



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

RILEVAMENTO AUTOMATICO DI SPECIE ITTICHE NELL'ADRIATICO MEDIANTE TECNICHE DI DEEP LEARNING

Relatore:

Prof. Loris Nanni

Laureando:

Dario Mameli

1216337

ANNO ACCADEMICO 2021/2022

Sommario

Il monitoraggio dell’ambiente marino rappresenta un’attività essenziale per la comprensione dei profondi cambiamenti che stanno coinvolgendo il nostro pianeta.

In questo progetto di tesi, svolto in collaborazione con l’Università Politecnica delle Marche, viene sviluppato un software basato sull’intelligenza artificiale per il rilevamento di diverse specie marine dell’Adriatico, ittiche nello specifico, al fine di favorire un domani il processo di catalogazione delle stesse, fondamentale per l’attività di monitoraggio ambientale.

Il software si fonda sull’implementazione di un modello allo stato dell’arte per quanto riguarda il problema dell’*object detection*, uno dei temi principali nell’ambito della *computer vision*. Si tratta di una versione del noto algoritmo YOLO, basato sulle reti neurali convoluzionali (CNN), molto diffuso grazie a una combinazione vincente di velocità, accuratezza e versatilità.

Dopo un’iniziale elaborazione dei dati disponibili attraverso tecniche di *Deep Learning* quali *image preprocessing* e *data augmentation*, ad una prima fase di addestramento tramite *transfer learning* segue una fase di *testing* su un dataset di video preregistrati a largo della costa abruzzese, fornito dall’Università Politecnica delle Marche.

Indice

Sommario	I
Indice	III
Introduzione.....	1
Capitolo 1	3
Object detection.....	3
1.1 Computer vision.....	3
1.2 Object recognition.....	4
1.3 Reti neurali convoluzionali (CNN)	7
1.3.1 Convolutional layer.....	8
1.3.2 Pooling layer	10
1.3.3 Fully-Connected layer.....	11
1.4 Transfer learning	12
1.4.1 Fine-tuning.....	13
1.4.2 Riutilizzo delle features	13
Capitolo 2	15
YOLO: You Only Look Once	15
2.1 Tipologie di CNN per l'object detection.....	15
2.2 Intersection over Union (IoU).....	16
2.3 You Only Look Once (YOLO)	17
2.3.1 Rilevamento e classificazione.....	18
2.3.2 Architettura della rete	23

2.3.3	Loss function	24
2.4	YOLOv2	27
2.4.1	Accuratezza	27
2.4.2	Velocità	28
2.4.3	Solidità.....	29
2.5	YOLOv3	29
2.6	YOLOv4	30
Capitolo 3		33
Implementazione.....		33
3.1	Rete	33
3.2	Datasets.....	33
3.2.1	Training	33
3.2.2	Testing	36
3.3	Preprocessing.....	38
3.4	Data Augmentation.....	39
3.4.1	jitterColorHSV	39
3.4.2	Bilanciamento del colore.....	43
3.4.2.1	Gray World.....	44
3.4.2.2	Bradford CAT.....	45
3.4.2.3	Esempio	46
3.4.3	Aggiustamento del contrasto	48
3.4.4	Wavelet fusion.....	50
3.4.5	Ribaltamento orizzontale e riscalamento	55
3.4.6	Aumento luminosità	55
3.4.7	Training e testing.....	57
3.5	Addestramento.....	58

Capitolo 4	61
Testing e risultati	61
4.1 Metriche	61
4.1.1 Precision.....	62
4.1.2 Recall	62
4.1.3 Grafico precision-recall (PR).....	62
4.1.4 Average precision (AP)	63
4.1.5 Mean average precision (mAP)	64
4.2 Test 1	64
4.2.1 Dataset	64
4.2.2 Risultati.....	64
4.3 Test 2	66
4.3.1 Dataset	66
4.3.2 Risultati.....	66
4.4 Test 3	67
4.4.1 Dataset	67
4.4.2 Risultati.....	67
Conclusioni.....	69
Riferimenti bibliografici.....	71

Introduzione

Gli ecosistemi marini coprono il 71% della superficie terrestre, ma la nostra conoscenza relativamente alla loro risposta ai cambiamenti climatici rimane infinitesimale rispetto agli ecosistemi terrestri. Tuttavia, ci sono prove crescenti che suggeriscono che le piante e gli animali marini potrebbero rispondere tanto quanto o addirittura più velocemente delle loro controparti terrestri [1]. Monitorare questi ecosistemi diventa pertanto un'operazione di inestimabile importanza per comprendere maggiormente gli effetti del cambiamento climatico sul nostro pianeta.

A tal scopo, in questo lavoro viene sviluppato uno strumento software che sia in grado di aiutare nel processo di monitoraggio della fauna marina, con particolare attenzione alle specie di pesce, inquadrandoli visivamente nell'immagine. In termini tecnici, viene sviluppato un *detector* mediante ausilio di potenti strumenti di *Deep Learning*, quali le reti neurali, che effettui il rilevamento di oggetti, in inglese *object detection*, relativamente a tali specie di pesce.

Il software viene in particolare sviluppato per un'attività di monitoraggio dell'ambiente marino, svolta dall'Università Politecnica delle Marche a largo della costa abruzzese, nell'Adriatico Centrale, presso una stazione localizzata sul fondo a 17 metri di profondità. L'obiettivo è quello di preparare il software per il rilevamento delle specie ittiche catturate dai video registrati dalla stazione.

Il presente lavoro si struttura in quattro capitoli principali. Nel primo capitolo viene presentato il problema dell'*object detection*, collocandolo nel panorama più ampio della *computer vision*, e ancora dell'intelligenza artificiale, andando a prendere in esame gli strumenti più validi per la sua implementazione, ossia le reti neurali convoluzionali, espandendo inoltre sulle migliori tecniche di addestramento delle stesse. Nel secondo capitolo viene presentato uno degli algoritmi allo stato dell'arte per l'*object detection*, ossia *You Only Look Once* (YOLO), trattandone le caratteristiche sia strutturali che funzionali, e andando a fare una disamina delle prime quattro versioni della rete. Nel terzo capitolo viene invece passata in rassegna l'effettiva implementazione dell'algoritmo, descrivendo i datasets utilizzati e successivamente

il flusso di esecuzione della rete, analizzando le principali tecniche di *Deep Learning* attuate. Nel quarto capitolo vengono riportati i test sviluppati e relativi risultati.

In un'ultima breve conclusione si effettua una sintetica analisi comparativa dei risultati e si elaborano delle proposte di sviluppo futuro per il progetto.

Capitolo 1

Object detection

In questo primo capitolo l'obiettivo è quello di formalizzare il problema dell'*object detection* (o rilevamento di oggetti), su cui si articola il presente lavoro di tesi. Vengono introdotti concetti cardine per lo studio dell'argomento, quali *computer vision*, *object recognition*, CNNs e *transfer learning*.

1.1 Computer vision

La “**computer vision**”, o visione artificiale, è un campo dell'intelligenza artificiale che permette ai computer e in generale ai sistemi di elaborazione di processare immagini digitali, video e altri input visivi, ed estrarne informazioni significative per intraprendere azioni o effettuare qualche tipo di segnalazione. Se l'IA in senso generale permette ai computer di pensare, la *computer vision* permette loro di vedere, osservare e capire. [8]

L'obiettivo della *computer vision* è pertanto quello di simulare il comportamento della vista umana, al fine di sostituire l'uomo nelle operazioni più monotone e ripetitive in contesto lavorativo o più in generale nella vita quotidiana.

La vista umana, tuttavia, ha il vantaggio di disporre di anni e anni di esperienza in cui si è allenata a distinguere gli oggetti, capire quanto sono lontani, se si stanno muovendo e se c'è qualcosa di sbagliato in un'immagine.

Inoltre, i sistemi di *computer vision*, ad esempio quelli per il monitoraggio real-time, devono in genere svolgere elaborazioni in tempi molto brevi, adoperando dati e algoritmi piuttosto che retine, nervi ottici e corteccia visiva.

Naturalmente, per raggiungere questo obiettivo è necessaria sia un'elevata potenza computazionale, sia il consumo di enormi quantità di dati. In questo modo, la *computer vision*

può superare rapidamente le capacità umane. Risulta quindi evidente come il Deep Learning rappresenti uno strumento chiave per risolvere problemi legati alla *computer vision*.

Fra i vari ambiti in cui si declina la *computer vision*, l'*object recognition*, o riconoscimento di oggetti, ricopre un ruolo chiave.

1.2 Object recognition

“**Object recognition**”, o riconoscimento di oggetti, è una tecnica di identificazione di uno o più oggetti all’interno di immagini e video. Può essere suddiviso in più sottocategorie:

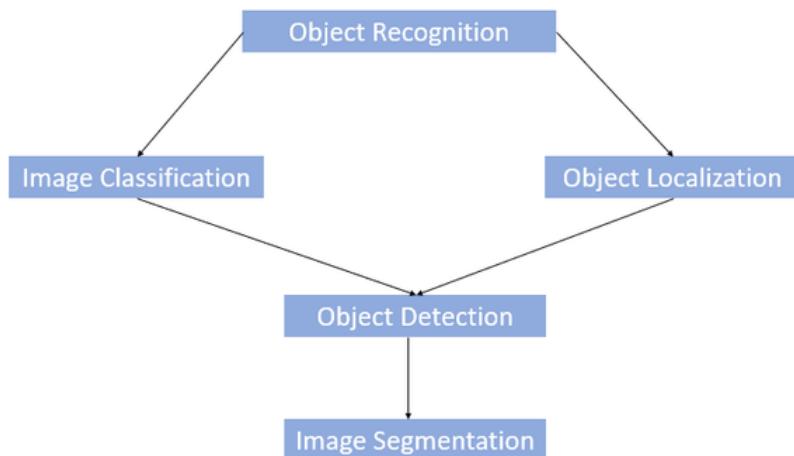


Figura 1.1: Schema dei problemi dell’object recognition [9]

- **Image classification** (Classificazione dell’immagine):

Si assegna una classe (o più) all’immagine in base ad una data metrica (esempio probabilità)



Figura 1.2: Esempio di image classification [9]

- **Object localization** (localizzazione dell'oggetto):

Si tratta di determinare la posizione dell'oggetto all'interno dell'immagine mediante l'uso di una *bounding box* (BB), ossia un rettangolo che inquadra l'oggetto. Generalmente combinata all'operazione di classificazione dell'immagine.



Figura 1.3: Esempio di object classification & localization [9]

- **Object detection** (rilevamento di oggetti):

Tutti gli oggetti di una data classe (o più) presenti nell'immagine vengono classificati e localizzati mediante l'uso delle *bounding boxes*.

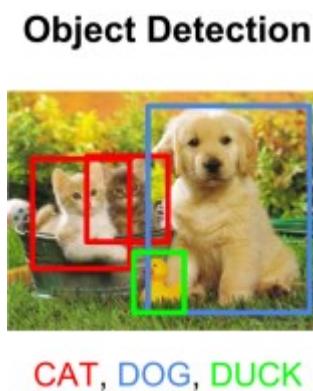


Figura 1.4: Esempio di object detection [9]

Più nello specifico, si compone dei seguenti passi:

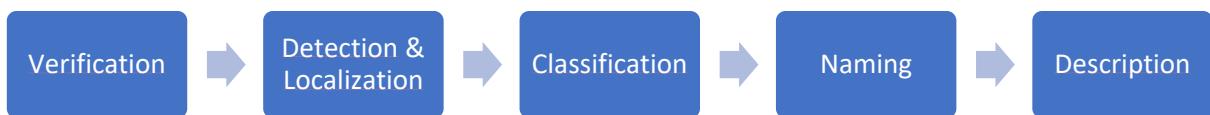


Figura 1.5: Diagramma di flusso dell'object detection [10]

- 1) *Verification*: si verifica la presenza degli oggetti all'interno dell'immagine.
- 2) *Detection & Localization*: si effettua il rilevamento e la localizzazione degli oggetti nell'immagine.
- 3) *Classification*: si effettua la classificazione degli oggetti.
- 4) *Naming*: vengono assegnate le *label* (etichette) con le informazioni relative, come punteggio e classe, a ciascun oggetto nell'immagine, mediante uso di *bounding boxes*.
- 5) *Description*: vengono introdotte eventuali descrizioni aggiuntive relativamente ad azioni o relazioni fra gli oggetti nel contesto dell'immagine.

- **Image segmentation** (segmentazione dell'immagine):

Estensione del rilevamento che si occupa di localizzare ciascun oggetto mediante una maschera di pixel. La granularità è maggiore, permettendo infatti di identificare anche la forma dell'oggetto in base ai pixel che la compongono.

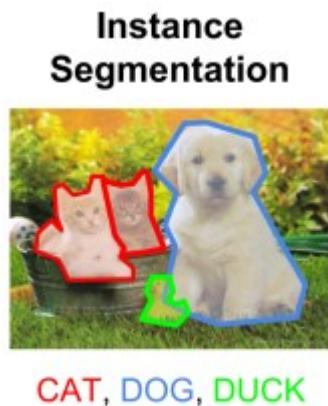


Figura 1.6: Esempio di instance segmentation, uno dei metodi di segmentazione [9]

Come già espresso, per risolvere problemi di questo tipo risulta particolarmente conveniente adoperare strategie di Deep Learning. In questo senso, le reti neurali maggiormente utilizzate sono le reti neurali convoluzionali (CNNs - *convolutional neural networks*), le cui caratteristiche strutturali le rendono particolarmente adatte a risolvere problemi che richiedono la manipolazione di immagini come tipo di dato.

1.3 Reti neurali convoluzionali (CNN)

Una **rete neurale convoluzionale (ConvNet / CNN)** è un tipo di rete neurale profonda (DNN) costituita da tre tipi di livelli: *convolutional*, *pooling* e *fully-connected* (FC).

Questi livelli sono organizzati in volumi, ossia raggruppamenti tridimensionali di neuroni.

I livelli intermedi sono deputati all'apprendimento delle *features* dell'immagine, ossia le sue caratteristiche. Gli ultimi livelli servono invece per la classificazione.

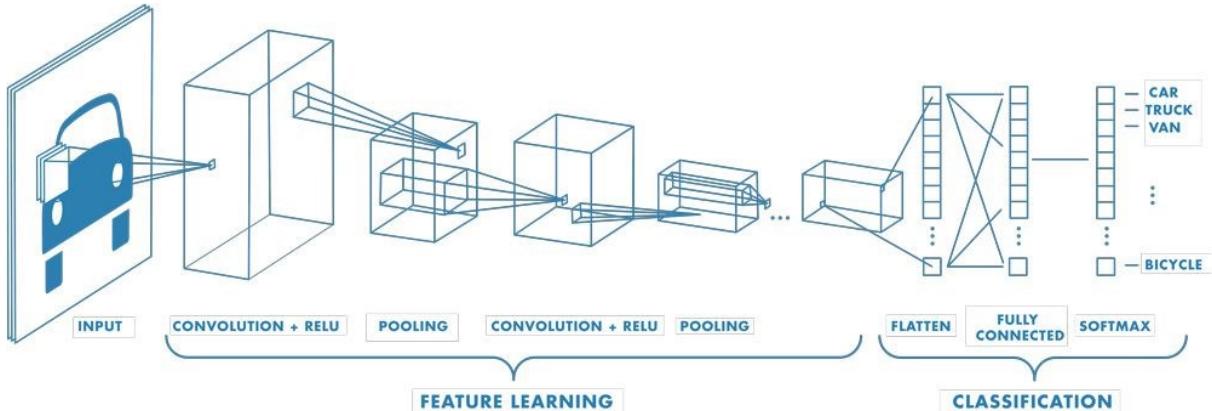


Figura 1.7: Esempio di struttura di una CNN [12]

L'architettura di una CNN è analoga a quella del modello di connettività dei neuroni nel cervello umano ed è stata ispirata dall'organizzazione della corteccia visiva. I singoli neuroni rispondono agli stimoli solo in una regione ristretta del campo visivo nota come **“receptive field”** (o “campo ricettivo”). L'insieme di tali campi permette di coprire l'intera area visiva.

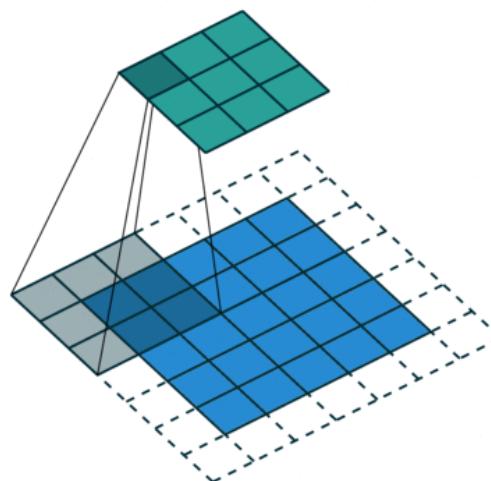


Figura 1.8: Visualizzazione esemplificativa del campo visivo di un neurone, quest'ultimo rappresentato da un singolo quadrato [11]

Una CNN effettua elaborazioni attraverso l'applicazione di *filtri digitali* (maschere di pesi) pertinenti. Questi filtri determinano il fatto che i neuroni di un dato livello sono connessi solo ad alcuni neuroni del livello precedente; dunque, ogni neurone è frutto di un'elaborazione locale. Inoltre, i pesi stessi sono condivisi fra tutti i neuroni del livello.

Nel complesso, questi due fattori determinano il fatto che i neuroni di un dato livello sono in grado di rilevare una stessa feature in diverse aree dell'input. Quindi, nel caso di un'immagine, rilevare un dato oggetto diventa indipendente, ad esempio, dalla sua posizione all'interno dell'immagine. [12] Una CNN è infatti in grado di catturare le dipendenze spaziali e temporali nelle immagini, e questo determina il fatto che l'architettura della CNN si adatti meglio a dataset di immagini.

Vengono di seguito discussi i tre livelli principali costitutivi di una CNN.

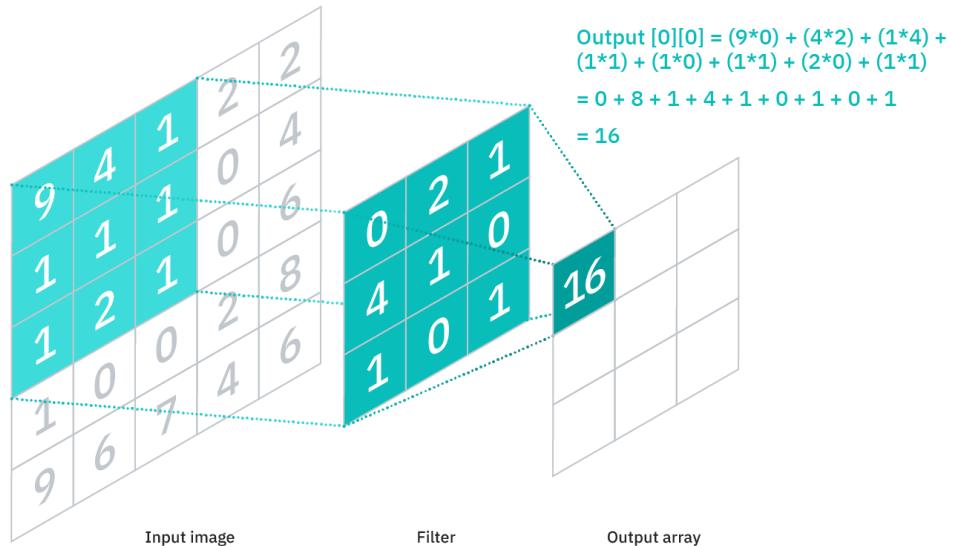
1.3.1 Convolutional layer

L'obiettivo dell'operazione di convoluzione è di estrarre le caratteristiche principali dall'immagine di input. Per fare ciò, le CNN sono costituite da numerosi livelli convoluzionali. Conventionalmente, il primo **ConvLayer** è responsabile dell'acquisizione delle caratteristiche di basso livello come bordi, colore, orientamento del gradiente, ecc. Con livelli aggiuntivi, l'architettura si adatta anche alle funzionalità di alto livello, sofisticando ad ogni livello la propria comprensione dell'immagine, similmente a quanto avviene con la vista umana.

Il livello convoluzionale è dunque l'elemento costitutivo principale di una CNN ed è dove si verifica la maggior parte del calcolo. Esso richiede tre componenti, ovvero un volume di input, un filtro e una matrice di output.

Il **filtro** è una struttura 3D composta da più *kernel* sovrapposti. Un **kernel** è un *array* 2D di pesi. Il filtro viene applicato a un'area dell'immagine e viene calcolato il prodotto scalare tra i pixel di input e il filtro. Il risultato viene quindi inserito nella matrice di output. Per effettuare l'operazione di prodotto scalare, esiste una relazione tra input e filtro che vincola la profondità (*depth*) di quest'ultimo: il filtro deve avere la stessa profondità dell'input.

Ad esempio, se l'immagine è in scala di grigi (*grayscale*), il filtro dovrà essere costituito da un singolo kernel e il prodotto scalare si svilupperà come in figura:

Figura 1.9: Esempio di prodotto scalare con immagine *grayscale* [13]

Se invece l'immagine è in RGB, il filtro dovrà essere costituito da 3 kernel, in quanto la profondità di bit è pari a 3, e il prodotto scalare si svilupperà come in figura:

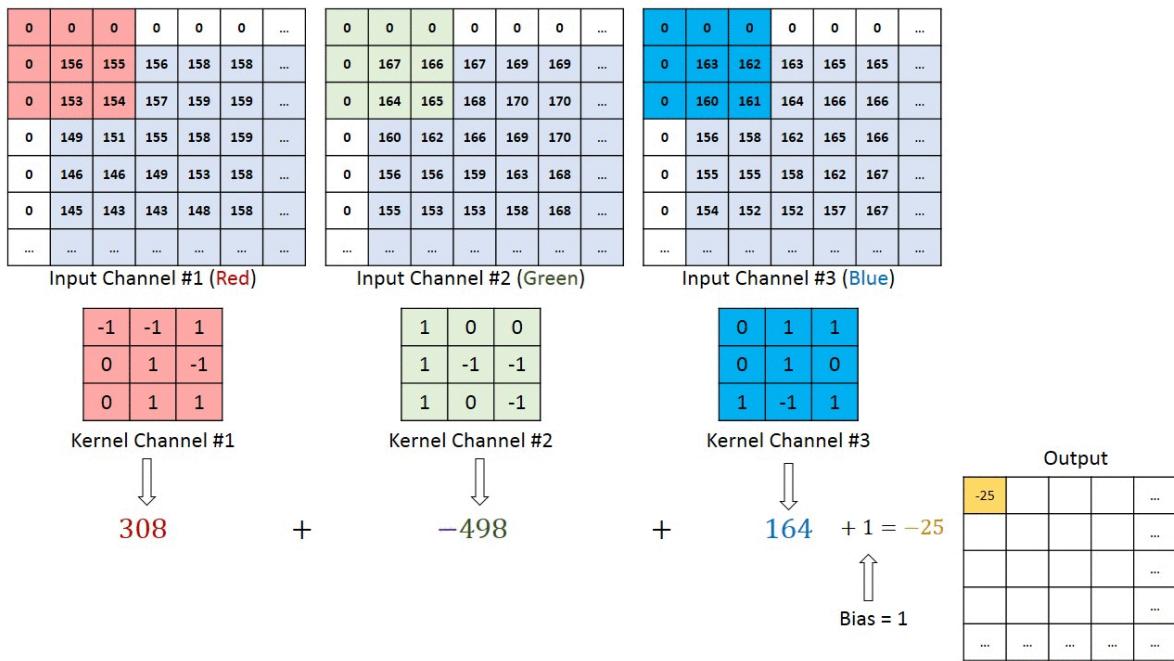


Figura 1.10: Esempio di prodotto scalare con immagine RGB [11]

Successivamente, il filtro si sposta di un passo, ripetendo il processo fino a quando non ha attraversato l'intera immagine.

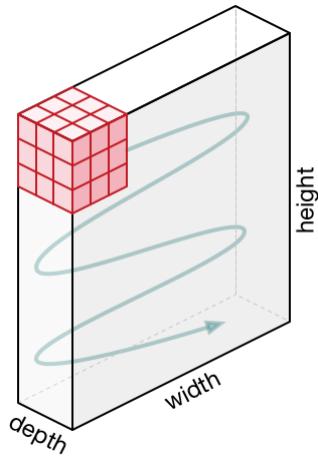


Figura 1.11: Scorrimento del filtro sul volume di input [11]

La successione di prodotti scalari tra l'input e il filtro costituisce la convoluzione.

La matrice di output finale, prodotta dalla convoluzione tra l'input e il filtro, è nota come “**feature map**”, o “mappa delle features”.

L'insieme delle feature maps costituisce la profondità del volume di output. Dunque, la profondità del volume di output sarà pari al numero di filtri applicati.

La larghezza (*width*) è invece determinata dalla seguente formula:

$$W_{out} = \frac{W_{in} - F_w + 2 \cdot Padding}{Stride} + 1 \quad (1.1)$$

Dove W_i è la larghezza del volume i, F_w è la larghezza del filtro F, *Padding* è lo spessore in pixel del bordo, *Stride* è lo scostamento in pixel del filtro sul volume di input.

Equivalentemente è determinata l'altezza (*height*).

1.3.2 Pooling layer

Il **pooling layer** (o livello di aggregazione), noto anche come *downsampling* (o sottocampionamento), esegue la riduzione di dimensionalità, riducendo il numero di parametri nell'input. Nonostante questo determini la perdita di molte informazioni, tale livello comporta una serie di vantaggi per la CNN. Aiuta infatti a ridurne la complessità computazionale, e migliorarne l'efficienza riducendo la dimensione della *feature map* di input.

Analogamente al livello convoluzionale, l'operazione di aggregazione scorre un filtro sull'intero input, ma la differenza è che questo filtro non ha pesi. Invece, il filtro applica una funzione di aggregazione, popolando l'array di output con valori che seguono principalmente due regole:

- **Max pooling:** quando il filtro si sposta attraverso l'input, seleziona il pixel con il valore massimo da inviare all'array di output.
- **Average pooling:** quando il filtro si sposta attraverso l'input, calcola il valore medio all'interno del campo ricettivo da inviare all'array di output.

Max pooling è di gran lunga il più utilizzato fra i due. Max Pooling, infatti, funziona anche come soppressore del rumore: elimina del tutto le attivazioni rumorose, generalmente associate a valori bassi, eseguendo così anche il *de-noising* insieme alla riduzione della dimensionalità.

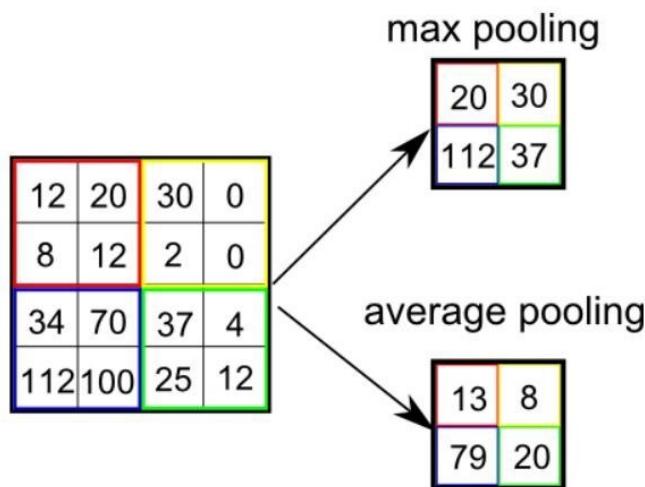


Figura 1.12: Illustrazione di max e average pooling. Filtro 2x2 viene fatto scorrere con stride = 2 sull'input 4x4 determinando un output 2x2 [11]

1.3.3 Fully-Connected layer

Come si desume dal nome stesso, nel livello completamente connesso (**FC**) ogni neurone del livello di output si connette direttamente ad ogni neurone del livello precedente.

Questo livello esegue il compito di classificare i *pattern*, ossia i tipi di dati, elaborati dalla rete in base alle relative features, estratte ai livelli precedenti.

Mentre i livelli convoluzionali e di pooling tendono a utilizzare le funzioni di attivazione ReLU, i livelli FC, di solito, sfruttano invece *softmax*, interpretando i valori di output come probabilità utili per la classificazione.

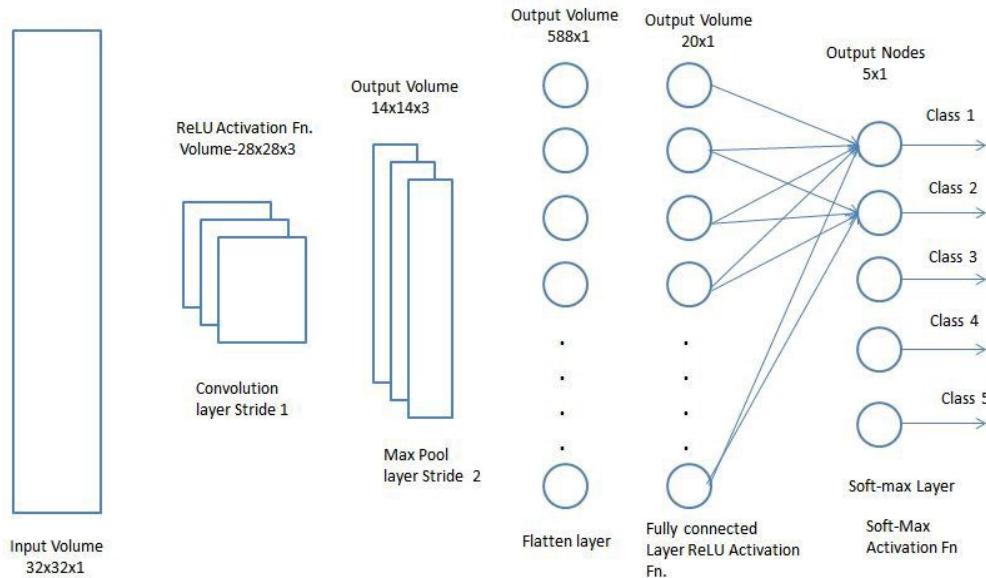


Figura 1.13: Esempio di CNN con particolare attenzione al livello FC finale

1.4 Transfer learning

In generale, l’addestramento di una CNN su dataset di grandi dimensioni, a partire da pesi inizializzati con valori casuali, richiede una quantità di tempo molto elevata. Se l’hardware a disposizione è limitato a singole GPU, come nel caso di utenti standard, l’addestramento potrebbe richiedere anche interi giorni o settimane. In questo modo, dunque, l’utilizzo delle CNN sarebbe limitato solo a chi dispone di macchine molto potenti, da più GPU in parallelo a interi *clusters* di calcolatori, fino al *cloud computing*.

Per ovviare a questo problema, e rendere lo strumento delle reti neurali più accessibile, è possibile sfruttare un tipo di meccanismo chiamato “**transfer learning**” che consente di ridurre drasticamente i tempi di addestramento anche con risorse computazionali più contenute.

Nel *transfer learning* si fa uso di reti pre-addestrate su problemi simili a quello di interesse, in particolare addestrate in precedenza da macchine molto potenti su enormi quantità di dati, in modo che i pesi della rete siano già tarati per ottenere prestazioni migliori rispetto a pesi inizializzati in modo casuale, come generalmente avviene per reti appena create.

Il *transfer learning* si può implementare principalmente seguendo *fine-tuning* oppure estraendo le features ai livelli intermedi della rete per addestrare un classificatore esterno.

1.4.1 Fine-tuning

Partendo dalla rete pre-addestrata, si operano i seguenti passaggi:

1. Si ridimensionano i patterns di input per renderli compatibili con il primo livello della rete.
2. Si rimpiazza il livello di output *softmax* con un nuovo livello, adeguandolo in base al numero di classi disponibili.
3. Si mantengono i pesi attuali in tutti i livelli tranne quello finale, dove sono inizializzati casualmente.
4. (Tendenzialmente) Si mantiene un learning rate basso, non volendo modificare i pesi eccessivamente e troppo velocemente, essendo già di buona qualità.
5. (A volte) Si bloccano i pesi dei primi livelli della rete, essendo deputati all'estrazione delle features universali per il dato problema.
6. Si effettua l'addestramento.

Ulteriori informazioni sono disponibili al [\[6\]](#).

1.4.2 Riutilizzo delle features

Partendo dalla rete pre-addestrata si estraggono ai suoi livelli intermedi le features generate dalla *forward propagation* dei patterns di input, ossia il processamento degli stessi e il passaggio in avanti di livello in livello delle informazioni, e si addestra un classificatore esterno per effettuare la classificazione dei patterns del problema, come *Support Vector Machine* (SVM). [\[7\]](#)

Solitamente, i livelli intermedi cui si fa riferimento sono quelli più vicini all'output, ma è possibile anche fermarsi a quelli più lontani da esso, che generalmente tuttavia processeranno patterns con dimensionalità elevata, tendenzialmente da ridurre mediante apposite tecniche.

Capitolo 2

YOLO: You Only Look Once

2.1 Tipologie di CNN per l'object detection

Riassumendo la definizione di *object detection* data al §1.2, si può dire che esso sia la procedura per determinare la classe e posizione all'interno di un'immagine o video di tutti gli oggetti di interesse, mediante l'uso di *bounding boxes*.

In base anche a quanto visto riguardo le CNN al §1.3, risulta naturale immaginare che queste possano fornire uno strumento molto potente ed efficiente per risolvere problemi legati all'*object detection*.

L'obiettivo è ora dunque quello di comprendere come le CNN possano essere utilizzate per implementare un algoritmo di *object detection*.

Innanzitutto, può essere interessante delineare le tappe fondamentali dello sviluppo dei modelli di *object detection* degli ultimi 20 anni. Nella figura seguente è possibile osservare la relativa linea temporale:

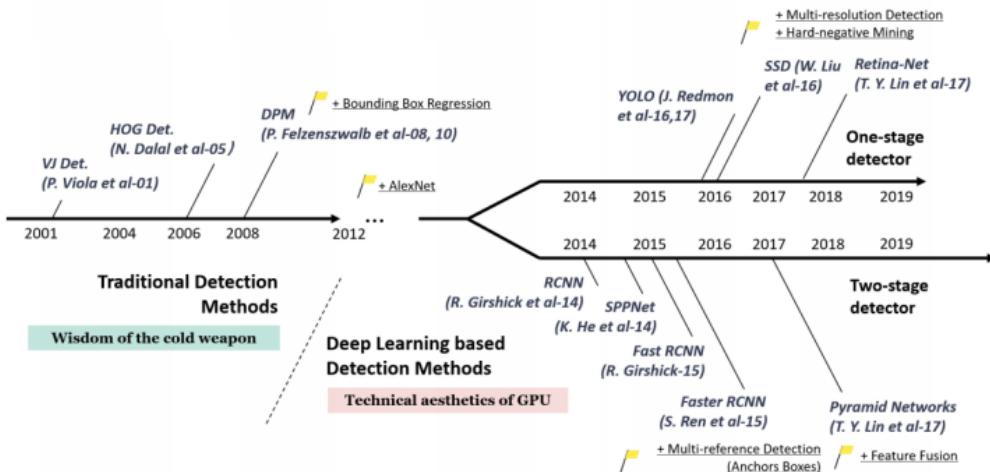


Figura 2.1: Linea del tempo dell'evoluzione dei modelli di object detection [14]

Come si può osservare dalla Figura 2.1, con l'avvento del deep learning si suole suddividere i cosiddetti *detectors*, ossia le reti neurali deputate all'*object detection*, in due categorie:

- **Two-stage detectors:** caratterizzati da elevata accuratezza nella localizzazione e nel riconoscimento di oggetti. Il primo stadio si separa dal secondo grazie a un livello di *RoI Pooling*, ossia un livello di aggregazione delle regioni di interesse che propone determinate regioni (sezioni di immagine) in base alla loro probabilità di contenere un oggetto di interesse. Queste regioni prendono il nome di *region proposals*, o “regioni proposte”. Nel secondo stadio vengono estratte le features delle regioni proposte e viene effettuata la classificazione.
- **One-stage detectors:** il processo di rilevazione avviene in un singolo stadio, in quanto viene saltato il passo di *RoI Pooling* e dunque le *bounding boxes* vengono predette direttamente sulle immagini di input. Questo consente un notevole risparmio di tempo e una possibile applicazione in sistemi *real-time*.

Tra i *two-stage detectors*, particolare attenzione va all'evoluzione dei modelli *R-CNN*, *Fast R-CNN* e *Faster R-CNN*, che hanno contribuito in maniera sostanziale al progresso del software nell'ambito del rilevamento di oggetti. Per approfondimenti in merito è possibile consultare [15] e [16].

Fra i *one-stage detectors*, invece, ha un ruolo di primo piano il famoso detector *You Only Look Once* (YOLO). Prima di trattarlo esaustivamente però, è opportuno introdurre un concetto fondamentale, ossia *Intersection over Union* (IoU).

2.2 Intersection over Union (IoU)

Gli outputs di un detector che riceve come input un'immagine I possono essere espressi come l'insieme delle triplette $O = \{(b_j, c_j, p_j)\}_j$, dove j è un dato oggetto, b_j è la sua BB (*bounding box*), c_j la sua classe predetta, p_j la relativa “confidence”, ossia la stima dell'incertezza sulla previsione.

Fissato j , data una tripletta $(b, c, p) \in O$, si definisce “**Intersection over Union**” (IoU), o intersezione su unione, relativamente alla coppia (b, b_c) , con b_c rappresentante la corretta BB dell'oggetto, o *ground truth box*, nel seguente modo:

$$IOU(b, b_c) = \frac{area(b \cap b_c)}{area(b \cup b_c)}$$

(2. 1)

Si può visualizzare il concetto in figura:

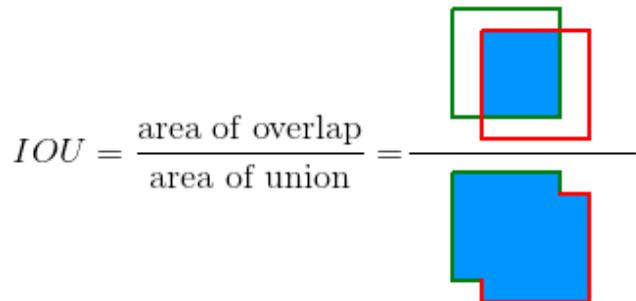


Figura 2.2: Visualizzazione di IoU [23]

Questa metrica è utile per calcolare l'accuratezza di localizzazione. Una soglia comune per determinare la qualità della BB predetta è 0.5.

Si riporta in seguito un esempio:

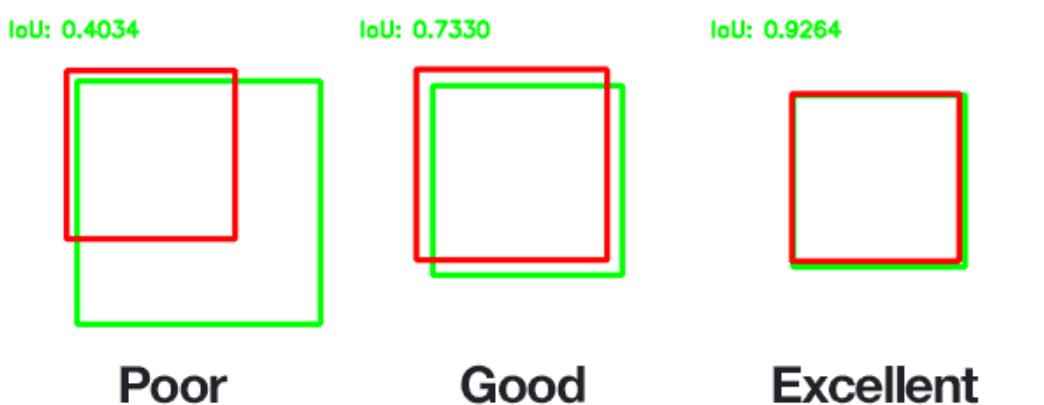


Figura 2.3: Esempio di tre IoU calcolate su tre diverse BB [24]

2.3 You Only Look Once (YOLO)

Nel panorama degli algoritmi per l'*object detection*, pochi riescono a raggiungere un compromesso di velocità e precisione della *detection* valido come “**You Only Look Once**” (**YOLO**), introdotto il 9 Maggio 2016 con la pubblicazione dell’articolo “*You Only Look Once: Unified, Real-Time Object Detection*”, di Joseph Redmond et al. [22]

In quanto detector a singolo stadio, YOLO è in grado di predire le *bounding boxes* e probabilità delle classi dei vari oggetti presenti nell'immagine in un unico passaggio, accelerando notevolmente la fase di *forward-propagation*, a scapito di una piccola perdita in precisione.

2.3.1 Rilevamento e classificazione

I due passaggi generalmente distinti di rilevamento dell'oggetto e classificazione vengono unificati: la rete, infatti, sfrutta le *features* dell'intera immagine per localizzare le *bounding boxes* di tutti gli oggetti di tutte le classi simultaneamente, ed è pertanto in grado di “ragionare” sull'immagine a livello globale, invece che in base alle singole regioni e ai singoli oggetti.

Entrando nel dettaglio, le immagini passate come input vengono divise in una griglia di $S \times S$ celle, dove ognuna di esse può contribuire al rilevamento di uno e uno solo oggetto. In particolare, se il centro di tale oggetto è contenuto nella cella, essa viene ritenuta “responsabile” per il rilevamento di tale oggetto (si veda al §2.2.3 per meglio comprenderne il significato).

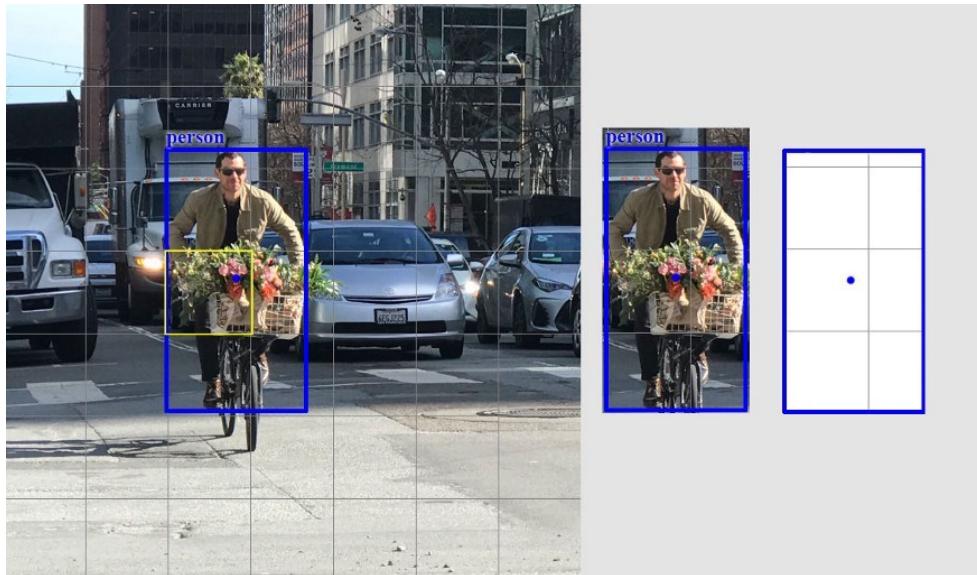


Figura 2.4: Griglia $S \times S$ con $S=7$; Cella gialla sarà “responsabile” per l’oggetto “persona”, in quanto il centro dell’oggetto (pallino blu) è contenuto in essa [25]

Ciascuna cella predice B *bounding boxes*, ognuna con un proprio punteggio, definito *box confidence score*.

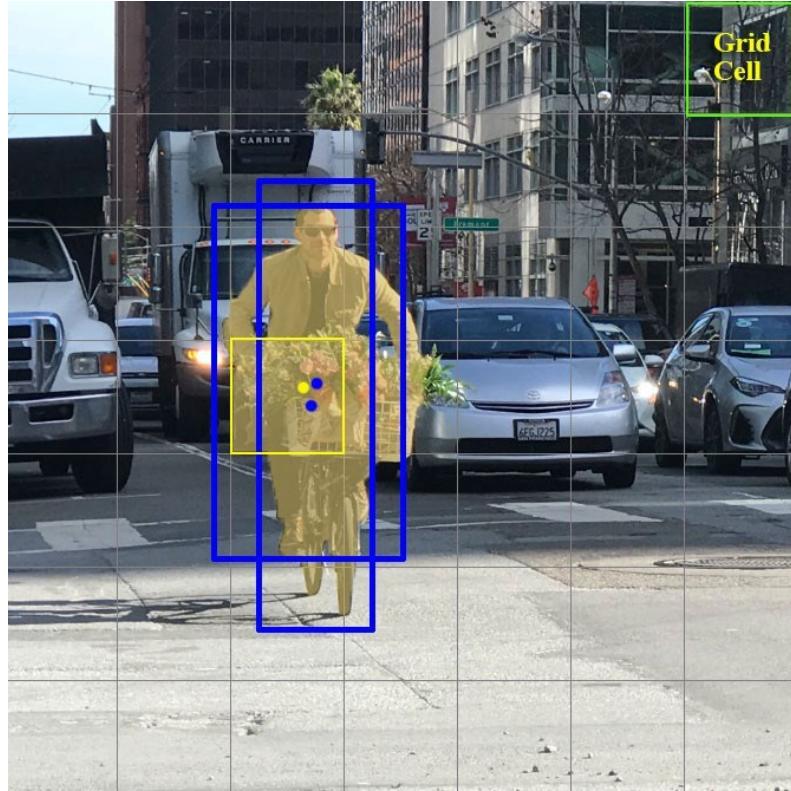


Figura 2.5: Considerando $B=2$, la cella gialla predice due *bounding boxes* (in blu) [25]

Il *box confidence score* riflette quanto sicuro è il modello della presenza di un oggetto all'interno della data *bounding box*, e della localizzazione della *bounding box* rispetto all'oggetto. In particolare, si definisce la probabilità che una data *bounding box* contenga un oggetto come $P(\text{Object})$ e la precisione di localizzazione della stessa come $IOU(b_j, b_c)$, con b_j j-esima *bounding box* predetta e $j = 1, \dots, B$, e b_c *bounding box* corretta. Il *box confidence score* di b_j ($conf_{b_j}$) è dato dal prodotto delle due espressioni, dunque:

$$conf_{b_j} := P(\text{Object}) \cdot IOU(b_j, b_c) \quad (2.2)$$

In definitiva, una *bounding box* (b_j) può essere rappresentata da una sequenza di cinque valori nel seguente modo:

$$b_j := (x_j, y_j, w_j, h_j, conf_{b_j}) \quad (2.3)$$

Con riferimento alla (2.3):

- (x_j, y_j) rappresentano le coordinate del centro della *bounding box* j rispetto ai bordi della cella della griglia.
- (w_j, h_j) rappresentano rispettivamente la larghezza (*width*) e l'altezza (*height*) normalizzate della *bounding box* j , ossia relativamente all'intera immagine.
- $conf_{b_j}$ è definito come alla (2.2).

In aggiunta, ogni cella della griglia SxS determina per ciascuna classe C_i , con $i = 1, \dots, C$ classi totali, la probabilità che l'oggetto rilevato appartenga a quella classe, definita come la probabilità condizionata $P(C_i|Object)$.

A questo punto è possibile definire il *class confidence score* di ogni classe, per ogni *bounding box*, come:

$$conf_{C_i,b_j} := conf_{b_j} \cdot P(C_i|Object) \quad (2.4)$$

Ossia, sostituendo la (2.2):

$$conf_{C_i,b_j} = P(Object) \cdot IOU(b_j, b_c) \cdot P(C_i|Object)$$

Che produce, per definizione di probabilità condizionata:

$$conf_{C_i,b_j} = P(C_i) \cdot IOU(b_j, b_c) \quad (2.5)$$

Dunque, il *class confidence score* di classe rappresenta un valore che codifica sia la probabilità che una data classe C_i sia associata ad una *bounding box* b_j , indipendentemente dall'oggetto cui fa riferimento, sia la precisione di localizzazione di quest'ultima. Il *class confidence score*, naturalmente, consente di determinare la classe della *bounding box* riferita, ossia quella che ha ottenuto il punteggio massimo.

Si osservi ora l'esempio in figura, che rappresenta il flusso di esecuzione dell'algoritmo descritto finora:

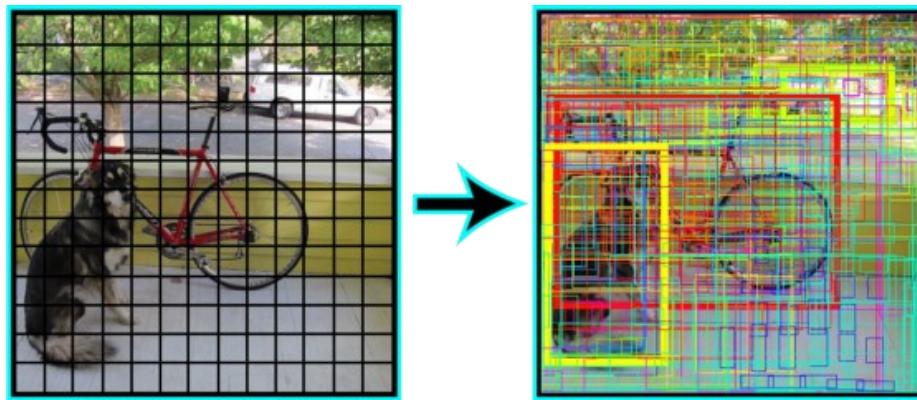


Figura 2.6: Predizione delle *bounding boxes* [25]

Con riferimento alla Figura 2.6, si può osservare la predizione delle *bounding boxes* a partire dalla griglia iniziale. Considerando che per ogni cella della griglia vengono generate B *bounding boxes*, è naturale assumere che buona parte di esse risultino ridondanti nella stima della posizione dell'oggetto cui si riferiscono. Per ridurre il numero di *bounding boxes* nell'immagine, migliorando la precisione di localizzazione dell'oggetto, e allo stesso tempo alleggerendo l'inferenza della rete, che dovrà infatti così processare meno elementi ai livelli successivi, viene applicato un processo chiamato *non-maximum suppression*.

Per ogni classe, “**Non-maximum suppression**” consente di fatto di eliminare tutte le *bounding boxes* relative a quella classe che hanno IoU sufficientemente elevato con una qualunque *bounding box* con punteggio di classe maggiore.

Viene riportato in seguito un possibile pseudocodice.

NON-MAXIMUM SUPPRESSION	
Input:	$b_j, conf_{C_i, b_j}, i = 1, \dots, C, j = 1, \dots, B$
	θ soglia di soppressione
Output:	Lista di bounding boxes filtrate
1	Old_BB $\leftarrow \{b_j\}$
2	New_BB $\leftarrow \{ \}$
3	while Old_BB non vuota
4	current $\leftarrow \underset{b_j \in Old_BB}{\operatorname{argmax}} \{conf_{C_i, b_j}\}$
5	rimuovi current da Old_BB
6	New_BB $\leftarrow current$
7	for $b_j \in Old_BB$
8	if $class(b_j) = class(current)$ e $IoU(b_j, current) > \theta$ then
9	rimuovi b_j da Old_BB
10	end if
11	end for
12	end while
13	return New_BB

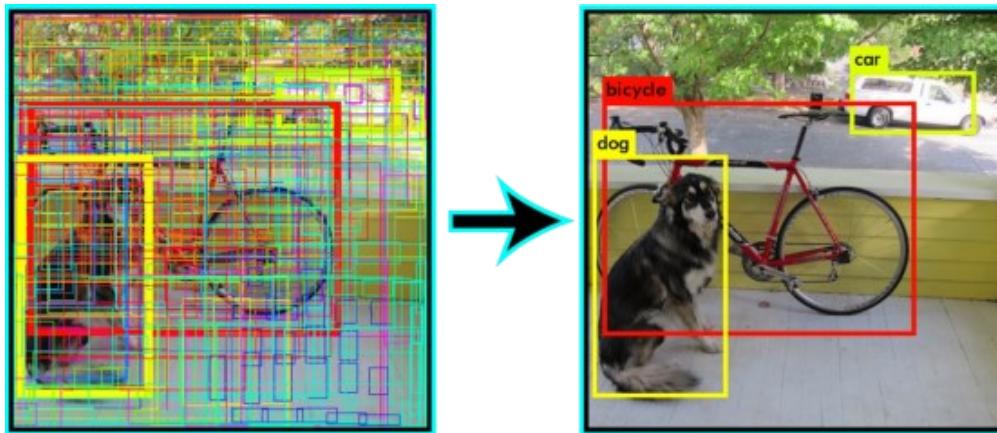


Figura 2.7: Output di non-max suppression [25]

2.3.2 Architettura della rete

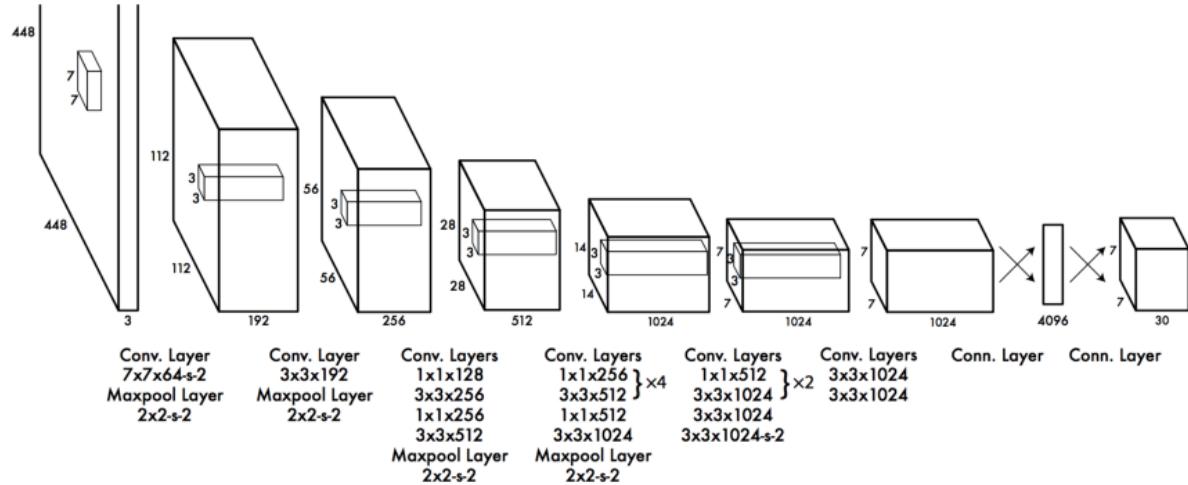


Figura 2.8: Schema dell'architettura di rete [22]

La rete proposta al [22] ha 24 livelli convoluzionali (§1.3.1) talvolta congiunti a livelli di max pooling (§1.3.2), seguiti da 2 livelli completamente connessi (§1.3.3). Alcuni livelli di convoluzione si alternano a livelli di riduzione 1×1 [23] per ridurre la profondità delle *feature maps*. L'ultimo livello di convoluzione emette un tensore con forma (7, 7, 1024) successivamente appiattito. Utilizzando 2 livelli completamente connessi come forma di regressione lineare, emette infine un tensore (7, 7, 30), che corrisponde a 2 previsioni di *bounding box* per cella della griglia SxS. Infatti, il tensore di output è codificato dalla formula:

$$y = (S, S, B \times 5 + C) \quad (2.6)$$

Per la rete descritta al [22], vengono considerati $S=7$, $B=2$, $C=20$.

Esiste inoltre una versione più veloce ma meno accurata di YOLO, chiamata Fast YOLO, la quale utilizza solo 9 livelli convoluzionali con *feature maps* meno profonde.

Come **funzione di attivazione**, funzione che codifica l'output di una serie di neuroni di input, viene utilizzata una funzione di attivazione lineare per l'ultimo livello, mentre per gli altri viene usata Leaky ReLU, nella versione che segue:

$$\phi(x) = \begin{cases} x, & \text{se } x > 0 \\ 0.1x, & \text{altrimenti} \end{cases} \quad (2.7)$$

2.3.3 Loss function

In generale, una “**loss function**”, letteralmente “funzione di perdita” o più propriamente “funzione di costo”, è una funzione che calcola la distanza tra l’output corrente dell’algoritmo e l’output previsto. È un metodo per valutare il modo in cui l’algoritmo elabora i dati, ed è utile per tarare i parametri della rete al fine di migliorare l’output.

Per semplificare il calcolo della *loss function*, producendo l’algoritmo in output più *bounding boxes* per ciascuna cella della griglia, per ogni oggetto da rilevare si definisce una e una sola *bounding box* come “responsabile” per esso. Con “responsabile” si intende l’unica *bounding box* che contribuirà nel calcolo della *loss*, selezionata come quella con il più alto IoU con la *bounding box* corretta.

La *loss function* scelta per la rete è “**sum of squares error**” (SSE), o somma dei quadrati residui, tra il valore predetto e il valore effettivo. Essa è in particolare composta da *classification loss*, *localization loss* e *confidence loss*.

1. Classification loss

Stima l’errore di classificazione di un oggetto sulla base della corrispondenza di classe fra quella predetta e quella effettiva. Per ogni cella, la *loss* viene calcolata come l’errore quadratico delle probabilità condizionate di classe, per ogni classe:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (2.8)$$

Dove $\mathbb{1}_i^{obj}$ vale 1 se un oggetto è contenuto nella *i-esima* cella, 0 altrimenti.

$\hat{p}_i(c)$ rappresenta la probabilità condizionata per la classe c nella *i-esima* cella.

2. Localization loss

Misura l’errore nella predizione delle *bounding box*, ovvero dell’altezza, della larghezza e delle coordinate del centro, ma solo per quelle responsabili per un dato oggetto. La formula della *loss* si articola nel seguente modo:

$$\begin{aligned}
& \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]
\end{aligned} \tag{2.9}$$

Dove $\mathbb{1}_{ij}^{obj}$ vale 1 se la j -esima bounding box della i -esima cella è responsabile per l'oggetto, 0 altrimenti.

λ_{coord} è un parametro che aumenta il peso della *localization loss* rispetto alle altre; il suo valore di default è 5.

Si osservi come vengono usate le radici delle larghezze ed altezze delle *bounding boxes*, per evitare che un errore relativo ad una *bounding box* maggiore pesi allo stesso modo dello stesso errore su una *bounding box* minore. E.g. un errore di 2 pixel deve pesare maggiormente su una *bounding box* di 30 pixel piuttosto che su una di 300 pixel.

3. Confidence loss

Valuta l'errore nella stima della probabilità di presenza dell'oggetto nella specifica regione di interesse. Si calcola distinguendo due possibilità:

- Oggetto rilevato nella *bounding box*:

$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_{ij} - \hat{C}_{ij})^2 \tag{2.10}$$

Dove $\mathbb{1}_{ij}^{obj}$ vale 1 se la j -esima bounding box della i -esima cella è responsabile per l'oggetto, 0 altrimenti.

\hat{C}_{ij} è il *box confidence score* della j -esima bounding box della i -esima cella.

- Oggetto non rilevato nella *bounding box*:

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_{ij} - \hat{C}_{ij})^2 \quad (2.11)$$

Dove $\mathbb{1}_{ij}^{\text{noobj}}$ è il complementare di $\mathbb{1}_{ij}^{\text{obj}}$.

\hat{C}_{ij} è il *box confidence score* della j -esima *bounding box* della i -esima cella.

λ_{noobj} diminuisce il peso della *confidence loss* rispetto alle altre, nel caso descritto, per evitare che la rete si concentri sullo sfondo maggiormente rispetto agli oggetti in primo piano. Il suo valore di default è 0.5.

4. Total loss

La *loss* complessiva è data dalla somma di tutte le formule precedenti.

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \quad (2.12)$$

2.4 YOLOv2

Si elencano di seguito i miglioramenti principali apportati alla rete YOLO nella sua versione successiva YOLOv2. Per ulteriori dettagli è possibile consultare [29].

2.4.1 Accuratezza

- **Batch normalization:** viene aggiunto il *batch normalization* nei livelli convoluzionali.
- **Anchor boxes:** nella versione precedente, all'inizio del training, le *bounding box* venivano predette in maniera del tutto arbitraria, causando oscillazioni significative del gradiente e un addestramento instabile. Nella realtà, tuttavia, le forme degli oggetti non sono del tutto arbitrarie. Tutti gli oggetti di una data classe tenderanno infatti ad assumere una forma simile, trovarsi in posizioni simili, con delle dimensioni simili. Naturalmente, le *bounding boxes* che le catturano avranno le stesse caratteristiche, e dunque si possono definire dei modelli in base ai quali costruire la *bounding box*. Questi modelli vengono chiamati *anchor boxes*. Ogni *bounding box* è determinata da un *anchor box* sulla quale viene applicato un *offset* per la traslazione e ridimensionamento.

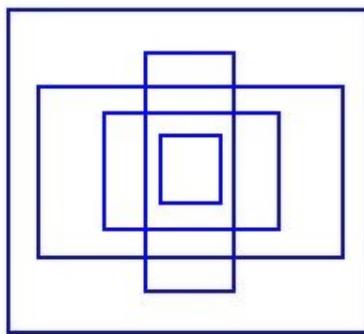


Figura 2.9: Esempio di 5 *anchor boxes* [25]

- **Modifica strutturale:** le *anchor boxes* vengono applicate in sostituzione ai livelli finali FC. L'input delle immagini viene impostato a 416x416 e, rimuovendo un livello di *pooling*, le feature maps assumono una dimensione dispari 13x13. In questo modo, esisterà una cella centrale che facilita il rilevamento di oggetti grandi. Questi, infatti,

trovandosi tendenzialmente in posizione centrale, avranno il proprio centro che ricade in una singola cella invece che quattro.

- **K-means clustering:** viene usato l'algoritmo *k-means clustering* per stimare le dimensioni delle *anchor boxes*, calcolate come i centroidi dei *k clusters* di *bounding boxes* più frequenti nel *training set*. Come misura delle distanze viene usato IoU.

2.4.2 Velocità

Il modello di rete implementata è *Darknet-19*, che con soli 19 livelli convoluzionali permette di ridurre notevolmente il numero di parametri, migliorando le prestazioni notevolmente, pur sacrificando poco in accuratezza.

	Top 1	Top 5	FLOPs	GPU Speed
VGG-16	70.5	90.0	30.95 Bn	100 FPS
Extraction (YOLOv1)	72.5	90.8	8.52 Bn	180 FPS
Resnet50	75.3	92.2	7.66 Bn	90 FPS
Darknet19	74.0	91.8	5.58 Bn	200 FPS

Tabella 2.1: Prestazioni di *Darknet-19* comparate [28]

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Tabella 2.2: struttura di *Darknet-19* [29]

2.4.3 Solidità

Vengono unificati i due dataset di training *ImageNet* [20] e *COCO* [30] secondo uno schema gerarchico per rafforzare l’addestramento della rete, andando a fondere classi equivalenti appartenenti ai due dataset.

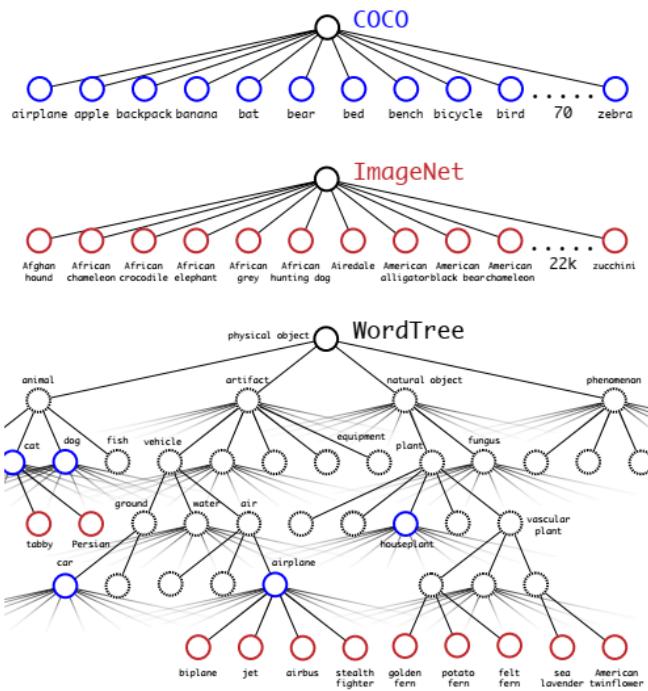


Figura 2.10: classificazione gerarchica di *ImageNet* e *Coco* [29]

2.5 YOLOv3

Di seguito vengono elencati i miglioramenti più significativi apportati alla rete YOLOv2 nella sua versione successiva YOLOv3. Per ulteriori approfondimenti, è possibile consultare [31].

- ***Darknet-53***: la precedente *Darknet-19* viene espansa con ulteriori 34 livelli convoluzionali, aumentando l’inferenza della rete ma migliorando di due punti percentuali l’accuratezza.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/S	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78

Tabella 2.3: Prestazioni *Darknet-53* comparate [32]

Per cercare comunque di limitare quanto più possibile i tempi di inferenza, vengono implementati meccanismi di *skip connection*, similmente a quanto avviene con ResNet.

	Type	Filters	Size	Output
1x	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3 / 2$	128×128
	Convolutional	32	1×1	
	Convolutional	64	3×3	
2x	Residual			128×128
	Convolutional	128	$3 \times 3 / 2$	64×64
	Convolutional	64	1×1	
	Convolutional	128	3×3	
8x	Residual			64×64
	Convolutional	256	$3 \times 3 / 2$	32×32
	Convolutional	128	1×1	
	Convolutional	256	3×3	
8x	Residual			32×32
	Convolutional	512	$3 \times 3 / 2$	16×16
	Convolutional	256	1×1	
	Convolutional	512	3×3	
4x	Residual			16×16
	Convolutional	1024	$3 \times 3 / 2$	8×8
	Convolutional	512	1×1	
	Convolutional	1024	3×3	
	Residual			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

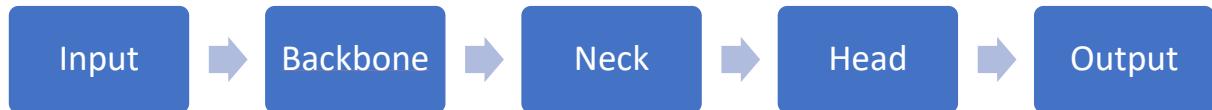
Tabella 2.4: Struttura di Darknet-53 [31]

- **Classificatore e loss function:** al posto di softmax vengono usati classificatori basati sulla regressione logistica e come loss function viene utilizzata *binary cross-entropy*.

2.6 YOLOv4

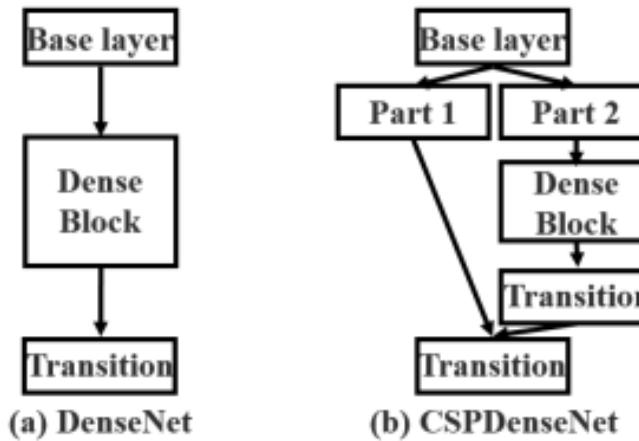
Come per i paragrafi precedenti, si elencano le caratteristiche più significative costituenti la rete YOLOv4. Si precisa che questa versione non appartiene agli sviluppatori originali. Si possono trovare tutti i dettagli al [33].

In generale, la rete di un generico *one-stage detector*, è composta da tre parti, che separano l'input dall'output, ossia *backbone*, *neck* e *head*, come nello schema di seguito:

Figura 2.11: Schema di un *detector*

Procedendo in ordine:

1. **Backbone:** con questo termine si definisce il *feature extractor*, dunque la parte della rete deputata all'estrazione delle *features* dai *patterns*. La rete utilizzata è *CSPDarknet-53*, che si differenzia da *Darknet-53* in quanto implementa il meccanismo *Cross-Stage-Partial connections*, che permette di ridurre l'inferenza della rete suddividendo il livello corrente in due parti, di cui una verrà processata da successivi livelli convoluzionali, mentre l'altra rimane invariata, procedendo con un livello finale di aggregazione. Si veda per maggiori dettagli [34]. In figura un esempio di quanto descritto, per quanto riguarda la rete *CSPDenseNet* cui *CSPDarknet-53* prende ispirazione:

Figura 2.12: Esempio di *Cross-Stage-Partial connections* [34]

2. **Neck:** in questa sezione sono presenti livelli aggiuntivi finalizzati al miglioramento dell'accuratezza della rete e riduzione dei tempi di inferenza. In particolare, vengono principalmente utilizzate due tecniche: per l'accuratezza, *Spatial pyramid pooling* (*SSP*), che aumenta il campo ricettivo della rete per migliorarne l'accuratezza di fatto eseguendo operazioni di *max-pooling* (§1.3.2); per l'inferenza, una variante di *Path Aggregation Network* (*PANet*). Per approfondirne le caratteristiche specifiche, le due tecniche sono descritte esaustivamente nei rispettivi articoli [35] e [36].

- **Head:** in questo blocco sono distribuiti i livelli della rete deputati alla realizzazione dell'output effettivo del rilevamento, in questo caso la localizzazione delle *bounding boxes*, e alla classificazione delle stesse. Non vengono qui apportate modifiche rispetto alla versione precedente.

In definitiva, con le modifiche apportate in questa versione, si può osservare un incremento notevole delle prestazioni, riportato nella tabella seguente.

Method	Backbone	Size	FPS	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	38 (M)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	31 (M)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	23 (M)	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%
YOLOv3: An incremental improvement [63]									
YOLOv3	Darknet-53	320	45 (M)	28.2%	51.5%	29.7%	11.9%	30.6%	43.4%
YOLOv3	Darknet-53	416	35 (M)	31.0%	55.3%	32.3%	15.2%	33.2%	42.8%
YOLOv3	Darknet-53	608	20 (M)	33.0%	57.9%	34.4%	18.3%	35.4%	41.9%
YOLOv3-SPP	Darknet-53	608	20 (M)	36.2%	60.6%	38.2%	20.6%	37.4%	46.1%

Tabella 2.5: Statistiche comparate di YOLOv4 e v3 relativamente a COCO dataset [33]

Capitolo 3

Implementazione

In questo capitolo si entrerà nel dettaglio per quanto riguarda l'implementazione della rete YOLO, in particolare nella sua versione YOLOv4, per risolvere il problema di rilevare in maniera automatica specie ittiche nel loro ambiente naturale marino.

3.1 Rete

Come anticipato, il modello utilizzato è YOLOv4 (§2.6), pre-addestrato sul dataset COCO [30]. In quanto tale, la rete è già in grado di rilevare oggetti appartenenti a 80 classi diverse, con le prestazioni viste in Tabella 2.5. Tuttavia, fra queste classi non compare quella dei pesci, con la conseguenza che la rete non è addestrata a riconoscerne le *features*; dunque, risulta necessario un addestramento intensivo su più datasets relativi a tale categoria.

3.2 Datasets

In questo paragrafo vengono elencati i datasets utilizzati per l'addestramento della rete e successivamente per il testing. Per tutti i *patterns*, si considera un'unica classe ‘Fish’.

3.2.1 Training

- **Labeled Fishes in the Wild** [37]: questo dataset è fornito da NOAA Fisheries (National Marine Fisheries Service). Include immagini di pesci che sono state raccolte utilizzando sistemi di telecamere dispiegati su un veicolo telecomandato (ROV) per le indagini sulla pesca. Include annotazioni relative alle *bounding boxes* che inquadrano i pesci. Il dataset è composto da due sezioni distinte, una per l'addestramento e una per il testing. La sezione per l'addestramento è composta da 575 immagini annotate.

Fra tutti i dataset, questo contiene le immagini a risoluzione più elevata, con campioni che raggiungono una risoluzione fino a 2048x1536, e contiene le immagini di qualità migliore.



Figura 3.1: Esempio di immagini di *Labeled Fishes in the Wild*

- **Aquarium Dataset [38]:** questo dataset è composto da 638 immagini raccolte da Roboflow da due acquari negli Stati Uniti: l'Henry Doorly Zoo di Omaha (16 ottobre 2020) e il National Aquarium di Baltimora (14 novembre 2020). Le immagini sono state etichettate per il rilevamento di oggetti dal gruppo di Roboflow. Le immagini contengono specie animali classificate come pesci, meduse, pinguini, squali, pulcinelle di mare, razze e stelle marine; per lo scopo del lavoro, vengono selezionate solo le annotazioni relative alle classi pesci, squali e razze, complessivamente raggruppate nella classe pesce. Il dataset è suddiviso in una sezione per l'addestramento e una per il testing. La sezione per l'addestramento, considerata la scrematura delle classi, ammonta a 374 immagini annotate.



Figura 3.2: Esempio di immagini di *Aquarium Dataset*

- **Fish Dataset 416x416 [39]:** questo dataset è stato assemblato da Jacob Solawetz raccogliendo immagini di pesci inseriti nell'habitat oceanico. Le immagini sono tutte

in risoluzione 416x416, adattate cioè per la versione *fast* di YOLOv4. Verranno comunque ridimensionate ad una risoluzione superiore come pre-processing ***. È composto da una sezione per l’addestramento e una per il testing. La sezione per l’addestramento è costituita da 1214 immagini annotate.



Figura 3.3: Esempio di immagini di *Fish Dataset 416x416*

- ***NorFisk Dataset* [40]:** questo dataset contiene immagini annotate per il riconoscimento delle specie ittiche di salmonidi d’allevamento e di merluzzo carbonaro. Queste immagini sono il risultato dell’elaborazione di riprese video all’interno e all’esterno di gabbie da diversi allevamenti ittici in Norvegia. Il dataset si può scomporre in due sezioni: una relativa alle immagini dei salmonidi, e una dei merluzzi. Non essendoci già una suddivisione in *training set* e *test set*, questi vengono ottenuti andando a prendere un certo numero di campioni per l’uno e un certo numero per l’altro. Essendo il dataset così come presentato molto più vasto rispetto agli altri finora presentati (più di 9000 immagini di salmonidi e più di 3000 di merluzzi), il numero di campioni da selezionare dovrà necessariamente essere contenuto, per evitare uno sbilanciamento nel numero di *patterns* di questo dataset rispetto agli altri. È stato dunque scelto un numero di campioni comparabile a quelli di *Labeled Fishes in the Wild*, che come già scritto presenta come *patterns* le immagini migliori dal punto di vista della qualità, in modo da mantenere un’incidenza elevata di tale dataset sull’addestramento della rete. In particolare, per l’addestramento, sono state estratte 402 immagini annotate relative ai merluzzi ed altrettanti per i salmonidi, corrispondenti ciascuno al 70% delle immagini annotate di *Labeled Fishes in the Wild*.



Figura 3.4: Esempio di immagini di *NorFisk Dataset*: (a) salmonidi, (b) merluzzi

- ***LifeCLEF 2015 Fish task* [41]**: questo dataset è costituito da 93 video di durata variabile e relative annotazioni. È il più grande come numero di immagini estraibili con più di 20000 immagini. Come per *NorFisk*, risulta necessario estrarre un numero limitato di immagini per comporre gli insiemi per *training* e *testing*, e ciò viene fatto con lo stesso criterio di suddivisione. Dunque, per l'addestramento, vengono usate 402 immagini con relative annotazioni.



Figura 3.5: Esempio di immagini di *LifeCLEF 2015 Fish task*

3.2.2 Testing

- ***Labeled Fishes in the Wild***: la sezione per il *testing* è composta da 207 frame annotati, appartenenti ad un video registrato dal ROV.
- ***Aquarium Dataset***: la sezione per il *testing* è complessivamente composta da 34 immagini con relative annotazioni.
- ***Fish Dataset 416x416***: la sezione per il *testing* è composta da 136 immagini annotate.

- **NorFisk Dataset:** la sezione per il *testing* è composta da 173 immagini annotate per i salmonidi ed altrettante per i merluzzi, corrispondenti ciascuno al 30% delle immagini di *Labeled Fishes in the Wild*.
- **LifeCLEF 2015 Fish task:** la sezione per il *testing* è composta da 173 immagini annotate.
- **Dataset dell'Università Politecnica delle Marche (UNIVPM):** questo dataset è composto da video di 3 minuti registrati ogni ora per 12 mesi (da Febbraio 2020 a Febbraio 2021) presso una stazione localizzata sul fondo a 17 metri di profondità al largo delle coste abruzzese (Adriatico Centrale), per un totale di 9034 elementi. Per l'assenza di annotazioni, i pesci presenti sono stati localizzati dal sottoscritto mediante *Image Labeler* di Mathworks su un numero limitato di frame che potessero catturare diverse specie immerse in un ambiente marino con diverse condizioni di torbidità e colore dell'acqua. In particolare, questi due fattori dipendono anche dalle condizioni non sempre ottimali della videocamera, che durante il periodo descritto è stata sottoposta a diversi interventi di manutenzione. Il dataset è utilizzato solo per il *testing*, e l'insieme di immagini annotate ammonta in definitiva a 207 immagini.



Figura 3.6: Esempio di immagini del dataset di UNIVPM

- **Dataset Liguria:** questo dataset è composto da 18 video registrati con una action camera dal sottoscritto nel mare della costa ligure. Da questi video sono stati estratti ed etichettati 160 frames contenenti specie di pesce in parte dissimili a quelle che compongono il dataset di UNIVPM. A differenza di quest'ultimo, inoltre, la camera non è statica ma dinamica, e le condizioni dell'acqua sono stabili e buone, gli sfondi sono molteplici. Nel complesso, questo insieme di caratteristiche permette di testare più approfonditamente le prestazioni della rete nel rilevamento dei pesci.

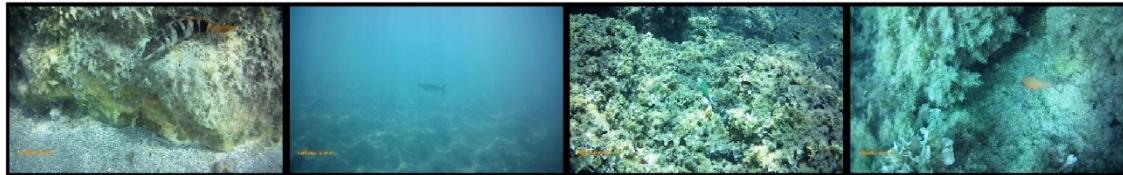


Figura 3.7: Esempio di immagini di *Dataset Liguria*

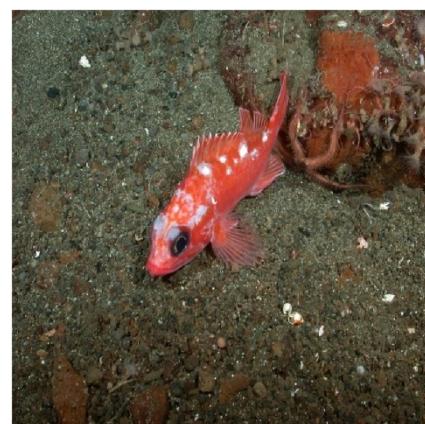
3.3 Preprocessing

La rete YOLOv4 prende in input un’immagine a qualunque risoluzione e ne effettua il ridimensionamento per renderla compatibile con il suo primo livello di input. Nel caso della rete implementata, il livello di input accetta un volume 608x608x3, dunque le immagini che vengono passate alla rete devono essere immagini con 3 canali (si suppone RGB) che vengono portate a quella risoluzione.

Nel flusso di esecuzione di una rete neurale, il *preprocessing* dell’input rappresenta tutte le operazioni che vengono svolte su di esso prima del suo effettivo processamento da parte della rete nel passaggio di inferenza. In questo lavoro, il *preprocessing* è limitato al ridimensionamento dell’immagine. In figura, si osserva il ridimensionamento dell’immagine da una risoluzione di 2048x1536 pixel a 608x608 pixel. Da notare in particolare il cambiamento all’*aspect ratio* dell’immagine, i.e. il suo rapporto fra le dimensioni orizzontale e verticale. Il ridimensionamento effettuato prima della *data augmentation* consente di migliorare le prestazioni della rete, in quanto lavorando con una risoluzione inferiore è possibile velocizzare i tempi di calcolo.



(a)



(b)

Figura 3.8: Esempio di immagine (a) sottoposta a *preprocessing* (b)

3.4 Data Augmentation

Uno dei problemi più importanti che può compromettere l'efficacia dell'addestramento di una rete neurale è quello dell'insufficienza di dati. Gli algoritmi di Deep Learning, infatti, richiedono una quantità di dati molto elevata, che spesso si traduce in lavoro ulteriore per il loro reperimento, ed occupazione massiccia dello spazio in memoria a disposizione.

Per ovviare a questi problemi, esistono tecniche di processamento dei dati che consentono di incrementare in maniera artificiale il numero di *patterns* a disposizione, modificando quelli già esistenti, senza tuttavia aumentare lo spazio occupato in memoria. Questo insieme di tecniche prende il nome di ***data augmentation*** e rappresenta uno dei punti cardine dell'addestramento di un *detector*. Durante l'addestramento, infatti, tutti i *patterns* del *training set* non vengono mai processati allo stesso modo, in quanto vengono passati alla rete a seguito di precedente modifica mediante una successione di metodi che tendenzialmente operano con un certo grado di casualità.

Si descrivono di seguito i metodi di *data augmentation* implementati.

3.4.1 jitterColorHSV

Questo metodo consente di alterare il colore di un'immagine RGB, andando a modificare i valori della stessa immagine convertita nello spazio di colore **HSV** (*Hue*, *Saturation*, *Value*).

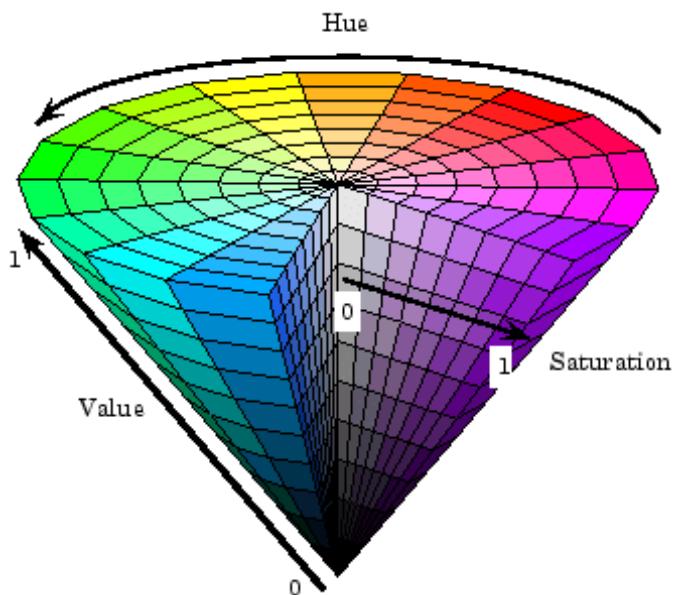


Figura 3.9: Rappresentazione dello schema HSV [42]

Con riferimento alla Figura 3.9, ogni punto interno allo spazio HSV rappresenta il colore visibile di un pixel.

Con “***hue***” si intende il colore puro del pixel, in particolare la sua posizione angolare rispetto al rosso, che, considerando ad esempio come unità di misura i gradi, è posto convenzionalmente sullo zero. L’intervallo dei colori puri è dunque [0,360], e si può aggiungere che, essendo la gamma di colori circolare, 0 e 360 coincidono con il colore rosso. È anche possibile esprimere *hue* come un numero scalare compreso nell’intervallo [0,1] normalizzando l’intervallo angolare [0,360], i.e. dividendo l’intervallo per 360.

Con “***saturation***” si intende la quantità di grigio in un dato colore, o, equivalentemente, la distanza del pixel dall’asse del cono (Figura 3.9), che rappresenta la scala di grigi. Più il pixel è lontano dall’asse del cono, e dunque dal grigio, più il colore visibile apparirà vivido; più è vicino, maggiore sarà la dominanza del grigio. In generale si rappresenta come uno scalare nell’intervallo [0,1] dove 0 indica un punto dell’asse, 1 la massima distanza orizzontale da esso.

Con “***value***”, o equivalentemente “***brightness***”, si indica la luminosità del colore, ossia la quantità di luce emessa dal pixel. In particolare, è rappresentata da uno scalare nell’intervallo [0,1] dove 0 indica il nero, dunque non emette luce, 1 invece è il bianco, ossia il punto di massima luminosità. In congiunzione agli altri due parametri, 1 consente di “rivelare” il colore visibile completamente, mentre 0 lo spegne del tutto.

Il metodo “*jitterColorHSV*”, dopo aver convertito l’immagine RGB in HSV, consente di modificare i seguenti parametri:

- ***hue_{offset}***: rappresenta lo scostamento del colore puro di ciascun pixel dal suo valore corrente. È esprimibile come valore scalare nell’intervallo [0,1], ed in tal caso viene aggiunto un valore compreso nell’intervallo [-*hue_{offset}*, *hue_{offset}*], oppure come un vettore di due numeri scalari compresi nell’intervallo [-1,1] (e.g. [0,0.5]), ed in tal caso viene aggiunto un valore scalare compreso fra gli estremi del vettore.
- ***saturation_{offset}***: rappresenta lo scostamento della saturazione di ciascun pixel dal suo valore corrente. È esprimibile come valore scalare nell’intervallo [0,1], ed in tal caso viene aggiunto un valore compreso nell’intervallo [- *saturation_{offset}* , *saturation_{offset}*], oppure come un vettore di due numeri scalari compresi

nell'intervallo $[-1,1]$ (e.g. $[0,0.5]$), ed in tal caso viene aggiunto un valore scalare compreso fra gli estremi del vettore.

- **brightness_{offset}**: rappresenta lo scostamento della luminosità di ciascun pixel dal suo valore corrente. È esprimibile come valore scalare nell'intervallo $[0,1]$, ed in tal caso viene aggiunto un valore compreso nell'intervallo $[-\text{brightness}_{\text{offset}}, \text{brightness}_{\text{offset}}]$, oppure come un vettore di due numeri scalari compresi nell'intervallo $[-1,1]$ (e.g. $[0,0.5]$), ed in tal caso viene aggiunto un valore scalare compreso fra gli estremi del vettore.
- **contrast**: il contrasto rappresenta un fattore di riscalamento della luminosità. In particolare, può essere definito come numero positivo, ed in tal caso la luminosità di ciascun pixel viene moltiplicata per un fattore compreso nella distribuzione $[1-\text{contrast}, 1+\text{contrast}]$, oppure come un vettore di due numeri scalari positivi, ed in tal caso la luminosità viene moltiplicata per un fattore compreso fra i due valori.

Infine, il metodo riconverte lo spazio HSV modificato nell'originario RGB.

(© 1994-2022 The MathWorks, Inc.)

Per il presente lavoro di tesi, il metodo viene utilizzato con i seguenti parametri:

- $\text{hue}_{\text{offset}} \leftarrow 0.1$: questo valore consente di cambiare il colore puro di ciascun pixel scostando tale valore sulla gamma dei colori, senza tuttavia determinare un cambiamento tale nei colori dell'immagine da snaturarne la realisticità. Infatti, è ragionevole assumere che ciascun elemento in un'immagine non possa assumere colori completamente diversi in un'altra immagine, se tali immagini sono fotografie.
-

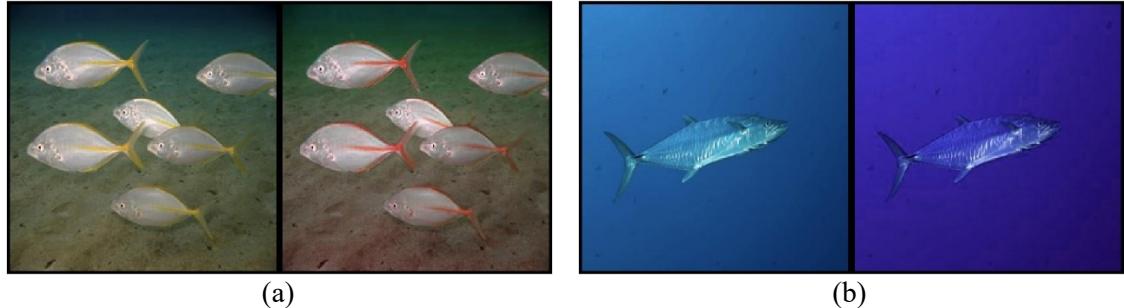


Figura 3.10: Esempio di $\text{hue}_{\text{offset}} = -0.1$ (a) e 0.1 (b)

- $saturation_{offset} \leftarrow 0.2$: questo valore consente di modificare la purezza del colore sempre cercando di preservare la realisticità dell'immagine, spegnendo o accendendo solo lievemente i colori. Uno spegnimento eccessivo causerebbe la degradazione della qualità dell'immagine alla scala di grigi, un'accensione esagerata non sarebbe naturale.
 -



Figura 3.11: Esempio di $saturation_{offset} = -0.2$ (a) e 0.2 (b)

- $brightness_{offset} \leftarrow 0$: questo valore mantiene la luminosità originale. La scelta nasce dal fatto che venendo applicata una variazione al contrasto, viene già modificato il campo *value* dell’immagine HSV. Una sovrapposizione degli effetti si è osservato portare ad un risultato insoddisfacente, dunque si è scelto di dare la priorità al parametro *contrast*.
 - $contrast \leftarrow 0.5$: questo consente con ugual probabilità di ridurre il contrasto, se il valore cade nell’intervallo [0.5,1), oppure di aumentarlo, nel caso il valore cada nell’intervallo (1,1.5].

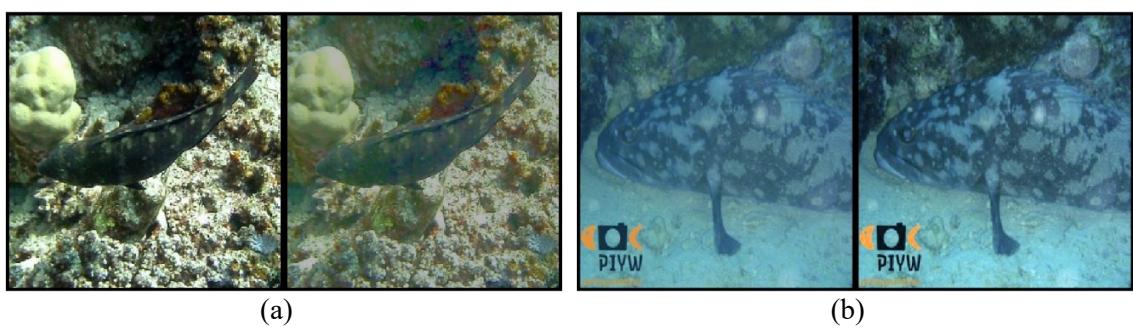


Figura 3.12: Esempio di riduzione del contrasto (a), e aumento (b)

Si mostrano di seguito un esempio relativo all'applicazione ripetuta del metodo su una stessa immagine, considerando la congiunzione dei parametri precedentemente espressi:

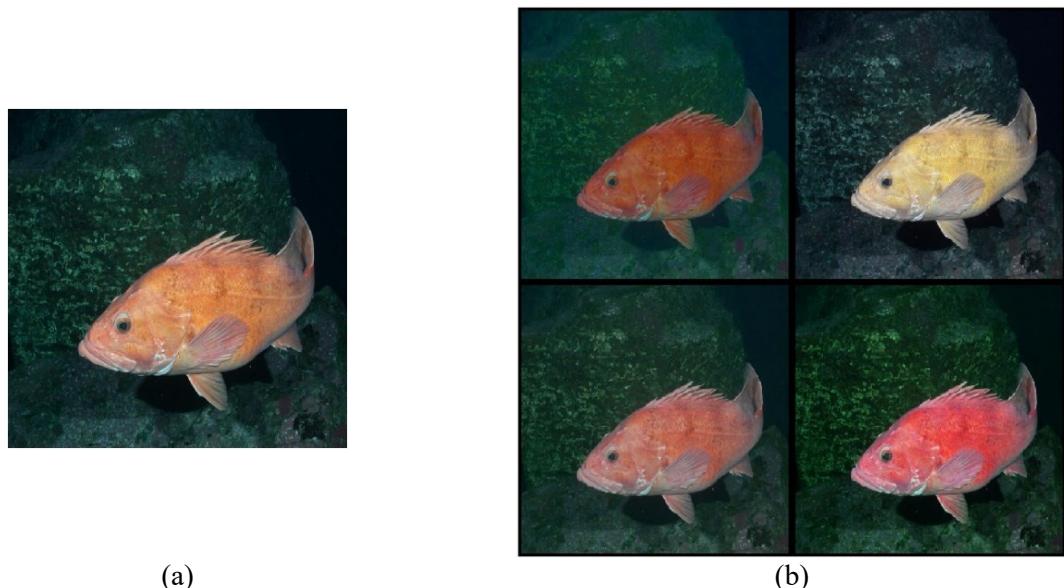


Figura 3.13: Esempio di quattro applicazioni del metodo (b) alla stessa immagine originale (a)

3.4.2 Bilanciamento del colore

Gli algoritmi di bilanciamento del colore servono a ripristinare il colore di un'immagine qualora sottoposta a condizioni di illuminazione sbilanciate che rendono dominante un colore o una gamma di colori sugli altri. L'idea è quella di restaurare i colori dell'immagine in modo che la scena appaia con i colori che avrebbe se fosse investita da una luce bianca, ossia con tutte le componenti dello spettro del visibile.

Il bilanciamento del colore è quanto mai rilevante nella manipolazione di immagini subacquee, a causa dell'assorbimento della luce da parte dell'acqua che provoca una perdita progressiva delle tonalità di rosso, arancione, giallo, verde e blu con il crescere della profondità. Dunque, se le fotografie non sono state scattate con una luce artificiale come il flash, è importante cercare di recuperare questi colori persi, mediante tecniche di bilanciamento.

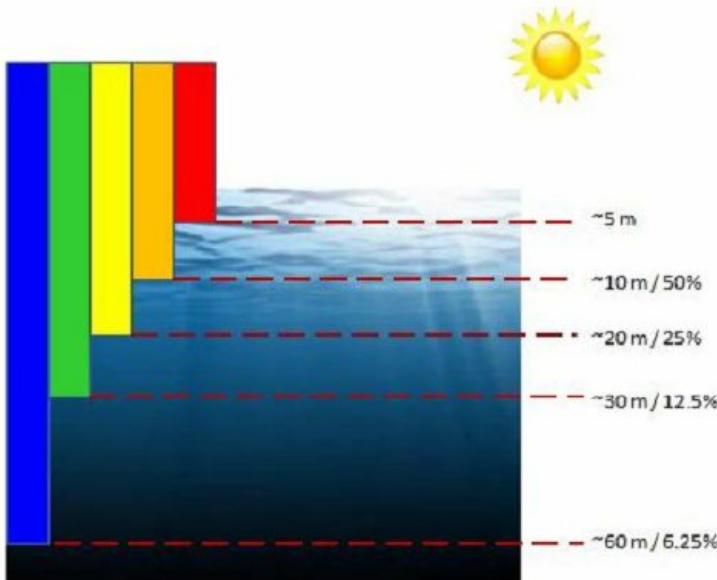


Figura 3.14: Assorbimento della luce da parte dell'acqua [43]

Un algoritmo di bilanciamento del colore si compone generalmente di due fasi: stima dell'illuminazione e adattamento cromatico.

3.4.2.1 Gray World

Per la stima dell'illuminazione dell'immagine RGB, viene adoperato l'algoritmo **Gray World**, che si basa sulla suposizione che il color medio in un'immagine bilanciata sia un grigio neutrale, che corrisponde ad un egual componente di rosso, verde e blu.

L'illuminazione dell'immagine viene stimata mediante il calcolo del suo illuminante, ossia un vettore di tre elementi corrispondenti all'illuminazione rispettivamente del rosso, verde e blu. In *Gray World*, a seguito dell'assunzione fatta, ciascuna delle tre illuminazioni viene calcolata come la media dei valori nell'immagine corrispondenti ad ogni canale, normalizzata rispetto al canale con massima intensità.

Per il calcolo dell'illuminante, è possibile escludere una percentuale di pixel con valori troppo bassi o troppo alti.

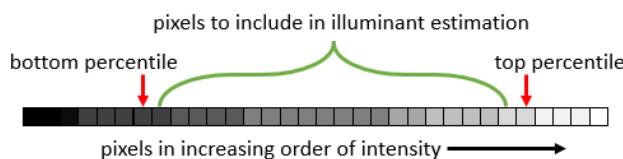


Figura 3.15: Esempio di esclusione percentuale di pixels [44]

Esistono altri metodi per la stima dell'illuminazione dell'immagine, tuttavia *Gray World* in particolare garantisce elevate prestazioni computazionali, limitandosi al calcolo di una media normalizzata su tre canali.

3.4.2.2 Bradford CAT

Stimata l'illuminazione dell'immagine, l'ultimo passaggio consiste nell'adattamento del colore, una tecnica che consente di convertire una combinazione sorgente di colori in una combinazione di destinazione attraverso una trasformazione lineare, o CAT (*Chromatic Adaptation Transform*).

In generale, data una sorgente di colori $[X_S, Y_S, Z_S]^t$, l'obiettivo è determinare la destinazione di colori $[X_D, Y_D, Z_D]^t$ che si ottiene moltiplicando la matrice $[M]$ (3.4) per il vettore sorgente, secondo la formula:

$$[X_D, Y_D, Z_D]^t = [M] [X_S, Y_S, Z_S]^t \quad (3.1)$$

Si definisce inoltre $[\rho_S, \gamma_S, \beta_S]^t$ come:

$$[\rho_S, \gamma_S, \beta_S]^t = [M_A] [X_{WS}, Y_{WS}, Z_{WS}]^t \quad (3.2)$$

Dove:

- $[M_A]$ rappresenta la matrice di adattamento, i cui valori dipendono dal metodo scelto, in questo caso Bradford, essendo da risultati sperimentali considerato il migliore. [45]
- $[X_{WS}, Y_{WS}, Z_{WS}]^t$ rappresenta l'illuminante dell'immagine sorgente.

$[\rho_S, \gamma_S, \beta_S]^t$ di fatto rappresenta la trasformazione nelle coordinate del *cono di risposta* dell'illuminante della sorgente.

Si definisce inoltre $[\rho_D, \gamma_D, \beta_D]^t$ come:

$$[\rho_D, \gamma_D, \beta_D]^t = [M_A] [X_{WD}, Y_{WD}, Z_{WD}]^t \quad (3.3)$$

Dove:

- $[X_{WD}, Y_{WD}, Z_{WD}]^t$ rappresenta l'illuminante dell'immagine destinazione. La sua scelta è arbitraria, in questo caso si è scelto *D65 CIE standard illuminant*, il più

comunemente utilizzato, che rappresenta l'illuminazione della luce solare di mezzogiorno.

La matrice $[M]$ si ottiene invece nel seguente modo:

$$[M] = [M_A]^{-1} \begin{bmatrix} \rho_D/\rho_S & 0 & 0 \\ 0 & \gamma_D/\gamma_S & 0 \\ 0 & 0 & \beta_D/\beta_S \end{bmatrix} [M_A] \quad (3.4)$$

Per ulteriori approfondimenti è possibile consultare [46].

3.4.2.3 Esempio

Si mostra ora un esempio dell'applicazione dell'algoritmo di bilanciamento del colore.



Figura 3.16: Input: immagine del *training set*; Output: immagine sottoposta a bilanciamento del colore; *Ground truth*: immagine [47]

In Figura 3.16, si mostra un'immagine di cernia a nido d'ape appartenente al *training set* sottoposta a bilanciamento del colore e successivamente confrontata con un'immagine *ground truth* (sotto le giuste condizioni di illuminazione) di un esemplare della stessa specie. Si può

osservare come l'algoritmo bilanci il colore con risultati molto simili a quello effettivo, in quanto nell'immagine di input, pur osservando una dominanza del canale verde, i canali rosso e blu sono comunque ben conservati. L'illuminante relativo è infatti [0.23,0.53,0.45].

Si osservi ora invece la seguente figura:

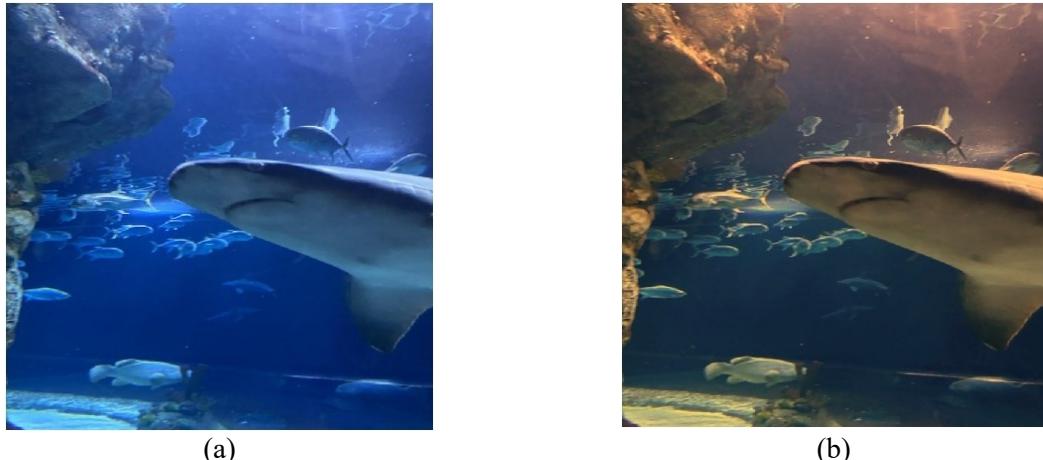


Figura 3.17: Esempio di applicazione non ottimale del bilanciamento del colore (b) all'immagine (a)

Come si può osservare dalla Figura 3.17, in questo caso il bilanciamento del colore comporta un eccesso nella compensazione del blu con l'aggiunta del suo complementare giallo. Questo effetto avviene qualora vi sia una predominanza netta di un canale sugli altri, e ciò è constatabile dall'illuminante relativo, che è [0.13,0.29,0.56].

Considerando la composizione totale del *training set*, tuttavia, le immagini risultano nel complesso bilanciate, rendendo dunque l'algoritmo utile all'addestramento.

Sebbene a primo impatto il bilanciamento del colore sembri negare gli effetti di *jitterColorHSV*, in realtà contribuisce a creare delle differenze meno sostanziali fra le diverse versioni dell'immagine, consentendo di proporre nell'addestramento immagini più fedeli alla realtà.

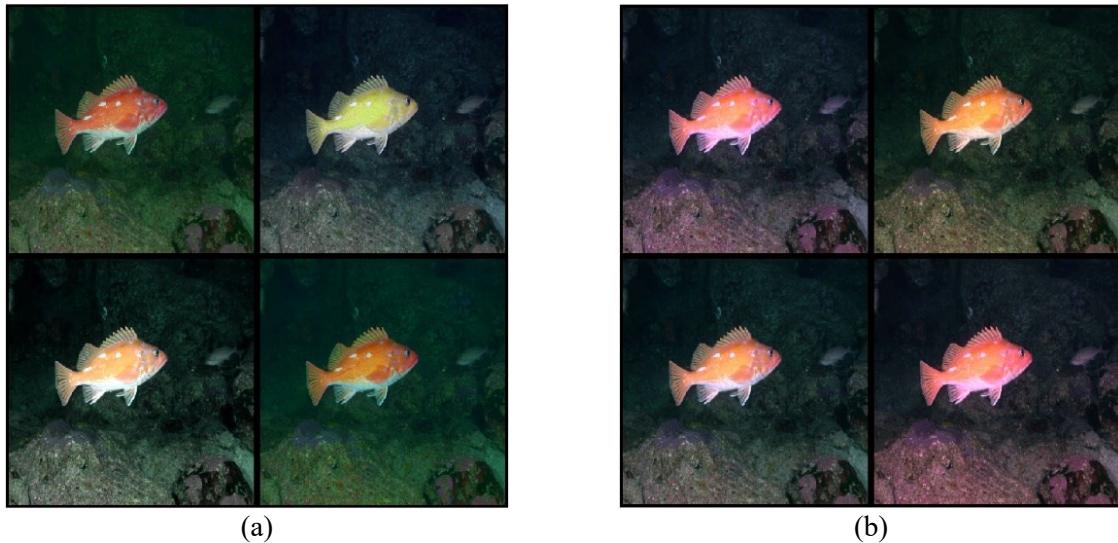


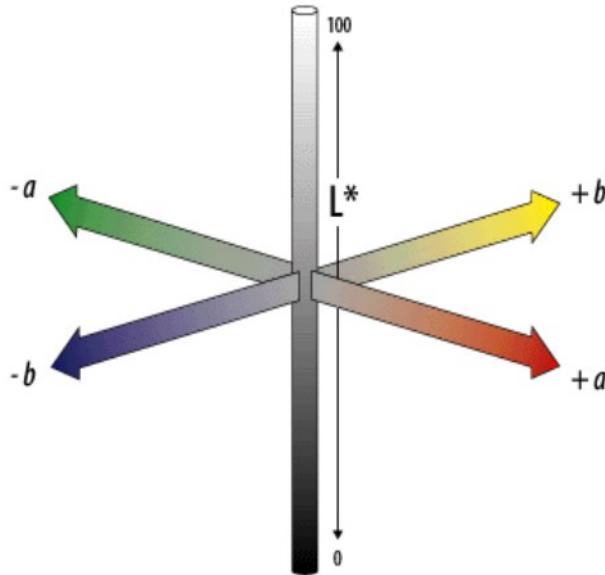
Figura 3.18: Esempio di applicazione del bilanciamento del colore (b) a immagini sottoposte a *jitterColorHSV* (a)

3.4.3 Aggiustamento del contrasto

Il passaggio successivo per il miglioramento dell’immagine è costituito dall’aggiustamento del contrasto. A questo scopo, fra le tecniche utilizzate, risalta in particolare ***Contrast-Limited Adaptive Histogram Equalization (CLAHE)***. Tuttavia, questo metodo può operare solo su un singolo canale di colore, in quanto l’input accettato è bidimensionale. Per applicarlo ad una immagine RGB, è necessario prima convertirla nello spazio di colore ***CIELAB*** ($L^*a^*b^*$).

Come suggerisce il nome, $L^*a^*b^*$ è composto da tre componenti:

- L rappresenta la luminosità, ed assume valori compresi fra 0 (nero) e 100 (bianco). È equivalente alla conversione in scala di grigi dell’immagine RGB.
- a è un parametro che indica la quantità di rosso per valori positivi, quantità di verde per valori negativi, essendo rosso e verde colori complementari.
- b è un parametro che indica la quantità di giallo per valori positivi, quantità di blu per valori negativi, essendo giallo e blu colori complementari.

Figura 3.19: Spazio di colore $L^*a^*b^*$ [49]

Naturalmente, *CLAHE* viene applicato alla componente L .

L'algoritmo si compone di tre parti principali: generazione di *tiles* (tessere), equalizzazione dell'istogramma e interpolazione bilineare.

1. L'immagine viene suddivisa in tessere come in un mosaico, permettendo così di lavorare sull'aggiustamento del contrasto a livello locale, migliorando la definizione dei bordi fra le regioni adiacenti dell'immagine. Nel presente lavoro si utilizza una griglia 8x8.
2. Si effettua l'equalizzazione dell'istogramma relativo a L su ogni tessera, considerando un limite, definito *clip limit*, oltre il quale l'istogramma viene tagliato. Così facendo, il *clip limit* permette di evitare la sovra amplificazione del rumore. L'eccesso dovuto al *clipping* viene quindi ridistribuito sotto al limite e i valori finali vengono mappati nei corrispettivi di L . In questo lavoro il *clip limit*, che può assumere valore compreso nell'intervallo $[0,1]$ viene impostato a 0.005, in modo da non accentuare eccessivamente il contrasto. L'istogramma segue la *distribuzione di Rayleigh* (a campana), che si è dimostrata essere particolarmente valida nell'equalizzazione degli istogrammi per immagini subacquee [50].

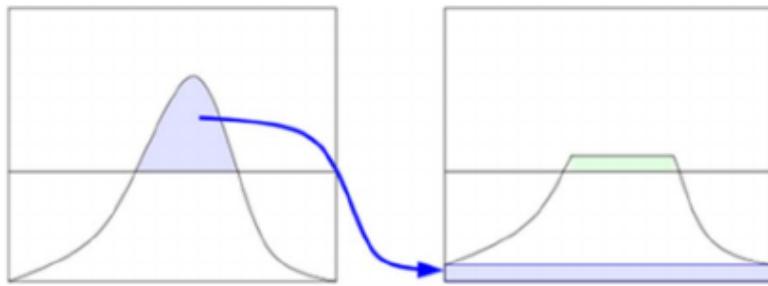


Figura 3. 20: Ridistribuzione dell'eccesso dovuto al clipping [51]

3. Le tessere risultanti vengono quindi ricucite tramite interpolazione bilineare ricostruendo l'immagine originaria.

Ottenuta la nuova componente L , si effettua la conversione dello spazio del colore a RGB ottenendo l'immagine finale con il contrasto aggiustato.

Si mostra di seguito un esempio di applicazione dell'aggiustamento del contrasto con *CLAHE*.

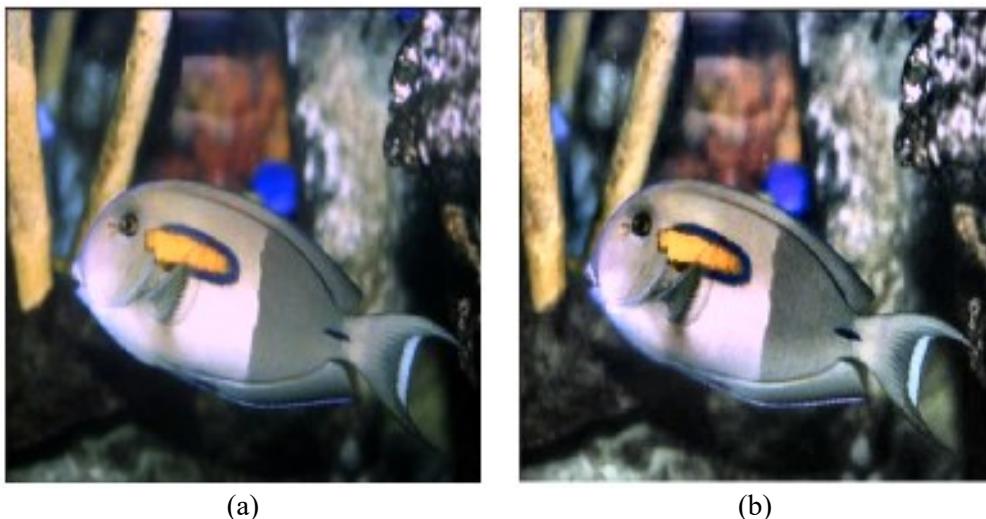


Figura 3.21: Esempio di applicazione di *CLAHE* (b) su un'immagine del *training set* con colori bilanciati (a)

3.4.4 Wavelet fusion

Con *image fusion*, “fusione di immagini”, si intende in generale una procedura attraverso la quale due o più immagini vengono unite in un'immagine unica che contiene le informazioni più significative di ciascuna delle immagini originali. [52]

Gli obiettivi principali di un metodo di *image fusion* si possono riassumere in:

- Diminuzione del volume di dati
- Conservazione delle *features* principali
- Rimozione degli artefatti
- Produzione di un’immagine più facilmente leggibile ed interpretabile

Fra le tecniche di *image fusion* una delle più moderne ed efficienti è rappresentata da *wavelet fusion*, che consiste nella fusione degli output di una decomposizione *wavelet* applicata a due o più immagini, dove con *wavelet*, o “ondicella”, si intende la rappresentazione di un segnale come un’onda oscillante.

Una **decomposizione *wavelet*** si ottiene da una sequenza di filtri passa-basso e passa-alto che vengono utilizzati per eliminare le basse e alte frequenze indesiderate presenti nel segnale, come forma di cancellazione del rumore, e per acquisire separatamente i coefficienti di dettaglio e approssimazione derivanti dall’applicazione dei filtri. Il numero di coefficienti varia in base alla dimensionalità del segnale e dal numero di livelli di decomposizione.

In particolare, considerate immagini bidimensionali come segnale di input, per ogni livello si ottengono quattro coefficienti: uno di approssimazione (LL) e tre di dettaglio, per mettere in luce le *features* verticali (LH), orizzontali (HL) e diagonali (HH). Questi coefficienti avranno caratteristiche che dipendono dalla *wavelet* utilizzata, ossia la funzione che definisce ciascun filtro, che, nel caso del presente lavoro, è la *wavelet Haar* [53].

I nomi dei coefficienti non sono casuali, e dipendono infatti dall’ordine di applicazione dei filtri, che avviene come in figura:

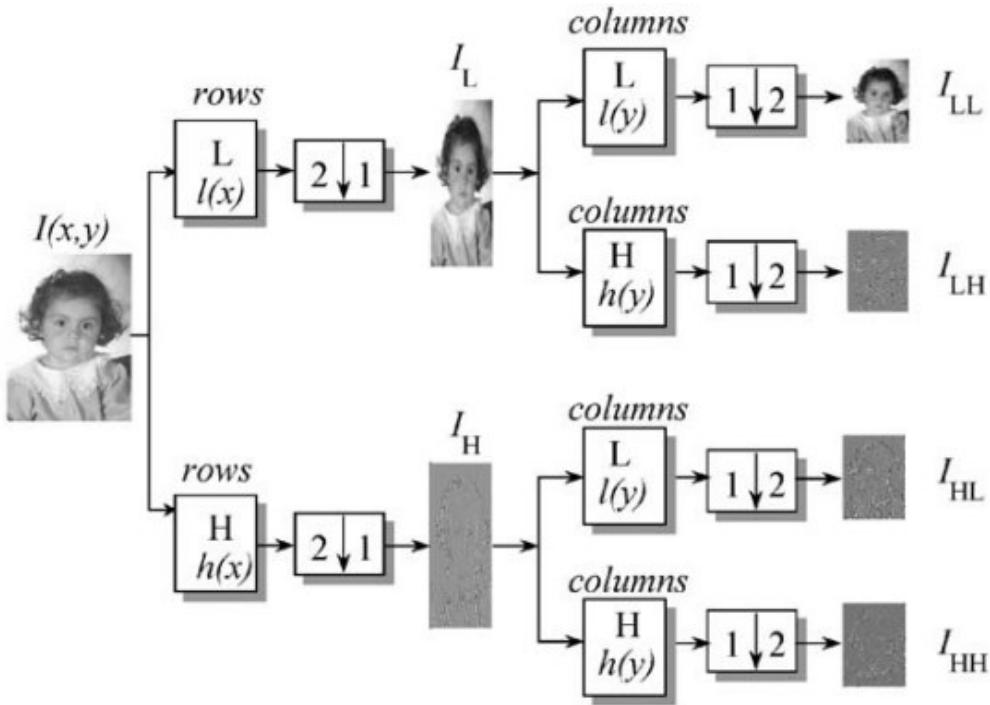


Figura 3.22: Schema di decomposizione 2-D singolo livello [54]

Con riferimento alla Figura 3.22, L indica l'applicazione di un filtro passa basso; H passa alto. Dall'immagine di input, rappresentabile come una matrice (x,y) , vengono dapprima applicate sulle righe filtri passa basso e pass alto, con successivo sottocampionamento, producendo rispettivamente approssimazioni orizzontali e dettagli orizzontali. A questo punto vengono filtrate le colonne e viene effettuato il sottocampionamento, producendo i quattro coefficienti descritti in precedenza.

In questo lavoro si utilizzano due livelli, dunque l'approssimazione LL ottenuta viene utilizzata come input al secondo livello, che procede in modo equivalente al primo. In definitiva, si otterranno sette coefficienti, in quanto la precedente approssimazione viene sostituita con i quattro nuovi coefficienti.

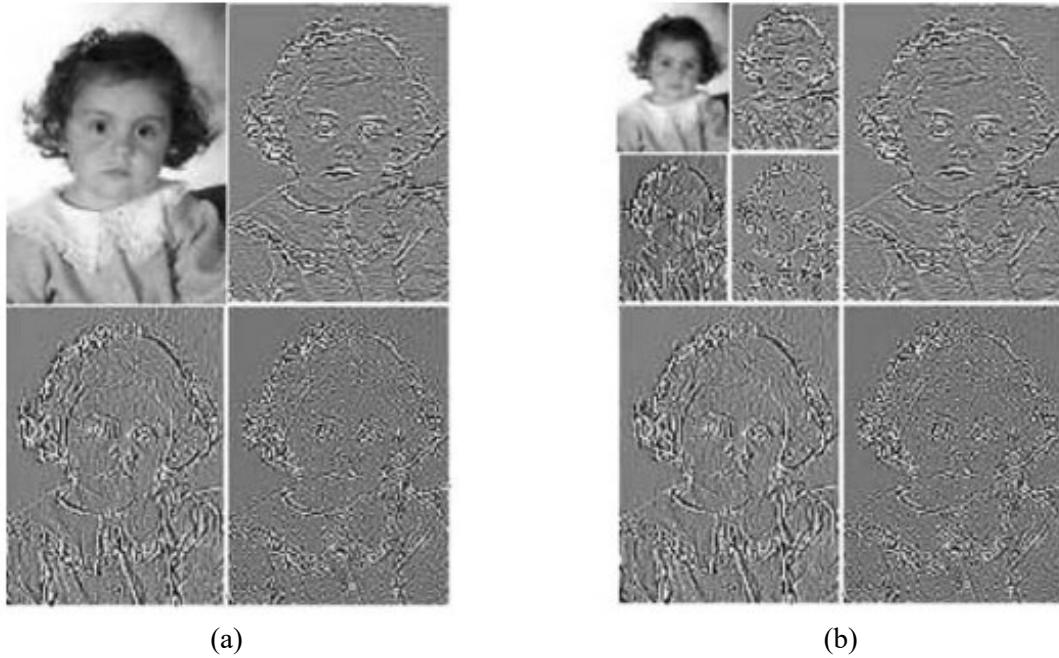
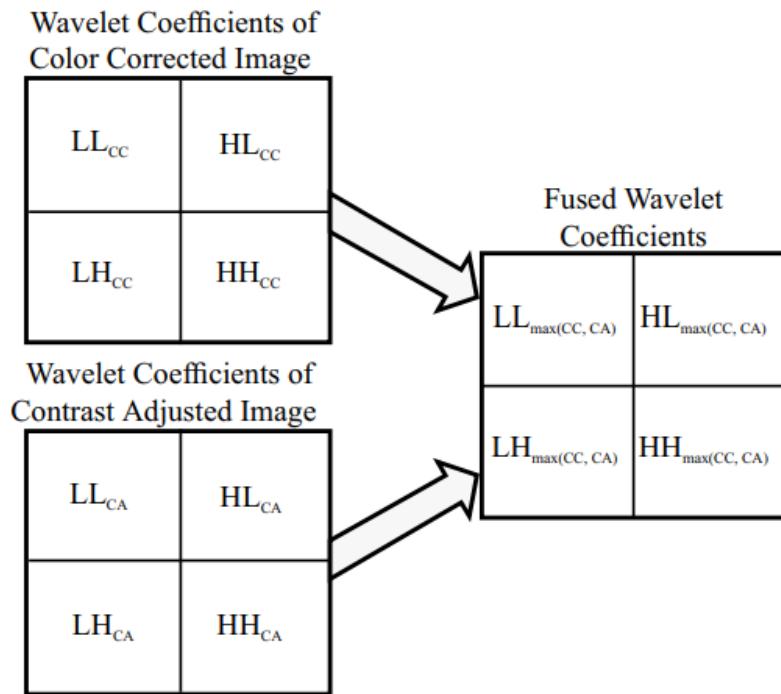


Figura 3.23: Confronto tra decomposizione 2D a singolo livello (a) e due livelli (b) [54]

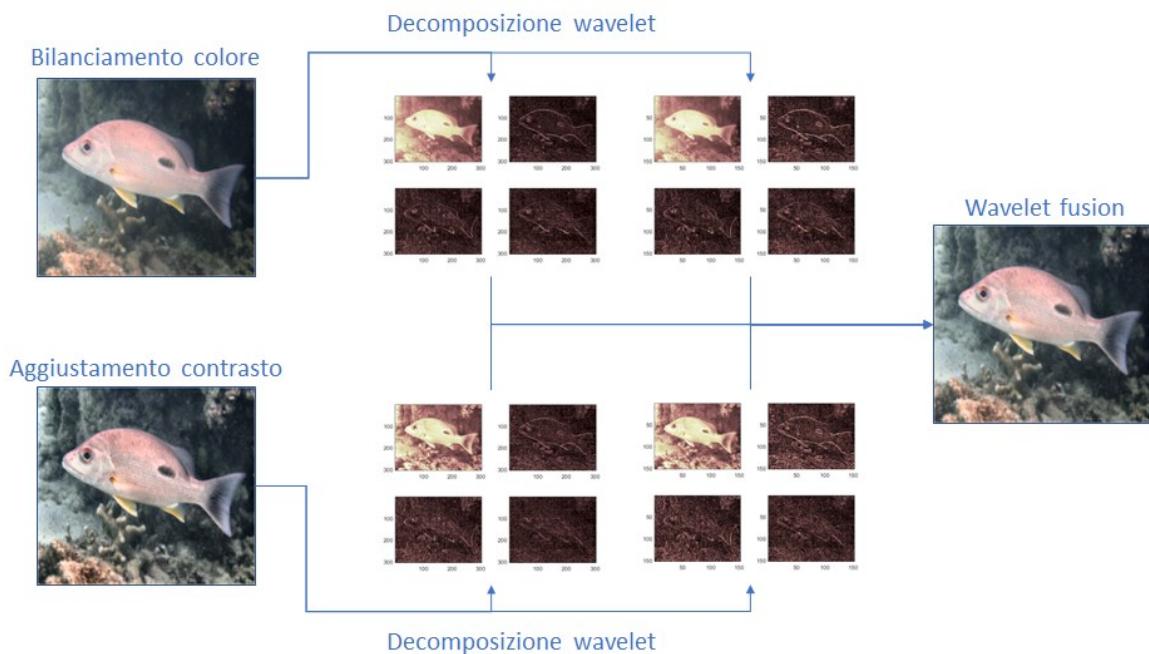
Si precisa inoltre che il formato RGB di tutte le immagini manipolate in questo lavoro viene reso compatibile alla bidimensionalità dell'input della decomposizione lavorando sulla componente L dell'immagine convertita sul piano $L^*a^*b^*$, già descritto al §3.4.3.

A questo punto è possibile implementare la fusione. L'operazione di decomposizione a due livelli viene ripetuta per due volte: una sull'immagine di output del bilanciamento del colore (§3.4.2) e l'altra sull'immagine ottenuta dall'aggiustamento del contrasto (§3.4.3). I coefficienti delle decomposizioni vengono quindi fusi seguendo la regola *mean-mean*, i.e. media dei coefficienti di approssimazione e media di quelli di dettaglio.

Infine, la decomposizione così ottenuta viene ricomposta invertendo il processo, ottenendo l'immagine finale. Il processo è visualizzabile nello schema riportato in seguito.

Figura 3.24: Schema di *wavelet fusion* implementata [51]

Si mostra di seguito un esempio del processo di *wavelet fusion* applicato ad un'immagine *training set*, sottoposta ad *augmentation* come descritta finora.

Figura 3.25: Esempio di *wavelet fusion* per un immagine del training set

3.4.5 Ribaltamento orizzontale e riscalamento

Le immagini e relative *bounding boxes* vengono ribaltate orizzontalmente con una probabilità del 50%, in modo da coprire le direzioni possibili di spostamento dei pesci. È noto, infatti, che l'unica specie di pesce a nuotare verticalmente è il cavalluccio marino, non presente nei dataset di testing, e pertanto rotazioni di 90 gradi vengono escluse dall'*augmentation* per questo motivo.

Non esistono altresì specie di pesce che nuotino supini, pertanto sono esclusi sia ribaltamenti verticali che rotazioni di 180 gradi.

Le immagini vengono infine soggette a riscalamento del 10% con una probabilità del 50%, e adattate le *bounding box* relative di conseguenza.

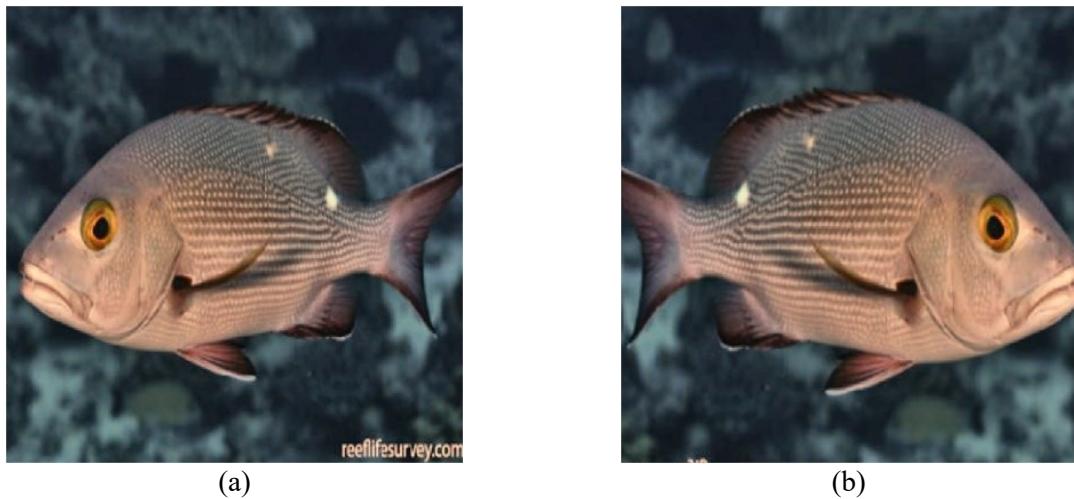


Figura 3.26: Esempio di applicazione del metodo (b) sull'immagine (a)

3.4.6 Aumento luminosità

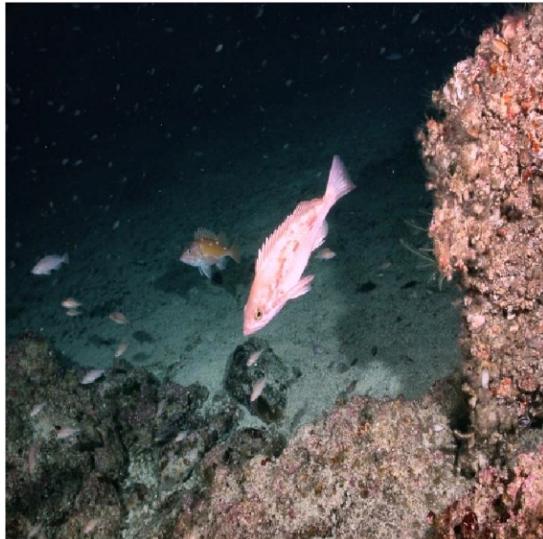
L'ultimo metodo è l'aumento della luminosità dell'immagine. Questo metodo nasce dalla necessità di cercare di migliorare la precisione di *detection* per alcuni *patterns* del dataset di testing di UNIVPM. A causa della posizione frontale del flash della videocamera, infatti, alcune immagini soffrono di un problema di bagliore diffuso sulla superficie del pesce, estremamente riflettente a causa delle squame che la ricoprono, che maschera la capacità della rete di rilevare le *features* più basilari dei pesci come la loro forma e il loro colore.



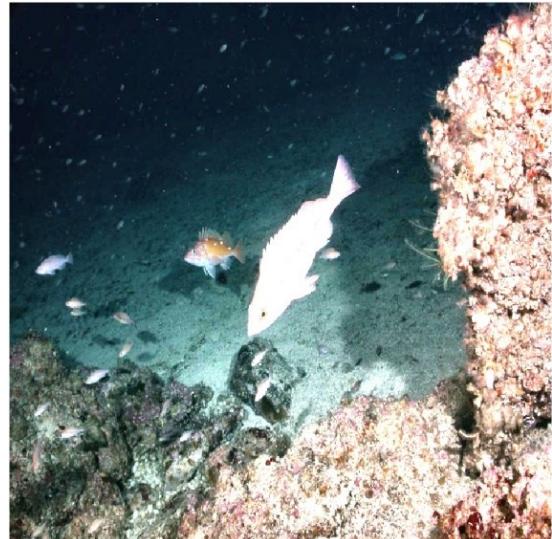
Figura 3.27: Frame rappresentativo del problema del bagliore

Aumentando la luminosità con una probabilità del 25%, per un fattore estratto nell'intervallo (1.5,1.75) con probabilità uniforme, si cerca di abituare il *detector* a rilevare le *features* anche in situazioni più complesse, pur mantenendo una predominanza di immagini con luminosità normale nell'addestramento della rete.

Si mostra di seguito un esempio dell'applicazione del metodo ad un'immagine del *training set*, modificata dall'*augmentation* precedente.



(a)



(b)

Figura 3.28: Esempio di aumento di luminosità x1.7 (b) all'immagine (a)

3.4.7 Training e testing

Avendo discusso ampiamente i metodi di *data augmentation* implementati, vengono ora distinti i due flussi di esecuzione della stessa relativamente a *training* e *testing*.

- Il ***training*** sfrutta tutti i metodi di *data augmentation* finora descritti, nella sequenza descritta. Si mostra uno schema in figura rappresentante il flusso di esecuzione.

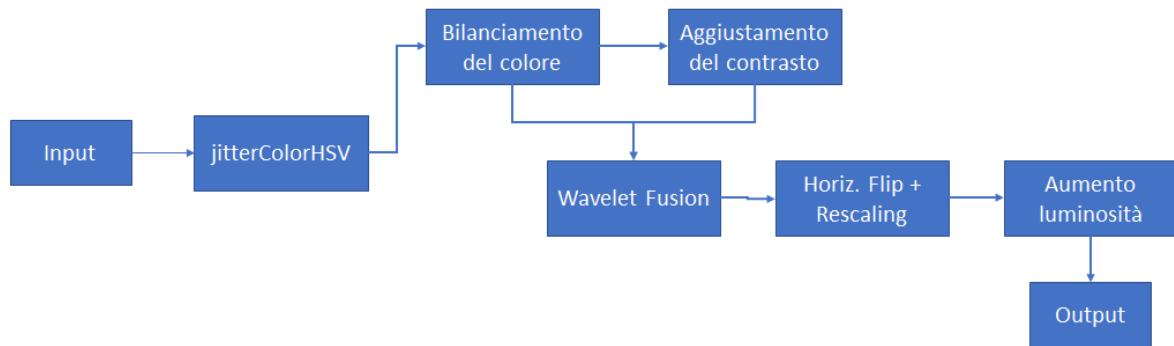


Figura 3.29: Schema di *data augmentation* per il *training*

- Il ***testing*** implementa solo i metodi descritti ai paragrafi §3.4.2, §3.4.3 e §3.4.4, dunque solo quelli deputati al miglioramento della qualità dell'immagine, e non alla creazione di pattern aggiuntivi. Il più semplice flusso di esecuzione è descritto nello schema riportato di seguito.

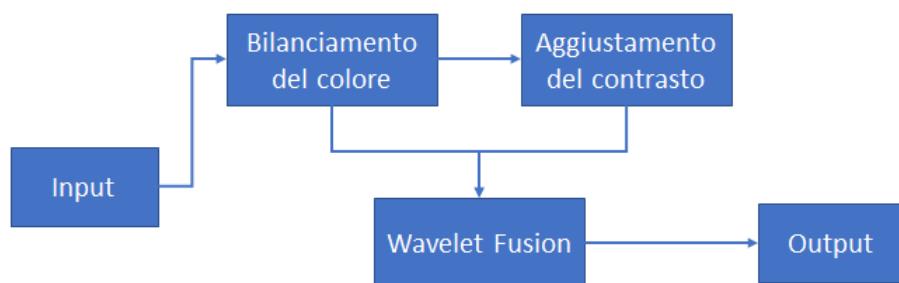


Figura 3. 30: Schema di *data augmentation* per il *testing*

3.5 Addestramento

In generale, in una rete neurale *feed-forward* come è YOLOv4, il processamento delle informazioni avviene in avanti, in quel passaggio che viene definito *forward propagation*, o inferenza. Durante l’addestramento però, al fine di migliorare l’output della rete, vengono aggiornati i pesi interni mediante il calcolo del gradiente della *loss function*, che nel caso della rete implementata è già stata descritta al §2.3.3, in quel processo che prende il nome di *backward propagation*, o “retropropagazione”. In base all’approccio implementativo scelto per il passo di *backward propagation*, gli algoritmi per l’addestramento sono molteplici.

In questo lavoro, si è scelto come algoritmo per l’addestramento *mini-batch gradient descent* [55]. Il *training set* viene ordinato casualmente e suddiviso in un certo numero di *mini-batches*, sottoinsiemi di n elementi del *training set*, rappresentanti l’unità di processamento della rete. Ogni iterazione di addestramento, ossia ogni presentazione alla rete dell’insieme degli n *patterns* costituenti un *mini-batch*, la rete calcola la *loss*, il gradiente relativo, ed aggiorna i pesi della rete in direzione opposta, ossia quella che minimizza la *loss function*, di un valore il cui modulo dipende dal *learning rate* preimpostato. Le iterazioni si susseguono fino al termine dei *mini-batches* con la presentazione alla rete di tutti i *patterns* del *training set*. A questo punto, quindi, l’addestramento prosegue per un numero predefinito di *epochs*, o “epoche”, eventualmente terminando prima (*early-stopping*) qualora la *loss function* assuma un valore piccolo e stabile, i.e. non varia significativamente susseguitesi un certo numero di iterazioni.

Lo pseudocodice viene riportato di seguito:

MINI-BATCH GRADIENT DESCENT	
<i>Input:</i> <i>training set, numEpochs, miniBatchSize, learningRate, loss, weights</i>	
1	do <i>epoch</i> \leftarrow <i>epochs</i> +1
2	shuffle <i>training set</i>
3	For each <i>mini-batch B of size miniBatchSize</i>
4	<i>params_gradient</i> \leftarrow <i>evaluate_gradient(loss, B, weights)</i>
5	<i>weights</i> \leftarrow <i>weights</i> – <i>learningRate</i> \times <i>params_gradient</i>
	end for
6	while <i>epoch</i> < <i>numEpochs</i> or early stopping
7	end do-while

La rete implementata utilizza i seguenti parametri di addestramento

- $\text{numEpochs} \leftarrow 50$
- $\text{miniBatchSize} \leftarrow 5$
- $\text{learningRate} \leftarrow 0.001$

Capitolo 4

Testing e risultati

In questo ultimo capitolo viene descritta nel dettaglio la procedura di testing, in particolare analizzando le metriche adottate per la misurazione delle prestazioni e raccogliendo per ogni dataset i risultati finali ottenuti.

Innanzitutto, però, si inizializzano due parametri fondamentali:

- *confidence threshold* (CT) $\leftarrow 0.5$: soglia di confidenza, se una *detection* non supera questa soglia di confidenza viene scartata.
- *overlap threshold* (OT) $\leftarrow 0.5$: soglia di sovrapposizione, indica l'IoU (§2.2) minimo fra una *detection* e il *ground truth* affinché essa possa essere corretta.

Preprocessing e *augmentation* avvengono come descritto rispettivamente al §3.3, e al §3.4.7 nella sezione “testing”.

4.1 Metriche

Prima di descrivere le metriche adottate per la valutazione delle prestazioni, è bene introdurre delle definizioni relative alla qualità delle *detection*.

- *True Positive* (TP): predetto positivo (con *bounding box* b) un oggetto positivo (con *ground truth* b_c) con $IOU(b, b_c) > OT$.
- *True Negative* (TN): predetto negativo (nessuna *bounding box*) un oggetto negativo (nessun *ground truth*).
- *False Positive* (FP): predetto positivo (con *bounding box* b) un oggetto negativo (nessun *ground truth*), oppure con *ground truth* b_c ma $IOU(b, b_c) < OT$. Viene anche definito errore di tipo *False*.

- *False Negative* (FN): predetto negativo (nessuna *bounding box*) un oggetto positivo (con *ground truth* b_c). Viene anche definito errore di tipo *Miss*.

4.1.1 Precision

La metrica *precision* indica quante *bounding boxes* vengono correttamente predette sul totale di quelle identificate. Dà una stima sulla precisione della rete.

$$\text{Precision} = \frac{TP}{TP + FP}$$
(4. 1)

4.1.2 Recall

La metrica *recall* indica quante *bounding boxes* vengono correttamente predette sul totale di quelle pertinenti. Dà una stima della sensibilità della rete.

$$\text{Recall} = \frac{TP}{TP + FN}$$
(4. 2)

4.1.3 Grafico precision-recall (PR)

Le metriche *precision* e *recall* possono essere rappresentate graficamente nell'omonima curva, ponendo nell'asse delle ascisse *recall*, e nelle ordinate *precision*.

Il grafico dovrebbe rappresentare una curva decrescente, dove al crescere di *recall* decresce *precision*, in quanto le due misurazioni sono in *trade-off*.

Un *detector* è quanto più performante quanto maggiore è l'area sottesa dalla curva. In particolare, il *detector* perfetto, ossia quello che non sbaglia né manca nessuna *detection*, ha area unitaria e la sua curva è uno scalino che per ogni *recall* assume valore di *precision* unitario.

In generale, un modello con elevato *recall* ma bassa *precision* tende a rilevare correttamente la maggior parte degli oggetti presenti, ma ne rileva anche molti di inesistenti.

Un modello con elevata *precision* ma basso *recall*, invece, tende a rilevare meno oggetti degli effettivi, tuttavia perlopiù correttamente.

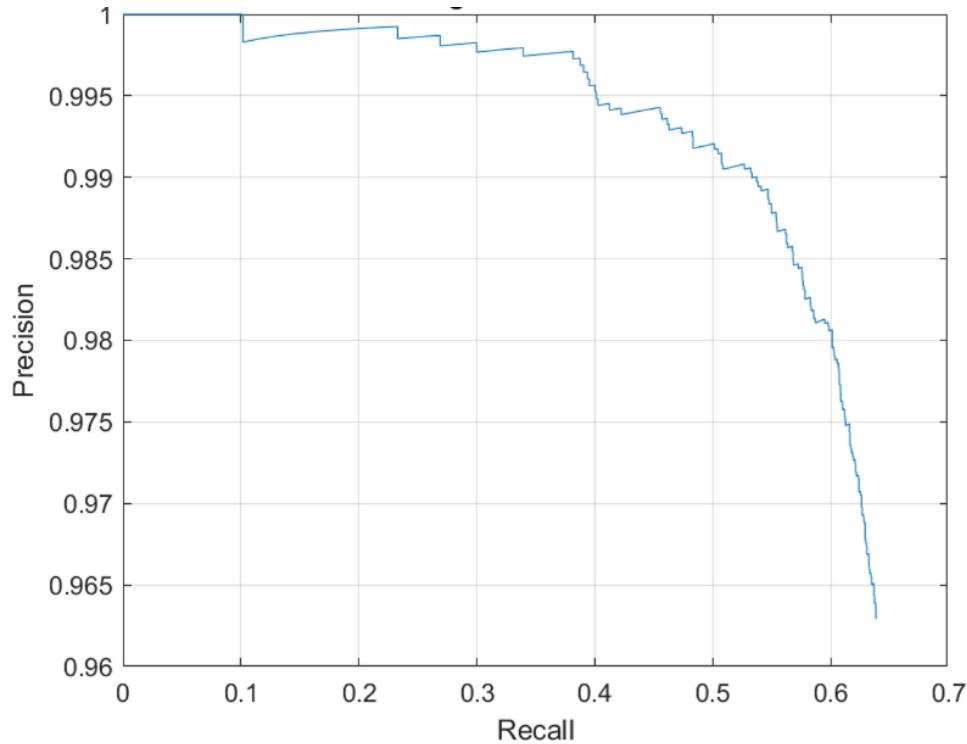


Figura 4.1: Esempio di grafico *precision-recall*

4.1.4 Average precision (AP)

La metrica *average precision* (AP), introdotta nell'articolo al [56], è definita come la precisione media (*mean precision*) corrispondente a 11 livelli equispaziati di *recall*.

Segue la formula relativa:

$$AP = \frac{1}{11} \sum_{Recall_i} Precision(Recall_i) \quad (4.3)$$

Dove:

- $i \in [0, 0.1, \dots, 1.0]$
- $Recall_i$ è l'intervallo di *recall* corrispondente al valore i
- $Precision(Recall_i)$ è la precisione relativa all'intervallo $Recall_i$

Di fatto AP è una stima dell'integrale della curva, dunque della sua area sottesa, che come già scritto al paragrafo precedente rappresenta un indicatore della performance del *detector*.

4.1.5 Mean average precision (mAP)

La metrica *mean average precision* (mAP) non è altro che la media delle AP di ciascuna classe. In questo lavoro, essendovi un'unica classe ('Fish'), mAP coincide con AP.

4.2 Test 1

4.2.1 Dataset

Il primo test viene effettuato su un dataset di 1974 elementi, comprendente:

- *Labeled Fishes in the Wild*
- *Aquarium Dataset*
- *Fish Dataset 416x416*
- *NorFisk Dataset*
- *LifeCLEF 2015 Fish task*

La lista completa dei datasets di testing si trova al §3.2.2.

Questi datasets sono quelli che hanno una sezione più consistente adibita al *training*.

4.2.2 Risultati

Si riporta innanzitutto un esempio di quattro frames di *detection*:



Figura 4.2: Esempio di *detection* in Test 1

Si riporta in seguito il grafico PR relativo:

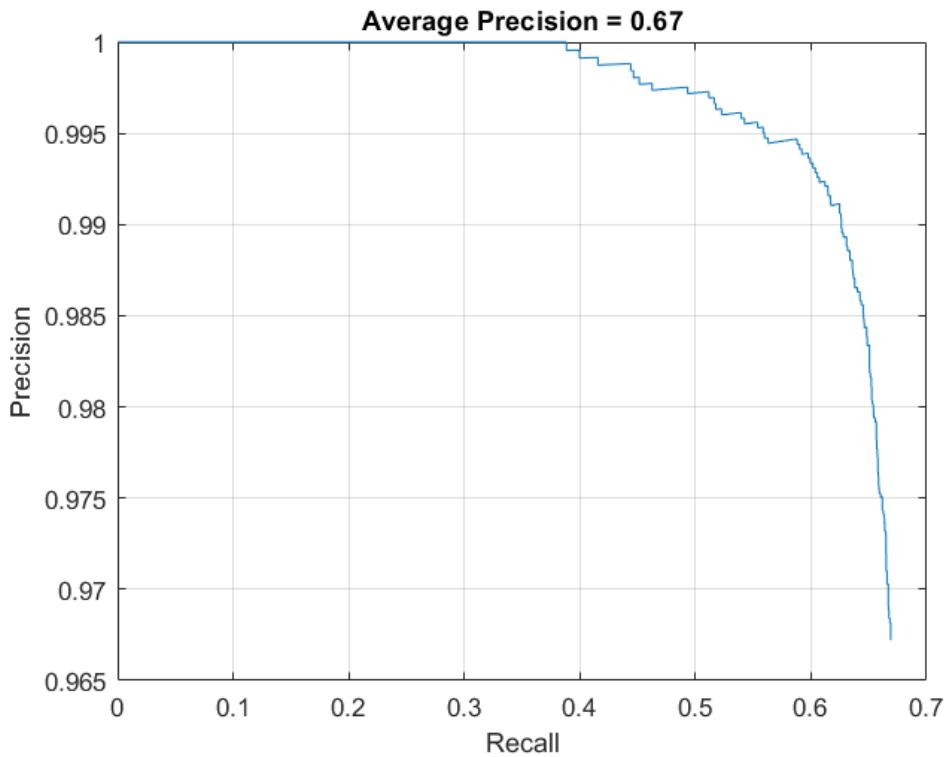


Figura 4.3: Grafico PR Test 1

Si riporta nella seguente tabella il valore di AP ottenuto.

Tests	AP
Test 1	0.6674

Tabella 4.1: Risultato Test 1

Si può osservare come *precision* tenda a mantenere un valore elevato, mentre *recall*, non supera 0.7. Questo significa in generale che il numero di FN prodotti dalla rete è piuttosto elevato, e le conseguenze sono descritte al §4.1.3.

Il valore ottenuto per AP è il punto di partenza per la valutazione dei tests successivi. Essendo l'*augmentation* effettuata volta ad ottimizzare le prestazioni della rete per il dataset di UNIVPM, è atteso un miglioramento di AP nel relativo test.

4.3 Test 2

4.3.1 Dataset

Il dataset utilizzato per valutare le prestazioni della rete nel secondo test è *Dataset dell'Università Politecnica delle Marche (UNIVPM)*, comprendente 207 immagini annotate su cui valutare la rete.

4.3.2 Risultati

Si riporta innanzitutto un esempio di quattro frames di *detection*:



Figura 4.4: Esempio di *detection* in Test 2

Si riporta in seguito il grafico PR relativo:

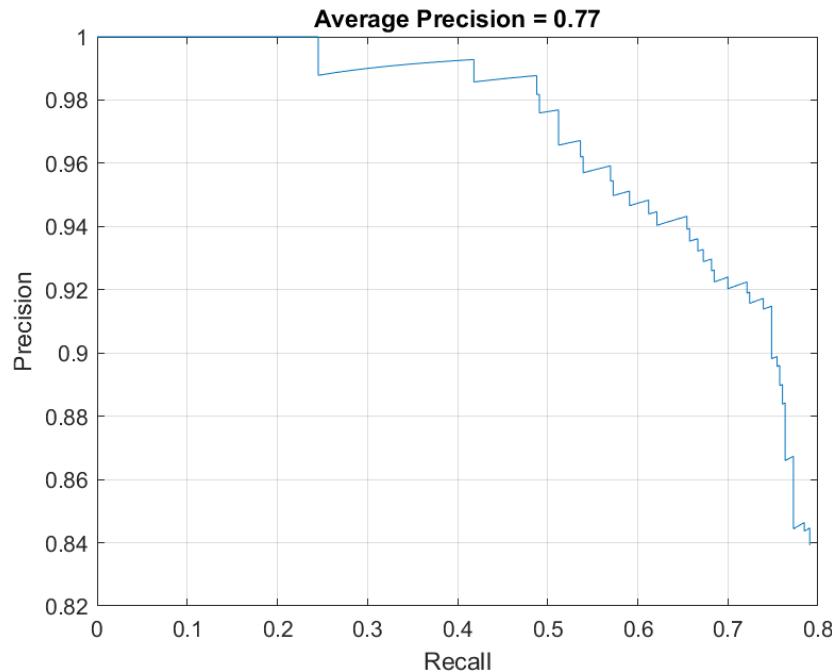


Figura 4.5: Grafico PR Test 2

Si riporta nella seguente tabella il risultato ottenuto relativamente ad AP.

Tests	AP
Test 1	0.6674
Test 2	0.7683

Tabella 4.2: Risultato Test 2

Come da previsione, il valore di AP è cresciuto di ben dieci punti percentuali rispetto al dataset precedente, dimostrando l'efficacia dei metodi implementati. Ad una lieve diminuzione della precisione della rete corrisponde un significativo aumento nella sua sensibilità, che migliora notevolmente AP.

4.4 Test 3

4.4.1 Dataset

Il dataset utilizzato per valutare le prestazioni della rete nel terzo ed ultimo test è *Dataset Liguria*, consistente di 160 immagini annotate su cui valutare le prestazioni della rete.

4.4.2 Risultati

Si riporta innanzitutto un esempio di quattro frames di *detection*:

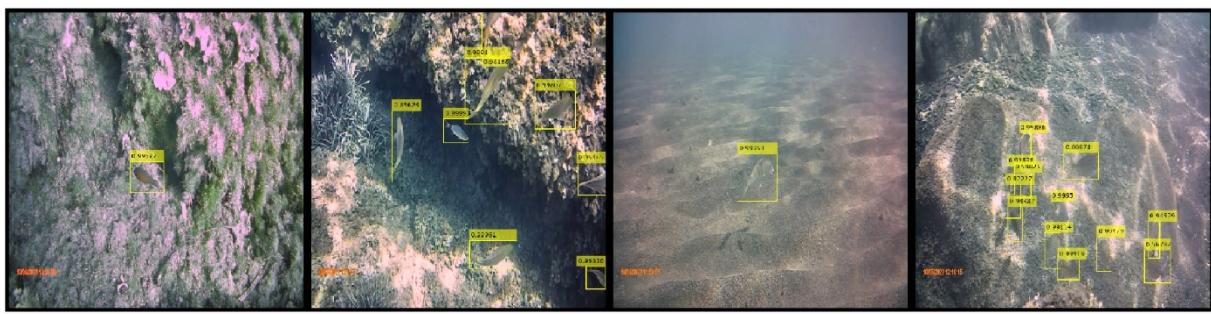


Figura 4.6: Esempio di *detection* in Test 3

Si riporta in seguito il grafico PR relativo:

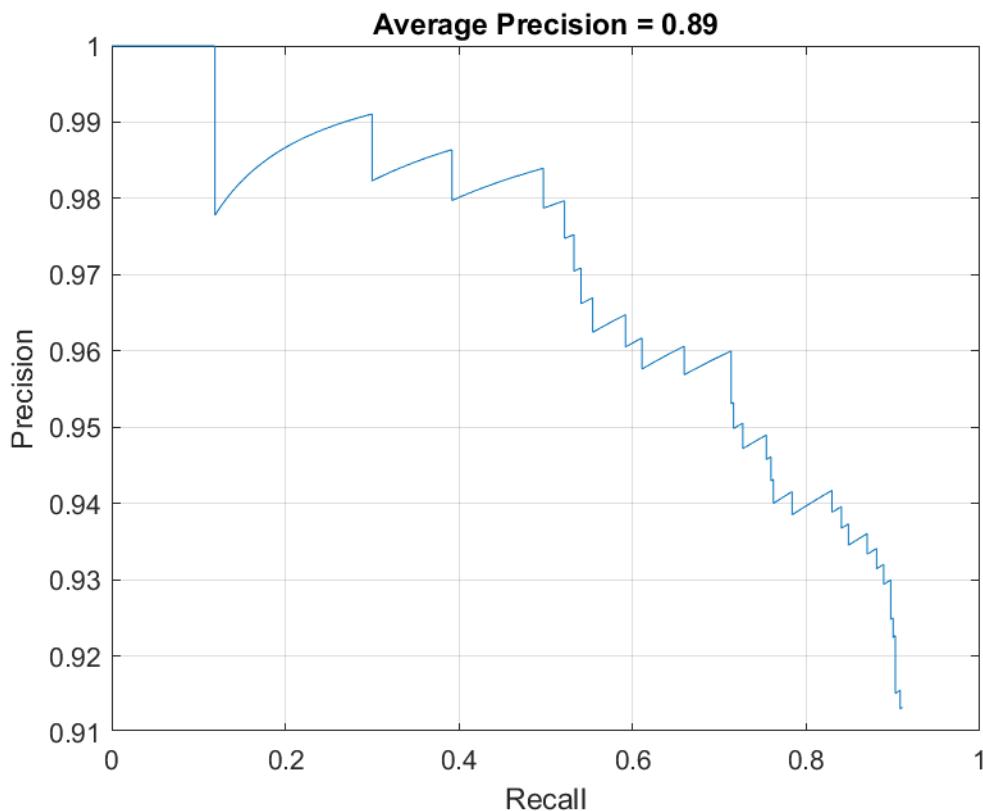


Figura 4.7: Grafico PR Test 3

Si riporta nella seguente tabella il valore di AP ottenuto.

Tests	AP
Test 1	0.6674
Test 2	0.7683
Test 3	0.8852

Tabella 4.3: Risultato Test 3

A questo dataset corrispondono le migliori prestazioni della rete, constatabile in un lieve aumento della precisione ed un incremento più significativo della sensibilità, che portano AP ad assumere il valore massimo fra i test.

Conclusioni

Il presente lavoro di tesi si prefiggeva come obiettivo quello di addestrare mediante *transfer learning* la rete neurale di YOLOv4, preaddestrata sul dataset *COCO*, al fine di eseguire il rilevamento automatico di specie ittiche marine nel loro ambiente naturale, con particolare riferimento al dataset fornito dall'Università Politecnica delle Marche.

L'obiettivo è di fatto stato raggiunto, con la produzione di una rete che con sole 50 epoche di addestramento raggiunge nel Test 2 un AP del 77%, complici le tecniche di *data augmentation* implementate, e con una velocità di inferenza elevata, grazie alle caratteristiche che contraddistinguono la rete YOLOv4. Questo risultato è di fatto notevole se si considera il fatto che la rete originaria YOLOv4, preaddestrata su *COCO* dataset, raggiunge su di esso un AP pari al 65.7% (Tabella 2.5), a parità di risoluzione di input (608x608) e di *overlap threshold* (OT) (§4.1), pari a 0.5. Bisogna però porre attenzione anche alle dimensioni dei datasets considerati, in quanto quello su cui è stata valutata nel Test 2 la rete prodotta da questo lavoro è enormemente ridotto rispetto a *COCO*, e dunque il valore ottenuto di AP è soggetto ad oscillazioni possibilmente significative qualora si dovesse espandere il dataset di test. Infatti, nel Test 1, AP si osserva scendere ad un valore di 67%, probabilmente a causa del dataset utilizzato, molto più ampio ed eterogeneo anche in termini di qualità. In casi dove l'immagine è di bassa qualità, infatti, è difficile per un *detector* distinguere chiaramente le *features* degli oggetti, e dunque aumenteranno i falsi negativi, che comporta la diminuzione di *recall* ed conseguentemente di AP. Nel Test 3, invece, AP si osserva salire a 89% per il motivo opposto, ossia le immagini sono perlopiù di alta qualità, con gli oggetti ben visibili, rendendo più accurato il processo di *detection*.

In base a quanto osservato, è possibile gettare le basi per un miglioramento futuro:

- Espansione del dataset di *testing* procedendo con l'inserimento di una quantità superiore di immagini di qualità ed annotate, per ottenere una stima più affidabile sul parametro AP

- Espansione del dataset di *training*: si sono riscontrate difficoltà nel reperimento di datasets per l’addestramento, a causa di una generale carenza di *patterns* adeguati con accesso libero online. Maggiori pubblicazioni future possono aiutare a creare un dataset di addestramento più bilanciato e nutrito, sempre con il fine di migliorare le prestazioni della rete.
- Potenziamento dell’*hardware* deputato all’addestramento della rete: l’accesso a *hardware* più potente può certamente aiutare la rete a generalizzare meglio le *features* dei *patterns* di addestramento, aumentando in particolare il numero di epoche di addestramento e la dimensione dei *mini-batches* in cui viene suddiviso il *training set*. Anche in questo caso l’effetto previsto coinciderebbe con l’aumento di AP.
- Nuove pubblicazioni relative alle tecniche di *data augmentation* implementate possono certamente aiutare a ridurre i tempi di elaborazione e migliorare la resa dell’immagine, sempre con l’obiettivo finale di migliorare le prestazioni della rete.
- Implementazione di un classificatore esterno che, a partire dalle *Regions of Interest* (ROI) prodotte dal *detector*, sia in grado di classificare i pesci rilevati secondo la propria specie, in modo da favorire le operazioni fondamentali di monitoraggio ambientale.

Riferimenti bibliografici

- [1] A. J. Richardson et al., «Climate change and marine life», *Biology Letters*, vol. 8, n. 6. The Royal Society, pagg. 907–909, lug. 11, 2012 [Online]. Available: <http://dx.doi.org/10.1098/rsbl.2012.0530>
- [6] “A comprehensive guide to fine-tuning Deep Learning models in keras (part I),” *Github.io*. [Online]. Available: <https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html>.
- [7] T. Evgeniou e M. Pontil, «Support Vector Machines: Theory and Applications», *Machine Learning and Its Applications*. Springer Berlin Heidelberg, pagg. 249–257, 2001 [Online]. Available: http://dx.doi.org/10.1007/3-540-44673-7_12
- [8] IBM topics, 2022. *Cos'è la Computer Vision?*. [online] Ibm.com. Available at: <<https://www.ibm.com/it-it/topics/computer-vision>>.
- [9] “Object detection vs object recognition vs image segmentation,” *GeeksforGeeks*, 27-Feb-2020. [Online]. Available: <https://www.geeksforgeeks.org/object-detection-vs-object-recognition-vs-image-segmentation/>.
- [10] A. R. Pathak, M. Pandey, e S. Rautaray, «Application of Deep Learning for Object Detection», *Procedia Computer Science*, vol. 132. Elsevier BV, pagg. 1706–1717, 2018 [Online]. Available: <http://dx.doi.org/10.1016/j.procs.2018.05.144>
- [11] S. Saha, «A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,» 15 Dicembre 2018. [Online]. Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.

- [12] The Mathworks, Inc., n.d. *Rete neurale convoluzionale*. [online] it.mathworks.com. Available at: <<https://it.mathworks.com/discovery/convolutional-neural-network-matlab.html>>.
- [13] IBM Cloud Education, 2020. *What are Convolutional Neural Networks?*. [online] Ibm.com. Available at: <<https://www.ibm.com/cloud/learn/convolutional-neural-networks#toc-how-do-con--z4UwR2M>>.
- [14] Z. Zou, Z. Shi, Y. Guo, e J. Ye, «Object Detection in 20 Years: A Survey». arXiv, 2019. doi: 10.48550/ARXIV.1905.05055.
- [15] L. Liu et al., «Deep Learning for Generic Object Detection: A Survey». arXiv, 2018 [Online]. Available: <https://arxiv.org/abs/1809.02165>
- [16] L. Jiao et al., «A Survey of Deep Learning-Based Object Detection», IEEE Access, vol. 7. Institute of Electrical and Electronics Engineers (IEEE), pagg. 128837–128868, 2019 [Online]. Available: <http://dx.doi.org/10.1109/ACCESS.2019.2939201>
- [20] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, e Li Fei-Fei, «ImageNet: A large-scale hierarchical image database», 2009 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, giu. 2009 [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2009.5206848>
- [22] J. Redmon, S. Divvala, R. Girshick, e A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection». arXiv, 2015 [Online]. Available: <https://arxiv.org/abs/1506.02640>
- [23] Padilla, Rafael & Netto, Sergio & da Silva, Eduardo. (2020). A Survey on Performance Metrics for Object-Detection Algorithms. 10.1109/IWSSIP48289.2020.
- [24] A. Rosebrock, “Intersection over Union (IoU) for object detection,” *PyImageSearch*, 07-Nov-2016. [Online]. Available: <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>.
- [25] J. Hui, “Real-time object detection with YOLO, YOLOv2 and now YOLOv3,” *Medium*,

- 18-Mar-2018. [Online]. Available: <https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>.
- [28] S.-H. Tsang, “Review: YOLOv2 & YOLO9000 — You Only Look Once (object detection),” *Towards Data Science*, 21-Nov-2018. [Online]. Available: <https://towardsdatascience.com/review-yolov2-yolo9000-you-only-look-once-object-detection-7883d2b02a65>.
- [29] J. Redmon e A. Farhadi, «YOLO9000: Better, Faster, Stronger». arXiv, 2016 [Online]. Available: <https://arxiv.org/abs/1612.08242>
- [30] T.-Y. Lin et al., «Microsoft COCO: Common Objects in Context». arXiv, 2014 [Online]. Available: <https://arxiv.org/abs/1405.0312>
- [31] J. Redmon e A. Farhadi, «YOLOv3: An Incremental Improvement». arXiv, 2018 [Online]. Available: <https://arxiv.org/abs/1804.02767>
- [32] S.-H. Tsang, “Review: YOLOv3 — You Only Look Once (object detection),” *Towards Data Science*, 07-Feb-2019. [Online]. Available: <https://towardsdatascience.com/review-yolov3-you-only-look-once-object-detection-eab75d7a1ba6>.
- [33] A. Bochkovskiy, C.-Y. Wang, e H.-Y. M. Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection». arXiv, 2020 [Online]. Available: <https://arxiv.org/abs/2004.10934>
- [34] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, e J.-W. Hsieh, «CSPNet: A New Backbone that can Enhance Learning Capability of CNN». arXiv, 2019 [Online]. Available: <https://arxiv.org/abs/1911.11929>
- [35] K. He, X. Zhang, S. Ren, e J. Sun, «Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition», Computer Vision – ECCV 2014. Springer International Publishing, pagg. 346–361, 2014 [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10578-9_23

- [36] S. Liu, L. Qi, H. Qin, J. Shi, e J. Jia, «Path Aggregation Network for Instance Segmentation». arXiv, 2018 [Online]. Available: <https://arxiv.org/abs/1803.01534>
- [37] Cutter, G.; Stierhoff, K.; Zeng, J. (2015) "Automated detection of rockfish in unconstrained underwater videos using Haar cascades and a new image dataset: labeled fishes in the wild," IEEE Winter Conference on Applications of Computer Vision Workshops, pp. 57-62.
- [38] Roboflow, "Aquarium Object Detection Dataset." 02-Aug-2022. Available: <https://public.roboflow.com/object-detection/aquarium>.
- [39] J. Solawetz, "Fish Object Detection Dataset - 416x416." 26-Aug-2020. Available: <https://public.roboflow.com/object-detection/fish/1>.
- [40] C. A. Maximiliano, «NorFisk Dataset». DataverseNO, 2020 [Online]. Available: <https://dataverse.no/citation?persistentId=doi:10.18710/H5G3K5>
- [41] "LifeCLEF 2015 fish task," *Imageclef.org*. [Online]. Available: <https://www.imageclef.org/lifeclef/2015/fish>.
- [42] G., Ravindran. (2017). Integrated Feature Extraction for Image Retrieval.
- [43] Yussof, Wan. (2013). Performing Contrast Limited Adaptive Histogram Equalization Technique on Combined Color Models for Underwater Image Enhancement.
- [44] "Estimate illuminant using gray world algorithm - MATLAB illumgray - MathWorks Italia," *Mathworks.com*. [Online]. Available: <https://it.mathworks.com/help/images/ref/illumgray.html>.
- [45] B. J. Lindbloom, "Chromatic Adaptation Evaluation" *Brucelindbloom.com*. [Online]. Available: <http://www.brucelindbloom.com/index.html?ChromAdaptEval.html>.
- [46] B. J. Lindbloom, "Chromatic Adaptation" *Brucelindbloom.com*. [Online]. Available: http://www.brucelindbloom.com/index.html?Eqn_ChromAdapt.html.
- [47] B. Dupont, "Honeycomb Grouper (Epinephelus merra)". flickr.com. [Online].

- Available: <https://www.flickr.com/photos/berniedup/8502593975/in/photostream/>.
- [49] D. J. Bora, A. K. Gupta, e F. A. Khan, «Comparing the Performance of L*A*B* and HSV Color Spaces with Respect to Color Image Segmentation». arXiv, 2015 [Online]. Available: <https://arxiv.org/abs/1506.01472>
- [50] M. Mathur e N. Goel, «Enhancement of Underwater images using White Balancing and Rayleigh-Stretching», 2018 5th International Conference on Signal Processing and Integrated Networks (SPIN). IEEE, feb. 2018 [Online]. Available: <http://dx.doi.org/10.1109/SPIN.2018.8474042>
- [51] A. Khan, S. S. A. Ali, A. S. Malik, A. Anwer, e F. Meriaudeau, «Underwater image enhancement by wavelet based fusion», 2016 IEEE International Conference on Underwater System Technology: Theory and Applications (USYS). IEEE, 2016 [Online]. Available: <http://dx.doi.org/10.1109/USYS.2016.7893927>
- [52] S. Masood et al., «Image Fusion Methods: A Survey», Journal of Engineering Science and Technology Review, vol. 10, n. 6. International Hellenic University, pagg. 186–195, 2017 [Online]. Available: <http://dx.doi.org/10.25103/jestr.106.24>
- [53] A. Haar, «Zur Theorie der orthogonalen Funktionensysteme», Mathematische Annalen, vol. 69, n. 3. Springer Science and Business Media LLC, pagg. 331–371, set. 1910 [Online]. Available: <http://dx.doi.org/10.1007/BF01456326>
- [54] G. Pajares e J. Manuel de la Cruz, «A wavelet-based image fusion tutorial», Pattern Recognition, vol. 37, n. 9. Elsevier BV, pagg. 1855–1872, set. 2004 [Online]. Available: <http://dx.doi.org/10.1016/j.patcog.2004.03.010>
- [55] S. Ruder, «An overview of gradient descent optimization algorithms». arXiv, 2016 [Online]. Available: <https://arxiv.org/abs/1609.04747>
- [56] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, e A. Zisserman, «The Pascal Visual Object Classes (VOC) Challenge», International Journal of Computer Vision, vol. 88, n. 2. Springer Science and Business Media LLC, pagg. 303–338, set. 09, 2009 [Online]. Available: <http://dx.doi.org/10.1007/s11263-009-0275-4>

