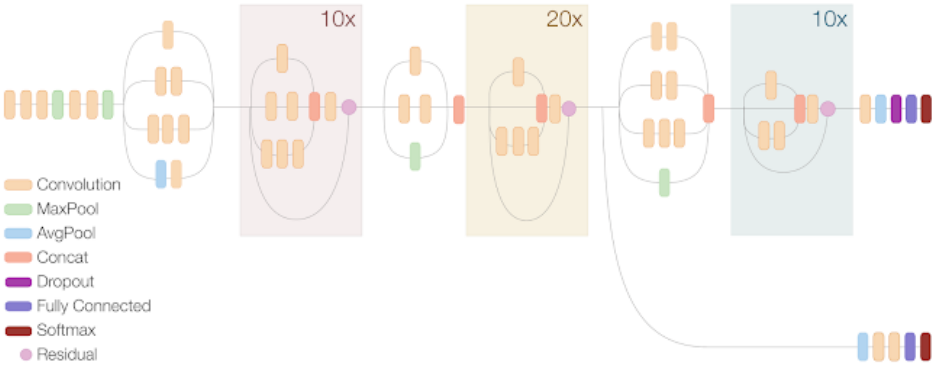


Figura 5.3. Vista compressa dell'architettura di una rete InceptionResNetv2 (Fonte: https://medium.com/@zahraelhamraoui1997/inceptionresnetv2-simple-introduction-9a2000edcdb6).

DenseNet

La *DenseNet (Dense Convolutional Network)* è una rete proposta per la prima volta nella competizione CVPR 2017 dove ha ottenuto il migliore risultato [34]. Essa ha l'obiettivo di ridurre il numero di parametri delle reti CNN attraverso l'utilizzo di connessioni dense.

Utilizzando delle normali reti convoluzionali, l'immagine di input passa per una serie di convolutional layer che permettono l'estrazione di feature. Nel caso delle DenseNet la rete propone che ogni livello convoluzionale concateni le proprie mappe di funzionalità ai livelli successivi. Dunque, come è possibile vedere in Figura 5.4, ogni livello riceve una conoscenza da parte dei livelli precedenti.

Il riutilizzo delle mappe caratteristiche da livelli precedenti permette di definire la rete in modo più compatto e, quindi, di utilizzare meno parametri.

La DenseNet è costituta da diversi Dense Block, ciascuno dei quali viene collegato ad un blocco di transizione. Per ogni Dense Block abbiamo un aumento del

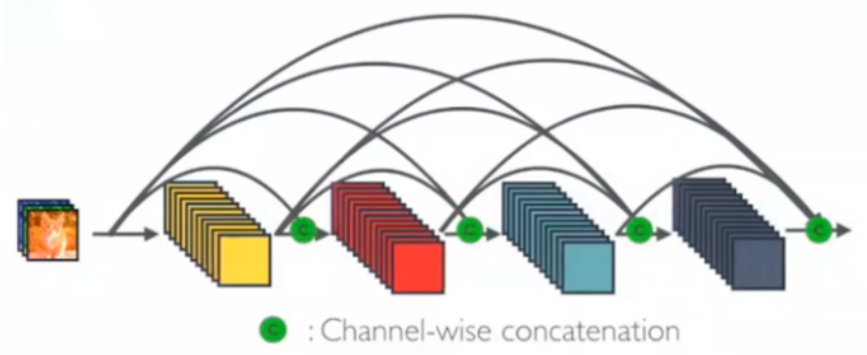


Figura 5.4. Esempio di blocco denso in una DenseNet (Fonte: `https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803`).

numero di canali pari a K. La Figura 5.5 mostra una DenseNet con soli tre blocchi densi.

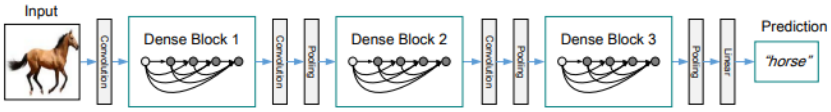


Figura 5.5. Esempio di rete DenseNet con tre blocchi densi (Fonte: Densely Connected Convolutional Networks [34]).

Nel caso delle DenseNet, in [34] vegono proposte differenti topologie di reti, quelle utilizzate per i nostri test sono la DenseNet169 e DenseNet201.

5.2 Test e analisi dei risultati

Per eseguire i test e per effettuare un confronto con le reti addestrate nel Capitolo 4, si è deciso di utilizzare il resampled dataset, già definito nella Sezione 4.3, utilizzando lo stesso schema di valutazione.

I test saranno eseguiti, sia con colore dominante (calcolato attraverso la liberia `ColorThief`) che senza colore dominante, facendo uso di tutte le reti presentante in precedenza. Ogni rete è stata implementata con il linguaggio *Python* e la libreria `Tensorflow`. Nel seguito mettiamo a confronto i risultati ottenuti:

- *ResNet50-v2* e *ResNet152-v2*: le reti sono state implementate utilizzando il modulo `tf.keras.applications.resnet_v2` che va ad integrare diverse tipologie di *ResNet*. I parametri per i test sono così impostati:
  - ottimizzatore utilizzato SGD;
  - learning rate pari a 0.1;
  - momentum pari a 0.001;

  - nella fase di traning, per evitare overfitting, è stata inserita una funzione di *Early Stopping*[3];
- *InceptionResNet-v2*: la rete è stata implementata utilizzando la libreria *Tensorflow* che integra in `tf.keras.applications.inception_resnet_v2` le varie reti *Inception-v2*. I parametri per i test sono così impostati:
  - ottimizzatore utilizzato SGD;
  - learning rate pari a 0.01;
  - momentum pari a 0.001;
  - nella fase di traning è stato inserita una funzione di *Early Stopping*.
- *DenseNet169-v2* e *DenseNet201-v2*: le reti sono totalmente implementate utilizzando `tf.keras.applications.densenet` che integra i modelli di diverse tipologie di *DenseNet*. I parametri per i test sono così impostati:
  - ottimizzatore utilizzato SGD;
  - learning rate pari a 0.1;
  - momentum pari a 0.001;
  - anche in questo caso, nella fase di training è stata inserita una funzione di *Early Stopping*.
- *CNN-differenze*: la rete è stata implementata come descritto nella Sezione 5.1.1. I parametri per i test sono così impostati:
  - ottimizzatore utilizzato ADAM;
  - learning rate pari a 0.001;
  - epsilon pari a $1e-07$;
  - anche in questo caso, nella fase di training, è stata inserita una funzione di *Early Stopping*.

Tutte le reti sono costituite da uno strato di input che prende immagini di dimensioni 100×100 con 36 canali.

5.2.1 Confronto risultati

La Tabella 5.1 mostra il confronto tra i vari test effettuati senza colore dominante, riportando una media dei valori di Precision, Accuracy, F1-Score e Recall. La media viene eseguita tra i 5 modelli addestrati per ogni singola rete ottenuti utilizzando la *k-cross fold validation*.

Si può notare che il migliore risultato è stato ottenuto utilizzando la rete *CNN-differenze*; essa si comporta molto meglio delle altre reti riuscendo ad ottenere un valore in accuratezza pari al $72,8\%$. Inoltre, la *Resnet50-v2* è l'unica, insieme alla *CNN-differenze*, a raggiungere un valore di accuratezza superiore al 70%.

Il secondo test eseguito si è svolto in maniera analoga al primo; in questo caso, però, abbiamo utilizzato il colore dominante. La Tabella 5.2 mostra i risultati ottenuti.

Possiamo vedere che i risultati ottenuti senza colore dominante risultano migliori di quelli presentati in Tabella 4.3 effettuati con *Color Deep*, ottenendo un valore di F1-score maggiore del 2%.

[3] L'*Early Stopping* è una tecnica largamente utilizzata in Deep Learning che evita l'overfitting durante l'addestramento.

Rete	Precision	Recall	F1-Score	Accuracy
CNN-differenze	0.752	0.724	0.716	0.728
Resnet50-v2	0.726	0.648	0.688	0.708
Resnet152-v2	0.66	0.646	0.622	0.646
InceptionResNet-v2	0.724	0.686	0.674	0.686
DenseNet169-v2	0.714	0.688	0.67	0.688
DenseNet201-v2	0.684	0.664	0.65	0.664

Tabella 5.1. Confronto senza colore dominante mediante il metodo delle differenze con le reti CNN-differenze, ResNet50-v2, ResNet152-v2, DenseNet169-v2, DenseNet201 e InceptionResNet-v2.

Rete	Precision	Recall	F1-Score	Accuracy
CNN-differenze	0.688	0.658	0.646	0.658
Resnet50-v2	0.554	0.548	0.516	0.548
Resnet152-v2	0.584	0.582	0.544	0.58
InceptionResNet-v2	0.592	0.586	0.552	0.584
DenseNet169-v2	0.578	0.552	0.526	0.552
DenseNet201-v2	0.592	0.586	0.552	0.584

Tabella 5.2. Confronto con colore dominante mediante il metodo delle differenze con le reti CNN-differenze, ResNet50-v2, ResNet152-v2, DenseNet169-v2, DenseNet201 e InceptionResNet-v2.

6

Modelli e analisi con Ensemble Network

In questo capitolo si introdurrà la tecnica di Ensemble Network specificando le possibili alternative di applicazione. Individuata la soluzione, si illustreranno i modelli che prendono parte all'ensemble mostrando e commentando i risultati ottenuti.

6.1 Ensemble Network

L'*Ensemble Network (EN)* è una tecnica largamente utilizzata nel *Deep Learning* che ha come obiettivo quello di combinare le prestazioni di differenti modelli per migliorare le performance.

Le reti neurali sono particolarmente flessibili e risultano sensibili alle specifiche dei dati di addestramento. Una stessa rete può generare pesi diversi, dunque modelli diversi, se addestrata con gli stessi dati [16].

Per evitare il problema della varianza possiamo combinare più reti neurali aggiungendo un bias che contrasta la varianza del modello. Dunque, attraverso l'utilizzo dell'ensemble generiamo delle predizioni meno sensibili ai dati di addestramento; inoltre, la composizione dei modelli manifestano prestazioni migliori rispetto alla singola rete.

Esistono diverse tipologie di ensemble network, che possono essere classificate come segue:

- *Bagging (Bootstrap aggregating)* - Il Bagging è una tecnica standard che viene largamente usata per generare algoritmi di ensemble. L'obiettivo è generare diversi modelli variando i dati. La tecnica genera una serie di campioni di addestramento differenti, con stessa dimensione e distribuzione dei dati originali. Dunque, esegue un ricampionamento casuale con rimpiazzo del dataset iniziale. Generati i vari modelli, questi vengono combinati per effettuare delle previsioni [28]. Approcci equivalenti a quelli appena descritti possono sfruttare anche la procedura di k-cross fold validation oppure eseguire suddivisioni holdout differenti del dataset originale (con random state iniziale diverso). In questo modo utilizziamo sottoinsiemi dei dati originali velocizzando la fase di addestramento dei modelli [16].

- *Boosting* - Il Boosting è una tecnica utilizzata per realizzare EN. Essa cerca di convertire un modello di apprendimento debole in un altro con migliore generalizzazione. A differenza del *Bagging* non si esegue una variazione sui dati, ma i modelli si apprendono in modo adattivo. Quindi, un modello appreso dipende dai precedenti. La tecnica non è molto impiegata con il *Deep Learning* poichè è molto complessa e computazionalmente svantaggiosa.
- *Stacking* - Lo stacking utilizza un algoritmo che riceve in input i dati restituiti in output da una serie di sottomodelli. Per capire meglio come funziona possiamo pensare al processo come uno stack a due livelli:
  - *Livello 0* : i dati di livello 0 sono gli input del set di dati di addestramento e i modelli di livello 0 imparano a fare previsioni da essi.
  - *Livello 1*: i dati di livello 1 prendono l'output dei modelli di livello 0 come input e il singolo modello di livello 1 impara a fare previsioni da questi dati [18].

  Approcci simili allo stacking sono quelli che utilizzano la media mobile; questo vuol dire che vari modelli vengono combinati attraverso il calcolo della media mobile per predire i risultati.
- *Hyperparameter Tuning Ensemble* - Questa tecnica sfrutta la natura stocastica delle reti addestrando diversi modelli sullo stesso insieme di dati. Variando gli iperparametri della rete, quali ottimizzatori, learning rate, batch size e numero di epoche oppure utilizzando variazioni della stessa rete, ad esempio, *kernel regularization*[1] differenti, possiamo ottenere diversi modelli che hanno appreso in modo diverso come mappare gli input agli output. Concatenando questi modelli otteniamo una EN in grado di generalizzare meglio il problema [16]. Una soluzione alternativa prevede di usare reti completamente differenti combinando insieme i modelli.

In letteratura sono presenti ulteriori metodi che non verranno trattati nella tesi; per approfondimenti si rimanda a [16] e [28].

6.1.1 Implementazione Ensemble per CNN

Nel Capitolo 5 abbiamo visto come la rete *CNN-Differenze* abbia ottenuto i migliori risultati nel riconoscimento del colore; per cercare di migliorare ulteriormente le sue prestazioni e la sua capacità di generalizzazione implementeremo l'EN. Tra le soluzioni precedentemente descritte si utilizza la tecnica di *Hyperparameter Tuning Ensemble* addestrando la rete con iperparametri differenti e ottenendo tre diversi modelli. Inoltre, si è deciso di eseguire anche una variazione dei dati di addestramento, utilizzando, anche in questo caso, il resempled dataset descritto nella Sezione 4.1.2. Infatti, invece di effettuare una *k-fold cross validation* abbiamo eseguito l'*holdout*.

L'obiettivo prefissato è quello di poter addestrare le reti su tutto il set di dati, evitando, quindi, ulteriori suddivisioni dal momento che il dataset è di piccole dimensioni. Volendo, però, fornire una maggiore variazione tra i vari modelli, si è

[1] I regolarizzatori consentono di applicare penalità sui parametri o sull'attività del livello durante l'ottimizzazione. Tali penalità sono sommate nella funzione di perdita che la rete ottimizza [7].

deciso, attraverso l'*holdout*, di utilizzare differenti `random state` in modo che ogni singolo modello sia addestrato con dati pseudo-differenti[2]. Variando, infatti, il random seed nella fase di split e shuffle del dataset, generiamo train e convolution set diversi per ciascun modello addestrato.

Per quanto concerne la fase di composizione dei modelli, essa è stata effettuata utilizzando la funzione `argmax`. In questo modo nel momento in cui effettuiamo delle previsioni, prendiamo la classe che è stata più frequentemente predetta tra i 3 modelli. Questo tipo di soluzione è stata già discussa nella Sezione 2.2.2 in cui veniva utilizzata per combinare i diversi classificatori SVM. Il Listato 6.1 mostra come è stata implementata, nel linguaggio *Python* e attraverso le librerie `Tensorflow` e `Numpy`, la composizione.

```
1  #Result
2  results = np.zeros( (X_test.shape[0],12) )
3  for model in model_ens:
4      results = results + model.predict(X_test)
5
6  results = np.argmax(results,axis = 1)
```

Listato 6.1. Codice di implementazione della composizione con `argmax` dei modelli

Nel Listato 6.1, `model_ens` indica la lista dei modelli addestrati.

6.2 Analisi dei modelli

Le tre reti utilizzate per formare l'EN sono cosi definite:

- Rete1 - Questa rete è stata definita in maniera analoga a quella descritta nella Sezione 5.1.1. I parametri per i test sono così definiti:
  - ottimizzatore ADAM;
  - learning rate pari a 0.001;
  - epsilon pari a $1e-07$;
  - addestramento effettuato con *Early Stopping* per evitare l'*overfitting*.
- Rete2 - Anche questa rete è stata definita in maniera analoga a quella descritta in Sezione 5.1.1. I parametri utilizzati nella fase di addestramento sono così definiti:
  - ottimizzatore ADAM;
  - learning rate pari a 0.0001;
  - epsilon pari a $1e-08$;
  - addestramento con *Early Stopping*.

  La scelta di utilizzare un epsilon minore è descritta in [35]; il documento introduce l'ottimizzatore *Adam* consigliando di impostare in questo modo il parametro di apprendimento per applicazioni di *computer vision*.

[2] Definiamo "pseudo-differente" la suddivisione tra i vari modelli perché non abbiamo certezza matematica che i dati di train e convalidation, dopo la suddivisione, siano effettivamente differenti per ogni singolo modello. Infatti la procedura di split utilizza una funzione di shuffle pseudo-casuale.

- Rete3 - La terza rete addestrata aggiunge un kernel regularizer L2 (per maggiori dettagli vedere [22]) nello strato dense costituito da 128 neuroni (Sezione 5.1.1). I parametri di apprendimento sono così impostati:
  - ottimizzatore SGD;
  - learning rate pari a 0.01;
  - utilizzo del *Nesterov momentum*[3] momentum;
  - addestramento con *Early Stopping*.

I modelli sono tutti stati addestrati e testati utilizzando il resampled dataset (Sezione 4.3). Utilizza lo stesso test set ma, come spiegato in precedenza, diversi set di training set e convalidation set, i quali risultano suddivisi in maniera casuale per ogni rete attraverso l'holdout 80/20.

La Tabella 6.1 mostra i risultati medi ottenuti dalle singole reti.

Rete	**Precision**	**Recall**	**F1-Score**	**Accuracy**
Rete1	0.77	0.76	0.76	0.76
Rete2	0.74	0.75	0.74	0.75
Rete3	0.73	0.73	0.72	0.73

Tabella 6.1. Risultati ottenuti dalle singole reti.

Il modello migliore è stato generato dalla Rete1 raggiungendo un valore di accuratezza pari al 76%.

6.2.1 Risultati Ensemble Network

Componendo i modelli come descritto nella Sezione 6.1.1, e utilizzando lo stesso test set otteniamo il *classification report* in Tabella 6.2.

Dai risultati ottenuti possiamo notare come il modello ottiene delle performance migliori utilizzando l'EN, aumentando l'accuratezza del 4% rispetto al modello generato dalla Rete1 (il migliore nei test effettuati in precedenza). Inoltre, la rete è molto più precisa e con una sensibilità più elevata, rispettivamente, del 79% e 80%.

Il vero problema di tutte le prove effettuate è dato dal riconoscimento del colore Viola. Guardando la matrice di confusione in Figura 6.1 notiamo che i tessuti di questa classe vengono scambiati per Ciano o Blu Oltremare. Questo problema è, probabilmente, legato al fatto che la rete non ha raggiunto una buona capacità nel riconoscere colori tra loro vicini o simili. Inoltre nel dataset sono presenti elementi di colore Viola che hanno trame più chiare rispetto al Viola di riferimento, e questo tende a rendere complicato il riconoscimento. La stessa problematica la si può riscontrare tra i colori Rosso, Magenta e Terra.

I risultati ottenuti con l'EN sono molto positivi; questo dimostra che il metodo delle differenze di colore può essere ulteriormente migliorato.

[3] Il Nesterov momentum, anche chiamato Gradiente accellerato di Nesterov, è un'estensione dell'algoritmo di ottimizzazione della discesa del gradiente. Attraverso di esso l'aggiornamento dei parametri non avviene rispetto al valore corrente, ma considerando la previsione della posizione futura. Per maggiori dettagli vedere [45].

Classe	Precision	Recall	F1-Score
Arancione	0.95	1.00	0.98
Bianco	0.90	0.95	0.90
Blu	0.74	1.00	0.885
Ciano	0.65	0.65	0.65
Giallo	0.70	0.70	0.70
Magenta	0.95	1.00	0.98
Nero	0.95	0.95	0.95
Rosso	0.85	0.85	0.85
Terra	0.75	0.60	0.67
Verde	0.76	0.65	0.70
Verde Smeraldo	0.74	0.85	0.79
Viola	0.54	0.35	0.42
		Accuracy	**0.80**
Media	**0.79**	**0.80**	**0.79**

Tabella 6.2. Report Ensemble CNN-Differenze.

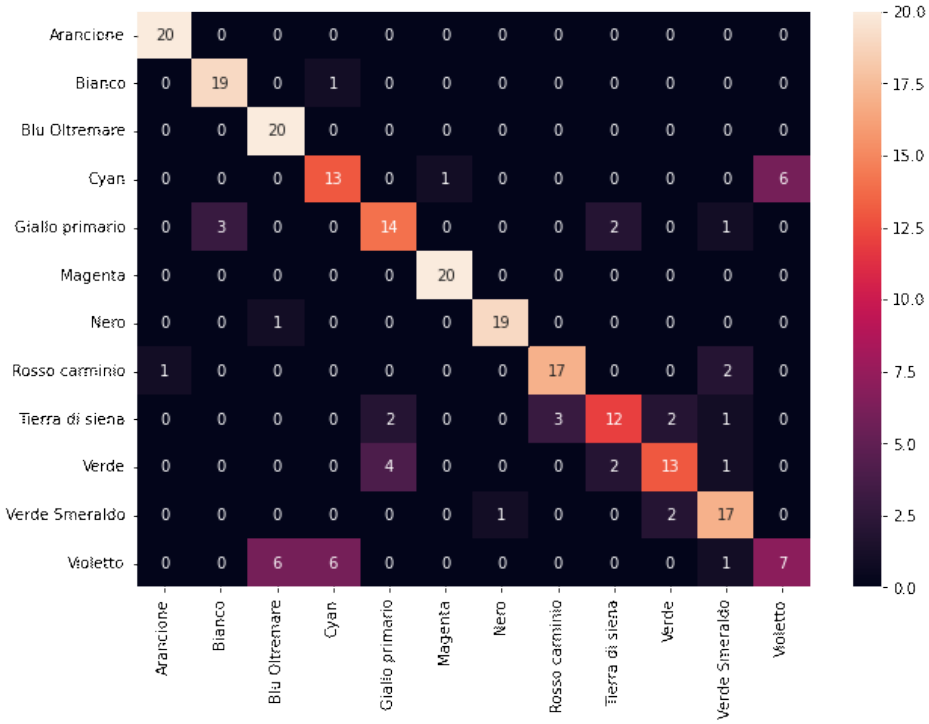


Figura 6.1. Matrice di confusione del test effettuato con l'Ensemble CNN-Differenze.

7

Letteratura correlata

Nel capitolo corrente verrà illustrato lo stato dell'arte della letteratura sul riconoscimento del colore.

7.1 Riconoscimento del colore dei veicoli

In letteratura non sono presenti lavori analoghi a quelli proposti nella tesi, nel quale ci si pone l'obiettivo del riconoscimento del colore per immagini di tessuti.

Studi simili si pongono l'obiettivo del riconoscimento del colore di veicoli come visto nel Capitolo 4, dove, per i nostri test, si è utilizzata una rete realizzata per questo scopo. Infatti, per quanto concerne quest'ultimo la letteratura fa largo utilizzo di tecniche di ML, definendo e studiando diverse tipologie di soluzione.

7.1.1 Uno sguardo al riconoscimento del colore dei veicoli tramite Deep Learning

In [42], come già visto nella Sezione 2.3.2, si fa utilizzo della CNN per il riconoscimento del colore dei veicoli, presentando la rete che abbiamo chiamato *Color Deep*, già ampiamente illustrata nel Capitolo 4. Il Dataset utilizzato è quello proposto in [36], con un numero di classi pari a 8 così definite: Giallo, Bianco, Blu, Ciano, Rosso, Grigio, Nero e Verde. I risultati ottenuti con questa soluzione sono molto positivi, raggiungendo un valore di accuratezza pari al $94,47\%$.

Lo studio, inoltre, si fa carico di capire quale sia il migliore *Color Space* per individuare il colore utilizzando la CNN proposta. La Figura 7.1 mostra i risultati ottenuti, da Reza Fuad Rachmadi e I Ketut Eddy Purnama in [42], utilizzando diversi spazi di colore (RGB, HSV, Lab e CIE XYZ).

I risultati migliori sono stati ottenuti con lo spazio di colore RGB. Questo risultato è particolarmente interessante; infatti, sappiamo che lo spazio di colore LAB risulta, per definizione, essere il più percettivamente uniforme (Capitolo 1) ma, dai risultati, la rete mostra un comportamento migliore con lo spazio RGB.

Un ulteriore approccio viene definito da Hsieh et al. [33], che propongono di applicare la *color correction* alle immagini dei veicoli. Inoltre, per un migliore riconoscimento essi eliminano i vetri dell'auto utilizzando solo la parte inferiore del

	Color Space				Chen et al. [2]
Color Class	RGB	HSV	CIE Lab	CIE XYZ	
yellow	0.9794	0.9450	0.9656	**0.9828**	0.9553
white	**0.9666**	0.9624	0.9561	0.9649	0.9423
blue	0.9410	**0.9576**	0.9410	0.9484	0.9535
cyan	0.9645	0.9716	0.9645	0.9716	**0.9787**
red	**0.9897**	0.9866	**0.9897**	0.9886	0.9878
gray	0.8608	0.8503	**0.8668**	0.8647	0.8466
black	**0.9738**	0.9703	0.9703	0.9709	0.9730
green	**0.8257**	0.8215	0.8215	0.7676	0.7884
average	**0.9447**	0.9372	0.9414	0.9432	0.9282

Figura 7.1. Accuratezza ottenuta addestrando la rete con 4 tipologie diverse di spazio di colori (Fonte: Vehicle Color Recognition using Convolutional Neural Network [42]).

veicolo per effettuare la classificazione (tale procedura viene chiamata segmentazione). La *color correction* viene descritta come fondamentale per avere tutte le immagini dei veicoli con analoghe condizioni di luce,. Per farlo vengono adottati delle tecniche che mappano nello spazio colore LAB un pixel dell'immagine di riferimento in un pixel nell'immagine sorgente. I classificatori utilizzati sono tre: un G-Classifier, un Classificatore DC e un Classificatore DG. Ogni classificatore riconosce un determinato colore: il G-Classifier si occupa di riconoscere il colore grigio, mentre il classificatore DC e il classificatore DG sono addestrati per il riconoscimento dei colori Rosso, Verde, Blu, Giallo, Bianco e Nero. I risultati ottenuti prima della *color correction* e della segmentazione froniscono un valore di accuratezza medio di 86.58%, mentre dopo la correzione e segmentazione si raggiunge il 92.59%. Inoltre, per determinate classi si raggiunge una precisione pari al 100%.

In [20] viene proposta una soluzione differente, già trattata nel Capitolo 2. In questo caso il lavoro si concentra su come produrre delle feature a partire da determinate regioni di interesse (ROI) individuate nelle immagini. La classificazione viene, anche in questo caso, eseguita attraverso l'utilizzo dell'SVM, ed i test effettuati adottano diverse tipologie di preprocessing come PCA, CAC e PCA + CAC. Il valore di accuratezza più elevato è stato ottenuto con preprocessing PCA + CAC ed è pari a 92.49%.

Son e altri [51] utilizza approcci simili al precedente. In questo caso, per l'estrazione delle feature, si utilizza un Grid Kernel; quest'ultimo calcola una somiglianza tra una coppia di immagini utilizzando funzionalità estratte implicitamente. La feature è definita come un frammento dell'immagine. Prima di estrarre i frammenti, l'approccio adottato da Son e altri, converte l'immagine in uno spazio colore HSV. La conversione viene effettuata per applicare la *color correction* all'immagine, aggiustando i valori di H e S. La Figura 7.2 ci mostra come è stato generato il modello. Possiamo notare che, nella fase di addestramento, il Grid Kernel svolge il ruolo di

confronto nella SVM. Il kernel, per confrontare due immagini, utilizza i valori di H e S precedentemente corretti.

Per quanto concerne la fase di classificazione, il processo viene descritto dalla Figura 7.3 . Dagli esperimenti effettuati con questo metodo si è raggiunto un valore di precision e recall molto elevato, vicino al 100%.

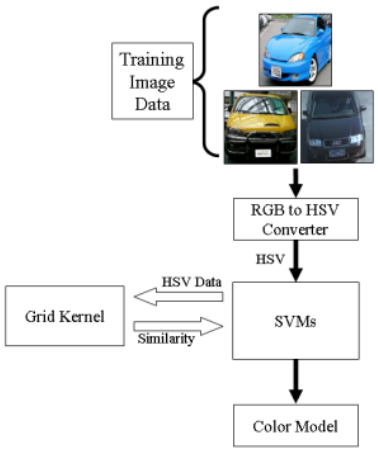


Figura 7.2. Processo di generazione del modello in [51] (Fonte: A Convolution Kernel Method for Color Recognition [51]).

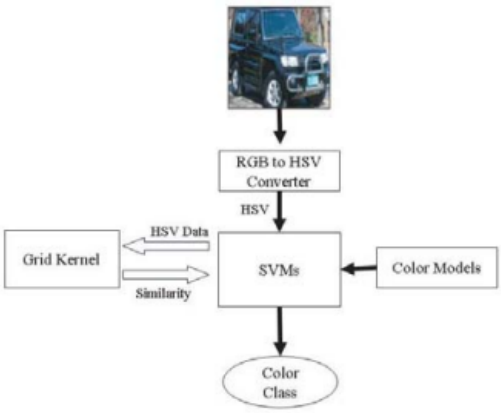


Figura 7.3. Processo di classificazione in [51] (Fonte: A Convolution Kernel Method for Color Recognition [51]).

7.2 Riconoscimento del colore mediante intensità dei pixel

Janya Sainui e Paiboon Pattanasatean presentano in [46] una soluzione al riconoscimento del colore basandosi sull'intensità dei pixel. Questa soluzione, a differenza delle precedenti, non ha come obiettivo quello di riconoscere il colore di un particolare oggetto, ma va ad individuare il colore di cose che ci circondano. Gli autori propongono, anche, una possibile applicazione per smartphone che può essere utilizzata da persone con problemi di daltonismo per capire meglio i colori che stanno fotografando. La Figura 7.4 mostra il funzionamento dell'app.

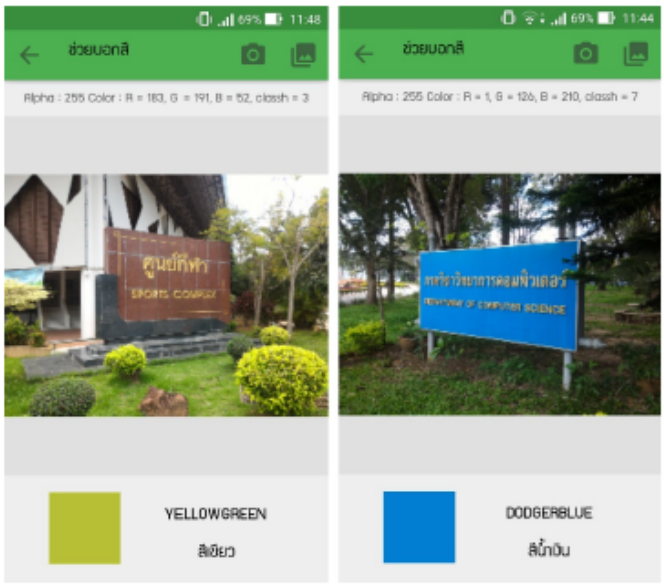


Figura 7.4. Applicazione per il riconoscimento del colore [46] (Fonte: Color Classification based on Pixel Intensity Values [46]).

Daremo, ora, una breve spiegazione sul funzionamento della classificazione del colore in base ai valori di intensità dei pixel. Janya Sainui e Paiboon Pattanasatean utilizzano lo spazio di colore HSV per il riconoscimento; il metodo è descritto in pseudocodice dall'algoritmo nel Listato 7.1.

```
1   Input: (xi, yi) and z
2   Output: a color name of z
3   minDistance <- inf
4   compute hue value of z, representing as hz.
5   compute no. of bins for hz.
6   for i 1 to n do
7       compute hue value of xi, representing as hx.
8       compute no. of bins for hx.
9       if (hz and hx are in the same bin) then
10          distance = xi - z
11          if (distance < minDistance) then
12              minDistance <- distance
13              colorNome <- yi
14              end if
```

```
15        end if
16    end for
17    Return colorName
```

Listato 7.1. Pseudo codice dell'algoritmo per la classificazione del colore basato sull'intesità dei pixel [46]

All'algoritmo vengono passati in input una serie di colori x_i, che rappresentano i riferimenti e il colore da riconoscere z_i. Quest'ultimo viene convertito in HSV ottenendo il valore di tonalità h_z. Successivamente, viene calcolato il numero di bin, attraverso la Tabella 7.1, in base al valore normalizzato di h_z.

Numero di bin	Valore di tonalità
1	$0 - 0.1$
2	$0.1 - 0.2$
3	$0.2 - 0.3$
4	$0.3 - 0.4$
5	$0.4 - 0.5$
6	$0.5 - 0.6$
7	$0.6 - 0.7$
8	$0.7 - 0.8$
9	$0.8 - 0.9$
10	$0.9 - 1.0$

Tabella 7.1. Valore della tonalità per ogni bin.

Per ogni colore di riferimento viene calcolata la sua tonalità normalizzata h_x e viene calcolato il suo bin. Se h_z e h_x si trovano nel medesimo bin, viene calcolata la distanza RGB tra i due colori (una possibile formula è stata introdotta in Sezione 1.3.1). Se la distanza è minore rispetto a quella calcolata all'iterazione precedente, viene aggiornato il valore di `minDistance`. La distanza minima individua il nome del colore z.

Questa soluzione è molto simile a quella introdotta nella Sezione 2.1; in questo caso, però, otteniamo una soluzione computazionalmente meno complessa, grazie all'utilizzo della tonalità per filtrare quali sono i possibili candidati tra i colori di riferimento.

8

Conclusioni e uno sguardo al futuro

Il capitolo trae delle conclusioni riguardanti il lavoro svolto. Inoltre, verranno esposti i possibili sviluppi futuri per migliorare i risultati ottenuti.

8.1 Conclusioni

Il presente elaborato è nato dall'intenzione di individuare la direzione da prendere per la risoluzione del problema del riconoscimento del colore in immagini di tessuti. Quanto mostrato nei Capitoli 4, 5 e 6 fa intendere che la direzione migliore è quella di utilizzare il metodo delle differenze.

Il migliore risultato ottenuto è quello che fa utilizzo della tecnica di ensemble, ottenendo l'80% di accuratezza. I modelli che prendono parte all'EN sono tutti addestrati senza l'utilizzo del colore dominante (Sezione 6.2.1). Quanto ottenuto, però, risulta sicuramente in contraddizione con quello che la teoria suggerisce; infatti, per la rete, il riconoscimento del colore utilizzando immagini con texture e differenti illuminazioni è sicuramente più complesso rispetto al singolo colore omogeneo.

Il motivo per il quale otteniamo risultati migliori è dovuto, come già illustrato nella Sezione 5.2.1, al fatto che l'estrazione del colore dominante ci individua un colore non corretto. Il medesimo problema lo abbiamo nel momento in cui si va ad utilizzare la rete *Color Deep* (Sezione 4.3.1), anche in questo caso otteniamo risultati peggiori e dal confronto effettuato notiamo che con le texture (senza colore dominante) la rete riesce ad ottenere un valore di accuratezza maggiore del 9% rispetto all'utilizzo del colore dominante.

Inoltre, nella Sezione 5.2.1, mostriamo il confronto tra le reti *ResNet*, *DenseNet* e *InceptionResNet*. Le performance di queste reti risultano di gran lunga peggiori rispetto alla rete *CNN-Differenze* introdotta in questa tesi. Tale risultato è dovuto al numero di dati del nostro dataset, ovvero 1200, troppo piccolo per addestrare reti con elevato numero di neuroni come quelle che sono state proposte nella Sezione 5.1.2.

Da quanto illustrato nell'elaborato possiamo concludere che il metodo delle differenze è il più efficace per il riconoscimento del colore e la rete migliore è la *CNN-Differenze*. Inoltre, dai test eseguiti risulta che il *Deep Learning* è la soluzione adatta per risolvere il task di apprendimento prefissato.

8.2 Sviluppi futuri

Il lavoro eseguito può considerasi un punto di partenza per il riconoscimento del colore in immagini di tessuti. Per migliorare i risultati mostrati il primo passo potrebbe essere quello di ampliare il dataset, aggiungendo immagini in condizioni di luce differenti. Un punto fondamentale è l'aggiunta di immagini parzialmente in ombra questo permette di realizzare reti maggiormente robuste in grado di rispecchiare gli ambienti di acquisizione reale.

L'incremento del dataset potrebbe portare a rivalutare i modelli generati con reti maggiormente complesse (Capitolo 4), in grado di ottenere una generalizzazione migliore. Sfruttando tali reti è possibile sicuramente migliorare i risultati discussi nel Capitolo 6 adottando tecniche di EN più sofisticate e variando completamente le reti che compongono l'insieme.

Per quanto concerne l'utilizzo del colore dominante si possono valutare soluzioni di miglioramento dell'immagine o tecniche più sofisticate, che permettono di individuare i colori corretti attraverso una selezione. Abbiamo, infatti, visto in letteratura che applicare tecniche di *color correction* (Capitolo 7) porta ad un miglioramento nell'individuazione del colore. Invece, individuare delle zone di maggiore interesse del tessuto fotografato potrebbe aiutare l'algoritmo di selezione del colore dominante a valutare meglio il colore.

La tesi si concentra solo su tecniche di Deep Learning, ma come visto nel Capitolo 7, in letteratura sono presenti altre soluzioni di ML particolarmente efficaci per il riconoscimento del colore. Pertanto, si potrebbe valutare la risoluzione del problema attraverso l'uso di tecniche classiche per l'apprendimento automatico.

Riferimenti bibliografici

1. Albedo. https://it.wikipedia.org/wiki/Albedo, Visualizzato il 09/2021.
2. Algoritmo Support Vector Machine. https://www.lorenzogovoni.com/support-vector-machine/, Visualizzato il 09/2021.
3. Color difference. https://en.wikipedia.org/wiki/Color_difference, Visualizzato il 09/2021.
4. Color Names. https://github.com/codebrainz/color-names, Visualizzato il 09/2021.
5. Funzione test (ottimizzazione). https://it.wikipedia.org/wiki/Funzione_test_(ottimizzazione), Visualizzato il 09/2021.
6. ImageNet. https://it.wikipedia.org/wiki/ImageNet, Visualizzato il 09/2021.
7. Layer weight regularizers. https://keras.io/api/layers/regularizers/, Visualizzato il 09/2021.
8. Neural Networks, Multilayer Perceptron and the Backpropagation Algorithm. https://medium.com/@tiago.tmleite/neural-networks-multilayer-perceptron-and-the-backpropagation-algorithm-a5cd5b904fde, Visualizzato il 09/2021.
9. Spazio dei colori. https://it.wikipedia.org/wiki/Spazio_dei_colori, Visualizzato il 09/2021.
10. Transfer Learning. https://cs231n.github.io/transfer-learning/, Visualizzato il 09/2021.
11. YCbCr. https://it.wikipedia.org/wiki/YCbCr, Visualizzato il 09/2021.
12. Mahmoud Afifi. Semantic white balance: Semantic color constancy using convolutional neural network, 2019.
13. Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.
14. D. Bloomberg and Leptonica. Color quantization using modified median cut. 2008.
15. Jason Brownlee. A Gentle Introduction to Transfer Learning for Deep Learning. https://machinelearningmastery.com/transfer-learning-for-deep-learning/, Visualizzato il 09/2021.
16. Jason Brownlee. Ensemble Learning Methods for Deep Learning Neural Networkss. https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/, Visualizzato il 09/2021.
17. Jason Brownlee. How to Configure Image Data Augmentation in Keras. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/, Visualizzato il 09/2021.

18. Jason Brownlee. Stacking Ensemble for Deep Learning Neural Networks in Python. https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/, Visualizzato il 09/2021.
19. Carl Burch. A survey of machine learning. 01 2002.
20. Pan Chen, Xiang Bai, and Wenyu Liu. Vehicle color recognition on urban road by feature context. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):2340–2346, 2014.
21. Abhijeet Ghosh Sotiris Malassiotis Christos Kampouris, Stefanos Zafeiriou. Fine-grained material classification using micro-geometry and reflectance. In *ECCV 2016*, October 2016.
22. Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels, 2012.
23. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.
24. Michele Diodati. Mdelli di rappresentazione del colore. https://medium.com/michele-diodati/modelli-di-rappresentazione-del-colore-a5fa5cd003bf, Visualizzato il 09/2021.
25. Shi Dong, Ping Wang, and Khushnood Abbas. A survey on deep learning and its applications. *Computer Science Review*, 40:100379, 2021.
26. Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.
27. Graham Finlayson, Michal Mackiewicz, and Anya Hurlbert. Color correction using root-polynomial regression. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 24:1460–1470, 05 2015.
28. M. A. Ganaie, Minghui Hu, M. Tanveer*, and P. N. Suganthan*. Ensemble deep learning: A review, 2021.
29. Elia Giacomini. *Convolutional Neural Network per il riconoscimento di nudità nelle immagini*. PhD thesis, Dipartimento di Scienze Statistiche/Università degli Studi di Padova, 2016/2017.
30. Ram Deepak Gottapu and Cihan H Dagli. Densenet for anatomical brain segmentation. *Procedia Computer Science*, 140:179–185, 2018. Cyber Physical Systems and Deep Learning Chicago, Illinois November 5-7, 2018.
31. Thanh H. Ha, Chyuan-Tyng Wu, Peter Majewicz, Kurt R. Bengtson, and Jan P. Allebach. Glare and shadow reduction for desktop digital camera capture systems. In Reiner Eschbach, Gabriel G. Marcu, and Alessandro Rizzi, editors, *Color Imaging XVIII: Displaying, Processing, Hardcopy, and Applications*, volume 8652, pages 35 – 41. International Society for Optics and Photonics, SPIE, 2013.
32. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
33. Jun-Wei Hsieh, Li-Chih Chen, Sin-Yu Chen, Duan-Yu Chen, Salah Alghyaline, and Hui-Fen Chiang. Vehicle color classification under different lighting conditions through color correction. *IEEE Sensors Journal*, 15(2):971–983, 2015.
34. Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
35. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
36. Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.
37. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

38. Zewen Li, Wenjie Yang, Shouheng Peng, and Fan Liu. A survey of convolutional neural networks: Analysis, applications, and prospects, 2020.
39. Massimiliano Morrelli. Addestramento con dataset sbilanciati, 2020.
40. Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.
41. Andrea Provino. ResNet CNN Networks. https://andreaprovino.it/resnet/, Visualizzato il 09/2021.
42. Reza Fuad Rachmadi and I. Ketut Eddy Purnama. Vehicle color recognition using convolutional neural network. *CoRR*, abs/1510.07391, 2015.
43. A. Sai Bharadwaj Reddy and D. Sujitha Juliet. Transfer learning with resnet-50 for malaria cell-image classification. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 0945–0949, 2019.
44. Qiubing Ren, Mingchao Li, and Shuai Han. Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives. *Big Earth Data*, 3:1–18, 02 2019.
45. Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.
46. Janya Sainui and Paiboon Pattanasatean. Color classification based on pixel intensity values. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 302–306, 2018.
47. Tushar Sandhan and Jin Young Choi. Anti-glare: Tightly constrained optimization for eyeglass reflection removal. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
48. Yuita Arum Sari, R. V. Hari Ginardi, and Nanik Suciati. Color correction using improved linear regression algorithm. In *2015 International Conference on Information Communication Technology and Systems (ICTS)*, pages 73–78, 2015.
49. Khoshgoftaar Taghi M. Shorten, Connor. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6, 07 2019.
50. Oleksii Sidorov. Conditional gans for multi-illuminant color constancy: Revolution or yet another approach?, 2019.
51. Jeong-Woo Son, Seong-Bae Park, and Ku-Jin Kim. A convolution kernel method for color recognition. In *Sixth International Conference on Advanced Language Processing and Web Information Technology (ALPIT 2007)*, pages 242–247, 2007.
52. Kenji Suzuki. *Artificial Neural Networks - Methodological Advances and Biomedical Applications*. 04 2011.
53. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.
54. Shiting Ye, Jia-Li Yin, Bo-Hao Chen, Dewang Chen, and Yunbing Wu. Single image glare removal using deep convolutional networks. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 201–205, 2020.
55. Mayaluri zefree lazarus and Supratim Gupta. Spectacle problem removal from facial images based on detail preserving filtering schemes. *Journal of Intelligent Fuzzy Systems*, 36:1–11, 01 2019.
56. Yongli Zhang. Support vector machine classification algorithm and its application. In Chunfeng Liu, Leizhen Wang, and Aimin Yang, editors, *Information Computing and Applications*, pages 179–186, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Ringraziamenti

Il primo ringraziamento va alla mia famiglia che mi è sempre stata accanto dandomi la possibilità, per nulla scontata, di intraprendere e portare a termine questo percorso universitario.

Ringrazio di cuore la mia ragazza Michela. La nostra relazione è iniziata in concomitanza con il mio percorso universitario e oggi lo concludo avendo sempre lei al mio fianco. Compagna di tutto e migliore amica, lei è la persona su cui posso fare più affidamento, parte del risultato è anche il suo.

Ringrazio il prof. Domenico Ursino, per aver accettato di essere il mio relatore, per la grande pazienza avuta nel mio percorso di tirocinio fatto di alti e bassi e per la straordinaria disponibilità dimostrata nel corso della stesura della tesi.

Ulteriore ringraziamento, va alla correlatrice prof.ssa Alessia Amelio che durante tutto il periodo di tirocinio mi ha supportato nella decisione e nella scelta dei diversi esperimenti eseguiti. La sua enorme esperienza mi ha permesso di riuscire a portare a termine il lavoro presentato.

Ringrazio di cuore AESYS e tutte le persone che ne fanno parte, sono una grande famiglia che mi ha fatto sentire super integrato nel progetto. In particolare vorrei ringraziare il mio tutor Gregorio e il team leader Denny che mi hanno seguito durante lo svolgimento del tirocinio.

Come dimenticarsi degli amici universitari e gli amici di sempre, i loro nomi saranno per tutta la vita nella mia mente e nei ricordi che ci accomunano, grazie per avermi donato tanti momenti di svago.

Infine, un caloroso ringraziamento va a tutte le persone che hanno sempre creduto in me dimostrandomi giorno per giorno che ciò che sembrava impossibile in realtà non lo era.