



Politecnico di Bari

Dipartimento di Ingegneria Elettrica e dell'Informazione

Corso di laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea Triennale in Calcolo Numerico

**Tecniche di Deep Learning
e Object Detection
per la foto-identificazione dei cetacei**

Relatori

Prof. Tiziano POLITI

Correlatore:

Dott. Vito RENÒ

Laureando

Gianvito LOSAPIO

ANNO ACCADEMICO 2018 – 2019



L'intero lavoro, inclusi i codici redatti, è soggetto alla Creative Commons Licence (CC-BY-SA).

Il file in formato .pdf è disponibile online, insieme al sorgente .tex, all'indirizzo:

<https://github.com/gvlos/Tesi-triennale>

Dichiaro sotto la mia personale responsabilità di avere rispettato tutte le leggi relative alla proprietà intellettuale e di copyright di tutti i brani citati.

Bari, 6 ottobre 2019

Gianvito LOSAPIO

.....

Sommario

Sono state create due differenti routine Matlab (cap. 5) in grado di riconoscere e ritagliare automaticamente pinne dorsali di delfino a partire da una collezione di immagini scattate durante campagne di avvistamento in mare.

Lo scopo finale del lavoro è facilitare lo studio dei cetacei, attorno al quale si riunisce grande interesse scientifico (par. 1.1), incentivando l'utilizzo di una tecnica non-invasiva basata su algoritmi innovativi e su una grande disponibilità di dati: la fotoidentificazione automatica degli esemplari (par. 1.2).

Le metodologie utilizzate spaziano dall'*Image Processing* (cap. 2) al *Machine Learning* (cap. 3). Una quantità considerevole di tecniche di calcolo numerico è alla base dei metodi utilizzati: alcune di esse sono state oggetto di studio durante il lavoro di tesi.

Complessivamente, il riconoscimento di una pinna all'interno dell'immagine è effettuato con i seguenti passaggi salienti, comuni ad entrambe le routine, ma implementati in maniera differente:

1. Segmentazione dell'immagine basata sui colori (par. 4.1)
2. Filtraggio e ritaglio adattivo delle regioni individuate sulla base di caratteristiche geometriche (par. 4.2 e 4.3)
3. Classificazione binaria: pinna/no pinna (4.4).

La fase di sperimentazione è stata effettuata a partire da un ampio dataset di immagini comprendente scatti collezionati negli ultimi anni da associazioni di ricerca nel Golfo di Taranto e alle isole Azzorre (par. 1.1). In particolare, a partire dalle proposte mutate da (Forenza, [4]):

- sono state valutate le prestazioni di un metodo basato sulla sogliatura di Otsu (par. 4.1.1);
- è stato introdotto un metodo di segmentazione automatica innovativo, basato esclusivamente su filtri progettati appositamente nello spazio di colore CIE 1976 $L^*a^*b^*$ (par. 4.1.2);
- sono state valutate tecniche di filtraggio delle regioni binarie ottenute in seguito alla segmentazione automatica (par. 4.2);

- è stata introdotta una tecnica di ritaglio adattivo, utile ad isolare le pinne in modo più preciso all'interno delle immagini, a partire dalle regioni binarie (par. 4.3);
- è stata progettata da zero l'architettura di una rete neurale convoluzionale, utile per effettuare una classificazione binaria delle immagini ritagliate (par. 4.4.2);
- sono state proposte due differenti strategie di classificazione dei ritagli. La prima di tipo *major voting* con cinque differenti classificatori addestrati secondo una procedura di *5-Cross Fold Validation*; l'altra basata su uno studio statistico effettuato sulle dimensioni dei ritagli e supportata da un unico classificatore (par. 4.4);
- è stato creato un nuovo dataset di dimensione $n \simeq 15000$ in cui, a ciascuna immagine, sono state associate le coordinate dei rettangoli che contengono le pinne dorsali, nell'ottica di indirizzare possibili sviluppi futuri (cap.7).

Il risultato finale del lavoro è la routine Matlab CropFin v2 (par. 5.2): sulla base di criteri di valutazione delle performance introdotti appositamente per le routine proposte (par. 4.5), essa ha registrato un'efficienza del 77 % nella corretta localizzazione e ritaglio delle pinne, percentuale che, ragionevolmente, consente di identificare la quasi totalità degli esemplari durante una campagna di avvistamento.

Ringraziamenti

Desidero ringraziare in primis coloro i quali hanno supervisionato il mio tirocinio presso il Consiglio Nazionale delle Ricerche, un'esperienza che ricorderò con immenso piacere poiché mi ha arricchito molto sia dal punto di vista professionale che dal punto di vista umano. In primo luogo ringrazio il Dott. Vito Renò per avermi concesso questa opportunità e per avermi dato fiducia sin dall'inizio e in ogni momento del lavoro. Colgo l'occasione per rinnovare la mia profonda stima nel suo ruolo di ricercatore, di docente e di tutor: la sua passione e dedizione al lavoro sono stati per me uno stimolo importante. Ringrazio la Dott.ssa Rosalia Maglietta, per la pazienza e la gentilezza con cui ha saputo coinvolgermi e indirizzarmi con utili consigli sul lavoro. Desidero ringraziare il prof. Politi, per la sua disponibilità nel ruolo di relatore e supervisore dell'intero lavoro di tesi. Ringrazio i miei colleghi e tutti gli altri ragazzi conosciuti al CNR: in particolare Tommaso Monopoli, senza cui probabilmente questo lavoro non sarebbe stato possibile, ed Emanuele Seller, per il confronto sempre aperto e la voglia di trovare soluzioni sempre migliori.

Ringrazio la mia Famiglia, perchè crede sempre in me ed è in ogni parte di questo lavoro.

Ringrazio Daniela, perchè, con i suoi umili granelli di saggezza, mi ha sostenuto in ogni momento e mi ha aiutato, in questi anni, nella incessante ricerca dei veri valori della carriera universitaria e degli stimoli necessari per affrontarla.

Indice

Elenco delle tabelle	8
Elenco delle figure	9
I Introduzione	13
1 Problema e obiettivi	15
1.1 Contesto del lavoro ed interessi scientifici	16
1.2 Foto-identificazione automatica del <i>Grampus Griseus</i>	17
1.3 Lavori simili	21
II Metodologie	25
2 Image processing	27
2.1 Pixel e tensori	27
2.1.1 Spazio di colore Lab	29
2.1.2 Collezioni di immagini in Matlab	31
2.2 Trasformazioni	31
2.2.1 Ridimensionamento	31
2.2.2 Rotazioni, riflessioni, traslazioni	33
2.2.3 Convoluzione	34
2.3 Problemi di <i>Computer Vision</i>	38
2.3.1 Segmentazione	38
2.3.2 Object detection	43
3 Machine Learning	45
3.1 Supervised Learning	47
3.1.1 Statistical Learning Theory	49
3.1.2 Underfitting vs Overfitting	53
3.2 Classificazione	55
3.3 Reti neurali	57

3.3.1	Metodo del gradiente	61
3.4	Reti neurali convoluzionali	62
3.4.1	Input Layer	64
3.4.2	Convolution Layer	64
3.4.3	Pooling layer	68
3.4.4	Fully Connected Layer	70
3.4.5	Image Preprocessing	71
3.5	Cross-validation	75
III	Soluzioni proposte	77
4	Esperimenti e risultati	79
4.1	Segmentazione basata sui colori	80
4.1.1	Soglia multipla secondo il metodo Otsu	80
4.1.2	Poliedri nello spazio Lab	85
4.2	Filtraggio di regioni connesse	92
4.3	Ritaglio adattivo	98
4.4	Classificazione	102
4.4.1	Creazione del dataset	102
4.4.2	Architettura	104
4.4.3	Addestramento	108
4.4.4	Salvataggio dei ritagli	111
4.5	Prestazioni	114
5	Descrizione delle routine	119
5.1	CropFin v1	120
5.2	CropFin v2	123
6	Listato di ulteriori codici	139
6.1	Convoluzione e crosscorrelazione	139
6.2	Reti convoluzionali e poliedri nello spazio Lab	144
7	Conclusioni e sviluppi futuri	153
	Bibliografia	155

Elenco delle tabelle

4.1	Modelli individuati per la soluzione al problema di segmentazione, a partire dal campione di dati selezionato	88
4.2	Codifica delle tonalità predominanti del mare utilizzate per la scomposizione del campione di dati selezionato. L'ultima colonna riporta la percentuale di immagini corrispondente rispetto al campione (di dimensione 1033).	89
4.3	Parametri delle reti CNN utilizzate	107
4.4	Media delle performance delle cinque reti neurali convoluzionali rispetto ai rispettivi <i>fold</i> di test. <i>True Positive</i> si riferisce alle classificazioni corrette della classe <i>Pinna</i>	110
4.5	Performance della classificazione rispetto ad un campione di ritagli ottenuti dall'elaborazione di un dataset differente, composto da immagini scattate nelle isole Azzorre	111
4.6	Parametri statistici ricavati dal campione dei ritagli della classe <i>Pinna</i> . Dimensione del campione $n = 7851$	113
4.7	Performance a confronto delle routine CropFin v1 e CropFin v2	116
4.8	Confronto dei parametri di efficienza per le routine CropFin v1 e CropFin v2	116

Elenco delle figure

1.1	Illustrazioni delle specie <i>Tusiops Truncatus</i> e <i>Grampus Griseus</i>	18
1.2	Esemplare di grampo nel Golfo di Taranto	18
1.3	Avvistamenti nel Golfo di Taranto e output dell'algoritmo <i>SPIR</i>	19
1.4	Visualizzazione del problema di ritaglio automatico affrontato nella presente tesi	20
1.5	Pipeline innovativa per la foto-identificazione del <i>Grampus Griseus</i>	22
1.6	Lavori simili	23
2.1	Visualizzazione grafica di un tensore	28
2.2	Scomposizione intuitiva di un'immagine nei canali RGB	29
2.3	Visualizzazione degli spazi di colore RGB, Lab	30
2.4	Visualizzazione grafica della tecnica di interpolazione bicubica	32
2.5	Visualizzazione delle trasformazioni geometriche	34
2.6	Cross-correlazione tra due matrici	36
2.7	Esempio numerico di convoluzione discreta	38
2.8	Proprietà di orientazione e connettività delle regioni connesse	42
3.1	Diagramma di Venn AI	46
3.2	Distribuzione di probabilità dei dati	50
3.3	Underfitting e overfitting rispetto a capacità, bias e varianza di uno stimatore puntuale	55
3.4	Neurone	60
3.5	Modello computazionale del neurone	60
3.6	Esempio di rete CNN	64
3.7	Convolution layer in una rete neurale convoluzionale	65
3.8	Output di un Convolution Layer	66
3.9	Esempio di (a) convoluzione dilatata con fattore L=2 (b) padding equidistribuito con fattore P=1	66
3.10	Profondità dei filtri attraverso una CNN	67
3.11	Operazione di pooling : (a) effetto di downsampling e (b) calcolo del max pooling	69
3.12	Utilizzo del pooling	70
3.13	Livello fully connected	71
4.1	Istogrammi e spazio colore Lab: (a) dell'immagine originale e (b) dell'immagine sottoposta a CLAHE. Scatto del 7 giugno 2019, grampi nel golfo di Taranto, crediti Emanuele Seller	81

4.2	Soglia di Otsu calcolata sugli istogrammi L e b	81
4.3	Segmentazione dell'immagine basata sulle soglie di Otsu per i canali L e b: (a) a colori, (b) binaria.	83
4.4	Esempio di inefficienza della sogliatura nel caso di mare con tonalità grigie: (a) immagine originale, (b) immagine binaria dopo la sogliatura. Scatto del 2018, grampi nel mare delle Azzorre.	84
4.5	Esempio di inefficienza della sogliatura in presenza di sfondo paesaggistico: (a) immagine originale, (b) immagine binaria dopo la sogliatura. Scatto del 2018, grampi nel mare delle Azzorre.	84
4.6	Esempio di inefficienza della sogliatura a causa del riflesso della luce: (a) immagine originale, (b) immagine binaria dopo la sogliatura. Scatto del 2016, grampi nel golfo di Taranto.	85
4.7	Esempio di segmentazione ottenuta mediante l'applicazione del filtro nello spazio Lab: (a) immagine originale, (b) immagine binaria dopo la sogliatura. Scatto del 2018, grampi nel golfo di Taranto.	88
4.8	Esempio di creazione del filtro per il modello "Grigio": (a) immagine iniziale, (b) annerimento dei pixel corrispondenti alla tonalità del mare "Grigio", (c) risultato della segmentazione manuale mediante app <i>Color Threshold</i> , (d) poliedro 3D corrispondente alle tonalità individuate . .	91
4.9	Esempio di filtraggio eseguito nella Routine v2: (a) risultato della segmentazione mediante filtro nello spazio Lab, (b) filtro sale e pepe, (c) riempimento degli <i>holes</i> , (d) risultato finale: mantenimento delle regioni con area più grande	94
4.10	Esempio di filtraggio eseguito nella Routine v2: (a) risultato della segmentazione mediante filtro nello spazio Lab, (b) filtro sale e pepe, (c) riempimento degli <i>holes</i> , (d) risultato finale: mantenimento delle regioni con area più grande	95
4.11	Filtraggio delle regioni connesse a due fasi: (a) applicazione del primo filtro sull'immagine intera a seguito della sogliatura con metodo Otsu, (n) estrazione di una regione dall'immagine originale, (o) applicazione ripetuta della sogliatura (senza CLAHE) , (p) applicazione del secondo filtro sul singolo ritaglio.	97
4.12	Esempio di ritaglio adattivo eseguito nella Routine v2: (a) immagine di partenza (b) calcolo dell'altezza di taglio (linea blu) e della coordinata h, (c) calcolo dell'ampiezza di taglio, (d) risultato finale, (e) risultato finale relativo all'altra pinna localizzata all'interno dell'immagine	101
4.13	Illustrazione del ritaglio adattivo eseguito nella Routine v1: (a) regione binaria di partenza, (b) riempimento degli <i>holes</i> , (c) localizzazione dei punti di interesse, (d) identificazione del rettangolo che li contenga, (e) traslazione ed estensione del rettangolo, (f)-(g) ritaglio di partenza e risultato finale.	103
4.14	Campione del dataset di addestramento dei classificatori	104
4.15	Architettura del classificatore	105
4.16	Addestramento del classificatore	111

4.17	Studio statistico delle dimensioni delle immagini	114
5.1	Processing dell'immagine nella routine CropFin v1	121
5.2	Diagramma di flusso della routine CropFin v2	125
5.3	Processing dell'immagine nella routine CropFin v2	126

Alcuni la chiamano la religione del nostro tempo.

Anche se ha rotto lo specchio del mondo in mille frammenti.

La scienza.

A volte sembra che sia lei a condurci per mano, senza dirci dove stiamo andando.

[V. JALONGO, Il senso della bellezza: arte e scienza al CERN]

Parte I

Introduzione

Capitolo 1

Problema e obiettivi

L'obiettivo principale del presente lavoro di tesi è stato quello di creare una routine Matlab in grado di riconoscere e ritagliare automaticamente pinne dorsali di delfini a partire da una collezione di immagini (1.4).

Lo scopo finale è facilitare lo studio dei cetacei, attorno al quale si riunisce grande interesse scientifico (par. 1.1), ed incentivare l'utilizzo di una tecnica non-invasiva basata su algoritmi innovativi e su una grande disponibilità di dati: la foto-identificazione automatica degli esemplari (par. 1.2).

Tecnicamente, il problema affrontato può essere ricondotto ad una particolare classe di problemi della *computer vision* nota come *object detection*, finalizzata al riconoscimento automatico di oggetti all'interno di immagini (par. 2.3.2).

Le metodologie risolutive utilizzate spaziano dall'*Image Processing* (cap. 2) al *Machine Learning* (cap. 3). Una quantità considerevole di tecniche di calcolo numerico è alla base dei metodi utilizzati all'interno del presente lavoro e, più in generale, all'interno di questo ampio spettro disciplinare: alcune di esse sono state oggetto di studio durante il lavoro di tesi e sono presentate nei capitoli successivi.

Le strategie sperimentate sono descritte in dettaglio nel capitolo 4. Il capitolo 5 contiene una descrizione delle routine implementate, mentre nel capitolo 7 sono esposte considerazioni finali sulle soluzioni proposte.

Il lavoro può essere, inoltre, inquadrato nell'ambito di una disciplina emergente al confine tra ecologia e informatica, nota come *ecological informatics* ovvero "ecoinformatica". Come riporta l'omonima rivista scientifica¹, essa si fonda sull'incontro tra la grande quantità di dati intrinsecamente connessa all'ecologia e la crescente capacità delle tecnologie dell'informazione di sfruttare dati complessi, con l'obiettivo di accelerare l'avvento dei processi sostenibili in vista dei cambiamenti climatici e ambientali globali.

¹*Ecological Informatics*, Elsevier

Nei paragrafi seguenti si chiarisce il contesto da cui è scaturito il presente progetto di tesi, si fornisce una descrizione più dettagliata del problema affrontato ed infine si espongono lavori simili all'interno del panorama scientifico internazionale.

1.1 Contesto del lavoro ed interessi scientifici

Il presente lavoro di tesi è stato sviluppato nel corso di un'attività di tirocinio presso l'Istituto *STIIMA*² del Consiglio Nazionale delle Ricerche di Bari, sotto la supervisione del Dott. Vito Renò e della Dott.ssa Rosalia Maglietta.

In particolare, la proposta di tesi è nata nell'ambito di una collaborazione dell'istituto con le associazioni di ricerca scientifica *Jonian Dolphin Conservation*³, finalizzata allo studio dei cetacei del Golfo di Taranto nel Mar Ionio Settentrionale, e *Nova Atlantis*⁴, finalizzata allo studio dei cetacei nell'area delle isole Azzorre (situate nell'Oceano Atlantico e politicamente appartenenti al Portogallo).

Il principale risultato della collaborazione con *Jonian Dolphin Conservation*, presentato in (Maglietta *et al.*, [1]), è stata la creazione di una piattaforma innovativa per lo studio dei delfini della specie *Grampus Griseus* nel Golfo di Taranto, chiamati comunemente delfini di Risso o grampi. La piattaforma⁵ contiene una serie di dati di avvistamento e foto georeferenziate collezionate tra il 2013 ed il 2016. È stato inoltre introdotto un tool innovativo per la foto-identificazione automatica degli esemplari, creando i principali presupposti per le idee alla base del presente lavoro di tesi. Questo aspetto è chiarito con maggior dettaglio nel paragrafo seguente 1.2. Si vogliono qui invece evidenziare i motivi principali dell'interesse scientifico verso lo studio dei delfini.

Il monitoraggio della distribuzione dei cetacei e della sua mutazione nel tempo costituisce un elemento determinante per la comprensione dell'alterazione degli ecosistemi marini. È noto, infatti, che lo stato di salute di ecosistemi con catene alimentari lunghe, come quello marino, si rifletta nella presenza e nell'abbondanza dei predatori dei livelli trofici superiori; tra questi figurano proprio i delfini.

Studiare i delfini significa, quindi, conoscere indirettamente lo stato di salute dell'ecosistema marino in cui sono inseriti. Ciò risulta di particolare rilevanza per il Golfo di Taranto, poichè soggetto (in passato) ad un grave livello di inquinamento ambientale a causa degli insediamenti industriali [2]. Il monitoraggio degli ecosistemi marini è, in ogni caso, considerato necessario secondo alcune recenti direttive emesse dall'Unione

²Sistemi e Tecnologie Industriali Intelligenti per il Manifatturiero Avanzato, stiima.cnr.it/it

³joniandolphin.it

⁴nova-atlantis.org

⁵<http://dolphin.ba.issia.cnr.it>

Europea⁶.

Quello marino è solo un esempio dei tanti ecosistemi costantemente minacciati dall'impatto delle attività antropiche; al giorno d'oggi si stima che alterazioni irreversibili siano già in atto⁷. Il tema dei cambiamenti climatici ha assunto, non a caso, importanza centrale nei dibattiti internazionali recenti e causa continue mobilitazioni di carattere globale.

1.2 Foto-identificazione automatica del *Grampus Griseus*

La foto-identificazione è una tecnica di studio non invasiva basata sull'ipotesi generale che ciascun individuo può essere identificato in maniera univoca all'interno della popolazione grazie a specifiche caratteristiche fisiche distintive.

L'identificazione dei singoli individui nell'ambito dello studio dei cetacei è ritenuta di particolare utilità poichè consente di ottenere conoscenze più dettagliate sulle relazioni sociali all'interno della popolazione e sull'ambiente marino in cui gli esemplari vivono.

I delfini della specie *Grampus Griseus* presentano numerose peculiarità che li rendono differenti dalle altre specie spesso associate all'immaginario comune, una per tutte quella del *Tursiops Truncatus*, talvolta detto, infatti, delfino comune. Inoltre, presentano un tratto distintivo, descritto in dettaglio più avanti, che li rende particolarmente adatti alla tecnica della foto-identificazione automatica. La fig. 1.1 mette a confronto le specie citate.

La specie *Grampus Griseus*, ai cui esemplari ci si riferisce nel seguito semplicemente come grampi, è distribuita dai tropici fino alle regioni temperate in entrambi gli emisferi terrestri. Un individuo in età adulta può avere una lunghezza compresa tra 2,5 e 4 metri ed un peso di 500-600 kg. A differenza del delfino comune, il capo si presenta senza rostro sul davanti e con un largo sfiatatoio nella parte superiore. La pinna dorsale, posta circa a metà del corpo, è particolarmente alta ed appuntita e presenta una curvatura piuttosto accentuata.

Come si può ben notare nella fig.1.2, l'intero corpo dei grampi, pinna dorsale inclusa, è generalmente ricoperto da una quantità notevole di graffi di tonalità molto chiara. La comparsa di questi segni avviene in modo graduale nel corso della vita dell'animale: alla nascita, infatti, il colore del corpo è scuro e uniforme. Si sostiene che essi siano

⁶EU Marine Strategy Framework Directive (<https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1495097018132&uri=CELEX:32017L0845>), Maritime Spatial Planning Directive (<https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32014L0089>)

⁷Si riporta la recente inchiesta giornalistica di Presa Diretta sullo stato di salute dell'Artico, disponibile online all'indirizzo <https://www.raiplay.it/video/2018/09/Presa-Diretta-Caldo-artico-84d14554-ad20-4524-a37e-80eb0c94e60c.html>



Figura 1.1: Illustrazione delle specie a confronto: (a) *Tursiops Truncatus* ovvero tursiope o delfino comune (crediti: animalbase.uni-goettingen.de), (b) *Grampus Griseus* ovvero grampo o delfino di Risso (crediti: associaciocetacea.org).

provocati dalle interazioni sia di carattere alimentare con le prede sia di carattere sociale con gli altri individui. Gli esemplari più anziani ne vantano così tanti da risultare praticamente bianchi.



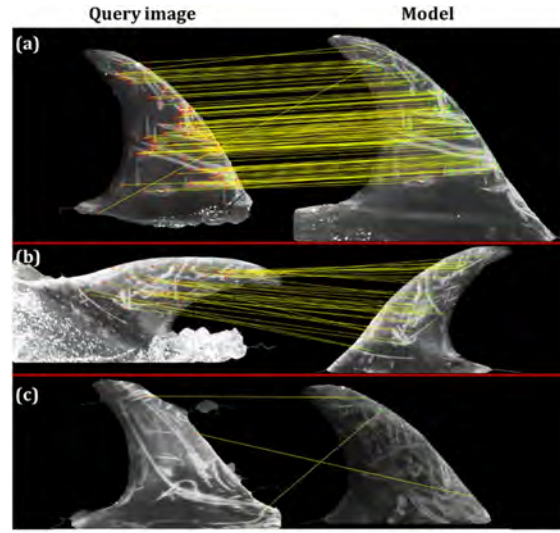
Figura 1.2: Esemplare di grampo nel Golfo di Taranto (crediti: Emanuele Seller)

La peculiarità appena descritta è proprio il tratto fisico distintivo che consente di applicare la tecnica di foto-identificazione agli esemplari di grampo. I soli graffi presenti sulla pinna dorsale sono, infatti, paragonabili alle impronte digitali dell'uomo: ogni individuo presenta un *pattern* di segni unico che lo contraddistingue all'interno della popolazione.

Nell'ambito dello studio (Maglietta *et al.*, [1]) già citato in precedenza, si è fatto uso di questa tecnica per stimare, a partire dalla grande quantità di immagini raccolta nel



(a)



(b)

Figura 1.3: Immagini tratte da (Maglietta *et al.*, [1]): (a) mappa del Golfo di Taranto con l'area degli avvistamenti effettuati; (b) esempio di output dell'algoritmo *SPIR*: nell'ordine predizione corretta, predizione corretta di una pinna ruotata, predizione non corretta;

corso di campagne di avvistamento effettuate negli ultimi anni dall'associazione *Jonian Dolphin Conservation*, la distribuzione, l'abbondanza e le relazioni sociali di una popolazione di grampi stanziata nel Golfo di Taranto, in un'area ristretta caratterizzata da un ripido pendio a circa 800 metri di profondità a nord del canyon della Valle di Taranto, mostrata in fig. 1.3(a).

Il risultato ottenuto si rivela di grande importanza poichè sopperisce ad una quasi totale assenza di dati: la *IUCN Red List of Threatened Species* considera, infatti, i grampi come specie "data deficient" nella zona del Mar Mediterraneo [3].

Come accennato nel paragrafo precedente, un altro risultato importante è stato l'introduzione di un tool innovativo per la foto-identificazione automatica, denominato *SPIR*

(*Smart Photo-Identification of Risso's dolphins*). Esso permette l'identificazione degli esemplari di grampo senza alcun intervento da parte dell'utente (fig. 1.3(b)). *SPIR* opera secondo i seguenti passaggi principali:

1. accetta in input un'immagine contenente una pinna dorsale, ritagliata a partire da uno degli scatti collezionati durante una campagna di avvistamento in mare;
2. applica algoritmi di *feature extraction* in grado di ottenere una rappresentazione dell'identità dell'individuo in termini di "descrittori" calcolati a partire dai graffi presenti sulla pinna;
3. applica algoritmi di *feature matching* in grado di associare l'identità calcolata ad uno degli individui precedentemente identificati e classificati all'interno di un database.

L'introduzione di *SPIR* costituisce il passo fondamentale per la concretizzazione della possibilità di impiegare la tecnica di foto-identificazione nell'ambito di nuovi studi su larga scala, superando parte delle criticità, temporali e non solo, legate all'analisi di grandi quantità di immagini in maniera manuale.

La completa automatizzazione della procedura resta, tuttavia, vincolata al superamento di un'ulteriore criticità: la necessità di ritagliare le pinne dorsali a partire dalle immagini iniziali (fig. 1.4).

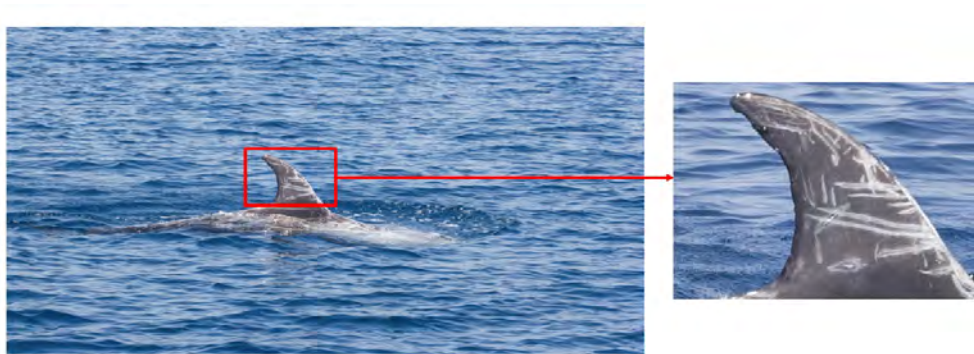


Figura 1.4: Visualizzazione del problema affrontato nella presente tesi: ritaglio automatico delle pinne dorsali a partire da un'immagine

La fase del ritaglio risulta, infatti, un collo di bottiglia all'interno della procedura di foto-identificazione automatica, poichè richiede un intervento manuale particolarmente dispendioso in termini temporali, paragonabile a quello richiesto per la foto-identificazione vera e propria.

Si inserisce in questo contesto l'obiettivo del presente lavoro di tesi: la possibilità di ritagliare automaticamente le pinne dorsali, a partire dalla collezione delle immagini scattate in mare, consacra in maniera definitiva la foto-identificazione automatica come la tecnica di studio non invasiva di maggiore efficienza per lo studio della specie

Grampus Griseus su larga scala.

Nel momento di avvio dei lavori, diversi esperimenti nella direzione del ritaglio automatico erano stati già compiuti nell'ambito di un precedente lavoro di tesi: (Forenza, [4]).

Il lavoro è consistito nel valutare i metodi proposti, integrare ed introdurre nuove tecniche per la risoluzione del problema di in esame (cap. 4). Come precedentemente accennato, esso può essere inquadrato nell'ambito di una più ampia classe di problemi della *computer vision*. Si rimanda, pertanto, al paragrafo dedicato alla descrizione del generico problema di *object detection* (2.3.2) per la menzione di alcune tecniche allo stato dell'arte, utilizzate come riferimento.

Il risultato del lavoro è stato la creazione di due differenti routine in grado di identificare e ritagliare automaticamente le pinne dorsali (cap. 5).

Un ulteriore lavoro di tesi, sviluppato in contemporanea al presente, ha portato all'ottimizzazione dell'algoritmo *SPIR* nella nuova versione *SPIR v2* (fig. 1.5) ed alla sua inclusione, insieme alla nuova procedura di ritaglio automatico, all'interno di una pipeline innovativa per la foto-identificazione del *Grampus Griseus* (Seller, [5]).

Nel seguito si menzionano alcuni lavori simili a quello appena esposto nell'ambito della *ecological informatics* (par. 1.3), rilevati all'interno della comunità scientifica.

1.3 Lavori simili

Lo studio (Buehler *et al.*, [6]) è particolarmente vicino al presente lavoro di tesi sia dal punto di vista contenutistico sia dal punto di vista cronologico. Esso inquadra, in termini più generali, nell'ambito dell'ecoinformatica, il problema del ritaglio automatico di regioni di interesse all'interno di immagini finalizzato alla foto-identificazione degli individui di una specie. In particolare, si propone:

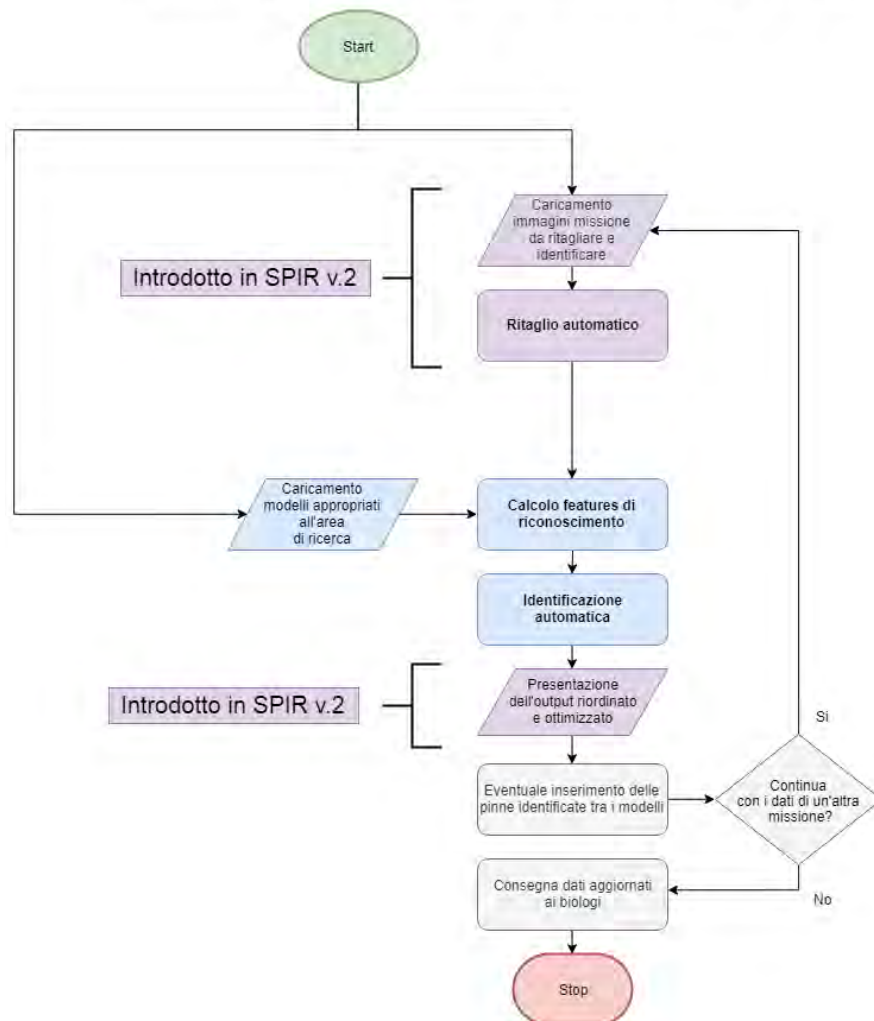
- il calcolo dei descrittori di HOG come tecnica principale per la *feature extraction*;
- l'utilizzo di un approccio *sliding window* per l'identificazione dell'area di interesse, supportato dalla classificazione di una *Support Vector Machine*.

Il dominio di applicazione affrontato è, in tal caso, il ritaglio automatico dei torsi delle giraffe (fig. 1.6(a)).

Nell'ambito dello studio e della foto-identificazione dei cetacei, si menziona la competizione *Humpback Whale Identification* pubblicata meno di un anno fa sulla piattaforma *Kaggle*⁸, con premio in denaro di 25,000\$. Obiettivo della competizione era sviluppare un sistema di foto-identificazione automatica delle balene, basato sulla forma delle code e sui segni su esse presenti (fig. 1.6(b)). Le soluzioni pubblicate utilizzano numerose

⁸<https://kaggle.com/c/humpback-whale-identification>

Flowchart per l'identificazione automatica degli esemplari

Figura 1.5: Flowchart di *SPIR v2*. Immagine tratta da (Seller, [5])

tecniche differenti, sia di *computer vision* che di *deep learning*. L'ente committente, *Happy Whales*⁹, è un'associazione di ricerca con obiettivi molto simili alle associazioni citate nel par. 1.1.

Si menziona, infine, anche il recente studio pubblicato da alcuni ricercatori della *Oxford University* (Schofield *et. al*, [7]), in cui si affronta il problema del riconoscimento automatico delle facce di un gruppo di scimpanzè all'interno di video mediante tecniche innovative di *deep learning* (fig. 1.6(c)). In particolare, il ritaglio automatico viene

⁹happywhale.com

risolto mediante l'utilizzo di un metodo denominato *Single Shot Detector*. Questo metodo è ripreso nel (cap. 7) poichè ritenuto di particolare interesse per gli sviluppi futuri del presente lavoro di tesi.

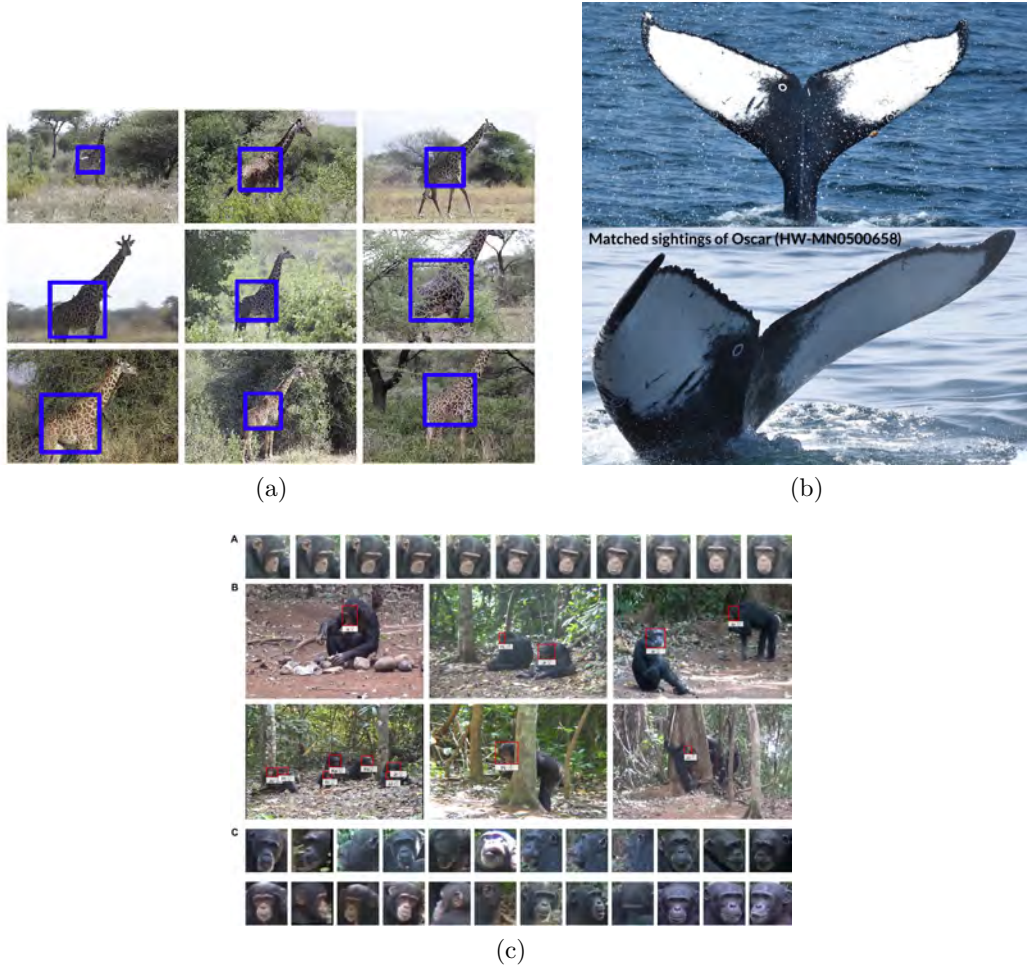


Figura 1.6: Immagini tratte dai lavori simili citati: (a) (Buehler *et al.*, [6]), ritaglio automatico del torso delle giraffe; (b) competizione *Humpback Whale Identification*, foto-identificazione automatica delle balene; (c) (Schofield *et. al.*, [7]), foto-identificazione automatica degli scimpanzè.

Parte II

Metodologie

Capitolo 2

Image processing

Nel corso di questo capitolo si introducono i concetti principali legati alla rappresentazione ed alla gestione delle immagini, con le relative tecniche di implementazione in Matlab utilizzate all'interno del presente lavoro di tesi. I principali riferimenti sono (Szeliski R., [8]), (Naldi *et al.*, [9]), (Luise *et al.*, [10]), (Giua *et al.*, [11]) .

2.1 Pixel e tensori

L'elaborazione delle immagini rientra in una più ampia branca delle tecnologie dell'informazione, nota in letteratura come elaborazione dei segnali digitali, spesso abbreviata come DSP (*Digital Signal Processing*).

Un'immagine è, infatti, considerata come un segnale discreto dello spazio bidimensionale. In particolare, nella grafica di tipo *raster*, lo spazio è costituito da una struttura dati a matrice detta *dot matrix*, visualizzabile attraverso un display. La più piccola unità indirizzabile all'interno dello spazio così definito è detta pixel.

L'intensità di ciascun pixel alla posizione (w, h) è proporzionale ad un certo valore di intensità luminosa, secondo un preciso modello di rappresentazione.

Generalmente, lo scatto di una fotocamera digitale consiste nella memorizzazione dell'intensità di ogni pixel alla posizione (w, h) , attraverso l'uso di tre componenti discrete dette Rosso (R, *Red*), Verde (G, *Green*) e Blu (B, *Blue*), ciascuna delle quali è compresa tra 0 e 255:

$$p(w, h) = [R(w, h), G(w, h), B(w, h)]$$

Questo tipo di rappresentazione è conforme ad uno dei modelli maggiormente utilizzati: lo spazio di colore sRGB (standard RGB, standardizzato nel 1999). Le immagini così rappresentate sono dette, per estensione, immagini RGB. Le componenti sono ottenute integrando il segnale luminoso catturato ad ogni pixel $L(\lambda)$ (funzione della lunghezza

d'onda λ):

$$\begin{aligned} R &= \int d\lambda L(\lambda) S_R(\lambda) \\ G &= \int d\lambda L(\lambda) S_G(\lambda) \\ B &= \int d\lambda L(\lambda) S_B(\lambda) \end{aligned}$$

dove $S_R(\lambda)$, $S_G(\lambda)$ e $S_B(\lambda)$ costituiscono la sensitività spettrale dei sensori del Rosso, del Verde e del Blu.

La struttura dati utilizzata per la rappresentazione di tutti i pixel di un'immagine RGB è detta matrice multidimensionale o tensore, visualizzata in fig. 2.1. Essa è costituita da tre matrici di uguale dimensione. Ciascun pixel p è individuato per mezzo di tre indici (i, j, k) :

- i, j è la coppia di indici che consente il consueto accesso a ciascuna delle tre matrici, indicando la riga i e la colonna j ;
- k è utilizzato per indicare a quale delle matrici si riferisce effettivamente la coppia i, j . Questa terza coordinata è detta livello oppure pagina.

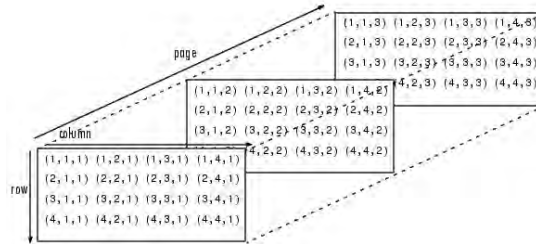


Figura 2.1: Visualizzazione grafica di un tensore. Immagine riprodotta da <https://it.mathworks.com/help/matlab/math/multidimensional-arrays.html>

La matrice del primo livello rappresenta le intensità del Rosso di tutti i pixel dell'immagine ed è detta, pertanto, canale Rosso; seguono, nell'ordine, il canale Verde ed il canale Blu.

In Matlab è possibile importare un'immagine RGB con la funzione:

```
im = imread("immagine.jpg");
```

ottenendo in `im` un dato di tipo `multidimensional-array` in cui ciascun elemento è di tipo `uint8` (4 byte), con dimensione $H \times W$, dove H rappresenta l'altezza dell'immagine e W la sua larghezza (in numero di pixel).

I canali Rosso `R`, Verde `G` e blu `B` possono essere ottenuti come:

```
R = im(:,:,1);
G = im(:,:,2);
B = im(:,:,3);
```

La tonalità effettiva del pixel \mathbf{p} visualizzata sul display alla posizione (i, j) (relativa all'immagine) è quindi fornita dalla tripletta di colore RGB:

```
p = [im(i,j,1) im(i,j,2) im(i,j,3)];
```

La figura 2.2 mostra in maniera intuitiva la scomposizione di un'immagine nei tre canali.

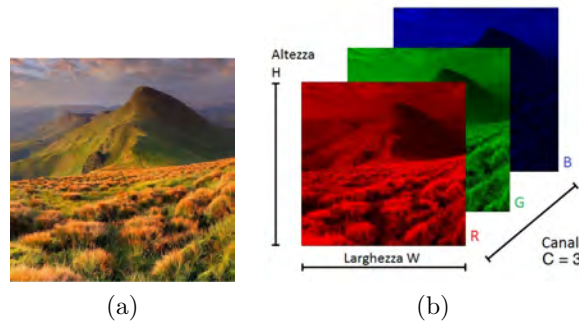


Figura 2.2: Scomposizione intuitiva di un'immagine nei canali RGB: (a) originale, (b) scomposta

Utilizzando il simbolo \mathbf{X} per la rappresentazione di un tensore, un'immagine RGB di dimensione $W \times H$ può essere indicata con la seguente notazione matematica:

$$\mathbf{X} \in \{0, \dots, 255\}^{W \times H \times 3}$$

2.1.1 Spazio di colore Lab

Oltre al modello RGB descritto al paragrafo precedente, esistono ulteriori spazi di colore che consentono di ottenere una differente rappresentazione delle medesime tonalità dei pixel.

Nel presente lavoro di tesi (par. 4.1) si utilizza una conversione delle immagini dallo spazio di colore iniziale sRGB allo spazio di colore CIE 1976 $L^*a^*b^*$, nel seguito abbreviato come Lab, standardizzato dalla *International Commission on Illumination*.

Nello spazio Lab le tonalità sono ancora espresse da triplette di valori (L^* , a^* e b^*), ma con un significato diverso rispetto a RGB: il valore L^* rappresenta la luminanza (variazione di luminosità), mentre le altre due rappresentano la cromaticanza (variazione di colore), rispettivamente una scala verde-rosso (a^*) ed una scala blu-giallo (b^*).

Lo spazio Lab è definito secondo una mappatura non lineare di un ulteriore spazio di colore denominato XYZ. La sua introduzione nel 1976 è stata motivata dalla necessità

di ottenere una rappresentazione tricromatica delle tonalità che rispecchiasse maggiormente la natura logaritmica del sistema visivo umano, il quale può percepire differenze relative di luminanza di solo 1%.

Proiettando tutte le tonalità definite nello spazio sRGB (fig. 2.3(a)) all'interno dello spazio Lab si ottiene un particolare poliedro, mostrata in fig. 2.3(b). A partire dal-

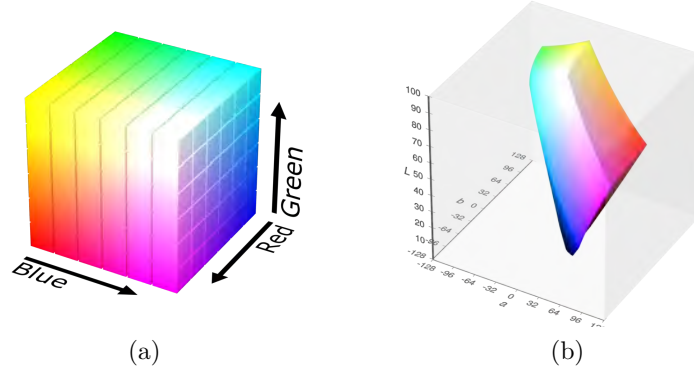


Figura 2.3: Visualizzazione degli spazi di colore (*gamut*): (a) RGB come cubo di dimensioni $255 \times 255 \times 255$, (b) Lab

l'immagine `im` è possibile convertire lo spazio di colori da RGB a Lab, ottenendo una nuova immagine `im_lab`:

```
im_lab = rgb2lab(im);
```

La funzione `rgb2lab` opera una doppia conversione:

1. da RGB a XYZ

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{0.17697} \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

2. da XYZ a Lab:

- $L = 116 f(Y)$, dove f rappresenta un'approssimazione della radice cubica, usata anche per le componenti a e b :

$$f(t) = \begin{cases} t^{1/3} & \text{se } t > \delta^3 \\ t/(3\delta^2) + 2\delta/3 & \text{altrimenti} \end{cases}$$

con $\delta = 0.2069$;

- $a = 500 \left[f\left(\frac{X}{0.9504}\right) - f(Y) \right];$
- $b = 200 \left[f(Y) - f\left(\frac{Z}{1.0888}\right) \right].$

2.1.2 Collezioni di immagini in Matlab

Una delle necessità principali del lavoro affrontato è stata la gestione di grandi quantità di immagini. Si riportano i tipi di dato che Matlab mette a disposizione per lo scopo:

- **3-D double** di dimensione $m \times n \times r$ **double** : stack di r immagini in bianco e nero di dimensione $m \times n$. In questo caso, la terza coordinata del tensore indicizza ogni singola immagine.
- **4-D double** di dimensione $m \times n \times r \times d$ **double**: uno stack di immagini RGB, in cui la quarta coordinata indicizza ogni singola immagine.
- **imageDatastore**: oggetto progettato appositamente per gestire ed elaborare rapidamente una grande quantità di immagini. A partire da un percorso **path** è possibile definire un oggetto che contenga l'elenco di tutte le immagini contenute in esso. È possibile, inoltre, includere le immagini contenute nelle sottocartelle (opzione **'IncludeSubfolders', true**) ed assegnare a ciascuna immagine un'etichetta data dal nome della cartella in cui è contenuta (opzione **'LabelSource', 'foldernames'**).

```
imds = imageDatastore(path,'IncludeSubfolders',true, ...  
'LabelSource','foldernames');
```

L'elenco delle immagini è restituito nel campo **imds.Files**. Il comando riportato è stato utilizzato nella funzione 6.2 sulla base di quanto riportato al par. 4.4.3.

2.2 Trasformazioni

Si riportano le operazioni fondamentali che hanno consentito, nel presente lavoro di tesi, di trasformare le immagini in una forma maggiormente adatta ad un'analisi successiva.

2.2.1 Ridimensionamento

Il ridimensionamento consente, in generale, di ottenere una nuova immagine a risoluzione differente. Questo può significare sia ridurre sia aumentare il numero dei pixel utilizzati per la sua rappresentazione.

Nel caso del presente lavoro, le dimensioni delle immagini sono state ridotte (par. 4.1): questa operazione di *preprocessing* è stata adottata per diminuire il costo computazionale delle operazioni successive. A partire da un'immagine **im** si è calcolata la sua versione ridimensionata **im_res** a 1200×800 :

```
im_res = imresize(im,[800 1200]);
```

Generalmente, la risoluzione di partenza delle immagini utilizzate è stata 6000×4000 , ottenendo una riduzione drastica di pixel del 96%, da 24 milioni a 960 mila.

La funzione **imresize** attua, per lo scopo, una tecnica avanzata di calcolo numerico: l'interpolazione bicubica, descritta nel paragrafo successivo.

Interpolazione bicubica

Le tecniche di interpolazione consentono di approssimare una funzione f tramite una funzione g , a partire dalla conoscenza del valore che la funzione f assume in alcuni punti x_i . Si richiede, in particolare, che sia verificata la condizione $g(x_i) = f(x_i)$ per tutti i punti x_i noti.

Il ridimensionamento risulta, a tutti gli effetti, una operazione di interpolazione: si vuole costruire la nuova immagine ridimensionata g a partire dall'originale f , approssimando f nei punti dello spazio della nuova dimensione.

Uno degli algoritmi più utilizzati è, come nel caso della funzione `imresize`, l'interpolazione bicubica. Tale metodo tenta di ricostruire, mediante una funzione cubica bidimensionale, l'esatta "superficie" tra gruppi di 16 pixel al fine di stimare l'intensità di un pixel intermedio, come mostrato in fig. 2.4 Si considera anzitutto una matrice F

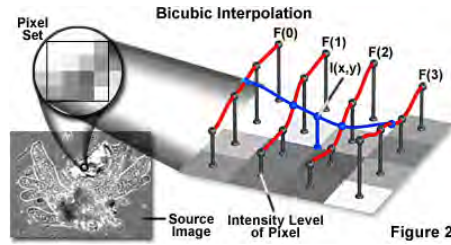


Figura 2.4: Visualizzazione grafica della tecnica di interpolazione bicubica

di dimensione 4×4 composta da pixel vicini dell'immagine di partenza, la cui intensità è denotata con f :

$$F = \begin{bmatrix} f(-1, -1) & f(-1, 0) & f(-1, 1) & f(-1, 2) \\ f(0, -1) & f(0, 0) & f(0, 1) & f(0, 2) \\ f(1, -1) & f(1, 0) & f(1, 1) & f(1, 2) \\ f(2, -1) & f(2, 0) & f(2, 1) & f(2, 2) \end{bmatrix}$$

La funzione cubica cercata è nella forma:

$$f(x, y) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j$$

Riscrivendo $f(x, y)$ in forma matriciale si ottiene:

$$f(x, y) = \begin{bmatrix} x^3 & x^2 & x & 1 \end{bmatrix} \underbrace{\begin{bmatrix} a_{3,3} & a_{3,2} & a_{3,1} & a_{3,0} \\ a_{2,3} & a_{2,2} & a_{2,1} & a_{2,0} \\ a_{1,3} & a_{1,2} & a_{1,1} & a_{1,0} \\ a_{0,3} & a_{0,2} & a_{0,1} & a_{0,0} \end{bmatrix}}_A \begin{bmatrix} y^3 \\ y^2 \\ y \\ 1 \end{bmatrix} \quad (2.1)$$

La matrice A può essere ottenuta considerando l'equazione $F = BAB^\top$, ottenuta considerando:

$$F = \underbrace{\begin{bmatrix} (-1)^3 & (-1)^2 & -1 & 1 \\ 0^3 & 0^2 & 0 & 1 \\ 1^3 & 1^2 & 1 & 1 \\ 2^3 & 2^2 & 2 & 1 \end{bmatrix}}_B \underbrace{\begin{bmatrix} a_{3,3} & a_{3,2} & a_{3,1} & a_{3,0} \\ a_{2,3} & a_{2,2} & a_{2,1} & a_{2,0} \\ a_{1,3} & a_{1,2} & a_{1,1} & a_{1,0} \\ a_{0,3} & a_{0,2} & a_{0,1} & a_{0,0} \end{bmatrix}}_A \underbrace{\begin{bmatrix} (-1)^3 & 0^3 & 1^3 & 2^3 \\ (-1)^2 & 0^2 & 1^2 & 2^2 \\ -1 & 0 & 1 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{B^\top}$$

Risolvendo per A si ottiene:

$$F = BAB^\top \implies A = B^{-1}F(B^\top)^{-1} = B^{-1}F(B^{-1})^\top$$

Sostituendo il risultato ottenuto in 2.1 si ottiene la soluzione al problema di interpolazione bicubica. L'intensità di ogni pixel $g(x, y)$ compreso tra i 4 punti centrali della matrice di partenza può essere calcolata nella nuova immagine ridimensionata come:

$$g(x, y) = \begin{bmatrix} x^3 & x^2 & x & 1 \end{bmatrix} B^{-1}F(B^{-1})^\top \begin{bmatrix} y^3 \\ y^2 \\ y \\ 1 \end{bmatrix} \quad (2.2)$$

in cui

$$B = \begin{bmatrix} -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \end{bmatrix}, \quad B^{-1} = \begin{bmatrix} -1/6 & 1/2 & -1/2 & 1/6 \\ 1/2 & -1 & 1/2 & 0 \\ -1/3 & -1/2 & 1 & -1/6 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

2.2.2 Rotazioni, riflessioni, traslazioni

In questo paragrafo si prendono in considerazione alcune semplici trasformazioni geometriche, utilizzate all'interno del presente lavoro nell'ambito dell'*image augmentation* (par. 4.4.3).

Le operazioni consistono generalmente nel trasformare le coordinate di ciascun pixel (x, y) in nuove coordinate (x', y') e, eventualmente, nell'utilizzare un'operazione di interpolazione per calcolare l'intensità dei pixel nelle nuove posizioni.

La rotazione può essere effettuata mediante il prodotto matriciale

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

in cui α è l'angolo di rotazione misurato in senso antiorario rispetto all'asse x .

La riflessione rispetto all'asse x può essere, invece, calcolata mediante il prodotto matriciale:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

La terza trasformazione utilizzata è la traslazione rispetto all'asse x: a differenza delle precedenti operazioni, la traslazione è una trasformazione affine ma non lineare, quindi la rappresentazione con le matrici richiede il passaggio ad un diverso tipo di coordinate dette omogenee:

$$\begin{bmatrix} x & y \end{bmatrix}^T \mapsto \begin{bmatrix} x & y & 1 \end{bmatrix}^T$$

A tal punto, è possibile ottenere le nuove coordinate traslate (x', y') a partire dalla forma matriciale:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

A partire dall'immagine `im` è possibile utilizzare le seguenti funzioni Matlab per eseguire le operazioni appena descritte, nell'ordine:

```
rotated = imrotate(im,20);  
translated = imtranslate(im,[-100 0]);  
flipped = flipdim(im,2);
```

I risultati sono mostrati nella figura 2.5.



(a)



(b)



(c)



(d)

Figura 2.5: Visualizzazione delle trasformazioni geometriche: (a) immagine originale, (b) rotazione antioraria di 20 gradi, (c) traslazione sull'asse x di -100 pixel, (d) immagine speculare.

2.2.3 Convoluzione

Nell'ambito dell'*image processing* esiste una quantità notevole di operatori definiti "locali": il loro valore non dipende dall'intensità di un solo pixel, ma anche da quella di

un gruppo di pixel nelle immediate vicinanze.

L'operatore locale maggiormente usato è l'operatore di convoluzione, grazie al quale è possibile realizzare un filtro lineare. Nell'ambito del presente lavoro, esso ha un'importanza centrale per la *feature extraction* delle immagini, all'interno del classificatore utilizzato (par. 3.4). Nel seguito si presenta, pertanto, una definizione rigorosa dell'operazione di convoluzione e si forniscono semplici esempi di implementazione.

Dati due segnali reali $f(t)$ e $g(t)$, si definisce convoluzione di f con g una nuova funzione h della variabile reale t

$$h(t) = f * g(t) \stackrel{\text{def}}{=} \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau$$

La notazione $f * g$ indica che il risultato dell'operazione si costruisce applicando l'operatore $*$ detto integrale di convoluzione (o talvolta impropriamente prodotto di convoluzione).

L'operatore di convoluzione gode delle proprietà:

- commutativa: $f * g(t) = g * f(t)$
- associativa: $[f * g] * r(t) = f * [g * r](t)$
- distributiva rispetto alla somma $[f(t) + g(t)] * r(t) = [f(t) * r(t)] + [g(t) * r(t)]$

Nel caso in cui i segnali siano definiti a intervalli discreti $f(k)$ e $g(k)$, si definisce (somma di) convoluzione tra f e g una nuova funzione $h(k)$ definita come

$$s(k) = x * w(k) \stackrel{\text{def}}{=} \sum_{a=-\infty}^{+\infty} x(a)w(k - a) \quad , \quad a \in \mathbb{Z}$$

La somma di convoluzione gode naturalmente delle stesse proprietà previste per l'integrale di convoluzione tra segnali reali continui.

Nell'ambito della *computer vision* si utilizza una versione multidimensionale dell'operazione, utilizzando la seguente terminologia per indicare le componenti del filtro:

- il primo segnale x è detto input, generalmente costituito da un'immagine od una sua elaborazione;
- il secondo segnale w è detto filtro o *kernel*, solitamente costituito da una matrice di dimensioni ridotte rispetto all'input;
- il risultato s è detto *feature map*, poichè l'operazione di filtraggio è spesso utilizzata per l'estrazione di *feature* a partire dall'input.

È chiaro che la somma di infiniti termini prevista dalla definizione di convoluzione con segnali discreti si riduce ad una somma limitata alle dimensioni dell'immagine.

Nel caso in cui l'input sia una matrice bidimensionale, ad esempio un'immagine in scala

di grigi $I \in \mathbb{R}^{m \times n}$, anche il kernel impiegato è solitamente una matrice bidimensionale di dimensioni ridotte $K \in \mathbb{R}^{l \times w}$, ottenendo l'operazione:

$$S(i, j) = (I * K)(i, j) = \sum_{k=1}^l \sum_{r=1}^w I(i + k - 1, j + r - 1) K(l - k + 1, w - r + 1)$$

L'implementazione è fornita alla sezione 6.1.

Molte librerie, in realtà, implementano la convoluzione attraverso la funzione di cross-correlazione. In tal caso ciascun elemento della *feature map* si ottiene come

$$F_{map}(i, j) = (I \circledast K)(i, j) = \sum_{k=1}^l \sum_{r=1}^w I(i + k - 1, j + r - 1) K(k, r)$$

in cui si è usato \circledast per distinguere l'operatore di crosscorrelazione da quello di convoluzione pura. La fig. 2.6 mostra una visualizzazione grafica intuitiva dell'operazione, come scorrimento del kernel sull'input.

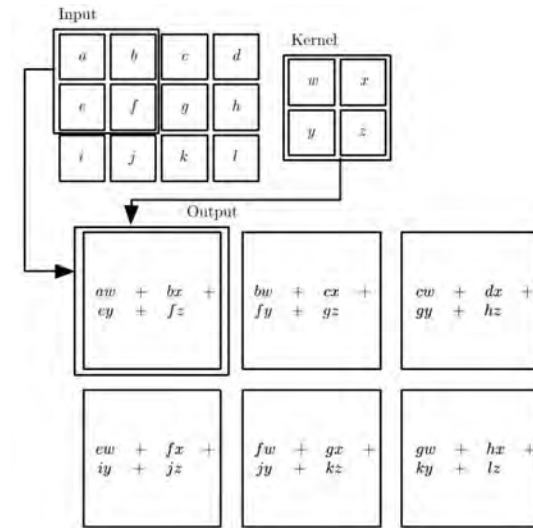


Figura 2.6: Un esempio di cross-correlazione ovvero convoluzione discreta 2-D senza il ribaltamento del kernel. Immagine riprodotta da [14], pag.334

Se ne forniscono due differenti implementazioni: una versione che impiega direttamente la definizione (sez. 6.1) ed un'altra che sfrutta operazioni matriciali che simulano lo scorrimento del kernel(sez. 6.1).

In effetti, l'operazione di cross-correlazione è identica a quella di convoluzione, a meno del flipping del kernel. Il risultato della convoluzione pura può essere quindi ottenuto applicando una crosscorrelazione con medesimo kernel e ribaltando, alla fine, il risultato ottenuto.

Nel caso in cui l'input sia un tensore, ad esempio un'immagine a colori $\mathbf{I} \in \mathbb{R}^{m \times n \times o}$,

si richiede che il kernel abbia lo stesso numero di livelli, ad esempio $\mathbf{K} \in \mathbb{R}^{l,w,o}$. Si esegue l'operazione di crosscorrelazione bidimensionale appena descritta tra ciascun livello dell'input e ciascun livello del kernel, quindi si sommano gli elementi di tutti i livelli con stessa coordinata (i, j) :

$$F_{map}(i, j) = (I \circledast K)(i, j, k) = \sum_{k=1}^o \sum_{z=1}^l \sum_{r=1}^w I(i+z-1, j+r-1, k) K(z, r, k)$$

L'implementazione è riportata alla sez. 6.1.

Si evidenzia che l'operazione effettivamente implementata per la *feature extraction* all'interno del classificatore impiegato nel presente lavoro di tesi (par. 3.4) è quella appena esposta, presentando però una leggera differenza rispetto alla definizione presentate.

Al filtro \mathbf{K} è associato un ulteriore termine detto *bias*, che viene addizionato elemento-per-elemento alla *feature map* risultante. Inoltre, a partire da un tensore iniziale \mathbf{I} , anziché applicare un unico filtro \mathbf{K} , si applica contemporaneamente ma in maniera indipendente uno stack di filtri della medesima dimensione, ottenendo un nuovo tensore come *feature map*. Ogni filtro avrà, quindi, il suo termine di *bias* associato.

Si fornisce, quindi, anche l'implementazione di quest'ultima versione dell'operazione alla sez. 6.1 ed un esempio numerico in fig. 2.7.

Si evidenzia, inoltre, che le operazioni presentate possono essere ulteriormente modificate attraverso la definizione di alcuni parametri che servono a regolare lo scorrimento del kernel sull'input (*stride*), la sua modalità di applicazione (*dilation factor*) ed il comportamento ai bordi (*padding*). Una descrizione più dettagliata di tali parametri è riportata nel par. 3.4. Essi influiscono sulla dimensione della *feature map* risultante. Per completezza, si osserva che nelle implementazioni presentate si è assunto: *stride* e *dilation factor* pari a 1, padding nullo (che rispecchiano uno scorrimento intuitivo del kernel, applicando quella che talvolta è chiamata "convoluzione valida").

Si nota anche che, a differenza delle implementazioni riportate, le librerie spesso impiegano algoritmi di convoluzione basati sulla *Fast Fourier Transform* (FFT) o, più semplicemente, moltiplicazione tra matrici appositamente costruite al fine di velocizzare la computazione.

Come inciso finale, si osserva che l'operazione di convoluzione è molto utilizzata in ambito ingegneristico, poichè particolarmente importante per la caratterizzazione dei sistemi. Nella letteratura dell'*image processing*, in particolare, esiste una quantità notevole di kernel noti utilizzati per effetti ben precisi (*smoothing*, estrazione del contorno, ecc.).

Dal punto di vista matematico, ha anche suscitato l'interesse di Alan Turing, il quale aveva l'abitudine di illustrare il concetto di convoluzione come espansione di una funzione nella maniera definita da un'altra funzione, ricorrendo ad un particolare esempio biologico: la crescita concentrica del micelio di certi funghi, detto anche "il cerchio delle streghe" (*fairly ring*) (Hodges, [12]).

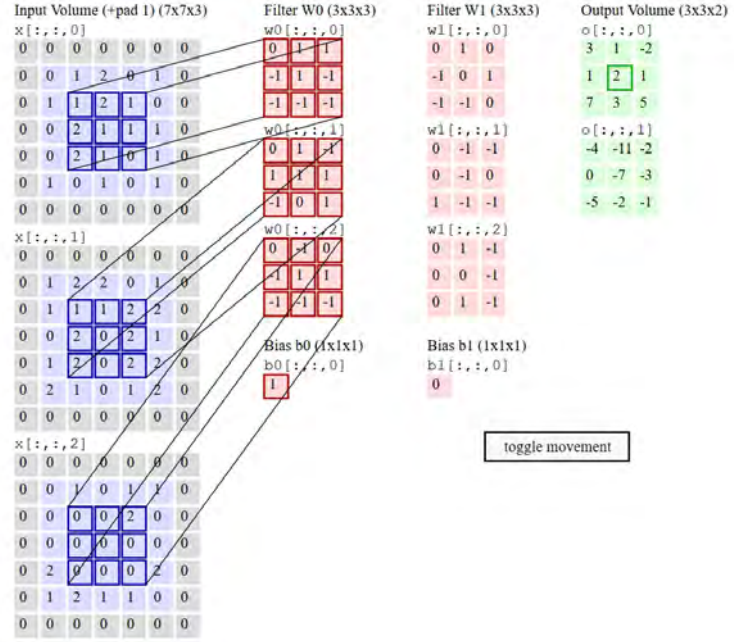


Figura 2.7: Esempio numerico di convoluzione discreta su 3 dimensioni con uso di padding pari a 1 e stride pari a 2. Il risultato evidenziato con un riquadro verde è ottenuto sommando il bias alla somma delle tre convoluzioni, una per ogni livello. Tratto dall'esempio interattivo di Andrej Karpathy, disponibile alla pagina <http://cs231n.github.io/convolutional-networks/>

2.3 Problemi di *Computer Vision*

In questa sezione si riportano alcuni problemi caratteristici della *computer vision*, affrontati nel corso del lavoro e finalizzati ad una comprensione di alto livello del contenuto delle immagini (e dei video) da parte del computer. Il nome italiano della disciplina, visione artificiale, richiama in questo senso l'obiettivo di rendere artificiali i compiti svolti dal sistema visivo umano.

2.3.1 Segmentazione

Il problema della segmentazione consiste nel partizionamento di un'immagine in diversi gruppi di pixel, detti segmenti, accomunati da un certo significato di alto livello. Nella pratica, questo significa, eventualmente, assegnare a ciascun pixel dell'immagine un'etichetta, sulla base di alcune caratteristiche o proprietà, come il colore, la luminosità o la presenza di un motivo.

Spesso, la segmentazione è utilizzata per distinguere classi di oggetti all'interno di un'immagine: tutti i pixel associati alla rappresentazione di un oggetto di interesse costituiscono un segmento. È il caso applicativo del presente lavoro di tesi: si utilizza il

colore dei pixel per etichettare automaticamente le regioni che possono eventualmente contenere una pinna dorsale di delfino (par. 4.1). Poichè l'obiettivo è quello di distinguere una sola classe di oggetti, l'etichettatura utilizzata è di tipo logico: 1 in corrispondenza dei pixel di interesse e 0 altrove. Il risultato è detto immagine binarizzata. Si consideri, a titolo esemplificativo, la figura mostrata nel corso della discussione successiva del metodo: 4.7.

L'automatizzazione della segmentazione può essere ottenuta, nel caso si vogliano distinguere due classi di oggetti, attraverso il metodo Otsu, descritto nel paragrafo 2.3.1 e utilizzato all'interno degli esperimenti (par. 4.1.1). È stato anche introdotto un metodo innovativo di segmentazione, specifico per il dominio del problema in esame (par. 4.1.2).

Si evidenzia, inoltre, che la segmentazione è stata utilizzata come operazione di *pre-processing* finalizzata alla risoluzione del problema originario di *object detection*, leggermente differente e descritto al par. 2.3.2.

Soglia di Otsu

Il metodo Otsu per la sogliatura automatica (Otsu, [13]) è finalizzato al calcolo di un valore di soglia per l'intensità dei pixel utile a segmentare l'immagine in due classi, generalmente *background* e *foreground*. Nel seguito si descrive il procedimento utilizzato per il calcolo della soglia.

Si consideri un'immagine avente dimensione $d = M \times N$ con L livelli distinti di intensità per i pixel e sia n_i il numero di pixel di intensità i , con $i = 0, \dots, L - 1$. Il suo istogramma normalizzato ha, quindi, componenti $p_i = n_i/d$.

Si supponga di selezionare una soglia k , con $0 < k < L - 1$, e di dividere in base ad essa l'immagine in due classi:

- classe C_1 : contenente tutti i pixel con intensità compresa nell'intervallo $[0, k]$;
- classe C_2 (contenente tutti i pixel con intensità compresa nell'intervallo $[k+1, L-1]$).

Con tale soglia, la probabilità che un pixel sia assegnato alla classe C_1 è:

$$P_1(k) = \sum_{i=0}^k p_i = \sum_{i=0}^k \frac{n_i}{d}$$

E, dualmente, la probabilità che un pixel sia assegnato alla classe C_2 è:

$$P_2(k) = 1 - P_1(k) = \sum_{i=k+1}^{L-1} p_i = \sum_{i=k+1}^{L-1} \frac{n_i}{d}$$

Il valore medio di intensità dei pixel appartenenti alla classe C_1 è dato da:

$$m_1(k) = \sum_{i=0}^k i P(i|C_1) = \sum_{i=0}^k i P \frac{P(C_1|i)P(i)}{P(C_1)} = \frac{\sum_{i=0}^k i p_i}{P_1(k)}$$

In maniera simile, si ricava per la seconda classe

$$m_2(k) = \frac{\sum_{i=k+1}^{L-1} i p_i}{P_2(k)}$$

Si definisce la media cumulativa fino al livello k :

$$m(k) = \sum_{i=0}^k p_i$$

e, analogamente, la media delle intensità dell'intera immagine:

$$m_G(k) = \sum_{i=0}^{L-1} p_i$$

Per valutare la bontà della soglia k si definiscono:

- la varianza globale dell'immagine σ_G^2 :

$$\sigma_G^2 = \sum_{i=0}^{L-1} (i - m_G)^2 p_i$$

- la varianza interclasse σ_B^2 :

$$\sigma_B^2 = P_1(m_1 - m_g)^2 + P_2(m_2 - m_g)^2 = P_1 P_2 (m_1 - m_2)^2 = \frac{(m_G P_1)^2}{P_1(1 - P_1)}$$

La soglia k^* è calcolata come

$$\sigma_B^2(k^*) = \max_{0 \leq k \leq L-1} \sigma_B^2(k)$$

cioè è tale da massimizzare la varianza interclasse.

Il metodo è stato originariamente concepito per le immagini con toni di grigio. In seguito sono state introdotte diverse varianti, tra cui quella multilivello, utilizzata nel presente lavoro, attraverso la funzione `multithresh`, come descritto al par. 4.1.1.

Regioni connesse

In seguito alla segmentazione, solitamente si è interessati ad analizzare le proprietà dei segmenti. Una volta individuati, questi vengono raggruppati in regioni di pixel adiacenti con la medesima etichetta detti regioni (o componenti) connesse. Nel corso di questo lavoro si utilizza anche il termine *blob* come sinonimo di regione connessa, sebbene la sua definizione sia più genericamente quella di una regione con proprietà costanti all'interno dell'immagine.

Il raggruppamento dei pixel in regioni connesse, così come lo studio delle loro proprietà,

è una funzionalità generalmente implementata, con diversi metodi, all'interno di una libreria di *computer vision*.

Si descrivono nel seguito le funzioni offerte in Matlab ed utilizzate nel corso del presente lavoro.

Le statistiche di interesse sono:

- **Area**: numero totale di pixel della regione
- **BoundingBox**: il più piccolo rettangolo contenente la regione, specificato attraverso un vettore contenente, in ordine, le coordinate del punto in alto a sinistra e le dimensioni larghezza e altezza.
- **Centroid** : coordinate del centro di massa della regione (media delle coordinate x e y di tutti i punti)
- **Extent**: rapporto tra Area e numero di pixel del BoundingBox
- **Orientation**: angolo di orientazione dell'asse maggiore dell'ellisse che contiene la regione, misurato rispetto all'asse x (fig. 2.8)(a).

A partire da un'immagine binarizzata BW è possibile ottenere le componenti connesse e loro proprietà in differenti modi:

- Attraverso la funzione `bwconncomp`, ottenendo uno `struct` con le informazioni delle regioni connesse

```
cc = bwconncomp(BW);
```

grazie a cui è possibile ottenere le statistiche desiderate, specificandole tra apici come argomento della funzione `regionprops`:

```
stats = regionprops(cc, 'Area','Orientation');
```

- Istanziando un oggetto di tipo `BlobAnalysis`

```
Hblob = vision.BlobAnalysis;
```

attraverso il quale è possibile ottenere di default le proprietà **Area**, **Centroid** e **BoundingBox** come:

```
[area,centroid,bbox] = Hblob(BW);
```

Si evidenzia che le funzioni presentate utilizzano di default un algoritmo di *clustering* per la ricerca delle regioni connesse, tale per cui ciascuna regione è identificata dai pixel connessi secondo la *8-connectivity*: per ciascun pixel con valore logico 1 si verifica il valore degli 8 pixel che lo racchiudono, come mostrato in fig. 2.8(b).

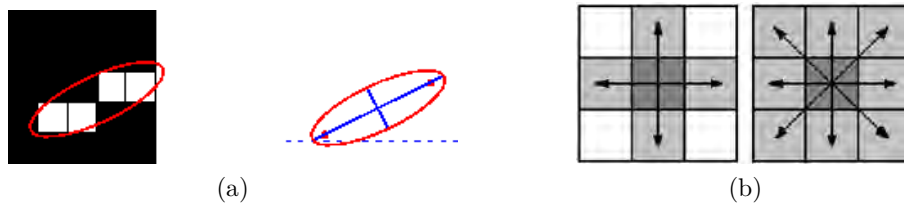


Figura 2.8: Proprietà delle regioni connesse: (a) visualizzazione dell'ellisse a partire da cui è calcolato il parametro "orientazione"; (b) ricerca delle regioni secondo la 8-connectivity

Queste statistiche sono utilizzate nel presente lavoro per filtrare le componenti delle immagini (par. 4.2) e per ritagliare adattivamente le pinne dorsali dei delfini a partire dall'immagine originale (par. 4.3). Il calcolo delle regioni connesse in seguito ad un'operazione di segmentazione automatica può, infatti, provocare l'individuazione di regioni non effettivamente interessanti. L'utilizzo delle due differenti modalità di gestione delle regioni connesse descritto pocanzi è utile proprio a facilitare la creazione dei filtri basati esclusivamente sulle proprietà delle regioni.

Nell'ambito del filtraggio delle regioni connesse, si utilizzano, oltre ai parametri citati, due tecniche legate ad ulteriori proprietà delle regioni, qui di seguito descritte.

Riempimento degli *holes* In seguito al calcolo delle regioni connesse, possono essere identificate all'interno dell'immagine alcune aree dette *holes*: esse sono costituite dai pixel del background (cioè con valore logico 0) che non possono essere raggiunti dalle parti di sfondo ai bordi dell'immagine (quindi letteralmente dei buchi all'interno delle regioni binarie).

Gli *holes* dell'immagine BW possono essere riempiti cambiando il loro valore logico in 1 attraverso la funzione `imfill`:

```
BW = imfill(BW,'holes');
```

Filtro mediano Talvolta, l'immagine binarizzata in seguito ad una segmentazione automatica presenta numerose regioni connesse di dimensione estremamente ridotta, fonte di disturbo per la localizzazione dei segmenti, causando alla vista un effetto denominato "sale e pepe".

Per rimuovere questo tipo di rumore si utilizza un filtro mediano: l'intensità di ciascun pixel viene sostituita dal valore mediano calcolato in un intorno limitato. In Matlab è disponibile allo scopo la funzione `med2filt`:

```
BW = med2filt(BW);
```

2.3.2 Object detection

Il problema di *object detection* consiste nel riconoscere e localizzare automaticamente una o più classi di oggetti, stabilite a priori, all'interno di un'immagine.

La classe di oggetti di interesse all'interno del presente lavoro è una sola: si vogliono identificare, e conseguentemente ritagliare, le porzioni dell'immagine contenenti pinne dorsali di delfini.

Il compito si rivela, in generale, particolarmente complesso a causa della difficoltà di estrarre conoscenza di alto livello che sia invariante ai seguenti fattori (banali per un essere umano):

- variazione del punto di vista: un oggetto può essere orientato in molti modi
- variazione di scala: le classi di oggetti possono presentare istanze con dimensioni differenti sia all'interno dell'immagine sia nel mondo reale
- deformazione: molti oggetti di interesse non sono corpi rigidi e possono essere deformati in modi estremi
- occlusione: a volte solo una piccola porzione di oggetto può essere visibile
- condizioni di illuminazione: l'intensità dei pixel è estremamente variabile a causa dell'illuminazione
- variazione intra-classe: le classi di oggetti possono avere istanze molto diverse tra loro.

In letteratura sono stati proposti numerosi metodi per la risoluzione efficiente del problema di *object detection*, a partire dall'algoritmo noto come Viola-Jones, proposto nel 2001. Tra i metodi allo stato dell'arte si distinguono due principali approcci:

- estrazione di *feature* di interesse (forma di conoscenza di livello intermedio) mediante algoritmi come *Scale-invariant feature transform* (SIFT) e *Histogram of oriented gradients* (HOG) seguita da una fase di classificazione con tecniche di *machine learning*. Solitamente questa combinazione è effettuata mediante un meccanismo di tipo *sliding window*: le due fasi sono effettuate in diverse posizioni di ciascuna immagine, a differenti scale, simulando lo scorrimento di una finestra sull'immagine di input;
- utilizzo di tecniche di *deep learning* per la risoluzione del problema *end-to-end*: le fasi di estrazione di *feature* e di classificazione sono condensate all'interno di metodi basati sulla proposta di regioni di interesse (R-CNN, Fast R-CNN, Faster R-CNN) o sul affinamento di regioni predefinite (SSD ovvero *Single Shot Multi Box Detector* e YOLO ovvero *You Look Only Once*)

Gli approcci basati sul *deep learning* hanno permesso di compiere una svolta decisiva per la risoluzione del problema. I principali concetti e punti di forza del *deep learning*

sono presentati nel capitolo successivo (3).

Nel presente lavoro, si è utilizzato un metodo risolutivo non ascrivibile a nessuna delle classificazioni precedenti. Il dominio ristretto del problema in esame ha consentito di utilizzare come approccio principale una forma di conoscenza di alto livello direttamente disponibile nella rappresentazione delle immagini: il colore (par. 4.1). I risultati ottenuti sono stati quindi raffinati attraverso una classificazione finale con tecniche di *deep learning* (par. 4.4).

Si vuole, infine, riportare, a titolo di esempio, un caso applicativo del problema di *object detection* ritenuto di estrema importanza oggi: i veicoli a guida autonoma. Compagnie come Waymo e MobilEye utilizzano complessi sistemi di visione con questo tipo di tecnologie per identificare qualsiasi oggetto di interesse sulla carreggiata: segnali stradali, pedoni, altri veicoli, ecc.¹

¹<https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503>

Capitolo 3

Machine Learning

All'interno del presente capitolo si trattano brevemente le metodologie di *machine learning* utilizzate all'interno del lavoro di tesi.

Un algoritmo di machine learning è un algoritmo che è, in generale, è capace di imparare dai dati. Mitchell fornisce la seguente definizione:

Si dice che un programma per computer impara dall'esperienza E rispetto ad una qualche classe di compiti T ed una misura di performance P se le sue prestazioni nei compiti in T , misurate attraverso P , migliorano con l'esperienza E .

Quasi tutti gli algoritmi di machine learning sono costruiti combinando almeno quattro building blocks fondamentali: un dataset, una funzione costo, un modello ed un metodo di ottimizzazione.

Il deep learning è un tipo specifico di machine learning, che negli ultimi anni ha dato una svolta decisiva alla più vasta branca dell'intelligenza artificiale (fig. 3.1). Dalla pubblicazione dello studio intitolato *Deep Learning* (Le Cun *et. al.*, [18]) nel 2015, l'importanza ed il successo di questa nuova tecnologia ha permesso agli autori di ricevere il premio Turing per l'anno 2018¹.

Le performance di un algoritmo di machine learning sono fortemente influenzate dalla rappresentazione dei dati a disposizione. Molti compiti di intelligenza artificiale possono essere risolti designando il giusto insieme di *feature* da estrarre, per poi fornire tali feature ad un semplice algoritmo di *machine learning*. Tuttavia, per alcuni compiti è difficile sapere quali *feature* devono essere estratte. Esistono problemi, inoltre, per cui le feature rappresentate in un generico input x non sono individualmente molto informative ai fini del task in esame, nel senso che è difficile pensare che esista un legame lineare tra le feature a disposizione e l'output del problema. Ad esempio, è impensabile descrivere la presenza di un oggetto all'interno di un'immagine attraverso

¹[nytimes.com/2019/03/27/technology/turing-award-ai.html](https://www.nytimes.com/2019/03/27/technology/turing-award-ai.html)

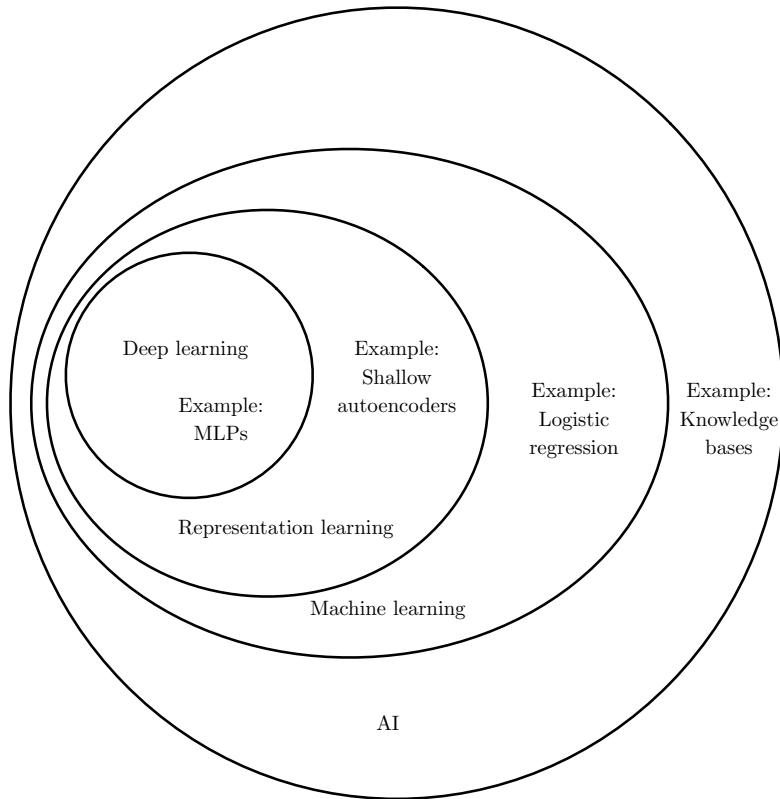


Figura 3.1: Un diagramma di Venn che mostra come il deep learning sia un tipo di representation learning, che a sua volta è un tipo di machine learning. L'intelligenza artificiale è una branca molto più ampia, rispetto a cui il deep learning è uno degli approcci più in voga e di successo al momento. Immagine riprodotta dal repository di [14], disponibile all'indirizzo https://github.com/goodfeli/dlbook_notation

un legame lineare con i singoli pixel. È piuttosto la combinazione di pixel che può fornire indicazioni utili in un problema di *object detection* (par. 2.3.2).

Una soluzione a questo problema è quello di usare il machine learning non solo per trovare una relazione tra dati e output, ma anche per la rappresentazione stessa dei dati. Questo approccio è detto **representation learning**. L'obiettivo dell'individuazione di *feature* è in generale (sia esso fatto a mano sia attraverso un algoritmo di ML) quello di disaccoppiare dai dati i cosiddetti fattori di variazione a cui non si è interessati. Ad esempio, in una foto di un delfino a pelo d'acqua, i fattori di variazione possono essere la posizione del delfino rispetto all'acqua, l'angolo di scatto, la luminosità fornita dalla luce del sole, i riflessi dell'acqua. Chiaramente il design delle feature può essere un compito molto difficile e molti fattori di variazione possono essere identificati solo attraverso una comprensione quasi umana dei dati (es. l'accento di uno speaker all'interno di un problema di speech recognition).

Il deep learning risolve questo problema centrale introducendo rappresentazioni dei dati che sono espresse in termini di altre, più semplici rappresentazioni. Il computer è quindi in grado di interpretare concetti ad un alto livello di complessità semplicemente componendo concetti di complessità inferiore. Il concetto di una pinna (in generale di un oggetto all'interno di una foto) può essere interpretato in termini di concetti più semplici, componendo angoli e forme, che a loro volta sono definiti in termini di bordi.

Nel paragrafo successivo si descrive in maggior dettaglio il problema del *supervised learning*.

3.1 Supervised Learning

Molti problemi pratici, come ad esempio la classificazione, possono essere formulati ricorrendo al concetto di funzione matematica

$$f : X \rightarrow Y$$

che associ ad ogni elemento nello spazio degli input X uno ed un solo elemento dello spazio degli output Y .

Si precisa che $x \in X$ e $y \in Y$ possono essere dati monodimensionali oppure multidimensionali, a seconda del tipo di problema affrontato. In particolare, è possibile distinguere due tipi di problemi principalmente affrontati con il paradigma dell'apprendimento supervisionato, sulla base del solo spazio di output Y :

- problema di regressione: $Y \subset \mathbb{R}$
- problema di classificazione: $|Y| = k$, con k numero delle classi

Un problema di *object detection* può essere formulato come un problema di classificazione, in cui X sia uno spazio di immagini e Y sia l'insieme delle classi di oggetti che possono essere individuati al suo interno. Nella presente tesi, generalmente x è un'immagine nel formato RGB e y è un dato binario dell'insieme $Y = \{\text{"Pinna"}, \text{"No Pinna"}\}$.

Nella realtà, spesso non si è in grado di specificare una funzione f siffatta che permetta al computer di risolvere il problema in maniera esatta. Nel caso in esame, non è chiaro come poter scrivere un algoritmo che consenta di individuare con esattezza la presenza di una pinna all'interno di un'immagine (al limite si può ricorrere all'individuazione di *feature* mediante tecniche di computer vision che facilitino il problema, come esposto nel capitolo precedente).

Il paradigma dell'apprendimento supervisionato (*supervised learning*) offre un approccio alternativo per la specifica di f . L'utilizzo del lemma *apprendimento* (in inglese *learning*) fa riferimento al fatto che la funzione f non è codificata manualmente. La

macchina è invece programmata per risolvere un problema di ottimizzazione (piuttosto complesso) formulato in modo da ottenere una funzione h^* che fornisca una stima quanto più precisa della funzione f , cioè

$$h^* \approx f$$

a partire da:

- una collezione di dati rappresentativa del problema, detta *training set*, composta da un certo numero m di esempi rispetto ai quali si conosce la soluzione al problema. Il *training set* è cioè formalmente composto da generici elementi x_i dello spazio X per i quali è noto il valore della funzione cercata $y_i = f(x_i)$, nello spazio Y , per $i = 1, \dots, m$.

$$\text{Training set: } \mathcal{S}_{train} = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}.$$

Il valore y_i è inteso genericamente come un'etichetta (*label*) o valore vero in corrispondenza di un dato x_i . Tale informazione è fornita da un supervisore (umano) esterno, il quale si comporta pertanto da *insegnante* rispetto al programma che deve apprendere. È in questo senso che si parla di apprendimento *supervisionato*.

- uno spazio di funzioni candidate \mathcal{H} in cui cercare $h^* \approx f$. La lettera utilizzata \mathcal{H} rappresenta l'iniziale della parola *hypothesis*, a significare che la scelta dello spazio di funzioni è indicativo di un'ipotesi fatta a priori sulla forma della funzione cercata f . Comunemente, \mathcal{H} è uno spazio di funzioni parametrico, con Θ che indica lo spazio dei parametri. Ciascuna funzione candidata $h \in \mathcal{H}$ è quindi del tipo

$$h : (X; \Theta) \rightarrow Y.$$

- una funzione

$$\begin{aligned} \mathcal{L} : Y \times Y &\rightarrow [0, \infty) \\ (h(x), y) &\mapsto \mathcal{L}(h(x), y) = \mathcal{L}(\hat{y}, y) \end{aligned}$$

detta *loss function* (*funzione errore* o *perdita*) che fornisce una misura dell'errore in cui si incorre nell'approssimare ciascuna etichetta y con la funzione $h(x)$, con $h \in \mathcal{H}$. Generalmente, il valore $h(x)$ è inteso come *predizione* o *ipotesi* rispetto al dato x ed è indicato con il simbolo \hat{y} in modo che sia distinto dal valore vero y fornito nel *training set*.

La procedura risolutiva utilizzata per determinare h^* è considerata a tutti gli effetti un processo di apprendimento da parte della macchina, con il quale si desidera che la macchina impari a risolvere il problema di partenza, a partire da un insieme di dati di esempio.

3.1.1 Statistical Learning Theory

La validità dell'approccio propugnato dall'apprendimento supervisionato va ricercata nel campo della statistica. In particolare, si menziona l'esistenza di una branca denominata teoria statistica dell'apprendimento (*statistical learning theory*), in grado di fornire gli aspetti teorici necessari e gli strumenti adatti per la scelta degli elementi $\{\mathcal{S}_{train}, \mathcal{H}, h^*, \mathcal{L}\}$ che consentano efficientemente di risolvere un problema reale esprimibile come relazione tra dati.

Anzitutto, al fine di estendere il dominio del problema di partenza ad un generico insieme di dati nella realtà, si deve modellare l'incertezza presente sia nei dati che nel problema stesso. Con riferimento all'approccio frequentista, si suppone quindi che esista una distribuzione di probabilità D con densità $p(x, y)$, fissata ma sconosciuta, in accordo alla quale i dati siano identicamente e indipendentemente distribuiti rispetto allo spazio $X \times Y$ in cui è definito il problema (detto spazio dei dati).

Si assume inoltre che $p(x, y)$ possa essere fattorizzata come

$$p(x, y) = p_X(x)p(y|x)$$

in cui:

- la distribuzione di probabilità condizionata $p(y|x)$ descrive il problema di partenza f come una relazione non deterministica tra dati di input $x \in X$ e dati di output $y \in Y$, modellando l'incertezza legata al problema stesso.
- la distribuzione di probabilità marginale $p_X(x)$ modella l'incertezza con cui sono generati i dati di input x dallo spazio X

La situazione è indicata intuitivamente nella figura 3.2.

A queste considerazioni ci si riferisce per brevità come *ipotesi i.i.d.* o equivalentemente con la scrittura

$$(x, y) \sim D$$

Con ciò è possibile asserire che la funzione cercata $h^* \approx f$ costituisce effettivamente la migliore approssimazione di f se

$$h^* = \arg \min_{h \in \mathcal{H}} \mathcal{E}(h) \quad (3.1)$$

in cui $\mathcal{E}(h)$ è detto *expected loss* o *expected risk* (cioè *perdita attesa* o *rischio atteso*) e rappresenta il valore atteso della *loss function* calcolata rispetto ad un generico "universo" di dati $(x, y) \sim D$:

$$\begin{aligned} \mathcal{E}(h) &\triangleq \mathbb{E}_{(x,y) \sim D} \left\{ \mathcal{L}(h(x), y) \right\} \\ &= \int_{X \times Y} dx dy p(x, y) \mathcal{L}(h(x), y) \end{aligned} \quad (3.2)$$

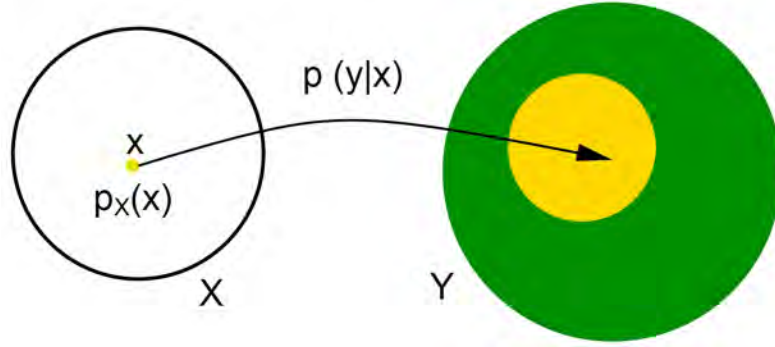


Figura 3.2: Illustrazione intuitiva di un problema espresso come una relazione non deterministica tra due insiemi di dati. Per ogni input x , ottenuto secondo la distribuzione $p_X(x)$ nello spazio X , esiste una distribuzione di possibili output $p(y|x)$ nello spazio Y , in cui è contenuta la soluzione del problema, mostrata in giallo. L'area verde costituisce lo spazio rimanente degli output Y . Immagine tratta da (Rosasco, [20])

dove $\mathbb{E}\{\cdot\}$ denota l'operatore di aspettazione.²

Sostituendo 3.2 in 3.1 si ottiene

$$h^* = \arg \min_{h \in \mathcal{H}} \mathbb{E}_{(x,y) \sim D} \left\{ \mathcal{L}(h(x), y) \right\}. \quad (3.3)$$

Il motivo per cui si rende in tal modo h^* la migliore approssimazione di f è immediato: la minimizzazione della *loss function* riguarda idealmente, secondo l'approccio frequentista, la totalità dei dati $(x, y) \sim D$ a cui la relazione f può essere applicata.

Mutuando dalla statistica il concetto di stima di parametri, è possibile definire in maniera più rigorosa in che termini h^* costituisce un'approssimazione di f rispetto ad un insieme di dati $(x, y) \sim D$.

Dato un campione di m dati indipendenti e identicamente distribuiti rispetto ad una distribuzione di probabilità X

$$(x_1, x_2, \dots, x_m) \sim X$$

²Si riporta di seguito la definizione di $\mathbb{E}\{\cdot\}$ rispetto ad una generica funzione g di variabile aleatoria $Z \in \text{Dom}(Z)$, con funzione densità di probabilità $p_Z(z)$:

$$\mathbb{E}_{z \sim p_Z} [g(Z)] \triangleq \int_{\text{Dom}(Z)} dz g(z) p_Z(z).$$

si definisce *stimatore puntuale* (o *statistica*) $\hat{\theta}_m$ una qualsiasi funzione dei dati che abbia l'obiettivo di fornire la predizione di una quantità di interesse arbitraria:

$$\hat{\theta}_m = g(x_1, x_2, \dots, x_m). \quad (3.4)$$

Sulla base della definizione appena riportata, si perviene alla seguente più rigorosa formulazione.

Un algoritmo di apprendimento (*learning algorithm*) è una procedura che calcola, a partire da un *training set* \mathcal{S}_{train} di dati $(x, y) \sim D$, uno stimatore puntuale h^* tale che

$$\mathcal{E}(h^*) \approx \mathcal{E}(f)$$

Il problema di ottimizzazione appena formulato risulta però intrattabile poichè si è supposto (ragionevolmente) di non conoscere la distribuzione $p(x, y)$ che descrive la distribuzione dei dati rispetto al problema.

L'ipotesi *i.i.d.* permette però di utilizzare l'approssimazione

$$\mathcal{E}(h) \approx \frac{1}{m} \sum_{i=1}^m \mathcal{L}(h(x_i), y_i) \quad (3.5)$$

trasformando il problema in

$$h^* = \arg \min_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \mathcal{L}(h(x_i), y_i). \quad (3.6)$$

La perdita attesa rispetto all'universo di dati è approssimata con una media semplice della *loss function* calcolata sui dati disponibili all'interno del *training set*. In tal modo, il problema diventa trattabile ma l'approssimazione utilizzata penalizza la possibilità di raggiungere l'obiettivo generale di un (buon) algoritmo di *machine learning*, insito nella formulazione del problema originale 3.3: la capacità di **generalizzazione**.

Idealmente, sebbene si insegni alla macchina a risolvere un dato problema a partire dalla collezione limitata di dati che costituisce il *training set*, si desidera che la macchina sia in grado di risolvere il problema rispetto a qualsiasi dato nuovo che viene ad essa sottoposto (purchè il problema sia ad esso applicabile).

L'obiettivo della generalizzazione può essere riassunto con la scrittura

$$h^*(x^{\text{new}}) \approx y^{\text{new}}$$

intendendo che la soluzione cercata h^* deve essere in grado di stimare f rispetto ad un qualsiasi dato $(x^{\text{new}}, y^{\text{new}})$ non contenuto all'interno del *training set* (ma analogo ai dati inclusi in esso).

L'importanza del concetto di generalizzazione giustifica la necessità di introdurre un'ulteriore collezione di dati, detta *test set*, contenente esempi che non siano utilizzati in fase di addestramento, ma siano impiegati per la sola misura delle performance con cui

l'algoritmo sia in grado di generalizzare la risoluzione del problema rispetto a dati mai visti.

L'obiettivo della generalizzazione inoltre rende necessario modificare il criterio di scelta di h^* , enunciato dal problema di ottimizzazione 3.6. Qualsiasi modifica che abbia il solo scopo di aumentare la capacità di generalizzazione dell'algoritmo per la risoluzione del problema prende il nome di **regolarizzazione**. Generalmente, viene aggiunto un termine scalare $R(h)$, come segue:

$$h^* = \arg \min_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \mathcal{L}(h(x_i), y_i) + R(h). \quad (3.7)$$

$R(h)$ è in qualche modo una misura della complessità di una funzione ed è impiegata per fornire in fase di apprendimento la preferenza per un certo tipo di funzioni presenti nello spazio \mathcal{H} . Tale aggiunta può essere parzialmente giustificata ricorrendo al principio del *rasoio di Occam*, che può essere enunciato come:

«A parità di fattori la spiegazione più semplice è da preferire»

Più in generale, i metodi di regolarizzazione sono utilizzati per alleviare l'approssimazione del problema di ottimizzazione originario 3.3, seguendo opportuni criteri di preferenza per la ricerca della soluzione h^* .

Riassumendo, il paradigma dell'apprendimento supervisionato consente di risolvere un problema esprimibile come una relazione $f : X \rightarrow Y$ tra dati, richiedendo di identificare una funzione

$$h^* \approx f$$

a partire da:

- un *training set* di esempi $\mathcal{S}_{train} = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ con cui addestrare l'algoritmo
- uno spazio di funzioni \mathcal{H} parametrizzato in Θ in cui cercare $h^* : (X; \Theta) \rightarrow Y$
- una *loss function* scalare $\mathcal{L}(\hat{y}, y)$ che misuri l'errore che si commette nell'approssimare il valore vero y con la predizione $\hat{y} = h(x)$, $h \in \mathcal{H}$
- un termine scalare di regolarizzazione $R(h)$ che codifichi la preferenza verso un certo tipo di funzioni $h \in \mathcal{H}$

In particolare, ipotizzando i dati *i.i.d.* secondo la distribuzione D , h^* è soluzione del problema di ottimizzazione

$$h^* = \arg \min_{h \in \mathcal{H}} \mathbb{E}_{(x,y) \sim D} \left\{ \mathcal{L}(h(x), y) \right\} \quad (3.8)$$

reso trattabile ricorrendo all'approssimazione

$$h^* = \arg \min_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \mathcal{L}(h(x_i), y_i) + R(h). \quad (3.9)$$

Giacchè lo spazio delle soluzioni \mathcal{H} è parametrizzato in Θ , risolvere il problema 3.9 richiede in pratica di *apprendere* un certo numero di parametri θ con cui restano definite le funzioni candidate $h \in \mathcal{H}$, secondo il problema equivalente

$$\theta = \arg \min_{\theta \in \Theta} \frac{1}{m} \sum_{i=1}^m \mathcal{L}(h(x_i; \theta), y_i) + R(\theta) \quad (3.10)$$

che definisce in maniera concisa un problema di *supervised learning*.

L'identificazione degli elementi che definiscono il problema, nonchè la scelta della soluzione, deve essere condizionata dalla capacità di generalizzazione dell'algoritmo

$$h^*(x^{\text{new}}) \approx y^{\text{new}}$$

misurabile, una volta che è stata appresa una funzione h^* , a partire da dati $(x^{\text{new}}, y^{\text{new}})$ che non siano contenuti nel *training set*, ma siano eventualmente parte di un *test set*.

Comunemente, la forma scelta per le funzioni in \mathcal{H} è piuttosto complessa, motivo per cui si rende necessario l'apprendimento di un numero molto elevato di parametri θ . Il problema di ottimizzazione 3.10 risulta in tali casi chiaramente *non convesso*, pertanto può essere affrontato solo ricorrendo a procedure iterative con opportuni criteri di stop.

Nel caso della classificazione binaria effettuata nel presente lavoro di tesi:

- il *training set* è costituito da un insieme di immagini (par. 4.4.1);
- lo spazio \mathcal{H} è definito dalle reti neurali convoluzionali utilizzate ed è parametrizzato da tutti i pesi ed i bias in esse presenti, appresi in fase di addestramento (par. 3.4);
- si utilizza una *cross-entropy loss function* (par. 3.2)
- il termine di regolarizzazione è detto norma L2: è proporzionale alla somma dei quadrati di tutti i pesi secondo un fattore detto *decay factor*
- la risoluzione del problema di ottimizzazione 3.10 è effettuata mediante la procedura iterativa *stochastic gradient descent with momentum* (par. 3.3.1)

3.1.2 Underfitting vs Overfitting

Un problema di apprendimento supervisionato ha come obiettivo teorico quello di progettare il miglior *stimatore puntuale* della funzione f (che si ricorda descrive un qualche

problema di partenza come relazione tra dati) a partire dagli esempi contenuti all'interno del *training set*.

Dal momento che la definizione di stimatore puntuale appare in realtà estremamente generica, è necessario ricorrere a specifiche proprietà degli stimatori puntuali al fine di poter valutare se uno stimatore è effettivamente migliore di altri per lo scopo considerato. Solitamente si utilizzano i seguenti criteri di riferimento.

- La capacità del modello è informalmente la sua abilità di rappresentare un'ampia varietà di funzioni ed è primariamente determinata dalla scelta dello spazio di funzioni \mathcal{H} in cui cercare la soluzione
- Il bias misura l'errore atteso rispetto al valore vero θ

$$\text{Bias}(\hat{\theta}_m) \triangleq \mathbb{E}\{\hat{\theta}_m\} - \theta$$

in cui l'aspettazione è calcolata rispetto ai dati (x_1, x_2, \dots, x_m) considerati come realizzazioni di una variabile aleatoria con distribuzione X .

- La varianza fornisce una misura di quanto vari il calcolo dello stimatore $\hat{\theta}_m$ rispetto a diversi insiemi di dati generati indipendentemente dalla medesima distribuzione X .

$$\text{Var}(\hat{\theta}_m) \triangleq \mathbb{E}\left\{\left(\hat{\theta}_m - \mathbb{E}\{\hat{\theta}_m\}\right)^2\right\}$$

- L'errore quadratico medio fornisce una misura della deviazione attesa complessiva tra lo stimatore $\hat{\theta}_m$ ed il valore vero θ

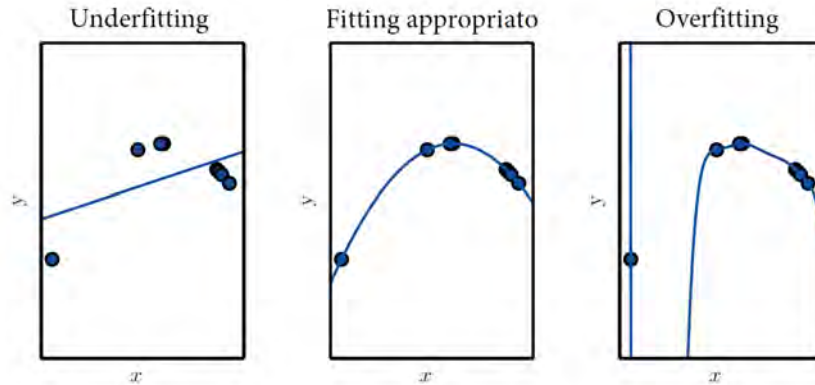
$$\begin{aligned}\text{MSE} &\triangleq \mathbb{E}\{(\hat{\theta}_m - \theta)^2\} \\ &= \left(\text{Bias}(\hat{\theta}_m)\right)^2 + \text{Var}(\hat{\theta}_m)\end{aligned}$$

Uno stimatore è detto corretto (*unbiased*) se

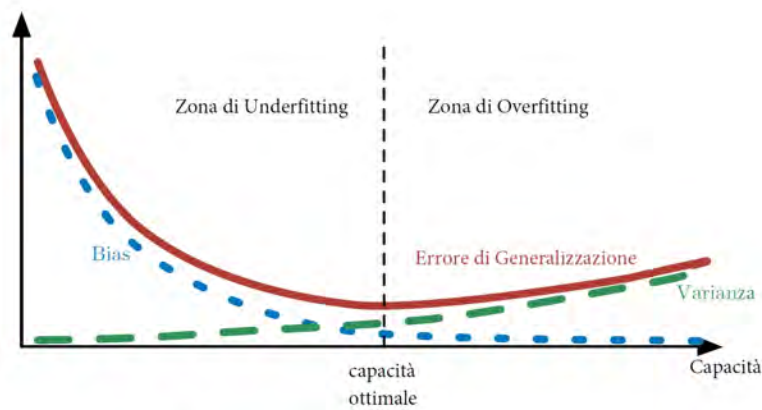
$$\text{Bias}(\hat{\theta}_m) = 0$$

Sebbene la proprietà di correttezza risulti chiaramente desiderabile, essa non è sufficiente per la costruzione del miglior stimatore puntuale. La sua progettazione deve tener conto al contempo sia del bias sia della varianza. Si può osservare, dalla sua definizione, come l'errore quadratico medio sia ad esempio una misura che rispetti tale esigenza.

La fig. 3.3 mostra il *trade-off* necessario per raggiungere la capacità ottimale del modello, a metà tra la zona di *underfitting* e la zona di *overfitting*. Generalmente, ciò non può essere valutato a priori ma dipende dai risultati dell'addestramento.



(a)



(b)

Figura 3.3: Quando la capacità del modello cresce (asse x), il bias decresce mentre la varianza subisce un incremento. Ciò causa la forma ad U della curva dell'errore di generalizzazione. La capacità ottimale si posiziona nel minimo della curva di generalizzazione e richiede un trade-off tra bias e varianza.

3.2 Classificazione

In un problema di classificazione l'insieme di output Y all'interno del *training set* è discreto e finito: ciascun $y \in Y$ indica una classe di appartenenza del dato x corrispondente.

Sulla base del numero di classi ovvero della cardinalità dell'insieme Y , si distinguono due tipi di classificazione: binaria ($|Y| = 2$) o multiclasse ($|Y| > 2$).

A partire dal risultato di una elaborazione lineare dell'input, la funzione sigmoide logistica

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

viene utilizzata per "spalmare" l'output nell'intervallo $(0,1)$, interpretando tale valore come la probabilità di appartenere ad una delle due classi.

Si supponga che la funzione lineare utilizzata sia $\theta^\top x$, dove x non necessariamente coincide, in questo caso, con l'input. Supponiamo di voler risolvere un problema di classificazione binaria in cui $y \in \{0,1\}$. Allora se consideriamo

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

possiamo assumere che essa rappresenti la probabilità stimata che $y = 1$ dato l'input x , indicando tale valore con h_θ :

$$h_\theta(x) = p(y = 1|x; \theta) = \frac{1}{1 + e^{-\theta^\top x}}$$

Ciò è sufficiente in un problema di classificazione binaria, poichè le probabilità di classificazione sono complementari

$$p(y = 0|x; \theta) = 1 - p(y = 1|x; \theta) = 1 - h_\theta(x)$$

Chiaramente, a questo punto, la classificazione è simmetrica:

$$\begin{cases} h_\theta(x) \geq 0.5 \implies \text{classifica } y = 1 \\ h_\theta(x) < 0.5 \implies \text{classifica } y = 0 \end{cases}$$

Anche se questo approccio è noto come regressione logistica, in realtà serve a risolvere un problema di classificazione (binaria) e non di regressione, come suggerisce il nome.

Nel caso di classificazione multiclasse, l'impiego della funzione di attivazione softmax consente di trasformare un insieme di valori ottenuti sulla base di una certa eleaborazione in probabilità associate alle classi.

Si consideri un problema di classificazione su un dataset costituito da m esempi, ad esempio m foto:

$$\mathbb{X} = \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$$

cui siano associate le rispettive etichette

$$\mathbb{Y} = \{y^{(1)}, y^{(2)}, \dots, y^{(m)}\}$$

con ciascuna etichetta nella forma

$$y = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

in cui $y_i = 1$ identifica l'appartenenza di un esempio x alla classe i -esima.

Se z è il vettore colonna di output di dimensione $C \times 1$ con C numero delle classi, allora la funzione softmax calcola per ciascun $z_i \in z$

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

consente di produrre un vettore di output \hat{y} in cui ciascun elemento

$$\hat{y}_i = \text{softmax}(z)_i = p(y = i|x)$$

rappresenta la probabilità che l'input x appartiene alla classe i . ($0 \leq y_i \leq 1$, $\sum_i \hat{y}_i = 1$, $i = 1, \dots, C$).

Questo approccio può essere visto come la generalizzazione della funzione sigmoide, utilizzata per rappresentare una distribuzione di probabilità di una variabile aleatoria discreta con n possibili valori, piuttosto che binaria.

Cross-Entropy Loss Function La *loss function* comunemente usata per i problemi di classificazione è detta *cross-entropy* ed è definita da:

$$\mathcal{L}(\hat{y}, y) \triangleq - \sum_i p(y_i) \log(q(\hat{y}_i)) = - \log(q(\hat{y}))$$

La prima equazione corrisponde alla definizione di *cross-entropy* tra le due distribuzioni di probabilità $p(y_i)$ e $q(\hat{y}_i)$. La seconda equazione si ottiene considerando che in uno scenario di classificazione $p(y_i) = 1$ solo in corrispondenza del valore corretto y_i .

La funzione $(q(\hat{y}_i))$ corrisponde, invece, alla funzione softmax per la classe i nel caso di classificazione multiclasse. Si può quindi scrivere:

$$\mathcal{L}(\hat{y}; i) = - \log(\text{softmax}(z)_i) = - \log \left(\frac{e^{z_i}}{\sum_{j=1}^C e^{z_j}} \right)$$

in cui z rappresenta un generico stadio di elaborazione dell'input

$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_C \end{bmatrix}$$

La funzione appena presentata è implementata in `softmaxlayer` (par. 3.4.4).

3.3 Reti neurali

Le reti neurali artificiali (ANN), dette anche percettroni multilivello (MLP) costituiscono i modelli di deep learning per eccellenza.

Una rete neurale può essere interpretata come un insieme di strati (layer) composti ciascuno da un certo numero di unità computazionali, dette neuroni, in grado di fornire una nuova rappresentazione dell'input. Le funzioni sono composte a formare una catena di rappresentazioni sconosciute (per questo dette *hidden layers*), nel senso che ciascun layer calcola una funzione dell'output del layer precedente : a partire da rappresentazioni più semplici, esse vengono raggruppate fino ad un livello di "arbitraria" complessità

$$f(\mathbf{x}) = f^{(d)} \left(\underbrace{\dots (\mathbf{h}^{(3)}(\mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}))))}_{\text{hidden layers}} \right)$$

in cui:

- d è la profondità della rete ovvero il numero di layers
- $\mathbf{h}^{(1)}$ è il primo (hidden) layer, $\mathbf{h}^{(2)}$ il secondo (hidden) layer e così via: ciascuno di essi rappresenta una trasformazione parametrica non lineare delle feature relative ad un input \mathbf{x} : $\mathbf{h}^{(i)}(\mathbf{x}) = \Phi(\mathbf{x}; \Theta)$. In particolare, ogni hidden layer \mathbf{h} compie generalmente la seguente operazione: accetta un vettore in input \mathbf{x} , calcola una trasformazione affine $\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$, quindi applica una funzione non lineare $g(\mathbf{z})$ elemento per elemento, detta funzione di attivazione. Si ottiene così:

$$\begin{aligned} \text{Layer 1: } \quad \mathbf{h}^{(1)} &= g^{(1)}(\mathbf{W}^{(1)\top} \mathbf{x} + \mathbf{b}^{(1)}) \\ \text{Layer 2: } \quad \mathbf{h}^{(2)} &= g^{(2)}(\mathbf{W}^{(2)\top} \mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\ \text{Layer 3: } \quad \mathbf{h}^{(3)} &= g^{(3)}(\mathbf{W}^{(3)\top} \mathbf{h}^{(2)} + \mathbf{b}^{(3)}) \\ &\vdots \end{aligned}$$

Fino a giungere all'output layer.

- $f^{(d)}$ è l'output layer, che ha il ruolo di fornire un'ultima trasformazione al fine di completare il task che la rete deve eseguire. Le scelte solitamente sono:

- Layer di output lineare: viene calcolata una ulteriore trasformazione affine, del tipo

$$\hat{\mathbf{y}} = \mathbf{W}^\top \mathbf{h}^{(d-1)} + \mathbf{b}$$

- Layer di output sigmoide: viene applicata una trasformazione affine, quindi usata la funzione sigmoide σ per convertire il risultato in una probabilità:

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{h}^{(d-1)} + b)$$

Questo approccio è usato nel caso di classificazione binaria, come descritto al par. 3.2.

- Layer di output softmax: viene calcolata una trasformazione affine

$$\mathbf{z} = \mathbf{W}^\top \mathbf{h}^{(d-1)} + \mathbf{b}$$

Quindi la funzione softmax

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

consente di produrre un vettore di output $\hat{\mathbf{y}}$ in cui ciascun elemento

$$\hat{y}_i = p(y = i | \mathbf{x})$$

rappresenta la probabilità dell'input \mathbf{x} di appartenere alla classe i . ($0 \leq y_i \leq 1$, $\sum_i \hat{y}_i = 1$), come descritto al par. 3.2

Generalmente si raccomanda di utilizzare, come funzione di attivazione, la seguente ReLU (utilizzata nel presente lavoro, par. 4.4.2):

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

Altre possibilità sono la sigmoide $g(z) = \sigma(z)$ oppure la tangente iperbolica $g(z) = \tanh(z)$.

L'aggettivo *deep* associato a *deep network* e, per estensione, a *deep learning* non individua una misura universalmente riconosciuta per un modello. In generale, si usa per distinguere i tradizionali metodi di machine learning da quelli recenti che includono una maggiore quantità di composizione di funzioni da apprendere, ovvero una maggiore profondità d di una rete neurale, con un numero di parametri Θ molto elevato.

Anzichè pensare ad un layer come ad una singola funzione vettoriale, possiamo considerare che il layer sia costituito da tante unità computazionali che agiscono in parallelo, ognuna delle quali calcola la propria funzione vettore-scalare: esegue una somma pesata degli input ed applica un bias (fig. 3.5). Ogni unità è analoga ad un neurone (fig.3.4) nella misura in cui riceve input da molte altre unità (dendriti) e calcola una funzione di attivazione trasmessa come output ad altre unità del layer successivo (assone). La scelta delle funzioni $f^{(i)}(\mathbf{x})$ è generalmente legata ad una rappresentazione molto semplificata della funzione che il neurone biologico calcola.

Graficamente, il modello di una rete neurale può essere rappresentato da un grafo diretto aciclico (*computational graph*) che descrive in che modo le funzioni sono composte. L'aggettivo *feedforward*, spesso abbinato a reti neurali, si riferisce al fatto che per ottenere l'output $\hat{\mathbf{y}}$, l'informazione fluisce attraverso tutte le computazioni intermedie utilizzate per definire l'approssimazione $f(\mathbf{x}; \Theta) \sim f^*(\mathbf{x})$.

La non linearità introdotta negli hidden layer di una rete neurale attraverso le funzioni di attivazione e l'elevata dimensionalità del numero dei parametri rendono la *loss function* non convessa, per cui si è obbligati ad utilizzare procedure iterative di ottimizzazione per risolvere il problema 3.10, descritte nel paragrafo successivo.

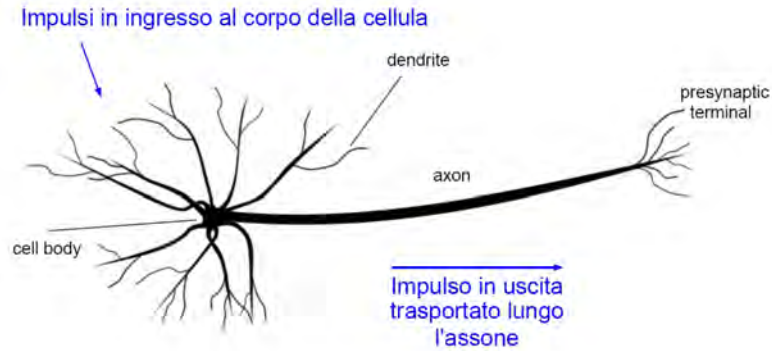


Figura 3.4: Neurone. Immagine disponibile all'indirizzo <https://thenounproject.com/term/neuron/214105/>

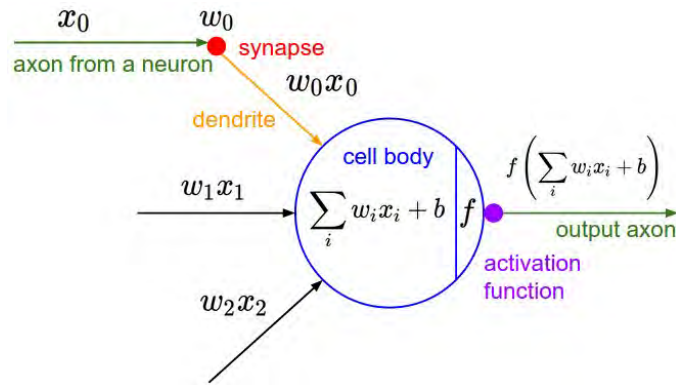


Figura 3.5: Modello computazionale del neurone secondo Hebb

Le reti neurali traggono ispirazione dal funzionamento biologico del cervello. In realtà è bene pensare ad esse non come modelli computazionali del cervello, ma come approssimatori universali di funzioni progettati per ottenere generalizzazione statistica. È stato dimostrato che una rete neurale feedforward con un layer di output lineare e almeno un hidden layer che abbia una funzione di attivazione tra quelle elencate in precedenza può approssimare una funzione continua su un insieme chiuso e limitato di \mathbb{R}^n . L'errore può essere arbitrario se si utilizza una rete con profondità elevata.

Questo teorema fornisce una giustificazione teorica molto generica del funzionamento delle reti neurali, ma non specifica nessun dettaglio riguardante l'architettura da utilizzare per poter ottenere alcune garanzie nell'approssimazione. Le scelte riguardanti il numero di layer, il numero di pesi per ciascun layer, il tipo di connettività tra i neuroni, le funzioni di attivazione, il layer di output, ecc. derivano quasi sempre da risultati sperimentali piuttosto che teorici.

3.3.1 Metodo del gradiente

La derivata è utile per minimizzare una funzione $f(x)$ perchè suggerisce come cambiare x al fine di produrre un piccolo incremento in y . Preso un ϵ sufficientemente piccolo si ha che:

$$f(x - \epsilon \text{sign}[f'(x)]) < f(x)$$

- $f'(x) > 0 \implies \text{sign}(f'(x)) = 1$, con $f(x)$ crescente $\implies f(x - \epsilon) < f(x)$
Il valore di $f(x)$ può essere ridotto decrementando x di ϵ , cioè con un piccolo spostamento a sinistra sull'asse delle ascisse
- $f'(x) < 0 \implies \text{sign}(f'(x)) = -1$, con $f(x)$ decrescente $\implies f(x + \epsilon) < f(x)$
Il valore di $f(x)$ può essere ridotto incrementando x di ϵ , cioè con un piccolo spostamento a destra sull'asse delle ascisse

Nel caso di funzione di n variabili

$$f : \mathbb{R}^n \rightarrow \mathbb{R} \quad f(\mathbf{x}) = f(x_1, x_2, \dots, x_n)$$

la derivata parziale $\frac{\partial}{\partial \theta_i} f(\boldsymbol{\theta})$ misura come f cambia rispetto alla variazione della sola variabile θ_i . Nelle ipotesi in cui f sia sufficientemente regolare (le derivate esistono ed eventualmente verificano qualche regolarità in termini di continuità), è definito il gradiente di f nel punto $\mathbf{x} = [x_1, x_2, \dots, x_n]$:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial}{\partial \theta_1} f(\mathbf{x}), \frac{\partial}{\partial \theta_2} f(\mathbf{x}), \dots, \frac{\partial}{\partial \theta_n} f(\mathbf{x}) \right]^\top$$

È possibile quindi utilizzare la stessa strategia del caso monodimensionale: si vuole generare una successione di vettori che convergono verso il punto di minimo (metodo di Cauchy).

A partire da un punto $\mathbf{x} = (x_1, x_2, \dots, x_n)$ ci si vuole spostare da tale punto nella direzione \mathbf{d} di un passo α , arrivando cioè in $\mathbf{y} = \mathbf{x} + \alpha \mathbf{d}$, in modo tale che $f(\mathbf{y}) < f(\mathbf{x})$.

Utilizzando lo sviluppo in serie di Taylor centrato nel punto \mathbf{x} si ottiene

$$f(\mathbf{y}) = f(\mathbf{x}) + \alpha \nabla f(\mathbf{x}) \cdot \mathbf{d} + \frac{\alpha^2}{2} \mathbf{d} \cdot H(\mathbf{z}) \mathbf{d}$$

dove H è la matrice hessiana e i prodotti che compaiono sono prodotti scalari tra vettori e \mathbf{z} è un punto intermedio tra \mathbf{x} e \mathbf{y} .

Se $\nabla f \cdot \mathbf{d} < 0$ allora, per α sufficientemente piccolo, si ha che $f(\mathbf{y}) < f(\mathbf{x})$. La direzione \mathbf{d} è detta, in tal caso, direzione di discesa.

In particolare, $\mathbf{d} = -\nabla f$ è sicuramente una direzione di discesa, giacchè il gradiente fornisce la direzione di massima variazione locale della funzione.

La funzione *loss function* \mathcal{L} da minimizzare nell'ambito delle reti neurali è parametrica in $\boldsymbol{\theta}$, quindi l'aggiornamento dei parametri è nella direzione negativa del gradiente:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$

Ogni singolo parametro θ è quindi aggiornato a θ^{new} come segue:

$$\theta_j^{\text{new}} \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n) \quad \text{per } j = 0, \dots, n$$

Il calcolo del gradiente può essere eseguito in maniera efficiente ed in modo esatto grazie all'algoritmo *Back Propagation*. Le iterazioni sono interrotte con un'opportuna tolleranza oppure dopo un numero prefissato.

Varianti stocastiche

Nel metodo del gradiente, $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ è calcolato utilizzando l'intero *training set*. Per velocizzare la procedura, si utilizzano le seguenti varianti.

SGD Il metodo *stochastic gradient descent* (SGD), invece, prevede di utilizzare un sottoinsieme dei dati di *training*, detto *minibatch* al fine di valutare $\alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$. Il risultato è una stima non esatta del gradiente. Il numero di iterazioni necessario a valutare il gradiente rispetto a tutti i possibili sottoinsiemi presenti nel *training set* è detto *epoch*. La discesa stocastica del gradiente non presenta garanzie di convergenza ed è molto sensibile alla scelta dei parametri iniziali.

SGDM Al fine di evitare oscillazioni nella procedura iterativa dello *stochastic gradient descent*, si aggiunge un termine detto *momentum*:

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) + \gamma(\boldsymbol{\theta} - \boldsymbol{\theta}^{\text{old}})$$

Questo metodo è utilizzato per l'addestramento dei classificatori all'interno del presente lavoro di tesi (par. 4.4.3).

3.4 Reti neurali convoluzionali

Una rete neurale si dice convoluzionale se, in almeno uno dei layers, è utilizzata un'operazione di convoluzione al posto della consueta moltiplicazione tra matrici.

A causa della natura dell'operazione di convoluzione (par. 2.2.3), le reti neurali convoluzionali si sono rivelate particolarmente adatte al riconoscimento di pattern sia all'interno di segnali monodimensionali come audio o testo sia all'interno di segnali bidimensionali come le immagini. Questo è il motivo principale per cui sono state adottate nell'ambito della classificazione binaria di immagini adottata nel presente lavoro (par. 4.4).

Il Neocognitron di Fukushima (1979) fu probabilmente la prima rete neurale a potersi definire tale e che incorporava le recenti scoperte neurofisiologiche di Huber e Wiessel riguardo alla percezione visiva. Nel seguito si descrive in dettaglio l'architettura tipica utilizzata negli ultimi anni.

Una rete neurale convoluzionale è costituita da uno stack di layers, mostrato in fig. 3.6: In input c'è un'immagine e in output c'è un vettore contenente il responso della classificazione (binaria o con punteggi per ciascuna classe).

- **Input:** generalmente un immagine di dimensioni contenute (es. 224x224)
- **Convolutional layer (CONV):** vengono eseguite diversi filtri sull'input, in maniera indipendente l'uno dall'altro, cioè un certo numero di convoluzioni con lo stesso dato in input ma con diversi kernel. (par. 2.2.3). Viene prodotto un set di *feature map*.
- **Activation layer** (detto anche *detector stage*): ciascuna *feature map* dello stadio precedente è sottoposta ad una funzione di attivazione non lineare, come ad esempio la funzione ReLU. Si ottiene una mappa di attivazione.
- **Pooling layer (POOL):** Viene eseguito un sottocampionamento dell'immagine attraverso una statistica riassuntiva delle regioni
- **Fully-Connected layer (FC):** strati finali analoghi a quelli di una classica rete neurale, finalizzati ad effettuare la classificazione vera e propria a partire dalle *feature* estratte dai livelli precedenti.

In questo modo, una rete CNN trasforma l'immagine originale dai valori originali dei pixel ai punteggi della classe finale.

Si noti che alcuni livelli contengono parametri e altri no. In particolare, i livelli Convolutional e Fully-Connected eseguono trasformazioni che sono funzione non solo delle attivazioni nel volume di input, ma anche dei parametri (i pesi e i bias dei neuroni). Invece, gli strati di attivazione e pooling implementeranno una funzione fissa. I parametri dei livelli CONV/FC vengono appresi in fase di addestramento attraverso l'applicazione dello *stochastic gradient descent*, in modo che i punteggi di classe calcolati all'ultimo livello siano coerenti con le etichette del *training set* per ogni immagine.

Si descrivono nel seguito con maggiore dettaglio i singoli layer, con le relative tecniche di implementazione in Matlab, utilizzate nel presente lavoro (par. 4.4.2).

Come principio generale di implementazione, è possibile definire l'architettura di una rete convoluzionale come un array **layers** che contiene gli oggetti layers desiderati, elencati sequenzialmente così come devono essere connessi (par. 4.4.2). In Matlab esistono numerosi layer specificati come oggetti, ciascuno con la propria funzionalità, nel seguito descritti. Tale array può quindi essere passato come argomento della funzione **trainNetwork** al fine di addestrare la rete.

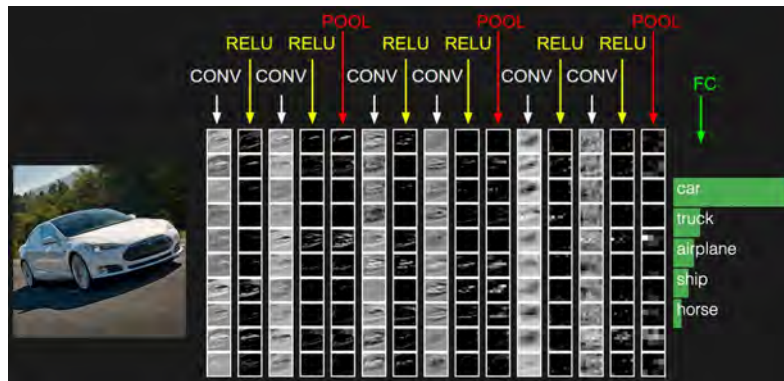


Figura 3.6: Le attivazioni di un esempio di architettura ConvNet. Il volume iniziale memorizza i pixel dell'immagine grezza (a sinistra) e l'ultimo volume memorizza i punteggi di classe (a destra). Ogni volume di attivazioni lungo il percorso di elaborazione è mostrato come una colonna, in cui ogni immagine è una mappa di attivazione. Il volume dell'ultimo livello contiene i punteggi per ogni classe

3.4.1 Input Layer

Rappresenta il layer di input alla rete per immagini 2D.

```
layer = imageInputLayer(inputSize, 'Property', Value)
```

- **inputSize**

Dimensione delle immagini in input, specificata come un vettore riga di 3 interi [h w c], con hwx dimensione dell'immagine e c numero di canali. Nel caso di immagini RGB, c deve essere pari a 3.

- **'Normalization', 'zero-centered' oppure 'none'**

Specifica la trasformazione dei dati applicata in questo layer. Di default, viene applicata una centratura dei dati intorno allo zero, sottraendo l'immagine media del training set da ogni immagine di input. L'immagine media viene calcolata automaticamente a tempo di training dalla funzione `trainNetwork`. In alternativa è possibile specificare esplicitamente l'immagine media da usare come parametro `'AverageImage'`

3.4.2 Convolution Layer

Il layer di convoluzione è il "cuore" delle reti neurali convoluzionali, in cui vengono effettuate importanti operazioni di filtraggio. Per ogni finestra dell'input, viene eseguita una moltiplicazione elemento per elemento dei pixel per i pesi del kernel e viene aggiunto un termine di bias al risultato (fig. 3.7) (analogamente all'operazione implementata in 6.1 ed illustrata nel par. 2.2.3).

In Matlab, la creazione del layer avviene attraverso la funzione `convolution2dLayer`:

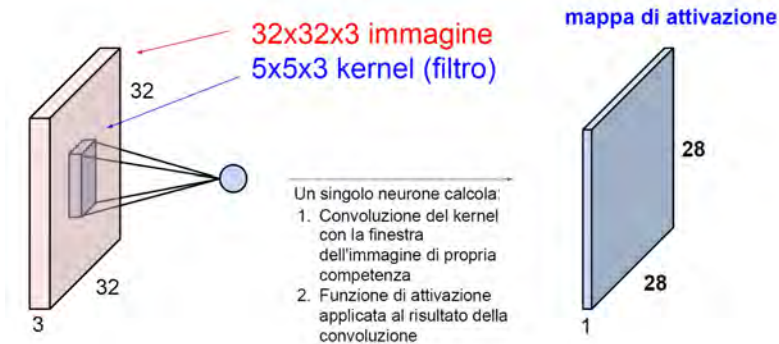


Figura 3.7: Esempio di computazione di un neurone attraverso un layer di convoluzione e di attivazione. Utilizzando una convoluzione valida con stride pari a 1 e senza input padding, la convoluzione di un'immagine $32 \times 32 \times 3$ con un filtro $5 \times 5 \times 3$ produce una matrice $28 \times 28 \times 1$, cui viene applicata la funzione d'attivazione scelta per ottenere una activation map. Tutti i 28×28 neuroni impiegati per il calcolo di ogni singolo elemento che compone la mappa di attivazione utilizzano lo stesso filtro, cioè un kernel con gli stessi parametri (parameter sharing)

```
layer = convolution2dLayer(filterSize,numFilters,'Property',Value)
```

Gli argomenti sono:

- **filterSize**

Dimensione del kernel, specificato come un intero F per ottenere un filtro rettangolare $F \times F$. Se si vogliono utilizzare dimensioni differenti è possibile specificare un vettore $[h \ w]$.

- **numFilters**

Numero dei filtri, specificato come un intero positivo K . Questo parametro corrisponde al numero di neuroni nel layer di convoluzione che sono connessi alla stessa regione dell'input. La mappa di feature in output avrà esattamente K livelli di profondità. (fig. 3.8)

Gli eventuali parametri che possono essere specificati sono:

- **'Stride', S**

Parametro che regola lo scorrimento della finestra sull'input. Se si specifica un intero positivo S , gli scorrimenti della finestra sono caratterizzati da una traslazione di S pixel in orizzontale e di S pixel in verticale. Se si vogliono utilizzare traslazioni differenti rispetto alle due dimensioni è possibile specificare un vettore $[h \ w]$

- **'DilationFactor', L**

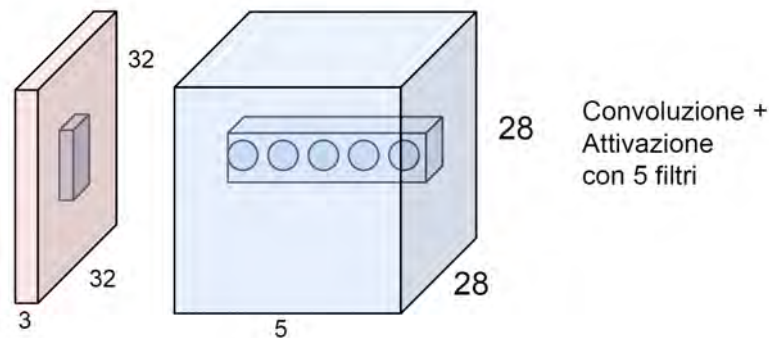


Figura 3.8: I 5 neuroni mostrati applicano un diverso filtro (un kernel con diversi parametri) alla stessa regione spaziale dell'immagine in input, in maniera indipendente l'uno dall'altro.

Fattore che specifica una convoluzione "dilatata". Se si specifica un intero positivo L maggiore di 1, è come se il filtro venisse ampliato inserendo $L-1$ zeri tra ogni elemento. Il risultato è quello di ampliare la dimensione del receptive field rispetto all'input. Se si vuole utilizzare una dilatazione differente rispetto alle due dimensioni è possibile specificare un vettore $[h \ w]$.

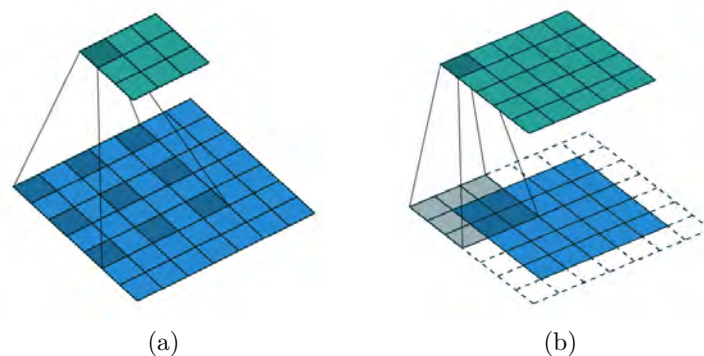


Figura 3.9: Esempio di (a) convoluzione dilatata con fattore $L=2$ (b) padding equidistribuito con fattore $P=1$

- 'PaddingSize', $[t \ b \ l \ r]$

Dimensione del padding da applicare all'input, specificata come un vettore di quattro interi positivi. Indicano rispettivamente il numero di righe nulle da aggiungere in alto (top) e in basso (bottom) e il numero di colonne nulle da aggiungere a sinistra (left) e a destra (right). Di default, il padding è nullo.

- 'PaddingMode', 'manual' oppure 'same'

Questa proprietà è utile se si vuole preservare la dimensione dell'input attraverso un layer di convoluzione. Specificando il valore `'same'`, infatti, viene calcolato automaticamente il valore della dimensione del padding adatto a perseguire tale scopo. Per un filtro quadrato di dimensione F , ad esempio, è necessario applicare un padding uniforme $[P \ P \ P \ P]$ con $P = (F-1)/2$.

- `'NumChannels'`, `'auto'` oppure n

Numero di canali (livelli di profondità) per ciascun filtro. Di default, questo parametro è sempre uguale alla profondità del tensore di input al livello di convoluzione

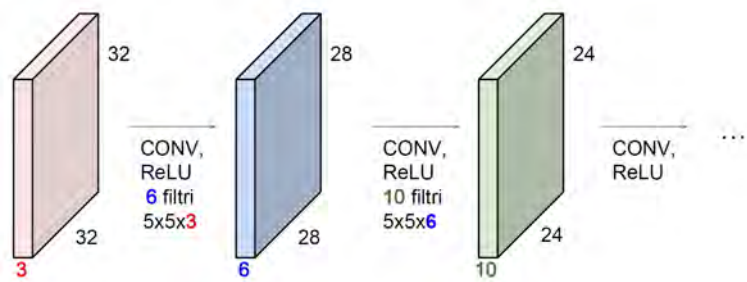


Figura 3.10: La figura mostra come la profondità dei filtri di ogni livello di convoluzione debba essere equivalente alla profondità dell'input del rispettivo livello

- `'Weights'`, W

`Weights` è l'array contenente i pesi del layer, ovvero i parametri che definiscono le convoluzioni e che vengono appresi dalla rete durante la fase di addestramento. Se l'input al layer è di profondità D_i , il kernel è di dimensione F e il numero di filtri è pari a K , allora `Weights` sarà un array 4-D `single` di dimensione $F \times F \times D_i \times K$, in cui la quarta dimensione indicizza ogni filtro. Specificando questa proprietà, è possibile inizializzare i pesi con un vettore a propria scelta W .

- `'Bias'`, B

`Bias` è l'array contenente i parametri di bias, ovvero i parametri che vengono sommati al risultato delle convoluzioni, anch'essi appresi dalla rete durante la fase di addestramento. Se K è il numero di filtri applicato nel layer, allora `Bias` sarà un array 3-D `single` di dimensioni $1 \times 1 \times K$

- `'WeightsInitializer'`, `'glorot'` | `'he'` | `'narrow-normal'` | `'zeros'` | `'ones'`

Funzione utilizzata per l'inizializzazione dei pesi in `Weights`. Le alternative sono:

- `'glorot'`: generatore pseudocasuale utilizzato di default, con una distribuzione uniforme a media nulla e varianza $2/(\text{numIn} + \text{numOut})$, con $\text{numIn} = F \times F \times D_i$ e $\text{numOut} = F \times F \times K$

- `'he'`: generatore pseudocasuale di una distribuzione uniforme a media nulla e varianza $2/\text{numIn}$, con $\text{numIn}=F \times F \times D_i$
 - `'narrow-normal'`: generatore pseudocasuale di una distribuzione uniforme a media nulla e varianza 0.01
 - `'zeros'`: inizializza con degli zeri
 - `'ones'`: inizializza con uno
 - `function handle`: utilizza una funzione custom
- `'BiasInitializer','zeros'|'narrow-normal'|'ones'|function handle`

Learn rate e regolarizzazione

- `'WeightLearnRateFactor',alfa`
Fattore da moltiplicare per il parametro learning rate globale reattivo all'apprendimento dei pesi, di default pari a 1
- `'BiasLearnRateFactor',beta`
Fattore learning rate da moltiplicare per il parametro learning rate globale reattivo all'apprendimento dei biases, di default pari a 1
- `'WeightL2Factor',lambda1`
Fattore da moltiplicare per il parametro globale di regolarizzazione L2 reattivo all'apprendimento dei pesi, di default pari a 1
- `'BiasL2Factor',lambda2`
Fattore da moltiplicare per il parametro globale di regolarizzazione L2 reattivo all'apprendimento dei bias, di default pari a 1

Il layer di attivazione, generalmente specificato subito dopo un layer di convoluzione, è implementato, nel caso di funzione ReLU, mediante `reluLayer`.

3.4.3 Pooling layer

Una funzione di pooling opera un sottocampionamento sull'immagine, come mostrato in figura 3.11. Ogni regione di una certa dimensione dell'immagine in input (es. 4×4) viene ridotta ad un unico pixel, il cui valore è calcolato tramite una statistica riassuntiva della regione di partenza (es. max, norma L2).

L'operazione di pooling è utile per diverse ragioni. Aiuta a rendere la rappresentazione di feature approssimativamente invariante a traslazioni dell'input. Ad esempio, l'operazione di max pooling è sensibile solo al massimo valore in una regione rettangolare, a prescindere da dove essa sia posizionata all'interno dell'immagine. Se tale regione subisce una traslazione, l'operazione di max pooling produrrà un risultato simile o al più uguale rispetto all'assenza di traslazione. (l'uguaglianza non è sempre verificata, per questo si parla di rappresentazione invariante in maniera approssimativa).

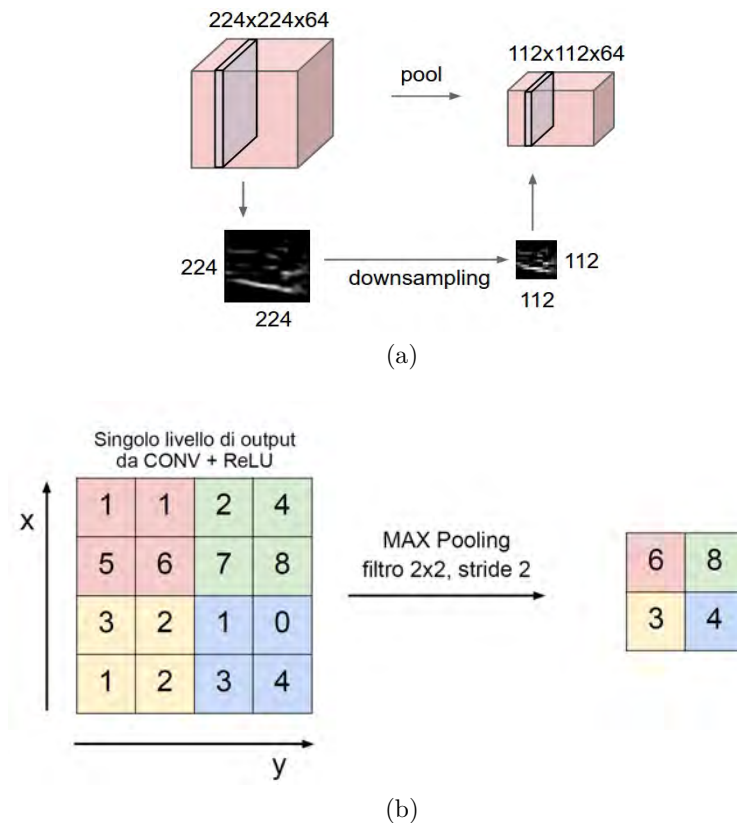


Figura 3.11: Operazione di pooling : (a) effetto di downsampling e (b) calcolo del max pooling

Se l'operazione di pooling è calcolata rispetto agli output di diverse convoluzioni (diversi filtri), anziché rispetto a diverse regioni di un'unica immagine filtrata, è possibile insegnare alla rete altre trasformazioni rispetto a cui diventare invariante (es. rotazioni, scala). Infatti, un'unità di pooling che calcola una statistica riassuntiva rispetto a diversi filtraggi eseguiti con parametri differenti può imparare ad essere invariante rispetto a tali filtraggi. La figura 3.12 mostra un esempio intuitivo di come un insieme di tre filtri ed una unità di max pooling può rendere la rete invariante rispetto alla rotazione dell'input.

L'altra utilità del pooling è quella di ridurre progressivamente la dimensione spaziale della rappresentazione per ridurre la quantità di parametri e di calcolo nella rete. Tipicamente, l'operazione è eseguita mediante un filtro di dimensioni 2x2 (o anche 3x3) applicato con uno stride di 2. Il risultato è quello di ridurre del 75% il numero delle attivazioni propagate. Ogni operazione MAX richiederebbe in questo caso un massimo di 4 numeri (regione 2x2). La dimensione della profondità del tensore di attivazione rimane invariata.

La funzione utilizzata per la sua creazione è `maxPooling2dLayer`

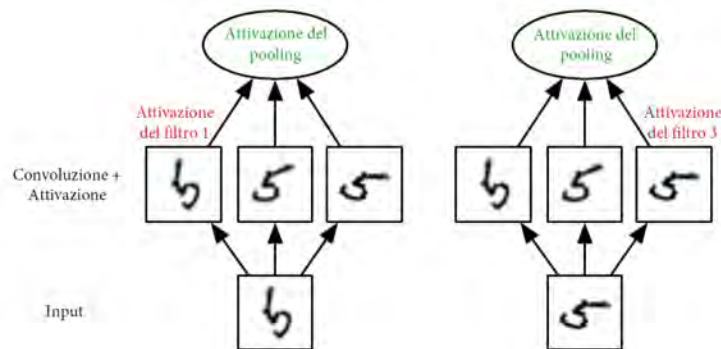


Figura 3.12: I tre filtri mostrati nello strato intermedio servono a riconoscere un 5 scritto a mano, ma ciascuno di essi produce un'attivazione solo in corrispondenza di una particolare orientazione del numero. Eseguendo un'operazione di max pooling rispetto al risultato di attivazione dei tre filtri, l'output del pooling sarà simile, in presenza di diversi input, a prescindere dal filtro attivato. La rete potrà in questo modo riconoscere la presenza di un 5 a prescindere dalla sua orientazione.

```
layer = maxPooling2dLayer(poolSize, 'Property', Value)
```

- **PoolSize**

Dimensione delle regioni di pooling, specificata come un intero positivo F , in caso di finestra quadrata oppure un vettore di due interi positivi $[h \ w]$, con h altezza e w larghezza della finestra

Le proprietà di questo layer sono del tutto analoghe a quelle del `convolution2dLayer` per ciò che riguarda il comportamento ai bordi e le modalità di traslazione della finestra. Si possono perciò parimenti specificare le proprietà `'Stride'`, `'PaddingSize'` e `'PaddingMode'`.

3.4.4 Fully Connected Layer

Moltiplica l'input per un tensore di pesi e aggiunge un vettore bias. Tutti i neuroni sono connessi tra loro. Un esempio numerico è mostrato in fig. 3.13.

```
layer = fullyConnectedLayer(outputSize, 'Property', Value)
```

- **outputSize**

Dimensione di uscita del fully connected layer, specificato come un numero intero positivo C . In caso di classificazione, C deve essere uguale al numero delle classi da individuare all'interno del dataset

La funzione softmax di attivazione finale, inserita in coda all'ultimo *fully connected layer* è implementata mediante `softmaxLayer`.

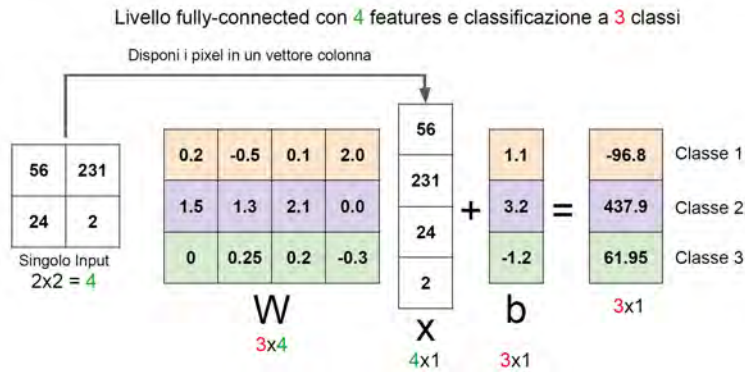


Figura 3.13: Esempio numerico di livello fully connected con un singolo input

3.4.5 Image Preprocessing

Al fine di migliorare la capacità di generalizzazione di un modello di machine learning, certamente una possibilità è quella di utilizzare più dati per il training. Nella pratica però la quantità di dati a disposizione è sempre limitata, a seconda del contesto e del task. Un modo per aggirare questo problema è quello di creare aggiungere dati "falsi" al training set, dove per "falsi" si intende prodotti attraverso modifiche ai dati a disposizione. Questa pratica va sotto il nome di Dataset Augmentation.

Questo approccio si rivela particolarmente efficiente per i problemi di classificazione e, in modo particolare, per i problemi di image recognition. Le classi rispetto a cui il classificatore deve produrre un output sono infatti spesso invarianti rispetto ad un'ampia varietà di trasformazioni negli input, facilmente realizzabili. Operazioni come traslazioni, rotazioni, scalatura possono essere implementate come semplici operazioni geometriche sui pixel (trasformazioni affini) e si rivelano particolarmente utili (se eseguite in modo casuale ma accorto) per la riduzione dell'errore di generalizzazione della maggior parte di modelli di computer vision.

Una delle tradizionali attività di preprocessing nell'ambito della computer vision è la normalizzazione dei pixel. Solitamente un'immagine a colori rappresentata come un tensore a 3 canali RGB ha ogni pixel rappresentato da un intero senza segno di 1 byte, quindi nel range $[0; 255]$. Si può quindi fare in modo che tutti i pixel siano normalizzati in un range ragionevole, come $[0,1]$ oppure $[-1; 1]$

Altro preprocessing riguarda la dimensione delle immagini. Spesso l'architettura del modello di computer vision (es. una rete neurale convoluzionale) richiede immagini di una stessa dimensione standard, per cui tutte le immagini devono essere ritagliate oppure scalate per essere adattate a tali dimensioni. Esistono anche modelli convoluzionali che accettano input variabili e/o output variabili.

Altre idee di preprocessing sono:

- Mostra al modello diverse versioni dello stesso input (ad esempio la stessa immagine ritagliata in diversi punti)
- Ridurre un qualche tipo di variazione nei dati facilmente descrivibile da parte del progettista umano e di cui egli è consapevole non sia rilevante per il task
- Per larghi dataset solitamente si fanno poche operazioni di preprocessing di questo tipo poichè si lascia imparare al modello a quale tipo di fattori di variazione esso deve essere invariante (per esempio il sistema AlexNet per ImageNet prevede un solo step di preprocessing: sottrarre la media) attraverso gli esempi di training di ciascun pixel)

Normalizzazione

Il contrasto rappresenta l'entità della differenza tra i pixel luminosi e i pixel scuri all'interno di un'immagine. Il contrasto è considerato una sorgente di variazione poco interessante e facilmente rimovibile. Nell'ambito del deep learning il contrasto si quantifica spesso come la deviazione standard dei pixel all'interno dell'immagine o di una porzione di essa. Detta \mathbf{X} l'intensità media di una immagine RGB $\mathbf{X} \in \mathbb{R}^{m \times n \times 3}$

$$\bar{\mathbf{X}} = \frac{1}{3mn} \sum_{i=1}^m \sum_{j=1}^n \sum_{k=1}^3 x_{i,j,k}$$

Il contrasto dell'intera immagine è dato da

$$\text{contrasto}(\mathbf{X}) = \sqrt{\frac{1}{3mn} \sum_{i=1}^m \sum_{j=1}^n \sum_{k=1}^3 (x_{i,j,k} - \bar{\mathbf{X}})^2}$$

Le tecniche più diffuse per la riduzione del contrasto sono:

- Global contrast normalization (GCN). Posiziona tutte le immagini del dataset sulla stessa scala di contrasto, in modo da evitare all'algoritmo di gestire diverse scale di contrasto
- Local contrast normalization (LCN). Prevede che il contrasto sia normalizzato all'interno di ciascuna piccola finestra piuttosto che sull'intera immagine. Ciò permette al modello di focalizzarsi sui soli bordi di una immagine ad esempio.

Augmentation

Si descrive brevemente l'oggetto `augmentedImageDatastore`, progettato appositamente per l'addestramento di un classificatori come reti neurali convoluzionali. Esso consente di trasformare data set per il training, la validazione, il test e la predizione con eventuali operazioni di preprocessing come il ridimensionamento, la rotazione e la riflessione. Il ridimensionamento è necessario affinché tutte le immagini rispettino la dimensione di input della rete neurale, che è fisso.

augmentedImageDatastore Per le fasi di addestramento e predizione di una CNN è possibile creare un oggetto `augmentedImageDatastore` a partire da una collezione di immagini di tipo `ImageDatastore`. Un oggetto `augmentedImageDatastore` può essere passato semplicemente come argomento della funzione `trainNetwork`, per essere utilizzato come *training set* di una rete neurale. In tal caso, per ogni epoch, i dati di training vengono casualmente perturbati, in modo che ciascuna epoch usi un data set leggermente diverso. Il numero effettivo di immagini usate per ciascuna epoch non cambia. Le immagini trasformate non vengono memorizzate. L'`imageInputLayer` di una CNN normalizza le immagini ad ogni epoch, sottraendo un'intensità media costante e calcolata un'unica volta, subito dopo la augmentation relativa alla prima epoch.

La creazione di un oggetto `augmentedImageDatastore` avviene invocando la funzione

```
auimds = augmentedImageDatastore(outputSize,imds)
```

Le sue proprietà sono:

- **NumObservations:** Proprietà READ-ONLY. Numero totale di "osservazioni" nella collezione di immagini aumentate. Coincide con la lunghezza di ogni training epoch, quindi coincide con l'omonima proprietà dell'oggetto `trainingOptions` e può essere modificato solo di lì.
- **Files:** oggetto `cell array` contenente le directory delle immagini (ereditate dall'oggetto `imageDatastore` con cui è stato costruito)
- **MiniBatchSize:** Proprietà READ-ONLY (coincide con l'omonima proprietà dell'oggetto `trainingOptions` e può essere modificato solo di lì.) Dimensione del minibatch utilizzato per addestramento, predizione, classificazione (coincide con l'omonima proprietà definita nelle specifiche di ottimizzazione del `trainingOptions`, da passare a sua volta in input alla funzione `trainNetwork`)
- **DataAugmentation:** oggetto di tipo `imageDataAugmenter` che consente di definire trasformazioni di preprocessing da applicare alle immagini
- **ColorPreprocessing:** eventuali conversioni `'gray2rgb'` o `'rgbtogray'` per fare in modo che tutte le immagini abbiano lo stesso numero di canali richieste dall'`imageInputLayer` della rete neurale.
- **OutputSize:** dimensione delle immagini di output, come vettore di due interi positivi. L'operazione di ridimensionamento è l'unica operazione di augmentation prevista di default, al fine di rendere le immagini compatibili con la dimensione di input specificata nell'`imageInputLayer`
- **OutputSizeMode:** metodo utilizzato per il ridimensionamento delle immagini
 - `'resize'`: le immagini vengono scalate utilizzando la interpolazione bilineare con antialiasing (operazione più veloce ma con risultati di qualità inferiore rispetto alla interpolazione bicubica utilizzata da `imresize`. Evita distorsioni causate dalla interpolazione nearest-neighbor). Questa opzione è di default.

- **'centercrop'**: viene effettuato un ritaglio al centro dell'immagine della dimensione specificata in **OutputSize**
- **'randcrop'**: viene effettuato un ritaglio in una posizione casuale dell'immagine della dimensione specificata in **OutputSize**

imageDataAugmenter La creazione di un oggetto **imageDataAugmenter** consente di definire operazioni di preprocessing (par. 2.2.2) effettuate all'interno di una collezione di immagini **imageDatastore**. La sintassi è:

```
aug = imageDataAugmenter('Property',Value)
```

in cui è possibile definire le opzioni di image augmentation esprimendo una o più coppie proprietà-valore. Le possibilità sono

- **'FillValue',[x y z]**

Valore di riempimento utilizzato per definire i punti fuori campo al momento del ricampionamento, specificato come un vettore numerico od uno scalare, a seconda del numero di canali dell'immagine.

- **'RandXReflection', true|false**
'RandYReflection', true|false

Ogni immagine subisce con una probabilità del 50% una riflessione orizzontale (X) oppure verticale (Y)

- **'RandRotation',[a b]**

Applica una rotazione con un angolo, espresso in gradi, estratto da una distribuzione uniforme pseudocasuale nell'intervallo $[a, b]$

- **'RandScale',[a b]**
'RandXScale',[a b]
'RandYScale',[a b]

Applica una scalatura isotropica oppure orizzontale (X) o verticale (Y), con fattore di scala estratto da una distribuzione uniforme pseudocasuale nell'intervallo $[a, b]$

- **'RandXShear',[a b]**
'RandYShear',[a b]

Applica una mappa di tipo shear orizzontale (X) oppure verticale (Y) all'immagine. $[ab]$ è l'intervallo da cui è estratto l'angolo di shear, compreso tra -90 e 90 gradi

- **'RandXTranslation',[a b]**
'RandYTranslation',[a b]

Applica una traslazione orizzontale (X) oppure verticale (Y) all'immagine. L'entità della traslazione, misurata in pixel, è estratta da una distribuzione uniforme pseudocasuale nell'intervallo $[a, b]$

3.5 Cross-validation

La *cross-validation* è una tecnica usata principalmente nel *machine learning* per stimare l'abilità di un modello di apprendimento su dati non visti e ottenere una variabilità nell'apprendimento (par. 4.4.3). Si tratta di un metodo popolare perché è semplice da capire e perché generalmente si traduce in una stima meno parziale o meno ottimistica delle capacità del modello rispetto ad altri metodi, come ad esempio una semplice suddivisione *train/test*.

La procedura generale è la seguente:

1. Mescolare il set di dati in modo casuale;
2. Dividere il set di dati in k gruppi detti *fold*;
3. Per ogni singolo gruppo:
 - (a) Selezionare un gruppo per il *test set*
 - (b) Prendere i gruppi rimanenti come *training set*
 - (c) Addestrare un modello con il *training set* e valutarlo sul *test set*;

È importante sottolineare che ogni osservazione del campione di dati viene assegnata ad un singolo gruppo e rimane in quel gruppo per tutta la durata della procedura. Ciò significa che ad ogni campione viene data l'opportunità di essere utilizzato nell'intervallo di tempo impostato 1 volta e utilizzato per allenare il modello $k - 1$ volte.

Questo approccio comporta la suddivisione casuale dell'insieme di osservazioni in gruppi k , o *fold*, di dimensioni approssimativamente uguali. La prima piega viene trattata come un set di validazione, e il metodo si adatta alle restanti pieghe $k-1$.

La suddivisione dei dati in *fold* può essere regolata da criteri quali la garanzia che ogni piega abbia la stessa proporzione di osservazioni con un dato valore categorico, come il valore di risultato della classe. Questa è chiamata **stratified cross validation** (=validazione incrociata stratificata).

A partire da un insieme di etichette associate ad immagini `imds.Labels` in un oggetto `ImageDatastore`, l'invocazione del comando

```
c = cvpartition(imds.Labels,'kFold',5);
```

costruisce un oggetto `c` della classe `cvpartition` che definisce una partizione casuale stratificata per la validazione incrociata k -fold. Ogni partizione ha all'incirca le stesse dimensioni e presenta elementi di ogni classe con le medesime proporzioni.

L'oggetto `c` creato per il dataset del presente lavoro (par. 4.4.3) ha, per esempio, i seguenti campi:

Type	'kFold'
NumTestSets	5

```
TrainSize      [12183,12182,12182,12182,12183]
TestSize       [3045,3046,3046,3046,3045]
NumObservations 15228
```

In questo modo si ottengono Fold 1 e Fold 5 composti da:

```
No pinna      2239
Pinna         806
```

e Fold 2, Fold 3 e Fold 4 costituiti da:

```
No pinna      2239
Pinna         807
```

Per costruire le partizioni è possibile utilizzare, per ogni caso i da 1 a $k = 5$, due vettori di indici logici

```
indici_training = c.training(i)
indici_test     = c.test(i)
```

che hanno la stessa dimensione del dataset `15228x1 logical` e hanno degli 1 come "segnaposto" degli elementi del dataset con cui costruire di volta in volta l' i -esima partizione di test e di training.

Parte III

Soluzioni proposte

Capitolo 4

Esperimenti e risultati

Sono state sviluppate due differenti routine Matlab, d'ora in avanti CropFin v1 e CropFin v2, che consentono, a partire da un dataset di immagini di delfini collezionate durante campagne di avvistamento in mare, di identificare e ritagliare automaticamente le pinne dorsali.

Complessivamente, il riconoscimento di una pinna all'interno dell'immagine è effettuato con i seguenti passaggi salienti, comuni ad entrambe le routine:

1. Segmentazione dell'immagine basata sui colori (par. 4.1)
2. Filtraggio e ritaglio adattivo delle regioni individuate sulla base di caratteristiche geometriche (par. 4.2 e 4.3)
3. Classificazione binaria: pinna/no pinna (4.4).

Ciò che differenzia le due routine è l'implementazione dei singoli passaggi:

- la routine CropFin v1 (par. 5.1) è stata sviluppata mutuando alcune idee già proposte in (Forenza, [4]) ed è stata utilizzata in (Seller, [5]) per la foto-identificazione automatica degli esemplari di *grampus griseus* nel golfo di Taranto.
- la routine CropFin v2 (par. 5.2) costituisce un tentativo di miglioramento di CropFin v1, ottenuto rivisitando completamente i metodi utilizzati in essa e proponendo un nuovo approccio per la fase iniziale di segmentazione. Rappresenta, pertanto, il risultato principale del lavoro di tesi, cui si è riservata maggior dedizione.

Nel seguito, si descrivono dettagliatamente i singoli passaggi, facendo riferimento alle tecniche di implementazione utilizzate ed alle routine cui appartengono. Si discutono, inoltre, di volta in volta, le motivazioni che hanno spinto ad introdurre variazioni in CropFin v2 rispetto alle soluzioni utilizzate in CropFin v1.

Il capitolo successivo 5 ha, invece, l'obiettivo di descrivere sinteticamente il funzionamento complessivo di entrambe le routine. In particolare, il paragrafo relativo alla routine CropFin v2 contiene il listato di alcuni codici descritti e citati nel capitolo corrente. È inoltre riportato l'indirizzo web al quale sono state pubblicate le routine.

4.1 Segmentazione basata sui colori

La segmentazione iniziale costituisce evidentemente il passaggio più importante e, quindi, di maggiore criticità all'interno dell'intera soluzione proposta. Sono stati sperimentati due metodi differenti, basati su:

- sogliatura automatica secondo il metodo Otsu
- applicazione di un filtro nello spazio dei colori

Si evidenzia che entrambi i metodi prevedono l'utilizzo dello spazio di colore CIE 1976 $L^*a^*b^*$, nel seguito abbreviato come Lab (par. 2.1.1). Tale preferenza, rispetto al consueto RGB, viene giustificata per ciascun metodo nelle seguenti descrizioni.

Si fa inoltre presente che, prima dell'elaborazione, le foto sono sottoposte a ridimensionamento (par. 2.2.1), al fine di ridurre i tempi di calcolo. A tal riguardo, la routine CropFin v2 introduce un miglioramento sostanziale rispetto alla routine CropFin v1, con effetti descritti in dettaglio alla fine del par. 4.2.

4.1.1 Soglia multipla secondo il metodo Otsu

Il metodo Otsu per la sogliatura automatica (par. 2.3.1) è utilizzato nella sua variante multilivello rispetto agli istogrammi dei canali L e b. Prima della sua applicazione, la foto è sottoposta ad una equalizzazione adattiva dell'istogramma a contrasto limitato (CLAHE), con l'obiettivo di migliorare il contrasto e le performance del metodo Otsu.

Se `im_lab` è l'immagine con codifica nello spazio Lab, la soglia `thr_L` per il canale L e la soglia `thr_b` per il canale b possono essere calcolate come

```
thr_L = multithresh(im_lab(:,:,1));  
thr_b = multithresh(im_lab(:,:,3));
```

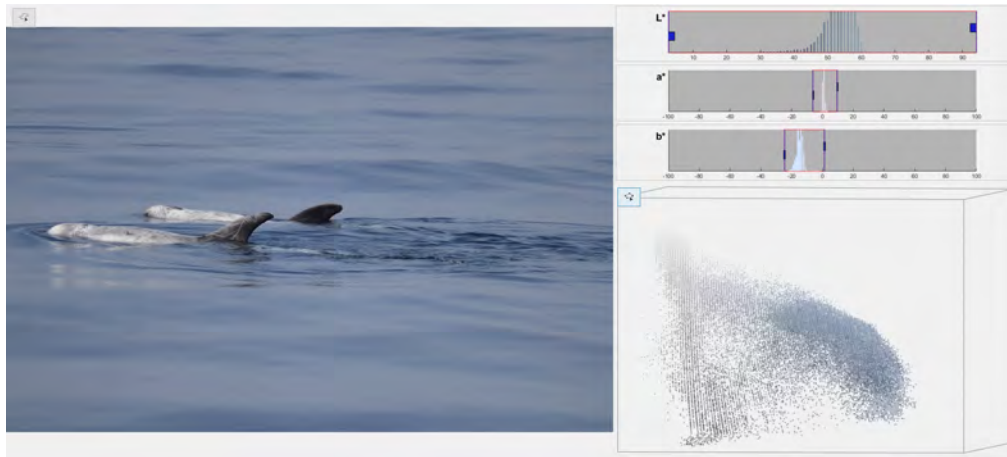
Si consideri, a titolo di esempio, l'immagine seguente (fig. 4.1), con i rispettivi istogrammi e occupazione nello spazio colore Lab, prima e dopo l'applicazione del CLAHE, ottenuti mediante l'applicazione *Color Threshold* di Matlab.

Per essa si ottengono i valori di soglia `thr_L = 52.4417` e `thr_b = -14.6364`, che dividono i rispettivi istogrammi in due parti (fig. 4.2).

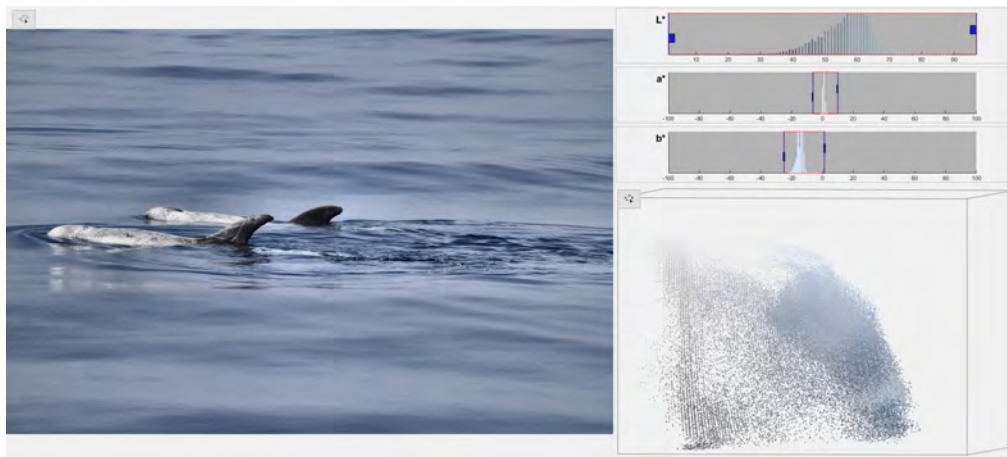
Avendo a disposizione tali soglie, l'ipotesi avanzata è che le pinne dorsali possono essere isolate considerando le regioni di immagine che siano contemporaneamente:

- nella regione più scura del canale L, cioè a sinistra di `thr_L`
- nella regione contenente il grigio del canale b, cioè a destra di `thr_b`

In Matlab è possibile applicare questo tipo di segmentazione e ottenere l'immagine binaria corrispondente attraverso i seguenti passaggi:

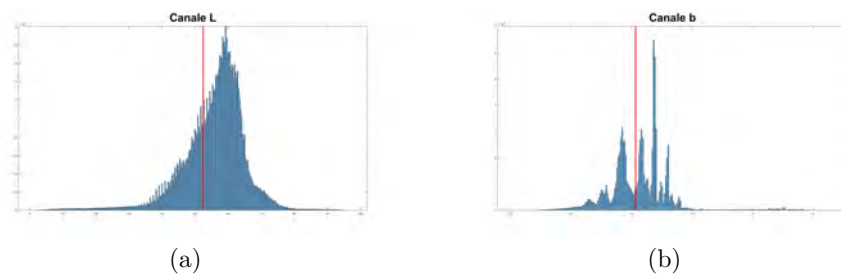


(a)



(b)

Figura 4.1: Istogrammi e spazio colore Lab: (a) dell'immagine originale e (b) dell'immagine sottoposta a CLAHE. Scatto del 7 giugno 2019, grampi nel golfo di Taranto, crediti Emanuele Seller



(a)

(b)

Figura 4.2: Soglia di Otsu calcolata sugli istogrammi L e b

1. Creazione di un'immagine bianca, della stessa dimensione dell'immagine di partenza

```
im_thr = im_lab(:,:,1)*0;
```

2. Annerimento dei pixel che non verificano le condizioni sopra dette

```
im_thr(im_lab(:,:,1) < thr_L & im_lab(:,:,3) > thr_b) = 255;
```

3. Binarizzazione dell'immagine ottenuta

```
imLog = logical(im_thr);
```

Il risultato è mostrato nella figura 4.3.

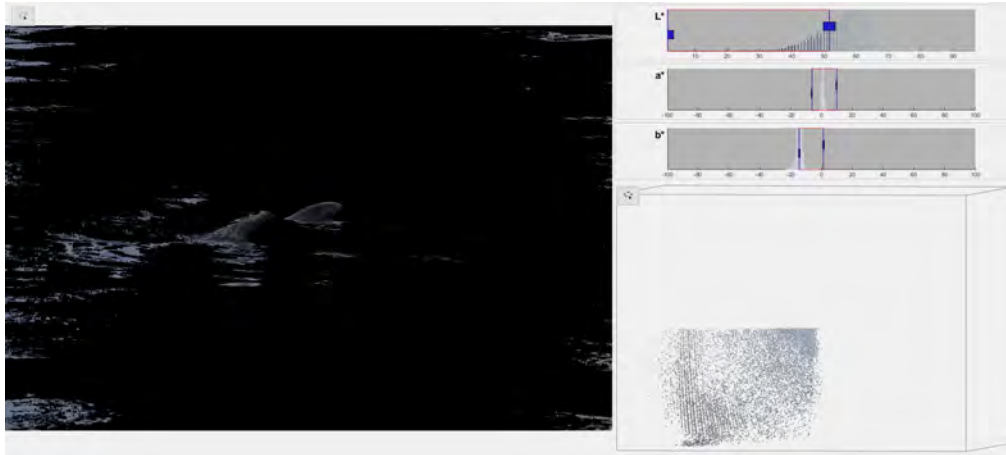
Il metodo appena descritto è implementato nella funzione `Threshold_LAB_L_b` della Routine v1 (par. 5.1).

L'idea alla base di questo approccio nasce da una precisa conoscenza del dominio e da alcune ipotesi a priori riguardanti il contenuto delle immagini. In particolare, si suppone che esse contengano generalmente solo mare (*background*) e cetacei (*foreground*), e che queste due classi di oggetti contribuiscano alla creazione di due aree distinte e separabili degli istogrammi dei canali L e b. Volendo dare un'interpretazione intuitiva, si tratta di separare ciò che è grigio e più scuro da ciò che è blu e più chiaro. La scelta dello spazio di colori Lab è motivata proprio dalla possibilità di automatizzare questo tipo intuitivo di segmentazione.

Sebbene l'idea possa sembrare semplicistica a primo impatto, la maggior parte delle immagini verifica le ipotesi suddette e, di conseguenza, si riesce ad ottenere un ottimo risultato di segmentazione. Per un'analisi quantitativa, si rimanda al confronto delle prestazioni delle due routine (par. 4.5). L'efficacia di questa tecnica richiede in primo luogo una condizione favorevole di scatto ed una certa accuratezza da parte di chi lo esegue.

La foto scelta a titolo di esempio costituisce certamente una conferma delle ipotesi. Nonostante ciò, si nota dal risultato di fig. 4.3 che l'immagine binaria risultante contiene una discreta quantità di "rumore". Il filtraggio delle regioni binarie descritto nel seguito (par. 4.2) contribuisce a migliorare notevolmente il risultato; tuttavia si nota, dalle analisi quantitative (par. 4.5), che la quantità di regioni che non contengono pinne risulta, su larga scala, di gran lunga superiore. Questo è ciò che primariamente motiva l'introduzione di una fase di classificazione finale, che consenta di automatizzare quasi completamente la procedura di *object detection*.

La soluzione originale, proposta in (Forenza, [4]), prevede, ancor prima dell'applicazione del CLAHE, una riduzione dei colori delle immagini a sole 32 tonalità. Questo passaggio, tuttavia, non provoca grosse differenze nel calcolo dei valori di soglia (nel caso dell'immagine di esempio sono identici), anzi talvolta peggiora lievemente il risultato



(a)



(b)

Figura 4.3: Segmentazione dell'immagine basata sulle soglie di Otsu per i canali L e b: (a) a colori, (b) binaria.

della segmentazione, senza fornire un reale guadagno in termini di tempo di calcolo. Quest'ulteriore operazione di *preprocessing* è stata, pertanto, rimossa in CropFin v1.

I principali limiti di questo metodo emergono quando il contenuto delle foto è diversificato rispetto alle ipotesi enunciate in precedenza, rendendo i valori di soglia inaffidabili per una corretta localizzazione dei cetacei. In particolare, si evidenzia che:

- nei casi in cui il mare abbia una tonalità prevalentemente grigia, l'istogramma del canale b risulterebbe concentrato sulle tonalità di grigio, rendendo le due classi difficilmente separabili (fig. 4.4).
- nei casi in cui l'immagine presenti, oltre alle due classi di oggetti previsti nell'ipotesi di lavoro del metodo (mare e cetacei), ulteriori elementi di dimensione



Figura 4.4: Esempio di inefficienza della sogliatura nel caso di mare con tonalità grigie: (a) immagine originale, (b) immagine binaria dopo la sogliatura. Scatto del 2018, grampi nel mare delle Azzorre.

considerevole (ad esempio uno sfondo paesaggistico od un'imbarcazione), essi potrebbero diversificare la composizione degli istogrammi rispetto a quella attesa (fig. 4.5).



Figura 4.5: Esempio di inefficienza della sogliatura in presenza di sfondo paesaggistico: (a) immagine originale, (b) immagine binaria dopo la sogliatura. Scatto del 2018, grampi nel mare delle Azzorre.

- nei casi in cui la pinna abbia un colore più chiaro e presenti un numero consistente di graffi, potrebbe essere caratterizzata da una zona predominante a luminanza elevata a causa del riflesso della luce; rientrerebbe, in tal caso, nella zona destra dell'istogramma L, esclusa in fase di sogliatura (fig. 4.6).

La tecnica innovativa di segmentazione introdotta in CropFin v2 e presentata nel paragrafo successivo costituisce un tentativo di miglioramento rispetto a quella appena presentata, a partire dai casi critici evidenziati, con l'obiettivo di rendere l'intera soluzione proposta maggiormente scalabile.



Figura 4.6: Esempio di inefficienza della sogliatura a causa del riflesso della luce: (a) immagine originale, (b) immagine binaria dopo la sogliatura. Scatto del 2016, grampi nel golfo di Taranto.

4.1.2 Poliedri nello spazio Lab

Il nuovo metodo introdotto in CropFin v2 per risolvere automaticamente il problema di segmentazione è basato sulla seguente idea: definendo, a priori, un insieme ristretto di tonalità di colore che permette di distinguere le pinne dorsali all'interno delle immagini, è possibile considerare come regioni di interesse quelle composte dai soli pixel che assumono tali tonalità.

Al fine di rendere quanto più generale e robusto questo metodo di segmentazione, si devono evidentemente considerare tutte le possibili condizioni di scatto che contribuiscono a diversificare le tonalità delle pinne all'interno delle immagini.

La numerosità e l'eterogeneità dei dati a disposizione hanno consentito, a tal proposito, di costruire un campione che includesse una quantità notevole di condizioni di scatto. A partire da circa 10.000 foto scattate tra il 2013 ed il 2018 nel golfo di Taranto e circa 14.000 foto scattate nel 2018 a largo delle isole Azzorre, sono state manualmente selezionate 1.033 immagini con condizioni diversificate (strategia di campionamento *purposive sampling*).

All'interno di questo campione, si è osservato che le principali cause di variabilità delle tonalità delle pinne sono le condizioni meteorologiche, l'orario dello scatto ed i riflessi dovuti alla luce o al paesaggio circostante. Attraverso numerosi esperimenti si è constatato che non è, in realtà, possibile individuare un unico insieme di triplette di colore che consenta di localizzare le pinne in maniera univoca. Capita, infatti, che il colore delle pinne in alcune immagini sia pressoché identico a quello del mare in altre immagini, con diverse condizioni di scatto. Si è reso, quindi, necessario scomporre il problema originario in diversi sottoproblemi.

Sono stati sviluppati cinque differenti modelli risolutivi, mostrati nella tabella 4.1. Ciascun modello è costituito da due insiemi di triplette nello spazio di colore Lab:

- il primo insieme identifica una tonalità predominante del mare.

- il secondo insieme identifica le possibili tonalità che consentono di localizzare in maniera univoca le pinne, limitatamente alle condizioni di scatto che derivano dalla tonalità del mare espressa dal primo insieme

Sulla base dei modelli individuati, la procedura di segmentazione automatica delle immagini è sviluppata in due fasi:

1. Stima della tonalità predominante del mare per l'individuazione del modello risolutivo di appartenenza
2. Applicazione del filtro corrispondente al modello individuato per la localizzazione delle pinne

L'implementazione di questa strategia risolutiva si trova nella funzione `fin_segmentation` della routine `CropFin v2` (sez. 5.2).

Si descrivono ora dettagliatamente le singole fasi e, in seguito, si discutono i criteri di scelta e le modalità di creazione dei modelli risolutivi utilizzati.

Per stimare la tonalità predominante del mare si calcola il numero di pixel che rientra nelle soglie Lab riportate nella tabella 4.2. Il modello che riporta il conteggio più numeroso è scelto come modello risolutivo di appartenenza. La validità di questo criterio si basa sull'ipotesi che il mare occupa sempre una porzione consistente delle immagini, quindi il conteggio dei pixel è sufficiente a fornire una stima approssimativa della sua tonalità predominante.

Se `lab` è l'immagine nel formato Lab, e `channelMin` e `channelMax` indicano le soglie dei tre canali di una determinata tonalità del mare, il conteggio dei pixel può essere effettuato creando una matrice logica che abbia degli 1 in corrispondenza dei pixel il cui valore rientra nelle soglie:

```
sliderBW = (lab(:,:,1) >= channel1Min ) & (lab(:,:,1) <= channel1Max) & ...  
(lab(:,:,2) >= channel2Min ) & (lab(:,:,2) <= channel2Max) & ...  
(lab(:,:,3) >= channel3Min ) & (lab(:,:,3) <= channel3Max);
```

e poi contando gli elementi non nulli di tale matrice con la funzione Matlab `nnz`:

```
tot = nnz(sliderBW);
```

Le operazioni appena presentate sono eseguite con le soglie di ciascun modello da funzioni omonime:

```
a = azzurro(lab);  
b_g = blu_grigio(lab);  
b_s = blu_scuero(lab);  
c_v = celeste_verde(lab);  
g = grigio(lab);
```

Gli esiti dei conteggi sono quindi disposti in un vettore, in modo tale che l'indice dell'elemento massimo possa identificare univocamente il modello risolutivo.

```
esiti_conteggio = [a b_g b_s c_v g];  
[~,sea_idx] = max(esiti_conteggio);
```

Ciò è utile per effettuare un accesso indicizzato alla struttura dati denominata **poliedri** in cui sono memorizzati i filtri (file **Lab_filters.mat** nel repository online) e selezionare, tra tutti, il filtro corrispondente al modello risolutivo individuato:

```
poliedri{1,sea_idx}
```

La procedura appena descritta è implementata nella funzione **get_sea_color_index** della routine CropFin v2 (par. 5.2).

A titolo di esempio, si consideri che l'immagine di figura 4.7(a) (di dimensione 1200×800), produce il seguente vettore

```
[0 878743 4335 787667 10]
```

e viene, quindi, segmentata con il filtro corrispondente all'indice 2, cioè relativo al modello "Blu-Grigio".

Si evidenzia, a questo punto, che i filtri, essendo costituiti da un insieme di triplette di colori Lab, sono di fatto equivalenti a nuvole di punti nello spazio tridimensionale Lab. Pertanto creazione, modifica, applicazione e salvataggio dei filtri sono gestiti mediante oggetti di tipo **alphaShape**, disponibili in Matlab per la manipolazione di poliedri 3D a partire da punti dello spazio.

Se l'immagine **lab** ha dimensione $m \times n$, è possibile ottenere l'elenco di triplette di colore **tripl** dell'immagine disponendo i livelli L, a, b in colonna, uno affianco all'altro, mediante la funzione **reshape**:

```
tripl = reshape(lab,[m*n 3]);
```

Detto **shp** l'oggetto di tipo **alphaShape** che rappresenta il filtro, le triplette dell'immagine appena ottute possono essere filtrate mediante la funzione **inShape**

```
tf = inShape(shp,tripl);
```

ottenendo un vettore colonna logico **tf** che contiene il valore 1 in corrispondenza delle triplette contenute nel poliedro. L'immagine binaria risultante è ottenuta ridisponendo il vettore logico secondo la dimensione originaria:

```
BW = logical(reshape(tf,m,n));
```

La binarizzazione effettuata con i passaggi appena descritti è implementata nella funzione **ptCloud_LAB_mask** (par. 5.2). La figura 4.7 mostra un risultato di esempio. Si noti, in questo caso, il miglioramento del risultato di segmentazione rispetto a quello di figura 4.3.

I cinque filtri utilizzati, disponibili all'interno della struttura **poliedri** precedentemente citata, sono stati creati manualmente secondo una procedura iterativa ripetuta per ciascun modello risolutivo, di cui si illustrano nel seguito i passaggi:



Figura 4.7: Esempio di segmentazione ottenuta mediante l'applicazione del filtro nello spazio Lab: (a) immagine originale, (b) immagine binaria dopo la sogliatura. Scatto del 2018, grampi nel golfo di Taranto.







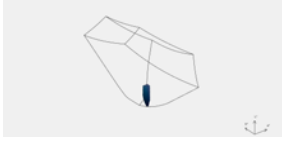








ID	Nome del modello	Tonalità del mare	Filtro corrispondente	Esempio
1	Azzurro			
2	Blu-Grigio			
3	Blu Scuro			
4	Celeste-Verde			
5	Grigio			

Tabella 4.1: Modelli individuati per la soluzione al problema di segmentazione, a partire dal campione di dati selezionato

Nome del modello	L	a	b	Percentuale sul campione
Azzurro	[25; 60]	[−14; 20]	[−60; −40]	7 %
Blu-Grigio	[40; 65]	[−8; 2]	[−18; 3]	28 %
Blu Scuro	[1; 30]	[−4; 8]	[−28; −18]	3 %
Celeste-Verde	[50; 71]	[−31; 7]	[−40; −9]	56 %
Grigio	[10; 65]	[−3; 0]	[−3; 0]	6 %

Tabella 4.2: Codifica delle tonalità predominanti del mare utilizzate per la scomposizione del campione di dati selezionato. L'ultima colonna riporta la percentuale di immagini corrispondente rispetto al campione (di dimensione 1033).

1. Creazione di una cartella contenente le foto del campione che presentano la stessa tonalità del mare. L'appartenenza alla cartella è determinata attraverso la strategia a conteggio dei pixel descritta in precedenza (funzione `get_sea_color_index`)
2. Individuazione di una foto, ad esempio caratterizzata da tonalità di interesse delle pinne o da condizioni di scatto particolari, a partire da cui creare (o ampliare) il filtro. La fig. 4.8(a) mostra l'immagine iniziale scelta per il modello "Grigio".
3. Annerimento dei pixel corrispondenti alla tonalità del mare di riferimento (implementato utilizzando il medesimo accesso condizionale ad array mostrato in precedenza per la creazione della variabile `sliderBW`), come mostrato in fig.4.8(b)
4. Segmentazione manuale dei pixel rimanenti attraverso l'app *Color Threshold* di Matlab, individuando una regione binaria minima a garantire la localizzazione delle pinne presenti; salvataggio dell'immagine risultante a colori (fig. 4.8(c))
5. Creazione di un poliedro 3D nello spazio Lab contenente le tonalità individuate in fase di segmentazione. A partire dall'elenco `tripl` delle triplette Lab dell'immagine segmentata a colori (`maskedRGB` di dimensione $m \times n$)

```
tripl = reshape(maskedRGB, [m*n 3]);
```

si crea un nuovo elenco `triplets_list` in cui non compaiano triplette duplicate

```
triplets_list = unique(tripl, 'rows');
```

e che non contenga la tripletta $[0,0,0]$ (che rappresenta i pixel anneriti nelle fasi precedenti e che è sicuramente memorizzata alla prima riga a causa dell'applicazione della funzione `unique`)

```
triplets_list(1,:) = [];
```

Con la lista di triplette così ottenuta è possibile creare il corrispondente poliedro `shp`:

```
shp = alphaShape(triplets_list);
```

Il poliedro così creato può essere visualizzato con il comando `plot(shp)` o, alternativamente, mediante la funzione `colorcloud` (fig.4.8(d)).

6. Test del filtro `shp` appena creato su tutte le immagini della cartella di partenza (applicando il processo di binarizzazione descritto in precedenza, funzione `ptCloud_LAB_mask`). Sulla base dei risultati ottenuti, si distinguono i seguenti casi:

- se si riescono a localizzare correttamente le pinne all'interno di tutte (o quasi) le immagini, allora la procedura è terminata ed il filtro `shp` può essere incluso nel modello risolutivo della tonalità del mare in esame;
- se si riscontrano diversi casi in cui la localizzazione delle pinne fallisce a causa di eccessivo "rumore" (cioè le regioni binarie includono molto altro rispetto alle pinne), si procede con lo scarto del poliedro `shp` e la ripetizione della procedura dal passo 2;
- se si riscontrano immagini in cui la localizzazione delle pinne non è avvenuta (cioè non è stata trovata alcuna regione binaria corrispondente), si sceglie una tra queste e si procede con l'ampliamento del poliedro `shp`, ripetendo la procedura dal passo 2. In tal caso, il passo 5 si modifica come segue. Se `trip1` è l'elenco di triplette Lab della nuova immagine segmentata, si verifica quali triplette siano già contenute nel poliedro `shp`:

```
tf = inShape(shp,trip1);
```

Il vettore di nuove triplette `new_triplets` che, eventualmente, contribuisce ad aumentare il poliedro `shp` è ottenuto azzerando le righe segnalate con un 1 dal vettore logico `tf`

```
new_triplets = t1;  
new_triplets(repmat(tf,[1 3])) = 0;
```

e, successivamente, rimuovendo le triplette duplicate e la prima riga nulla (come già descritto nella fase 5). Quindi, l'operazione vera e propria di ingrandimento del poliedro si esegue concatenando il vettore delle triplette Lab già contenute in `shp`

```
old_triplets = shp.Points;
```

con il vettore delle nuove triplette `new_triplets` mediante la funzione `cat`:

```
up = cat(1,old_triplets,new_triplets);
```

A partire dal nuovo vettore ottenuto è possibile creare un nuovo oggetto `alphaShape` che costituisce il poliedro aumentato ed è utilizzabile per la fase di test successiva:

```
augmented_shp = alphaShape(up);
```

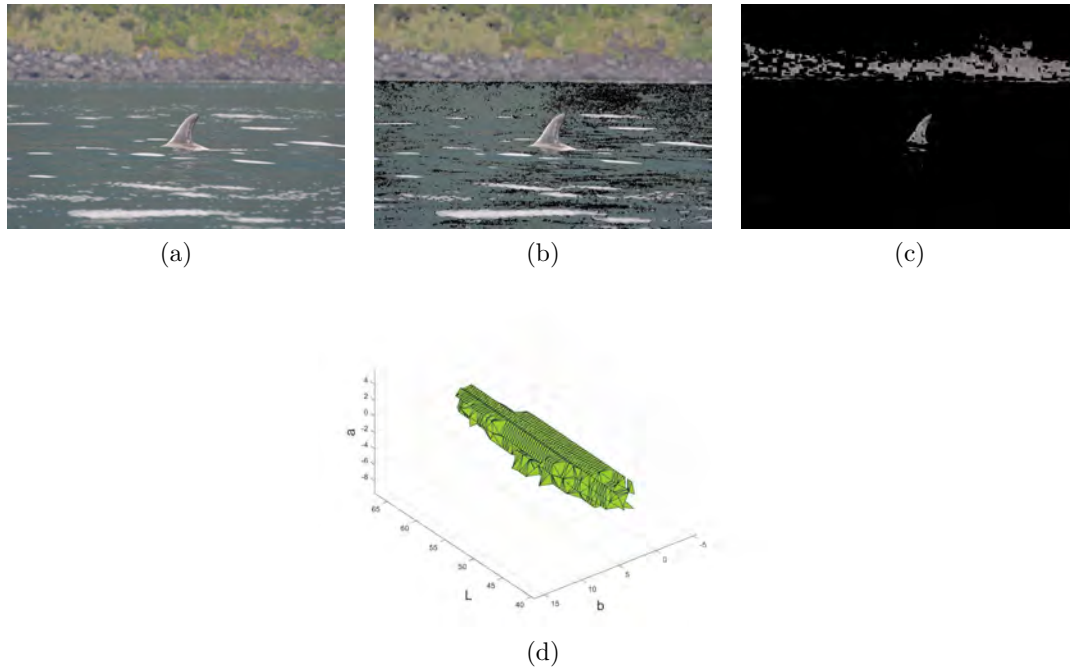


Figura 4.8: Esempio di creazione del filtro per il modello "Grigio": (a) immagine iniziale, (b) annerimento dei pixel corrispondenti alla tonalità del mare "Grigio", (c) risultato della segmentazione manuale mediante app *Color Thresholder*, (d) poliedro 3D corrispondente alle tonalità individuate

Alla sezione 6.2 è riportato un link in cui sono disponibili le funzioni e i file Matlab utilizzati come supporto alla procedura iterativa appena illustrata.

Si evidenzia, inoltre, che in seguito alla creazione dei cinque filtri secondo i passaggi appena descritti, è stato effettuato un "raffinamento" della procedura, introducendo un ulteriore test finale. Dal momento che la strategia di campionamento utilizzata in fase preliminare per la costruzione del dataset è stata di tipo *purposive sampling*, si è ritenuto opportuno testare i filtri su un nuovo campione costruito con strategia *random sampling*.

Sono state selezionate in maniera casuale 1000 nuove immagini, 500 dagli scatti del golfo di Taranto e 500 dagli scatti delle isole Azzorre. Per alcuni modelli, sono state effettuate alcune ulteriori iterazioni della procedura per ampliare il filtro esistente e contemplare nuovi casi emersi.

In conclusione, si precisa che:

- la creazione dei filtri è stata provata anche nello spazio di colore RGB secondo la medesima procedura iterativa. Tuttavia, nel caso di immagini con mare a tonalità grigie, il metodo fallisce poichè i tentativi di ampliamento del filtro devono essere continuamente scartati senza aver raggiunto l'obiettivo desiderato. Con lo spazio

di colore Lab, invece, questa difficoltà non viene riscontrata e, in tutti gli altri casi, presenta prestazioni pressochè identiche ai filtri RGB. Ciò giustifica la scelta dello spazio di colore.

- le tonalità del mare specificate nei cinque modelli risolutivi sono state individuate non in maniera aprioristica ma contestualmente alla creazione dei filtri, in modo da garantire, in tutti i casi, un'elevata performance di segmentazione.

4.2 Filtraggio di regioni connesse

L'immagine binaria ottenuta in seguito alla segmentazione (par. 4.1) viene filtrata in modo che siano scartate le regioni binarie che non presentano caratteristiche tali da poter rappresentare, verosimilmente, una pinna dorsale.

Si precisa che nel seguito si utilizzeranno in maniera intercambiabile i termini "regioni binarie" e "blob", con il significato di regioni logiche connesse secondo la *8-connectivity*, come spiegato nel par. 2.3.1. Si faccia riferimento al medesimo paragrafo per i dettagli riguardanti la gestione e le proprietà ad esse relative e per le tecniche di implementazione disponibili in Matlab.

Si descrive ora la strategia di filtraggio utilizzata nella routine CropFin v2, implementata nella funzione `blob_filter` (par. 5.2).

Si supponga che `BW` sia l'immagine segmentata binaria e `BW_out` sia il risultato del filtraggio. Le regioni binarie sono ottenute come

```
cc = bwconncomp(BW);
```

di cui si calcolano i parametri `Area` e `Orientation` come segue:

```
stats = regionprops(cc, 'Area','Orientation');
```

Il processo di filtraggio consta dei seguenti passaggi:

1. Applicazione del filtro "sale e pepe" (par. 2.3.1) per una riduzione del "rumore"

```
BW_out = medfilt2(BW);
```

2. Riempimento degli *holes* (par. 2.3.1):

```
BW_out = imfill(BW_out,'holes');
```

Ciò è utile principalmente per un aumento del parametro `Area` delle regioni binarie con pinne, verosimilmente con diversi *holes* causati dai graffi, le cui tonalità sono escluse in fase di segmentazione.

3. Selezione delle regioni con area maggiore di 800 pixel

```
idx = find([stats.Area] > 800);  
BW_af = ismember(labelmatrix(cc), idx);
```

4. Stima delle performance di segmentazione attraverso il numero totale n di regioni individuate

```
n = cc.NumObjects;
```

- se le regioni binarie individuate sono, in totale, inferiori a 100, allora si suppone che il risultato della segmentazione sia affetto da "poco rumore", quindi si mantengono semplicemente le 8 regioni con area più grande

```
BW_out = bwareafilt(BW_af, 8);
```

- se le regioni binarie individuate sono, in totale, superiori a 100, allora si individuano nell'immagine binaria BW_o1 le regioni che hanno il parametro Orientation minore di -0.7 gradi

```
idx = find([stats.Orientation] <= -0.7);  
BW_o1 = ismember(labelmatrix(cc), idx);
```

e in BW_o2 maggiore di 0.7 gradi

```
idx = find([stats.Orientation] >= 0.7);  
BW_o2 = ismember(labelmatrix(cc), idx);
```

Si presuppone che quelle escluse siano porzioni di immagini uniformi contenenti sfondo o cielo, che tipicamente presentano un'orientazione quasi nulla; le pinne, invece, a causa della loro forma e posizione, presentano spesso un'orientazione elevata. Quindi, di queste, si mantengono quelle individuate nella fase precedente in BW_af con area maggiore di 800 pixel. Ciò è implementato attraverso le condizioni logiche

```
BW_out = or(BW_o1, BW_o2);  
BW_out = and(BW_out, BW_af);
```

Infine, si mantengono le 16 regioni con area più grande, cioè il doppio del caso precedente, poichè in questo caso è possibile che ci siano numerose regioni binarie "rumorose" con area superiore rispetto a quelle con pinne, quindi il filtro sull'area deve essere meno selettivo:

```
BW_out = bwareafilt(BW_out, 16);
```

La scelta di tali parametri è stata effettuata osservando i risultati del filtraggio rispetto ad un insieme di foto contenenti pinne a diverse scale (medesimo dataset utilizzato per la creazione dei filtri Lab, descritto nel par. 4.1.2) in modo che fossero preservati tutti i ritagli con dimensione sufficiente a garantire, in media, un livello accettabile di definizione delle pinne.

L'applicazione del filtraggio appena descritto, consecutivamente al risultato di fig. 4.7,

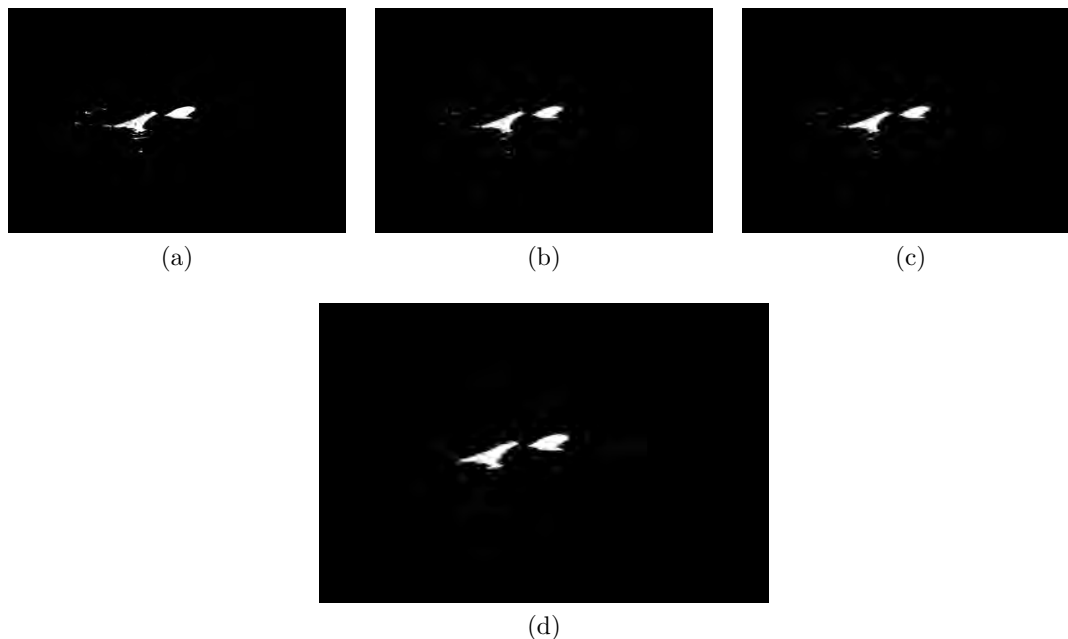


Figura 4.9: Esempio di filtraggio eseguito nella Routine v2: (a) risultato della segmentazione mediante filtro nello spazio Lab, (b) filtro sale e pepe, (c) riempimento degli *holes*, (d) risultato finale: mantenimento delle regioni con area più grande

è mostrato in fig. 4.9.

La figura 4.10 mostra, invece, i risultati della segmentazione e del filtraggio ottenuti applicando i nuovi metodi introdotti in CropFin v2 rispetto alle immagini in cui la sogliatura di Otsu utilizzata in CropFin v1 risulta fallimentare (figg. 4.4, 4.5, 4.6).

Nella routine CropFin v1 si adotta, invece, una strategia diversa di filtraggio, realizzata in due fasi distinte e mutuata da (Forenza [4]), la quale prevede la creazione di due filtri con scopi differenti:

1. Il primo filtro è applicato all'intera foto per migliorare il risultato della sogliatura iniziale basata sul metodo Otsu; esso è configurato per mantenere, nell'ordine, le regioni connesse con proprietà:
 - prime 15 in ordine decrescente di area
 - 'Area', [1600, 40000]
 - 'Extent', [-Inf, 0.55]

Il risultato, seguendo l'evoluzione della foto di fig. 4.3, è mostrato in fig. 4.11(a), in cui si possono distinguere cinque regioni che hanno superato il filtro.

2. Il secondo filtro è impiegato come segue:

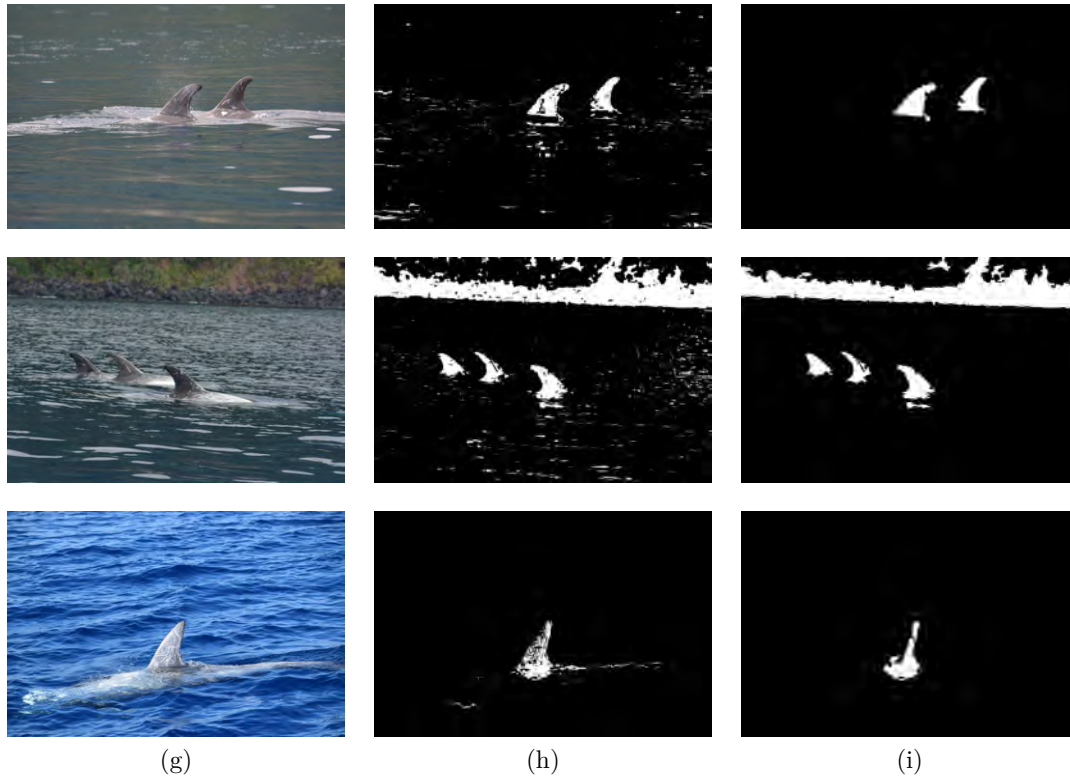


Figura 4.10: Esempio di filtraggio eseguito nella Routine v2: (a) risultato della segmentazione mediante filtro nello spazio Lab, (b) filtro sale e pepe, (c) riempimento degli *holes*, (d) risultato finale: mantenimento delle regioni con area più grande

- (a) Si ritaglia la foto originale in corrispondenza delle regioni mantenute in seguito all'applicazione del primo filtro, sulla base delle coordinate dei bounding box. Per ottenere ritagli leggermente più larghi rispetto ai blob, ogni dimensione è aumentata del 20%.
- (b) Si applica nuovamente, a ciascun ritaglio, la sogliatura basata sul metodo Otsu (par. 4.1.1). In questo caso è omesso il miglioramento del contrasto mediante CLAHE prima del calcolo dei valori di soglia.
- (c) Si introduce a questo punto il secondo filtro, applicato alle regioni binarie ottenute per ciascun ritaglio. L'unico parametro utilizzato in questo caso è il seguente:

- 'Area', [20000, 1000000]

con lo scopo di isolare l'eventuale pinna (che rappresenta sicuramente la regione di area maggiore all'interno di ciascun ritaglio) in modo che possa essere sottoposta all'algoritmo di ritaglio adattivo (par. 4.3). Ulteriore scopo è quello di scartare i ritagli che non presentino regioni binarie siffatte: questi

risultano completamente anneriti all'uscita del filtro poichè, presumibilmente, non contengono alcuna pinna. Il risultato è mostrato, rispetto a tutte le cinque regioni estratte dal primo filtro, nella fig. 4.11. In questo caso, ad esempio, si nota che il primo e l'ultimo ritaglio possono essere correttamente scartati.

L'implementazione dei filtri si trova rispettivamente nelle funzioni `Properties_Fins_Extraction` (par. 5.1) e `filterRegionsCropped2` (par.5.1) della Routine v1.

Si evidenziano, di seguito, le motivazioni che hanno portato allo sviluppo della tecnica di filtraggio a fase singola nella Routine v2 in sostituzione della strategia a due fasi prevista nella Routine v1:

- Sullo stesso campione di test, si è notato che i limiti previsti per l'area del primo filtro nella strategia a due fasi ([1600, 40000]) risultano penalizzanti sia per pinne con dimensione piccola ma con qualità accettabile, sia per pinne che occupano gran parte della foto (scatti ravvicinati). La tolleranza di regioni con area relativamente piccola nella Routine v2 (>800) resta, tra l'altro, giustificata dalla maggiore efficienza della segmentazione ottenuta con i filtri nello spazio Lab.
- Il parametro `Orientation` (utilizzato nel filtro della strategia a fase singola) si rivela talvolta più efficiente per lo scarto di regioni "chiaramente rumorose" rispetto al parametro `Extent` (utilizzato nel primo filtro della strategia a due fasi).
- La strategia a singola fase permette di eseguire direttamente il ritaglio adattivo di ciascuna regione, evitando l'applicazione multipla della sogliatura e l'eventuale calcolo doppio di coordinate per ciascun ritaglio, semplificando quindi l'elaborazione complessiva.

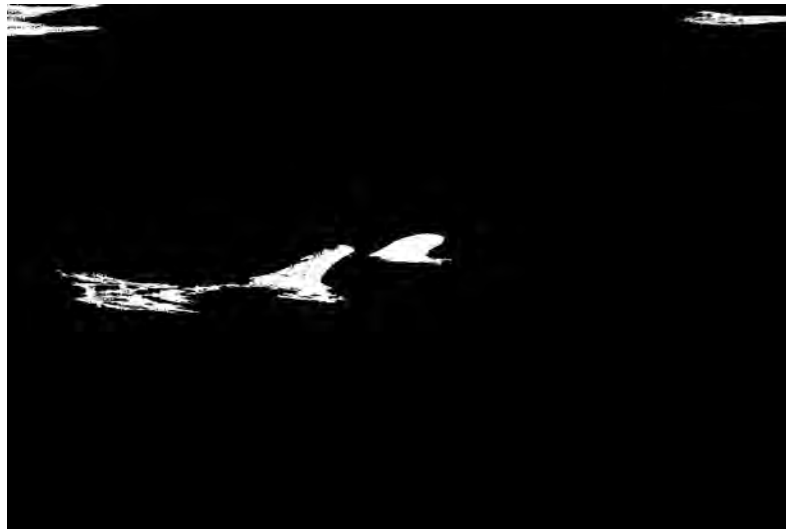
In entrambe le soluzioni enunciate, si deve notare, in ogni caso, che il filtro di maggior importanza resta pur sempre quello legato all'area delle regioni connesse, mentre il secondo criterio filtrante (sia esso attraverso il parametro `Orientation` o il parametro `Extent`) comporta solo un lieve margine di miglioramento rispetto all'effetto del primo.

Si vuole, infine, porre in evidenza un miglioramento introdotto in CropFin v2 che ha ripercussioni sostanziali in questa fase e, di conseguenza, nei risultati finali.

Le dimensioni in pixel riferite alla proprietà `Area` nei filtri precedentemente descritti sono riferiti ad una precisa dimensione dell'immagine: 1200x800.

In CropFin v1 si presuppone che le immagini iniziali abbiano risoluzione nativa 6000x4000 (standard condiviso da tutte le foto a disposizione). Applicando, prima della segmentazione, un ridimensionamento con fattore di scala 0.2 si ottiene il formato 1200x800. Non sono però contemplate altre risoluzioni iniziali: anche immagini con risoluzione nativa 1200x800 non sarebbero correttamente processate poichè subirebbero uno scarto delle regioni binarie durante la fase di filtraggio.

Il limite sulle dimensioni iniziali è stato rimosso in CropFin v2, applicando a tutte le



(a)



(n)

(o)

(p)

Figura 4.11: Filtraggio delle regioni connesse a due fasi: (a) applicazione del primo filtro sull'immagine intera a seguito della sogliatura con metodo Otsu, (n) estrazione di una regione dall'immagine originale, (o) applicazione ripetuta della sogliatura (senza CLAHE) , (p) applicazione del secondo filtro sul singolo ritaglio.

immagini un ridimensionamento che specifica la risoluzione target 1200x800. Per l'operazione di ritaglio successiva, le coordinate sono coerentemente scalate rispetto alla risoluzione originale (come spiegato nel par. 4.3). Ciò attribuisce alla nuova routine

piena compatibilità rispetto a diverse risoluzioni delle immagini iniziali; resta, tuttavia, appannaggio dell'utente assicurarsi che la qualità delle immagini originali sia sufficiente per ottenere ritagli utili allo scopo della routine.

4.3 Ritaglio adattivo

Le regioni binarie mantenute in seguito alla fase di filtraggio (4.2) sono sottoposte ad un algoritmo che consente di ottenere un ritaglio preciso in corrispondenza delle pinne. Tale operazione si può definire "adattiva" nella misura in cui la regione di ritaglio è ottenuta a partire da precisi punti geometrici calcolati per ciascuna regione binaria.

Si descrive, nel seguito, la procedura implementata nella funzione `adaptive_cropping` della Routine v2 (5.2).

Obiettivo della procedura è creare un vettore `rect` contenente le coordinate del rettangolo da utilizzare come argomento della funzione Matlab `imcrop` per il ritaglio all'interno dell'immagine originale:

`[w h Width Height]`

in cui `(w,h)` sono le coordinate del punto in alto a sinistra del rettangolo e `Width` e `Height` ne rappresentano le dimensioni.

In breve, si pone:

- `Height` pari all'altezza del centroide (relativa alla regione binaria).
- `h` poco più su del punto più in alto della regione binaria
- `Width` leggermente più ampia della distanza tra i punti estremi a sinistra e a destra della regione binaria (presi entro l'altezza `Height`)
- `w` poco più a sinistra del punto all'estrema sinistra

Le scelte elencate sono state effettuate sulla base di numerosi esperimenti effettuati su un campione di $n = 153$ immagini scattate nel golfo di Taranto nel 2019 e sottoposte alle procedure di segmentazione e filtraggio previste in CropFin v2.

La scelta del parametro `Height` è quella di maggiore criticità poichè, ai fini di un corretto calcolo dei descrittori previsto per la foto-identificazione automatica all'interno della pipeline innovativa proposta in (Seller, [5]), risulta di grande importanza riuscire a separare, eventualmente, la pinna sia dal mare sia dal dorso al di sotto di essa.

Rispetto alle regioni binarie con pinne ottenute dal campione considerato, si è osservato che la porzione del blob superiore al centroide contiene sempre la zona di interesse e, nella maggior parte dei casi, permette di escludere anche le zone indesiderate. La scelta di `Height` è dunque motivata dalla sua semplicità ed efficienza.

Una volta stabilita l'altezza di taglio `Height`, è sufficiente considerare i punti estremi

nella porzione del blob al di sopra di essa per assicurarsi di includere interamente la regione. Infine, per evitare di tagliare i contorni delle pinne, in genere non precisamente definiti in seguito alla segmentazione a causa della presenza dei graffi, si considera un distacco dai bordi della regione di una quantità limitata: 10% dell'altezza dal bordo superiore e 10% della larghezza dai bordi sinistro e destro.

Si osserva che il campione considerato, sebbene a numerosità limitata, è stato ritenuto rappresentativo della popolazione ai fini delle valutazioni effettuate. I risultati della routine CropFin v2 hanno confermato l'ipotesi (par. 4.5).

Si descrivono ora i dettagli implementativi. Si utilizzano le coordinate del bounding box e del centroide delle regioni binarie. Anzitutto, come illustrato nel par. 2.3.1, si istanzia un oggetto di tipo `BlobAnalysis`

```
Hblob = vision.BlobAnalysis;
```

grazie al quale è possibile ottenere le coordinate cercate per l'immagine BW

```
[~,centroid,bbox] = Hblob(BW);
```

Quindi, si procede come segue:

1. Supponendo di considerare la regione *i*-esima all'interno dell'immagine, si ottengono le rispettive coordinate del bounding box come

```
rect_BW = bbox(i,:);
```

Attraverso esse si ritaglia la regione binaria `rit_BW` a partire dall'immagine BW

```
rit_BW = imcrop(BW,rect_BW);
```

Quindi si ottengono, con la funzione `find`, le coordinate (*w,h*) di tutti i punti della regione, nei due vettori *Ws* e *Hs*

```
[Ws, Hs] = find(rit_BW');
```

2. Al fine di porre `Height` pari all'altezza del centroide, si deve rendere la misura presente in `centroid(i,2)` relativa alla regione binaria, anziché all'intera immagine. Per far ciò si sottrae lo spiazzamento della regione, dato da `bbox(i,2)`:

```
Height =centroid(i,2)-bbox(i,2);
```

3. Per il calcolo di *h* si deve determinare anzitutto il punto più in alto del blob. Dal momento che la funzione `find` utilizzata in precedenza ha restituito tutti i punti della regione ispezionando per colonne l'immagine trasposta `rit_BW'`, l'altezza del punto più in alto è sicuramente il primo elemento del vettore *Hs*

```
hTop = Hs(1);
```

Successivamente, si calcola una percentuale **fph** del 10% rispetto all'altezza **Height** che possa essere usata per spostare più in alto la coordinata **h** in modo da includere interamente la pinna con un distacco generalmente sufficiente. Per eseguire la divisione si esegue una conversione a **double**, quindi si ritorna al formato di default delle coordinate **int32**:

```
fph = int32((double(Height*10))/100);
```

Dal momento che **hTop** è una coordinata relativa alla regione binaria, in questo caso si deve, oltre a sottrarre la percentuale **fph** calcolata, aggiungere lo spiazzamento della regione **bbox(i,2)**. Inoltre, al fine di evitare coordinate negative, prive di significato, si soglia il risultato a zero mediante la funzione **max**, ottenendo:

```
h = max([0 bbox(i,2)+hTop-fph]);
```

4. Per il calcolo di **Width** si devono determinare i punti estremi al di sopra dell'altezza di ritaglio **hMax** relativa alla regione. Essa è calcolata come

```
hMax = hTop-fph+Height;
```

Tutti i punti del blob al di sopra dell'altezza **hMax** sono determinati mediante condizione logica applicata ai vettori **Hs** e **Ws** precedentemente calcolati:

```
Hs = Hs(Hs<=hMax);  
Ws = Ws(1:length(Hs));
```

A partire da questi, le coordinate di estrema sinistra e di estrema destra si ottengono, rispettivamente, come:

```
wTopLeft = min(Ws);  
wTopRight = max(Ws);
```

Detta **dw** la loro distanza

```
dw = wTopRight - wTopLeft;
```

si pone **Width** pari a **dw** più un 20%, in modo da includere sufficientemente le parti laterali delle pinne:

```
tpw =int32((double(dw*10))/100);  
Width = dw + 2*tpw;
```

5. Per distribuire equamente la percentuale aggiuntiva del 20% a sinistra e a destra, si pone la coordinata **w** del 10% più a sinistra del punto estremo **wTopLeft**. Anche in questo caso, si deve evitare un risultato negativo e si deve aggiungere lo spiazzamento della regione binaria al fine di ottenere una coordinata relativa all'immagine:

```
w = max([0 bbox(i,1)+ wTopLeft - tpw]);
```

6. Avendo a disposizione le quattro coordinate del rettangolo è possibile eseguire il ritaglio dall'immagine originale. Dal momento che le dimensioni dell'immagine binaria utilizzata per il calcolo delle coordinate (1200x800) sono frutto di un ridimensionamento, è necessario applicare un'ulteriore operazione per rendere le coordinate compatibili con l'immagine originale. Se la risoluzione nativa è $m \times n$ (altezza x larghezza), è sufficiente moltiplicare w e $Width$ per il fattore di scala $n/1200$ e h e $Height$ per il fattore di scala $m/800$.

La figura 4.12 mostra il risultato del ritaglio adattivo appena descritto rispetto alle regioni binarie di figura 4.9.

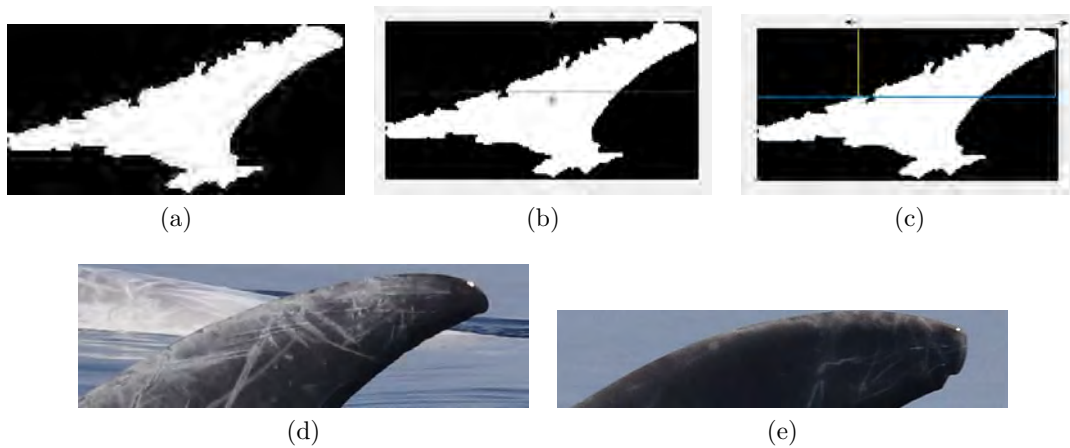


Figura 4.12: Esempio di ritaglio adattivo eseguito nella Routine v2: (a) immagine di partenza (b) calcolo dell'altezza di taglio (linea blu) e della coordinata h , (c) calcolo dell'ampiezza di taglio, (d) risultato finale, (e) risultato finale relativo all'altra pinna localizzata all'interno dell'immagine

Nella routine CropFin v1 è invece impiegata una diversa strategia di ritaglio adattivo, basata sulla proposta di (Forenza, [4]). Si evidenzia sin da subito che il funzionamento ed i risultati di entrambe le soluzioni presentate sono molto simili. L'algoritmo introdotto in CropFin v2 è un semplice tentativo di "razionalizzazione" di quello presente in CropFin v1, con lo scopo principale di rendere più semplice ed intuitivo il procedimento seguito.

Si passa, quindi, ad una breve descrizione della procedura utilizzata in CropFin v1. La figura successiva 4.13 illustra i singoli passaggi seguendo l'evoluzione della precedente fig. 4.11:

1. Si sottopone la regione binaria al riempimento dei cosiddetti *holes* (par. 2.3.1)

2. Si individuano quattro punti di interesse: nell'ordine punto più in alto, punto medio tra questo ed il centroide, punti di estrema sinistra e destra all'altezza del punto medio.
3. Si identifica un rettangolo che racchiuda i punti precedentemente trovati
4. Si trasla e si estende il rettangolo trovato in modo che contenga l'intera pinna, a seconda della sua orientazione.

4.4 Classificazione

I ritagli ottenuti in seguito all'applicazione di uno degli algoritmi descritti al paragrafo precedente (4.3) sono sottoposti ad una classificazione binaria che consenta di discernere quelli che effettivamente contengono una pinna dorsale (classe *Pinna*) da quelli che presentano qualsiasi altro tipo di contenuto (classe *No Pinna*).

I classificatori utilizzati, in entrambe le CropFin v1 e v2, sono delle reti neurali convoluzionali (par. 3.4) progettate da zero. Nei paragrafi successivi, si descrivono i dettagli della progettazione: creazione del dataset (par. 4.4.1), architettura delle reti (par. 4.4.2) e loro addestramento (par. 4.4.3).

Il codice utilizzato è contenuto nello script `reti_cnn` (sez. 6.2).

4.4.1 Creazione del dataset

Il dataset è stato costruito a partire dagli scatti collezionati nel Golfo di Taranto durante campagne di avvistamento effettuate dall'associazione di ricerca *Jonian Dolphin Conservation* (per maggiori informazioni si veda il cap. 1). In totale, erano a disposizione:

- 7034 foto di grampi scattate tra luglio 2013 e agosto 2018
- 3159 foto di tursiopi scattate tra luglio 2016 e settembre 2017

Tutte le foto originali hanno dimensione 6000×4000 con una occupazione di memoria intorno ai 10 MB, per un totale di 10194 foto e circa 90 GB.

Ognuna delle 10194 foto ha subito lo stesso processo di segmentazione, filtraggio e ritaglio adattivo previsto nella routine CropFin v1 (par. 5.1). I ritagli ottenuti sono stati manualmente selezionati, per la creazione di un dataset di dimensione $n = 15228$ immagini, di cui:

- 4033 ritagli della classe *Pinna*
- 11195 ritagli della classe *No Pinna*

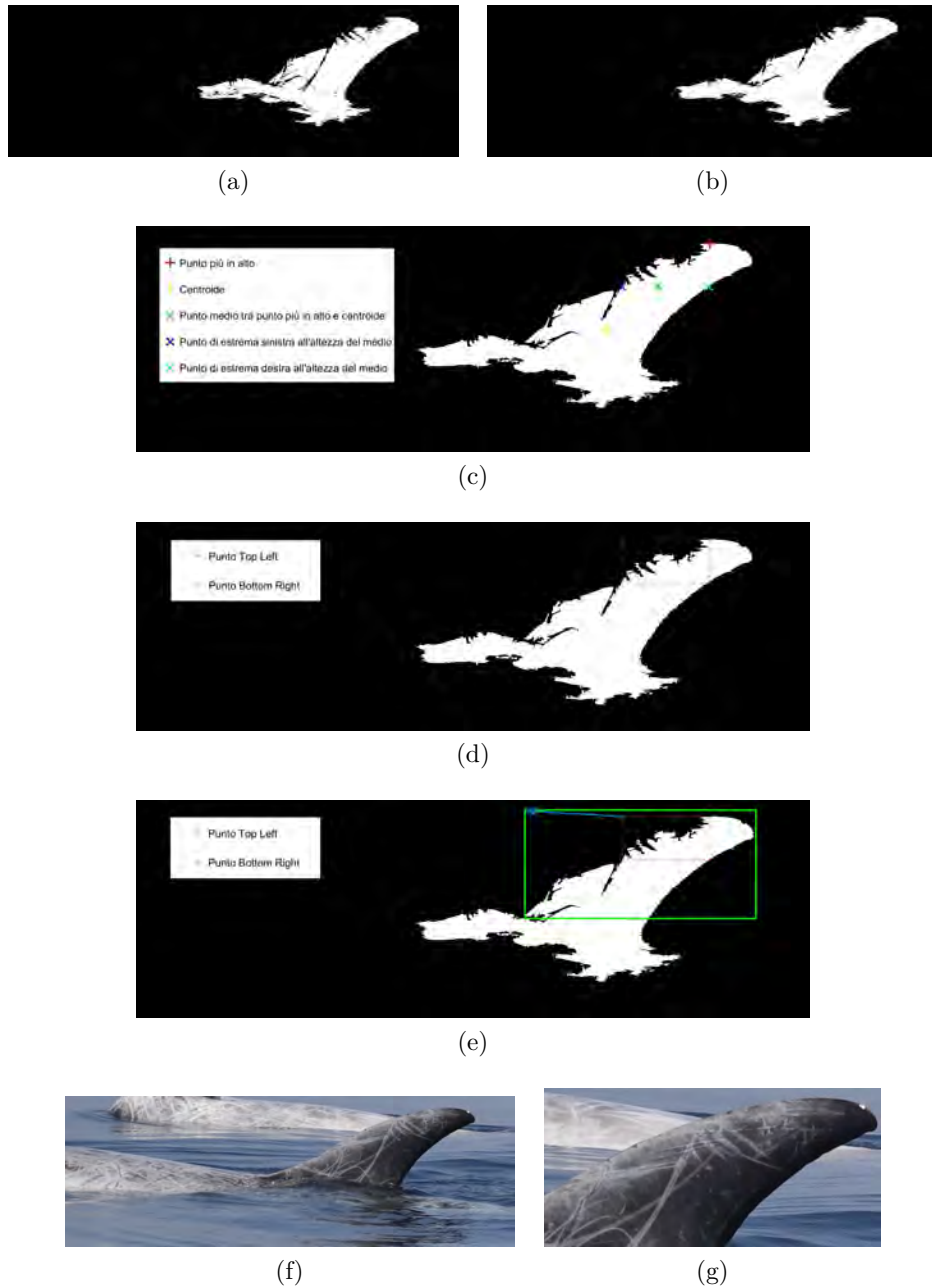


Figura 4.13: Illustrazione del ritaglio adattivo eseguito nella Routine v1: (a) regione binaria di partenza, (b) riempimento degli *holes*, (c) localizzazione dei punti di interesse, (d) identificazione del rettangolo che li contenga, (e) traslazione ed estensione del rettangolo, (f)-(g) ritaglio di partenza e risultato finale.

Si pone in evidenza che i ritagli della classe *No Pinna* così ottenuti sono l'esito "fallimentare" della procedura di segmentazione, filtraggio e ritaglio adattivo adottati in

CropFin v1. Si noti, a tal proposito, come la dimensione del dataset sia evidentemente sbilanciata in favore della classe *No Pinna*. Questa osservazione è ciò che primariamente ha motivato l'introduzione di un classificatore finale, che sia in grado di filtrare opportunamente i risultati ottenuti dalle fasi precedenti.

Si precisa che, nella fase di selezione manuale, sono stati attribuiti alla classe *Pinna* tutti e soli i ritagli contenenti una sola pinna in primo piano, intera o leggermente tagliata, escludendo quelli con pinne multiple e quelli con una presenza preponderante del dorso dei delfini. I ritagli con tali caratteristiche, infatti, sono considerati maggiormente affidabili ai fini del calcolo dei descrittori previsti per la foto-identificazione automatica (par. 1.2). Inoltre, questa scelta è stata anche motivata dall'intenzione di creare un "concetto univoco" utile a semplificare sia la selezione manuale sia l'apprendimento del classificatore.

La figura 4.14 mostra un campione del dataset.

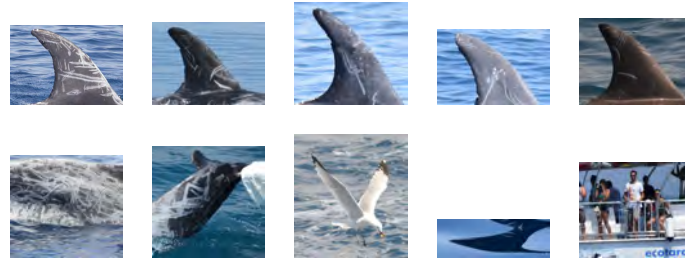


Figura 4.14: Campione del dataset: nella prima riga esempi di classe *Pinna*, nella seconda riga esempi di classe *No Pinna*

4.4.2 Architettura

L'architettura scelta per le reti neurali convoluzionali utilizzate è mostrata nella seguente fig. 4.15 come evoluzione dell'immagine attraverso diversi blocchi, ciascuno dei quali rappresenta un diverso stadio dell'elaborazione per l'estrazione di *feature* fino alla classificazione binaria finale. La struttura complessiva può essere così schematizzata:

```
INPUT --> [CONV -> RELU -> CONV -> RELU -> POOL] * 3 -->
[FC -> RELU] * 2 --> FC
```

dove CONV indica un layer di convoluzione, RELU un layer di attivazione con funzione ReLU, POOL un layer di pooling e FC un layer *fully-connected*.

Segue una descrizione dettagliata dell'architettura e, successivamente, una giustificazione di tale scelta:

- Primo blocco: due layer di convoluzione che preservano la dimensione di input 224×224 , con 8 filtri ciascuno di dimensione 3×3 e funzione di attivazione di tipo ReLU. Il risultato è sottoposto ad un layer di max pooling con dimensione 2 e *stride* 2, che dimezza le dimensioni dell'output.

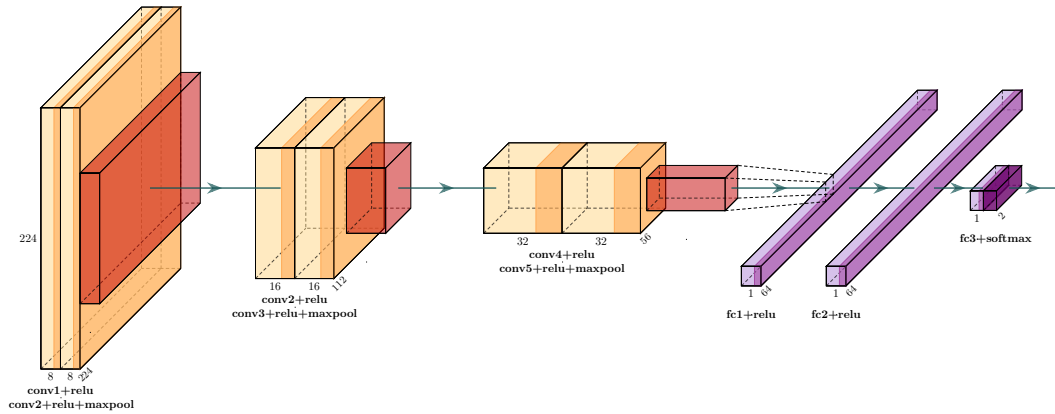


Figura 4.15: Architettura della rete neurale convoluzionale utilizzata in fase di classificazione. Nell'angolo in basso a destra di ciascun blocco è riportata, in obliquo, la misura delle dimensioni altezza e larghezza. Alla base, invece, sono riportati la misura della profondità di ciascun blocco ed i nomi identificativi dei layer. I blocchi giallo-arancio rappresentano gli strati di convoluzione e pooling, mentre quelli in viola rappresentano gli strati finali di classificazione. Rappresentazione ottenuta in L^AT_EX, grazie alla libreria disponibile al link <https://github.com/HarisIqbal88/PlotNeuralNet>

- Secondo blocco: due layer di convoluzione che preservano la dimensione di input 112×112 con 16 filtri di dimensione 3×3 e attivazione di tipo ReLU, seguiti da un layer di max pooling.
- Terzo blocco: due layer di convoluzione che preservano la dimensione di input 56×56 con 32 filtri di dimensione 3×3 e attivazione di tipo ReLU, seguiti da un layer di max pooling
- Quarto blocco e quinto blocco: layer *fully-connected* con uscita di dimensione 64 e attivazione di tipo ReLU
- Sesto blocco: layer di classificazione binaria finale con funzione di attivazione di tipo *softmax*.

Si riporta il codice della sua definizione in Matlab all'interno di un oggetto `layers` (seguendo la documentazione riportata al par. 3.4):

```
inputSize = [224 224 3];
numClasses = 2;

layers = [

imageInputLayer(inputSize)
```

```
convolution2dLayer(3,8,'Padding','same','Name','CONV1-8')
reluLayer

convolution2dLayer(3,8,'Padding','same','Name','CONV2-8')
reluLayer

maxPooling2dLayer(2,'Stride',2,'Name','MAXPOOL1')

convolution2dLayer(3,16,'Padding','same','Name','CONV3-16')
reluLayer

convolution2dLayer(3,16,'Padding','same','Name','CONV4-16')
reluLayer

maxPooling2dLayer(2,'Stride',2,'Name','MAXPOOL2')

convolution2dLayer(3,32,'Padding','same','Name','CONV5-32')
reluLayer

convolution2dLayer(3,32,'Padding','same','Name','CONV6-32')
reluLayer

maxPooling2dLayer(2,'Stride',2,'Name','MAXPOOL3')

fullyConnectedLayer(64)
reluLayer

fullyConnectedLayer(64)
reluLayer

fullyConnectedLayer(numClasses)
softmaxLayer

classificationLayer ];
```

La tabella 4.3 tiene traccia delle dimensioni in memoria, del numero di pesi e bias di ciascun blocco all'interno dell'architettura proposta. A partire da questi dati, è possibile calcolare

- l'occupazione di memoria richiesta da ogni immagine per un *forward pass* attraverso la rete, sommando le dimensioni di ciascun blocco e moltiplicando per 4 byte (tipo di dato `uint32` o `single` utilizzato da Matlab nei diversi layer):

$$1730 K * 4 \text{ byte} = 6920 \text{ KB} \simeq 7 \text{ MB}$$

- le dimensioni totali della rete, sommando il numero dei parametri e dei bias e moltiplicando per 4 byte

$$1628 K * 4 \text{ byte} = 6512 \text{ KB} \simeq 6.4 \text{ MB}$$

I risultati ottenuti sono significativamente inferiori rispetto ai dati che caratterizzano le architetture disponibili in letteratura (ad esempio 1 % e 7 % rispettivamente rispetto alla rete VGGNet [21]).

Nome blocco	Dimensioni	Memoria	Pesi	Bias
input	[224x224x3]	150K	0	0
conv1-8:	[224x224x8]	401K	$(3 * 3 * 3) * 8 = 216$	8
conv2-8:	[224x224x8]	401K	$(3 * 3 * 8) * 8 = 576$	8
maxpool1:	[112x112x8]	100K	0	0
conv3-16:	[112x112x16]	201K	$(3 * 3 * 8) * 16 = 1152$	16
conv4-16:	[112x112x16]	201K	$(3 * 3 * 16) * 16 = 2304$	16
maxpool2:	[56x56x16]	50K	0	0
conv5-32:	[56x56x32]	100K	$(3 * 3 * 16) * 32 = 4608$	32
conv6-32:	[56x56x32]	100K	$(3 * 3 * 32) * 32 = 9216$	32
maxpool3:	[28x28x32]	25K	0	0
fc1:	[1x1x64]	64	$(28 * 28 * 32) * 64 = 1606K$	64
fc2:	[1x1x64]	64	$64 * 64 = 4096$	64
fc3:	[1x1x2]	2	$64 * 2 = 128$	2

Tabella 4.3: Dimensioni e numero di parametri delle reti neurali convoluzionali utilizzate

Le scelte architettureali appena presentate sono frutto di un *trade-off* tra semplicità ed efficienza del classificatore.

In primo luogo, la struttura utilizzata prevede una precisa ripetizione di pattern, mutuata da un'idea di (Karpathy, [16]): la ripetizione dei layer di convoluzione consente di estrarre una quantità di feature sufficientemente significativa prima dell'operazione "distruttiva" del pooling attraverso filtri di dimensione 3×3 . Si noti che:

- ciascun neurone del primo blocco di convoluzione (conv1-8, conv2-8) è connesso ad una finestra 3×3 dell'immagine di input
- ciascun neurone del secondo blocco di convoluzione (conv3-16, conv4-16) è connesso ad una finestra 3×3 dell'output del primo blocco di max-pooling (maxpool1), quindi per estensione ad una finestra 5×5 rispetto all'immagine di input
- analogamente, ciascun neurone del terzo blocco di convoluzione (conv5-32, conv6-32) è connesso, per estensione, ad una finestra 7×7 dell'immagine di input, dimensione ragionevole per estrazione di feature non locali.

La preferenza di più layer con filtri a dimensione ridotta piuttosto che un unico layer con filtri di dimensione 7×7 è giustificata dalla possibilità di ottenere un'estrazione di feature maggiormente significativa con un minor numero di parametri. Infatti:

- l'introduzione di non-linearità nei tre differenti blocchi di convoluzione rende le feature estratte maggiormente espressive rispetto ad un unico layer di convoluzione;
- supponendo che tutti i blocchi abbiano C livelli, si vede facilmente che il singolo layer convoluzionale con filtri 7×7 avrebbe un numero di pesi pari a $C \times (7 \times 7 \times C) = 49C^2$, mentre i tre layer di convoluzione con filtri 3×3 conterebbero solo $3 \times (C \times (3 \times 3 \times C)) = 27C^2$ pesi.

Al fine di ottenere una confrontabilità maggiormente significativa con alcune architetture note in letteratura, come VGGNet, si è scelto di utilizzare:

- una dimensione di input 224×224 , ragionevole rispetto a quella dei ritagli come mostra l'analisi statistica effettuata (par. 4.4.4);
- un numero di filtri identico all'interno di uno stesso blocco e con proporzione doppia nei blocchi successivi.

Si descrive, infine, il criterio di *trade-off* utilizzato per la scelta del numero dei filtri e la dimensione di output degli ultimi layer.

Sono stati effettuati numerosi addestramenti e, a parità di altri parametri, si è provato a dimezzare di volta in volta il numero dei filtri. Le performance si sono mantenute costanti, con una riduzione del tempo di addestramento, fino all'architettura descritta. Semplificando ulteriormente la struttura del classificatore, si assiste ad un calo di performance, con una riduzione dell'*accuracy* sul medesimo *test set* da 99% a 90%. L'architettura utilizzata è stata, cioè, la più semplice, tra quelle testate, in grado di fornire un *accuracy* molto elevata. Il paragrafo successivo descrive dettagliatamente i risultati quantitativi.

4.4.3 Addestramento

Il dataset è stato partizionato secondo la tecnica *k-Fold Cross Validation* stratificata con $k = 5$ (i dettagli teorici ed implementativi sono riportati al par. 3.5). Poiché la dimensione del dataset è $n = 15228$, si ottengono le seguenti partizioni utilizzate, a turno, per l'addestramento di cinque reti convoluzionali differenti:

TrainSize	[12183, 12182, 12182, 12182, 12183]
TestSize	[3045, 3046, 3046, 3046, 3045]

Ciascuna coppia di *fold* è rappresentata da un oggetto `imageDatastore` (par. 2.1.2), rispettivamente `imds_Train` per il *fold* di *training* e `imds_Test` per il *fold* di *test*.

Ogni *fold* di *training* è sottoposto ad *augmentation* durante la fase di addestramento. A ciascuna immagine del *fold* è applicata, con una probabilità del 50%:

- una riflessione rispetto all'asse x;
- una rotazione di un angolo casualmente estratto dall'intervallo $[-20; 20]$ gradi;
- una traslazione sull'asse x di una quantità casualmente estratta dall'intervallo $[-60; 60]$ pixel.

Le operazioni applicate contribuiscono in maniera casuale ad ottenere ulteriore variabilità del dataset impiegato per la fase di addestramento. Si è ritenuto opportuno impiegare poche trasformazioni ad effetto limitato: le uniche che non alterassero eccessivamente il contenuto dei ritagli di classe *Pinna* rispetto al problema in esame ed al dataset a disposizione, già di per sé contenente grande variabilità relativa alla classe *Pinna*.

Secondo le tecniche di implementazione descritte al par. 3.4.5, è stato definito allo scopo un oggetto `imageDataAugmenter` con i seguenti parametri:

```
rotation = [-20 20];
translation = [-60 60];

augmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandRotation',rotation, ...
    'RandXTranslation',translation);
```

Tale oggetto è stato poi impiegato per la definizione di un `augmentedImageDatastore`, a partire dalle immagini del *fold* di *training* `imds_Train` e dalla dimensione di input della rete `inputSize`:

```
augimds_Train = augmentedImageDatastore(inputSize,imds_Train, ...
    'DataAugmentation',augmenter);
```

Anche il *fold* di *test* `imds_Test` è utilizzato per la creazione di un oggetto `augmentedImageDatastore`, ma senza la specifica della proprietà `'DataAugmentation'`, cioè senza l'applicazione delle trasformazioni di *augmentation*.

```
augimds_Test = augmentedImageDatastore(inputSize,imds_Test);
```

Per ogni classificatore, l'addestramento è stato eseguito con il metodo *stochastic gradient descent with momentum*, con dimensione del *minibatch* pari a 20, numero di epoche pari a 30 e *learning rate* iniziale pari a 0.0003. Ciò ha portato alla definizione del seguente oggetto `trainingOptions`:

```
miniBatchSize = 20;
valFrequency = floor(numel(augimds_Train.Files)/miniBatchSize);

options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
```

```
'MaxEpochs',30, ...
'InitialLearnRate',3e-4, ...
'Shuffle','every-epoch', ...
'ValidationData', augimds_Test, ...
'ValidationFrequency',valFrequency, ...
'Verbose',false, ...
'Plots','training-progress');
```

L'addestramento, per ogni coppia di *fold*, è stato lanciato mediante la funzione `trainNetwork`, specificando l'architettura della rete nell'oggetto `layers` (definito nel paragrafo precedente 4.4.2):

```
miacnn = trainNetwork(augimds_Train, layers, options);
```

I risultati globali (media dei cinque classificatori) sono riportati nella seguente tabella 4.4.

Il tempo di addestramento è stato di circa 3 ore e 20 minuti per ciascuna delle cinque reti, utilizzando la modalità a singolo processore grafico su una macchina equipaggiata di Intel Core i5-6400T @ 2.20 GHz con 8 GB di RAM e Nvidia GeForce 930M con 2 GB dedicati in modalità DDR3.

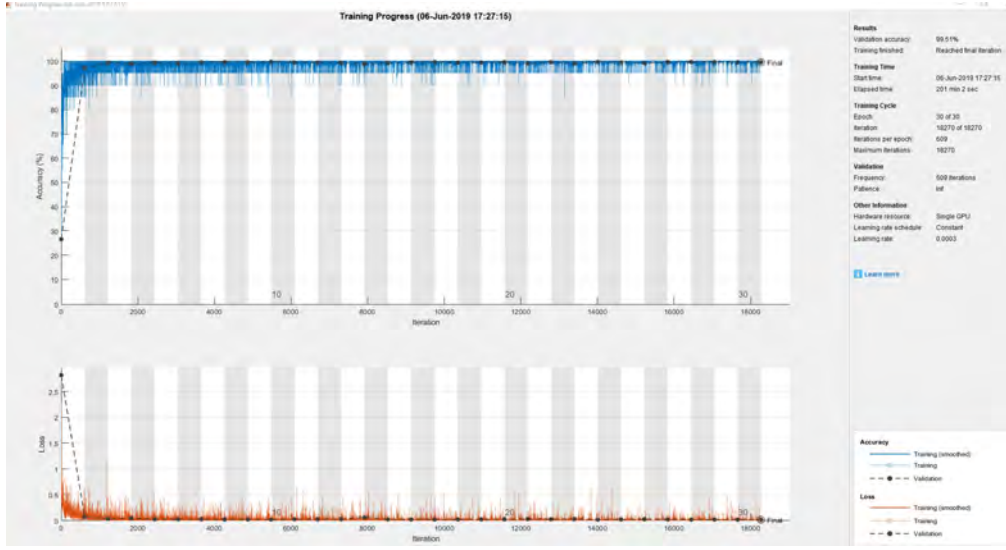
Accuracy	Sensitivity	Specificity	TP	FP	TN	FN
99.36 %	99.26 %	99.42 %	801	13	2226	6

Tabella 4.4: Media delle performance delle cinque reti neurali convoluzionali rispetto ai rispettivi *fold* di test. *True Positive* si riferisce alle classificazioni corrette della classe *Pinna*

Si mostrano, inoltre, nella fig. 4.16, l'andamento della procedura di addestramento relativo alla prima rete e la matrice di confusione ottenuta come media dei risultati delle cinque reti.

Le performance sono state, inoltre, misurate su un campione di 20888 ritagli ottenuti dall'elaborazione, attraverso la routine CropFin v1 (par. 5.1), di un dataset differente, composto da immagini scattate alle isole Azzorre nel giugno 2018.

Per lo scopo, sono stati controllati manualmente i ritagli salvati nelle varie cartelle di destinazione. Sono state considerate *True Positive* solo le immagini contenute nelle cartella *Pinne* che rispecchiassero il criterio di scelta delle immagini della classe *Pinna* utilizzato in fase di creazione del dataset (par. 4.4.1). I ritagli presenti nelle cartelle *Incerte* sono stati considerati, a prescindere, come un errore di classificazione: *False Positive* o *False Negative* a seconda del loro contenuto. I risultati ottenuti sono mostrati nella seguente tabella 4.5.



(a)



(b)

Figura 4.16: Risultati dell’addestramento: (a) *Training Progress* relativo alla prima coppia di *fold*, (b) Matrice di confusione globale, ottenuta come media dei risultati di ciascun classificatore

Accuracy	Sensitivity	Specificity
92 %	85 %	95 %

Tabella 4.5: Performance della classificazione rispetto ad un campione di ritagli ottenuti dall’elaborazione di un dataset differente, composto da immagini scattate nelle isole Azzorre

4.4.4 Salvataggio dei ritagli

Sebbene le reti convoluzionali siano addestrate per la classificazione binaria *Pinna* oppure *No Pinna*, entrambe le routine prevedono il salvataggio di ciascun ritaglio in

una di tre sottocartelle: *Pinne* oppure *No Pinne* oppure *Incerte*, rendendo, di fatti, la classificazione finale di tipo ternario. Si illustrano, di seguito, le differenti strategie usate in ciascuna routine per ottenere la classificazione ternaria come un processo di "rifinitura" di quella binaria.

In CropFin v1 si adotta una classificazione multipla. Le cinque reti convoluzionali, addestrate secondo la procedura di *5-Cross Fold Validation* descritta al paragrafo precedente, eseguono la propria classificazione binaria per ciascun ritaglio. L'esito definitivo è stabilito secondo la seguente strategia di tipo *major voting*: se le classificazioni *Pinna*

- sono maggiori o uguali a 4, la cartella di destinazione è *Pinne*
- sono pari a 2 o a 3, la cartella di destinazione è *Incerte*
- sono minori o uguali a 1, la cartella di destinazione è *No Pinne*

In CropFin v2, invece, a fronte di un'analisi statistica preliminare, viene eseguito un controllo sull'*aspect ratio* di ciascun ritaglio (rapporto larghezza/altezza):

- se esso risulta un *outlier* rispetto alla distribuzione dell' *aspect ratio* per la classe *Pinna*, allora il ritaglio viene scartato, senza che sia salvato in alcuna delle sottocartelle di output
- altrimenti, il ritaglio viene sottoposto ad una singola classificazione binaria da parte di un'unica rete convoluzionale (scelta a priori tra le cinque addestrate):
 - se la classificazione è *No Pinna* viene salvato nella rispettiva cartella *No Pinne*
 - se la classificazione è *Pinna* viene eseguito un ulteriore controllo sull'*aspect ratio*:
 - * se esso rientra in una certa porzione della distribuzione di classe *Pinna* definita come "fascia di incertezza", allora il ritaglio viene salvato nella cartella *Incerte*
 - * altrimenti il ritaglio viene salvato nella cartella *Pinne*

La procedura appena descritta è implementata nella funzione `classification` (sez. 5.2).

La fig. 4.17 contiene i grafici che mostrano i risultati principali dell'analisi statistica effettuata sui ritagli. I parametri ricavati sono riportati nella tabella 4.6. Il campione utilizzato ha dimensione $n = 7851$ ed è stato ottenuto applicando CropFin v2 a:

- 7034 immagini scattate nel golfo di Taranto tra il 2013 ed il 2018
- 11290 immagini scattate alle isole Azzorre nel giugno 2018.

I ritagli ottenuti sono stati manualmente selezionati con lo stesso criterio adottato per la creazione del dataset destinato all'addestramento dei classificatori (par. 4.4.1).

Il campione appena descritto è stato creato, inoltre, allo scopo di indirizzare alcuni sviluppi futuri discussi al capitolo 7.

Guardando alla figura 4.17(c) si nota come la distribuzione dell'*aspect ratio* dei ritagli di classe *Pinna* sia di forma molto simile ma solo in minima parte sovrapposta a quella dei ritagli di classe *No Pinna*. Questo è ciò che motiva principalmente la scelta di introdurre delle soglie di *aspect ratio* che contribuiscano alla classificazione dei ritagli.

In particolare, la figura 4.17(d) mostra le aree di interesse considerate:

- Fascia dei ritagli candidati per la cartella *Pinne*: compresa tra 0.40 e 4.70 (area centrale compresa tra le linee verdi tratteggiate)
- Fasce dei ritagli candidati per la cartella *Incerte*: comprese tra 0.30 e 0.40 e tra 4.70 e 5.00 (comprese tra ciascuna linea rossa e linea verde vicina)
- Fasce dei ritagli scartati: minore di 0.30 e maggiore di 5.00 (a sinistra della prima linea rossa e a destra della seconda linea rossa)

Sulla base di questa suddivisione, solo i ritagli il cui *aspect ratio* rientri nelle fasce candidate sono sottoposte alla classificazione della rete convoluzionale. Rispetto alla strategia di tipo *major voting* adottata in CropFin v1, il nuovo approccio introdotto in CropFin v2 consente, a parità di numero di ritagli da classificare, di:

- risparmiare tempo e memoria necessari per le cinque classificazioni con reti convoluzionali effettuate per ciascun ritaglio; di particolare rilevanza è la possibilità di evitare classificazioni e scrittura di tutti i ritagli che, a causa delle loro dimensioni, hanno una probabilità pressochè nulla di contenere pinne dorsali;
- razionalizzare il contenuto delle cartelle di salvataggio, facilitando l'eventuale selezione manuale dei ritagli di output.

Si precisa, infine, che la rinuncia alla classificazione multipla in CropFin v2 è giustificata anche dall'uniformità delle classificazioni riscontrata nei test di CropFin v1. Il dominio fortemente limitato del dataset e la casualità introdotta in fase di addestramento rendono poco significativa la variabilità dell'apprendimento dei cinque classificatori ottenuta mediante la procedura di *5-Cross Fold Validation*. La scelta, tra le cinque, della rete impiegata in CropFin v1 non ha, pertanto, particolare rilevanza ed è stata effettuata in maniera casuale.

Mediana	Max	Min	1Q	2Q	3Q	Media	Dev. Std
1.40	6.75	0.33	1.11	1.40	1.81	1.54	0.66

Tabella 4.6: Parametri statistici ricavati dal campione dei ritagli della classe *Pinna*. Dimensione del campione $n = 7851$.

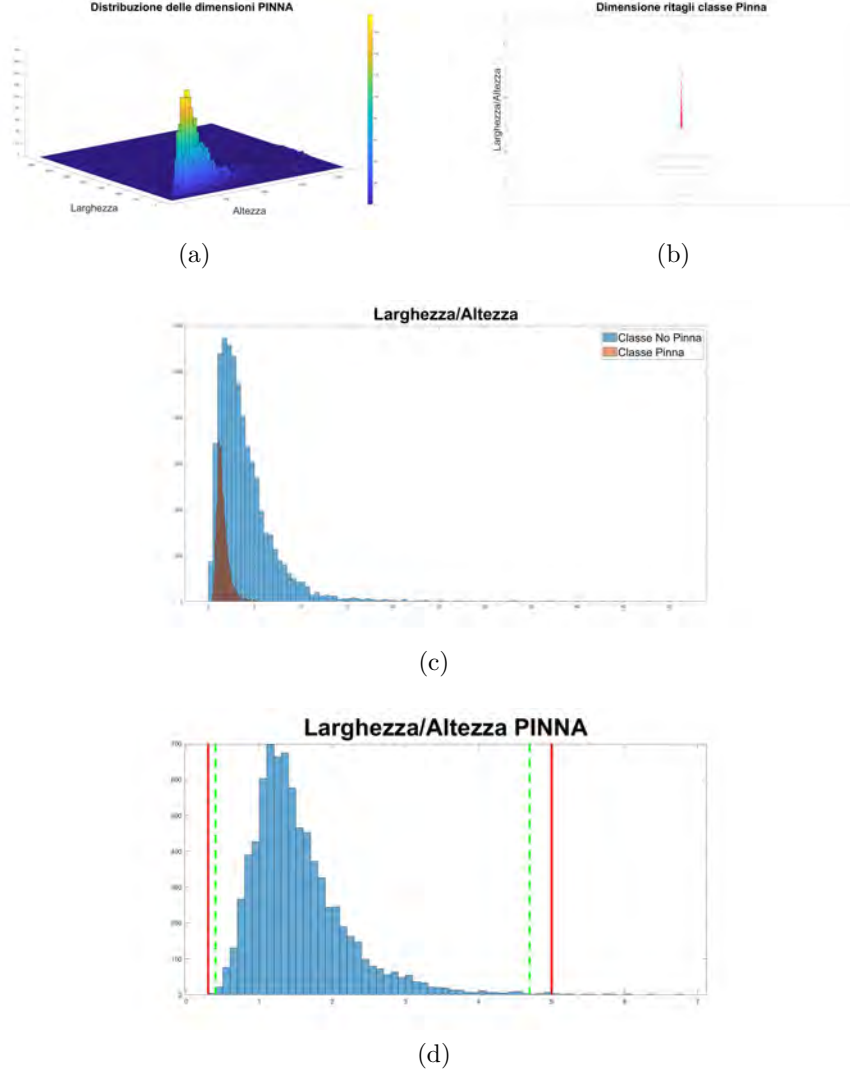


Figura 4.17: Statistiche dei ritagli della classe *Pinna*: (a) Istogramma bivariato delle dimensioni, (b) Box-plot dell'*aspect ratio*, (c) Istogramma comparato dell'*aspect ratio* dei ritagli della classe *Pinna* e della classe *No Pinna*, (d) Istogramma dell'*aspect ratio* della classe *Pinna* con le soglie utilizzate per la classificazione: le linee rosse rappresentano le soglie per l'identificazione degli *outlier*, mentre le linee verdi rappresentano le soglie di incertezza

4.5 Prestazioni

A conclusione del capitolo, si riporta un'analisi che consente di quantificare le performance dei metodi finora descritti, attraverso un confronto dei risultati prodotti dalle due routine CropFin v1 (par. 5.1) e CropFin v2 (par. 5.2).

Il campione utilizzato per il test ha dimensione $n = 500$ ed è stato selezionato in maniera casuale a partire da circa 3000 scatti collezionati nel 2018 dalla *Nova Atlantis Foundation* (par. 1.1) durante campagne di avvistamento alle isole Azzorre.

La scelta del campione è basata sulle seguenti motivazioni:

- possibilità di testare l'accuratezza delle classificazioni effettuate dalle reti convoluzionali su un dataset differente da quello utilizzato per l'addestramento (costruito con immagini scattate nel Golfo di Taranto, come descritto al par. 4.4.1);
- possibilità di testare le differenti strategie di segmentazione utilizzate nelle due routine (par. 4.1) rispetto a condizioni di scatto particolari e diversificate.

L'analisi effettuata prevede, anzitutto, un confronto della quantità dei ritagli salvati in ciascuna delle tre cartelle di output: *Pinne*, *No Pinne*, *Incerte* (par. 4.4.4).

Quindi, anzichè misurare i parametri standard legati alle *performance* della classificazione (già riportati nel par. 4.4.3), si è ritenuto maggiormente significativo contare il numero totale di pinne manualmente ritagliabili ed effettivamente identificabili all'interno del campione scelto (indicato con RID) e, sulla base di questo, valutare:

- il numero di pinne correttamente ritagliate e salvate all'interno della cartella *Pinne* (indicato con RS)
- il numero di pinne correttamente ritagliate ma non salvate all'interno della cartella *Pinne* (indicato con RNS)

Ciò permette di ottenere delle misure maggiormente realistiche delle prestazioni, che riflettono sia gli interessi sia la percezione dell'efficienza della routine dal punto di vista dell'utente.

Infatti, per come è stata concepita la fase di classificazione all'interno delle soluzioni proposte, la misurazione di *accuracy*, *sensitivity* e *specificity* è utile solo a stimare l'efficienza del filtraggio finale, senza invece quantificare le prestazioni delle fasi precedenti, decisive per la risoluzione del problema di *object detection* in esame.

A fronte di $RID \simeq 430$ per il campione scelto, i risultati delle misurazioni per entrambe le routine sono riportati nella seguente tabella 4.7. Si nota, innanzitutto, che il numero totale di ritagli salvati è più che dimezzato in CropFin v2 rispetto a CropFin v1 (riduzione del 56%).

Tale riduzione si rivela di carattere positivo confrontando gli altri dati. Si vede, infatti, che il risparmio è sostanzialmente ottenuto riducendo il contenuto della cartella *No Pinne* del 68% e quello della cartella *Incerte* del 95%.

Inoltre, nonostante il contenuto della cartella *Pinne* sia leggermente inferiore, si nota che la quantità di pinne correttamente ritagliate e classificate (RS) sia incrementata del 33% in CropFin v2 rispetto a CropFin v1.

	CropFin v1	CropFin v2
Totale ritagli	2030	882
Cartella <i>Pinne</i>	485	444
Cartella <i>No Pinne</i>	1352	429
Cartella <i>Incerte</i>	193	9
RS	250	332
RNS	2	29

Tabella 4.7: Performance a confronto delle routine CropFin v1 e CropFin v2

L'efficienza della routine η può essere definita come la percentuale delle pinne correttamente ritagliate e classificate (RS) rispetto al numero totale di pinne effettivamente ritagliabili e identificabili all'interno del campione (RID):

$$\eta = \frac{RS * 100}{RID} \quad (4.1)$$

Analogamente, è possibile definire l'efficienza complessiva η_{od} di risoluzione del problema di *object detection* come la percentuale delle pinne correttamente ritagliate (RS+RNS) rispetto al numero totale di pinne effettivamente ritagliabili e identificabili all'interno del campione (RID):

$$\eta_{od} = \frac{(RS + RNS) * 100}{RID} \quad (4.2)$$

Questo parametro è definito tale poichè quantifica in maniera significativa le prestazioni delle fasi decisive di segmentazione, filtraggio e ritaglio adattivo che precedono la fase di classificazione all'interno di entrambe le routine.

La seguente tabella 4.8 mostra i parametri di efficienza misurati per entrambe le routine sul campione in esame.

	CropFin v1	CropFin v2
η	58 %	77 %
η_{od}	58 %	84 %

Tabella 4.8: Confronto dei parametri di efficienza per le routine CropFin v1 e CropFin v2

Sulla base dei risultati ottenuti si può concludere osservando che i metodi introdotti in CropFin v2, descritti nel corso del presente capitolo, sono effettivamente in grado di offrire un miglioramento generale delle performance rispetto a CropFin v1.

In particolare, il valore di 84 % rilevato per l'efficienza complessiva mostra che l'introduzione della tecnica innovativa di segmentazione (par. 4.1.2), unita con la razionalizzazione delle fasi successive di filtraggio delle regioni binarie (par. 4.2) e di ritaglio adattivo (par. 4.3), consente di rendere maggiormente robusta la soluzione al problema di *object detection*.

Si evidenzia anche che riuscire a ritagliare l'84 % delle pinne, a fronte di $RID \simeq 430$, significa riuscire ad identificare la quasi totalità degli esemplari durante una campagna di avvistamento. Infatti, le immagini raccolte nel corso di un'unica missione contengono spesso una quantità considerevole di scatti duplicati e riferiti allo stesso esemplare.

Infine, si osserva che:

- all'interno delle soluzioni proposte, considerando poco probabili progressi sostanziali nella fase di filtraggio delle regioni binarie e di ritaglio adattivo, l'efficienza rimane principalmente legata alle prestazioni della fase di segmentazione automatica.
- l'eventuale differenza tra η ed η_{od} è legata alle prestazioni della fase di classificazione finale. Nel caso di CropFin v2, essa può essere colmata migliorando la precisione della rete neurale convoluzionale utilizzata in fase di classificazione.

Entrambe le considerazioni sono riprese nel capitolo 7, nell'ottica di indirizzare gli sviluppi futuri del lavoro.

Capitolo 5

Descrizione delle routine

Si descrive sinteticamente, nel paragrafo relativo, il funzionamento di ciascuna delle routine Matlab sviluppate per l'identificazione ed il ritaglio di pinne dorsali a partire da una collezione di immagini.

Si riporta il codice di alcune delle funzioni più importanti, citate durante la descrizione delle tecniche utilizzate nel capitolo precedente (4). Le routine sono interamente disponibili online con le istruzioni per il loro utilizzo ai link successivamente indicati. Ad esse si applica la *Creative Commons License* CC-BY-SA¹. La versione Matlab utilizzata è R2019a.

Il par. 6 riporta ulteriori codici citati nel capitolo precedente, non inclusi nelle routine, ma ad esse direttamente connessi.

Entrambe le routine sono strutturate in un repository del tipo

```
/Files  
/Functions  
main.m
```

Dal punto di vista dell'utente, il funzionamento di entrambe le routine è identico e può essere riassunto dal *folwchart* di fig. 5.2. La procedura di ritaglio automatico si avvia dal file `main.m` inserendo:

- il percorso contenente le immagini da ritagliare nella variabile `input`;
- il percorso in cui salvare i ritagli nella variabile `output`.

Vengono elaborate, una ad una, tutte le cartelle che contengono immagini a partire dal percorso specificato in `input`. Per ciascuna cartella, vengono create tre sottocartelle *Pinne*, *No Pinne* e *Incerte* in cui verranno salvati i ritagli, a seconda della classificazione ottenuta.

¹<https://creativecommons.org/licenses/by-sa/4.0/>

Se i percorsi specificati in `input` e `output` sono differenti, le sottocartelle vengono create replicando interamente la struttura della directory presente in `input` a partire dal percorso `output`.

Requisiti ed istruzioni maggiormente dettagliate per l'esecuzione delle routine sono disponibili nei relativi repository online.

5.1 CropFin v1

La routine CropFin v1 può essere descritta con il medesimo *flowchart* della routine CropFin v2 (fig. 5.2), con l'unica eccezione di non escludere mai dal flusso la fase di salvataggio.

Il codice è disponibile online all'indirizzo <https://github.com/gvlos/CropFinv1>.

Si descrivono brevemente i passaggi eseguiti su ogni immagine, facendo riferimento ai paragrafi in cui sono descritti i metodi e le tecniche implementative utilizzate:

1. Ridimensionamento con fattore di scala 0.2 (par. 2.2.1);
2. Conversione nello spazio colore CIE 1976 L*a*b* (par. 2.1.1);
3. Segmentazione e binarizzazione basate sul metodo Otsu (par. 4.1.1);
4. Filtraggio delle regioni binarie secondo la strategia a due fasi descritta nel par. 4.2;
5. Ritaglio adattivo dell'immagine originale a partire da ciascuna regione binaria, sulla base del secondo metodo descritto nel par. 4.3;
6. Classificazione di ciascun ritaglio con cinque reti convoluzionali (par. 4.4) e conseguente salvataggio nella cartella stabilita dall'applicazione di una strategia di tipo *major voting* (par. 4.4.4).

La figura 5.1 mostra l'evoluzione dei passaggi. Segue il listato di alcune funzioni.

Threshold_LAB_L_b

Implementazione della segmentazione basata sul metodo Otsu (par. 4.1.1).

```
1
2 %% Threshold_LAB_L_b
3
4 % Binarizzazione dell'immagine basata sulla sogliatura con metodo Otsu
  per
5 % la localizzazione delle pinne dorsali
6
7 %% INPUT
8
9 % im = immagine RGB
```


Capitolo 7

Conclusioni e sviluppi futuri

È stato affrontato il problema della creazione di una routine Matlab in grado di riconoscere e ritagliare automaticamente pinne dorsali di delfino a partire da una collezione di immagini, al fine di facilitare la foto-identificazione automatica degli esemplari della specie *Grampus Griseus*.

I tratti distintivi che consentono al nostro occhio di riconoscere una pinna dorsale di delfino all'orizzonte sono, indubbiamente, la sua forma caratteristica che affiora dall'acqua ed il suo colore. Riuscire a codificare in maniera univoca questo preciso tipo di conoscenza (per noi estremamente semplice) costituisce tutt'oggi una sfida aperta per la comunità della *computer vision*.

Il dominio ristretto del problema in esame ha consentito un approccio mediante tecniche automatiche di segmentazione. Con il metodo Otsu si ottengono buoni risultati, con qualche eccezione in caso di condizioni variabili di scatto (par. 2.3.1). Il metodo di segmentazione innovativo introdotto, basato esclusivamente su colori manualmente etichettati all'interno dello spazio Lab (par. 4.1.2) ha consentito di passare da un approccio di tipo *domain-oriented* ad un approccio interamente *domain-based*, con un evidente beneficio nelle performance (par. 4.5). La routine finale, CropFin v2, ha infatti riportato un'efficienza dell'84% nella localizzazione delle pinne. Ciò è stato possibile grazie alla grande disponibilità di immagini in numerose condizioni di scatto.

Pur migliorando, per quanto possibile, le tre fasi iniziali, il risultato finale è costituito mediamente da un numero di ritagli che non contengono pinne almeno pari a quelli che ne contengono. Ciò ha reso necessario, nell'ottica di una procedura di foto-identificazione completamente automatizzata, l'impiego di un classificatore che funga da filtro finale.

La grande disponibilità dei dati e la relativa semplicità del problema di classificazione consentono, anche con un'architettura molto semplice, di ottenere specifiche incoraggianti: 99.36% di accuracy misurata sul *test set*, consentendo di filtrare utilmente i risultati della fasi di *processing* precedenti.

Grazie al metodo innovativo di segmentazione introdotto, è stato, inoltre, creato un

nuovo dataset di dimensione $n \simeq 15000$ in cui, a ciascuna immagine, sono state associate le coordinate dei rettangoli che contengono le pinne dorsali. Questo può essere utilizzato come *training set* per uno dei metodi allo stato dell'arte SSD o YOLO, al fine di tentare un approccio completamente differente al problema di *object detection* ed interamente basato su tecniche di deep learning. Lo script utilizzato per la creazione automatica del nuovo dataset è disponibile online: <https://github.com/gvlos/CFUtil/NewDataset/>.

Bibliografia

- [1] Maglietta R., Renò V. *et. al.*, *DolFin: an innovative digital platform for studying Risso's dolphins in the Northern Ionian Sea*, Scientific Reports 8, 2018: <https://www.nature.com/articles/s41598-018-35492-3>. Si descrive lo studio di un gruppo di cetacei della specie *Grampus Griseus* nel Golfo di Taranto, quindi si descrive la piattaforma DolFin e l'algoritmo *SPiR*, punti di partenza per l'intero lavoro della presente tesi.
- [2] Legnaro M., *Il caso Ilva tra società, ambiente e lavoro*. Descrizione esaustiva del caso ILVA a Taranto, disponibile online su unipd.it.
- [3] Kiszka J. *et. al.*, *Grampus griseus, IUCN Red List of Threatened Species, Assessment Information in detail*, <https://www.iucnredlist.org/species/9461/50356660>. Si segnala la specie *Grampus Griseus* come specie *data deficient* nel Mediterraneo.
- [4] Forenza F., *Tecniche innovative di Computer Vision per la Foto-Identificazione dei cetacei*, Università degli Studi di Bari, tesi di laurea triennale, a.a. 2017/2018, rel. prof. Giovanni Dimauro, correl. dr. Vito Renò. Lavoro precedente e punto di partenza per l'intero la presente tesi, nonché fonte delle principali idee utilizzate nell'implementazione della Routine v1.
- [5] Seller E., *Pipeline innovativa per la foto-identificazione del Grampus Griseus*, Università degli Studi di Bari, tesi di laurea triennale, a.a. 2018/2019, rel. prof. Giovanni Dimauro, correl. dr.ssa Rosalia Maglietta. Lavoro svolto in contemporanea a quello effettuato nella presente tesi, frutto del coordinamento delle attività di tirocinio presso il CNR, in cui è trattata nel dettaglio la parte di foto-identificazione degli esemplari di grampo, successiva al ritaglio automatico qui proposto.
- [6] Buehler P. *et. al.*, *An automated program to find animals and crop photographs for individual recognition*, Ecological Informatics 50 (2019), 191-196. Si espone una tecnica di *object detection* per l'identificazione dei torsi delle giraffe.
- [7] Schofield D. *et. al.*, *Chimpanzee face recognition from videos in the wild using deep learning*, Science Advances, vol.5, no.9. Si espone una tecnica di *object detection* e di foto-identificazione automatica per il riconoscimento delle facce di un gruppo di scimpanzè.

- [8] Szeliski R., *Computer Vision: Algorithms and Applications*, Springer 2010. Principale libro di riferimento per i problemi di *Computer Vision*, disponibile online: szeliski.org/Book
- [9] Naldi G., Pareschi L., Russo G., *Introduzione al calcolo scientifico*, Mc-Graw Hill, 2001. Utile riferimento per la discesa del gradiente e l'approccio alla rete neurali come approssimazione di funzione e l'utilizzo di matrici per le trasformazioni di oggetti nel piano (immagini ruotate, traslate, ecc.)
- [10] Luise M., Vitetta G., *Teoria dei segnali*, McGraw-Hill, 2009. Trattazione completa ed esaustiva di segnali continui e discreti e della teoria della probabilità; tratta anche di algoritmi di convoluzione veloce basati su FFT e loro implementazioni in Matlab.
- [11] Giua A., Seatzu C., *Analisi dei sistemi dinamici*, Springer-Verlag Italia, 2009, pp. 492-496. Utile per la definizione e le proprietà dell'operazione di convoluzione tra segnali, nonché sua applicazione nell'ambito dell'analisi dei sistemi.
- [12] Hodges A., *Alan Turing. Storia di un enigma*, Bollati Boringhieri, 2014, p.167. Hodges rievoca alcune note scritte durante un corso di lezioni sui metodi della matematica tenuto nel 1944 come passatempo serale da Alan Turing ad Hanslope Park, luogo di ibernazione in attesa della fine della guerra per il gruppo che aveva lavorato per i servizi segreti inglesi a Bletchley Park.
- [13] Otsu N., *A Threshold Selection Method from Gray-Level Histograms*, IEEE Trans. Syst. Man. Cybern., vol. 9, no. 1, pp. 62–66, 1979. Paper originale in cui si introduce il metodo di sogliatura automatica, nominato metodo Otsu in onore dell'autore.
- [14] Goodfellow I., Bengio Y. , Courville A., *Deep Learning*, MIT Press, 2016. Riferimento principale per la gran parte dei contenuti teorici riguardanti il deep learning.
- [15] The MathWorks, Inc., *Matlab R2019a*, Documentazione ufficiale disponibile all'indirizzo <https://it.mathworks.com/help/matlab/>. Le principali pagine consultate sono relative al Deep Learning Toolbox <https://it.mathworks.com/help/deeplearning/>
- [16] Karpathy A., *Convolutional Neural Networks for Visual Recognition*, note per la *Stanford CS class CS231n*, disponibili alla pagina <http://cs231n.github.io/>. Il corso, intitolato *Convolutional Neural Networks for Visual Recognition*, offre una trattazione esaustiva delle reti neurali semplici e convoluzionali, con esempi in Python.
- [17] Karpathy A., *Connecting Images and Natural Languages*, dissertazione per la tesi di dottorato in *Computer Science* presso la *Stanford University*, disponibile online alla pagina <http://cs.stanford.edu/people/karpathy/>. Viene affrontato un problema avanzato di *computer vision* e viene fornita una descrizione concisa del *deep learning*.

- [18] LeCun Y., Bengio Y., Hinton G., *Deep Learning*, Nature 521, 436-444, 2015. Pubblicazione che ha reso popolare il successo del *deep learning* nella comunità scientifica. Gli autori hanno ricevuto, per i loro meriti scientifici nel campo, il premio Turing nel 2018.
- [19] Schmidhuber J., *Deep Learning in Neural Networks: An Overview*, Neural Networks, Volume 61, January 2015, Pages 85-117. Offre una panoramica generale della nascita e dello sviluppo del deep learning nell'ambito delle reti neurali. In particolare, individua origine e svolta delle reti neurali convoluzionali.
- [20] Rosasco L., *Introductory Machine Learning Notes*, Note del corso di Machine Learning, Università degli studi di Genova, disponibile all'indirizzo <http://lcs1.mit.edu/courses/ml/1819/MLNotes.pdf>. Utile riferimento per inquadrare il machine learning nell'ambito della cosiddetta teoria dell'apprendimento statistico. I concetti introduttivi sono illustrati dallo stesso autore nel corso Statistical Learning Theory and Applications, MIT course 9.520/6.860S, Class 02, disponibile all'indirizzo <https://youtu.be/SFxypsvhhMQ>
- [21] Simonyan K., Zisserman A., *Very Deep Convolutional Networks for Large-Scale Image Recognition*, arXiv technical report, 2014, http://robots.ox.ac.uk/~vgg/research/very_deep/. Si descrivono i dettagli di alcune note architetture di CNN, indicate come VGG.