Simone Barbaro

Analisi di tecniche di preprocessing per la classificazione di coralli con deep learing

5 luglio 2019

Autore: Simone Barbaro

Matricola: 0000789448

Relatore: Dr.ssa Alessandra Lumini

Titolo: Analisi di tecniche di preprocessing per la classificazione di coralli con deep learing

Abstract: Si vuole comprendere quanto le operazione di estrazione di feature basate su colore e tessitura possano essere utili nella classificazione dei coralli. Quindi vengono esplorati gli effetti che le operazioni di preprocessing di cambiamento dello spazio di colore in HSV e CIELAB, local binary pattern, Wavelet e tiling abbiano sui modelli ResNet e DenseNet addestrati nei dataset EILAT e RSMAS. I risultati vengono riportati e confrontati attraverso diverse misure, alcune conclusioni vengono tratte da questi risultati e si espande su possibili ricerche future.

Keywords: Classificazione coralli, machine learning, deep learning, preprocessing, estrazione feature

Elenco delle figure

Elenco delle tabelle

Indice

Introduzione

In questa tesi viene affrontato il problema della classificazione automatica di coralli attraverso estrazione di feature e modelli di deep learning.

Il deep learning si basa su reti profonde che vengono addestrate per apprendere in maniera autonoma le caratteristiche delle immagini da elaborare e i pesi da usare per la classificazione. Per questo motivo in genere non richiedono alcun preprocessing dell'immagine in input. Lo studio di tecniche di preprocessing e estrattori di feature è stato, però, per anni uno degli obiettivi principali del machine learning ed ha portato molti successi.

L'obiettivo della tesi, quindi, è quello di esplorare come combinare il meglio di deep learning e preprocessing nell'ambito della classificazione dei coralli.

Il problema è rilevante per la ricerca nel cambiamento climatico (Burke et al. 2011) perché gli esperti ambientali hanno bisogno di strumenti automatici per aiutarli ad annotare le immagini raccolte sulle barriere coralline (Mahmood et al. 2017). Inoltre è importante per combattere i danni umani alle barriere coralline, permettendo di costruire veicoli autonomi che eseguano azioni di ripopolazione delle colonie di coralli (Elawady 2015).

La tesi si pone come obiettivo di riportare e confrontare le performance ottenute da modelli di successo basati su reti neurali (Gómez-Ríos et al. 2019), nello specifico ResNet e DenseNet, combinandoli con alcune trasformazioni che hanno avuto successo in studi precedenti (Elawady 2015), in particolare il cambiamento dello spazio di colore in HSV e CIELAB; local binary pattern; Wavelet e tiling.

Lo scopo principale non solo è quello di ottenere un miglioramento delle prestazioni rispetto allo stato dell'arte, ma di identificare trasformazioni che possano essere utilizzate con successo assieme a modelli di deep learning in future ricerche sulla classificazione di coralli. Quindi anche performance vicine a quelle ottenibili senza preprocessing sono considerate un successo.

Per raggiungere questo obiettivo sono stati scelti due dataset che contengono immagini di specie di coralli, EILAT e RSMAS (Shihavuddin 2017), e gli iperparametri che hanno avuto più successo su questi dataset (Gómez-Ríos et al. 2019). I modelli sono stati addestrati su questi dataset a partire dai migliori iperparametri e sono state sperimentate diverse combinazioni

di trasformazioni di preprocessing. Infine sono stati usati diversi indicatori di performance per interpretare i risultati dell'addestramento, in particolare accuratezza, F_1 score e matrice di confusione.

I risultati riportati hanno rivelato che l'operazione di tiling ottiene risultati migliori con consistenza, perfino superando l'accuratezza del metodo migliore in (Gómez-Ríos et al. 2019). Inoltre è stato scoperto che il cambio dello spazio di colori può essere utile se le condizioni delle immagini sono giuste. Infine la trasformata Wavelet ha ottenuto risultati migliori del local binary pattern ma non ha mostrato chiari vantaggi rispetto alle immagini non modificate.

Questa tesi è suddivisa nel seguente modo:

- Il primo capitolo descrive il problema della classificazione dei coralli, le motivazioni per cui è importante affrontarlo con metodi automatici, le tecniche di machine learning ed i dataset che verranno utilizzati in questo progetto.
- Il secondo capitolo contiene una panoramica sulle reti neurali e convoluzionali, concentrandosi su quelle che sono utilizzate negli esperimenti.
- Il terzo capitolo descrive i metodi di preprocessing esplorati successivamente.
- Il quarto capitolo spiega i framework utilizzati nello sviluppo del progetto e l'uso che è stato fatto di diverse librerie.
- Il quinto capitolo riguarda la progettazione dell'applicazione, spiegando come sono stati eseguiti gli esperimenti, i protocolli di testing, gli indicatori di performance usati per il confronto delle prestazioni e cosa è stato implementato per facilitare l'esecuzione degli esperimenti ed il confronto dei dati sulle performance.
- La tesi termina con un'analisi riassuntiva dei risultati ottenuti e descrive i possibili sviluppi futuri.

1 La classificazione di coralli

Le barriere coralline sono ecosistemi formati dal deposito di scheletri di coralli, accumulati nel corso di centinaia o migliaia di anni (Burke et al. 2011).

Oltre a fare da habitat per numerose specie marine, le barriere coralline sono utili per milioni di persone (Burke et al. 2011; Mahmood et al. 2017).

- Diverse zone del mondo dipendono da esse per procurarsi nutrimento. Per esempio molte specie di pesci che vengono mangiate in paesi in via di sviluppo hanno bisogno delle barriere coralline per sopravvivere.
- Molti stati hanno industrie turistiche bastate sulle barriere coralline.
- Alcuni prodotti farmaceutici sono estratti da prodotti chimici derivati da questi ecosistemi.
- In alcune coste le barriere coralline offrono protezione da erosione, inondazioni e tempeste.

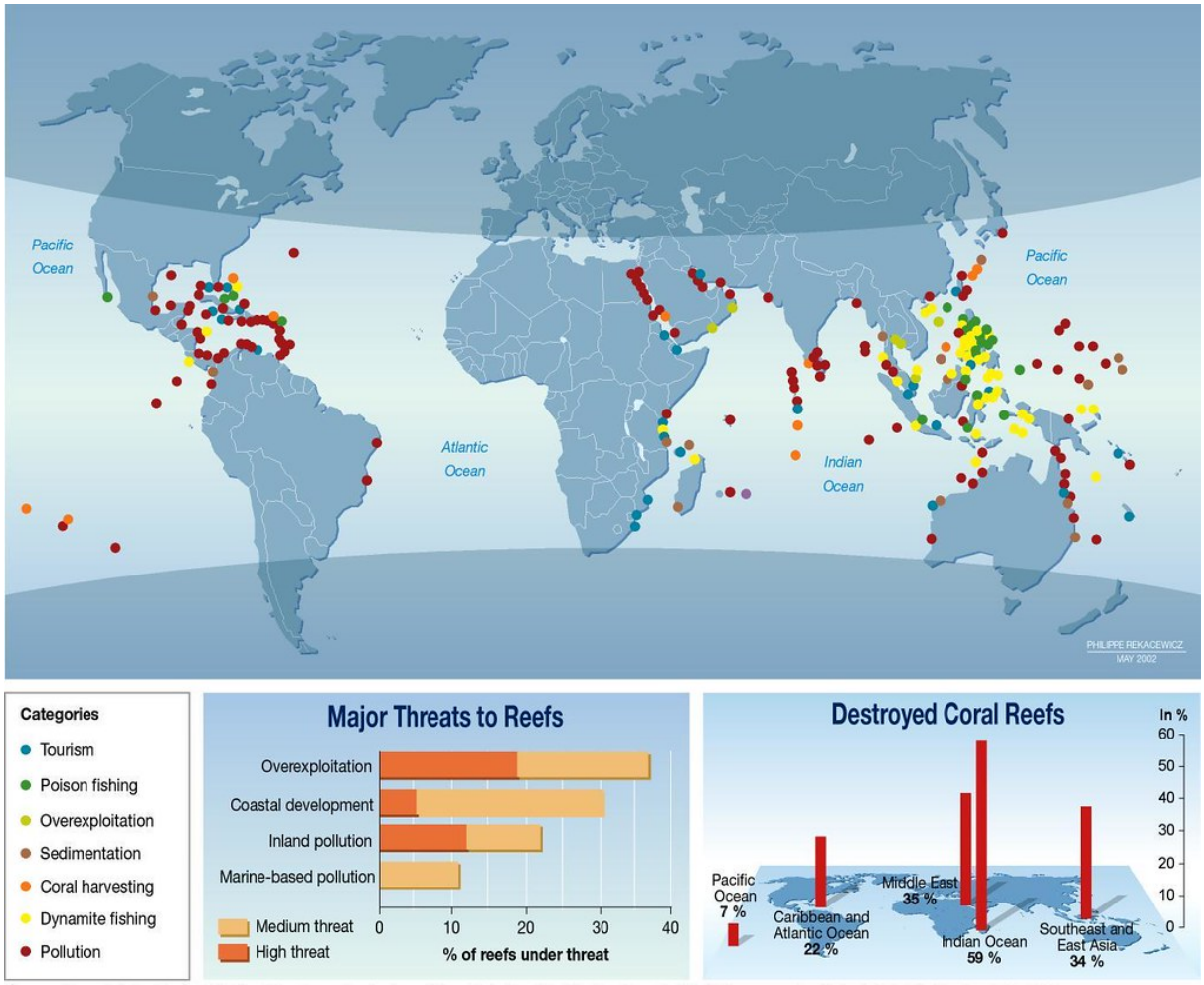
In questo capitolo vengono spiegate le due principali motivazioni per cui è necessario sviluppare metodi per la classificazione automatica di barriere coralline: trapianto di coralli sani in zone danneggiate e monitoraggio degli ecosistemi marini. Successivamente vengono descritti alcuni metodi di classificazione usati in letteratura e basati su estrazione di feature definite manualmente e le motivazioni per cui non sono ottimali. Infine si descrivono i dataset che sono stati prodotti per addestrare modelli di classificazione su questo problema ed in particolare i due usati in questa tesi.

1.1 Motivazioni

1.1.1 Effetti dell'uomo sulle barriere coralline

Le attività umane hanno avuto effetti devastanti sulle barriere coralline, in figura 1 sono riassunte le stime sulle cause di questi danni, le loro rispettive gravità e la relativa distruttività (Elawady 2015).

I danni sono causati principalmente da attività inquinanti, pesca incontrollata e dal cambiamen-



Source: Bryant et al., *Reefs at Risk; a Map-Based Indicator of Threats to the World's Coral Reefs*, World Resources Institute (WRI), Washington DC, 1998.

Figura 1: Grafico riassuntivo dei pericoli per barriere coralline e mappa delle zone interessate (Rekacewicz 2006)

to climatico. Il problema è già evidente ed è stimato che tutti i danni aumenteranno nei prossimi anni (Burke et al. 2011).

1.1.2 Trapianto di coralli

Il modo migliore per evitare questi danni è la prevenzione, tuttavia questo richiederebbe sforzi da parte di tutte le nazioni per ridurre l'impatto ambientale. Poiché questo non è facilmente ottenibile, sono stati sviluppati altri metodi per proteggere le barriere coralline.

Uno dei metodi attualmente in sviluppo è il trapianto di coralli, che consiste nel portare campioni di coralli in barriere danneggiate allo scopo di far ricrescere le colonie (Elawady 2015).

Il metodo funziona ma il tempo necessario ad eseguirlo da operatori umani è troppo dispendioso. Per questo sono stati sviluppati veicoli sottomarini autonomi (AUV) per eseguire questo lavoro più velocemente, in particolare usando tecnologie di intelligenza di sciame. Per poter effettuare il trapianto correttamente, però, questi veicoli hanno bisogno di identificare sul posto dove effettuare il trapianto. Questo problema viene risolto addestrando un modello capace di riconoscere le specie di coralli presenti nel loro campo visivo (Elawady 2015).

1.1.3 Monitoraggio habitat marini

Un altro motivo per sviluppare la classificazione automatica di coralli viene dalla necessità di monitorare gli habitat marini per studiare l'evoluzione dei cambiamenti ambientali e capire meglio gli effetti dell'uomo sull'ambiente. Monitorare aree a lungo termine è una strategia necessaria per studiare i cambiamenti nel tempo di un ecosistema (Mahmood et al. 2017).

Lo sviluppo di nuove tecniche per ottenere immagini sottomarine e l'aumento esponenziale di immagini e video prodotti ha portato alla produzione di dati con una frequenza molto maggiore rispetto alla velocità con cui gli scienziati riescano a catalogarli. Questo perché l'annotazione manuale di immagini richiede molto tempo e serve un occhio esperto per farlo correttamente. (Mahmood et al. 2017).

Quindi è chiaro che servono tecnologie in grado di classificare le immagini generate senza alcuna supervisone da esperti umani (Mahmood et al. 2017).

1.2 Metodi proposti in letteratura

Nel corso del tempo sono stati sviluppate diverse tecniche per classificare coralli basate su estrazione di feature dalle immagini e classificazione basata sull'analisi di queste feature.

Tipicamente le feature sono generate usando local binary pattern (LBP) o sue varianti; estrazione di informazioni da diversi spazi di colore come HSV e L*a*b* ed uso di metodi come descrittori di texture. Alcuni di questi metodi verranno analizzati più in dettaglio in una sezione successiva della tesi (Mahmood et al. 2017).

La classificazione avviene usando classificatori general purpose come i Support Vector Machine (SVM); analisi del discriminante lineare (LDA); k-nearest neighbor (KNN) e reti neurali

(NN) (Mahmood et al. 2017).

Nella tabella 1 sono riassunti alcuni di questi metodi ed i relativi articoli dove sono stati sperimentati. Si può notare come le informazioni usate da questi studi sono quasi sempre legate a texture e colori. Infatti le immagini estratte dalle barriere coralline presentano spesso un pattern tessiturale, inoltre (Beijbom et al. 2012) ha mostrato che le feature basate sui colori sono molto informative per la classificazione di coralli se usate correttamente.

Pubblicazione	Feature estratte	Classificatore utilizzato
Clement, Dunbabin e Wyeth 2005	Texture: local binary pattern (LBP)	Log-likelihood similarity
Mehta et al. 2007	Colore: spazio di colore RGB	Suppot vector machines (SVM)
Pizarro et al. 2008	Colore: istogramma di coordinate cromatiche normalizzate Texture: bag of words con trasformazione a scala invariante Salienza: filtro di gabor	voting matching
Marcos et al. 2007	Colore: istogramma di coordinate cromatiche normalizzate Texture: LBP	Analisi del discriminante lineare (LDA)
Johnson-Roberson, Kumar e Willams 2007	Colore: RGB e HSV Texture: wavelet globali	SVM
Purser et al. 2009	Texture: fliltri di Gabor non lineari	reti neurali artificiali
Stokes e Deane 2009	Colore: istogramma normalizzato RGB Texture: trasformata discreta del coseno	Probability density weighted mean distance (PDWMD)
Beijbom et al. 2012	Colore: spazio di colori L*a*b* Texture: filtro di massima risposta	SMV
Schoening et al. 2012	Colore e Texture: standard MPEG7	SVM
Stough, Greer e Benson 2012	Colore: funzioni quantili Texture: trasformazioni a scala invariante	SVM lineare
Shihavuddin et al. 2013	LBP, matrice di cooccorrenze dei livelli di grigio, flitri di risposta di Gabor e istogrammi di tinta del colore	k-nearest neighbor, rete neurale, SVM, PDWMD

Tabella 1: Diversi metodi usati in letteratura per la classificazione di coralli (Elawady 2015)

Di recente è stato mostrato che metodi di classificazione basati su reti neurali convoluzionali sono in grado di produrre risultati equivalenti a quelli precedenti o persino superiori (Gómez-Ríos et al. 2019). Le reti neurali sono modelli di classificazione molto potenti ma richiedono grandi quantità di dati per l'addestramento. Al contrario degli altri metodi come SVM, con le reti neurali non è necessario usare feature per ottenere risultati allo stato dell'arte in quanto possono apprendere direttamente dalle immagini originali (Gómez-Ríos et al. 2019).

Questa tesi si concentra sulle reti neurali convoluzionali, che verranno descritte più in dettaglio nel capitolo successivo, combinate con metodi di preprocessing delle immagini di input per cercare di capire se è possibile migliorare il modello risultante combinando entrambi gli approcci.

1.3 Descrizione dei dataset

Ci sono diversi dataset per la classificazione di coralli sottomarini, di questi EILAT, RSMAS, MLC, EILAT2 e Red Sea Mosaic sono a colori (Shihavuddin 2017; Gómez-Ríos et al. 2019) .

EILAT e RSMAS sono dataset di piccole/medie dimensioni per i quali c'è già stato del lavoro di ricerca che ha prodotto reti neurali capaci di ottenere accuratezza allo stato dell'arte (Gómez-Ríos et al. 2019). Per questo motivo la mia tesi si concentra su questi due dataset.

Di seguito vengono descritti in dettaglio i dataset con cui si lavora nella fase sperimentale del progetto di tesi.

1.3.1 EILAT

EILAT è composto da 1123 immagini di dimensione 64 x 64 che sono frammenti di foto di coralli trovati nel mar rosso (Shihavuddin 2017).

Il dataset è diviso in otto classi dalle dimensioni sbilanciate. I nomi delle classi sono identificativi progressivi che non rappresentano nessuna nomenclatura effettiva. La distribuzione dei dati è riportata nella tabella 2 ed in figura 2.

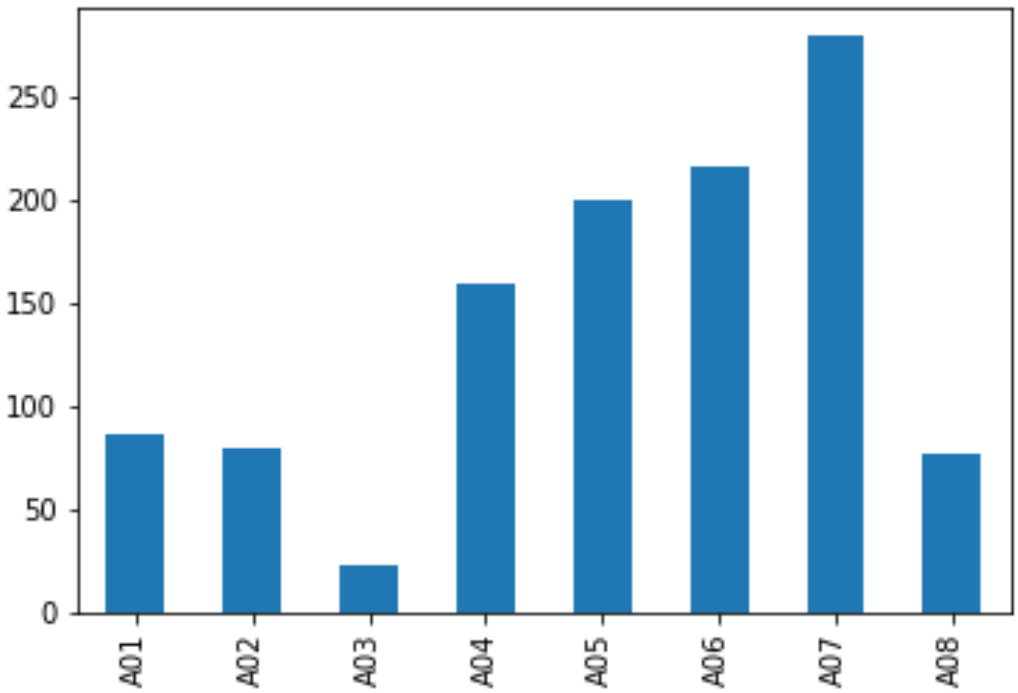


Figura 2: Grafico a barre della distribuzione delle classi del dataset EILAT

Classe	Numero istanze
A01	87
A02	80
A03	23
A04	160
A05	200
A06	216
A07	280
A08	77

Tabella 2: Numero istanze per classe nel dataset EILAT

Tutte le immagini sono state catturate nelle stesse condizioni di illuminazione e con gli stessi strumenti (Gómez-Ríos et al. 2019).

In figura 3 viene riportato un esempio per classe del dataset, disposti in ordine di nomenclatura.

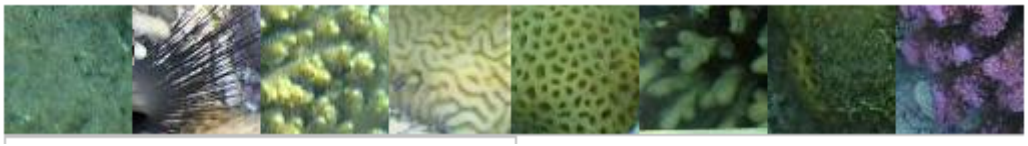


Figura 3: Esempi di immagini di coralli dal dataset EILAT. Viene riportata un'immagine per ogni classe

1.3.2 RSMAS

RSMAS è composta da 766 immagini di dimensione 256 x 256 (Shihavuddin 2017).

Il dataset è composto da 14 classi sbilanciate. I nomi delle classi sono effettivamente nomi di specie di coralli. La distribuzione dei dati è riportata nella tabella 3 ed in figura 4.

Classe	Numero istanze
ACER	109
APAL	77
CNAT	57
DANT	63
DSTR	24
GORG	60
MALC	22
MCAV	79
MMEA	54
MONT	28
PALY	32
SPO	88
SSID	37
TUNI	36

Tabella 3: Numero istanze per classe nel dataset RSMAS

Al contrario del dataset EILAT, queste immagini sono state catturate in tutto il mondo e quindi sia le condizioni esterne come la luce che la qualità delle fotocamere usate variano (Gómez-Ríos et al. 2019).

In figura 5 viene riportato un esempio di due immagini del dataset. Si può chiaramente vedere

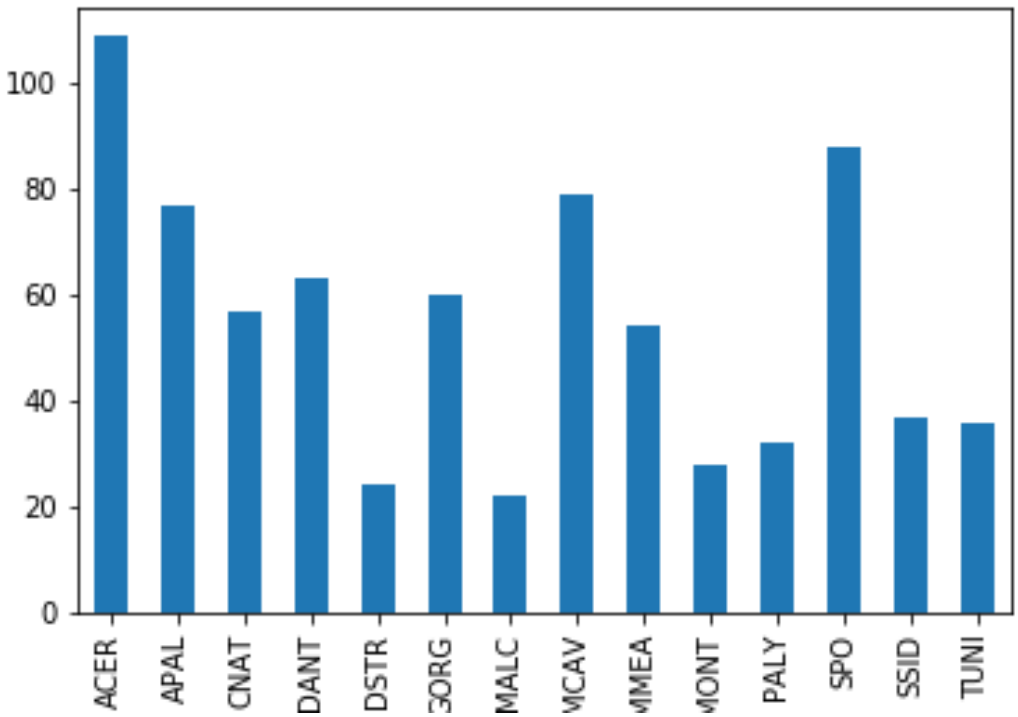


Figura 4: Grafico a barre della distribuzione delle classi del dataset RSMAS

come le immagini siano più grandi. Per via della dimensione delle immagini e del numero maggiore di classi ho deciso di non riportare più di due esempi in questa tesi.

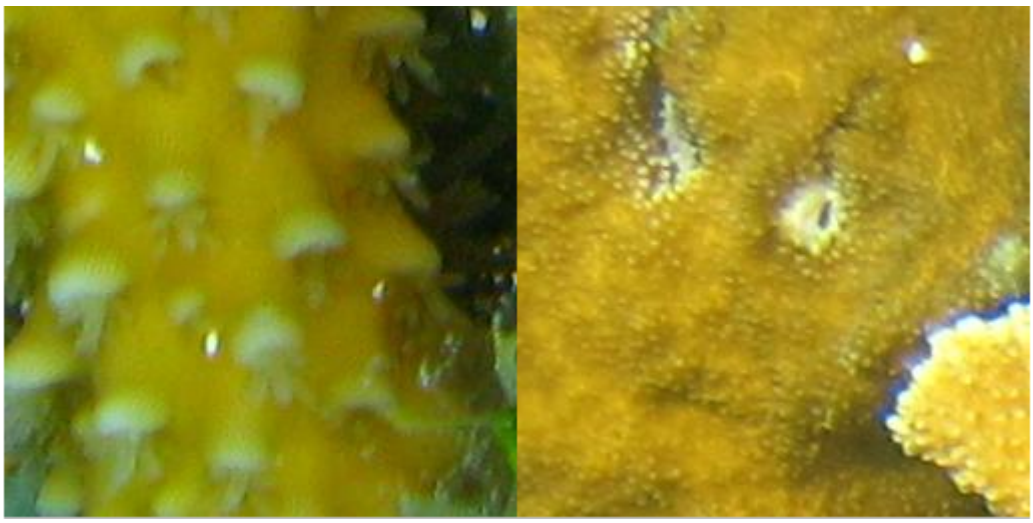


Figura 5: Esempi di immagini di coralli dal dataset RSMAS. Vengono solo riportate un'immagine per la classe ACER ed una per la classe APAL

2 Le reti neurali

2.1 Panoramica classificazione

Nel machine learning, un algoritmo di apprendimento supervisionato è un algoritmo il cui risultato è una funzione $y(x)$ selezionata tra un insieme di funzioni in modo da minimizzare l'errore tra $y(x)$ ed un target y rispetto ad un insieme di coppie di valori (x_i, y_i) detto set di training (Bishop 2006).

Nel problema della classificazione i target y_i corrispondono ad un numero finito di classi. Di conseguenza anche l'output del modello $y(x_i)$ deve restituire un valore intero, rappresentante la classe che il modello pensa che corrisponda ad x_i (Bishop 2006).

Di solito, l'insieme delle funzioni tra cui scegliere viene rappresentata da un insieme di parametri, θ. A diversi valori dei pesi θ corrispondono diversi modelli $y(x;\theta)$.

Inoltre, invece di predire solo un valore $y(x)$ data una x, nei problemi di classificazione i modelli tendono ad approssimare l'effettiva distribuzione di y condizionato ad x presente nel training set, $p_{data}(y|x)$, con una distribuzione parametrizzata da θ (Goodfellow, Bengio e Courville 2016):

$$p_{model}(y|x;\theta) \tag{2.1}$$

Il problema diventa, quindi, scegliere il valore di θ che approssima meglio la distribuzione dei dati. Per valutare la qualità di un modello, il metodo più usato consiste nel calcolare la cross entropy tra le due distribuzioni (Goodfellow, Bengio e Courville 2016):

$$J(\theta) = -\mathbb{E}_{x,y\sim p_{data}} log p_{model}(y|x;\theta) \tag{2.2}$$

Minimizzando questa funzione, detta funzione di loss, rispetto a θ si trova il modello che rappresenta meglio i dati di training tra quelli rappresentabili dalla famiglia di modelli su cui si fa la ricerca.

2.2 Reti neurali

Molti modelli consistono nell'applicazione di una funzione non lineare, detta funzione di attivazione, su una combinazione lineare di trasformazioni dell'input (Bishop 2006):

$$p_{model}(y|x;\theta) = f(\sum_{i=0}^{k} \theta_i \phi_i(x)) \tag{2.3}$$

Dove f è la funzione di attivazione e le ϕ_i sono dette funzioni base. $\phi_0(x) = 1$ e quindi θ_0 è un peso che non dipende da x ed è detto bias. Le altre k funzioni base sono solitamente trasformazioni non lineari di x.

Nella maggior parte dei modelli di machine learning i θ_i sono definiti a priori, come l'estrazione delle feature di cui si è parlato nel contesto della classificazione dei coralli. Tuttavia una rete neurale è un modello che apprende i ϕ_i dai dati, rappresentando anch'essi con funzioni parametriche (Bishop 2006).

Un altro modo per formalizzare una rete neurale è il grafo computazionale. Questo è un grafo in cui ogni nodo rappresenta una variabile ed ogni operazione è rappresentata da archi che vanno dalle variabili di input alla variabile di output (Goodfellow, Bengio e Courville 2016). I nodi che corrispondono agli input del modello hanno solo archi in uscita mentre i nodi di output hanno solo archi in entrata. In questo contesto la funzione di attivazione definita come sopra è l'operazione che produce l'output mentre il resto degli archi rappresenta le operazioni che producono i ϕ_i.

Il grafo computazionale di una rete neurale è tipicamente organizzato in layer: il primo layer è composto dall'input x ed una variabile $x_0 = 1$. Ogni layer è composta da un numero variabile di celle che applicano una trasformazione come all'equazione (2.3) sui valori del layer precedente ed in aggiunta un $z_0 = 1$ per continuare ad avere un valore indipendente dall'input ad ogni layer (Bishop 2006).

Se ogni layer prende come input l'intero output del layer precedente ed i pesi della matrice Θ_i vengono addestrati indipendentemente tra loro ad ogni passo il layer si dice completamente connesso. Se non ci sono pesi che vanno da un layer al precedente la rete si dice feed forward.

Le funzioni di attivazione più comuni sono la sigmoide:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.4}$$

L'unità lineare rettificata (ReLU):

$$ReLU(z) = max(0,z) \tag{2.5}$$

E l'arcotangente.

Per la sua semplicità e gradiente costante a tratti, ReLU si è dimostrata la funzione di attivazione migliore per le layer interne di una rete neurale profonda (Goodfellow, Bengio e Courville 2016).

Softmax è la trasformazione che si usa per produrre l'output finale di una rete usata per la classificazione perché può essere interpretata come una distribuzione di probabilità sulle label:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{2.6}$$

Dove C è il numero di valori $z_1, ..., z_C$ che il target della classificazione y può assumere.

2.3 Addestramento delle reti neurali

L'ottimizzazione dei parametri si basa sull'algoritmo della discesa del gradiente, che consiste nel derivare la funzione di loss e spostare i parametri nella direzione di massima opposta a quella di massima crescita del gradiente (Goodfellow, Bengio e Courville 2016).

Un'iterazione della discesa del gradiente aggiorna i pesi nel seguente modo:

$$\theta' = \theta - \varepsilon \nabla_\theta J(\theta) \tag{2.7}$$

Il parametro ε si chiama learning rate e rappresenta quanto velocemente i pesi vengono aggiornati. Più grande è questo valore più veloce è l'algoritmo ma se è troppo grande potrebbe spostare i pesi in una posizione dove $J(\theta)$ è persino maggiore dell'iterazione precedente.

Il calcolo del gradiente avviene attraverso la backpropagation, un algoritmo per calcolare le derivate parziali rispetto ai pesi in ogni layer in funzione dell'output del modello. L'algoritmo di backpropagation si basa sull'applicazione della regola della catena per derivare funzioni composte al grafo di computazione della rete neurale (Goodfellow, Bengio e Courville 2016):

$$\nabla_x z = J^T \nabla_y z \tag{2.8}$$

Dove J è la matrice Jacobiana che contiene le derivate di y rispetto ad x. Nel caso della backpropagation si usa ricorsivamente questa formula per calcolare le derivate dei pesi rispetto solo ad informazioni locali.

Il metodo della discesa del gradiente non garantisce di trovare una soluzione ottima, ma è computazionalmente efficiente anche per reti con milioni di pesi ed in pratica riesce a trovare modelli sufficientemente corretti per la maggior parte dei problemi di classificazione.

Le reti neurali con molti parametri possono imparare modelli che non generalizzano correttamente rispetto a dati fuori dal training set. Questo avviene quando la distribuzione dei dati contiene del rumore e questo rumore viene incorporato nella predizione effettuata dal modello.

Per garantire che la generalizzazione avvenga correttamente, di solito si valida il modello in un insieme di dati distinto dal training set, detto validation set. Confrontando i modelli su questo insieme si può avere un'approssimazione dell'accuratezza del modello su dati mai visti.

La regolarizzazione consiste in ogni modifica all'algoritmo di addestramento che punta a ridurre l'errore sul set di validazione senza dover usare i dati di quel set nell'addestramento (Goodfellow, Bengio e Courville 2016), altrimenti non rappresenterebbero più un insieme di dati mai visti.

Alcuni metodi di regolarizzazione includono la modifica della funzione di loss, per esempio aggiungendo una penalità sulla dimensione dei pesi oppure sul numero di pesi non zero. Un'altra forma di regolarizzazione consiste nell'aumentare il training set con semplici trasformazioni dell'input per aggiungere una maggiore resistenza ai rumori casuali (Goodfellow, Bengio e Courville 2016).

Infine, in alcuni casi, condividere il valore di alcuni parametri tra loro può migliorare la ge-

neralizzazione. Nel contesto della classificazione di immagini, la condivisione di parametri che avviene nelle reti convoluzionali, permette di apprendere invarianze transitive e rotazionali nelle immagini (Goodfellow, Bengio e Courville 2016). Nella sezione successiva verrà trattata più in dettaglio questo tipo di regolarizzazione.

2.4 Reti neurali convoluzionali

Come descritto sopra, le reti convoluzionali sono state introdotte per migliorare l'invarianza dei modelli rispetto a trasformazioni sulle immagini come rotazioni e traslazioni.

La convoluzione su un'immagine è un'operazione che consiste nell'applicare una media pesata dei valori attorno ad ogni pixel, dove i pesi sono rappresentati in una matrice K di dimensione $m * n$.

Matematicamente una convoluzione su I con il filtro K di dimensione $m * m$ si rappresenta con:

$$S(i,j) = (I * K)(i,j) = \sum_{x=1}^{m} \sum_{y=1}^{n} I(i + \lceil m/2 \rceil - x, j + \lceil n/2 \rceil - y) K(x,y) \tag{2.9}$$

Un layer convoluzionale effettua una convoluzione sull'input. Nel caso della convoluzione su due dimensioni la layer apprende m*n pesi, i valori del filtro della convoluzione, che solitamente sono molti di meno dei pesi che necessita un layer completamente connesso (Goodfellow, Bengio e Courville 2016).

Di solito la convoluzione è seguita da una funzione di attivazione come ReLU e da un passo di pooling che consiste nel prendere la media o il massimo di un patch di pixel, diminuendo quindi la dimensione del risultato.

Questo insieme di operazioni è detto feature map ed ogni layer può essere composto da più feature map parallele che apprendono filtri diversi.

In diversi casi le reti neurali convoluzionali sono state in grado di ottenere risultati allo stato dell'arte in diversi problemi di classificazione particolarmente difficili (Krizhevsky, Sutskever e Hinton 2012).

Di solito le reti convoluzionali terminano con dei layer completamente connessi per effettuare

la classificazione usando le feature apprese dalle feature maps.

2.5 Reti neurali di successo

In questa sezione vengono elencati alcuni modelli che sono attualmente allo stato dell'arte per la classificazione e che poi verranno usati nella fase sperimentale di questa tesi.

2.5.1 ResNet

ResNet è un'architettura in ogni layer convoluzionale, prima di effettuare l'operazione sull'input, viene effettuata una somma elemento per elemento con l'ouptut di un layer precedente (He et al. 2015).

Se gli output dei layer sono delle stesse dimensioni e se non è presente un'operazione di pooling che riduce la dimensione dell'output, questa connessione non aggiunge nuovi parametri. Altrimenti è necessario applicare una proiezione lineare per sommare la feature map precedente a quella di dimensione minore (He et al. 2015).

In pratica ResNet ottiene risultati migliori della versione senza connessioni residuali. Il risultato può essere attribuito alla migliore propagazione del gradiente per merito di queste connessioni che saltano alcuni passaggi (He et al. 2015).

In figura 6 sono confrontate le versioni schematiche di una rete convoluzionale tradizionale ed una ResNet.

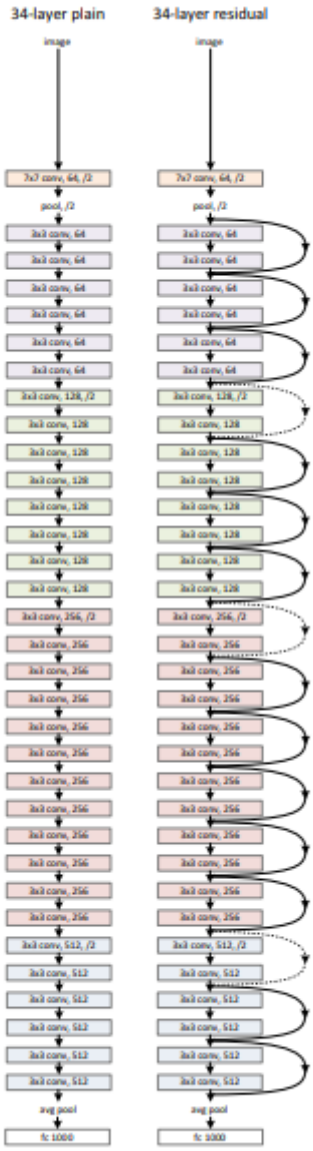


Figura 6: Confronto tra una rete convoluzionale normale ed una ResNet delle stesse dimensioni con diversi blocchi residuali. Le frecce tratteggiate rappresentano proiezioni (He et al. 2015, © 2015 IEEE)

2.5.2 DenseNet

DenseNet, al contrario di ResNet, invece di fare una somma tra l'output e quello dei layer precedenti, usa l'operazione di concatenazione (Huang, Liu e Weinberger 2016).

Un blocco denso consiste in una sequenza di feature maps convoluzionali, senza polling, dove l'input di una mappa è la concatenazione degli output di tutte le mappe precedenti. Un esempio di blocco denso è indicato in figura 7.

Una rete DenseNet è composta da una serie di blocchi densi intervallati da layer di pooling (Huang, Liu e Weinberger 2016).

Per via della concatenazione il calcolo dell'output del modello è più costoso che resnet, dato che bisogna applicare la convoluzione ad un input più grande ad ogni layer. Tuttavia l'approccio non richiede un numero maggiore di parametri perché le layer convoluzionali usano gli stessi pesi indipendentemente dalla dimensione dell'input e la concatenazione non richiede nuovi parametri (Huang, Liu e Weinberger 2016).

Sperimentalmente, DenseNet ha raggiunto risultati allo stato dell'arte in dataset complessi come ImageNet (Huang, Liu e Weinberger 2016).

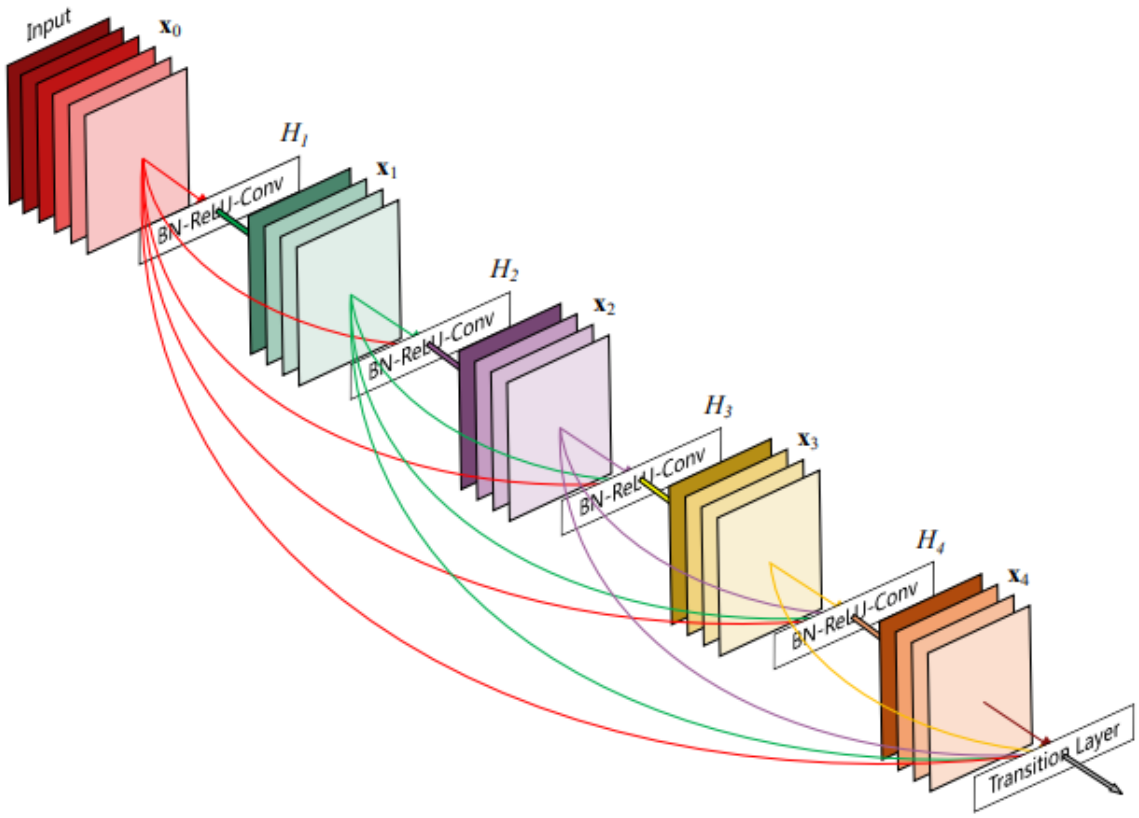


Figura 7: Un blocco denso, l'elemento di cui è composta una DenseNet (Huang, Liu e Weinberger 2016, © 2016 IEEE)

2.6 Transfer learning

I modelli di successo discussi sopra sono molto potenti ma addestrarli richiede molto tempo e dati, il che è un problema per dataset che non contengono milioni di immagini ed è ancora più problematico per effettuare numerosi esperimenti con configurazioni diverse. Tuttavia questi problemi possono essere alleviati usando la tecnica del transfer learning.

Il metodo consiste nel prendere il corpo di una rete convoluzionale addestrata su un dataset ed usarlo per addestrare un modello su un altro dataset, cambiando solo le ultime layer completamente connesse. La tecnica si basa sull'ipotesi che le layer più interne apprendano feature generiche e quindi trasferibili a problemi diversi dall'originale ed i risultati sperimentali mostrano che l'approccio funziona (Donahue et al. 2013).

In particolare ImageNet (Deng et al. 2009) è il dataset più usato per trasfer learning in problemi di classificazione di immagini poiché è un dataset che contiene una vasta quantità di classi e milioni di immagini. Quindi ogni rete addestrata su questo dataset dovrà necessariamente imparare ad estrarre feature molto generiche così da riconoscere tra migliaia di classi di oggetti differenti.

Nonostante le classi di ImageNet sono molto diverse dalle classi di coralli nei dataset in questo progetto, la parte sperimentale di questa tesi discuterà i meriti di transfer learning per la classificazione dei coralli.

3 Metodi di preprocessing

Nel capitolo 1 sono state elencate le tecniche di estrazione di feature più usate nel preprocessing per modelli di classificazione di coralli. In questo capitolo vengono approfonditi i metodi che sono stati utilizzati nel progetto di questa tesi.

Come notato sopra, le feature estratte riguardano sempre colore oppure texture, entrambi i tipi di preprocessing sono stati sperimentati in questa tesi.

3.1 Spazi di colori

Le immagini a colori vengono normalmente rappresentate nello spazio di colori RGB, dove ogni pixel è composto da tre valori distinti, ciascuno con valore da 0 a 255, che consistono nella quantità rispettivamente di rosso, verde e blu che si trova in quel pixel. Usando questi tre colori in diverse misure si ottengono tutti gli altri per via del modo in cui vengono percepiti dalla retina.

In figura 8 viene rappresentato lo spazio di colori RGB come un cubo nello spazio tridimensionale.

Lo spazio RGB è quello più utilizzato per la rappresentazione di immagini su uno schermo, ma per alcune applicazioni altre rappresentazioni possono essere più utili.

Come visto nella tabella 1, gli spazi di colori HSV e L*a*b* si sono dimostrati utili in alcuni contesti di classificazione di coralli. In questa sezione vengono descritti questi due spazi di colori in maggiore dettaglio.

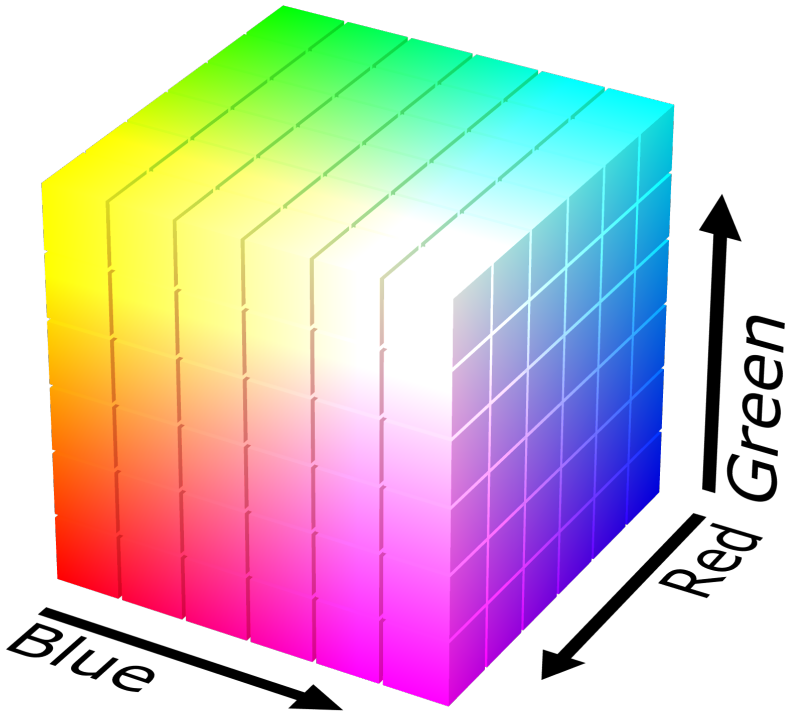


Figura 8: Rappresentazione a cubo dello spazio di colori RGB (Horvath 2008)

3.1.1 HSV

Lo spazio HSV è un modello che rappresenta i colori in funzione di attributi più simili a come gli uomini vedono i colori: tinta, saturazione e valore.

La tinta è una proprietà circolare che rappresenta il tipo di colore: rosso, giallo, verde, blu, ecc.

La saturazione indica l'intensità del colore. Se vale 0 il colore è in bianco e nero, indipendentemente dalla tinta. Più la saturazione aumenta, più si fa vivo il colore rappresentato.

Infine il valore è associato alla luminosità del colore. A 0 il colore rappresentato è nero e si schiarisce al suo crescere. Il bianco è rappresentato dal massimo valore e 0 saturazione. A differenza del modello HSL dove a luminosità massima il colore è sempre bianco, nel modello HSV con saturazione diversa da 0, il colore dipende anche dalla tinta.

Questo spazio di colori è rappresentabile da un cono come si può vedere in figura 9.

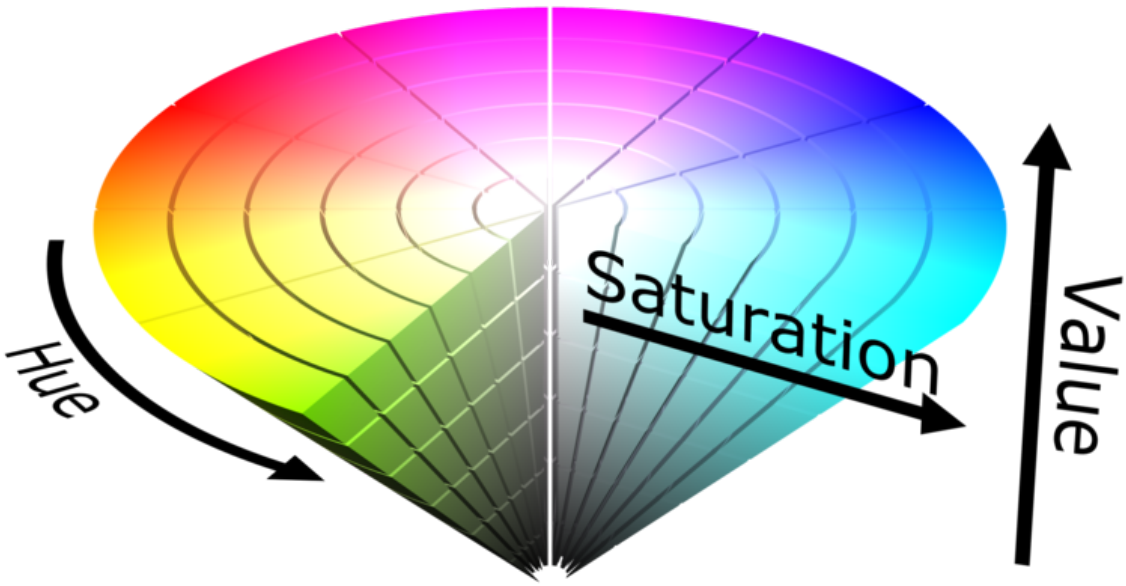


Figura 9: Rappresentazione a cono dello spazio di colori HSV (Horvath 2008)

3.1.2 L*a*b*

Lo spazio di colori CIELAB, detto anche L*a*b*, è un'altra alternativa al modello RBG dove i colori sono rappresentati da tre variabili: la luminosità, L*; a*, che va da verde a rosso, e b* che va da blu a giallo.

In questo spazio di colori ogni cambiamento numerico nei tre valori corrisponde ad un cambiamento nel colore percepito di misura proporzionale.

La modalità Lab copre l'intero spettro visibile dall'occhio umano e lo rappresenta in modo uniforme. Esso permette quindi di descrivere l'insieme dei colori visibili indipendentemente da qualsiasi tecnologia grafica.

Nelle figure 10 e 11 lo spazio di colori L*a*b* è mostrato prima con una vista dall'alto per vedere i colori rispetto ad a* e b* e poi una laterale per vedere come cambiano in funzione di L*.

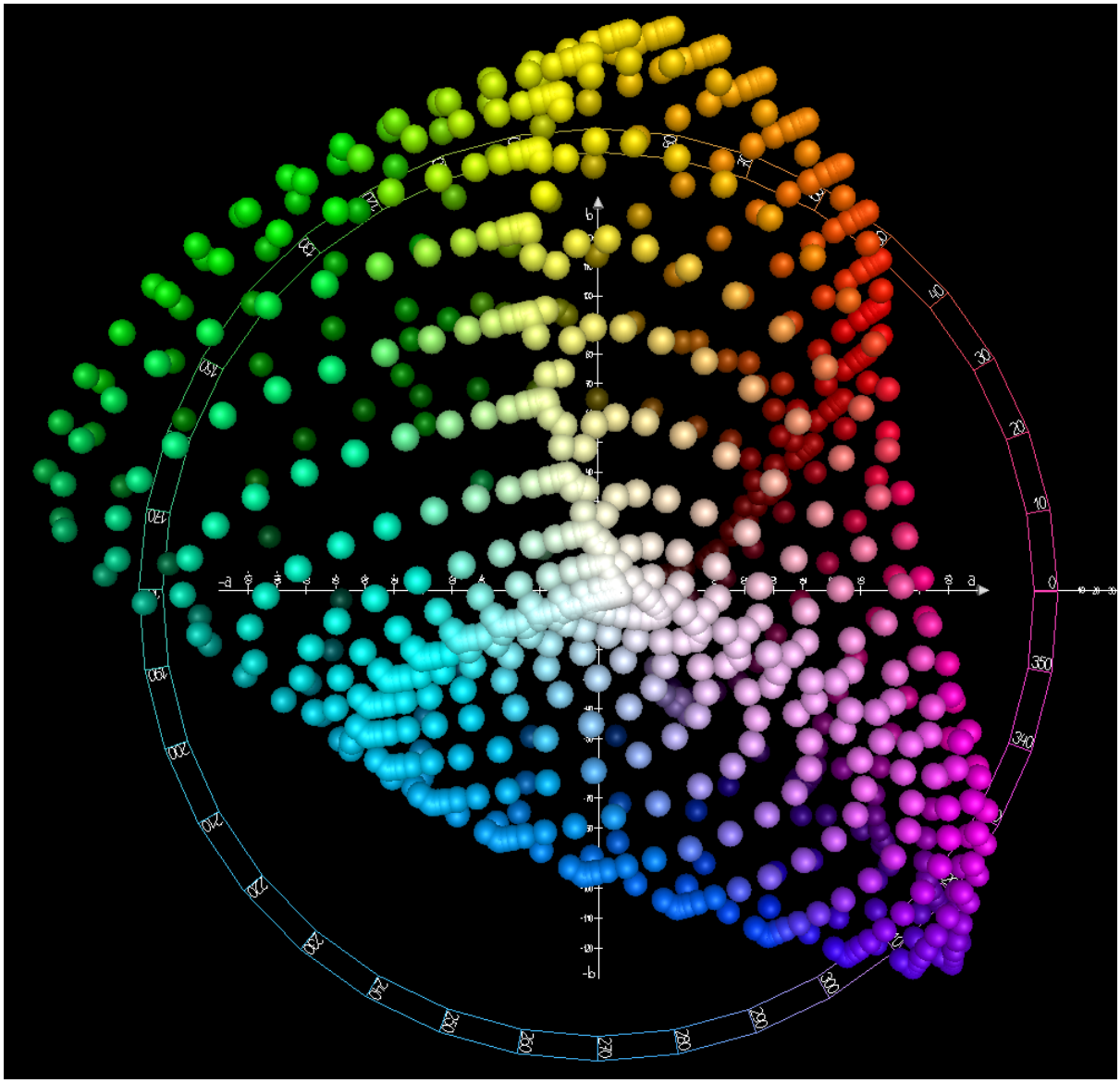


Figura 10: Vista dall'alto dello spazio di colori L*a*b* dove a* è nell'asse orizzontale e b* in quello verticale (Everding 2015a)

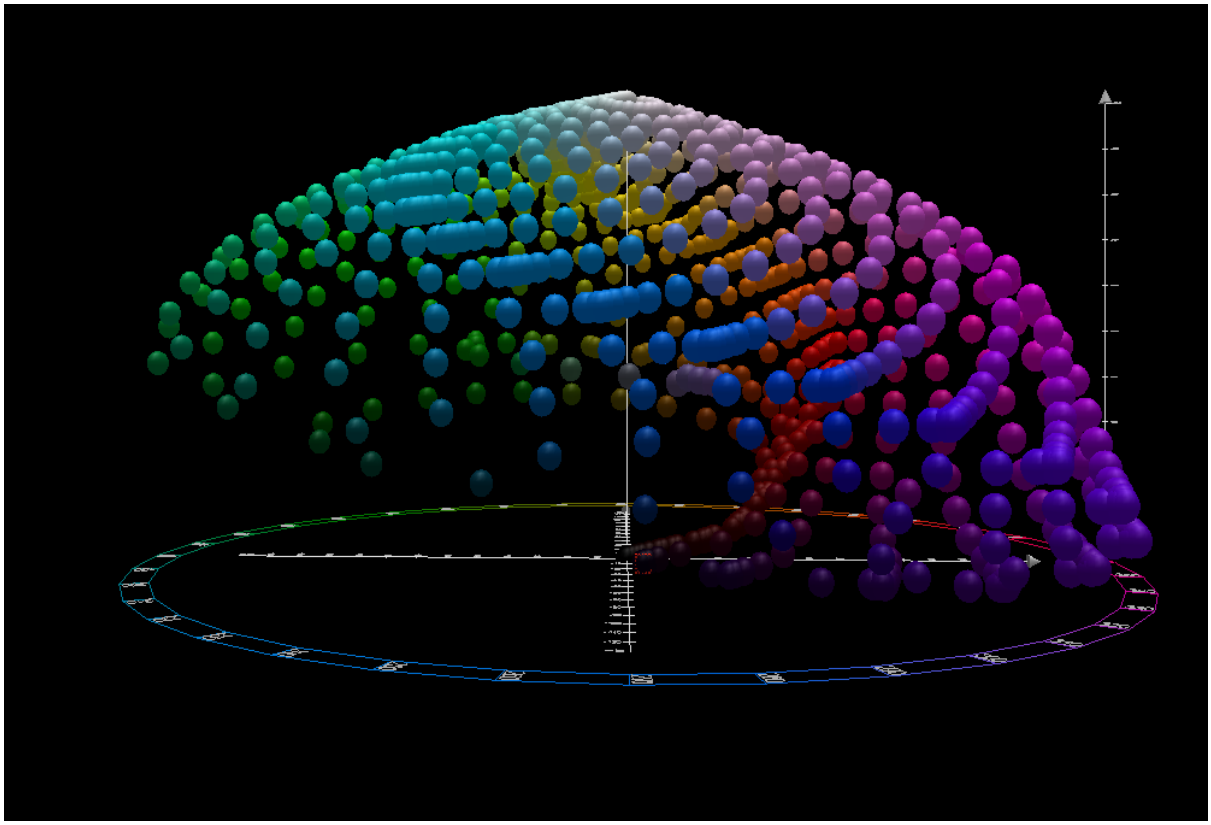


Figura 11: Vista laterale dello spazio di colori L*a*b* dove a* è nell'asse orizzontale e L* in quello verticale (Everding 2015b)

3.2 Tiling

Una delle proprietà discusse nel capitolo 1 riguardo alla classificazione dei coralli è la presenza di pattern tessiturali: le immagini di coralli contengono molte forme che si ripetono molto simili in posizioni diverse.

Questa caratteristica dovrebbe consentire di riconoscere la corretta classe di appartenenza di un'immagine anche avendo solo una frazione contigua di essa, non perdendo informazioni essenziali tagliandola.

Il tiling consiste nell'estrarre un rettangolo, spesso quadrato, da un'immagine ed usare solo quello nel processo di classificazione. Questo metodo è una forma di data augmentation: per ogni immagine originale, il set di training ne contiene diverse, una per ogni scelta della posizione del tile. Il set di training, quindi, viene incrementato di un fattore moltiplicativo senza dover raccogliere nuove immagini.

Questa forma di preprocessing migliora la generalizzazione perché è più difficile che avvenga overfitting su un numero maggiore di dati. In questo caso, inoltre, non dovrebbe peggiorare molto le performance per via della scarsa perdita di informazioni come spiegato sopra.

L'operazione di tiling può essere combinata con altre estrazioni di feature, specialmente se queste mantengono le proprietà tessiturali dell'immagine originale.

In figura 12 viene mostrato un esempio di tiling su un'immagine dal dataset RSMAS. Si può vedere come pattern simili si ritrovano in ogni tile individuato eppure non sono tanto uguali da essere semplici copie l'uno dell'altro.

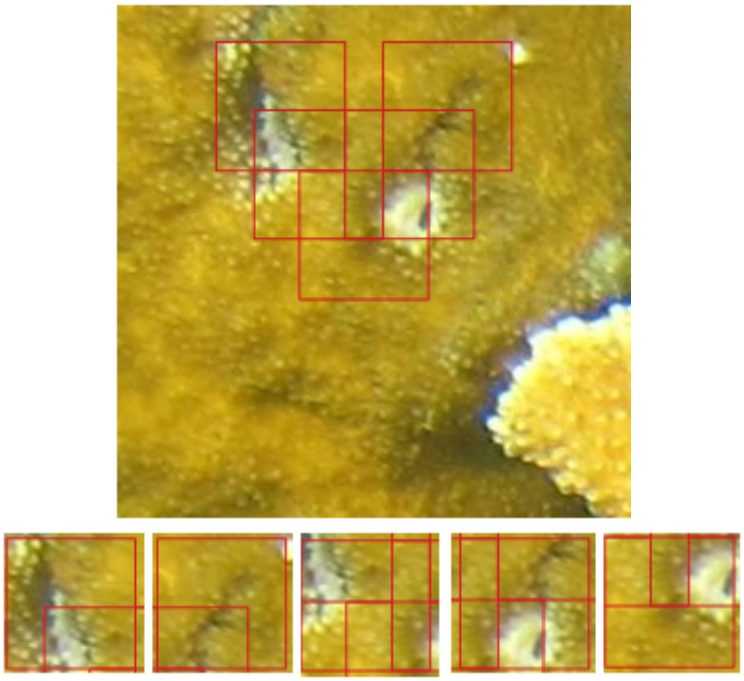


Figura 12: Esempi di tiling su una delle immagini di RSMAS mostrate in figura 5. Sopra) Immagine originale dove ogni rettangolo rosso rappresenta un tile diverso ed ha la stessa dimensione delle immagini in EILAT. Sotto) Tile estratti dall'immagine sopra

3.3 Local binary pattern

Local binary pattern (LBP) è una delle tecniche di estrazione di feature più prominenti nella tabella 1. Si tratta di una trasformazione su immagini che identifica informazioni tessiturali.

L'operazione consiste nell'esaminare un intorno di pixel estratto uniformemente da una circonferenza attorno ad ogni pixel dell'immagine. Questi valori esaminati vengono binarizzati rispetto al valore del pixel centrale: se sono minori si ottiene 0, altrimenti 1. I numeri così calcolati vengono concatenati insieme, scorrendo la circonferenza in senso orario, ed inseriti nell'immagine risultato in corrispondenza del pixel valutato (Huang et al. 2011).

Il numero di vicini da considerare ed il raggio del cerchio sono parametri che influenzano il risultato, cambiando la distanza dei pattern identificabili e la precisione con cui vengono trovati (Huang et al. 2011).

LBP è definito su immagini grayscale ma si estende facilmente ad immagini RGB, come quelle trattate in questa tesi, applicando l'operazione sui tre canali dell'immagine separatamente e concatenando le tre immagini risultato come se fossero canali di colori.

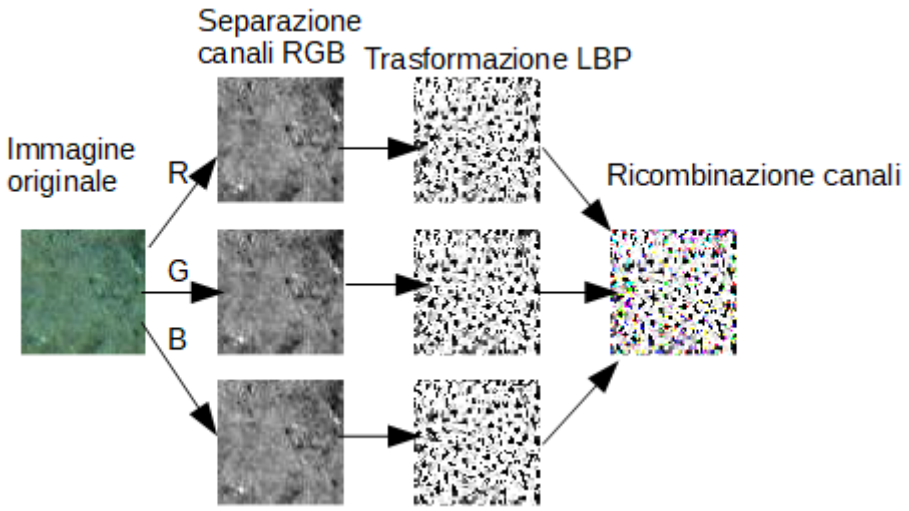


Figura 13: Dimostrazione dell'applicazione del local binary pattern su una delle immagini presentate in figura 3. L'operazione è stata eseguita con un raggio di 3 pixel e considerando 24 pixel vicini. I valori dell'immagine sono stati rimappati nell'intervallo $[0, 255]$ per migliore visualizzazione

La trasformazione LBP spesso si usa per ottenere un istogramma sui valori dei pixel nell'immagine risultato, che poi è usato in congiunta con classificatori più semplici come SVM. Siccome in questa tesi si lavora con reti neurali profonde si è deciso, invece, di utilizzare l'immagine intera.

In figura 13 viene mostrato un esempio di questa trasformazione, applicata ad un immagine dal dataset EILAT. I parametri dell'operazione sono stati scelti con una ricerca manuale e sono gli unici utilizzati poiché una ricerca esaustiva dei parametri migliori non era lo scopo della tesi. L'immagine risultato ha i valori rimappati per disegnarla ma i modelli creati in questo progetto ricevono come input l'immagine trasformata originale.

Un altro esempio è visibile in figura 14 con un'immagine a dimensione maggiore.

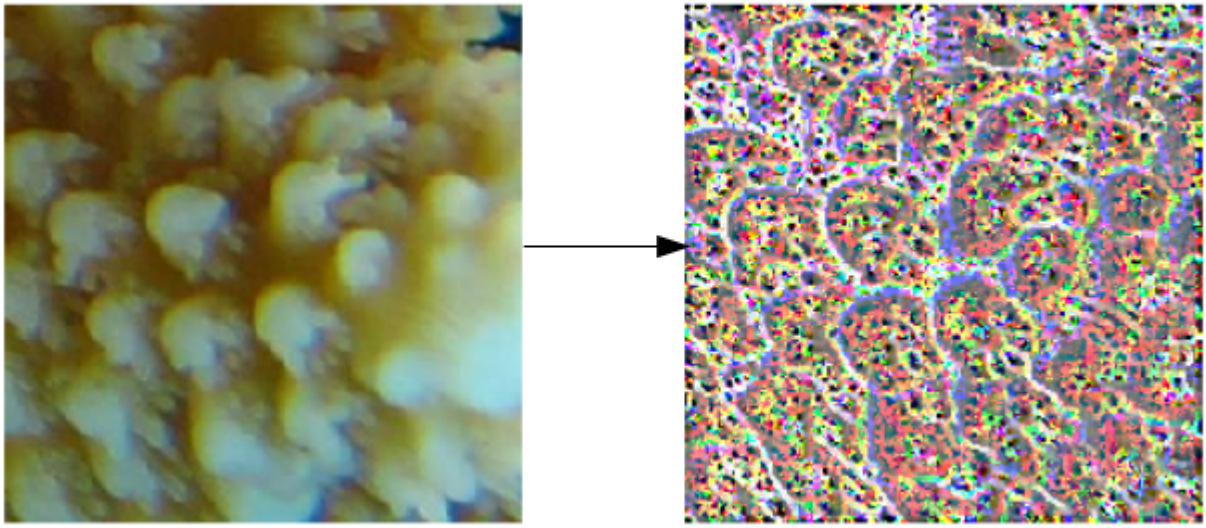


Figura 14: Applicazione di LBP su un'immagine dal dataset RSMAS

3.4 Wavelet

La trasformata Wavelet è un'altra operazione per l'estrazione di informazioni da texture, originata come variante della trasformata di Fourier. La Wavelet bidimensionale produce un'immagine composta da quattro quadranti, ciascuno con significato diverso. Il primo consiste in un'approssimazione dell'immagine originale e può essere usato anche per effettuare compressione con perdita. Gli altri tre quadranti evidenziano rispettivamente feature verticali, orizzontali e diagonali (Liu 2010).

La Wavelet può essere applicata ricorsivamente al primo quadrante per ottenere diversi livelli di precisione nel calcolo delle feature.

In questa tesi verrà usata solo una profondità del primo livello per via della bassa dimensione delle immagini.

A differenza delle altre operazione, dopo l'applicazione della Wavelet l'immagine viene divisa in quattro sotto immagini diverse fra loro, quindi l'uso del tiling non ha lo stesso vantaggio di produrre immagini simili indipendentemente dalla posizione del tile.

Un esempio delle feature estratte dalla trasformata Wavelet è visibile in figura 15 dove l'operazione è applicata sull'immagine usata nell'esempio della figura 13. Invece in figura 16 viene riportala l'operazione sull'immagine mostrata in figura 14.

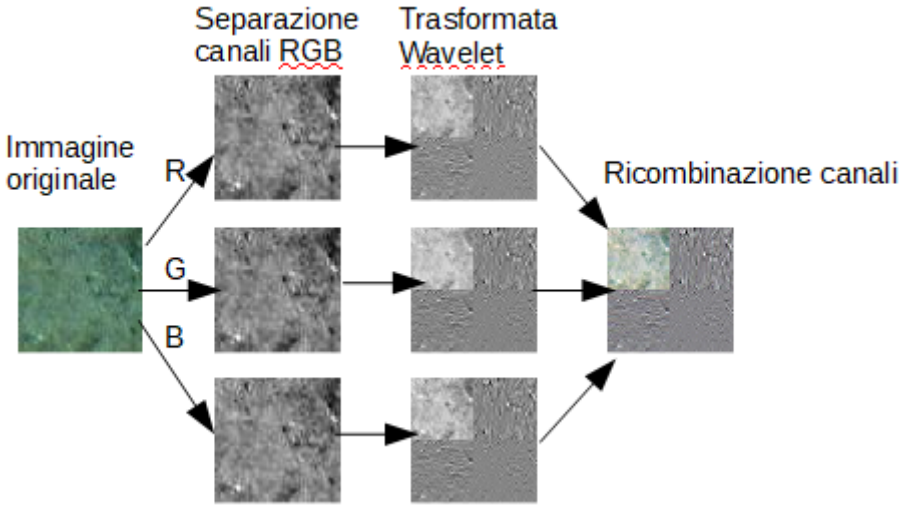


Figura 15: Dimostrazione dell'applicazione della trasformata Wavelet sulla stessa immagine dimostrata prima per LBP. I valori dell'immagine sono stati rimappati nell'intervallo [0, 255] per migliore visualizzazione

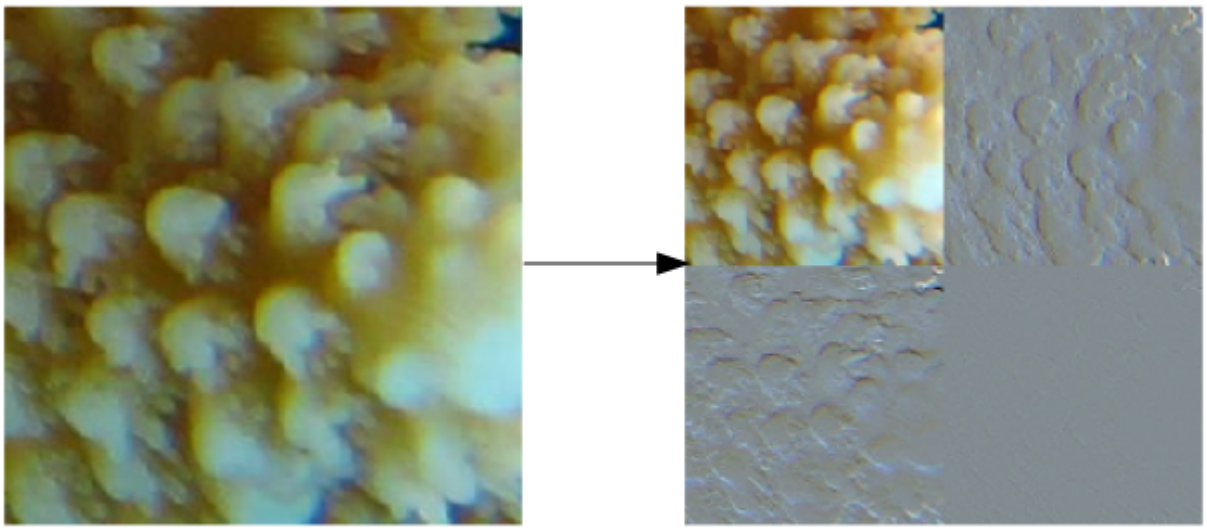


Figura 16: Applicazione di Wavelet su un'immagine dal dataset RSMAS

4 Framework

In questo capitolo sono descritti i framework e le librerie che sono state usate per aiutare l'implementazione degli esperimenti i cui risultati verranno discussi nel capitolo 6.

I framework principali sono l'API di Keras[1] come front end e la libreria Tensorflow[2] come back end e verranno descritti nelle due sezioni successive. L'ultima sezione di questo capitolo fa una veloce panoramica del resto delle librerie utilizzate nel progetto.

4.1 Keras

Keras è un'API ad alto livello per addestrare e sperimentare con reti neurali in python. L'API è pensata per costruire facilmente modelli e velocizzare la sperimentazione.

In Keras un modello è definito da un grafo di moduli collegati insieme. Tra questi ci sono layer di reti neurali, funzioni di costo, algoritmi di ottimizzazione, schemi di inizializzazione e di regolarizzazione. La maggior parte delle volte il grafo è una sequenza di layer e una rete neurale feed forward.

Gli oggetti principali di Keras sono le layer, che definiscono operazioni usando pesi addestrabili, ed i modelli, che sono insiemi di layer applicate ai propri input seguendo schemi di esecuzione definiti in fase di creazione. La modularità degli oggetti di Keras permette di usare i modelli con la stessa API della layer, in modo da creare moduli di operazioni riutilizzabili.

I modelli, prima di essere addestrati ed eseguiti, devono essere compilati. Nella compilazione vanno definiti una funzione di costo, come la cross entropy vista nel capitolo 2, e un ottimizzatore, come la discesa del gradiente o sue le varianti che integrano forme di regolarizzazione nell'algoritmo.

Keras supporta esecuzione su GPU, il che è necessario per addestrare reti complesse parallelizzando il calcolo delle derivate ad ogni iterazione della discesa del gradiente, specialmente nel caso in cui si usino reti convoluzionali.

1. `https://keras.io/`
2. `https://www.tensorflow.org/`

Un'altro vantaggio di keras è che contiene un modulo[3] per creare modelli di successo, tra cui quelli descritti nel capitolo 2. Inoltre è possibile caricare nei modelli di successo pesi preaddestrati sul dataset ImageNet (Deng et al. 2009) per effettuare transfer learning.

Per via delle facilitazioni qui descritte Keras, è stata scelta in questa tesi per la creazione di modelli ed addestramento.

4.2 Tensorflow

Keras è un'API ad alto livello, quindi definisce i moduli e gli oggetti utilizzabili, tuttavia non fornisce un'implementazione dell'API. Per questo bisogna usare una libreria di back end che implementi il front end presentato da Keras.

Tensorflow è una delle librerie open source più popolari per il machine learning e un'implementazione dell'API di Keras.

Tensorflow è principalmente una libreria di manipolazione di tensori, che sono array multidimensionali immutabili di dati dello stesso tipo, similmente a come vengono definiti in matematica. Oltre ai tensori, la libreria offre diverse operazioni che lavorano su di essi. I tensori sono implementati in modo da poter essere mantenuti ed elaborati dalle GPU poiché la maggior parte delle operazioni su di essi sono parallelizzabili.

Per calcolare il gradiente delle funzioni eseguite dai modelli, Tensorflow incorpora metodi di derivazione automatica durante l'esecuzione delle operazioni.

Le variabili di Tensorflow sono oggetti mutabili. Per definire i pesi di un modello, la libreria usa variabili che contengono tensori.

I tensori e le operazioni sono organizzati in grafi computazionali basati su paradigmi di computazione a dataflow. Tensorflow può eseguire questi grafi su GPU per sfruttare il parallelismo offerto da questo modello computazionale.

L'API di Keras è implementata in Tensorflow usando i grafi computazionali per rappresentare i modelli di Keras, operazioni tensoriali per le layer e variabili per i pesi da addestrare.

3. `https://keras.io/applications/`

4.3 Altre librerie utilizzate

Le immagini vengono caricate dal filesystem con libreria Matplotlib[4] e manipolate come fossero array multidimensionali usando Numpy[5]. Matplotlib è stata utilizzata anche per salvare alcune figure poi usate in questo documento.

La composizione e scomposizione dei canali delle immagini è eseguita dalla libreria OpenCV[6], il cambiamento dello spazio dei colori e la trasformazione con LBP vengono effettuati dalla libreria Scikit-image[7], invece la trasformata Wavelet è eseguita dalle funzioni di PyWavelets[8].

Le Mappa di calore di Seaborn[9] sono usate per la creazione di matrici di confusione, che verranno discusse più in dettaglio nel capitolo successivo, e Pandas[10] viene utilizzata per migliorare la loro presentazione.

Infine Scikit-learn[11] è stata scelta per definire il protocollo di separazione dei dati tra insiemi di training e validazione e per la definizione delle metriche usate per valutare i risultati nel capitolo 6.

4. `https://matplotlib.org/`
5. `https://www.numpy.org/`
6. `https://pypi.org/project/opencv-python/`
7. `https://scikit-image.org/`
8. `https://pywavelets.readthedocs.io/`
9. `https://seaborn.pydata.org/`
10. `https://pandas.pydata.org/`
11. `https://scikit-learn.org/`

5 Progettazione applicazione

In questo capitolo viene spiegato come sono stati eseguiti gli esperimenti e quali sono gli iperparametri utilizzati. Inoltre viene indicato il metodo di addestramento, preprocessing e protocolli di validazione utilizzati. Successivamente vengono definiti gli indicatori di performance usati nel capitolo successivo per studiare i risultati. Infine si specifica il funzionamento dell'applicazione prodotta per eseguire gli esperimenti di questo progetto.

Il riferimento principale per questo progetto è (Gómez-Ríos et al. 2019), che affronta la classificazione sugli stessi dataset e con metodologie simili. Quindi il setup per gli esperimenti è fortemente ispirato dal loro lavoro.

5.1 Addestramento

Dato un modello compilato, Keras può eseguire automaticamente l'addestramento. La compilazione è la fase in cui viene stabilito come si esegue l'addestramento. Gli iperparametri con cui si compila il modello sono gli stessi usati da (Gómez-Ríos et al. 2019).

L'algoritmo di ottimizzazione è la discesa del gradiente stocastica, una variante della discesa del gradiente dove le derivate non sono calcolate sull'intero insieme di dati ma su un sottoinsieme più piccolo, detto batch. Si ripete la modifica dei pesi con batch diverse fino a visitare l'intero dataset. Una volta finito il dataset si dice che finisce un'epoca e corrisponde ad una iterazione dell'algoritmo originale.

Il numero di epoche è l'unico iperparametro che è stato cambiato rispetto a quelli testati in (Gómez-Ríos et al. 2019) e questa scelta verrà discussa più in dettaglio nel capitolo successivo.

La dimensione dei batch è stata presa con un compromesso tra efficienza e capacità di memoria della GPU. Per via delle dimensioni di RSMAS, si è visto che un batch da dimensione 32 è troppo oneroso per l'hardware dove sono stati fatti gli esperimenti, quindi si è scelto di usare il valore 16. Invece per EILAT si è potuto usare un valore più grande e si è scelto 64, che ha dato buoni risultati in (Gómez-Ríos et al. 2019).

La discesa del gradiente stocastica è compilata con i parametri specificati in tabella 4.

Iperparametro	Valore
Learing rate	10^{-3}
Decadenza learning rate	10^{-6}
Momento Nesterov	0.9

Tabella 4: Parametri della discesa del gradiente stocastico

Il learning rate è definito nel capitolo 2. Invece la sua decadenza è un parametro che viene decrementato al learning rate dopo ogni iterazione, così da stabilizzare l'apprendimento progressivamente durante l'esecuzione. Il momento Nesterov è usato in una variante della discesa del gradiente, equazione (2.7), basata su un concetto di velocità ed accelerazione come specificato nelle formule seguenti:

$$v' = \mu v - \varepsilon \nabla_\theta J(\theta + \mu v) \tag{5.1}$$

$$\theta' = \theta + v' \tag{5.2}$$

Dove v è una nuova variabile, detta velocità, che cambia ad ogni iterazione come θ e μ è il momento Nesterov. Nel progetto si è scelto di utilizzare questa variante in quanto, nella pratica, si è dimostrata migliore rispetto all'algoritmo classico.

Infine il modello è compilato utilizzando, come funzione di loss, la cress entropy definita nel capitolo 2.

Il transfer learning è stato applicato facendo partire i pesi da quelli di un modello addestrato su ImageNet, estratti facilmente tramite l'API di Keras.

Come in (Gómez-Ríos et al. 2019), il corpo convoluzionale di ResNet e DenseNet, viene terminato con un layer di pooling, una completamente connessa di dimensione 512 con attivazione ReLU ed un layer con attivazione softmax per l'output.

5.2 Preprocessing

Sono state testate tutte le forme di preprocessing discusse nel capitolo 3. Per via delle piccole dimensioni dei dataset è possibile caricare tutte le immagini in memoria ed applicare il preprocessing nell'interezza prima di addestrare il modello, così da velocizzare la discesa del gradiente. Tuttavia in casi reali sarebbe necessario applicare il preprocessing al momento dell'elaborazione delle immagini.

L'unica eccezione è il tiling: siccome il numero di immagini generato è molto maggiore della dimensione del dataset e non è pratico tenerle tutte in memoria, il tiling viene applicato alla creazione del batch e randomizzato ad ogni epoca.

5.3 Divisione del dataset

Come discusso nel capitolo 2, è necessario dividere il dataset in partizioni diverse per calcolare correttamente la generalizzazione su dati non visti durante l'addestramento. In particolare i dati di addestramento sono divisi in un set di training ed uno di validation con un rapporto $4 : 1$. Per stimare meglio l'accuratezza, si ripete l'addestramento 5 volte con set di validazione diversi. Il metodo si chiama K-fold cross validation e consiste nel dividere i dati in K partizioni, 5 in questo caso, e prenderne una come validazione ed il resto per training. L'accuratezza finale è stimata come la media delle accuratezze calcolate sui 5 fold di validazione.

Siccome le classi sono sbilanciate, viene usata la cross validation stratificata, che divide ogni fold in modo tale che la distribuzione delle classi sia la stessa in ogni partizione dei dati.

Per via della K-fold cross validation, ogni esperimento richiede di addestrare il modello 5 volte, il che aumenta molto i tempi di addestramento e costringe a ridurre il numero di iperpametri testati in questo progetto.

In aggiunta, un insieme di dati, detto set di testing, è stato tenuto da parte durante l'intero addestramento e ricerca degli iperparametri migliori. Questo insieme non è incluso nella k-fold cross validation e quindi non viene mai usato per l'addestramento del modello. In questo modo i modelli non possono fare overfitting su questi dati né con i parametri né con gli iperparametri. Il risultati sul set di testing riportati nel capitolo successivo sono stati calcolati dopo che tutti i modelli sono stati scelti.

5.4 Indicatori di performance

In accordo con (Gómez-Ríos et al. 2019), l'accuratezza viene usata come indicatore di performance principale allo scopo di poter confrontare i risultati. L'accuratezza è data dal numero di immagini classificate correttamente fratto il numero totale delle immagini viste.

L'accuratezza ha il problema di non tenere in conto dello sbilanciamento tra le classi. Per esempio, in un dataset dove il 90% dei dati è della stessa classe, un classificatore che sceglie sempre quella classe avrebbe un'accuratezza del 90% ma non avrebbe imparato nulla.

Per alleviare questo problema sono state inventate altre misure di performance. Due delle quali sono state usate in questo progetto.

La matrice di confusione è una tabella dove sia le righe che le colonne rappresentano le classi del dataset ed in ogni cella i, j viene indicato il numero di esempi della classe i nel dataset che il modello ha assegnato alla classe j.

Se il modello funziona correttamente, la matrice dovrebbe essere diagonale. Ogni elemento fuori dalla diagonale è un elemento classificato non correttamente. Dalla matrice di confusione si può calcolare l'accuratezza sommando gli elementi della diagonale e dividendo questa somma per la somma di tutti gli elementi della matrice.

$$accuratezza = \frac{\sum_{i=1}^{C} M_{i,i}}{\sum_{i=1}^{C} \sum_{j=1}^{C} M_{i,j}} \tag{5.3}$$

Sia M la matrice di confusione di dimensione CxC e sia $M_{i,j}$ l'elemento in riga i e colonna j. Si definiscono, per ogni classe, le seguenti misure:

$$precisione_i = \frac{M_{i,i}}{\sum_{j=1}^{C} M_{j,i}} \tag{5.4}$$

$$recall_i = \frac{M_{i,i}}{\sum_{j=1}^{C} M_{i,j}} \tag{5.5}$$

$$F_{1\_i} = 2\frac{precisione_i * recall_i}{precisione_i + recall_i} \tag{5.6}$$

Per tornare ad un indicatore composto da un solo numero si possono aggregare gli F_{1_i}, per esempio usando la media:

$$F_1 = \frac{1}{C}\sum_{i=1}^{C} F_{1\_i} \tag{5.7}$$

La misura appena definita, detta F_1 score, è stata utilizzata in questo progetto per confrontare i modelli tenendo conto dello sbilanciamento del dataset.

Tuttavia, si può notare che, confrontando l'$F_1 score$ e l'accuratezza nei vari esperimenti fatti su questi dataset, sembra esserci sempre una differenza molto lieve tra le due. Quindi l'accuratezza è significativa nonostante le classi siano sbilanciate.

Nel capitolo successivo verrà usata l'accuratezza per confrontare i modelli con le baseline in (Gómez-Ríos et al. 2019), lo score F_1 per confrontare i modelli tra loro se necessario ed infine verrà esaminata la matrice di confusione dei modelli peggio performanti per cercare di capire se c'è una concentrazione di errori rispetto a certe classi.

5.5 Descrizione dell'applicazione

Per eseguire gli esperimenti descritti ho progettato un'applicazione[1] in python. Il programma facilita l'esecuzione dell'addestramento dei modelli e la raccolta delle statistiche di performance su di essi.

Il path di un file di configurazione va specificato al lancio dell'applicazione. Questo deve contenere un oggetto json che descriva tutti i dettagli dello specifico esperimento da eseguire tra cui dataset, modello e una lista di operazioni di preprocessing da applicare.

L'applicazione carica il dataset corretto dal filesystem e trasforma l'insieme di immagini in array multidimensionali della libreria numpy. In caso serva applicare trasformazioni che non aumentano la dimensione del dataset, esse vengono eseguite immediatamente. Invece se c'è bisogno di applicare un preprocessing di data augmentation come il tiling, viene creato un generatore di immagini che applica la trasformazione all'estrazione di dati ed è usabile da Keras nelle iterazioni dell'addestramento.

1. `https://github.com/SimoneBarbaro/tesi/`

Uno stesso file di configurazione può contenere più esperimenti, definiti da parametri che accettano una lista di valori associati. Questi parametri sono dimensione del batch, operazioni di preprocessing e di data augmentation. Ogni combinazione tra gli elementi di queste liste viene eseguita come esperimento differente ed i risultati vengono salvati separatamente.

Per ogni esperimento è necessario addestrare 5 modelli, per via della cross validation discussa sopra. Il programma procede dividendo gli insiemi di dati di training e validazione, creando ed addestrando i modelli usando l'API di Keras.

Al termine di ogni addestramento, le performance vengono calcolate. Quando tutti i fold sono stati esplorati le performance vengono aggregate insieme e salvate. Quest'operazione conclude un esperimento e si passa al successivo.

Le performance di ogni esperimento vengono salvate su file di output se viene indicata una cartella dove salvare i file, altrimenti vengono stampate nello standard output. Ogni riga del file è un oggetto json con le performance al termine dell'addestramento. Infine è possibile specificare se si vuole calcolare le performance sul set di testing oppure no.

L'applicazione è anche in grado di riprendere esperimenti già iniziati e fermati, per esempio se la macchina è stata fermata tra un addestramento ed un altro. In base al numero di righe del file di output viene calcolato il numero di esperimenti già eseguiti e questi non vengono ripetuti. I modelli, inoltre, vengono salvati nella directory per riprendere gli esperimenti eseguiti solo in parte.

6 Risultati sperimentali

In questo capitolo sono riportati i risultati degli esperimenti eseguiti. Prima di tutto sono riportate le specifiche dell'hardware utilizzato e, nelle sezioni successive, vengono mostrate tabelle riassuntive delle performance nei vari esperimenti seguiti da commenti personali sui risultati.

6.1 Specifiche dell'hardware

Gli esperimenti sono stati eseguiti in Colab[1], una piattaforma che mette a disposizione gratuitamente un ambiente virtuale in cloud con accelerazione GPU.

Al momento della scrittura della tesi colab esegue su una CPU Xeon Processors @2.3Ghz a singolo core con hyper threading e 45MB di Cache; 12.6 GB di RAM ed una GPU 1xTesla K80 con 2496 core CUDA e 12GB di memoria.

6.2 Riproduzione risultati in letteratura

Prima di cominciare a sperimentare con i metodi di preprocessing, è necessario partire da una baseline con risultati soddisfacenti, quindi la prima parte di questo progetto consiste nel riprodurre risultati simili a quelli in letteratura.

In questo caso è stato preso come riferimento (Gómez-Ríos et al. 2019). Inizialmente congelando i pesi nelle layer convoluzionali come proposto dall'articolo, i pesi congelati non verranno modificati dalla discesa del gradiente, in questo modo impedendogli di cambiare da quelli appresi su ImageNet.

I risultati sono stati riportati in tabella 5.

1. `https://colab.research.google.com/`

Dataset	Modello	Epoche	Accuratezza training	Accuratezza validazione
EILAT	ResNet	500	0.9995	0.6430
	DenseNet	300	0.9975	0.4139
RSMAS	ResNet	700	0.9995	0.0978
	DenseNet	700	0.9978	0.5188

Tabella 5: Accuratezza nel set di training ed in quello di validazione per ogni combinazione dataset e modello usando il freezing e con numero di epoche come specificato nei risultati migliori di (Gómez-Ríos et al. 2019)

Purtroppo non si è riusciti a raggiungere la stessa accuratezza riportata nell'articolo, in particolare la differenza tra accuratezze suggerisce che ci sia stato overfitting.

La matrice di confusione, calcolata sui fold di validazione, esplora meglio il fallimento del modello nell'apprendere una rappresentazione accurata. In figura 17 viene riportata quella per il quarto esperimento della tabella 5. Il colore delle celle è più chiaro per i valori più grandi. La matrice rivela, confrontando la diagonale con il totale di ogni classe effettiva, che alcune classi sono state apprese correttamente ma la maggior parte no. Inoltre due delle classi vengono predette molto di più delle altre e non sono le più numerose, quindi non è un problema causato dallo sbilanciamento. Rispetto all'accuratezza in fase di training, sembra che i modelli abbiano fallito a generalizzare senza poter adattare i pesi nelle layer convoluzionali.

Al seguito di questo fallimento, si è provato ad addestrare le reti senza congelare i pesi e stavolta i risultati sono stati migliori ed il tempo di addestramento più breve.

In tabella sono 6 riassunti i risultati senza congelare il modello e con molte meno epoche di addestramento per arrivare a tale risultato. L'accuratezza raggiunta in questi test è simile a quella riportata in (Gómez-Ríos et al. 2019), specialmente nel dataset RSMAS, quindi questi saranno i modelli di riferimento che verranno modificati per sperimentare con le operazioni di preprocessing.

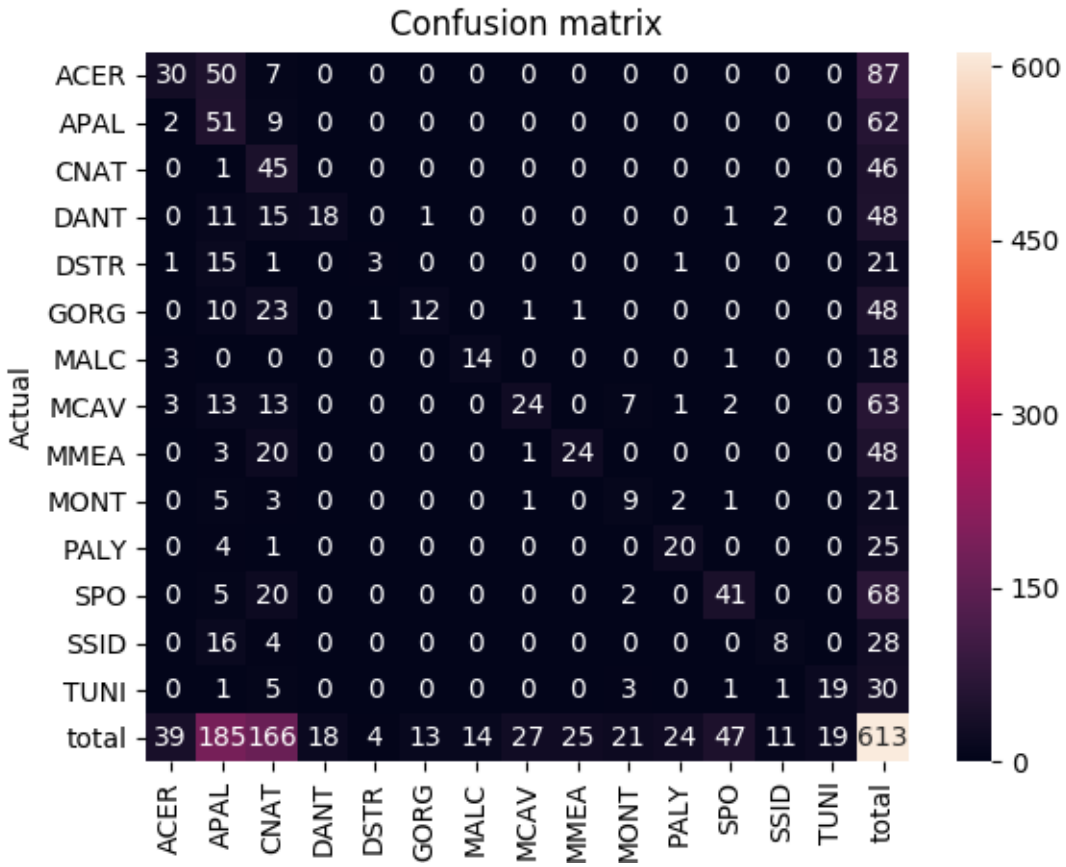


Figura 17: Matrice di confusione per l'esperimento riportato nell'ultima riga della tabella 5

Dataset	Modello	Epoche	Accuratezza	F_1
EILAT	ResNet	30	0.9295	0.9307
	DenseNet	30	0.9312	0.9320
RSMAS	ResNet	50	0.9790	0.9805
	DenseNet	50	0.9830	0.9870

Tabella 6: Risultati degli esperimenti senza congelare i pesi

Negli esperimenti successivi il numero di epoche è mantenuto a 30 per EILAT e 50 per RSMAS a meno che non è specificato altrimenti.

6.3 Importanza del trasnsfer learing

Nel capitolo 2 è stato introdotto il transfer learning ed è stato postulato che nonostante le differenze tra i dataset esaminati qui ed ImageNet, partire dai pesi addestrati su quel dataset è comunque necessario per addestrare bene i modelli.

In tabella 7 viene mostrato che, nonostante l'uso di più epoche, le performance sono più basse in tutti i modelli e per tutti i dataset se non viene usato il transer learning. Questo dimostra che le feature apprese su ImageNet sono utili anche per EILAT e RSMAS. Nel resto degli esperimenti il trasnfer learning viene mantenuto.

Dataset	Modello	Epoche	Accuratezza	F_1
EILAT	ResNet	100	0.6286	0.6241
	DenseNet	100	0.7634	0.7620
RSMAS	ResNet	100	0.8314	0.8328
	DenseNet	100	0.0993	0.0129

Tabella 7: Risultati degli esperimenti senza transfer learning, da confrontare con la tabella 6

6.4 Cambio dello spazio di colori

Il cambio dello spazio di colori discusso nel capitolo 3 è stato messo in pratica ed in questa sezione vengono discussi i risultati.

Siccome il pretraining su ImageNet è stato effettuato su immagini RGB ci si potrebbe aspettare che l'addestramento con spazi di colore diverso richieda più epoche per adattare i pesi alla nuova rappresentazione dei dati. Tuttavia, osservando la performance sul set di training, il modello sembra adattarsi abbastanza velocemente e raggiungere una classificazione quasi perfetta dopo poche iterazioni in tutti i casi. Quindi è stato scelto di tenere lo stesso numero di epoche usate nell'esperimento riportato in tabella 6.

La tabella 8 mostra i risultati per questi esperimenti. Si può notare come lo spazio L*a*b* sembri ottenere risultati molto superiori in EILAT rispetto allo spazio HSV; per RSMAS la differenza è, invece, molto più bassa e leggermente in favore di HSV.

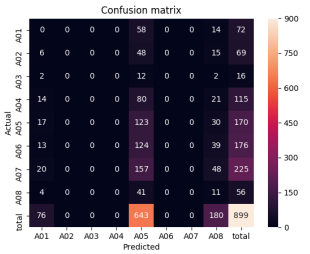
In EILAT, solo DenseNet con L*a*b* raggiunge una performance simile a quella ottenibile con RGB; invece in RSMAS tutte le trasformazioni portano a risultati più che accettabili. Quindi è possibile che le condizioni diverse dei due dataset abbiano portato a vantaggi diversi rispetto a questi due spazi di colore.

Il test non è conclusivo, ma sembra suggerire che l'uso di diversi spazi di colori potrebbe rilevarsi più utile per la classificazione di coralli.

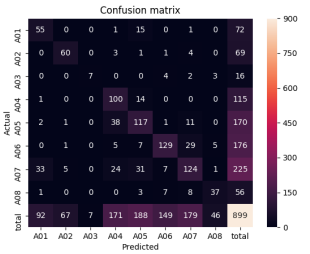
Dataset	Modello	Preprocessing	Accuratezza	F_1
EILAT	ResNet	HSV	0.1170	0.0289
		LAB	0.7437	0.7375
	DenseNet	HSV	0.7830	0.7848
		LAB	0.9286	0.9318
RSMAS	ResNet	HSV	0.9542	0.9573
		LAB	0.9425	0.9461
	DenseNet	HSV	0.9739	0.9705
		LAB	0.9660	0.9656

Tabella 8: Risultati dell'addestramento con cambiamento dello spazio di colore, da confrontare con la tabella 6

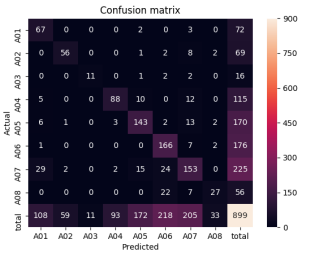
Gli unici risultati sotto al 90% di accuratezza sono le prime tre righe. Nella figura 18 vengono confrontate le matrici di confusione per capire meglio l'errore. Nel primo caso sembra che il modello non sia in grado di distinguere quasi mai le immagini e sceglie quasi sempre A05,



(a) Matrice di confusione per la prima riga



(b) Matrice di confusione per la seconda riga



(c) Matrice di confusione per la terza riga

Figura 18: Confronto delle matrici di confusione per gli esperimenti con risultati peggiori nella tabella 6

quindi deve aver memorizzato il training set senza apprendere un sistema di classificazione generale. Le altre due immagini mostrano una matrice più corretta ma non abbastanza diagonale, quindi è stato appreso un modello più capace a generalizzare ma che ancora fallisce su diversi esempi.

6.5 Local binary pattern

Similmente agli spazi di colori discussi sopra, sono stati fatti dei test sulla trasformazione local binary pattern ed i risultati sul set di training indicano che il modello ha appreso i dati.

Questa trasformazione era una di quelle più usate secondo la tabella 1 quindi ci si aspettavano risultati positivi usando LBP.

In tabella 9 si osservano i risultati di questo esperimento e si nota che questa trasformazione non raggiunge gli stessi livelli di performance sul set di testing rispetto all'uso di immagini RGB non trasformate.

Al momento non è chiaro se diverse configurazioni dei parametri della trasformata o varianti dell'operazione possano portare a risultati diversi, visto che questo progetto non aveva come scopo la ricerca dei valori migliori. Tuttavia questi risultati sembrano scoraggiare l'uso del LBP con modelli di deep learning.

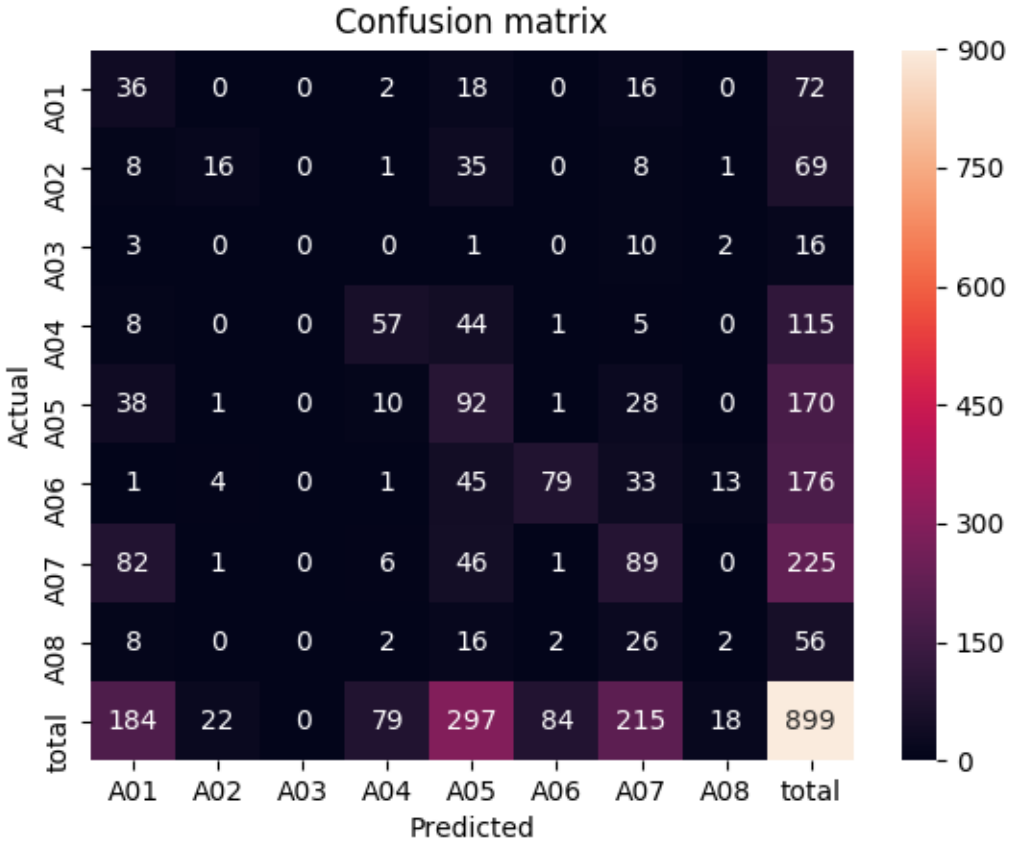


Figura 19: Matrice di confusione per l'esperimento riportato nella prima riga della tabella 9

Dataset	Modello	Accuratezza	F_1
EILAT	ResNet	0.3652	0.2798
	DenseNet	0.7223	0.6340
RSMAS	ResNet	0.8967	0.9080
	DenseNet	0.8993	0.9049

Tabella 9: Esperimenti con il pattern LBP, da confrontare con la tabella 6

L'esperimento peggiore è stato con ResNet su EILAT. In figura 19 si vede che anche in questo caso il modello pone un grande peso su un numero limitato di classi e persino 0 sulla classe meno numerosa.

6.6 Wavelet

Con un approccio simile a quello per il local binary pattern, la trasformata Wavelet è stata testata su tutte le combinazioni di modello e dataset ed i risultati sono riportati in tabella 10.

I risultati sono molto migliori di LBP, specialmente in RSMAS dove si raggiungono performance molto vicine alle immagini RGB.

Per via dell'immagine approssimata nel primo quadrante, la rete neurale ha potuto riutilizzare i pesi imparati su ImageNet, il che ha probabilmente aiutato. Da questi risultati non è però chiaro quale sia il contributo degli altri quadranti alla classificazione, altri esperimenti saranno necessari per capire se la trasformata Wavelet sia effettivamente utile.

Dataset	Modello	Accuratezza	F_1
EILAT	ResNet	0.8750	0.8679
	DenseNet	0.8910	0.8719
RSMAS	ResNet	0.9686	0.9774
	DenseNet	0.9647	0.9686

Tabella 10: Esperimenti con la trasformata Wavelet, da confrontare con la tabella 6

6.7 Tiling

Il tiling è stato applicato da solo ed assieme a diverse forme di preprocessing viste sopra e nel risultato sono riportati solo quelli per la dimensione di tile che ha dato risultati migliori nella misura F_1 score. La Wavelet è l'unica operazione che non è stata testata per via della perdita della texture dopo la trasformazione.

Come atteso, l'uso del tiling non ha portato a peggioramenti della performance per via della natura tessiturale delle immagini. Inoltre per RSMAS e con immagini RGB sia accuratezza che F_1 sembrano essere migliorate di un punto percentuale, arrivando anche a sorpassare i risultati migliori riportati in (Gómez-Ríos et al. 2019).

Il tiling è la forma di preprocessing che ha dato risultati migliori, probabilmente per via dell'aumento artificiale della dimensione del training set. L'uso di questa operazione in successivi lavori di classificazione di coralli potrebbe risultare in modelli migliori.

Dataset	Modello	Preprocessing	Tile	Accuratezza	F_1
EILAT	ResNet	Nessuno	16x16	0.9339	0.9357
		HSV	32x32	0.1518	0.0396
		Lab	32x32	0.8116	0.8122
		LBP	8x8	0.4321	0.3141
	DenseNet	Nessuno	32x32	0.9268	0.9272
		HSV	8x8	0.7884	0.7881
		Lab	16x16	0.9232	0.9237
		LBP	16x16	0.7214	0.6288
RSMAS	ResNet	Nessuno	16x16	0.9856	0.9881
		HSV	64x64	0.9634	0.9658
		Lab	32x32	0.9739	0.9752
		LBP	32x32	0.8954	0.9064
	DenseNet	Nessuno	16x16	0.9908	0.9921
		HSV	64x64	0.9673	0.9722
		Lab	64x64	0.9790	0.9817
		LBP	32x32	0.9150	0.9192

Tabella 11: Risultati dell'addestramento con tiling, anche combinato ad altre trasformazioni, da confrontare con le tabelle precedenti. La dimensione del tile riportata corrisponde a quella che ha dato risultati migliori rispetto all'F_1 score

6.8 Pretraining addizionale

L'ultimo test riguarda l'uso di un ulteriore pretraining prima di addestrare il modello. Questo è stato eseguito su RSMAS se si addestra poi su EILAT e viceversa. Il pretraining è avvenuto senza usare la cross validation ed il numero di epoche è stato scelto in base al dataset come fatto nei precedenti test.

In caso di preprocessing, l'operazione viene applicata anche durante il pretraining.

I risultati sono riportati in tabella 12 e riguardano tutte le trasformazioni eccetto il tiling, che non è stato eseguito per motivi di tempo.

Tra i risultati è interessante l'aumento di performance nello spazio HSV e della trasformazione LBP con ResNet. Questo suggerisce che il fallimento precedente può essere alleviato da un pretraining su immagini già trasformate. Viste le premesse di questa tesi, si è scelto di non sperimentare il pretraining su dataset più grandi. Ma la strada potrebbe essere promettente.

Dataset	Modello	Preprocessing	Accuratezza	F_1
EILAT	ResNet	Nessuno	0.9268	0.9306
		HSV	0.8035	0.7995
		Lab	0.8759	0.8800
		LBP	0.6732	0.5646
		Wavelet	0.8812	0.8843
	DenseNet	Nessuno	0.9241	0.9302
		HSV	0.8179	0.8246
		Lab	0.9071	0.9161
		LBP	0.7714	0.7057
		Wavelet	0.8901	0.8848
RSMAS	ResNet	Nessuno	0.9739	0.9781
		HSV	0.9725	0.9745
		Lab	0.9569	0.9573
		LBP	0.8915	0.9086
		Wavelet	0.9739	0.9769
	DenseNet	Nessuno	0.9791	0.9807
		HSV	0.9595	0.9590
		Lab	0.9765	0.9765
		LBP	0.9098	0.9109
		Wavelet	0.9399	0.9457

Tabella 12: Risultati dell'esperimento con pretraining sull'altro dataset

6.9 Commenti

I risultati di questo progetto mostrano che anche con l'uso di modelli di deep learning è possibile migliorare le prestazioni usando alcune delle forme di preprocessing ed estrazione di feature

già studiate in letteratura per il problema di classificazione dei coralli.

In particolare l'operazione di tiling sembra quella che ha aiutato di più mentre il local binary pattern è quella che ha avuto risultati peggiori.

Tuttavia potrebbe essere che con diverse configurazioni di LBP il risultato sia migliore.

Inoltre non sono state testate combinazioni di preprocessing che combinino immagini trasformate in modo diverso, per esempio concatenandole per creare più canali. Questo non è stato provato perché i pesi preaddestrati su ImageNet disponibili in Keras sono solo per immagini a 3 canali.

Esperimenti più esaustivi sono necessari per confermare la validità di queste osservazioni e dataset più grandi vanno esaminati per assicurarne la generalità.

Conclusione

La tesi si era posta come scopo quello di esplorare quali metodi di preprocessing possano essere utili all'addestramento di modelli per la classificazione di coralli.

In questa tesi si è portato a termine questo obiettivo riducendo i test a due dataset noti in letteratura, EILAT e RSMAS, e due modelli che hanno avuto risultati allo stato dell'arte su entrambi, ResNet e DenseNet. Quindi i modelli sono stati addestrati mantenendo le stesse configurazioni e variando il preprocessing sulle immagini e le performance sono state confrontate per stabilire quali trasformazioni hanno avuto effetti più positivi.

I test di questo progetto hanno indicato che l'operazione di tiling è l'unica che ha migliorato le performance rispetto alle baseline utilizzate per entrambi i dataset. Delle altre trasformazioni Wavelet non ha mostrato un guadagno notabile ma non ha peggiorato il risultato mentre local binary pattern ha avuto l'effetto peggiore. Infine le trasformazioni dello spazio di colori sembrano aver avuto effetti diversi nei due dataset: lo spazio HSV è stato peggiorativo in EILAT mentre ha portato a performance migliori di CIELAB su RSMAS, il quale, invece, ha portato ad un buon risultato su EILAT.

Questo lavoro lascia spazio a diversi possibili sviluppi futuri, qui vengono elencati alcuni di essi:

- Eseguire i test su dataset più grandi e vari di quelli sperimentati qui oppure su dataset dove c'è più di una classe per immagine, che sono più simili alle immagini effettivamente trovate dove questi sistemi dovranno essere applicati.
- Esplorare meglio i parametri di certe trasformazioni e provare a combinarle tra di loro per avere più feature insieme.
- Ripetere gli esperimenti su modelli diversi per accertare che i risultati ottenuti non sono solo specifici per ResNet e DenseNet.
- Estendere i test su altri tipi di preprocessing studiati in letteratura.
- Provare a capire se è possibile usare modelli appresi da queste estrazioni di feature per migliorare le prestazioni di altri modelli, per esempio usandoli per classificare immagini difficilmente distinguibili nello spazio RGB ma non in uno trasformato.

Bibliografia

Beijbom, Oscar, Edmunds Peter, Kline David, Greg Mitchell B e Kriegman David. 2012. "Automated Annotation of Coral Reef Survey Images". *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition:* 1170–1177. doi:10.1109/CVPR.2012.6247798.

Bishop, Christopher. 2006. *Pattern Recognition and Machine Learning.* 1ª edizione. Springer-Verlag New York. ISBN: 978-0-387-31073-2.

Burke, Lauretta, Reytar Katie, Spalding Mark e Perry Allison. 2011. *Reefs at Risk: A Map-Based Indicator of Threats to the World's Coral Reefs.* Washington, DC: World Resources Institute. ISBN: 978-1-56973-762-0.

Clement, Ryan, Matthew Dunbabin e Gordon Wyeth. 2005. "Toward robust image detection of crown-of-thorns starfish for autonomous population monitoring". *Australasian Conference on Robotics and Automation 2005.*

Deng, J., Dong W., Socher R., Li L., Li Kai e Fei-Fei Li. 2009. "ImageNet: A large-scale hierarchical image database". *2009 IEEE Conference on Computer Vision and Pattern Recognition:* 248–255. ISSN: 1063-6919. doi:10.1109/CVPR.2009.5206848.

Donahue, Jeff, Jia Yangqing, Vinyals Oriol, Hoffman Judy, Zhang Ning, Tzeng Eric e Darrell Trevor. 2013. "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition". *CoRR* abs/1310.1531. arXiv: 1310.1531. http://arxiv.org/abs/1310.1531.

Elawady, Mohamed. 2015. "Sparse Coral Classification Using Deep Convolutional Neural Networks". *CoRR* abs/1511.09067. arXiv: 1511.09067. http://arxiv.org/abs/1511.09067.

Everding, Holger kkk. 2015a. https://en.wikipedia.org/wiki/File:CIELAB_color_space_top_view.png.

———. 2015b. https://en.wikipedia.org/wiki/File:CIELAB_color_space_front_view.png.

Gómez-Ríos, Anabel, Tabik Siham, Luengo Julián, Shihavuddin ASM, Krawczyk Bartosz e Herrera Francisco. 2019. “Towards highly accurate coral texture images classification using deep convolutional neural networks and data augmentation”. *Expert Systems with Applications:* 315–328. doi:10.1016/j.eswa.2018.10.010.

Goodfellow, Ian, Yoshua Bengio e Aaron Courville. 2016. *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren e Jian Sun. 2015. “Deep Residual Learning for Image Recognition”. *CoRR* abs/1512.03385. arXiv: 1512.03385. http://arxiv.org/abs/1512.03385.

Horvath, Michael. 2008. http://lib.povray.org/searchcollection/index2.php.

Huang, D., C. Shan, M. Ardabilian, Y. Wang e L. Chen. 2011. “Local Binary Patterns and Its Application to Facial Image Analysis: A Survey”. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41 (6): 765–781. ISSN: 1094-6977. doi:10.1109/TSMCC.2011.2118750.

Huang, Gao, Zhuang Liu e Kilian Q. Weinberger. 2016. “Densely Connected Convolutional Networks”. *CoRR* abs/1608.06993. arXiv: 1608.06993. http://arxiv.org/abs/1608.06993.

Johnson-Roberson, Matthew, Suresh Kumar e Stefan Willams. 2007. “Segmentation and Classification of Coral for Oceanographic Surveys: A Semi-Supervised Machine Learning Approach”: 1–6. doi:10.1109/OCEANSAP.2006.4393835.

Krizhevsky, Alex, Ilya Sutskever e E. Geoffrey Hinton. 2012. “ImageNet Classification with Deep Convolutional Neural Networks”. *Neural Information Processing Systems* 25. doi:10.1145/3065386.

Liu, Chun-Lin. 2010. “A Tutorial of the Wavelet Transform”.

Mahmood, Ammar, Bennamoun Mohammed, Sohel Ferdous, An Senjian, Boussaid Farid, Hovey Renae, Kendrick Gary e Fisher Robert. 2017. “Deep Learning for Coral Classification”. *Handbook of Neural Computation.* doi:10.1016/b978-0-12-811318-9.00021-1.

Marcos, Ma. Shiela Angeli, Laura David, Eileen Peñaflor, Victor Ticzon e Maricor Soriano. 2007. “Automated benthic counting of living and non-living components in Ngedarrak Reef, Palau via subsurface underwater video”. *Environmental Monitoring and Assessment (2008):* 177–184. doi:10.1007/s10661-007-0027-2.

Mehta, Anand, Eraldo Ribeiro, Gilner Jessica e van Woesik Robert. 2007. “Coral reef texture classi cation using support vector machines”. *2007 International Conference on Computer Vision Theory and Applications, Barcelona, Spain.*

Pizarro, Oscar, Paul Rigby, Matthew Johnson-Roberson, Stefan Williams e Jamie Colquhoun. 2008. “Towards image-based marine habitat classification”: 1–7. doi:10.1109/OCEANS.2008.5152075.

Purser, A, M Bergmann, T Lundälv, J Ontrup e TW Nattkemper. 2009. “Use of machine-learning algorithms for the automated detection of cold-water coral habitats: a pilot study”. *Marine Ecology Progress Series* 397:241–251. doi:10.3354/meps08154. https://www.int-res.com/abstracts/meps/v397/p241-251/.

Rekacewicz, Philippe. 2006. “Coral reefs at risks”. http://grid-arendal.herokuapp.com/resources/5616.

Schoening, Timm, Melanie Bergmann, Jörg Ontrup, James Taylor, Jennifer Dannheim, Julian Gutt, Autun Purser e Tim W. Nattkemper. 2012. “Semi-Automated Image Analysis for the Assessment of Megafaunal Densities at the Arctic Deep-Sea Observatory HAUSGARTEN”. *PLOS ONE* 7 (6): 1–14. doi:10.1371/journal.pone.0038179. https://doi.org/10.1371/journal.pone.0038179.

Shihavuddin, A., Gracias N., Garcia R., Gleason A.C.R. e Gintert B. 2013. “Image-Based Coral Reef Classification and Thematic Mapping”. *Remote Sens.* doi:10.3390/rs5041809.

Shihavuddin, Asm. 2017. “Coral reef dataset”. *Mendeley Data, v2.* doi:10.17632/86y667257h.2.

Stokes, M. Dale, e Grant B. Deane. 2009. “Automated processing of coral reef benthic images”. *Limnology and Oceanography: Methods* 7 (2): 157–168. doi:10.4319/lom.2009.7.157. eprint: https://aslopubs.onlinelibrary.wiley.com/doi/pdf/10.4319/lom.2009.7.157. https://aslopubs.onlinelibrary.wiley.com/doi/abs/10.4319/lom.2009.7.157.

Stough, Joshua, Lisa Greer e Matt Benson. 2012. “Texture and Color Distribution-based Classification for Live Coral Detection”.