



## POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE - DEI  
Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Tesi di laurea in CALCOLO NUMERICO

# Analisi e classificazione di immagini mediante reti neurali convoluzionali per la foto-identificazione dei cetacei

Relatore:

**Prof. Tiziano POLITI**

Correlatore:

**Dott. Vito RENÒ**

Laureando:

**Tommaso MONOPOLI**

Matricola 568581



# Sommario

Il sistema terrestre è sempre stato soggetto alle conseguenze delle attività umane, che mettono fortemente a rischio la biodiversità degli ecosistemi acquatici e marini. Diversi studi cercano di capire in che modo la perdita della biodiversità possa alterare l'integrità e il funzionamento di tali ecosistemi.

Una risposta a questa domanda può essere ricercata negli studi effettuati sulla distribuzione e sullo stato di conservazione dei cetacei, oggetto di numerose ricerche negli ultimi anni. È noto, infatti, che lo stato di salute di ecosistemi con catene alimentari lunghe, come quello marino, si riflette 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.

Una tecnica non invasiva che permette di studiare i cetacei nel loro habitat naturale è la *foto-identificazione* degli esemplari, che prevede il riconoscimento - automatico o manuale - di uno stesso individuo in diverse immagini collezionate nel tempo, mediante l'analisi di suoi particolari segni distintivi (ad esempio dei graffi sul dorso, o la forma delle pinne) catturati in uno scatto. Quest'attività può essere effettuata manualmente, ma con un grande costo in termini di tempo per i ricercatori, che spesso hanno a disposizione diverse migliaia o milioni di fotografie, scattate nel corso di anni. L'evidente difficoltà nell'approccio manuale alla foto-identificazione dei cetacei (tutt'oggi ancora ampiamente operata) suggerisce l'applicazione di metodologie di *Computer Vision* e *Machine Learning* per automatizzare tale attività.

L'obiettivo del presente lavoro di tesi è la creazione di un modello di classificazione binario capace di discriminare la presenza o meno di una pinna dorsale di un delfino all'interno di un'immagine. Questa è una delle operazioni che i biologi svolgono prima di poter foto-identificare gli esemplari fotografati durante le campagne di avvistamento: renderla automatica significa permettere ai biologi di risparmiare tempo e risorse, nonché indirizzare il lavoro di ricerca su task più significativi e di più alto livello. Le metodologie impiegate sono quelle del *machine learning*; in particolare, si è scelto di utilizzare la tecnica del *transfer learning* per il riuso e l'adattamento di modelli pre-addestrati, usati per risolvere task di classificazione diversi da quello in esame. Gli esperimenti condotti su dati reali acquisiti in mare permettono di rilevare l'efficacia e le alte prestazioni del classificatore oggetto di sviluppo del presente elaborato.



# Ringraziamenti

Devo ringraziare innanzitutto le persone che mi hanno permesso di svolgere questo lavoro di tesi.

Ringrazio il Dott. Vito Renò, per avermi concesso l'opportunità di intraprendere questo percorso di tirocinio, seguendomi con dedizione, professionalità, continui stimoli ed una buona dose di pazienza durante tutte le fasi dello sviluppo della mia tesi di laurea e nonostante le mie innumerevoli incertezze.

Un ringraziamento al CNR per l'esperienza estremamente formativa che ho avuto il privilegio di intraprendere.

Porgo un ringraziamento al Prof. Tiziano Politi, per la sua disponibilità in qualità di relatore e per i tanti consigli dispensati.

Ringrazio tutte le persone conosciute in questi tre anni di università. In particolare, un sentito ringraziamento a Gianvito Losapio: la sua dedizione ed il rigore nello studio, la sua vivace curiosità e le innumerevoli altre qualità della sua persona sono stati per me un riferimento costante ed un inesauribile stimolo a migliorarmi.

Ringrazio la mia famiglia, per dimostrare giorno dopo giorno nei miei confronti un supporto ed un affetto sempre maggiore di quello che io dimostro loro, e che però si meritano.

Ringrazio i miei amici, scusandomi in anticipo con loro se per motivi di brevità non potrò citarli tutti. In particolare ringrazio la *Corteccia* per la sincera e disinteressata amicizia che ci lega da anni. Ringrazio Marcello, per aver per primo infuso in me un po' della sua passione per lo studio e per aver condiviso con me tanti interessi in comune. Ringrazio Silvia, per i tanti bei momenti musicali che abbiamo passato e che passeremo insieme. Ringrazio Cristina, a cui se davvero potessi dedicherei 15 pagine di ringraziamenti (come tante volte le ho promesso).



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Interessi scientifici . . . . .	2
1.2	Foto-identificazione automatica del <i>Grampus griseus</i> . . . . .	3
1.3	Problema ed obiettivi . . . . .	6
1.4	Lavori simili . . . . .	7
<b>2</b>	<b>Metodologie</b>	<b>11</b>
2.1	Immagini digitali . . . . .	11
2.1.1	Spazio di colore Lab . . . . .	13
2.1.2	Collezioni di immagini in Matlab . . . . .	13
2.2	Trasformazioni . . . . .	14
2.2.1	Ridimensionamento . . . . .	14
2.2.2	Rotazione . . . . .	14
2.2.3	Riflessione . . . . .	14
2.2.4	Traslazione . . . . .	15
2.2.5	Convoluzione . . . . .	15
2.3	Problemi di <i>Computer Vision</i> . . . . .	17
2.3.1	Object recognition . . . . .	17
2.3.2	Object detection . . . . .	19
2.3.3	ImageNet Database e ILSVRC . . . . .	19
2.4	Machine Learning e Deep Learning . . . . .	21
2.4.1	Supervised Learning . . . . .	22
2.4.2	Underfitting e overfitting . . . . .	24
2.5	Classificatore lineare . . . . .	26
2.6	Reti Neurali . . . . .	30
2.6.1	I neuroni . . . . .	31
2.6.2	Architettura delle reti neurali artificiali . . . . .	32
2.7	Reti neurali convoluzionali . . . . .	33
2.7.1	Input Layer . . . . .	36
2.7.2	Convolution Layer . . . . .	36
2.7.3	Activation layer . . . . .	40
2.7.4	Pooling layer . . . . .	40
2.7.5	Fully Connected Layer . . . . .	41
2.7.6	Softmax layer . . . . .	42
2.8	Image preprocessing e augmentation . . . . .	42

2.9	Addestrare una rete neurale . . . . .	46
2.9.1	Inizializzazione dei parametri . . . . .	46
2.9.2	Loss function . . . . .	47
2.9.3	Stochastic gradient descent . . . . .	48
2.9.4	Backpropagation . . . . .	51
2.9.5	Scomparsa ed esplosione del gradiente . . . . .	52
2.10	Transfer Learning . . . . .	52
2.11	Ensemble learning . . . . .	54
2.12	AlexNet . . . . .	56
2.12.1	Architettura di AlexNet . . . . .	56
2.12.2	Funzione di attivazione ReLU . . . . .	57
2.12.3	Local Response Normalization . . . . .	57
2.12.4	Overlapping Max Pooling . . . . .	57
2.12.5	Data Augmentation . . . . .	58
2.12.6	Dropout . . . . .	58
2.12.7	Addestramento di AlexNet . . . . .	59
2.13	GoogLeNet . . . . .	61
2.13.1	Convoluzione $1 \times 1$ . . . . .	62
2.13.2	Modulo <i>Inception</i> . . . . .	63
2.13.3	Classificatori ausiliari . . . . .	63
2.13.4	Global Average Pooling . . . . .	64
2.13.5	Data Augmentation . . . . .	65
2.13.6	Architettura di GoogLeNet . . . . .	65
2.13.7	Addestramento di GoogLeNet . . . . .	66
2.14	ResNet . . . . .	68
2.14.1	<i>Batch Normalization</i> . . . . .	68
2.14.2	Residual blocks . . . . .	70
2.14.3	Architettura di ResNet-18 . . . . .	71
2.14.4	Architettura di ResNet-50 . . . . .	72
2.14.5	Data Augmentation . . . . .	74
2.14.6	Addestramento di ResNet-18 e ResNet-50 . . . . .	74
<b>3</b>	<b>Esperimenti e risultati</b>	<b>77</b>
3.1	Descrizione dei dataset utilizzati . . . . .	77
3.2	CropFin v1 . . . . .	78
3.2.1	Fase di ritaglio . . . . .	79
3.2.2	Fase di classificazione . . . . .	85
3.3	Classificazione mediante CNN e Transfer Learning . . . . .	86
3.3.1	Scelta delle CNN . . . . .	87
3.3.2	Addestramento . . . . .	87
3.3.3	Classificatore ensemble . . . . .	94
<b>4</b>	<b>Conclusioni e sviluppi futuri</b>	<b>99</b>

<b>5 Listato del codice sorgente</b>	<b>101</b>
5.1 Funzioni utili per il transfer learning . . . . .	101
5.1.1 createLgraphUsingConnections.m . . . . .	101
5.1.2 findLayersToReplace.m . . . . .	101
5.1.3 freezeWeights.m . . . . .	102
5.1.4 plotConfusionMatrix.m . . . . .	103
5.2 Ri-addestramento CNN . . . . .	103
5.2.1 AlexNet . . . . .	103
5.2.2 GoogLeNet . . . . .	106
5.2.3 ResNet-18 . . . . .	108
5.2.4 ResNet-50 . . . . .	110
5.3 Test sul dataset delle Azzorre . . . . .	112
5.3.1 Singoli classificatori . . . . .	112
5.3.2 Classificatore ensemble . . . . .	114
<b>Bibliografia</b>	<b>117</b>



# Capitolo 1

## Introduzione

*"È importante e risaputo che le cose non sempre sono ciò che appaiono. Per esempio sul pianeta Terra gli uomini hanno sempre ritenuto di essere più intelligenti dei delfini. Sostenevano infatti che mentre loro avevano inventato un sacco di cose, come la ruota, New York, le guerre, ecc., i delfini non avevano fatto altro che sguazzare nell'acqua divertendosi. Al contrario invece, i delfini sapevano da tempo dell'imminente distruzione della Terra e avevano tentato più volte di avvertire l'umanità dell'incombente pericolo; ma i loro messaggi erano stati fraintesi e interpretati come divertenti tentativi di dare calci a palle da football o di fischiare per avere bocconcini prelibati. Così alla fine i delfini rinunciarono e se ne andarono dalla Terra coi propri mezzi, poco prima che arrivassero i vagoni."*

– Douglas Adams, *Guida galattica per gli autostoppisti*

Il presente lavoro di tesi ha come oggetto lo sviluppo di un modello di classificazione in grado di discriminare il contenuto informativo di un'immagine, verificando se essa raffiguri o meno una pinna dorsale di un delfino. Questo modello si propone come un'alternativa migliore (in quanto a prestazioni raggiunte) rispetto al modello nativo presente in *CropFin v1*, una routine che effettua il rilevamento e l'estrazione di eventuali pinne di delfini presenti in un'immagine. La routine *CropFin v1* è frutto di un precedente lavoro di tesi triennale [1], di cui questo elaborato vuole essere una naturale prosecuzione. Il miglioramento delle prestazioni della classificazione di CropFin v1 si innesta in un più ampio contesto di lavoro: quello della foto-identificazione dei cetacei (par. 1.2).

Il presente lavoro di tesi è stato sviluppato nel corso di un'attività di tirocinio presso l'Istituto *STIIMA*<sup>1</sup> del Consiglio Nazionale delle Ricerche di Bari, sotto la supervisione del Dott. Vito Renò. In particolare, la proposta di tesi è nata nell'ambito di una collaborazione dell'istituto con le associazioni di ricerca scientifica *Jonian Dolphin Conservation*<sup>2</sup>, finalizzata allo studio dei cetacei nel Golfo di Tarant

---

<sup>1</sup>Sistemi e Tecnologie Industriali Intelligenti per il Manifatturiero Avanzato, [stiima.cnr.it](http://stiima.cnr.it)

<sup>2</sup>[joniandolphin.it](http://joniandolphin.it)

---

to (Mar Ionio Settentrionale), e *Nova Atlantis Foundation*<sup>3</sup>, finalizzata allo studio dei cetacei nell'area delle isole Azzorre (Oceano Pacifico).



Figura 1.1: Logo della Jonian Dolphin Conservation



Figura 1.2: Logo della Nova Atlantis Foundation

Il principale risultato della collaborazione con *Jonian Dolphin Conservation*, presentato in [2], 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 piattaforma<sup>4</sup> 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.

Nei paragrafi seguenti si approfondiscono i motivi dell'interesse scientifico verso i cetacei, in seguito si fornisce una descrizione più dettagliata del problema affrontato ed infine si espongono lavori simili all'interno del panorama scientifico internazionale.

## 1.1 Interessi scientifici

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 riflette 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, poiché soggetto da anni ad un grave livello di inquinamento ambientale a causa degli insediamenti industriali presenti nella zona. Il monitoraggio degli ecosistemi marini è, in ogni caso, considerato necessario secondo alcune recenti direttive emesse dall'Unione Europea<sup>5</sup>.

---

<sup>3</sup>[nova-atlantis.org](http://nova-atlantis.org)

<sup>4</sup><http://dolphin.ba.issia.cnr.it>

<sup>5</sup>EU Marine Strategy Framework Directive ([https://ec.europa.eu/environment/marine/eu-coast-and-marine-policy/marine-strategy-framework-directive/index\\_en.htm](https://ec.europa.eu/environment/marine/eu-coast-and-marine-policy/marine-strategy-framework-directive/index_en.htm)), Ma-

## 1.2. FOTO-IDENTIFICAZIONE AUTOMATICA DEL *GRAMPUS GRISEUS*

---

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 atto. 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.

La foto-identificazione dei singoli individui nell'ambito dello studio dei cetacei è ritenuta di particolare utilità poiché consente di ottenere conoscenze più dettagliate sulle relazioni sociali all'interno della popolazione e sull'ambiente marino in cui gli esemplari vivono, senza arrecare disturbo agli stessi.

I delfini della specie *Grampus griseus* presentano numerose peculiarità che li rendono differenti dalle altre specie spesso associate all'immaginario comune, una su tutte quella del *Tursiops truncatus* (tursiope). Inoltre, presentano un tratto distintivo, descritto in dettaglio più avanti, che li rende particolarmente adatti ad essere studiati con la tecnica della foto-identificazione automatica. La fig. 1.3 mette a confronto le specie citate.

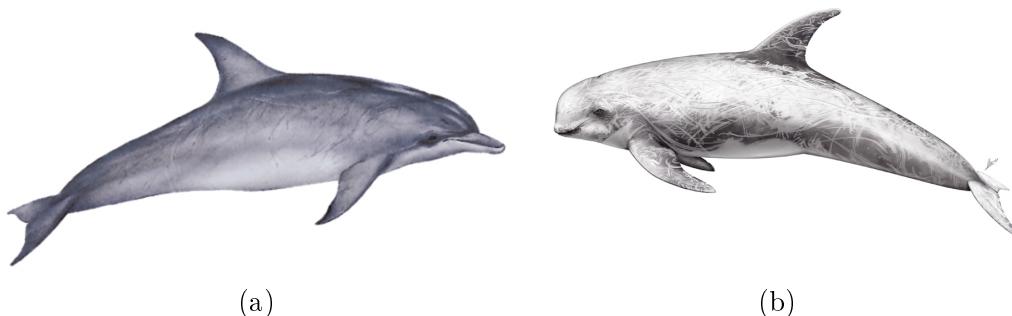


Figura 1.3: Illustrazione delle specie a confronto: (a) *Tursiops truncatus* (tursiope) (crediti: animalbase.uni-goettingen.de), (b) *Grampus griseus* (grampo) (crediti: associaciocetacea.org).

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 tursiope, 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.

---

ritime Spatial Planning Directive ([https://ec.europa.eu/maritimeaffairs/policy/maritime\\_spatial\\_planning\\_en](https://ec.europa.eu/maritimeaffairs/policy/maritime_spatial_planning_en))

---

Come si può ben notare nella fig.1.4, 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 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 1.4: Esemplare di grampo nel Golfo di Taranto (crediti: Emanuele Seller [3])

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 [2] già citato in precedenza, si è fatto uso di questa tecnica per stimare, a partire dalla grande quantità di immagini raccolta nel 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 stanziate nel Golfo di Taranto. Il risultato ottenuto si rivela di grande importanza poiché sopperisce ad una quasi totale assenza di dati: la IUCN *Red List of Threatened Species* ha considerato (fino a pochi anni fa) i grampi come specie "data deficient" nella zona del Mar Mediterraneo<sup>6</sup>.

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. 1.5).

---

<sup>6</sup><https://www.iucnredlist.org/species/16378423/16378453>

## 1.2. FOTO-IDENTIFICAZIONE AUTOMATICA DEL *GRAMPUS GRISEUS*

---

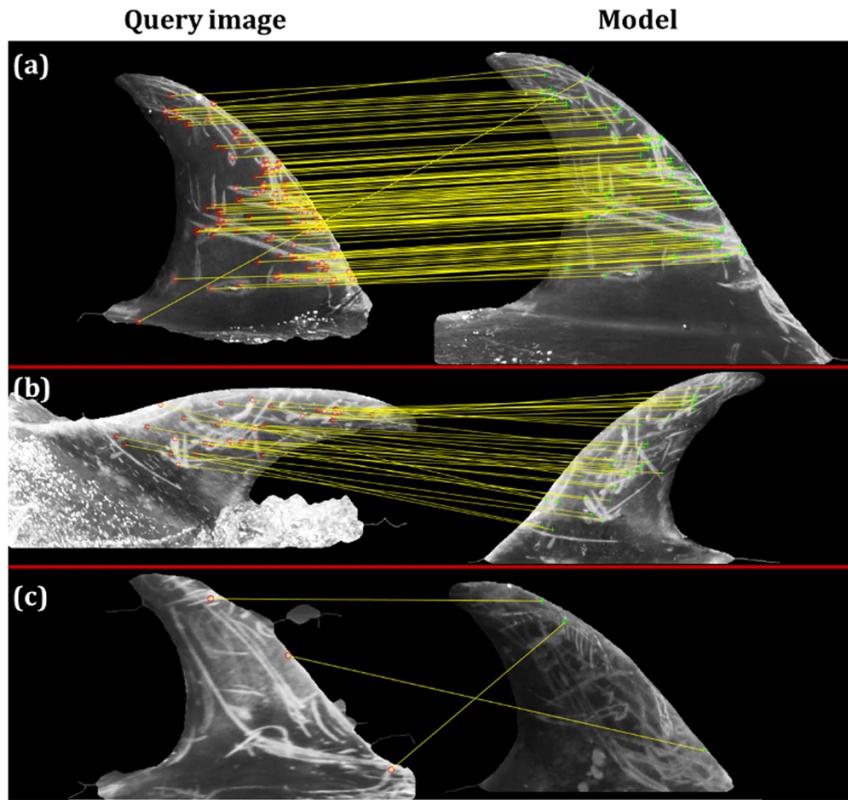


Figura 1.5: Esempio di output dell’algoritmo *SPIR*: nell’ordine predizione corretta, predizione corretta di una pinna ruotata, predizione non corretta. Immagine tratta da [2].

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 (*big data*) 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. 1.6).

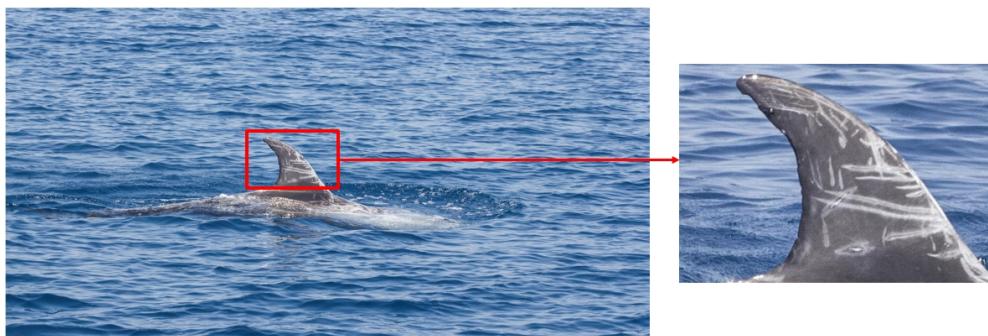


Figura 1.6: Ritaglio di una pinna dorsale di un delfino da un’immagine.

---

La fase del ritaglio, se fatta manualmente, risulta infatti un collo di bottiglia all'interno della procedura di foto-identificazione automatica, essendo particolarmente dispendiosa in termini di tempo.

L'algoritmo CropFin v1, presentato in [1] e basato su un precedente lavoro [4] ha avuto l'obiettivo di automatizzare la fase di ritaglio delle pinne dorsali a partire dalla collezione delle immagini scattate in mare, consacrando 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.

### 1.3 Problema ed obiettivi

Il presente lavoro di tesi vuole porsi come una naturale prosecuzione rispetto ai lavori [1] e [4]. Il lavoro è consistito nell'apportare un miglioramento delle prestazioni della fase di classificazione dei ritagli previsto dalla routine CropFin v1. La principale tecnica utilizzata per apportare questo miglioramento è il *transfer learning* (par. 2.10), il cui studio è centrale nel presente elaborato.

Il risultato del lavoro è stato la creazione di una routine CropFin migliorata rispetto alla sua versione v1, con prestazioni nettamente superiori ed in grado di rendere più fine ed efficace la discriminazione tra immagini contenenti pinne e quelle invece prive di valore informativo.

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* (par. 2.1) al *Machine Learning* (par. 2.4).

Nel cap. 2 vengono descritti le metodologie ed i concetti teorici utilizzati all'interno degli esperimenti svolti; il cap. 3 descrive dettagliatamente gli esperimenti svolti; nel cap. 4 sono espuse considerazioni finali sulle soluzioni proposte, e vengono presentati alcuni sviluppi futuri e miglioramenti apportabili alla routine messa a punto; infine, nel cap. 5 si riporta il codice sorgente Matlab scritto ed adoperato negli esperimenti.

Questo lavoro può essere inquadrato nell'ambito di una disciplina emergente al confine tra ecologia e informatica, nota come *ecological informatics*. Come riporta l'omonima rivista scientifica<sup>7</sup>, essa si fonda sull'incontro tra la grande quantità di dati intrinsecamente connessa all'ecologia e la crescente capacità delle tecnologie dell'informazione di analizzare grandi moli di dati (*big data*), talvolta complessi, con l'obiettivo di stimolare ed accelerare l'avvento dei processi sostenibili in vista dei cambiamenti climatici e ambientali globali.

---

<sup>7</sup> *Ecological Informatics*, Elsevier

## 1.4 Lavori simili

Negli ultimi anni, molte tecniche di computer vision e machine learning, che operano sinergicamente in una nuova disciplina chiamata *visual data analytics*, stanno assumendo un ruolo sempre più importante nelle strategie di monitoraggio e conservazione della fauna del nostro pianeta.

In questo contesto si inserisce il presente lavoro di tesi, che migliorando le prestazioni della fase che precede la foto-identificazione dei grampi vuole dare un contributo concreto all'importante causa della salvaguardia degli ecosistemi.

Sono molti i lavori affini a quello presentato in questo elaborato. La maggior parte di essi sono di recente pubblicazione o ancora in fase di sviluppo. In questo paragrafo se ne presentano alcuni, ritenuti particolarmente significativi anche in virtù del loro uso del *transfer learning*, tecnica su cui, come detto, si incardina il presente lavoro di tesi. L'impiego profuso di tale tecnica in queste pubblicazioni ne conferma l'efficacia e ne suggerisce l'utilizzo anche per risolvere il nostro task di classificazione delle pinne.

**A Hybrid Approach for Tiger Re-Identification [5].** Questo lavoro si inserisce nel contesto della "*Amur tiger re-identification (Re-ID) challenge*", tenuto dal workshop *Computer Vision for Wildlife Conservation* nell'ambito della *International Conference on Computer Vision 2019*. Il suo obiettivo è la creazione di un sistema di foto-identificazione degli esemplari di tigre siberiana (*Panthera tigris altaica*), reso possibile dal particolare manto striato della specie, il cui pattern è univoco per ogni individuo, e che pertanto rende la specie particolarmente adatta alla foto-identificazione.

Alcune criticità (dataset di immagini a disposizione limitato e di bassa qualità, variazioni nell'esposizione alla luce e nella posa degli esemplari e numero limitato di immagini di ciascun esemplare) rendono la foto-identificazione di questa specie un task difficile per i modelli di deep learning. Conseguentemente, si è proposto l'utilizzo sia delle tecniche e dei modelli del deep learning e sia della tradizionale tecnica di matching basata sui descrittori SIFT.

Il modello di classificazione progettato è una rete neurale convoluzionale profonda, DenseNet-121 [6], ri-addestrata con la tecnica del *transfer learning*. Numerose operazioni di *image augmentation* sono state adoperate per incrementare la dimensione e l'eterogeneità del dataset a disposizione, migliorando la capacità di generalizzazione della rete. In fig. 1.7 è riportato il *workflow* del sistema di foto-identificazione oggetto del paper.

**Individual Minke Whale Recognition Using Deep Learning Convolutional Neural Networks [7].** La balenottera minore o balenottera rostrata (*Balaenoptera acutorostrata*) abita un'area molto ristretta a nord della Grande Barriera Corallina australiana, durante la stagione invernale (maggio-agosto). È stato osservato che i pattern di colore di queste balene sono univoci per ogni individuo e rimangono stabili nel corso di molti anni, e sono sufficientemente complessi da rendere questa specie adatta alla foto-identificazione non equivoca degli esemplari (fig. 1.8).

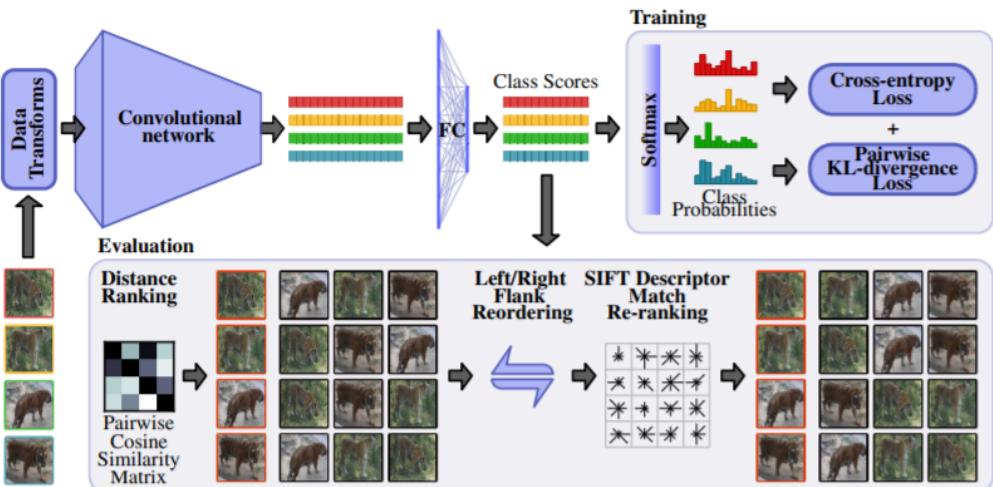


Figura 1.7: Workflow del sistema di foto-identificazione delle tigri siberiane sviluppato in [5]. Immagine riprodotta dal paper originale.

L’identificazione degli individui di questa specie è necessaria per studiare le caratteristiche della popolazione e per monitorare l’impatto che il turismo marittimo ha su di essa. Esiste un enorme database di immagini non etichettate e non foto-identificate di balenottere minori, raccolte dalle numerose imbarcazioni turistiche che operano nell’areale della specie. Questa grande mole di dati non si presta ad una foto-identificazione manuale da parte dei biologi, e si è pertanto pensato di mettere a punto un sistema per la sua automatizzazione, ricorrendo all’uso della tecnica del *transfer learning*. Una rete VGG16 pre-addestrata sul database ImageNet (par. 2.3.3) è stata ri-addestrata sul dataset a disposizione, per operare una segmentazione semantica delle immagini. Per migliorare la capacità di generalizzazione della rete sono state adoperate tecniche di *image augmentation*.

Il lavoro ha portato alla creazione della routine di foto-identificazione "*Automatic Minke Whale Recognizer (AMWR)*", che raggiunge ottime prestazioni ( $>90\%$ ) sui test set.



Figura 1.8: Un esemplare di Balaenoptera acutorostrata. Si notino i pattern di colore della pinna e le numerose "cicatrici", caratteristiche univoche di ogni esemplare.

## 1.4. LAVORI SIMILI

---

**Robust Re-identification of Manta Rays from Natural Markings by Learning Pose Invariant Embeddings [8].** In questo articolo viene presentato un nuovo metodo di foto-identificazione basato sui segni naturali univoci degli animali e robusto ad occlusioni, punti di vista e cambi di illuminazione. Il metodo è creato riadattando metodi già utilizzati per la foto-identificazione dei volti umani, ed in particolare riaddestrando la rete neurale convoluzionale FaceNet [9] mediante la tecnica del *transfer learning*. Questo nuovo metodo di foto-identificazione è generico e non specifico di una certa specie. Il sistema è stato testato su un database di immagini di ventri di mante (Manta alfredi e Manta birostris) e su un database di pinne caudali di megattere (*Megaptera novaeangliae*), registrando prestazioni ottime.

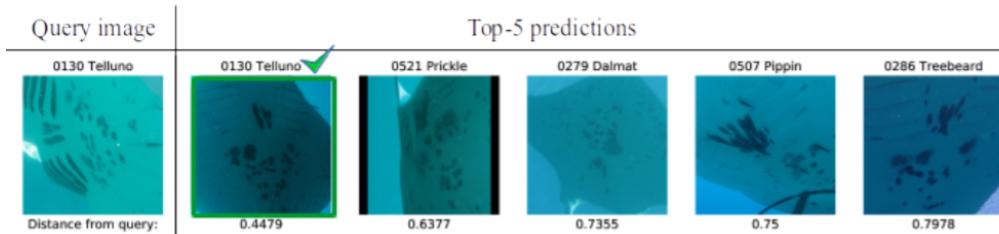


Figura 1.9: Esempio di foto-identificazione di una manta avvenuta con successo. I segni scuri sul ventre della manta sono univoci per ogni esemplare, e permettono la foto-identificazione non equivoca.



# Capitolo 2

## Metodologie

Nel corso di questo capitolo saranno introdotti progressivamente i concetti principali su cui si fonda la *computer vision* (visione artificiale) e il *machine learning* (apprendimento automatico). Questi contenuti permetteranno di comprendere da un punto di vista teorico quanto descritto nella sezione sperimentale della tesi (cap. 3).

Per i primi paragrafi, dedicati all'*image processing*, le principali fonti seguite sono [1] e [10].

Per i paragrafi dedicati al *machine* e *deep learning* le fonti di riferimento sono [11] e soprattutto [12].

### 2.1 Immagini digitali

Caratterizziamo intuitivamente il concetto di "immagine" dal punto di vista informatico.

Un'immagine digitale è una rappresentazione binaria di un'immagine (in generale a colori) a due dimensioni<sup>1</sup>; essa può essere definita matematicamente come un tensore  $\mathcal{I} \in \{0, \dots, 255\}^{h \times w \times c}$ , dove  $h$  e  $w$  sono rispettivamente dette **, la coppia  $(w, h)$   **mentre  $c$  è il numero di canali di colore<sup>2</sup>. Nello spazio di colore sRGB (standard RGB, d'ora in avanti abbreviato in RGB), ampiamente adoperato, i canali di colore sono rosso (R, Red), verde (G, Green) e blu (B, Blue), quindi  $c = 3$ . In mancanza di diverse indicazioni, ci si riferirà nel seguito allo spazio di colore RGB.****

Un pixel  $p(i, j)$  è definito come la funzione vettoriale

$$p(i, j) = [r(i, j), g(i, j), b(i, j)]$$

essendo  $r, g, b : \{0, \dots, h\} \times \{0, \dots, w\} \rightarrow \{0, \dots, 255\}$  le funzioni scalari che associano ad ogni posizione bidimensionale  $i, j$  dell'immagine un valore intero di  **compreso tra 0 e 255, uno per ciascuno dei tre canali RGB. Ogni pixel definisce univocamente un colore nello spazio RGB, il quale può quindi rappresentare in tutto  $256^3$  colori diversi, cioè circa 17 milioni.**

<sup>1</sup>Ci riferiamo in questa sede solo alle immagini di tipo raster, quelle cioè con risoluzione e numero di canali di colore fissati a priori, come ad esempio le immagini digitali in formato jpg.

<sup>2</sup>Spesso si scrive che  $\mathcal{I}$  è un immagine  $w \times h \times c$ , o più semplicemente  $w \times h$  (assumendo  $c = 3$ )

Si può immaginare il tensore immagine  $\mathcal{I}$  come una "pila" di tre matrici, una per ogni canale di colore, come mostrato in figura 2.1.

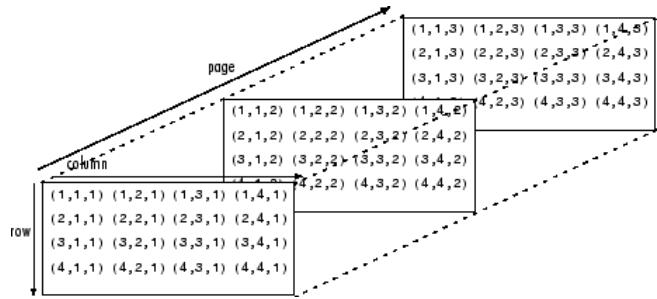


Figura 2.1: Rappresentazione grafica di un tensore tridimensionale; in ogni posizione compaiono gli indici del tensore

Un esempio di immagine digitale, scomposta nelle sue componenti RGB, è mostrato in figura 2.2.

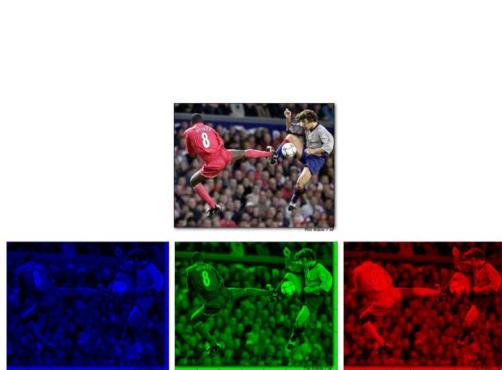


Figura 2.2: Canali RGB di un'immagine

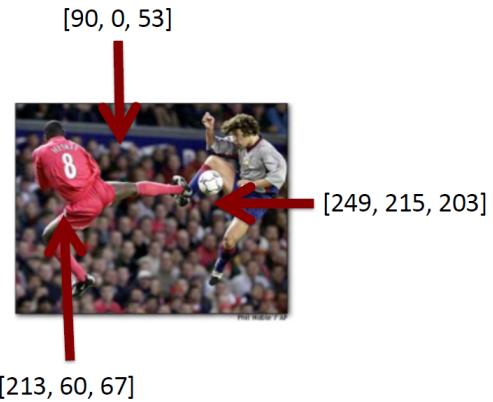


Figura 2.3: Pixel di un'immagine

In Matlab un'immagine digitale può essere rappresentata con il tipo di dato **multidimensional-array** con tre dimensioni (corrispondenti ai tre canali di colore RGB), in cui ciascun elemento è di tipo **uint8** (ma può appartenere anche ad altri tipi di dato<sup>3</sup>).

È possibile importare un'immagine RGB con la funzione:

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

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

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

Il pixel di posizione  $(i, j)$  è quindi:

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

<sup>3</sup>[https://it.mathworks.com/help/matlab/ref/image.html?s\\_tid=doc\\_ta#buqdlnb-C](https://it.mathworks.com/help/matlab/ref/image.html?s_tid=doc_ta#buqdlnb-C)

## 2.1. IMMAGINI DIGITALI

---

### 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. 3.2.1) si utilizza una conversione delle immagini dallo spazio di colore iniziale RGB allo spazio di colore **CIE 1976 L\*a\*b\*** (nel seguito abbreviato come Lab). 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 crominanza (variazione di colore), rispettivamente una scala verde-rosso ( $a^*$ ) ed una scala blu-giallo ( $b^*$ ).

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

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

I dettagli sulla conversione di spazio di colore possono essere letti al par. 2.1.1 di [1].

### 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 messi a disposizione da Matlab e utilizzati negli esperimenti del presente lavoro di tesi:

- `imageDatastore`: oggetto progettato appositamente per gestire ed elaborare rapidamente una grande quantità di immagini. Per istanziare un oggetto `imageDatastore` bisogna specificare l'argomento `path` che indica il percorso della collezione di immagini da importare. Altri argomenti (coppie argomento-valore) opzionali per l'inizializzazione di questo oggetto sono:

- `'IncludeSubfolders'`,`true`  
Include le immagini contenute nelle sottocartelle di `path`
- `'LabelSource'`,`'foldernames'`  
Assegna a ciascuna immagine un'etichetta data dal nome della cartella in cui è contenuta

Quindi, per creare l'oggetto:

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

L'elenco delle immagini è restituito nel campo `imds.Files`.

- `augmentedImageDatastore`: oggetto creato a partire da un `imageDatastore` applicando operazioni di preprocessing e augmentation specificate in un oggetto `imageDataAugmenter`. La sintassi di questo oggetto verrà approfondita nel par. 2.8.

---

## 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, una strategia chiamata *image augmentation* (par. 2.8).

### 2.2.1 Ridimensionamento

Il ridimensionamento consente, in generale, di ottenere una nuova immagine a risoluzione differente, riducendo o aumentando il numero di pixel utilizzati per rappresentarla. Nel caso del presente lavoro, le dimensioni delle immagini sono state ridotte (par. 3.2.1), per diminuire il costo computazionale delle operazioni successive. Per ridimensionare un'immagine `im` in Matlab si può usare la funzione `imresize`<sup>4</sup>, specificando la nuova lunghezza `w'` e la nuova altezza `h'`:

```
im_res = imresize(im, [h' w']);
```

### 2.2.2 Rotazione

La rotazione di un'immagine digitale può essere effettuata calcolando per ogni suo pixel  $(x, y)$  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  $\alpha$  è l'angolo di rotazione misurato in senso antiorario rispetto all'asse x e  $(x', y')$  le coordinate del pixel trasformato. L'immagine ruotata è l'insieme dei pixel  $(x', y')$  calcolati.

In Matlab, la rotazione di un'immagine `im` di un angolo `a` può essere effettuata con

```
rotated = imrotate(im, a);
```

### 2.2.3 Riflessione

La riflessione rispetto all'asse x di un'immagine digitale può essere effettuata calcolando per ogni suo pixel  $(x, y)$  il prodotto matriciale

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

dove  $(x', y')$  sono le coordinate del pixel trasformato. L'immagine riflessa è l'insieme dei pixel  $(x', y')$  calcolati.

In Matlab, la riflessione di un'immagine `im` rispetto all'asse x può essere effettuata con

```
flipped = flipdim(im, 2);
```

---

<sup>4</sup>La funzione `imresize` attua, per lo scopo, una tecnica avanzata di calcolo numerico: l'interpolazione bicubica. (par. 2.2.1 di [1] per i dettagli.)

### 2.2.4 Traslazione

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

$$[x \ y]^{\top} \mapsto [x \ y \ 1]^{\top}$$

A questo punto, è possibile ottenere le nuove coordinate traslate  $(x', y')$  calcolando:

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

In Matlab, la traslazione orizzontale di  $t$  pixel di un'immagine `im` può essere effettuata con

```
translated = imtranslate(im, [t 0]);
```

dove  $t > 0$  implica una traslazione verso destra,  $t < 0$  verso sinistra.

I risultati delle quattro trasformazioni finora viste sono visualizzate in figura 2.4

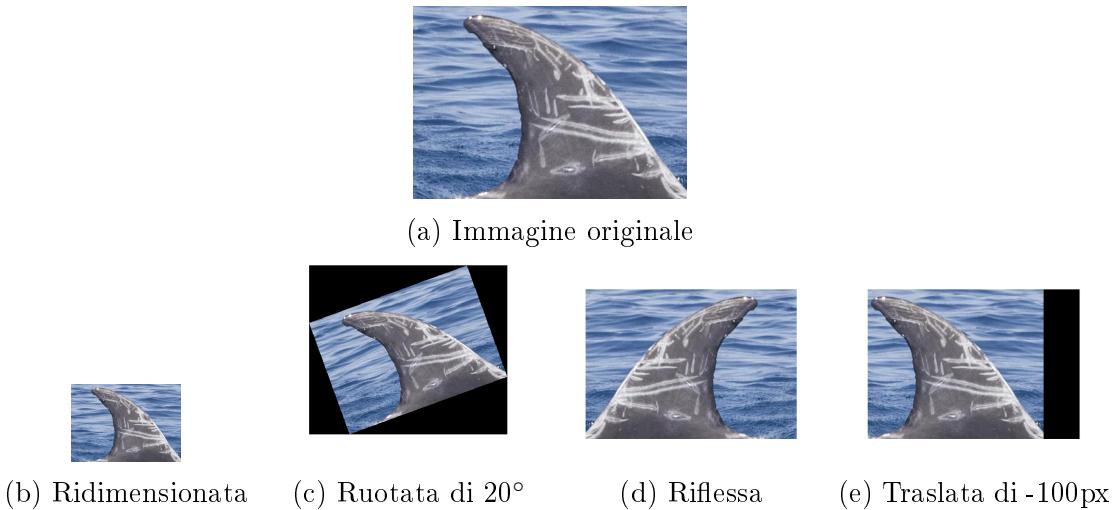


Figura 2.4: Trasformazioni applicate ad un'immagine digitale

### 2.2.5 Convoluzione

Nell'ambito dell'*image processing* esiste una quantità notevole di operatori definiti "locali", che operano cioè non su un singolo pixel ma su un gruppo di pixel contigui. L'operatore locale maggiormente usato è l'operatore di convoluzione. Nell'ambito del presente lavoro, esso ha un'importanza centrale per la *feature extraction* delle immagini, all'interno delle reti neurali convoluzionali (par. 2.7), e più specificamente nei layer convoluzionali (par. 2.7.2).

Nel seguito si presenta, pertanto, una definizione rigorosa dell'operazione di convoluzione e si forniscono semplici esempi di implementazione.

---

Dati due segnali discreti  $x(k)$  e  $w(k)$ , si definisce **(somma di) convoluzione tra  $x$  e  $w$**  una nuova funzione  $s(k)$  definita come

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

Si dimostra che questa operazione gode della proprietà commutativa, associativa e distributiva rispetto alla somma.

Nell'ambito della *computer vision* si utilizza una versione multidimensionale dell'operazione di convoluzione, utilizzando la seguente terminologia:

- 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**, poiché l'operazione di convoluzione è spesso utilizzata per l'estrazione di feature a partire dall'input (par. 2.7.2).

È ovvio 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  $I \in \mathbb{R}^{h \times w}$ , anche il kernel impiegato è solitamente una matrice bidimensionale di dimensioni ridotte  $K \in \mathbb{R}^{a \times b}$ , ottenendo l'operazione:

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

Molte librerie, in realtà, implementano la convoluzione attraverso la funzione di crosscorrelazione (indicata con  $\circledast$ ). In tal caso ciascun elemento della feature map si ottiene come

$$s(i, j) = (I \circledast K)(i, j) = \sum_{k=1}^a \sum_{r=1}^b I(i+k-1, j+r-1)K(k, r)$$

L'operazione appena esposta - che si dimostra essere equivalente ad una convoluzione tra  $I$  e  $K$  - ha una semplice interpretazione, visualizzata in fig. 2.5. Convolvere un'immagine con un kernel equivale a far scorrere la matrice che rappresenta il kernel lungo l'immagine, e sviluppare i prodotti *element-wise* (termine a termine) e sommarli tra loro, ottenendo l'elemento della feature map.

Nel caso (più comune) in cui l'input sia un tensore (ad esempio un'immagine a colori)  $\mathcal{I} \in \mathbb{R}^{h \times w \times c}$  si richiede che il kernel abbia lo stesso numero  $c$  di livelli, ad esempio  $K \in \mathbb{R}^{a \times b \times c}$ . L'operazione viene così ridefinita.

$$s(i, j) = (\mathcal{I} \circledast K)(i, j, k) = \sum_{k=1}^a \sum_{r=1}^b \sum_{s=1}^c \mathcal{I}(i+s-1, j+r-1, k)K(r, s, k)$$

Come si nota, la feature map ottenuta sarà sempre bidimensionale (a prescindere dalla profondità  $c$  del tensore in input).

Completeremo la trattazione sulla convoluzione nel par. 2.7.2 sul layer convoluzionale di una CNN.

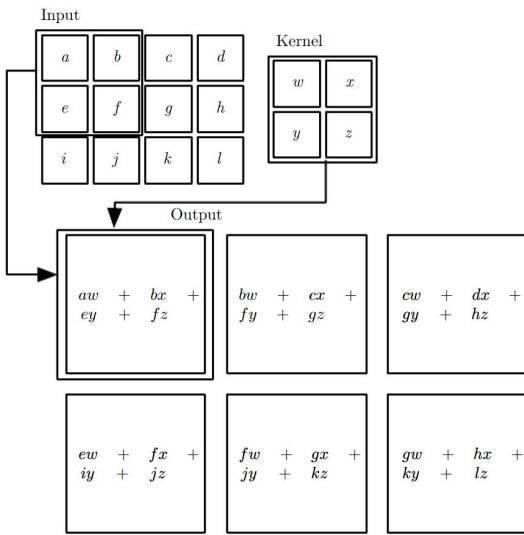


Figura 2.5: Un esempio di cross-correlazione (ovvero convoluzione discreta 2-D senza il ribaltamento del kernel)

## 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 Object recognition

L'*object recognition* (in italiano: riconoscimento di oggetti) nell'ambito della visione artificiale è il problema di assegnare una descrizione testuale o una o più etichette ad un'immagine, tipicamente sulla base di uno o più determinati oggetti che un computer riesce a riconoscere all'interno di essa.

Ogni categoria esistente di oggetti ha delle caratteristiche fondamentali che la differenziano da qualunque altra categoria di oggetti. Attraverso tecniche di *machine learning* è possibile ricavare una descrizione di una certa categoria addestrando la macchina a riconoscere le caratteristiche (*features*) fondamentali di quella categoria di oggetti, a partire da un insieme di immagini campione afferenti a quella categoria.

Per rendere affidabile il riconoscimento, è importante che l'insieme di caratteristiche estratte da ogni immagine campione sia insensibile a variazioni del punto di vista, di scala, delle condizioni di illuminazione, alle distorsioni geometriche, all'occlusione dell'oggetto, al *clutter* (ingombro) di altri oggetti non informativi sullo sfondo, alle variazioni intra-classe dell'oggetto, come mostrato in fig. 2.6.

L'uomo riconosce una moltitudine di oggetti in immagini con poco sforzo, nonostante i fattori di variabilità descritti. Questo compito è ancora una sfida aperta per la computer vision in generale.

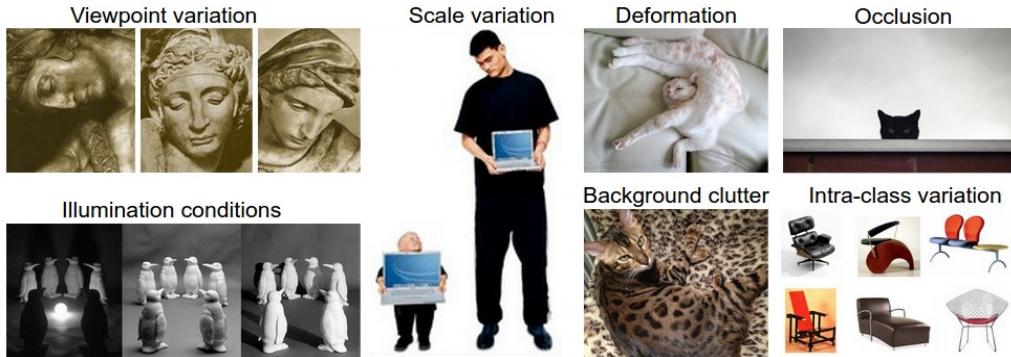


Figura 2.6: I principali "ostacoli" al riconoscimento automatico degli oggetti

Il problema dell'*image classification* è uno specifico problema di *object recognition* che consiste nell'assegnare una singola etichetta (o una distribuzione di probabilità su più etichette) ad un'immagine da un insieme fisso di etichette (anche dette "categorie" o "classi"). In fig. 2.7 è visualizzato il problema in esame.

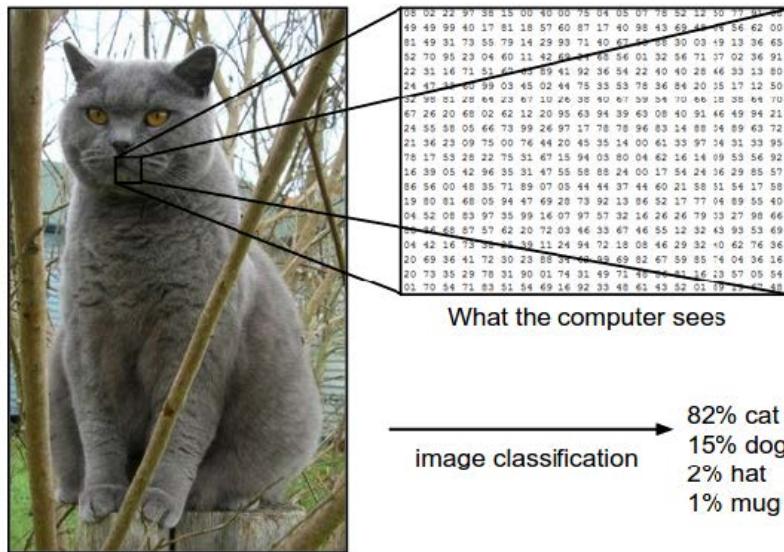


Figura 2.7: Il task della *image classification* consiste in questo caso nel calcolare una distribuzione di probabilità su quattro etichette (gatto, cane, cappello, tazza) per un'immagine digitale.

In letteratura sono stati proposti numerosi metodi per la risoluzione efficiente dei task di *object recognition*, attraverso l'impiego di diverse tecniche di *machine learning* (ad esempio l'algoritmo di clustering *k-Nearest Neighbor*). Negli ultimi anni i risultati più promettenti sono stati offerti dalle *reti neurali convoluzionali*, di cui parleremo diffusamente nel seguito del capitolo (par. 2.7) e che verranno impiegate negli esperimenti (par. 3.2.2).

### 2.3.2 Object detection

Un altro importante problema di *computer vision* è l'***object detection*** (in italiano: rilevamento di oggetti). Esso consiste nella localizzazione di oggetti di categorie stabilite a priori ed in seguito (o talvolta in contemporanea) la loro classificazione, per mezzo di un opportuno modello di classificazione.

Il compito si rivela, evidentemente, più difficile di quello della semplice classificazione di oggetti, essendo la localizzazione degli oggetti all'interno dell'immagine un problema anch'esso non banale.

L'obiettivo di questo lavoro di tesi è la risoluzione di una particolare istanza (*task*) di *object detection*: si vogliono identificare, localizzare e conseguentemente ritagliare, le eventuali porzioni di un'immagine contenenti pinne dorsali di delfini. La classe di oggetti di interesse è quindi una sola.

Grazie al dominio ristretto del problema in esame, il task di rilevamento è risolto con un approccio in due fasi:

- dapprima, si localizzano e si ritagliano le eventuali pinne presenti nell'immagine, sfruttando una forma di conoscenza di alto livello direttamente disponibile nella rappresentazione delle immagini: il colore<sup>5</sup>;
- in seguito, i ritagli sono sottoposti ad una fase di classificazione, che ne conferma la natura di 'Pinna' o ne smentisce il contenuto informativo ('No pinna').

Si rimanda direttamente al cap. 3 per la descrizione degli esperimenti condotti.

### 2.3.3 ImageNet Database e ILSVRC

***ImageNet*** è un'ampia base di dati di immagini, realizzata per l'utilizzo nel campo dell'*object recognition* [13]. Il dataset consiste in più di 14 milioni di immagini con diverse risoluzioni che sono state annotate manualmente con l'indicazione degli oggetti in esse rappresentati e, per circa un milione di esse, della bounding box che li delimita. Gli oggetti individuati sono stati classificati in più di 22.000 categorie. Alcune categorie di oggetti frequenti, come ad esempio "pallone" o "fragola", consistono di diverse centinaia di immagini. Le immagini sono state raccolte nel web ed etichettate manualmente tramite il tool di crowd-sourcing Amazon's Mechanical Turk.

A partire dal 2010, ogni anno viene indetta una competizione di *object recognition* e *detection* denominata ***ImageNet Large Scale Visual Recognition Challenge*** (ILSVRC): in tale occasione programmi software vengono fatti competere per classificare e rilevare correttamente oggetti e scene contenuti nelle immagini.

Nell'ambito della competizione viene impiegata un sottoinsieme di immagini del database ImageNet, con immagini appartenenti a 1000 categorie e circa 1000 immagini per ciascuna categoria. In tutto ci sono circa 1.2 milioni di immagini per l'addestramento dei modelli, 50.000 per la validazione e 150.000 per il testing.

---

<sup>5</sup>L'idea di sfruttare il colore per isolare le pinne dei cetacei deriva proprio dalla differenza di tonalità tra l'acqua (tendente al blu e al verde) e le pinne (tendenti al grigio)

---

Per i classificatori che competono nella ILSVRC le prestazioni vengono misurate sulla base di due parametri:

- *top-5 error*: la frazione di immagini del test set per le quali la classe corretta non è tra le cinque classi considerati le più probabili dal modello
- *top-1 error*: la frazione di immagini del test set classificate in maniera errata dal modello

Le tre reti neurali convoluzionali scelte per effettuare gli esperimenti presentati in questo lavoro sono risultate vincitrici della ILSVRC 2012 (Alexnet, par. 2.12), ILSVRC 2014 (GoogLeNet, par. 2.13) e ILSVRC 2015 (ResNet, par. 2.14).

## 2.4 Machine Learning e Deep Learning

Il **machine learning** è una branca dell'intelligenza artificiale (AI) che si occupa in generale di fornire ad una macchina la capacità di apprendere automaticamente dall'esperienza, essendo esplicitamente programmata su "come imparare" ma non su "cosa imparare".

Questa capacità è fornita da un certo algoritmo di machine learning, di cui T. Mitchell fornisce una celebre ed elegante 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.

L'obiettivo principale dell'apprendimento automatico è che una macchina sia in grado di generalizzare dalla propria esperienza[14], ossia che sia in grado di svolgere ragionamenti induttivi. In questo contesto, per generalizzazione si intende l'abilità di una macchina di portare a termine in maniera accurata esempi o compiti nuovi, che non ha mai affrontato, dopo aver fatto esperienza su un insieme di dati di apprendimento. Gli esempi di addestramento (in inglese chiamati *training examples*) si assume provengano da una qualche distribuzione di probabilità, generalmente sconosciuta e considerata rappresentativa dello spazio delle occorrenze del fenomeno da apprendere; la macchina ha il compito di costruire un modello probabilistico generale dello spazio delle occorrenze, in maniera tale da essere in grado di produrre previsioni sufficientemente accurate quando sottoposta a nuovi casi.

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

Un algoritmo di deep learning si basa su diversi livelli di rappresentazione dei dati, che corrispondono a differenti livelli di astrazione; questi livelli formano una gerarchia di concetti, in cui i concetti di più alto livello sono definiti sulla base di quelli di livello più basso.

Il modello computazionale utilizzato in maniera esclusiva nel *deep learning* è la rete neurale artificiale (par. 2.6)

Il deep learning è inquadrato in una più ampia branca del machine learning, chiamata **representation learning**. Nell'ambito del representation learning, l'approccio usato per la risoluzione dei problemi consiste non solo nel tentare di addestrare un computer a trovare una relazione tra dati e output, ma anche nel fornirgli strumenti per la rappresentazione stessa dei dati, in quanto in alcuni contesti il legame tra i dati in input e quelli attesi in output è tutt'altro che lineare ed è in generale complesso, e può essere più facile per la macchina acquisire diversi livelli di conoscenza e di astrazione dei dati in input per calcolare l'output.

Si pensi ad esempio ad un task di *image classification*. È impensabile descrivere la presenza di un oggetto all'interno di un'immagine attraverso un legame lineare con

---

i singoli pixel. È piuttosto la combinazione di pixel l'informazione da mappare (in maniera in generale non lineare) con la categoria di appartenenza di quell'oggetto.

Dagli studi scientifici sull'apparato visivo sappiamo che l'uomo riesce a riconoscere gli oggetti attraverso una rappresentazione gerarchica degli stessi:

- dapprima captiamo caratteristiche di basso livello degli oggetti che vediamo, come forme (spigoli, angoli) e tonalità di colore. Tutte queste caratteristiche sono "locali", nel senso che occupano una regione limitata del campo visivo, essendo parti più piccole dell'oggetto. In questa fase, non sappiamo ancora dare un nome all'oggetto su cui ci concentriamo;
- queste caratteristiche (*features*) sono combinate nella parte del cervello che si occupa della visione, a formare concetti di più alto livello come il perimetro dell'oggetto e le sfumature (gradienti) di colore;
- questa rappresentazione graduale e gerarchica dei concetti, arrivata a concetti di più alto livello, permette infine di associare alla "immagine" formatasi nel nostro campo visivo il nome dell'oggetto, o degli oggetti, in esso presenti.

Sulle medesime basi "biologiche" si fondano le reti neurali artificiali, attraverso gli *hidden layers* (strati nascosti) frapposti tra l'input e l'output della rete stessa che permettono di combinare le informazioni provenienti dagli strati precedenti per ottenere una rappresentazione dei dati di più alto livello.

#### 2.4.1 Supervised Learning

Il paradigma dell'**apprendimento supervisionato** (*supervised learning*) mira alla creazione di un algoritmo che analizzi dei "dati di addestramento", una collezione di esempi ideali costituiti da coppie di input e relativi output attesi, e da questi inferisca una funzione che può essere usata per mappare nuovi input ai corretti output [15]. Ciò richiede all'algoritmo la capacità di trovare una funzione che sappia generalizzare efficacemente dai dati di training, al fine di adattarsi bene a nuovi dati (per poterne mappare correttamente quanti più possibile).

Molti problemi pratici, come ad esempio la regressione e la classificazione delle immagini (par. 2.3.1, possono essere formulati ricorrendo ad una funzione matematica

$$\mathcal{F} : X \rightarrow Y$$

che associa ad ogni elemento dello spazio degli input  $X$  uno ed un solo elemento dello spazio degli output  $Y$ . Il concetto di funzione implica l'esistenza di un solo elemento di  $Y$  a cui ogni elemento di  $X$  è correttamente associato. Il problema consiste allora nel cercare una funzione  $\mathcal{F}$  in grado di ottenere esattamente tale associazione, per quanti più elementi di  $X$  possibile.

È evidente che questo tipo di problemi ben si presta ad essere approcciato con algoritmi di apprendimento supervisionato.

Prima di analizzare in dettaglio il problema di classificazione delle immagini oggetto della presente tesi, è necessario inquadrare il problema partendo da alcune

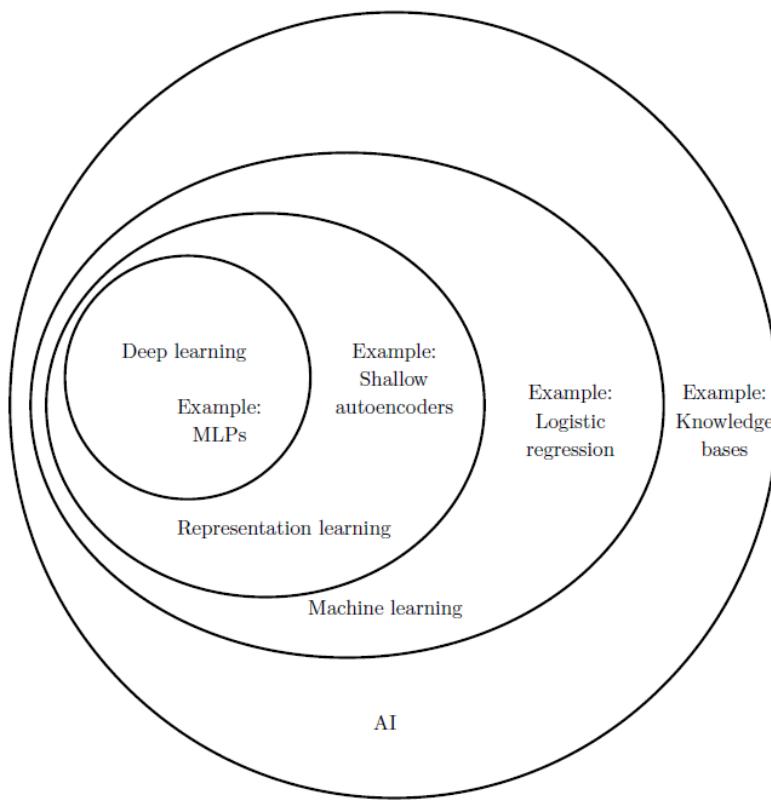


Figura 2.8: Un diagramma di Venn che mostra come il deep learning sia un tipo di representation learning, che a sua volta è un tipo di machine learning.

definizioni preliminari.

Un **dataset**  $X$  è una generica collezione di  $N$  dati

$$X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$$

Ogni dato  $\mathbf{x}^{(i)}$  è chiamato **esempio** (o **data point**). I data point possono essere anche non omogenei tra loro (cioè avere dimensioni differenti). Ciascun esempio si può caratterizzare come un vettore  $\mathbf{x}^{(i)} \in \mathbb{R}^D$ , in cui ciascun elemento  $x_i$  è detto **feature** e rappresenta una caratteristica di un oggetto o un evento misurato.  $D$  è il numero di feature in ogni esempio, o **dimensione** dell'esempio. In caso di esempi omogenei (cioè aventi stessa dimensione  $D$ ) un dataset può essere descritto attraverso una matrice detta **design matrix**, in cui ogni riga corrisponde ad un particolare esempio e ogni colonna corrisponde ad una precisa feature. Un dataset di cardinalità  $N$  e in cui ogni esempio ha  $D$  feature ha quindi una design matrix di dimensione  $N \times D$ .

In un problema di classificazione delle immagini sussiste la seguente caratterizzazione:

- $X$ : un insieme di  $N$  immagini digitali

- 
- $Y$ : un insieme di  $K$  classi predefinite di oggetti che possono essere individuati all'interno di un'immagine (possono essere dei "descrittori" testuali o, equivalentemente, dei numeri interi)

Un elemento di  $Y$  è solitamente chiamato **etichetta** o **categoria** (in inglese **label** o **class**); si dice quindi che ogni immagine  $\mathbf{x}^{(i)} \in X$  può essere *descritta da un'etichetta* (o *associata ad una categoria*)  $\mathbf{y}^{(i)} \in Y$  tramite una funzione di associazione  $f$ .<sup>6</sup>

Tipicamente, per l'addestramento di un modello di machine learning si usa un sottoinsieme del dataset  $X$  a disposizione. Questo sottoinsieme  $X^{train}$  è definito **training set**.

Nella pratica, l'espressione analitica di  $f$  non può essere trovata esattamente. Ad esempio, nel problema 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, poiché il concetto di "pinna" non è un concetto matematico e non può essere reso facilmente in un linguaggio "comprendibile" da un computer. Inoltre, il computer può disporre solamente di un numero limitato  $|X^{(train)}|$  di esempi, cioè un numero limitato di occorrenze di oggetti di tipo "pinna", che non permettono di generalizzare a tutte le forme ed i colori in cui una pinna può essere presente in un'immagine. Per questo motivo, ciò che il modello di machine learning da addestrare è chiamato ad imparare è un'approssimazione quanto più "plausibile" di  $f$ , dove per "plausibilità" si intende la capacità della funzione trovata di restituire il giusto output per il maggior numero possibile di input presentati.

## 2.4.2 Underfitting e overfitting

La sfida principale dell'apprendimento automatico è quella di rendere l'algoritmo di machine learning performante su nuovi input, diversi da quelli su cui il modello è stato addestrato. Questa abilità è chiamata **generalizzazione**.

Tipicamente, dopo l'addestramento di un modello di machine learning con un *training set* possiamo misurare le performance del modello con un parametro detto *training error*, definito come il rapporto tra il numero di esempi del training set che alla fine dell'addestramento l'algoritmo associa al giusto output,  $|X_{corrette}| < |X|$ , e la dimensione del *training set*,  $|X|$ :

$$\text{Training Error} = \frac{|X_{corrette}|}{|X|}$$

Ovviamente, con l'addestramento si vuole minimizzare questo rapporto. Quello descritto è un problema di ottimizzazione.

---

<sup>6</sup>Teoricamente una stessa immagine potrebbe essere descritta da più di un'etichetta o addirittura da nessuna, coerentemente col fatto che in essa potrebbero essere presenti più oggetti o nessun oggetto tra quelli previsti in  $Y$ . Tuttavia nella presente tesi questa ambiguità non può sussistere: la classificazione riduce qualsiasi immagine ad una di due categorie mutualmente esclusive e di cui soltanto una è quella corretta, cioè la presenza o meno di una pinna nell'immagine.

## 2.4. MACHINE LEARNING E DEEP LEARNING

---

Tuttavia non basta che il modello si comporti bene su una collezione di dati di cui fondamentalmente si conosceva già l'output (abilità di per sé abbastanza inutile), ma si vuole, come già spiegato, che esso lavori bene anche su un **test set**  $T$  di esempi mai forniti in input per l'addestramento del modello, e pertanto non presenti nel training set. Si può definire similmente al *training error* un parametro detto **generalization error** (errore di generalizzazione), o **test error**.

$$\text{Test Error} = \frac{|T_{corrette}|}{|T|}$$

Si vuole ovviamente che anche il test error, come il training error, sia piccolo. Più precisamente, si vuole che la differenza tra il training error e il test error sia quanto più piccola possibile.<sup>7</sup>

I fattori che determinano quanto bene un algoritmo di machine learning performerà sono in definitiva le sue capacità di

1. rendere piccolo il training error
2. rendere piccola la differenza tra training error e test error

Quando un modello non riesce ad ottenere un errore sufficientemente basso sul training set si dice che il modello soffre di **underfitting** (sottoadattamento). Quando il modello non riesce a rendere piccola a sufficienza la differenza tra training error e test error si dice che il modello soffre di **overfitting** (sovradattamento).

Possiamo controllare la tendenza di un modello all'underfitting o all'overfitting regolando la sua **capacità**. Informalmente, la capacità di un modello è la sua abilità ad adattarsi ad un insieme ampio di funzioni. Modelli con capacità bassa potrebbero avere difficoltà nell'adattarsi al training set (underfitting). Al contrario, modelli con capacità alta hanno una maggiore probabilità di adattarsi troppo bene al training set (overfitting), memorizzando molte caratteristiche e proprietà degli esempi del training set che non sempre aiutano nella generalizzazione ai nuovi casi, ad esempio quelli del test set.

In una rete neurale (par. 2.6) la capacità del modello può essere definita, ad esempio, come il numero di parametri addestrabili (pesi e bias) che la caratterizzano.

Gli algoritmi di machine learning performeranno generalmente bene quando la loro capacità è appropriata rispetto alla reale complessità del task che devono svolgere e alla quantità di esempi  $|X^{train}|$  forniti per l'addestramento. Modelli con una capacità insufficiente non riescono a risolvere task complessi. Modelli con alta capacità possono risolvere task complessi, ma se la loro capacità è eccessivamente alta rispetto alla complessità del task in esame potrebbero soffrire di overfitting. La fig. 2.9 presenta bene la situazione.

Negli esperimenti condotti una delle reti neurali utilizzate (ResNet-50) ha sofferto di overfitting a causa della sua enorme capacità di rappresentazione e del training set relativamente ristretto usato per il suo addestramento (par. 3.3.2).

---

<sup>7</sup>Il problema di ottimizzazione da risolvere è quello della minimizzazione del training error. Il test error non può essere minimizzato, in quanto esso viene valutato quando l'addestramento della rete è finito; anche in fase di training, comunque, il test set non è disponibile per l'addestramento (non può essere trattato come un'estensione del training set, pena la violazione della definizione stessa di "test set").

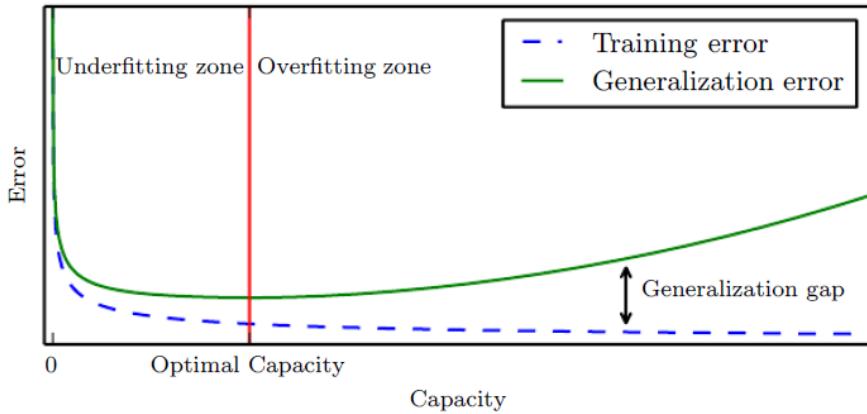


Figura 2.9: Training error e test error (asse y) al variare della capacità del modello (asse x)

## 2.5 Classificatore lineare

Il classificatore lineare è uno tra i più semplici modelli di classificazione. Ipotizziamo di avere un insieme di  $N$  immagini  $\mathbf{x}^{(i)}$  (*data points*), ciascuna con risoluzione fissa  $w \times h$  e in formato RGB ( $c = 3$ ), e un insieme di  $K$  distinte categorie di oggetti (*labels*). Un **classificatore lineare** è definito dalla funzione

$$f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b} \quad (2.1)$$

In questa espressione stiamo assumendo che  $\mathbf{x}^{(i)}$  sia un vettore colonna di dimensione  $D = hwc$  ottenuto incolonnando una ad una le righe dell' $i$ -esima immagine di tutti e tre i canali di colore,  $\mathbf{W}$  una matrice detta **matrice dei pesi** (*weights matrix*) di dimensione  $K \times D$  e  $\mathbf{b}$  un vettore colonna detto **vettore dei bias** (*bias vector*) di dimensione  $K$ . I pesi e i bias sono parametri della funzione  $f$ .

Ogni riga  $j$ -esima di  $\mathbf{W}$  e il relativo  $j$ -esimo valore di  $\mathbf{b}$  serve a calcolare la combinazione (lineare a meno del bias)  $\mathbf{w}_j \cdot \mathbf{x}^{(i)} + b_j$ . Ognuna delle  $K$  combinazioni calcolate è un numero reale che si può interpretare come un "punteggio" registrato dall' $i$ -esima immagine in ogni classe di oggetti in  $Y$  (*class score*): l' $i$ -esima immagine è classificata con l'etichetta  $\mathbf{y}_j \in Y$  se l'elemento  $j$ -esimo del vettore output  $f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b})$  è il massimo del medesimo vettore.

L'esempio in figura 2.10 mostra la classificazione di un'immagine di un gatto con  $|Y| = 3$  classi (*gatto, cane, barca*). Per semplicità, l'immagine input è ipotizzata  $2 \times 2$  e composta da un unico canale di colore ( $c = 1$ ) (quindi  $\mathbf{x}$ , scritta come vettore colonna, è  $4 \times 1$ ).

Come si vedrà nel par. 2.6.1, nelle reti neurali il classificatore lineare sarà utilizzato come un singolo "blocco da costruzione" per costruire una rete più grande.

### Interpretare un classificatore lineare

Poiché le immagini possono essere memorizzate come vettori colonna  $hwc$ -dimensionali, si possono immaginare le immagini di un dataset come dei punti nello spazio  $\mathbb{R}^{hwc}$ .

## 2.5. CLASSIFICATORE LINEARE

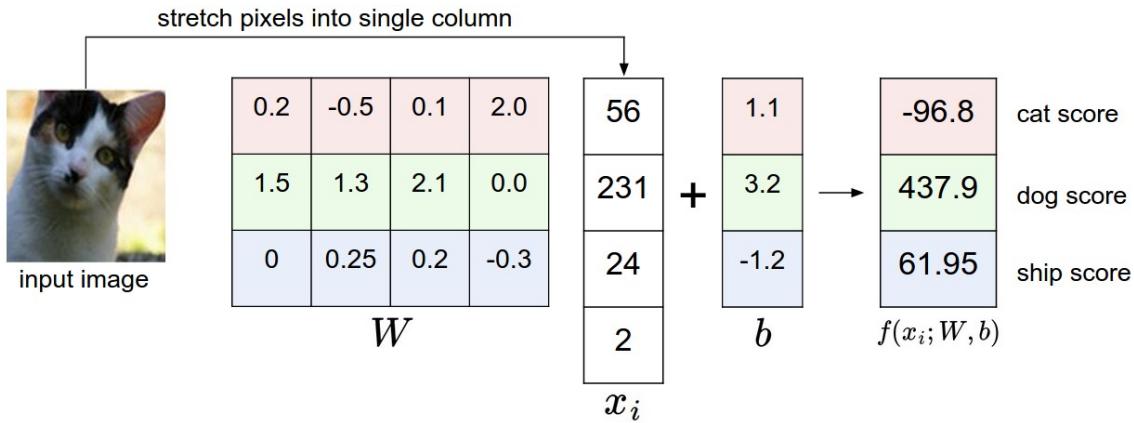


Figura 2.10: Mappatura di un’immagine ai punteggi di ogni classe mediante un classificatore lineare. Si noti che i pesi di  $\mathbf{W}$  non costituiscono un buon set di parametri: il punteggio assegnato alla classe "cane" (sbagliata) è alto e quello totalizzato dalla classe "gatto" (corretta) è basso. Il classificatore "è convinto" di aver classificato l’immagine di un cane.

Di conseguenza, il dataset può essere pensato come una collezione di punti multidimensionali. Ovviamente non possiamo visualizzare spazi con più dimensioni di  $\mathbb{R}^3$ , ma se immaginiamo di "comprimere" tutte le  $hwc$  dimensioni in sole due dimensioni otteniamo una visualizzazione del tipo in figura 2.11.

Le rette in figura devono in realtà essere pensate come degli iperpiani  $(hwc - 1)$ -dimensionali, associati a ciascuna classe di  $Y$  (cioè a ciascuna riga di  $\mathbf{W}$  e  $\mathbf{b}$ ), e il piano come lo spazio  $\mathbb{R}^{hwc}$ . Sussistono le seguenti interpretazioni geometriche:

- Le immagini sono dei punti nel piano. Ogni retta è il luogo dei punti che totalizzano un punteggio nullo per la classe associata a quella retta (la classe è scritta in figura accanto ad ogni retta). La freccia nella figura indica la direzione seguendo la quale i punti del piano aumentano (linearmente) il punteggio realizzato per quella classe.
- Modificare i pesi di  $\mathbf{W}$  significa regolare l’inclinazione delle rette (cioè ruotarle rispetto al punto di intercetta).
- Modificare i bias di  $\mathbf{b}$  significa regolare l’intercetta delle rette (cioè traslarle verticalmente).

Un altro modo di interpretare i pesi  $\mathbf{W}$  può essere quello di far corrispondere ogni riga di  $\mathbf{W}$  a un **prototipo** (in inglese **template**) per una delle classi. In questa interpretazione, il punteggio realizzato per ogni classe da un’immagine è ottenuto attraverso l’operazione di prodotto matriciale tra il prototipo della classe  $j$  ( $\mathbf{w}_j$ ) e l’immagine da classificare ( $\mathbf{x}^{(i)}$ ). Usando la terminologia introdotta, possiamo affermare che ciò che sta facendo il classificatore lineare è un’operazione di *template matching*, dove i *templates* sono oggetto di apprendimento da parte del classificatore<sup>8</sup>.

<sup>8</sup>Si introduciranno gli algoritmi di apprendimento (supervisionato) nel paragrafo 2.4.1.

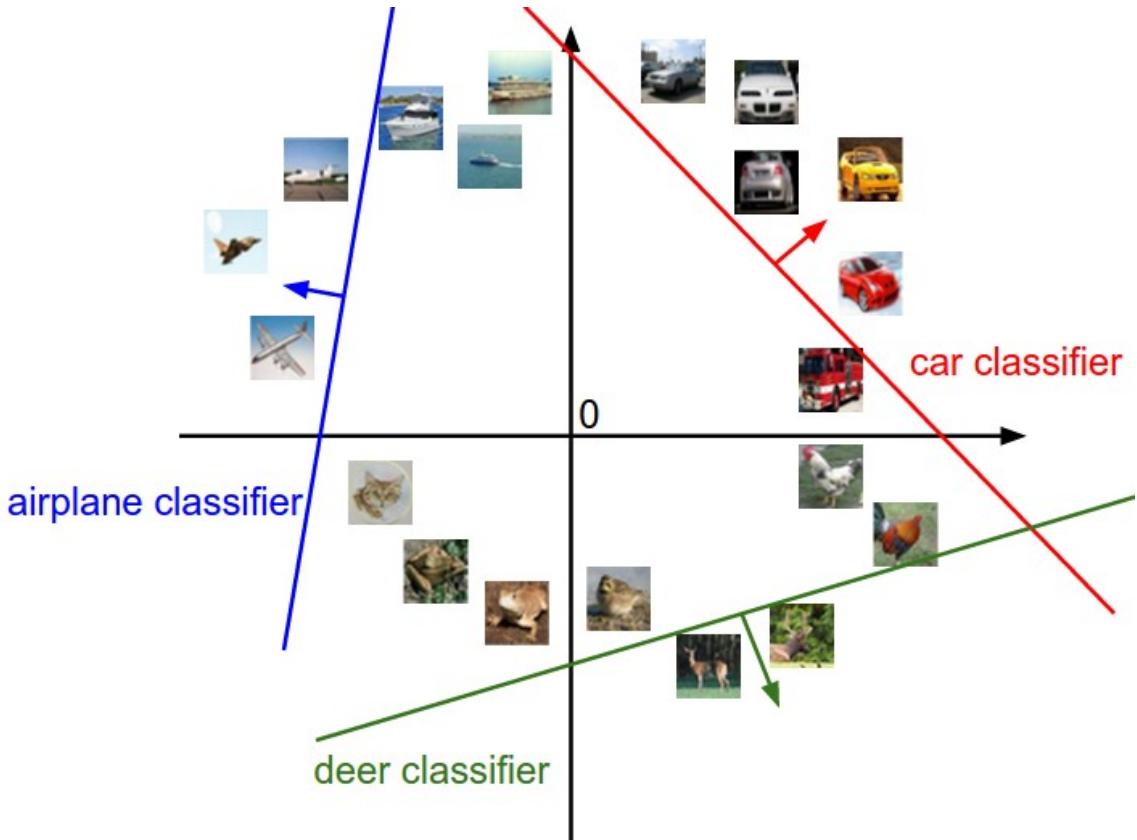


Figura 2.11: Visualizzazione di tre righe di un classificatore lineare, una per ciascuna delle classi "aereo", "auto", "cervo".

Ad esempio, analizziamo il dataset *CIFAR-10* [16]. Esso contiene immagini  $32 \times 32$  ciascuna appartenente ad una di 10 classi. Visualizzando<sup>9</sup> i pesi (e quindi i 10 templates) di un classificatore lineare addestrato su CIFAR-10 si ottengono i risultati in figura seguente:



Figura 2.12: Visualizzazione dei templates di un classificatore addestrato sul dataset CIFAR-10

Si possono fare alcune interessanti osservazioni.

Ad esempio, il prototipo della classe "barca" è composto da molti pixel blu disposti perlopiù lungo i margini, come ci si potrebbe aspettare dal momento che molte immagini di barche in CIFAR-10 raffigurano queste in mare aperto. Questo template allora assegnerà un punteggio alto quando l'immagine che si vuole classificare (cioè *rappresentare al template*) è una barca in mare aperto. In altre parole,

<sup>9</sup>Per i dettagli su come "visualizzare" i pesi si veda <https://it.mathworks.com/help/deeplearning/examples/visualize-activations-of-a-convolutional-neural-network.html>.

## 2.5. CLASSIFICATORE LINEARE

---

un'immagine realizzerà un punteggio tanto più alto in una certa classe quanto più essa è *simile* al template che il classificatore lineare *ha imparato* per quella classe.

Il prototipo per la classe "cavalo" sembra essere l'immagine di un cavallo a due teste; similmente, quello per la classe "auto" sembra una miscela di rappresentazioni di un'auto vista da più direzioni diverse. Ciò è coerente col fatto che il classificatore lineare è stato addestrato su immagini di cavalli visti rispetto a entrambi i profili e su immagini di auto raffigurate in tante direzioni diverse. Inoltre, il template per l'auto sembra rappresentare un'auto di colore rosso: evidentemente in CIFAR-10 la maggior parte delle automobili rappresentate sono di quel colore.

Come si vedrà nel seguito, questa operazione di *template matching* presenta una forte analogia con il funzionamento di un *Fully Connected Layer* di una rete neurale convoluzionale.

### Bias trick

Concludiamo questo capitolo menzionando un modo molto utilizzato per rappresentare  $\mathbf{W}$  e  $\mathbf{b}$  come un'unica matrice, semplificando la notazione 2.1. Possiamo aggiungere il vettore dei bias in coda alla matrice dei pesi e aggiungere un "1" in coda al vettore che rappresenta l'immagine. In questo modo, il classificatore lineare è rappresentato dalla funzione di associazione

$$f(\mathbf{x}^{(i)}; \mathbf{W}) = \mathbf{W}\mathbf{x}^{(i)} \quad (2.2)$$

In questa maniera,  $f$  calcola solo combinazioni lineari (un singolo prodotto matriciale), poiché il vettore dei bias è stato eliminato. Tale utile passaggio, noto come *bias trick*, è visualizzato nella seguente figura

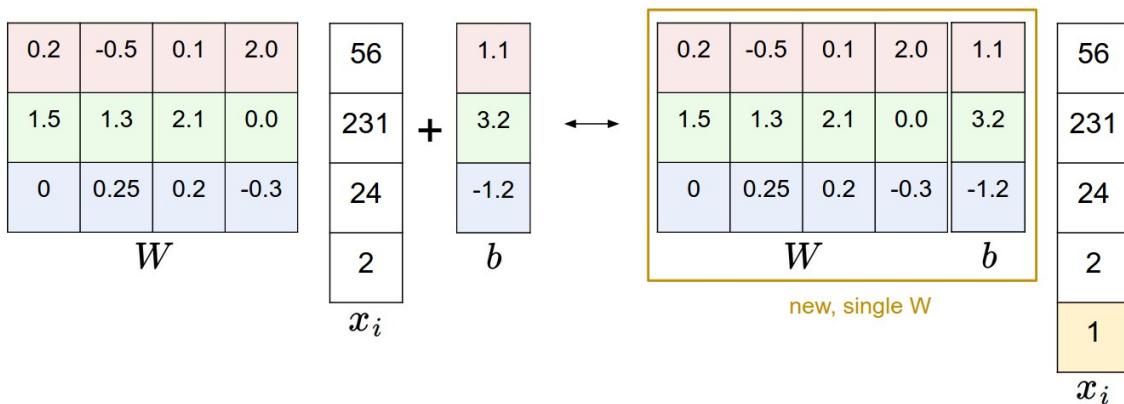


Figura 2.13: Bias trick

## 2.6 Reti Neurali

Le reti neurali artificiali (ANN, *artificial neural networks*), 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**, che nell'insieme sono in grado di fornire una nuova rappresentazione dell'input, secondo il paradigma del *representation learning*. 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

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

in cui

- $d$  è la profondità della rete, cioè il numero di layers che la compongono;
- $\mathbf{h}^{(1)}$  è il primo (hidden) layer,  $\mathbf{h}^{(2)}$  il secondo, e così via. Ciascuno di essi rappresenta una trasformazione parametrica in generale non lineare delle feature relative ad un input  $\mathbf{x}$ : ogni hidden layer  $\mathbf{h}$  accetta un vettore in input  $\mathbf{x}$ , calcola una trasformazione affine  $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ , quindi applica una funzione non lineare  $g(\mathbf{z})$  elemento per elemento, detta **funzione di attivazione**. Si ottiene così:

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

fino a giungere all'output layer:

$$\text{Layer } d: \quad \mathbf{f}^{(d)} = g^{(d)}(\mathbf{W}^{(d)}\mathbf{h}^{(d-1)} + \mathbf{b}^{(d)})$$

- $f^{(d)}$  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

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

- Layer di output softmax: viene calcolata una trasformazione affine

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

e viene quindi applicata la **funzione softmax**

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

Grazie alla funzione softmax, l'output  $\hat{\mathbf{y}}$  è la distribuzione di probabilità che  $\mathbf{x}$  appartenga alla classe  $i$ :

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

La funzione di attivazione più utilizzata nell'ambito delle ANN è la seguente **ReLU** (Rectified Linear Unit), anche detta **rettificatore**:

$$\text{ReLU}(x) = x^+ = \max(0, x)$$

Il rettificatore è la funzione di attivazione usata in tutte le reti neurali presentate in questo lavoro di tesi (par. 2.12, 2.13, 2.14). Altre possibilità sono la sigmoide  $g(x) = \sigma(x)$  oppure la tangente iperbolica  $g(x) = \tanh(x)$ .

### 2.6.1 I neuroni

L'area di ricerca sulle ANN trae le sue origini dall'obiettivo degli scienziati di modellizzare matematicamente i sistemi neurali biologici che governano le facoltà intellettive del cervello umano. È infatti possibile stabilire un confronto tra il neurone biologico e il "neurone" artificiale, definito nel precedente paragrafo, che ne costituisce una sua (semplicistica ma efficace) modellizzazione matematica.

Ogni neurone biologico (fig. 2.14a) riceve segnali input dai suoi dendriti, e produce il segnale output lungo il suo (unico) assone. L'assone può eventualmente ramificarsi e connettersi via sinapsi ai dendriti di altri neuroni.

Nel modello computazionale (fig. 2.14b) del neurone, i segnali che viaggiano lungo gli assoni (ad es. gli  $x_i$  in figura) sono pesati con un certo valore (i pesi  $w_i$ ) che quantifica la "plasticità sinaptica"<sup>10</sup> di quella sinapsi. I vari segnali pesati provenienti da tanti neuroni si sommano all'interno del corpo cellulare (assieme ad un bias  $b$ ). Il risultato numerico è presentato all'assone di output passando per una funzione di attivazione ( $f$  in figura), che rappresenta la frequenza di emissione del segnale da parte di un neurone biologico<sup>11</sup>.

Ogni neurone si comporta quindi come la composizione di una funzione di attivazione  $g$  con una funzione lineare (affine)  $f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b$ ; evidentemente, uno strato composto da neuroni si comporta come un classificatore lineare (par. 2.5): ogni neurone di un certo strato  $i$  aggiunge una riga alla matrice  $\mathbf{W}^{(i)}$  dei pesi ed una al vettore  $\mathbf{b}^{(i)}$  dei bias, associati a quello strato.

---

<sup>10</sup>La plasticità sinaptica è la capacità del sistema nervoso di modificare l'intensità delle relazioni interneuronali (sinapsi).

<sup>11</sup>È questa frequenza di emissione (più che il segnale in sé, che è assimilabile ad un impulso) a trasportare l'informazione.

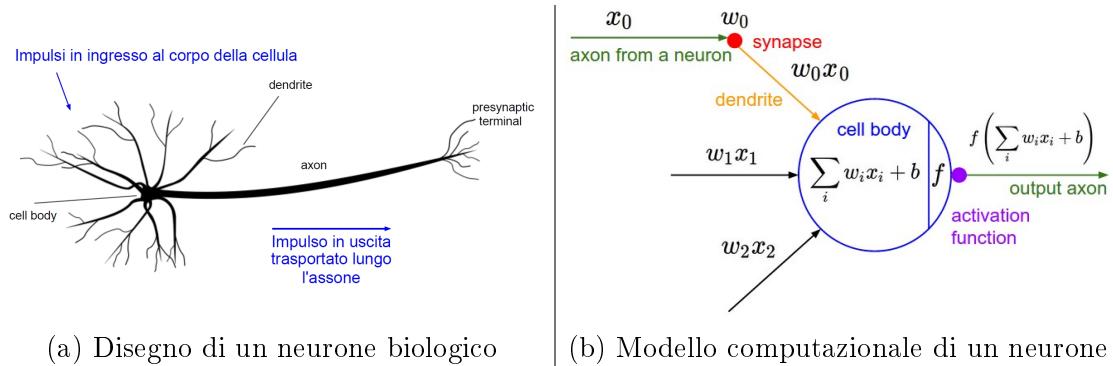


Figura 2.14: Neuroni. Immagini tratte da: (a) <https://thenounproject.com/term/neuron/214105/>, (b) [12]

## 2.6.2 Architettura delle reti neurali artificiali

Le reti neurali sono modellate come collezioni di neuroni connessi in un grafo aciclico diretto<sup>12</sup>, organizzati in strati distinti.

Le architetture più "standard" delle reti neurali sono costituite da uno strato di input, uno o più **strati completamente connessi** (*fully connected (FC) layer*) nei quali ciascun neurone è connesso a tutti i neuroni dello strato precedente e non esistono connessioni tra i neuroni dello stesso strato, ed infine uno strato di output. La fig. 2.15 mostra un esempio di rete neurale con soli layer FC.

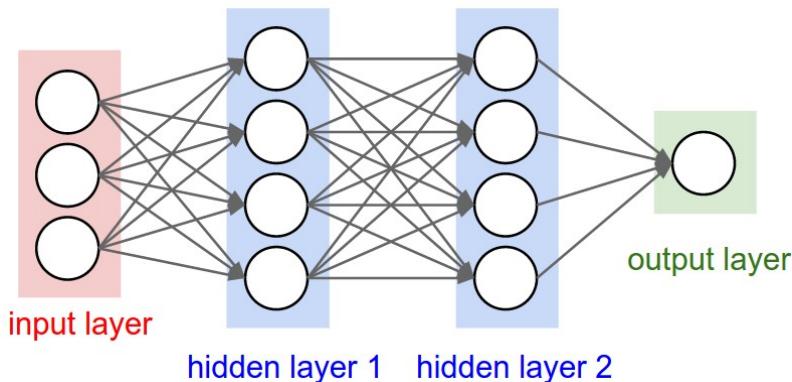


Figura 2.15: Una rete neurale con 3 strati (conventionalmente non si conta quello di input) di cui due nascosti (*hidden layers*) con 4 neuroni ciascuno e uno di output con un solo neurone.

I valori reali assunti dai neuroni sono chiamati **attivazioni** dei neuroni. Nei problemi di classificazione delle immagini (par. 2.3.1), lo strato di input contiene neuroni in numero esattamente uguale a  $hw \times c$ , essendo le immagini accettate in input dalla rete di risoluzione  $w \times h$  e con  $c$  canali di colore. Ogni neurone contiene uno dei tre valori scalari che compongono un pixel. I neuroni dello strato di output sono in numero uguale a  $|Y|$ , il numero di categorie di oggetti che si possono classificare.

<sup>12</sup>in inglese *direct acyclic graph* (DAG); particolare grafo orientato che non ha cicli diretti, ovvero comunque scegliamo un vertice del grafo non possiamo tornare ad esso percorrendo gli archi del grafo.

## 2.7. RETI NEURALI CONVOLUZIONALI

---

Spesso i neuroni dell'output layer non adoperano una funzione di attivazione (possiamo immaginare che usino la funzione identità). Nel caso di reti che svolgono un task di classificazione di immagini, come nel nostro caso, i valori delle attivazioni dei neuroni di output sono immessi nella funzione softmax per calcolare la distribuzione di probabilità da attribuire ad ogni classe di oggetti.

Le dimensioni di una rete neurale possono essere misurate con diverse metriche, tra cui il numero di neuroni o, più comunemente, il numero di parametri addestrabili. Ad esempio, la rete rappresentata in fig. 2.15 ha  $4 + 4 + 1 = 9$  neuroni e quindi  $3 \times 4 + 4 \times 4 + 4 \times 1 = 32$  pesi e  $4 + 4 + 1$  bias, per un totale di 41 parametri addestrabili.

Una rete neurale con strati completamente connessi può rappresentare una famiglia di funzioni (vettoriali di variabile vettoriale) parametrizzati dai pesi della rete. È stato dimostrato [17] 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 qualsiasi funzione continua su un insieme chiuso e limitato di  $\mathbb{R}^n$  con un errore arbitrario. Questo teorema fornisce una giustificazione teorica molto generica del funzionamento delle reti neurali, ma non specifica nessun dettaglio riguardante l'architettura da utilizzare (es. numero di layer FC e numero di neuroni in ciascuno di essi) per poter ottenere un errore di approssimazione desiderato. 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.

Tutti i parametri della rete che non sono addestrabili ma che sono invece scelti arbitrariamente dal progettista (es. numero di layer della rete, input size, numero di neuroni in ogni layer, ecc.) si chiamano **iperparametri** della rete (*hyperparameters*). Il loro valore ottimale non può essere trovato matematicamente (o meglio, spesso risulterebbe computazionalmente difficile farlo) ma possono essere scelti con un processo *trial and error*.

## 2.7 Reti neurali convoluzionali

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

Grazie alla natura dell'operazione di convoluzione (par. 2.2.5), 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 adoperate nell'ambito del task di classificazione binaria di immagini oggetto del presente lavoro (par. 3.3). Per il resto del presente lavoro, ci riferiremo a CNN che lavorano esclusivamente su immagini e che nel suo strato finale utilizza la funzione softmax per calcolare la distribuzione di probabilità associata ad ogni classe dello spazio delle classi.

Una rete neurale convoluzionale è costituita da uno stack di layers, mostrato in fig. 2.16: in input c'è un'immagine e in output c'è un vettore contenente la distri-

---

buzione di probabilità relativa alle categorie di oggetti classificabili.

L'architettura di una CNN può comporsi di strati di tipologie diverse<sup>13</sup>

- Input: generalmente un'immagine di dimensioni contenute (es.  $224 \times 224$ ). (par. 2.7.1)
- *Convolutional layer* (CONV): vengono eseguiti diversi filtraggi 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. 2.7.2). Viene prodotto un set di *feature map*.
- *Activation layer* (detto anche *detector stage*): ciascun neurone dello strato precedente è passato ad una funzione di attivazione, tipicamente la ReLU. (par. 2.7.3)
- *Pooling layer* (POOL): viene eseguito un sottocampionamento dell'immagine attraverso una statistica riassuntiva delle regioni. (par. 2.7.4)
- *Fully-Connected layer* (FC): già descritti nella trattazione sulle reti neurali classiche; nell'ambito delle CNN sono finalizzati ad effettuare la classificazione vera e propria a partire dalle *feature* estratte dai livelli convoluzionali precedenti. (par. 2.7.5)
- *Softmax layer*: applica la funzione di attivazione softmax alla fine della rete, per calcolare la distribuzione di probabilità delle classi da predire. (par. 2.7.6)

L'architettura di una CNN può prevedere una concatenazione di layer in serie (si parla in quest caso di *series networks*, es. AlexNet par. 2.12) ma anche una serie di modifiche che portano certi layer a funzionare in parallelo tra loro (ad es. il modulo *inception* di GoogLeNet, par. 2.13.2) o che comunque alterano il flusso lineare dell'informazione attraverso la rete mediante diramazioni (ad es. le *skip connections* in ResNet, par. 2.14.2).

In questo modo, una rete CNN associa l'immagine in ingresso alla distribuzione di probabilità associata alle categorie classificabili.

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, senza parametri addestrabili. I parametri degli strati CONV e FC vengono appresi in fase di addestramento attraverso l'applicazione dell'algoritmo di ottimizzazione *stochastic gradient descent*, in modo che la distribuzione calcolati all'ultimo livello siano coerenti con le etichette del *training set* per ogni immagine.

---

<sup>13</sup>Va comunque notato che esistono numerosi altri strati che non sono presentati in questa sede. Alcuni di questi sono descritti nei paragrafi sulle tre CNN usate negli esperimenti, a cui pertanto si rimanda. Per una descrizione accurata di molti altri tipi di layer si rimanda a <https://code.google.com/archive/p/cuda-convnet/wikis/LayerParams.wiki>

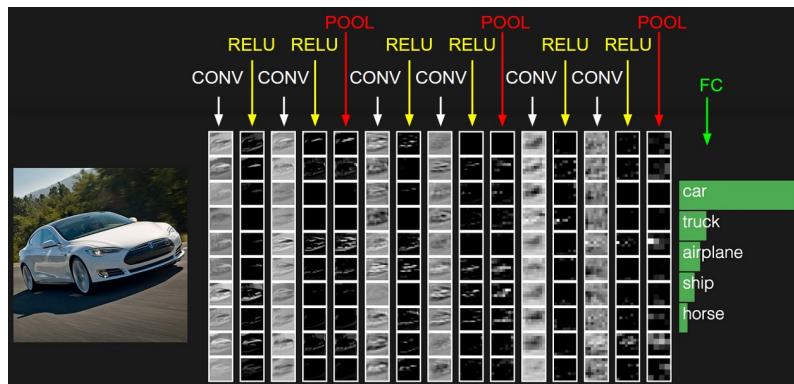


Figura 2.16: Le attivazioni di un esempio di architettura ConvNet. Il volume iniziale memorizza i pixel dell'immagine in input (a sinistra) e l'ultimo volume memorizza la probabilità per ogni classe (a destra). Ogni volume di attivazioni lungo il percorso di elaborazione è mostrato come una colonna, in cui ogni immagine è una mappa bidimensionale di attivazione (vd. par. 2.7.2).

## Volumi di attivazioni

Come detto, le reti neurali convoluzionali ricevono in input delle immagini. Questo rende possibile un più "sensato" ed intuitivo arrangiamento dei neuroni nei vari layer. I neuroni sono infatti arrangiati in **volumi tridimensionali di attivazioni**. Così come le immagini hanno pixel arrangiati lungo la lunghezza, l'altezza e i canali di colore (profondità) dell'immagine, i neuroni di un certo strato possono essere arrangiati spazialmente lungo le stesse tre dimensioni. Allora si può dire che la rete opera una serie di trasformazioni che portano da un volume di attivazioni iniziale (quello determinato dall'immagine  $w \times h \times 3$ ) ad uno finale: lo strato  $i$ -esimo trasforma il volume di attivazioni  $i - 1$ -esimo nel volume  $i$ -esimo per mezzo di una funzione differenziabile.

Si veda la fig. 2.17. Al variare dell'indice  $d$  del volume, che indica la terza dimensione del volume (la profondità, o *volume depth*), si può univocamente identificare una sezione bidimensionale del volume, che chiameremo **mappa di attivazioni** (*activation map*, anche detta *depth slice*). In figura, ogni mappa di attivazioni è una griglia che contiene  $a \times b$  neuroni.

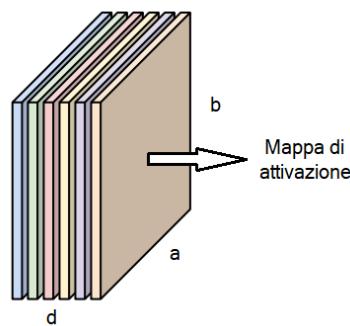


Figura 2.17: Volume di attivazioni  $a \times b \times d$

---

Si descrivono nel seguito con maggiore dettaglio i singoli layer, con le relative tecniche di implementazione in Matlab, utilizzate nel presente lavoro (par. 3.3). Ogni tipo di layer esiste in Matlab come un tipo di dato complesso a se stante, ciascuno con la propria funzionalità.

In Matlab è possibile definire l'architettura di una CNN in due modi diversi:

1. Le reti i cui layer sono tutti "in serie" tra loro (es. AlexNet, par. 2.12) sono rappresentate da un oggetto di tipo **SeriesNetwork**, il quale contiene come campo un array di tipo **Layer** che contiene (in ordine) gli strati della rete.
2. Le reti che hanno in generale un grafo aciclico diretto ma non in serie (es. GoogLeNet, par. 2.13, ResNet-18 e 50, par. 2.14) sono rappresentate da un oggetto di tipo **DAGNetwork**, il quale oltre al campo di tipo **Layer** sopra citato contiene anche un altro campo di tipo **table** di dimensioni  $p \times 2$ , dove  $p$  è il numero di connessioni da stabilire tra i layer e le due colonne specificano rispettivamente il layer di partenza e quello di arrivo di ciascuna connessione.

Entrambi questi oggetti possono essere passati come argomento della funzione **trainNetwork** al fine di addestrare la rete.

### 2.7.1 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  $h \times w$  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'**

### 2.7.2 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 un'operazione di convoluzione (tridimensionale) tra il volume di attivazione  $a \times b \times c$  e un kernel  $h \times k \times c$ , e per ogni mappa di attivazione ottenuta si aggiunge ai neuroni di quella mappa un termine di bias (vettore di bias  $1 \times 1 \times c$ ) (fig. 2.18).

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

## 2.7. RETI NEURALI CONVOLUZIONALI

---

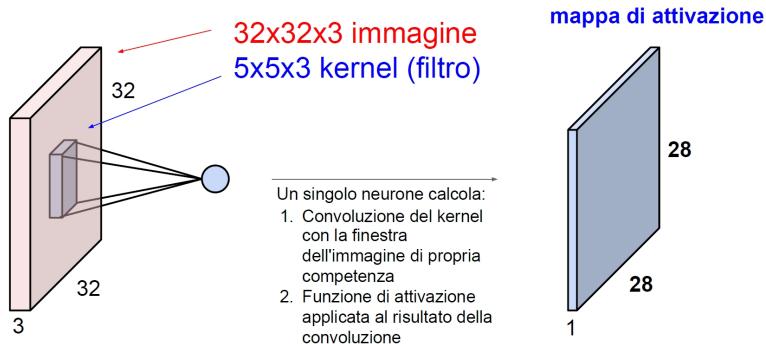


Figura 2.18: Esempio di computazione di un neurone attraverso un layer di convoluzione e di attivazione. Utilizzando una convoluzione valida con stride pari a 1 e senza input padding, la convoluzione di un’immagine  $32 \times 32 \times 3$  con un filtro  $5 \times 5 \times 3$  produce una matrice  $28 \times 28 \times 1$ , cui viene applicata la funzione d’attivazione scelta per ottenere una activation map. Tutti i  $28 \times 28$  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 quadrato  $F \times F$ . Se si vogliono utilizzare dimensioni differenti (filtro in generale rettangolare) 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. Il volume di output avrà esattamente  $K$  livelli di profondità (cioè  $K$  mappe di attivazioni). (fig. 2.19)

Gli eventuali parametri che possono essere specificati sono:

- **'Stride'**,  $S$

Parametro che regola lo scorrimento della finestra del filtro 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'**,  $L$

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

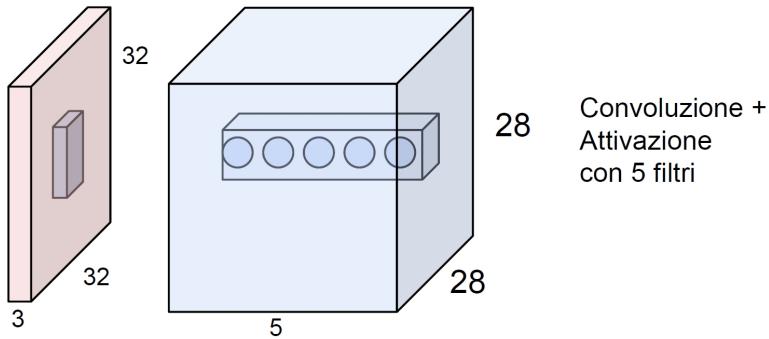


Figura 2.19: I 5 neuroni mostrati applicano un diverso filtro (un kernel con diversi parametri) alla stessa regione spaziale dell’immagine in input, in maniera indipendente l’uno dall’altro.

campo ricettivo (*receptive field*) rispetto all’input. Se si vuole utilizzare una dilatazione differente rispetto alle due dimensioni è possibile specificare un vettore  $[h \ w]$ . (fig. 2.20)

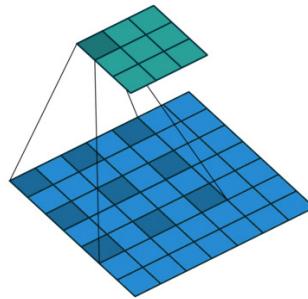


Figura 2.20: Dilatation di fattore  $L=2$

- **'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

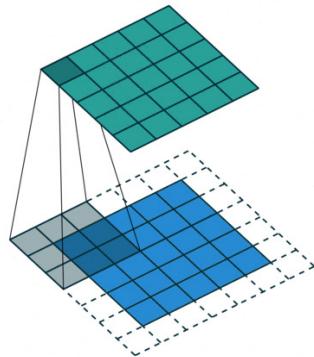
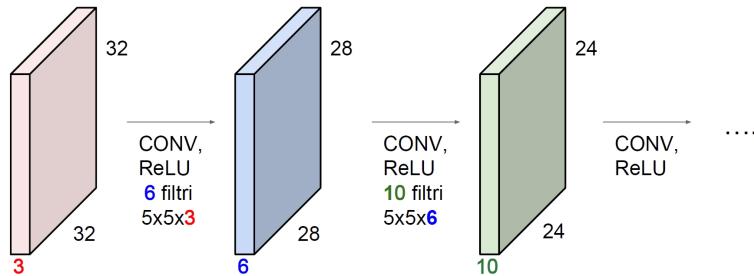

 Figura 2.21: Padding con  $t=b=l=r=1$ 


Figura 2.22: 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 tutti zeri
  - ‘ones’: inizializza con tutti 1
  - function handle: utilizza una funzione *custom*
- ‘BiasInitializer’, ‘zeros’ | ‘narrow-normal’ | ‘ones’ | function handle

Learning rate (locale) e regolarizzazione:

- ‘WeightLearnRateFactor’, alfa

Fattore da moltiplicare per il parametro learning rate globale relativo all’apprendimento dei pesi, di default pari a 1

- ‘BiasLearnRateFactor’, beta

Fattore da moltiplicare per il parametro learning rate globale relativo all’apprendimento dei bias, di default pari a 1

- ‘WeightL2Factor’, lambda1

Fattore da moltiplicare per il parametro globale di regolarizzazione L2 relativo all’apprendimento dei pesi, di default pari a 1

- ‘BiasL2Factor’, lambda2

Fattore da moltiplicare per il parametro globale di regolarizzazione L2 relativo all’apprendimento dei bias, di default pari a 1

### 2.7.3 Activation layer

Il layer di attivazione è implementato, nel caso di funzione ReLU (l’unica che utilizzeremo in questo lavoro di tesi), mediante `relulayer`.

### 2.7.4 Pooling layer

Una funzione di pooling opera un sottocampionamento sull’immagine, come mostrato in figura 2.23. Ogni regione di una certa dimensione dell’immagine in input (es.  $4 \times 4$ ) viene ridotta ad un unico pixel, il cui valore è calcolato tramite una statistica riassuntiva della regione di partenza (es. max, media, norma L2).

L’utilità del pooling è quella di ridurre progressivamente la dimensione spaziale dei volumi che “scorrono” nell’architettura, per ridurre la quantità di parametri e di calcolo nella rete. Tipicamente, l’operazione è eseguita mediante un filtro di dimensioni  $2 \times 2$  (o anche  $3 \times 3$ ) applicato con uno stride di 2. Il risultato è quello di ridurre del 75% il numero delle attivazioni propagate. Ogni operazione di max pooling richiederebbe in questo caso un massimo di 4 numeri (regione  $2 \times 2$ ). La

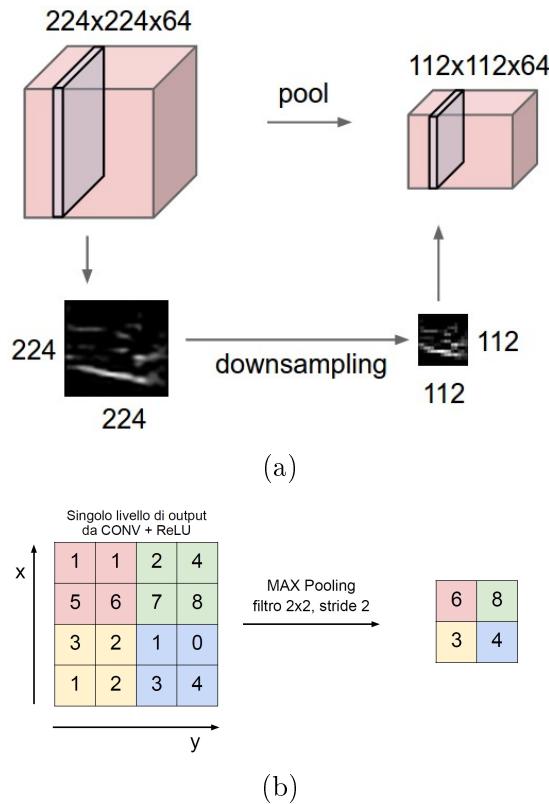


Figura 2.23: Operazione di pooling: (a) effetto di downsampling; (b) calcolo del max pooling

dimensione della profondità del tensore di attivazione rimane invariata.

La funzione utilizzata per la sua creazione è `maxPooling2dLayer`

```
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'`.

### 2.7.5 Fully Connected Layer

Moltiplica l'input per un tensore di pesi (di dimensioni pari a quelle del volume in input al layer) e aggiunge un vettore bias (di dimensione pari al numero di neuroni del layer FC). Ciascun neurone è connesso con tutti i neuroni del layer precedente.

Un esempio numerico è mostrato in fig. 2.24.

I layer fully connected si implementano in Matlab con:

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

dove:

- **outputSize**

Dimensione di uscita del fully connected layer, specificato come un numero intero positivo C. Nel caso in cui il layer sia posto alla fine della rete (layer FC di classificazione, che fornisce valori alla funzione softmax finale), C deve essere uguale al numero delle classi  $|Y|$  da individuare all'interno del dataset.

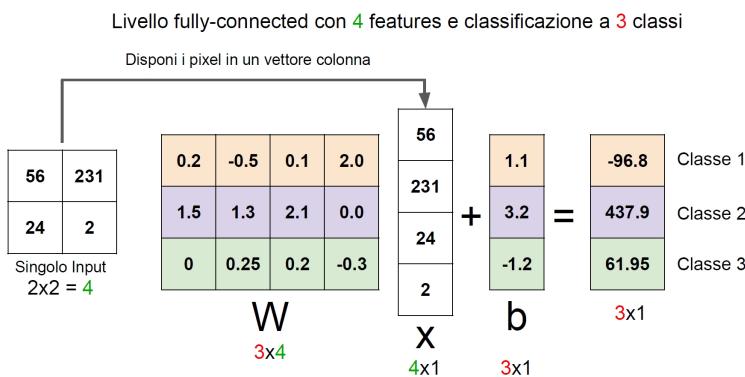


Figura 2.24: Esempio numerico di livello fully connected con un singolo input

### 2.7.6 Softmax layer

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

## 2.8 Image preprocessing e augmentation

La tecnica dell'*image preprocessing* consiste in generale nel trasformare le immagini (esempi) di un dataset per mezzo di operazioni sulle sue feature. Tipicamente, queste operazioni sono applicate alle immagini di un training set prima che ci si serva di esso per addestrare un classificatore, ma possono essere applicate anche sui test set. Le trasformazioni possono includere centraggio delle feature, normalizzazione, traslazioni, rotazioni, riflessioni, scalatura, e così via.

In generale, ci sono due motivi per cui si vuole effettuare il preprocessing di un training set:

- Rendere le immagini più semplici da elaborare da parte di una rete neurale. Ad esempio, il centraggio e la normalizzazione delle feature possono essere utili ad evitare eventuali problemi di overflow e stabilità numerica nelle operazioni eseguite dalla rete.

## 2.8. IMAGE PREPROCESSING E AUGMENTATION

---

- Migliorare la capacità di generalizzazione del modello. Nel caso dell'*image classification*, 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 geometriche come traslazioni, rotazioni, riflessioni, scalatura (già viste nel par. 2.2) possono essere implementate come semplici operazioni 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.

Tra le forme di preprocessing più comunemente utilizzate su un'immagine (esempio)  $\mathbf{x}$  di un training set per assolvere al primo dei due obiettivi del preprocessamento delle immagini citiamo:

- **Sottrazione della media** (*mean subtraction*), che consiste nel sottrarre ad ogni feature dell'esempio la media di tutte le feature. L'interpretazione geometrica di questa operazione è il centraggio delle feature attorno allo 0.
- **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]. Un'ulteriore forma di normalizzazione è la divisione di ogni feature per la deviazione standard di tutte le feature (operazione che va applicata solo dopo il centraggio attorno allo 0 delle feature, con la sottrazione della media).

Il secondo obiettivo del preprocessamento è particolarmente critico nel caso in cui il training set a nostra disposizione è relativamente piccolo e poco eterogeneo (cioè con poca variabilità nelle immagini di una stessa classe). Spesso training set piccoli non sono sufficienti per addestrare la rete in maniera soddisfacente, rendendo il rischio di overfitting particolarmente concreto.

Al fine di aggirare questo problema si mette in campo una strategia di *image augmentation*, una particolare forma di *image preprocessing* che ha come scopo quello di ingrandire (*augment*) il training set con nuovi esempi, generati in vario modo a partire da quelli del training set originale.

Le trasformazioni di *augmentation* sono in genere scelte in maniera casuale (con una certa probabilità per ciascuna) per ogni esempio da "aumentare", da un set di operazioni specificato a priori da chi addestra la rete.

Tra le operazioni di *image augmentation* più comuni citiamo:

- Estrarre da ogni esempio dei ritagli rettangolari da posizioni differenti.
- Applicare trasformazioni geometriche affini, quali quelle descritte nel par. 2.2.
- Applicare distorsioni fotometriche e cromatiche di vario tipo.

Bisogna comunque osservare che per dataset grandi ed eterogenei solitamente si fanno poche operazioni di augmentation di questo tipo, poiché si lascia imparare al

modello a quale tipo di fattori di variazione esso deve essere invariante.

In Matlab è possibile ottenere la versione aumentata di un dataset con un oggetto di tipo `augmentedImage datastore`. Esso consente di trasformare un dataset, definito in un oggetto `image datastore` (vd. par. 2.1.2) con eventuali operazioni di preprocessing e augmentation specificate in un oggetto di tipo  `imageDataAugmenter`. Un oggetto `augmentedImage datastore` può essere passato semplicemente come argomento della funzione `trainNetwork` (vd. par. 3.3.2), per essere utilizzato come *training set* di una rete neurale. In tal caso, per ogni *epoch* (quindi a *runtime*), i dati di training vengono casualmente perturbati e ridimensionati all'input size della rete<sup>14</sup>, in modo che ciascuna *epoch* (epoca), cioè un ciclo di addestramento sull'intero training set, usi un training 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 calcolata un'unica volta, subito dopo la augmentation relativa alla prima *epoch*. In fig. 2.25 viene mostrato il processo di addestramento di una rete neurale adoperando un `augmentedImage datastore`.

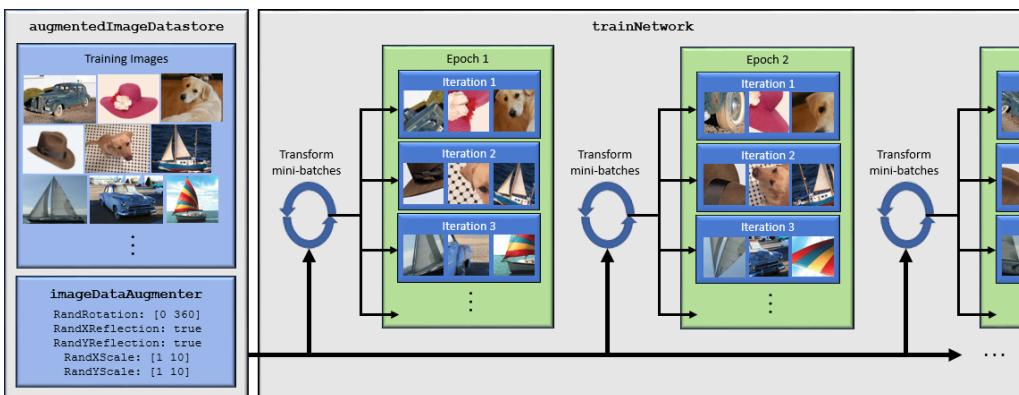


Figura 2.25: Processo di addestramento (mediante *stochastic gradient descent* con dimensione del mini-batch = 3) in cui il training set è definito con un oggetto `augmentedImage datastore`

La creazione di un oggetto `augmentedImage datastore` avviene invocando la funzione

```
auidms = augmentedImage datastore(outputSize, imds)
```

Altri argomenti opzionali sono:

- `NumObservations`: 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` (vd. par. 3.3.2) e può essere modificato solo di lì (proprietà READ-ONLY).

<sup>14</sup> Il ridimensionamento è necessario affinché tutte le immagini rispettino la dimensione di input della rete neurale, che è fisso.

- **Files**: oggetto `cell array` contenente le directory delle immagini ereditate dall'oggetto `imageDatastore` con cui è stato costruito)
- **MiniBatchSize**: dimensione del mini-batch utilizzato durante l'addestramento dal metodo *stochastic gradient descent* per il calcolo della funzione costo. Coincide con l'omonima proprietà dell'oggetto `trainingOptions` e può essere modificato solo di lì (proprietà READ-ONLY).
- **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`

Se non è specificato come argomento un oggetto  `imageDataAugmenter`, l'unica operazione di preprocessing svolta è il ridimensionamento all'input size della rete.

La sintassi per creare un oggetto  `imageDataAugmenter` è:

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

in cui è possibile definire le opzioni di *image augmentation* esprimendo una o più coppie proprietà-valore. Tra le varie possibilità, citiamo solo quelle effettivamente utilizzate all'interno del presente lavoro:

- '`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  $[a, b]$

- '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  $[a, b]$

## 2.9 Addestrare una rete neurale

Per poter essere impiegata per risolvere un certo task, una rete neurale deve essere opportunamente addestrata. Addestrare una rete neurale significa individuare un set di valori per i parametri addestrabili della rete che le permettono di approssimare la funzione desiderata (cioè che le permettano di risolvere efficacemente il task). Un addestramento efficace rende la rete capace di generalizzare in modo appropriato, cioè di dare risposte plausibili per input che non ha mai visto.

Riferendoci ad una rete che svolge un task di *image classification* (ma si può facilmente estendere la spiegazione ad una ANN standard), l'addestramento della rete avviene in due diversi stadi: *forward-pass* (propagazione in avanti) e *backward-pass* (propagazione all'indietro).

- Nel *forward-pass* a partire dalle immagini in input si calcolano di strato in strato tutti i volumi intermedi fino al volume finale, che permette la classificazione. Durante questa fase i valori dei parametri addestrabili sono tutti fissati.
- Nel *backward-pass* la risposta della rete (la distribuzione di probabilità della predizione) viene confrontata con l'uscita desiderata (la distribuzione di probabilità che assegna probabilità 1 alla effettiva classe di appartenenza dell'immagine e 0 alle rimanenti) ottenendo il segnale d'errore, calcolato per mezzo della funzione costo (par. 2.9.2). L'errore calcolato è propagato nella direzione inversa rispetto a quella del *forward-pass*. I parametri sono aggiornati ad ogni iterazione in modo da minimizzare la funzione costo, tramite un metodo di minimizzazione come la discesa stocastica del gradiente (par. 2.9.3), che calcola il gradiente della funzione costo rispetto ad ogni parametro con l'algoritmo di backpropagation (par. 2.9.4).

### 2.9.1 Inizializzazione dei parametri

Prima di addestrare la rete, i parametri devono essere inizializzati con valori casuali, che saranno poi alterati durante il training.

Delle scelte comuni per i pesi sono:

## 2.9. ADDESTRARE UNA RETE NEURALE

---

- estrazione di un valore casuale della variabile aleatoria  $\varepsilon \mathcal{X}$ , dove  $\varepsilon$  è piccolo (tipicamente 0.01) e  $\mathcal{X}$  è la v.a. gaussiana standard (media nulla e varianza unitaria).
- estrazione di un valore casuale della variabile aleatoria  $\frac{\mathcal{X}}{\sqrt{n}}$ , dove  $\mathcal{X}$  è la v.a. gaussiana standard e  $n$  è il numero di parametri associati al nodo a cui il parametro di cui si è estratta l'inizializzazione appartiene.
- estrazione di un valore casuale della variabile aleatoria  $\frac{\mathcal{X}}{\sqrt{2}}$ , con le stesse varia- bili descritte al punto precedente

### 2.9.2 Loss function

Abbiamo visto che una rete neurale per l'*image classification* accetta in input un'immagine e restituisce un punteggio per ogni classe da predire, o equivalentemente una probabilità per ogni classe (calcolata dalla funzione di attivazione softmax). In fig. 2.10 si era visto che quel particolare classificatore lineare assegnava all'immagine di un gatto un punteggio molto alto alla classe 'cane' ed uno molto basso alla classe 'gatto', classificandola pertanto come 'cane'. Il set di parametri costituito da  $\mathbf{W}$  e  $\mathbf{b}$  non è molto buono. Per misurare l'entità dell'errore commesso dalla rete nella classificazione definiamo una **funzione costo** (*loss function*) anche detta funzione obiettivo (*objective*).

Tra le varie loss function che si possono adoperare, quella utilizzata nel presente lavoro è la ***cross-entropy loss***, che è quella tipica delle reti che sfruttano la funzione softmax per effettuare la predizione. Dato un training set (eventualmente aumentato)  $X$  contenente  $|X|$  immagini  $\mathbf{x}^{(i)}$  ( $i = 1, \dots, |X|$ ), ciascuna appartenente alla classe  $i$ -esima, uno spazio delle etichette  $Y$  e una rete neurale che associa all'immagine  $|Y|$  punteggi diversi  $s_j$  per ciascuna classe  $j$ -esima ( $j = 1, \dots, |Y|$ ), si definisce cross-entropy loss relativa al training set  $X$  la funzione

$$L = \underbrace{\frac{1}{|X|} \sum_{i=1}^{|X|} L_i}_{\text{data loss}} + \underbrace{\lambda R(\mathcal{W})}_{\text{regularization loss}}$$

dove

•

$$L_i = -\log \left( \frac{e^{s_i}}{\sum_j e^{s_j}} \right) \text{ o equivalentemente } L_i = -s_i + \log \left( \sum_j e^{s_j} \right)$$

è il contributo alla cross-entropy loss di un singolo esempio del dataset. Esso è tanto maggiore quanto più la rete si comporta "male" sull'esempio  $\mathbf{x}^{(i)}$ . Si noti che nella prima parentesi è utilizzata la funzione softmax( $s_j$ ).

- $\mathcal{W}$  è l'insieme di tutti i parametri addestrabili della rete.

- 
- $R(\mathcal{W})$ , detto penalità di regolarizzazione, può essere scelto tra varie funzioni; quella che utilizzeremo in questo lavoro è la cosiddetta norma L2:

$$R(\mathcal{W}) = \sum_l \sum_k W_{l,k}^2$$

dove  $W_{l,k}$  rappresenta la  $k$ -esima matrice di pesi (e bias, con il bias trick) nell' $l$ -esimo layer della rete, e l'elevamento a potenza della matrice è da intendersi come la somma dei quadrati di tutti i suoi elementi. Esso è tanto maggiore quanto più i parametri addestrabili della rete si discostano da 0. Minimizzare il termine  $\lambda R(\mathcal{W})$  significa allora avvicinare i parametri a 0, tanto più velocemente ad ogni iterazione quanto più  $\lambda$  (iperparametro) è grande.  $\lambda R(\mathcal{W})$

È evidente che addestrare la rete significa allora risolvere un problema di ottimizzazione, che consiste nel minimizzare la loss function calcolata su tutto il training set. La minimizzazione può avvenire con il classico metodo del calcolo numerico della discesa del gradiente, visualizzato in fig. 2.27a.

Nella pratica, tuttavia, pensare di calcolare la loss function su tutto il training set è impensabile se la cardinalità del training set è molto elevata. Allora, ad ogni iterazione dell'addestramento, si preferisce utilizzare una versione approssimata della loss function, calcolata su un sottoinsieme piccolo di esempi del training set. La minimizzazione della loss function avviene in questo caso con il metodo della discesa stocastica del gradiente (par. 2.9.3 e fig. 2.27b).

### 2.9.3 Stochastic gradient descent

Si può immaginare il grafico della funzione  $L$  come una "vallata multidimensionale", differenziabile ovunque<sup>15</sup> e con numerosi minimi locali, uno (o alcuni) dei quali può essere un minimo globale, come mostrato in figura 2.26 in 3 dimensioni, nel caso (non molto plausibile ma semplice da visualizzare) di rete con soli due parametri addestrabili.

Minimizzare la loss function significa individuare un<sup>16</sup> suo punto di minimo globale, partendo da un punto casuale e seguendo la direzione opposta a quella del **gradiente di  $L(\mathcal{W})$**  (la "discesa" verso il punto di minimo è cioè "guidata" dal gradiente di  $L$ ). Il gradiente di  $L(\mathcal{W})$  si indica con  $\nabla L(\mathcal{W})$ .

Il metodo della **discesa stocastica del gradiente** (*stochastic gradient descent*, SGD), usato per minimizzare la loss function, prevede di calcolare  $L$  non più sull'intero training set ma su un suo sottoinsieme detto **mini-batch**, costituito da un numero ridotto di esempi (valori tipici sono 16, 32, 64, 128 o 256). La  $L^*$  così ottenuta è ovviamente una approssimazione della  $L$ , tanto più buona quanto più il mini-batch è grande e "vario" (cioè quanto più gli esempi di ogni classe sono proporzionalmente presenti nel mini-batch). L'addestramento prevede allora che un'epoca di addestramento sia in realtà divisa in  $\frac{|X|}{|\text{mini-batch}|}$  iterazioni per epoca, in ognuna

---

<sup>15</sup> poiché composizione di funzioni differenziabili

<sup>16</sup> l'esistenza di un minimo globale non ne implica l'unicità: possono esserci più minimi locali di  $L$  di uguale valore, che potrebbero anche essere minimi globali di  $L$ .

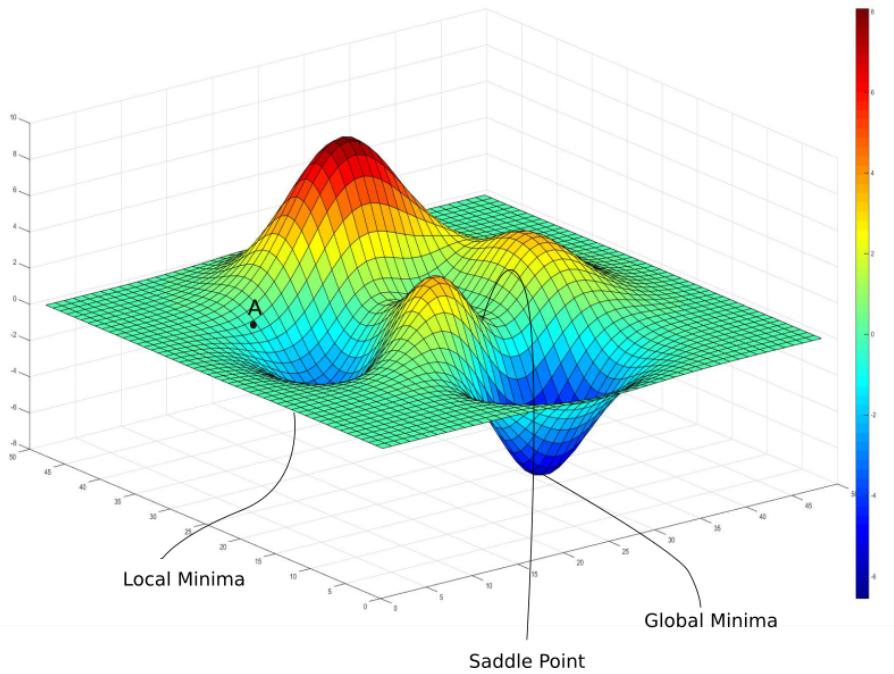


Figura 2.26: Grafico di una loss function con due sole variabili

delle quali si presenta alla rete un mini-batch, si calcola la relativa  $L^{*(i)}(\mathcal{W})$ , si calcola il gradiente  $\nabla L^{*(i)}(\mathcal{W})$  e si effettua un "salto"  $\Delta\mathcal{W}^{(i)}$  da un punto (calcolato all'iterazione precedente in base alla  $L^{*(i-1)}$ ) ad un altro del dominio di  $L^{*(i)}$ . Il salto  $\Delta\mathcal{W}^{(i)}$  è un vettore di  $|\mathcal{W}|$  componenti, ciascuna delle quali rappresenta l'aggiornamento  $\Delta w^{(i)}$  del parametro  $w^{(i)}$  ed è dato da (supponendo  $w^{(i)}$  appartenente all' $l$ -esimo layer):

$$\Delta w^{(i)} = -\text{glr} \times \text{llr}_l \times \frac{\partial L^{*(i)}}{\partial w^{(i)}}$$

dove con **glr** = **learning rate globale** e **llr<sub>l</sub>** = **learning rate locale** dell' $l$ -esimo layer. Questi sono iperparametri che caratterizzano rispettivamente la rete e i layer addestrabili della rete.

Il metodo della discesa del gradiente classico e quello stocastico sono confrontati in figura 2.27.

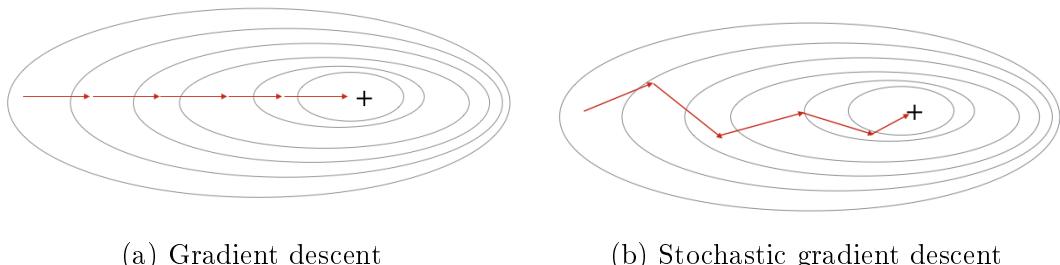


Figura 2.27

Alcune importanti osservazioni:

- 
- La discesa del gradiente converge verso uno dei punti di minimo locale di  $L$ , non necessariamente in un punto di minimo globale come invece si desidera. Si può dimostrare che la regolarizzazione dei parametri aiuta ad appiattire il grafico di  $L$ , rendendo i minimi locali e il minimo globale tanto più vicini quanto più i parametri sono "regolarizzati". In queste condizioni, non è un problema il particolare punto di minimo locale in cui la rete converge alla fine dell'addestramento.
  - Nel metodo SGD, ad ogni iterazione il grafico della loss function varia. Se il mini-batch è, come discusso in precedenza, abbastanza grande e vario, le approssimazioni  $L^{*(i)}$  sono abbastanza simili tra loro e a  $L$  stessa, quindi in questa ipotesi possiamo affermare con buona approssimazione che il punto di minimo verso cui il metodo SGD sta convergendo rimane inalterato ad ogni iterazione.
  - La scelta del learning rate globale e di quelli locali è fondamentale: un learning rate più grande porta ad una convergenza più veloce del metodo SGD verso un punto di minimo locale di  $L^{*(i)}$ , ma si corre il rischio che dopo molte iterazioni il metodo non sia ancora del tutto "sceso" in uno dei minimi. Infatti per convergere ad un punto di minimo quando si è nelle sue vicinanze c'è bisogno di fare salti di piccola ampiezza verso di esso, a causa della forma a "campana rovesciata" del grafico della funzione attorno ai suoi minimi. Al contrario, un learning rate più piccolo comporta una certa lentezza nella convergenza ma assicura, dopo un sufficiente numero di iterazioni, una loss function più vicina ad un suo minimo. Si può allora mediare tra i due bisogni prevedendo una strategia di **annealing** (decadimento) del learning rate nel tempo. Ci sono tre tipi comuni di annealing:
    - Step decay: si riduce il learning rate di un certo fattore dopo un certo numero di epoche (ad esempio si dimezza il learning rate ogni 5 epoche)
    - Exponential decay: si riduce il learning rate seguendo la funzione  $lr = lr_0 e^{-kt}$ , dove  $lr_0$  è il learning rate iniziale,  $k$  è un iperparametro che regola la velocità di decadimento,  $t$  è il "tempo" trascorso (di solito è il numero di iterazioni trascorse o, più comunemente, il numero di epoche trascorse).
    - $1/t$  decay: si riduce il learning rate seguendo la funzione  $lr = lr_0 / (1 + kt)$ ; le variabili che compaiono hanno lo stesso significato del punto precedente.

## SGD with momentum

Una variante frequentemente adoperata del metodo SGD è il metodo di **discesa stocastica del gradiente con momento** (*stochastic gradient descent with momentum*, SGDM). Il **momento** è un termine dipendente dalle iterazioni precedenti, che viene sommato al gradiente nel tentativo di regolarizzare il movimento nello spazio dei parametri. La differenza con l'SGD classico consiste nel salto da compiere nel dominio di  $L^{*(i)}$ : ad ogni iterazione il salto  $\Delta \mathcal{W}^{(i)}$  ha componente (lungo la

dimensione del generico parametro  $w^{(i)}$  appartenente all' $l$ -esimo layer):

$$\Delta w^{(i)} = -\text{glr} \times \text{llr}_l \times \frac{\partial L^{*(i)}}{\partial w^{(i)}} + \underbrace{\mu \times \Delta w^{(i-1)}}_{\text{momento}}$$

L'effetto del momento è simile a quello che in fisica ha la quantità di moto (in inglese *momentum*) nello spostamento di un oggetto: così come una forza che sposta un oggetto in una direzione gli fa acquistare velocità (quindi quantità di moto) in quella direzione, l'opposto del gradiente della loss function fa acquistare "velocità" nello spostamento verso il punto di minimo di  $L^{*(i)}$ , cosicché se il gradiente ha una certa stabilità nella sua direzione per più iterazioni, nell'iterazione successiva il salto avrà una componente lungo quella direzione, a prescindere dal particolare comportamento del gradiente in quella iterazione, "per inerzia".

Questo metodo introduce un ulteriore iperparametro  $\mu$  (compreso tra 0 e 1) per controllare l'effetto del momento, ovvero l'importanza dell'iterazione passata: più è grande  $\mu$  e più il termine momento assume importanza nell'aggiornamento  $\Delta W^{(i)}$ . Un valore empirico per tale parametro è solitamente fissato intorno a 0.9.

Il metodo SGDM è visualizzato in fig. 2.28

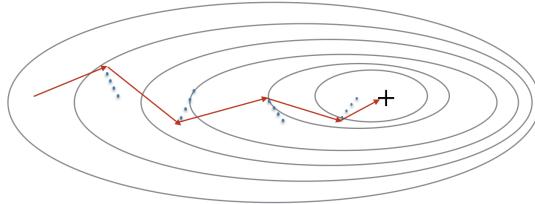


Figura 2.28: Stochastic gradient descent with momentum

#### 2.9.4 Backpropagation

Il calcolo del gradiente della loss function, nell'ambito del metodo SGD e SGDM, avviene analiticamente con il metodo di (*error*) **backpropagation** (retropropagazione dell'errore). Questo metodo ricorre all'uso ricorsivo della "regola della catena" della derivazione per calcolare le derivate parziali  $\frac{\partial L^{*(i)}}{\partial w^{(i)}}$ ; il calcolo si basa solamente sui valori di output della rete (i punteggi finali per ogni classe). Ad esempio, per una rete neurale standard (con soli strati completamente connessi seguiti ciascuno da una funzione di attivazione  $g$ ) l'algoritmo di backpropagation calcola:

$$\frac{\partial L^{*(i)}}{\partial w_{ij}^k} = g'(a_j^k) o_i^{k-1} \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}$$

dove  $w_{ij}^k$  è il peso del collegamento tra il neurone  $j$  del layer  $k$  e il neurone  $i$  del layer  $k-1$ ,  $a_j^k$  è il valore dell'attivazione del neurone  $i$  del layer  $k$  prima dell'applicazione di  $g$ ,  $o_i^k$  è l'output del neurone  $i$  nel layer  $k$ ,  $r_k$  è il numero di nodi nel layer  $k$ ,  $\delta_j^k = \frac{\partial L^{*(i)}}{\partial a_j^k}$ .

Per maggiori approfondimenti sull'algoritmo di backpropagation, si rimanda ad esempio a [11].

---

### 2.9.5 Scomparsa ed esplosione del gradiente

Il problema della **scomparsa del gradiente** (*vanishing gradient problem*) è un fenomeno che crea difficoltà nell'addestramento delle reti neurali profonde tramite SGD/SGDM e backpropagation.

Una delle principali cause è la presenza di funzioni di attivazione non lineari classiche, come la tangente iperbolica o la sigmoide, che hanno gradiente a valori nell'intervallo  $(0, 1)$ . Poiché nell'algoritmo di retropropagazione i gradienti ai vari livelli vengono moltiplicati tramite la regola della catena, il prodotto di  $n$  numeri in  $(0, 1)$  decresce esponenzialmente rispetto alla profondità  $n$  della rete, rendendo quasi nullo l'aggiornamento dei parametri negli strati più bassi della rete.

Inoltre, valori grandi (in valore assoluto) passati alle funzioni di attivazione ci-tate si trovano in zone del dominio delle funzioni dette "di saturazione", in cui la derivata è prossima allo 0. Ciò acuisce il problema numerico sopracitato.

Quando invece il gradiente delle funzioni di attivazione può assumere valori elevati, un problema analogo che può manifestarsi è quello dell'**esplosione del gradiente** (*exploding gradient problem*).

Come si vedrà nei paragrafi dedicati alla descrizione delle reti neurali convoluzionali utilizzate negli esperimenti, il problema della scomparsa e dell'esplosione del gradiente viene risolto tipicamente usando funzioni di attivazioni ReLU e, dall'avvento di ResNet, le cosiddette *skip connections* (par. 2.14.2).

## 2.10 Transfer Learning

Nella pratica, poche persone costruiscono e addestrano reti neurali da zero con inizializzazione casuale dei parametri (*from scratch*) per risolvere un task, poiché è relativamente raro disporre di un training set di dimensioni adeguate. Un'alternativa è riutilizzare una rete addestrata su un dataset molto grande (es. ImageNet, par. 2.3.3) e ri-addestrandola per risolvere il task di interesse, con dovuti accorgimenti che saranno spiegati nel seguito. Questa tecnica si chiama **transfer learning**.

Per esempio, possiamo riutilizzare una rete che è stata addestrata a classificare gatti e cani ri-addestrandola per riconoscere invece formiche e vespe. Se il dataset utilizzato nel task originale è più grande di quello disponibile per il secondo task, il transfer learning è in grado di aumentare significativamente la capacità di generalizzazione (e quindi le performance) del classificatore. Il motivo per cui il transfer learning funziona è che molte categorie condividono alcune caratteristiche di basso livello come bordi, forme geometriche, cambi nell'intensità dell'illuminazione ecc [18]. In generale, quindi, il transfer learning è una forma di representation learning usata con successo quando esistono concetti comuni ad un basso livello di astrazione tra le categorie del primo e del secondo task.

L'applicazione del transfer learning è schematizzata in fig. 2.29

Il metodo del transfer learning per l'addestramento di un modello di classificazione consiste nei seguenti passi:

1. Selezione di una rete pre-addestrata

## 2.10. TRANSFER LEARNING

---

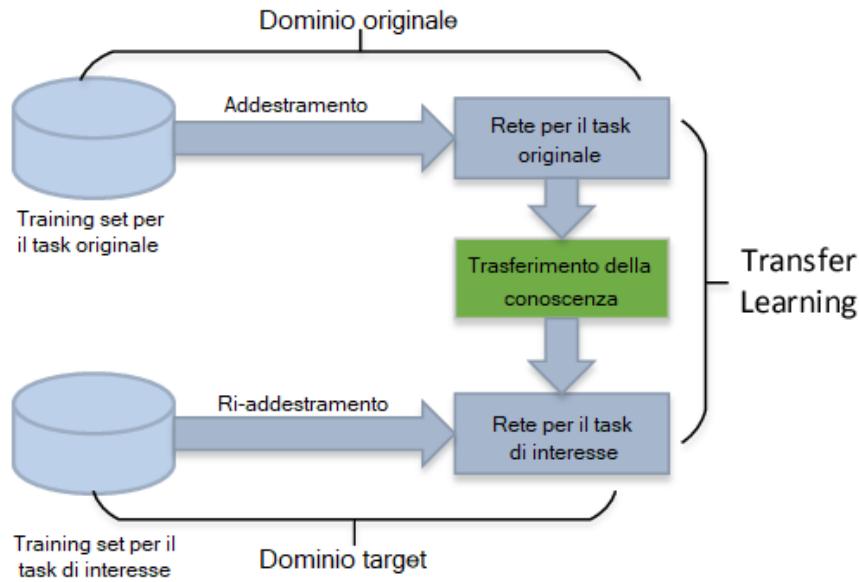


Figura 2.29: Schema di applicazione del transfer learning

Al giorno d'oggi è estremamente facile reperire reti pre-addestrate: molte istituzioni di ricerca rendono pubblici modelli addestrati su dataset ampi e complessi (es. ImageNet), che possono essere riutilizzati per la tecnica del transfer learning.

### 2. (Opzionale) Azzerare i learning rate dei layer addestrabili più bassi

Questo passaggio ha l'obiettivo di "congelare" l'apprendimento dei layer che rappresentano le feature di livello più basso delle immagini, cioè inibire modifiche ai parametri addestrabili di quei layer. Generalmente, più piccolo è il training set a disposizione per il task di interesse e più layer iniziali andrebbero congelati; con pochi dati, infatti, c'è il rischio di overfitting della rete, problema aggrabile limitando la capacità di rappresentazione del modello che si vuole addestrare.

### 3. (Opzionale) Aumentare i learning rate dei layer addestrabili finali

Questo passaggio ha l'obiettivo di velocizzare l'apprendimento dei layer che rappresentano le feature di livello più alto delle immagini, al fine di migliorare la capacità della rete di mettere insieme le feature estratte nei livelli intermedi per effettuare la predizione.

### 4. Rimpiazzare il layer di classificazione finale

Si rimpiazza questo layer per cambiare le classi da predire: non più le classi del task originale (es. le mille classi di ImageNet) ma quelle del task di interesse (es. 'Pinna' e 'No Pinna').

### 5. Addestrare la rete.

---

La rete pre-addestrata, optionalmente modificata a livello dei suoi learning rate locali, è usata come punto di partenza per il nuovo addestramento, che la porterà ad adattarsi al task di interesse.

## 6. Misurare le prestazioni del modello "trasferito" con un test set.

Il transfer learning è utilizzato nel presente lavoro per adattare quattro modelli pre-addestrati al task di classificazione delle pinne dorsali dei cetacei (cap. 3).

## 2.11 Ensemble learning

Al fine di migliorare l'accuratezza della classificazione, spesso si ricorre ad una strategia conosciuta come **apprendimento ensemble** (*ensemble learning*). L'idea chiave è quella di creare un insieme (*ensemble*) di classificatori di immagini, ciascuno dei quali è chiamato a "votare" circa l'esito della predizione; a seconda dello "schema di consenso" (*consensus scheme*) scelto per l'ensemble, cioè a seconda di quanto peso assume ciascun voto nella classificazione finale, l'output complessivo sarà la classe che avrà ricevuto "democraticamente" il maggior consenso. La fig. 2.30 mostra lo schema di funzionamento di un classificatore ensemble.

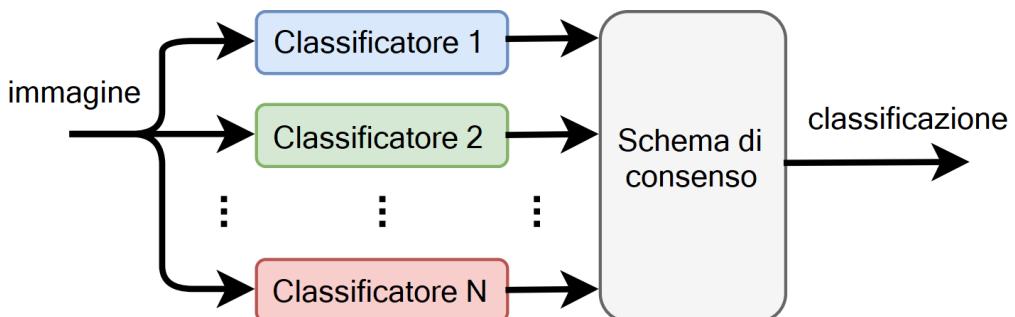


Figura 2.30: Schema di un classificatore ensemble

Gli schemi di consenso più utilizzati sono i seguenti:

- **Hard major voting** (HMV): la classe predetta per l'immagine in input è quella che ha ricevuto il maggior numero di "voti" (predizioni) da parte delle reti dell'ensemble.
- **Soft major voting** (SVM): la classe predetta per l'immagine in input è quella la cui probabilità di classificazione media (cioè la media delle probabilità softmax relative a quella classe attribuite dalle reti dell'ensemble) è la più grande.

L'efficacia dei metodi ensemble è dovuta al fatto che, solitamente, differenti modelli addestrati per risolvere uno stesso problema di classificazione non faranno tutti gli stessi errori sul *test set* (gli errori si possono cioè considerare, in buona sostanza, incorrelati). Se lo schema di consenso applicato è il *major voting*, si dimostra che l'ensemble di classificatori ha sempre prestazioni migliori o almeno uguali a quelle di

## 2.11. ENSEMBLE LEARNING

---

ciascuna rete dell'ensemble presa singolarmente; il miglioramento delle prestazioni (specificamente della *test accuracy*<sup>17</sup>) è tanto migliore quanto più gli errori commessi dalle reti dell'ensemble sono tra loro incorrelati.[11][19]

Il nuovo classificatore per i ritagli delle pinne, oggetto di sviluppo in questo lavoro di tesi, è in realtà un *ensemble* di tre reti neurali convoluzionali ri-addestrate con la tecnica del *transfer learning* (par. 3.3).

---

<sup>17</sup>training/validation/test accuracy = 1 – training/validation/test error, par. 2.4.2

## 2.12 AlexNet

AlexNet è una CNN creata tra il 2011 e il 2012 da Alex Krizhevsky, in collaborazione con Ilya Sutskever e Geoffrey Hinton [20]. La vittoria di AlexNet nella *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* (par. 2.3.3) nel 2012<sup>18</sup>, ottenuta con un netto distacco nei confronti degli altri concorrenti, ha segnato l'inizio dell'enorme successo ottenuto dalle reti neurali profonde in svariati domini di applicazione [21].

Il risultato principale di AlexNet, così come dichiarato dai suoi creatori nell'articolo originale, è il fatto che la profondità del modello è stato essenziale per conferirgli prestazioni così alte. Il costo computazionale dell'addestramento di AlexNet, reso oneroso appunto dalla profondità del modello (e quindi dal grande numero di parametri - circa 62.3 milioni), è stato affrontato con l'impiego di schede grafiche (GPU), che cominciavano in quegli anni a raggiungere notevoli potenze di calcolo.

### 2.12.1 Architettura di AlexNet

L'architettura di AlexNet è riportata schematicamente nella figura 2.31 e con maggiore dettaglio in tabella 2.1. La rete accetta in input immagini  $227 \times 227$ . Essa si compone di otto layer con parametri - cinque convoluzionali e tre completamente connessi. L'output dell'ultimo layer completamente connesso passa per un softmax layer a 1000 vie, il quale fornisce la distribuzione di probabilità per le 1000 classi del dataset ImageNet.

Tra ognuno degli otto strati parametrizzati sono interposti alcuni strati intermedi: ReLU layer, Local Response Normalization layer, Max Pooling layer, Dropout layer. Ognuno di questi sarà analizzato in maggiore dettaglio nei paragrafi successivi.

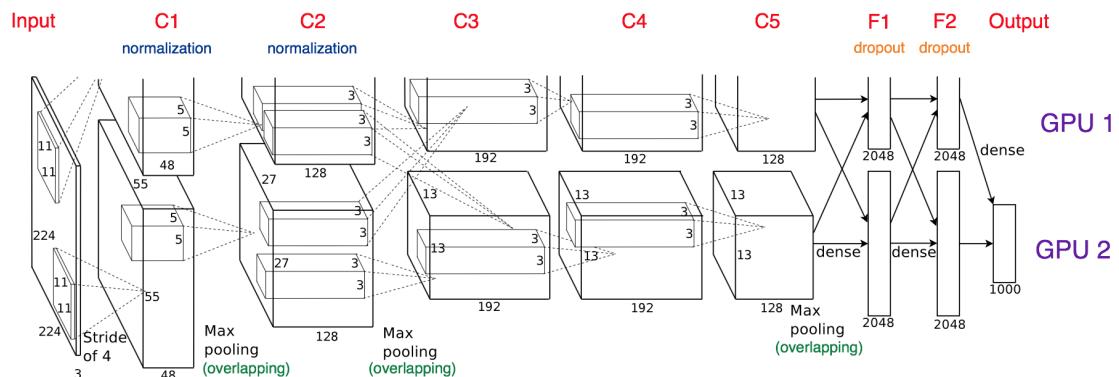


Figura 2.31: Architettura originale di AlexNet [20]

Come si evince dalla figura 2.31, la rete è composta da due "pipeline" parallele. Si scelse infatti di "estendere" la rete su due GPU NVIDIA® GeForce® GTX 580 3GB in fase di training, per raddoppiare la memoria massima disponibile (6GB in totale) per conservare la rete e i suoi parametri.

Queste GPU si prestano bene a lavorare in parallelo, poiché possono leggere e scrivere l'una sull'altra direttamente, senza passare dalla memoria della macchina

<sup>18</sup><http://image-net.org/challenges/LSVRC/2012/results>

host. Lo schema di parallelizzazione a due vie prevede che su ogni GPU risieda la metà dei kernel (o dei neuroni) di ciascuno strato parametrizzato. Le GPU possono comunicare tra loro solo in certi strati. In particolare, i kernel del layer convoluzionale 1 e 3 hanno in input l'intero output volume rispettivamente del layer di input e del layer convoluzionale 2, mentre i kernel dei rimanenti strati convoluzionali hanno in input la sola metà dell'output volume presente nella stessa GPU (*grouped convolution*<sup>19</sup>).

Sono di seguito passate in rassegna le principali scelte architetturali introdotte in AlexNet, ed alcuni dettagli relativi al suo addestramento.

### 2.12.2 Funzione di attivazione ReLU

Dopo ogni strato parametrizzato, i valori delle attivazioni sono passati alla funzione attivatrice "rettificatore":  $f(x) = x^+ = \max(0, x)$  [22]. Questa funzione attivatrice non-lineare e non soggetta a saturazione permette un addestramento molto più veloce delle reti convoluzionali profonde, in confronto a funzioni attivatrici fino ad allora più utilizzate come la funzione sigmoidea  $f(x) = (1 + \exp^{-x})^{-1}$  e la funzione tangente iperbolica  $f(x) = \tanh(x)$ .

### 2.12.3 Local Response Normalization

È stato verificato che la seguente normalizzazione delle attivazioni, *Local Response Normalization*, aumenta lievemente la capacità di generalizzazione del modello:

$$b_{x,y}^i = a_{x,y}^i / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^j)^2 \right)^\beta$$

dove  $a_{x,y}^i$  è l'attivazione del neurone ottenuto applicando il kernel  $i$ -esimo alla posizione  $(x, y)$  e applicando in seguito la funzione ReLU,  $b_{x,y}^i$  l'attivazione normalizzata,  $N$  il numero totale di kernel del layer corrente,  $k, n, \alpha, \beta$  sono iperparametri; sono stati usati i valori  $k = 2, n = 5, \alpha = 10^{-4}, \beta = 0.75$ .

Questa normalizzazione è adoperata solamente nel primo e nel secondo layer convoluzionale.

### 2.12.4 Overlapping Max Pooling

La funzione di max pooling in AlexNet è stata caratterizzata dalla scelta di una dimensione del filtro di pooling  $3 \times 3$  e uno stride di 2 (producendo quindi una sovrapposizione, o *overlap*, tra le regioni sottoposte a pooling). È stato osservato durante la fase di training che questa funzione di *max pooling con sovrapposizione* ha attenuato lievemente l'*overfitting* della rete.

---

<sup>19</sup>La scelta di questo pattern di connettività fra le due GPU parallele è il risultato di un problema di cross-validation.

---

### 2.12.5 Data Augmentation

Una delle difficoltà che si incontrano spesso quando si vuole addestrare una rete neurale con moltissimi parametri avendo a disposizione un dataset relativamente piccolo è il rischio del sovradattamento (*overfitting*) della rete al training set, che compromette anche seriamente le prestazioni della rete quando le vengono presentati nuovi dati. In AlexNet l'overfitting è stato ridotto grazie a tecniche di *data augmentation*. In particolare, dopo aver ridimensionato a  $256 \times 256$  tutte le immagini del training set (scalando prima in modo tale che il lato più corto dell'immagine sia di 256 e in seguito ritagliando un quadrato centrale  $256 \times 256$  dall'immagine risultante) e sottraendo l'*immagine media* da ciascuna immagine del training set, il training set così ottenuto è stato "arricchito" con le seguenti immagini:

- Estrazione casuale di ritagli  $224 \times 224$ <sup>20</sup> dalle immagini
- Riflessione orizzontale ("a specchio") delle immagini (50% di probabilità)
- Somma di un'immagine e le sue componenti principali (PCA)<sup>21</sup>

### 2.12.6 Dropout

Un altro modo per ridurre il problema del sovradattamento è l'impiego di tecniche di regolarizzazione dei parametri. AlexNet utilizza la tecnica del *dropout* [24]. Questa tecnica consiste nel settare a zero l'attivazione di ciascun neurone di un layer intermedio con probabilità  $p$  (AlexNet impiega un dropout con  $p = 0.5$ ), e solo in fase di addestramento. I neuroni "azzerati" sono essenzialmente eliminati dalla rete e non contribuiscono né alla propagazione all'indietro del gradiente né al calcolo delle attivazioni nello strato finale (in fase di addestramento). Questa tecnica riduce il *co-adattamento* tra neuroni: ogni neurone non può fare affidamento sulla presenza di altri neuroni, ed è costretto ad apprendere feature utili in congiunzione con diversi sottoinsiemi casuali degli altri neuroni, e non con un solo particolare sottoinsieme, migliorando la generalizzazione su nuovi dati. In fig. 2.32 è mostrato un esempio di dropout applicato ad una rete neurale standard.

In AlexNet, il dropout dei neuroni è utilizzato nei primi due layer completamente connessi. In fase di test non si utilizza il dropout, e i neuroni di questi due strati sono moltiplicati per 0.5 per tenere conto dell'impiego del dropout in fase di addestramento.

---

<sup>20</sup>Le immagini vengono portate a  $227 \times 227$ , cioè la risoluzione di input, attraverso uno *zero-padding* (3 pixel aggiuntivi ai bordi)

<sup>21</sup>L'*analisi delle componenti principali* (PCA, principal component analysis) è una tecnica per la semplificazione dei dati utilizzata nell'ambito della statistica multivariata. In questa sede ci limitiamo a specificare che il suo utilizzo nell'ambito della data augmentation è di evidenziare una importante proprietà delle immagini naturali, e cioè che l'identità di un oggetto è invariante rispetto ai cambi d'intensità e di colori nella sua illuminazione. Si rimanda ad esempio a [23] per approfondimenti sulla PCA.

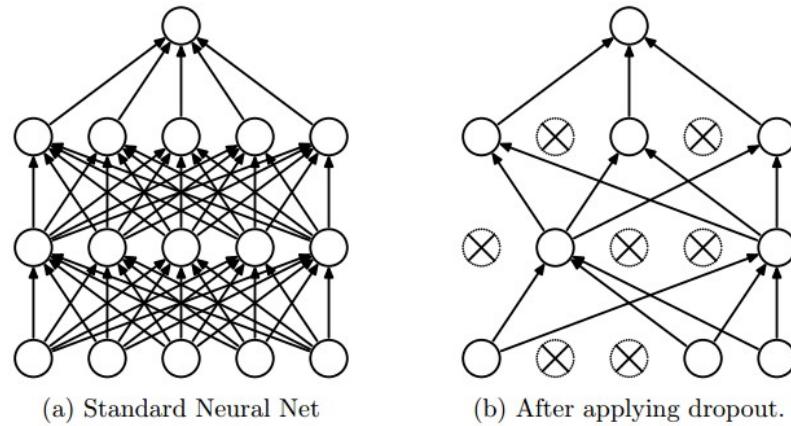


Figura 2.32: Figura tratta dal paper originale [24]. Durante l'addestramento, il dropout può essere immaginato come una "sotto-rete" neurale, che è soggetta ad addestramento al posto della rete originale.

### 2.12.7 Addestramento di AlexNet

Nella sua forma originale, AlexNet è stata addestrata usando la discesa stocastica del gradiente con momento = 0.9, mini-batch = 128 e decadimento dei pesi (weight decay) = 0.0005. Il training set è ovviamente quello di ImageNet. I pesi in ogni layer sono stati inizializzati con valori aleatori provenienti da una distribuzione gaussiana a media nulla e deviazione standard 0.01. I bias del secondo, quarto e quinto layer convoluzionale e dei tre layer completamente connessi sono stati inizializzati a 1; questa inizializzazione accelera le prime iterazioni dell'addestramento facendo in modo che alle funzioni di attivazione ReLU siano forniti input positivi. Nei rimanenti layer, i bias sono stati inizializzati a 0. Il *learning rate* iniziale è stato 0.01 ed è stato sottoposto ad una strategia di *annealing* che ha previsto la riduzione del learning rate di un fattore 10 ogniqualvolta il *validation error* si stabilizzava (questa regolazione è stata manuale durante l'addestramento). Questa regolazione è avvenuta tre volte durante l'addestramento della rete.

Ulteriori dettagli sulla fase di addestramento di AlexNet possono essere trovati nel paper originale [20].

N	Layer	Attivazioni	Parametri
1	INPUT	(227 × 227 × 3)	
2	CONVOLUTION	(55 × 55 × 96)	Pesi: (11 × 11 × 3) × 96 Bias: (96)
3	RELU	—	—
4	NORMALIZATION	—	—
5	MAX POOLING	(27 × 27 × 96)	—
6	GROUPED CONVOLUTION	(27 × 27 × 256)	Pesi: (5 × 5 × 48) × 128 × 2 Bias: (128) × 2
7	RELU	—	—
8	NORMALIZATION	—	—
9	MAX POOLING	(13 × 13 × 256)	—
10	CONVOLUTION	(13 × 13 × 384)	Pesi: (3 × 3 × 256) × 384 Bias: (384)
11	RELU	—	—
12	GROUPED CONVOLUTION	(13 × 13 × 384)	Pesi: (3 × 3 × 192) × 192 × 2 Bias: (192) × 2
13	RELU	—	—
14	GROUPED CONVOLUTION	(13 × 13 × 256)	Pesi: (3 × 3 × 192) × 128 × 2 Bias: (128) × 2
15	RELU	—	—
16	MAX POOLING	(6 × 6 × 256)	—
17	FULLY CONNECTED	4096	Pesi: 4096 × 9216 Bias: 4096
18	RELU	—	—
19	DROPOUT	—	—
20	FULLY CONNECTED	4096	Pesi: 4096 × 4096 Bias: 4096
21	RELU	—	—
22	DROPOUT	—	—
23	FULLY CONNECTED	1000	Pesi: 2 × 4096 Bias: 2
24	SOFTMAX	—	—
25	CROSS-ENTROPY LOSS	—	—

Tabella 2.1: Architettura originale di AlexNet

## 2.13 GoogLeNet

GoogLeNet (in origine *Inception v1*) è una rete neurale convoluzionale profonda presentata nel 2014 da Christian Szegedy, ricercatore presso Google, ed il suo team di ricerca. La pubblicazione del paper originale [25] è avvenuta poco dopo la schiacciante vittoria riportata da GoogLeNet nella *ILSVRC 2014*<sup>22</sup>, sia nel problema di *object classification* che in quello di *object detection*, introdotto nell'anno precedente.

Il grande contributo di GoogLeNet nell'ambito del deep learning è il miglioramento nell'utilizzo delle risorse computazionali da parte di una rete profonda, grazie all'introduzione del modulo *Inception* (par. 2.13.2). In particolare, se confrontata ad AlexNet (par. 2.12), GoogLeNet utilizza circa 10 volte meno parametri (circa 6,8 milioni) essendo però significativamente più profonda (22 layer con parametri) e performante (errore *top-5* su dataset ImageNet 6,7%).

GoogLeNet è stata progettata per risolvere alcuni problemi tipici delle reti convoluzionali particolarmente profonde:

- L'operazione di convoluzione su volumi molto profondi è computazionalmente onerosa. Per risolvere questo problema, GoogLeNet introduce l'operazione di convoluzione  $1 \times 1$  (par. 2.13.1).
- Le reti neurali molto profonde sono molto sensibili al rischio di *overfitting*, a causa del loro alto grado di astrazione e rappresentazione dei concetti (dovuto al grandissimo numero di parametri). Questo problema è attenuato grazie all'introduzione della già citata operazione di convoluzione  $1 \times 1$ , all'operazione di *global average pooling* (par. 2.13.4) e all'usuale pratica della *image augmentation* (par. 2.13.5).
- Le parti salienti di un'immagine possono avere delle dimensioni estremamente variabili all'interno di essa. Si guardi ad esempio la figura seguente:



Figura 2.33: Tre immagini di cani. L'area occupata da ciascun cane è differente e via via più piccola in ogni immagine

A causa di questa grande variabilità nella localizzazione dell'informazione la scelta delle dimensioni dei filtri convoluzionali diventa complessa. Un filtro

---

<sup>22</sup><http://image-net.org/challenges/LSVRC/2014/>

---

più ampio è preferito quando l'informazione è distribuita su un'area vasta dell'immagine; un filtro più piccolo è adeguato in quei casi in cui l'informazione è localizzata in un'area più ristretta.

Questo problema è risolto con l'introduzione dei moduli *inception* (par. 2.13.2).

- A causa della profondità di una rete, un altro tipico problema che si presenta durante l'addestramento è la scomparsa del gradiente (*vanishing gradient problem*, par. 2.9.5), nella fase di *backpropagation*. Questo problema è risolto prevedendo una particolare architettura della rete, che in fase di addestramento risulta in realtà composta da tre sottoreti diverse (par. 2.13.3).

### 2.13.1 Convoluzione $1 \times 1$

Mutuando un'idea precedentemente esposta in [26], GoogLeNet introduce l'operazione di convoluzione  $1 \times 1$  (seguita dalla funzione di attivazione ReLU). Lo scopo di questa operazione è quello di ridurre le dimensioni (in particolare, la profondità) di un volume di attivazioni per ridurre il costo computazionale delle operazioni di convoluzione successive. Questo permette di ottenere reti più profonde ma anche - con il modulo *inception* (par. 2.13.2) - più "larghe". La fig. 2.34 mostra schematicamente l'applicazione di un'operazione di convoluzione  $1 \times 1$ .

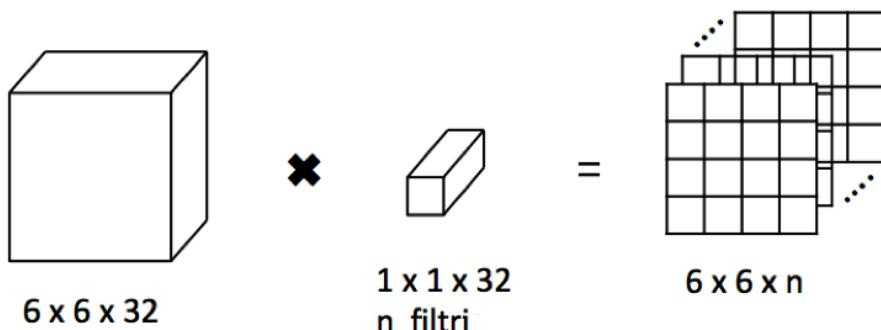


Figura 2.34: Convoluzione  $1 \times 1$  applicata ad un volume di attivazioni  $6 \times 6 \times 32$ . La profondità del volume di output dipende dal n. di filtri  $1 \times 1$  adoperati. In questo caso si ottiene una riduzione della profondità per  $n < 32$ .

Grazie all'introduzione della convoluzione  $1 \times 1$ , inoltre, il numero di pesi nei kernel delle operazioni di convoluzione successive diminuiscono, riducendo così il rischio di overfitting.

A conclusione del paragrafo viene riportata un'osservazione interessante. Le reti neurali artificiali "standard" (contenenti cioè solo strati completamente connessi) possono essere convertite in reti convoluzionali mediante sole convoluzioni  $1 \times 1$ . Infatti nelle ANN con soli strati completamente connessi tutti i volumi su cui si opera hanno dimensione  $1 \times 1 \times k$ , dove  $k$  è variabile ed è il numero  $k$  di neuroni dello strato che ha generato tale volume; si può allora pensare a ciascuno di questi

## 2.13. GOOGLENET

---

strati completamente connessi con  $k$  neuroni (biunivocamente associati al volume che generano in output) come ad uno strato di convoluzione  $1 \times 1$  che applica  $k$  filtri al volume in input. Ogni neurone del volume di output  $1 \times 1 \times n$  ha così una completa connettività con tutti i neuroni del volume (e quindi dello strato completamente connesso) precedente.

### 2.13.2 Modulo *Inception*

Per ovviare al problema della variabilità dell'area occupata da un oggetto da classificare in un'immagine, l'idea chiave è stata quella di operare molteplici convoluzioni, con diverse dimensioni dei kernel, in parallelo. Così facendo la rete tende a diventare più "larga" anziché più "profonda". Il modulo *inception* è stato sviluppato proprio sulla base di questa idea. In fig. 2.35 è mostrato il modulo *inception*, innestato più volte nell'architettura di GoogLeNet (par. 2.13.6)

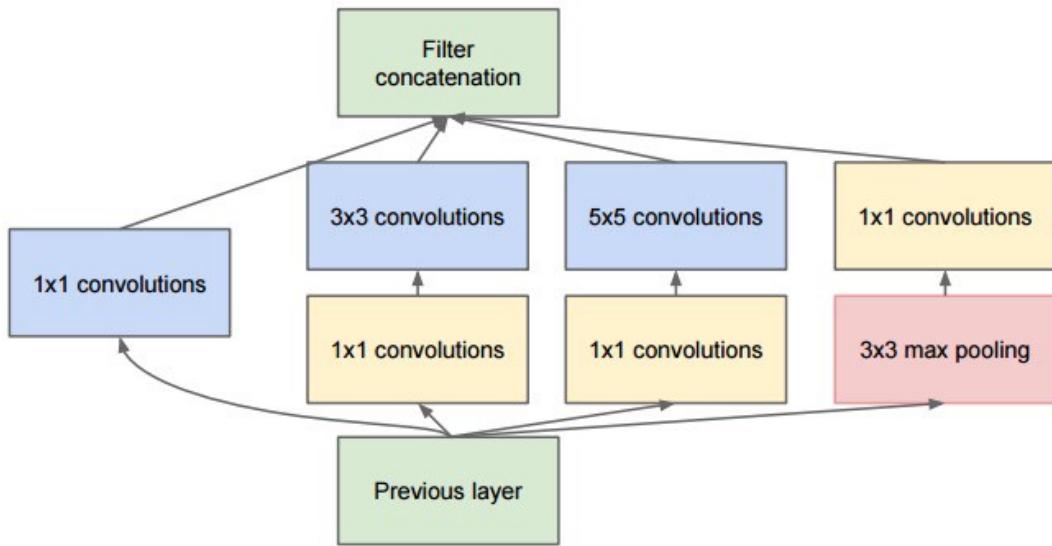


Figura 2.35: Modulo *inception*

Tale modulo riceve un volume di input da uno strato precedente e opera in parallelo su di esso tre convoluzioni (con filtri  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$  rispettivamente) e un *max pooling* con finestra  $3 \times 3$ . Per diminuire il costo computazionale delle operazioni di convoluzione, si è deciso di implementare una convoluzione  $1 \times 1$  prima delle convoluzioni  $3 \times 3$  e  $5 \times 5$  e dopo il *max pooling*  $3 \times 3$ . Gli output sono infine "impilati" tra loro lungo la dimensione della profondità, e il volume risultante viene inviato al successivo layer(o modulo *inception*).

### 2.13.3 Classificatori ausiliari

L'architettura di GoogLeNet prevede, per la sola fase di addestramento, delle "diramazioni", posizionate circa a metà della rete, evidenti in fig. 2.13.6. Questi rami della rete ospitano dei classificatori ausiliari che consistono di

- 
- Average Pooling  $5 \times 5$  (Stride 3)
  - Convoluzione  $1 \times 1$  (128 filtri)
  - ReLU layer
  - 1024 Fully Connected layer
  - 1000 Fully Connected layer
  - Dropout layer (70% di probabilità di dropout di ciascun output; vd. par. 2.12.6)
  - Softmax layer

Questa particolare architettura è stata ideata per attenuare il problema della scomparsa del gradiente (*vanishing gradient problem*, par. 2.9.5), tipico delle reti molto profonde. L'idea si basa su un'osservazione: le promettenti performance di reti meno profonde nei task di *image classification* suggeriscono che le *feature* estratte dagli strati intermedi della rete hanno un grande impatto sulla predizione finale. Pertanto, nel tentativo di propagare meglio il segnale gradiente all'indietro (cioè per amplificarlo), la funzione costo calcolata è amplificata sulla base delle predizioni dei classificatori intermedi. In fase di addestramento, dai valori di output dello strato softmax dei tre classificatori (i due intermedi e quello finale) viene calcolata la funzione costo (*cross-entropy loss*). Il costo totale sarà costituito dalla somma pesata del costo registrato dal classificatore finale (con peso 1) e quello registrato dai due classificatori ausiliari (con peso 0.3).

#### 2.13.4 Global Average Pooling

In precedenza, nella parte conclusiva delle architetture delle reti convoluzionali venivano posti due o più layer completamente connessi (ad esempio in AlexNet, par. 2.12, che presenta tre fully connected layer finali). Mutuando un'idea esposta in [26], in GoogLeNet si è previsto invece un singolo layer completamente connesso finale con 1000 neuroni (quello che fornisce valori alla funzione softmax) preceduto da uno strato di pooling denominato *global average pooling*. Questo strato opera un pooling su una superficie  $7 \times 7$  (le dimensioni di base e altezza del volume di output precedente) e restituisce una superficie  $1 \times 1$ , il cui valore è la media calcolata sui 49 valori dei neuroni della superficie. Gli autori hanno mostrato che passare da un fully connected layer a un average pooling layer ha contribuito a migliorare la *top-1 accuracy* nella ILSVRC del 0.6%, abbassando ulteriormente il numero dei pesi della rete (da circa 51.2 milioni se fosse stato utilizzato il FC layer agli 0 del global average pooling). In questo modo, GoogLeNet è stata resa anche leggermente più robusta all'*overfitting*. La fig. 2.36 mostra un confronto tra il funzionamento di un FC layer e un GAP (Global Average Pooling) layer.

## 2.13. GOOGLENET

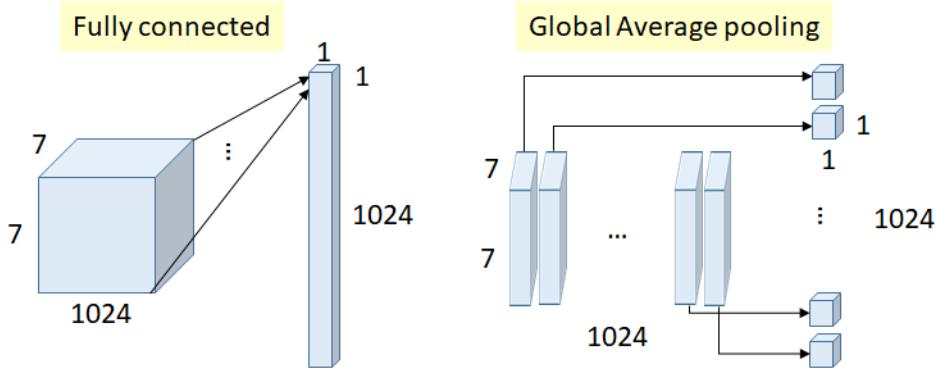


Figura 2.36: Confronto fra un FC layer e un GAP layer

### 2.13.5 Data Augmentation

Per ridurre ulteriormente l'overfitting al training set di ImageNet, GoogLeNet fa uso di una particolare tecnica di *data augmentation* (par. 2.8). Al posto delle immagini originali, per l'addestramento sono stati utilizzati dei ritagli (uno per ciascuna immagine di un mini-batch) di dimensioni distribuite uniformemente tra l'8% e il 100% dell'intera immagine e con *aspect ratio* (rapporto tra larghezza e altezza dell'immagine) distribuito uniformemente tra 3/4 e 4/3. In aggiunta, sono state operate alcune distorsioni fotometriche già adoperate in precedenza in [27].

### 2.13.6 Architettura di GoogLeNet

L'architettura di GoogLeNet è riportata in forma tabellare in tab. 2.37 e in forma grafica in fig. 2.38

type	patch size/ stride	output size	depth	#1x1	#3x3 reduce	#3x3	#5x5 reduce	#5x5	pool proj	params	ops
convolution	7x7/2	112x112x64	1							2.7K	34M
max pool	3x3/2	56x56x64	0								
convolution	3x3/1	56x56x192	2		64	192				112K	360M
max pool	3x3/2	28x28x192	0								
inception (3a)		28x28x256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28x28x480	2	128	128	192	32	96	64	380K	304M
max pool	3x3/2	14x14x480	0								
inception (4a)		14x14x512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14x14x512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14x14x512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14x14x528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14x14x832	2	256	160	320	32	128	128	840K	170M
max pool	3x3/2	7x7x832	0								
inception (5a)		7x7x832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7x7x1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7x7/1	1x1x1024	0								
dropout (40%)		1x1x1024	0								
linear		1x1x1000	1							1000K	1M
softmax		1x1x1000	0								

Figura 2.37: Architettura di GoogLeNet

---

La rete accetta in input immagini  $224 \times 224$ . I primi layer parametrizzati dall'inizio della rete sono 3 layer convoluzionali, a cui seguono 9 moduli *inception* (ognuno con 6 layer convoluzionali) e infine un fully connected layer (finale). Si evidenzia nuovamente, come già esposto nel par. 2.13.3, che le ramificazioni che ospitano i due classificatori ausiliari esistono solo in fase di addestramento, e vengono eliminati dopo di esso.

### 2.13.7 Addestramento di GoogLeNet

GoogLeNet fu addestrato sul dataset di ImageNet (par. 2.3.3) su una macchina che usava solamente le sue CPU per effettuare calcoli<sup>23</sup>, usando come algoritmo di ottimizzazione la discesa stocastica del gradiente con momento = 0.9, *learning rate*<sup>24</sup> con *annealing* del 4% ogni 8 epoche. Dettagli più specifici sulla fase di addestramento di GoogLeNet possono essere trovati nel paper originale [25].

---

<sup>23</sup>Tuttavia gli autori affermano che se fosse stato usato un ridotto numero di GPU di fascia alta l'addestramento avrebbe potuto essere effettuato in meno di una settimana (avendo come unica limitazione la memoria delle GPU stesse).

<sup>24</sup>Per la ILSVRC 2014, il team ha in realtà utilizzato un ensemble di 7 reti GoogLeNet diverse, ognuna addestrata con *learning rate* iniziale e *mini-batch size* diversi

### 2.13. GOOGLENET

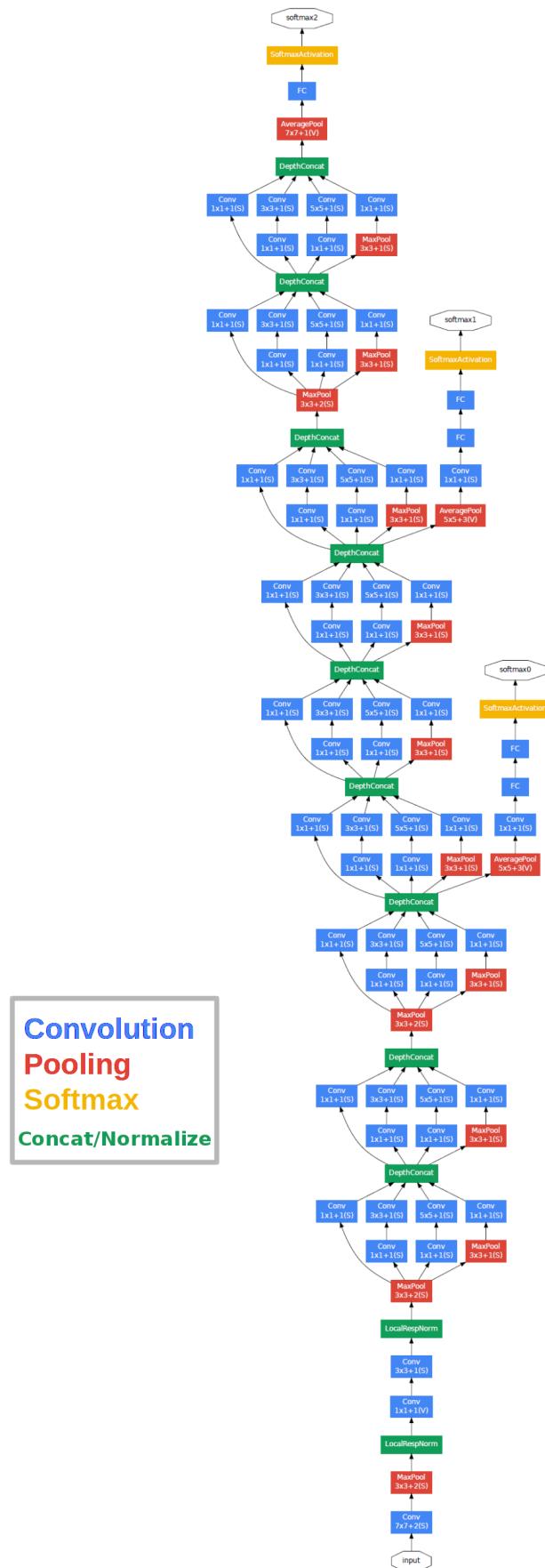


Figura 2.38: Architettura di GoogLeNet

---

## 2.14 ResNet

ResNet (abbreviazione di *residual network*) è il nome di un'ampia categoria di reti neurali convoluzionali profonde, caratterizzate dall'utilizzo di particolari blocchi di layer detti *residual blocks*, ideati per risolvere alcuni problemi delle reti molto profonde. Le reti ResNet propriamente dette sono state introdotte da un gruppo di ricercatori Microsoft nel 2015 nel paper [28]. Presentate nell'edizione del 2015 della *ILSVRC*<sup>25</sup> e risultate vincitrici con uno strabiliante errore *top-5* su dataset ImageNet 3.57% (più basso di quello umano, stimato a 5%), queste reti sono considerate tutt'oggi lo stato dell'arte nell'ambito delle reti neurali convoluzionali profonde.

Nell'ambito dei problemi di visione artificiale proposti nella ILSVRC, in quegli anni cominciava ad essere evidente ([25],[29]) che la profondità delle reti neurali era di fondamentale importanza per il miglioramento delle prestazioni delle reti su dataset di grandi dimensioni, quali il database ImageNet (par. 2.3.3). L'ovvia conseguenza di questa osservazione è stato il tentativo di aumentare progressivamente il numero di layer delle reti neurali, rendendole sempre più profonde e complesse, con l'aggravarsi di ostacoli già noti (quali ad esempio il problema della scomparsa del gradiente, par. 2.9.5) e il presentarsi di nuovi (un problema di degradazione esposto per la prima volta in [30], e descritto nel seguito): ResNet nasce per dare soluzione a questi problemi:

- Il problema della scomparsa e dell'esplosione del gradiente è attenuato da un insieme di strategie già messe in campo in precedenza, su tutte la *batch normalization* (par. 2.14.1) introdotta dal team dei creatori di GoogLeNet [31].
- Quando una rete molto profonda comincia a convergere verso un minimo della funzione costo (in fase di addestramento), si presenta un problema di degradazione: al crescere della profondità della rete la sua *training accuracy* tende a saturarsi e in seguito prende a degradarsi rapidamente, come mostrato in fig. 2.39.

È un problema inaspettato e diverso rispetto all'*overfitting* (che interessa solamente il valore della *validation accuracy*) e indica che la difficoltà nell'ottimizzare una rete convoluzionale "standard" cresce con la sua profondità. ResNet aggira questo problema con l'introduzione dei *residual blocks* (par. 2.14.2)

### 2.14.1 *Batch Normalization*

La *Batch Normalization* è il nome di una tecnica introdotta dal team di GoogLeNet e ad oggi ampiamente utilizzata che permette un addestramento più veloce e stabile delle reti neurali profonde [31].

Essa consiste in una normalizzazione delle attivazioni di un certo layer, generalmente prima di passare gli stessi ad un'eventuale funzione di attivazione ed in seguito all'eventuale layer successivo.

---

<sup>25</sup><http://image-net.org/challenges/LSVRC/2015/results>

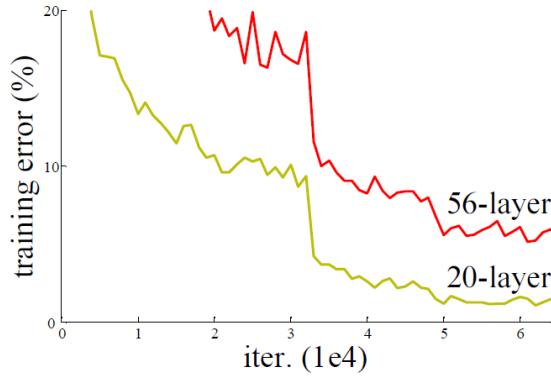


Figura 2.39: Problema di degradazione della *training accuracy* su CNN "standard" da 20 e 56 layer rispettivamente. La rete più profonda ha un *training error* più alto. [28] per più dettagli.

L'algoritmo per l'applicazione della tecnica è riportato di seguito

**Input:** Attivazioni  $\mathbf{x} = \{x_1, \dots, x_m\}$ ;  
Parametri da imparare:  $\gamma, \beta$   
**Output:** Attivazioni normalizzate  $y_i = BN_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\mu_{\mathbf{x}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{ media delle attivazioni} \\ \sigma_{\mathbf{x}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathbf{x}})^2 && // \text{ varianza delle attivazioni} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathbf{x}}}{\sqrt{\sigma_{\mathbf{x}}^2 + \varepsilon}} && // \text{ normalizzazione} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)\} && // \text{ scale e offset}\end{aligned}$$

I parametri  $\gamma$  e  $\beta$ , chiamati rispettivamente *scale* e *offset*, sono imparati dalla rete in fase di addestramento. La loro funzione è far sì che le attivazioni normalizzate non siano necessariamente a media nulla e varianza unitaria (in modo da poter variare arbitrariamente l'intervallo utilizzato nel dominio della eventuale funzione di attivazione successiva alla normalizzazione).

Nonostante sia oggi ampiamente utilizzata, le ragioni dell'efficienza della *batch normalization* sono ancora scarsamente comprese. Ricerche recenti [32] hanno mostrato che ciò che questa tecnica produce non è una riduzione dell'*internal covariate shift* (la variabilità statistica dei mini-batch usati, che ad ogni iterazione causa uno spostamento del punto di minimo della funzione costo), come affermato dagli autori del paper originale [31], ma è una "lisciatura" (*smoothing*) della funzione costo, che induce un comportamento più stabile e predicibile dei gradienti, comportando un addestramento più veloce.

---

In ogni caso, la *batch normalization* si è rivelata efficace per l'attenuazione del problema della scomparsa del gradiente o della sua esplosione; inoltre si è verificato che il suo utilizzo porta anche ad una maggiore regolarizzazione dei parametri dell'apprendimento, migliorando la robustezza all'overfitting.

Questa tecnica ha permesso quindi la progettazione di reti sempre più profonde, ma non elimina tutti i problemi collegati all'estrema profondità dell'architettura (persiste ad esempio il problema di degradazione descritto nel precedente paragrafo).

## 2.14.2 Residual blocks

Il principale contributo di ResNet nell'ambito del deep learning è sicuramente l'introduzione dei cosiddetti *residual blocks* e delle *shortcut connections* ("scorciatoie") per risolvere il problema di degradazione in precedenza descritto. In fig. 2.40 è mostrato uno schema generale dell'oggetto in esame.

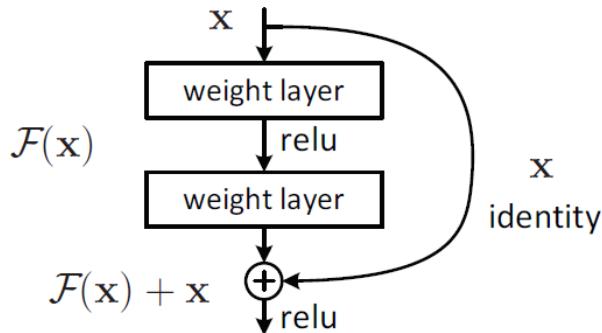


Figura 2.40: *Residual block* con *shortcut identity mapping*. Si noti che i particolari layer di questo blocco sono arbitrari.

Invece di sperare che un dato blocco di layer non lineari contigui impari ad approssimare una certa funzione  $\mathcal{H}(\mathbf{x})$  desiderata, si decide di creare una "scorciatoia" (propriamente *shortcut connection*) che connette l'input  $\mathbf{x}$  e l'output del blocco (vd. fig. 2.40); in questo modo il blocco deve imparare ad approssimare non più la funzione desiderata  $\mathcal{H}(\mathbf{x})$  ma una funzione residuale  $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$ . In definitiva il blocco addestrato darà in output la funzione  $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ .

Per introdurre la motivazione che spinge alla progettazione di questo particolare blocco, facciamo il seguente esperimento. Consideriamo una generica rete neurale con  $n$  layer. Costruiamo una rete neurale con  $n + k$  layer che ha come layer iniziali gli  $n$  layer della prima rete e i rimanenti  $k$  layer approssimano funzioni identità ( $\mathcal{F}(\mathbf{x}) = \mathbf{x}$ ). L'esistenza di una seconda rete più profonda della prima ma con le stesse prestazioni indica che una rete più profonda dovrebbe avere un *training error* più basso o almeno uguale a quella meno profonda. Gli esperimenti in [28] hanno mostrato tuttavia che non conosciamo nessun algoritmo di ottimizzazione che permetta almeno di eguagliare il *training error* della prima rete addestrando la

## 2.14. RESNET

---

seconda rete descritta (almeno in un tempo non lungo). Questa è una delle forme in cui si manifesta il *degradation problem* discusso in precedenza.

L'idea dei *residual blocks*, peraltro non nuova e già utilizzata in precedenza (ad es. [30]), è basata sull'ipotesi degli autori - corroborata dall'esperimento di cui sopra - che le reti neurali abbiano difficoltà ad approssimare attraverso i loro tanti layer non lineari una funzione lineare (come appunto l'identità  $\{(\mathbf{x}) = \mathbf{x}\}$ ); in altre parole, viene ipotizzato che sia più facile per un blocco di layer imparare la funzione residua rispetto alla funzione originale. Ecco perché si decide di fornire direttamente l'input in uscita con una *shortcut connection*, evitando al blocco lo sforzo di dover imparare a mappare un'identità.

Le straordinarie prestazioni raggiunte dalle reti ResNet estremamente profonde confermano che l'intuizione degli autori era corretta. Senza aver aggiunto complessità computazionale (escludendo le trascurabili somme dovute alle *shortcut connections*) resta così risolto il problema di degradazione della *training accuracy*.

### 2.14.3 Architettura di ResNet-18

ResNet-18 è una rete neurale convoluzionale profonda addestrata sul dataset ImageNet (par. 2.3.3). La rete accetta in input immagini  $224 \times 224$  ed è composta da 18 layer parametrizzati, di cui un layer convoluzionale iniziale con filtro  $7 \times 7$  (seguito da batch normalization, ReLU e max pooling), altri 16 layer convoluzionali  $3 \times 3$  raccolti a due a due in 8 *residual blocks* (descritti di seguito) e infine un layer completamente connesso (preceduto da ReLU e average pooling) dalle cui 1000 attivazioni si calcola la distribuzione di probabilità per le 1000 classi di ImageNet, per mezzo della funzione softmax. Il particolare *residual block* adoperato in ResNet-18 è del tipo mostrato in figura 2.41

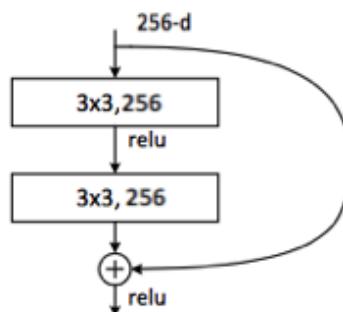


Figura 2.41: Uno degli otto *residual blocks* di ResNet-18

Il ramo che ospita i layer non lineari si compone delle seguenti operazioni:

- Convoluzione  $3 \times 3^{26}$
- Batch Normalization

---

<sup>26</sup>il numero di filtri di convoluzione, e quindi la profondità del volume di output, è riportata per ciascun *residual block* in fig. 2.44 e in fig. 2.43

- 
- ReLU
  - Convoluzione  $3 \times 3$
  - Batch Normalization

Inoltre, il ramo che realizza la *shortcut connection* si compone eventualmente di una convoluzione  $1 \times 1$  seguita da batch normalization per garantire che il volume di attivazioni che attraversa la "scorciatoia" abbia dimensioni confrontabili con il volume uscente dal ramo che ospita i layer non lineari. Questa eventualità è rappresentata in fig. 2.45 con un arco tratteggiato.

L'architettura di ResNet-18 è riportata schematicamente nella figura 2.44 e in forma tabellare, in confronto con ResNet-50, in fig. 2.43 nel prossimo sottoparagrafo.

#### 2.14.4 Architettura di ResNet-50

ResNet-50 è una variante più profonda di ResNet-18. Come la sua omologa meno profonda, questa rete accetta in input immagini  $224 \times 224$ ; essa è composta da 50 layer parametrizzati, di cui un layer convoluzionale iniziale con filtro  $7 \times 7$  (seguito da *batch normalization*, ReLU e max pooling), altri 48 layer convoluzionali di varie dimensioni raccolti a gruppi di tre in 16 *residual blocks* (descritti di seguito) e infine un layer completamente connesso con le usuali 1000 attivazioni. Il particolare *residual block* adoperato in ResNet-50 è del tipo mostrato in figura 2.42

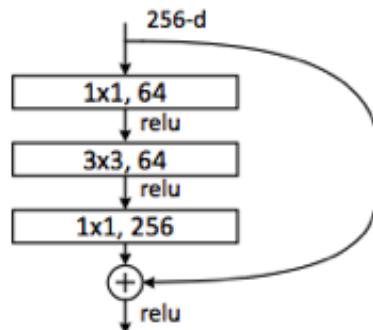


Figura 2.42: Uno dei sedici *residual blocks* di ResNet-50

Il ramo che ospita i layer non lineari si compone delle seguenti operazioni:

- Convoluzione  $1 \times 1$
- Batch Normalization
- ReLU
- Convoluzione  $3 \times 3$
- Batch Normalization

- ReLU
- Convoluzione  $1 \times 1$
- Batch Normalization

Inoltre, il ramo che realizza la *shortcut connection* si compone eventualmente di una convoluzione  $1 \times 1$  seguita da batch normalization per garantire che il volume di attivazioni che attraversa la "scorciatoia" abbia dimensioni confrontabili con il volume uscente dal ramo che ospita i layer non lineari. Questa eventualità è rappresentata in fig. 2.45 con un arco tratteggiato.

L'evidente differenza rispetto al *residual block* di ResNet-18 è motivata dalla premura di dover mantenere basso il tempo necessario ad addestrare la rete. Le convoluzioni  $3 \times 3$  sono computazionalmente costose, pertanto in reti molto profonde non si possono prevedere tutti *residual blocks* ciascuno operante due convoluzioni  $3 \times 3$ . Si decide allora di modificare il *residual block* usato: si decide di operare un'unica convoluzione  $3 \times 3$  per blocco, preceduta e seguita da una convoluzione  $1 \times 1$  (par. 2.13.1) per rispettivamente ridurre e ripristinare la profondità del volume di attivazioni su cui la convoluzione  $3 \times 3$  lavora, abbassando così il costo computazionale associato all'addestramento di ciascun *residual block*.

L'architettura di ResNet-50 è riportata schematicamente nella figura 2.45 e in forma tabellare, in confronto con quella di ResNet-18, in fig. 2.43 seguente.

layer name	output size	18-layer	50-layer
conv1	$112 \times 112$	$7 \times 7, 64$ , stride 2	
		$3 \times 3$ max pool, stride 2	
conv2_x	$56 \times 56$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$28 \times 28$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x	$14 \times 14$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5_x	$7 \times 7$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	$1 \times 1$	average pool, 1000-d fc, softmax	
FLOPs		$1.8 \times 10^9$	$3.8 \times 10^9$

Figura 2.43: Le architetture di ResNet-18 e ResNet-50 a confronto

---

### 2.14.5 Data Augmentation

Come d'uso, anche ResNet-18 e ResNet-50 adoperano una strategia di *data augmentation* per ridurre l'overfitting al training set di ImageNet. Ogni immagine è ridimensionata con il suo lato più corto estratto casualmente dall'intervallo [256, 480] e mantenendo l'*aspect ratio*. Viene ritagliata una *patch* (ritaglio)  $224 \times 224$  casuale dall'immagine così ridimensionata e ad ogni canale della patch viene sottratta la media dei valori R, G e B del dataset ImageNet (similmente ad AlexNet, par. 2.12.5). Il ritaglio è eventualmente capovolto orizzontalmente (50% di probabilità)

### 2.14.6 Addestramento di ResNet-18 e ResNet-50

Le reti ResNet-18 e ResNet-50 sono state addestrate sul database ImageNet usando la discesa stocastica del gradiente con momento = 0.9, mini-batch = 256 e decadimento dei pesi (*weight decay*) = 0.0001. Il *learning rate* iniziale è 0.1, ed è soggetto ad una strategia di *annealing* che lo riduce di un fattore 10 ogniqualvolta il *training error* si stabilizza. Il numero massimo di iterazioni di addestramento è  $6 \times 10^5$ . I pesi sono inizializzati secondo il metodo descritto in [33], creato dagli stessi autori di ResNet. Dettagli più specifici sulla fase di addestramento di ResNet-18 e ResNet-50 possono essere trovati nel paper originale [28].

## 2.14. RESNET

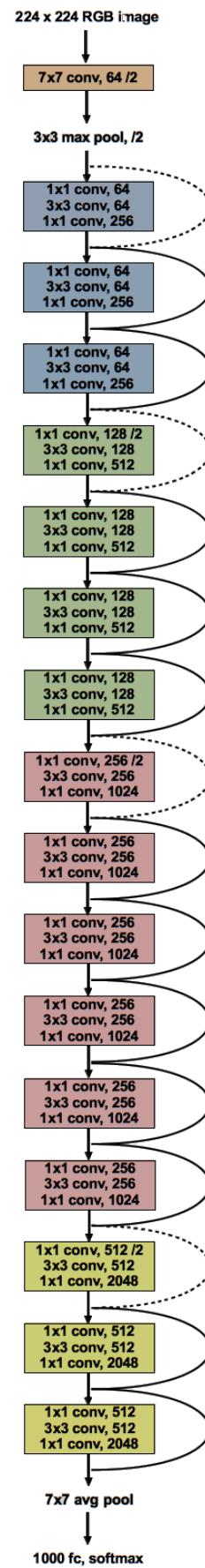
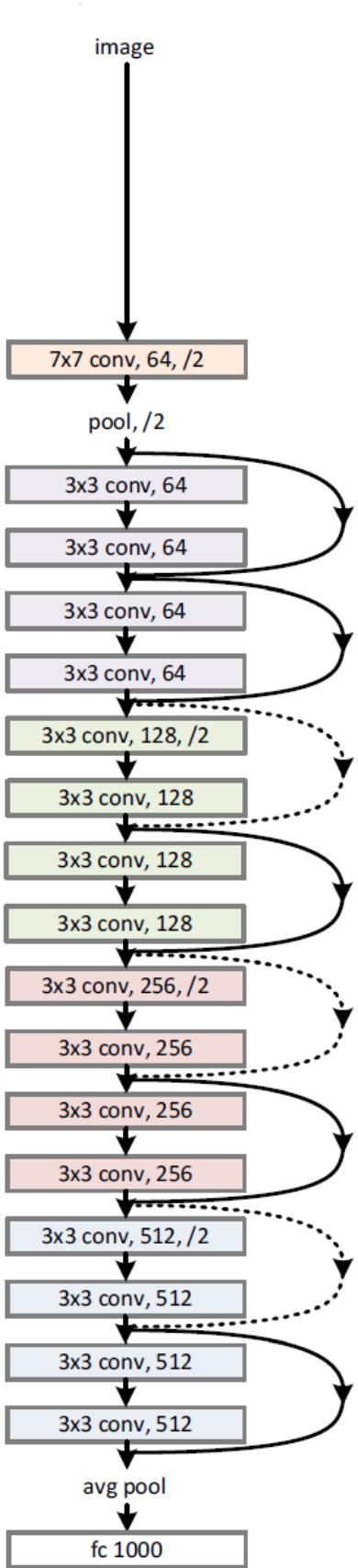


Figura 2.44: Architettura di ResNet-18

Figura 2.45: Architettura di ResNet-50



# Capitolo 3

## Esperimenti e risultati

In questo capitolo vengono presentati gli esperimenti condotti e si analizzano i risultati ottenuti.

### 3.1 Descrizione dei dataset utilizzati

Per la conduzione degli esperimenti sono stati adoperati due distinti dataset di immagini, di seguito descritti

- Il primo, usato per la fase di addestramento delle reti neurali, è una collezione di fotografie di tursiopi (scattate tra luglio 2016 e settembre 2017) e grampi (scattate tra luglio 2013 e agosto 2018) nel **Golfo di Taranto** (mar Ionio Settentrionale). Le fotografie sono state scattate e messe a disposizione dall'associazione *Jonian Dolphin Conservation* (par. 1.1). Il dataset contiene immagini acquisite in un'area di 14000 km<sup>2</sup> percorsa su un catamarano e seguendo rotte prestabilite. Il dataset acquisito contiene in totale n=10194 immagini, suddivise in cartelle in base alla specie ritratta e ulteriormente ramificate in sottocartelle in base alla data degli scatti.
- Il secondo, usato per testare le prestazioni dei classificatori binari precedentemente addestrati, consiste in un insieme di fotografie di grampi scattate nel mese di giugno 2018 nei pressi delle **Isole Azzorre** (Oceano Atlantico settentrionale) dall'associazione *Nova Atlantis Foundation* (par. 1.1). Questo dataset contiene in totale n=11290 immagini, anche questa volta suddivise in cartelle in base alla data degli scatti.

Entrambi i dataset contengono fotografie con una notevole risoluzione 6000 × 4000, con occupazione di memoria di circa 10MB per foto e occupazione totale di circa 100GB per dataset. Tuttavia, prendendo visione delle immagini in ciascuno dei due dataset ci si rende subito conto che non tutte contengono pinne dorsali di cetacei: in alcune foto sono totalmente assenti, rendendo lo scatto totalmente privo di contenuto informativo per i biologi. Anche laddove le pinne sono presenti, esse possono risultare sfocate o di bassa risoluzione se molto lontane, sovrapposte tra due esemplari vicini, disturbate da schizzi d'acqua o riflessi di luce. Infine, in tutte le fotografie sono inevitabilmente ritratti oggetti che non sono informativi ai fini dello

---

studio delle sole pinne dorsali quali barche, persone, uccelli, terraferma (paesaggi), porzioni di cielo, boe, lo specchio d'acqua ma anche parti dei cetacei diversi dalla loro pinna dorsale, quali pinne caudali e laterali, il dorso e la testa degli esemplari. Alcune di queste situazioni sono mostrate in figura 3.1.

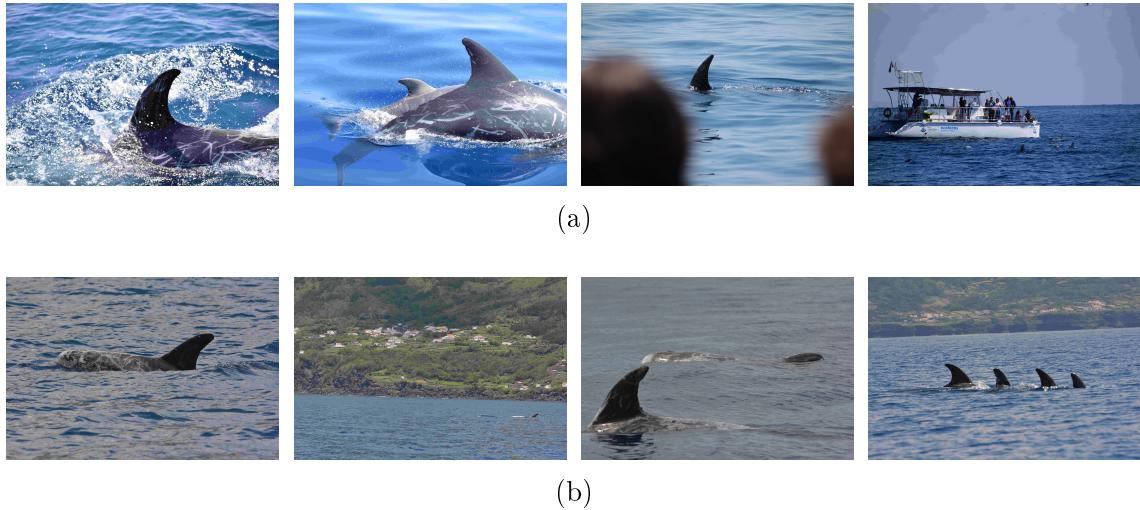


Figura 3.1: Alcune immagini estratte dai due dataset utilizzati. (a) Dataset degli scatti di Taranto, (b) Dataset degli scatti delle Azzorre. Si noti la presenza di foto con disturbi o addirittura senza alcun valore informativo ai fini dello studio dei cetacei.

Si rende perciò necessario filtrare in qualche modo le sole immagini che raffigurano al loro interno pinne dorsali di cetacei; è utile inoltre ritagliare da queste immagini filtrate le sole regioni in cui è effettivamente presente una pinna (si vuole cioè isolare l'informazione utile dal resto del dato originale). Per far questo, i due dataset sono stati rielaborati attraverso un **algoritmo di riconoscimento e cropping** delle pinne dorsali, di seguito descritto nelle sue caratteristiche salienti.

## 3.2 CropFin v1

Per ritagliare ed estrarre dalle immagini originali le sole pinne dorsali, è stata utilizzata la routine *CropFin v1* in linguaggio MATLAB sviluppata in [1] sulla base di un precedente lavoro [4]. Si può descrivere la routine in due fasi:

1. Segmentazione, filtraggio e ritaglio adattivo delle regioni delle immagini che possono verosimilmente contenere una pinna
2. Classificazione di ogni ritaglio ottenuto in due classi 'Pinna' e 'No Pinna', mediante una rete neurale artificiale creata *ad-hoc*.

La novità introdotta dal presente lavoro di tesi in merito al problema di estrazione delle pinne da un'immagine riguarda l'utilizzo di un metodo di classificazione basato sul *transfer learning*. In pratica, quindi, la principale differenza rispetto a CropFin v1 è nella seconda fase della routine: la classificazione avviene con l'utilizzo non più di una rete artificiale creata da zero per il problema in analisi, bensì

### 3.2. CROPFIN V1

---

riutilizzando un insieme di reti neurali profonde addestrate su un diverso problema di classificazione e adattate al nostro task. Questo nuovo modello è descritto dettagliatamente nel par. 3.3.

Al fine di ottenere i ritagli delle pinne, la routine adoperata attua una sequenza di operazioni di preprocessing su ciascuna immagine per poi individuare ed infine ritagliare e salvare separatamente le sole porzioni di immagini che possono eventualmente contenere pinne. Tale sequenza è implementata mediante un ciclo `for` che cicla su ogni immagine del dataset. Di seguito sono descritte sinteticamente le operazioni, nell'ordine in cui vengono applicate. Le figure esplicative per ogni fase provengono dalla tesi triennale dell'ing. Losapio [1], a cui vanno i crediti.

#### 3.2.1 Fase di ritaglio

##### Ridimensionamento

L'immagine è innanzitutto ridimensionata mediante la funzione MATLAB `imresize` con un fattore di scale  $\times 0.2$ , al fine di ottenere una nuova immagine di risoluzione più bassa ( $1200 \times 800$ ). Questa operazione di preprocessing è stata adottata per diminuire il costo computazionale delle operazioni successive.<sup>1</sup>

Il risultato dell'operazione è visualizzato in figura 3.2



(a) Prima del ridimensionamento



(b) Dopo il ridimensionamento

Figura 3.2: Ridimensionamento di un'immagine (scelta nel dataset degli scatti di Taranto)

## CLAHE

L'immagine ridimensionata è sottoposta ad una equalizzazione adattiva dell'istogramma a contrasto limitato (CLAHE). Questa operazione consente un miglioramento del contrasto dell'immagine, proprietà utile per migliorare l'efficienza della

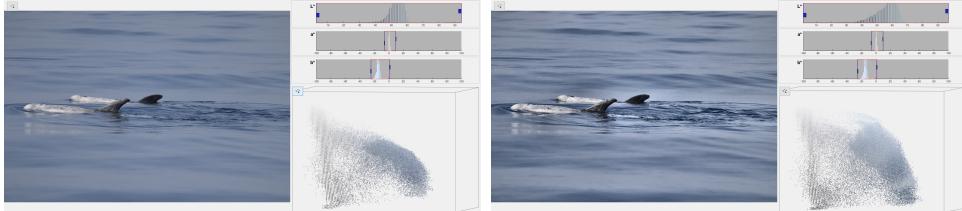
---

<sup>1</sup>La risoluzione di partenza delle immagini utilizzate è stata  $6000 \times 4000$ , ottenendo una riduzione drastica di pixel del 96%, da 24 milioni a 960 mila.

---

successiva operazione, la sogliatura dell’immagine secondo il metodo di Otsu.

Il risultato dell’operazione è visualizzato in figura 3.3



(a) Prima dell’applicazione di CLAHE    (b) Dopo l’applicazione di CLAHE

Figura 3.3: Applicazione dell’equalizzazione CLAHE all’immagine, con visualizzazione dell’istogramma nello spazio dei colori  $L^*a^*b^*$ . NB: per semplicità l’immagine è raffigurata riportandola alle sue dimensioni originali

## Segmentazione

L’immagine viene segmentata (cioè ogni pixel viene assegnato ad una di due classi: *background* e *foreground*) mediante il metodo di Otsu per la sogliatura automatica [34].

Il metodo di Otsu viene usato nella sua versione classica a due livelli, rispetto agli istogrammi dei canali L e b. In particolare, viene applicata la sogliatura secondo Otsu separatamente al canale L e b, cioè calcolate le soglie di Otsu per i due canali, mediante la funzione `multithresh`. Avendo a disposizione tali soglie, l’ipotesi avanzata è che le pinne dorsali possano essere isolate considerando le regioni di immagine che siano contemporaneamente:

- nella regione più scura del canale L, cioè a sinistra della soglia sul canale L
- nella regione contenente il grigio del canale b, cioè a destra della soglia sul canale b

L’immagine segmentata (binarizzata) finale è ottenuta quindi annerendo quei pixel dell’immagine che non verificano le seguenti condizioni (o, equivalentemente, rendendo bianchi i pixel che le verificano)<sup>2</sup>

- valore della componente L minore della soglia di Otsu sul canale L
- valore della componente b maggiore della soglia di Otsu sul canale b

Il risultato dell’operazione è visualizzato in figura 3.4

---

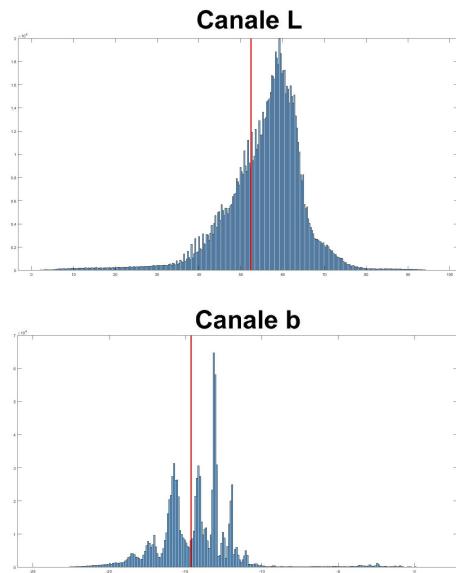
<sup>2</sup>L’idea alla base di questo approccio nasce da una precisa conoscenza del dominio e da alcune ipotesi a priori riguardanti il contenuto delle immagini. In particolare, si suppone che esse contengano generalmente solo mare (background) e cetacei (foreground), e che queste due classi di oggetti 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.

### 3.2. CROPFIN V1

---



(a) Prima della segmentazione se- (b) Dopo la segmentazione secondo  
condo Otsu Otsu



(c) Individuazione delle soglie di Otsu per i canali L e b sui relativi istogrammi

Figura 3.4: Applicazione della segmentazione secondo Otsu nello spazio dei colori  $L^*a^*b^*$

### Filtraggio delle regioni connesse

L’immagine binaria ottenuta in seguito alla segmentazione viene filtrata in modo che siano scartate quelle regioni binarie connesse (anche dette *blob*) che non presentano caratteristiche tali da poter rappresentare, verosimilmente, una pinna dorsale. In particolare vengono utilizzati, consecutivamente due filtri:

1. il primo è applicato all’intera immagine binarizzata e serve a migliorare il risultato della sogliatura secondo Otsu. Il filtro è configurato per mantenere, nell’ordine, le regioni connesse con le seguenti proprietà:

- prime 15 in ordine decrescente di **Area** (n. di pixel che compongono la regione connessa)
- **Area** nel range [1600, 40000]

- 
- **Extent** nel range [-Inf, 0.55] (rapporto tra **Area** e il n. di pixel del più piccolo rettangolo che racchiude l'intera regione connessa, con i lati paralleli a due a due paralleli ai bordi dell'immagine)

2. il secondo è applicato come segue

- (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, al fine di non perdere eventuali parti della pinna erroneamente anneriti dopo la binarizzazione, ogni dimensione è aumentata del 20%.
- (b) Si applica nuovamente, a ciascun ritaglio ottenuto, la sogliatura basata sul metodo di Otsu. 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** nel range [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, descritto nella sezione successiva.

Il risultato dell'operazione è visualizzato in figura 3.5

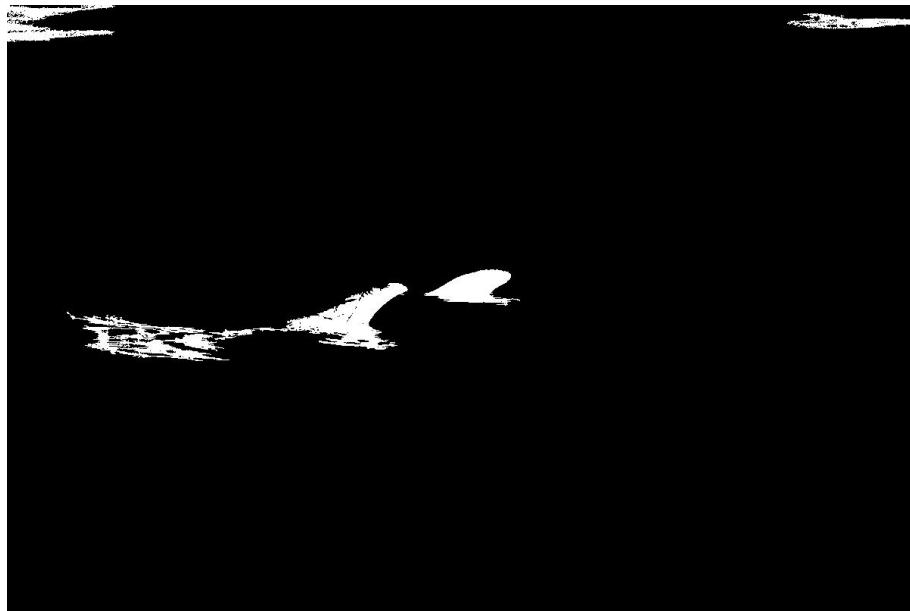
## Ritaglio adattivo

Le regioni binarie mantenute in seguito alla fase di filtraggio 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. Evitando di scendere nei dettagli implementativi e numerici (riportati nel par. 5.1 in [1]), si descrivono nell'ordine le operazioni effettuate sulle singole regioni binarie dall'algoritmo di ritaglio:

1. Si sottopone la regione binaria al riempimento dei cosiddetti *holes*, cioè "buchi" anneriti racchiusi in una regione connessa, mediante la funzione `imfill` con opzione '`'holes'`'
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 della regione connessa all'altezza del punto medio
3. Si identifica il più piccolo rettangolo che racchiude 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.

### 3.2. CROPFIN V1

---



(a) Prima dell'individuazione delle regioni connesse



(b) Estrazione delle regioni connesse dall'immagine binarizzata  
(c) Applicazione ripetuta della sogliatura secondo Otsu (senza CLAHE)  
(d) Applicazione del filtro sulle regioni connesse estratte

Figura 3.5: Fase di filtraggio delle regioni connesse

L'output di questa prima fase della routine sono i ritagli di quelle regioni dell'immagine originale che, verosimilmente, ritraggono una pinna dorsale. Questa ipotesi sul contenuto dei ritagli è sostenuta solamente sulla base del processo di segmentazione e filtraggio appena descritto.

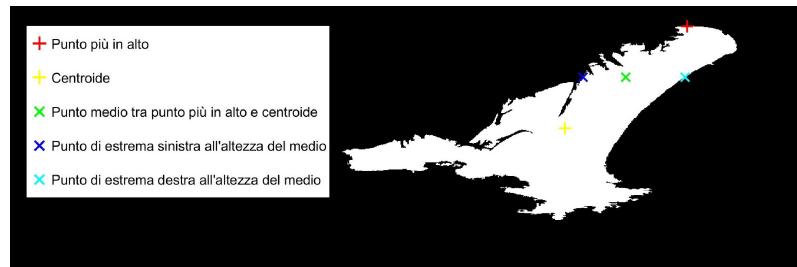
Il risultato dell'operazione è visualizzato in figura 3.6



(a) Regione binaria da sottoporre a ritaglio adattivo



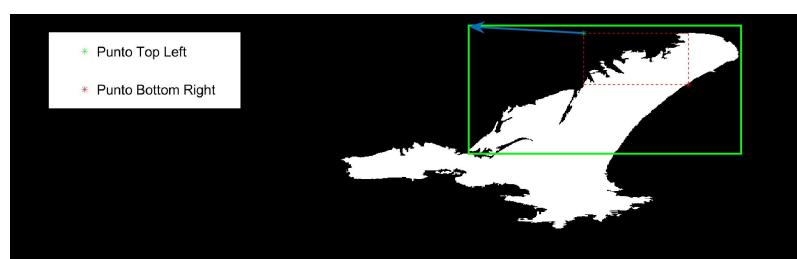
(b) Riempimento degli *holes*



(c) Individuazione dei quattro punti di interesse



(d) Individuazione del rettangolo che racchiude i quattro punti di interesse



(e) Traslazione ed estensione del rettangolo trovato



(f) Ritaglio di partenza



(g) Risultato finale

Figura 3.6: Fase di ritaglio adattivo

### 3.2. CROPFIN V1

---

La routine CropFin v1, nella sua prima fase di ritaglio adattivo, è stata applicata ai dataset degli scatti di Taranto e delle Azzorre. In tabella 3.1 è riportato il numero di ritagli (*crops*) prodotti da CropFin v1 con input i dataset sopracitati.

Dataset	N. foto	N. crop	di cui 'Pinna'	di cui 'No Pinna'
Taranto	10194	15228	4033	11195
Azzorre	11290	20395	3793	16602

Tabella 3.1: Output della prima fase di CropFin v1

#### 3.2.2 Fase di classificazione

È evidente da una rapida ispezione dell'output che la quantità di regioni estratte che però non contengono pinne risulta, su larga scala, superiore a quella che contiene effettivamente pinne. Numericamente questo fatto è evidenziato in tab. 3.1, compilata dopo una fase di etichettatura a mano dei ritagli prodotti, nelle classi 'Pinna' e 'No Pinna' (motivata e spiegata nel seguito del sottoparagrafo). I ritagli della classe 'No Pinna' ottenuti sono l'esito "fallimentare" della procedura di segmentazione, filtraggio e ritaglio adattivo adottati in CropFin v1.

Questa osservazione è ciò che primariamente motiva l'introduzione di una fase di classificazione finale in CropFin v1, che consenta di automatizzare completamente la procedura di object detection.

La fase di classificazione prevede l'impiego di un classificatore binario che sappia discriminare un ritaglio in 'Pinna' o 'No Pinna'. L'addestramento di un tale classificatore (a prescindere dalle sue caratteristiche) può essere fatto mediante tecniche di supervised learning (par. 2.4.1), avendo a disposizione molti esempi "etichettati" di entrambe le classi da predire.

#### Creazione del training set

Per consentire l'addestramento del classificatore si rende quindi necessario un lavoro di etichettatura manuale dei ritagli, attribuendo a ciascuno la classe 'Pinna' e 'No Pinna'. Questa operazione è stata svolta per entrambi i dataset a nostra disposizione; i risultati di questa etichettatura manuale sono presenti nella tab. 3.1.

Si precisa che, nella fase di etichettatura manuale, sono stati attribuiti alla classe 'Pinna' tutti e soli i ritagli contenenti una sola pinna in primo piano, intera o leggermente tagliata, escludendo invece 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 di una successiva foto-identificazione automatica delle pinne (ad esempio con la routine "*SPIR*" sviluppata e descritta in [2] e migliorata in [3]). 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. In fig. 3.7 sono riportati alcuni esempi di etichettatura dei ritagli.

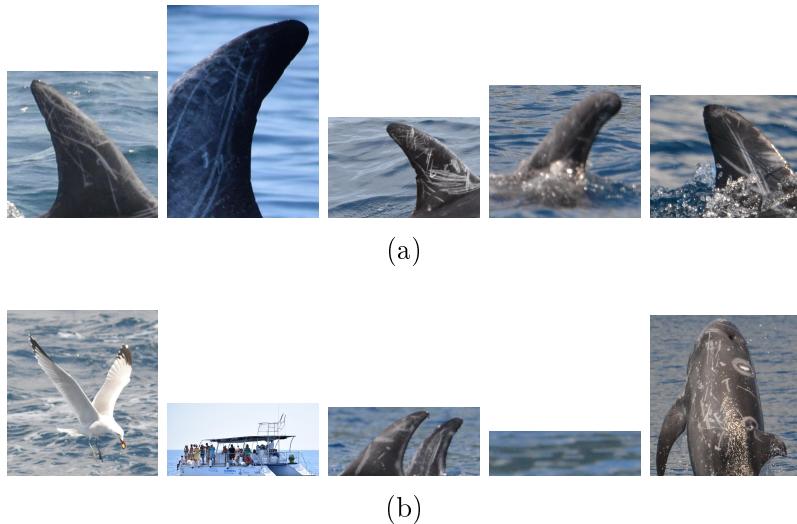


Figura 3.7: Alcuni ritagli etichettati manualmente come appartenenti alla classe (a) 'Pinna', (b) 'No Pinna'

I due dataset così etichettati si prestano bene ad essere usati per addestrare un classificatore in grado di risolvere il task in esame.

### Classificazione mediante rete *ad-hoc*

Tra i vari modelli di classificazione adottabili, in CropFin v1 si decide di progettare una rete neurale convoluzionale creata *ad-hoc*, costruita cioè appositamente per risolvere il task in questione, da zero (*from scratch*). L'addestramento del classificatore è avvenuto con i 15228 ritagli del dataset di Taranto. Per i dettagli sull'architettura, l'addestramento e le prestazioni di questo classificatore *ad-hoc* si rimanda al par. 4.4.2 di [1].

In questo lavoro di tesi si è tentato un approccio diverso alla classificazione: si è provato a risolvere il task mediante il riutilizzo di CNN pre-addestrate, con la tecnica del *Transfer Learning*.

### 3.3 Classificazione mediante CNN e Transfer Learning

Nel par. 2.10 sono state esposte molteplici motivazioni per le quali per risolvere un problema di classificazione (in particolare di *image classification*) può essere meglio usare la tecnica del *transfer learning*, adattando al task in esame una rete neurale pre-addestrata piuttosto che creare una rete da zero. Il nucleo principale di questo lavoro di tesi è quindi dedicato alla creazione di un nuovo modello di classificazione, basato su *transfer learning*, che possa migliorare la fase di classificazione di CropFin v1. Questi "miglioramenti" sono da valutare con rigore ingegneristico sulla base di alcuni parametri, che consentono un confronto di prestazioni con il

### 3.3. CLASSIFICAZIONE MEDIANTE CNN E TRANSFER LEARNING

---

classificatore di CropFin v1; l’analisi delle prestazioni e quindi il confronto è svolto nel par. 3.3.3.

#### 3.3.1 Scelta delle CNN

Sono state riutilizzate ed adattate mediante la tecnica del *transfer learning* quattro reti neurali convoluzionali (*CNN*) sviluppate nell’ambito della *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* ed addestrate sul dataset ImageNet (par. 2.3.3). La loro scelta è stata motivata dal loro successo e prestigio nell’ambito del problema di *object classification* proposto nella *ILSVRC*. La scelta di queste reti è stata altresì motivata da un interesse puramente didattico, cioè studiare il funzionamento di CNN diverse per scelte architetturali e prestazioni, a prescindere dalla risoluzione del problema in esame. Esse sono di seguito elencate

- **AlexNet** (par. 2.12)

La sua vittoria nella *ILSVRC 2012* con un tasso di errore *top-5* del 16.4% ha di fatto dimostrato alla comunità scientifica la straordinaria efficienza delle reti neurali convoluzionali nell’ambito dei problemi di *computer vision*.

- **GoogLeNet** (par. 2.13)

Grazie all’introduzione del modulo *Inception*, GoogLeNet è una rete profonda ma incredibilmente leggera e semplice da addestrare, se paragonata alle precedenti reti fino ad allora esistenti (tra tutte, AlexNet).

- **ResNet-18** (par. 2.14)

ResNet rappresenta lo stato dell’arte nell’ambito delle reti neurali convoluzionali; l’introduzione dei *residual blocks* ha permesso di avere reti con un grandissimo numero di layer, attenuando di molto i problemi legati all’estrema profondità dell’architettura.

- **ResNet-50** (par. 2.14)

Una variante di ResNet, più profonda e con migliori prestazioni sul dataset *ImageNet*.

#### 3.3.2 Addestramento

L’ambiente di sviluppo nel quale sono state addestrate le quattro reti è il software Matlab R2019a, esteso con il pacchetto *Deep Learning Toolbox*. Tutto il codice sorgente Matlab creato è riportato nel cap. 5.

Il dataset utilizzato per l’addestramento è quello con gli  $n = 15228$  ritagli che CropFin ha prodotto dagli scatti di Taranto (vd. tab. 3.1) e in seguito opportunamente etichettati a mano (come descritto nel par. 3.2.2). Questo dataset è di fatto lo stesso usato per l’addestramento del classificatore *ad-hoc* in [1].

Il dataset dei ritagli è rappresentato in Matlab da un oggetto *imageDatastore*, ed è stato suddiviso ( *splitted*) in maniera casuale in due ulteriori oggetti *imageDatastore*:

- 
- `cropTrain`, un *training set* contenente il 70% dei ritagli (10659 ritagli), cioè il 70% dei ritagli 'No Pinna' (7836) più il 70% dei ritagli 'Pinna' (2823)
  - `cropValidation`, un *validation set* contenente il rimanente 30% dei ritagli (4569 ritagli di cui 3359 'No Pinna' e 1210 'Pinna')

Il training set è sottoposto ad operazioni di *image augmentation* durante la fase di addestramento, per migliorare la capacità di generalizzazione del classificatore da addestrare ed evitare quindi l'*overfitting*. A ciascuna immagine del set sono applicate le seguenti trasformazioni, rappresentate in un opportuno oggetto `imageDataAugmenter`:

- una riflessione rispetto all'asse x (operata con il 50% di probabilità);
- 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.

Un oggetto `augmentedImageDatastore` parametrizzato con l'`imageDataAugmenter` sopra descritto e l'`inputSize` della rete è creato per rappresentare la versione "aumentata" di `cropTrain`. Le operazioni applicate in maniera casuale contribuiscono ad ottenere ulteriore variabilità del dataset impiegato per la fase di addestramento; i ritagli vengono inoltre ridimensionati all'input size della rete da addestrare. 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).



Figura 3.8: Un campione di ritagli sottoposti alle operazioni di *augmentation*

### 3.3. CLASSIFICAZIONE MEDIANTE CNN E TRANSFER LEARNING

---

Un secondo oggetto `augmentedImageDatastore` è usato per rappresentare la versione "aumentata" di `cropValidation`, ma in questo caso è parametrizzato solo con `inputSize` (viene cioè effettuato solamente un ridimensionamento dei ritagli all'input size della rete da addestrare, senza operazioni di augmentation).

Si noti che i due dataset "aumentati" hanno le stesse dimensioni dei loro omologhi dataset "non aumentati", nel senso che ogni ritaglio del dataset originale è presente esattamente una volta nel relativo dataset aumentato: l'unica differenza tra il ritaglio originale e quello nel dataset aumentato è appunto l'applicazione (a *runtime*) delle operazioni di ridimensionamento e – nel caso del *training set* – augmentation a ciascun ritaglio. Si tratta quindi di una semplice "perturbazione" dei due dataset, senza eliminazione di suoi elementi o aggiunta di nuovi.

L'inizializzazione delle quattro CNN all'interno dell'ambiente Matlab è particolarmente semplice, e può essere effettuato richiamando la rete col suo nome (`alexnet`, `googlenet`, `resnet18`, `resnet50`) assegnandola ad una variabile, avendo cura di scaricare il relativo *add-on* dall'*add-on explorer* integrato di Matlab, estendendo il *Deep Learning Toolbox*.

Le quattro reti, come visto nei rispettivi paragrafi, sono in origine state addestrate sul database di immagini *ImageNet* (par. 2.3.3) per classificare un'immagine scegliendo tra le mille categorie diverse del dataset. Applicare la tecnica del *transfer learning* (par. 2.10) per adattare ciascuna rete al problema di classificazione binaria delle pinne significa effettuare le seguenti operazioni

- analizzare l'architettura della rete e decidere quanti e quali layer (tra quelli dotati di parametri addestrabili) "congelare", cioè impedirne l'ulteriore *fine-tuning* dei parametri in fase di ri-addestramento azzerando il learning rate dei layer
- sostituire nella rete originale da "trasferire" l'ultimo *fully-connected layer* avente 1000 neuroni, ognuno associato alle 1000 classi del database ImageNet e il cui valore di attivazione permette di calcolare la probabilità per la relativa classe (attraverso la funzione *softmax*), con un nuovo *fully-connected layer* dotato di 2 soli neuroni, relativi alle classi 'Pinna' e 'No Pinna'; questo nuovo layer è inizializzato con un learning rate abbastanza alto (fissato a 10) per permettere un addestramento più veloce
- adattare la nuova rete così creata al problema di classificazione binaria delle pinne grazie al ri-addestramento della rete sul nuovo dataset dei ritagli, descritto in precedenza

Si noti che per la scelta di quanti e quali layer parametrizzati "congelare" non esistono regole o criteri generali; questa scelta è un nuovo set di iperparametri della rete, da trovare risolvendo uno specifico problema di ottimizzazione degli iperparametri. Poiché per ottimizzare questi iperparametri sarebbe necessario addestrare un gran numero di volte ciascuna delle quattro reti, con un notevole impiego di tempo, tale problema di ottimizzazione non è stato risolto in questa sede e si è preferito scegliere manualmente i layer da congelare.

---

Per garantire un buon *trade-off* tra capacità di generalizzazione sul nuovo dataset e velocità di addestramento della rete, si è scelto di congelare uno, due o tre layer convoluzionali iniziali di ciascuna rete, nell'ordine in cui sono presenti nell'architettura della rete. Tale scelta è motivata anche dal fatto che il dataset *ImageNet* e quello dei ritagli delle pinne hanno alcune caratteristiche in comune: nel primo ci sono almeno una decina di classi diverse di cetacei, come ad esempio 'gray whale' (balena grigia) e 'killer whale' (orca). Mutuando un'idea discussa in [18], si è quindi avanzata la plausibile ipotesi che tutte le reti abbiano imparato, nei livelli più bassi, ad estrarre *features* abbastanza generali utili per il riconoscimento di una variegata classe di animali marini e delle loro parti del corpo, e pertanto questi primi layer possono essere riutilizzati senza modifiche. Per ulteriori dettagli sull'implementazione in Matlab riferirsi al cap. 5.

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 6 e *learning rate* globale pari a 0.0003. Ciò ha portato alla definizione del seguente oggetto **trainingOptions**:

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

options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData', cropAugmentedValidation, ...
    'ValidationFrequency',valFrequency, ...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'CheckpointPath','.\Checkpoint [nome della rete]');
```

Il numero di epoche di addestramento, relativamente basso, è accettabile in quanto utilizzando il transfer learning per ri-addestrare una rete pre-addestrata tipicamente sono necessarie poche iterazioni sull'intero dataset per ottenere un buon adattamento della rete al nuovo task di classificazione<sup>3</sup>; inoltre un numero basso di epoche, assieme alle operazioni di augmentation, garantisce una discreta robustezza contro l'*overfitting*, problema in cui si può facilmente incappare in presenza di un dataset di dimensioni relativamente ridotte come in questo caso.

Per quanto riguarda gli altri iperparametri (*learning rate* globale e dimensione del *minibatch*), i loro valori sono stati scelti empiricamente, sulla base di alcune *best practices* generali presenti in letteratura<sup>3</sup>, che funzionano bene con un'ampia classe di reti neurali convoluzionali e di dataset. A rigore, i valori migliori per questi iperparametri possono essere trovati con la risoluzione di un preciso problema

---

<sup>3</sup>come evidenziato in <https://it.mathworks.com/help/deeplearning/examples/transfer-learning-using-alexnet.html>

### 3.3. CLASSIFICAZIONE MEDIANTE CNN E TRANSFER LEARNING

---

di ottimizzazione (che richiederebbe numerosi ri-addestramenti per ogni rete, con notevole dispendio di tempo). Tuttavia, come sarà evidente al momento della presentazione dei risultati, le prestazioni registrate dalle reti con questi iperparametri sono eccellenti, e non è necessario cercarne di migliori (un tale sforzo computazionale e di tempo non giustificherebbe un leggerissimo aumento dell'accuratezza).

L'addestramento di ciascuna rete è stato lanciato mediante la funzione

```
TL_net = trainNetwork(cropAugmentedTrain, layers, options)
```

specificando il *training set* sottoposto ad *augmentation*, l'architettura della rete (array di tipo **Layer** nel caso di AlexNet e di tipo **LayerGraph** per le restanti reti) e le opzioni per l'addestramento.

Nella tabella 3.2 sono riassunte le principali informazioni e i risultati globali (accuratezza sul *validation set* e dati della matrice di confusione) di ciascuna rete così addestrata. Nelle figure dalla 3.9 alla 3.12 sono riportati i grafici che mostrano l'andamento temporale dei quattro addestramenti. L'addestramento è avvenuto nella modalità a singolo processore grafico su due diverse macchine:

- l'addestramento di AlexNet, GoogLeNet e ResNet-18 è avvenuto su una macchina equipaggiata con CPU Intel Core i5-4670 @ 3.40 GHz con 8 GB di RAM e Intel HD Graphics 4600 (scheda video integrata)
- l'addestramento di ResNet-50, decisamente più costoso a livello computazionale per via dei suoi numerosi pesi, è avvenuto su una workstation HP Z840 equipaggiata con due CPU Intel Xeon E5-2699 v3 @ 2.30 GHz, 256 GB di RAM e scheda video NVIDIA Quadro K5200 con 8 GB dedicati.

Rete	Tempo addestramento	Accuracy	Sensitivity	Specificity	TP	FP	TN	FN
AlexNet	67m54s	99.87%	99.75%	99.91%	1207	3	3356	3
GoogLeNet	149m59s	99.89%	99.67%	99.97%	1206	1	3358	4
ResNet-18	168m17s	99.89%	99.75%	99.94%	1207	2	3357	3
ResNet-50	497m33s	90.30%	98.93%	87.20%	1197	430	2929	13

Tabella 3.2: Risultati del ri-addestramento delle quattro reti secondo la tecnica del *transfer learning*

Con la sola esclusione di ResNet-50, i valori di accuratezza ottenuti, calcolati sul *validation set*, sono molto elevati. Il successo ottenuto da questi classificatori nel risolvere il problema di classificazione delle pinne è dovuto probabilmente alla facilità con cui la rete riesce ad adattarsi al nuovo dominio di applicazione (d'altronde, è una classificazione binaria abbastanza semplice, a maggior ragione per reti molto profonde in grado di risolvere problemi di classificazione ben più complessi). Diverso è il caso di ResNet-50: per quanto comunque alta, l'accuratezza raggiunta del 90.30% non è paragonabile a quella delle rimanenti tre reti. Si avanza quindi l'ipotesi di *overfitting* della rete al *training set*. Questa ipotesi è corroborata dal fatto che l'accuratezza valutata sul *training set* è invece prossima al 100%, come si

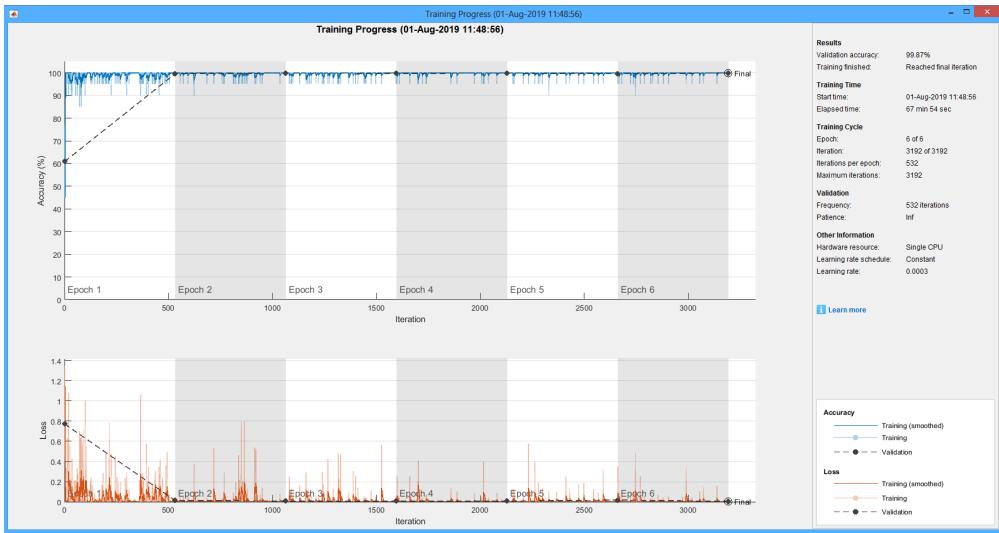


Figura 3.9: Grafico del ri-addestramento di AlexNet



Figura 3.10: Grafico del ri-addestramento di GoogLeNet

nota in fig. 3.12. Questa mancata capacità di generalizzazione può essere conseguenza dell'eccessiva complessità della rete (n. di parametri inutilmente elevato), in rapporto ad un *training set* relativamente piccolo<sup>4</sup>.

Le performance delle quattro reti sono state, inoltre, misurate sul dataset dei ritagli ottenuti con CropFin v1 dal dataset degli scatti nelle Azzorre, di dimensione  $n = 20395$  (usato quindi come *test set*). Si ricorda che questi ritagli erano stati in precedenza etichettati manualmente, come descritto nel par. 3.2.2. I risultati sono riassunti in tabella 3.3.

Come ci si aspettava, le reti hanno prestazioni abbastanza elevate, ad eccezione di ResNet-50. L'inefficienza di quest'ultima rete è aggravata dal fatto che la sua specificity sia risultata bassa: confondere un ritaglio 'No Pinna' con un ritaglio

<sup>4</sup>Delle considerazioni simili sono state fatte anche nel paper originale di ResNet [28].

### 3.3. CLASSIFICAZIONE MEDIANTE CNN E TRANSFER LEARNING

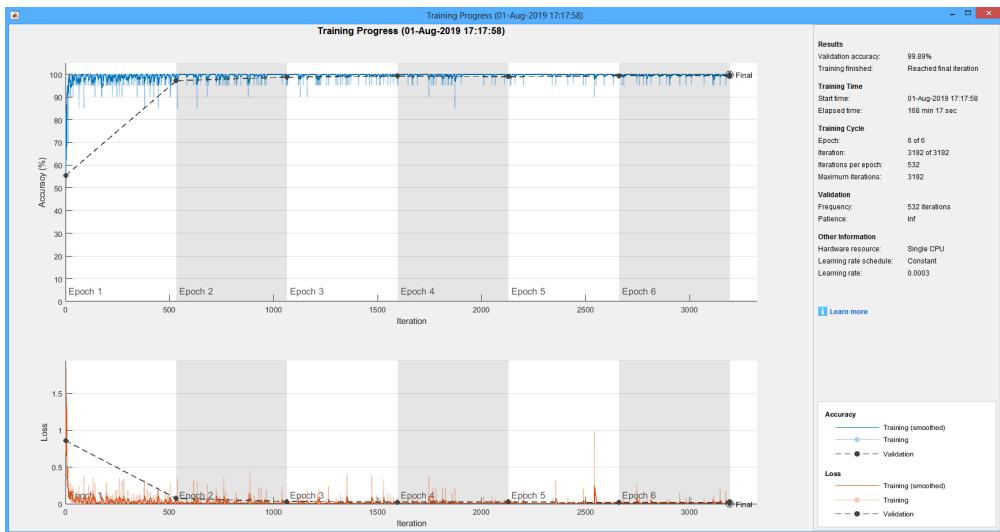


Figura 3.11: Grafico del ri-addestramento di ResNet-18

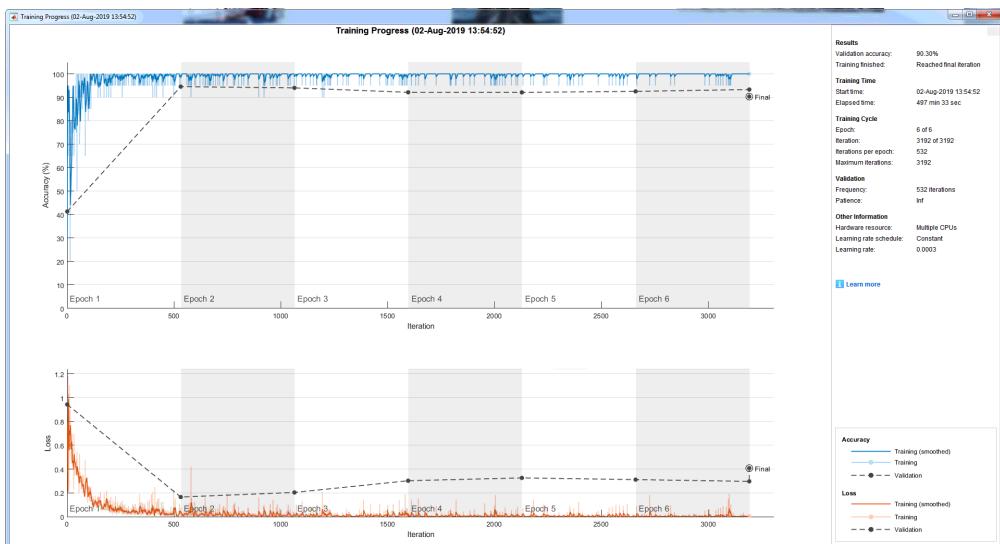


Figura 3.12: Grafico del ri-addestramento di ResNet-50. Si noti che la *training accuracy* è prossima al 100% mentre la *validation accuracy* si attesta al 90%, chiaro sintomo di *overfitting* al training set

'Pinna' è, dal punto di vista dell'utente (tipicamente un ricercatore), più grave del contrario. Infatti, è tollerabile perdere qualche pinna nei falsi negativi (in una spedizione uno stesso esemplare è spesso immortalato in più fotografie, quindi c'è un'alta probabilità che le sue pinne siano presenti in più ritagli di cui almeno uno correttamente classificato come 'Pinna'), ma è meno tollerabile avere molti falsi positivi (infatti gli algoritmi di foto-identificazione delle pinne come quello proposto in [3] danno sempre in output un *match* plausibile, anche se l'input non raffigura una pinna).

Per questo motivo, d'ora in avanti ResNet-50 non verrà più utilizzata.

Rete	Accuracy	Sensitivity	Specificity	TP	FP	TN	FN
AlexNet	97.1%	97.9%	96.9%	3712	511	16091	81
GoogLeNet	97.2%	98.0%	97.0%	3718	493	16109	75
ResNet-18	96.7%	99.0%	96.3%	3754	619	15983	39
ResNet-50	77.9%	98.1%	73.3%	3722	4431	12171	71

Tabella 3.3: Prestazioni valutate sul dataset dei ritagli delle Azzorre, usato come *test set*

### 3.3.3 Classificatore ensemble

Al fine di migliorare ulteriormente l'accuratezza della classificazione, si è sperimentato un metodo di apprendimento ensemble (*ensemble learning*, par. 2.11). L'idea chiave è quella di creare un insieme (*ensemble*) di classificatori, ciascuno dei quali è chiamato a "votare" circa l'esito della predizione; a seconda dello "schema di consenso" (*consensus scheme*) scelto per l'ensemble, cioè a seconda di quanto peso assume ciascun voto nella classificazione finale, l'output complessivo sarà la classe che avrà ricevuto "democraticamente" il maggior consenso.

Come già descritto nel par. 2.11, l'efficacia dei metodi ensemble è dovuta al fatto che, solitamente, differenti modelli addestrati per risolvere uno stesso problema di classificazione non faranno tutti gli stessi errori sul *test set* (gli errori si possono cioè considerare, in buona sostanza, incorrelati). Se lo schema di consenso applicato è il *major voting*, si dimostra che l'ensemble di classificatori ha sempre prestazioni migliori o almeno uguali a quelle di ciascuna rete dell'ensemble presa singolarmente; il miglioramento delle prestazioni è tanto migliore quanto più gli errori commessi dalle reti dell'ensemble sono tra loro incorrelati.[11][19]

L'ensemble utilizzato è costituito dalle reti AlexNet, GoogLeNet e ResNet-18, addestrate sul dataset dei ritagli di Taranto. Uno schema dell'ensemble è raffigurato in fig. 3.13

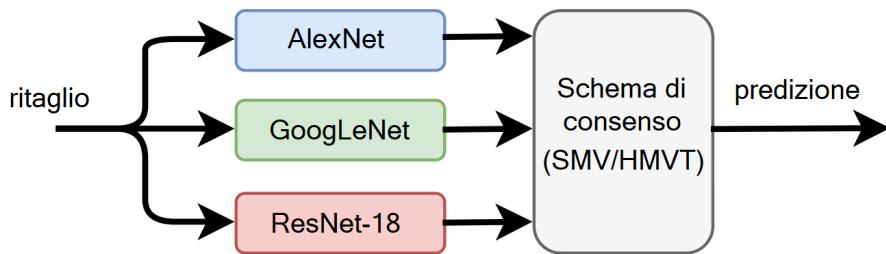


Figura 3.13: Schema del classificatore ensemble creato

Sono stati effettuati due esperimenti che adottano due diversi schemi di consenso, di seguito elencati:

- **soft major voting**: un ritaglio è classificato come 'Pinna' se la media delle probabilità attribuite alla classe 'Pinna' da ciascuna rete è maggiore del 50%.

### 3.3. CLASSIFICAZIONE MEDIANTE CNN E TRANSFER LEARNING

---

La nuova probabilità attribuita alla classe 'Pinna' è appunto la suddetta media. Vale allo stesso modo il viceversa relativamente alla classe 'No Pinna'.

- **hard major voting con soglia:** un ritaglio è classificato come 'Pinna' se almeno due delle tre reti lo classificano come 'Pinna' e se inoltre la media delle probabilità attribuite alla classe 'Pinna' da ciascuna rete è maggiore di una certa soglia, fissata arbitrariamente al 97%. La nuova probabilità attribuita alla classe 'Pinna' è appunto la suddetta media. Vale allo stesso modo il viceversa relativamente alla classe 'No Pinna'.

Gli esperimenti consistono nella classificazione del *test set* contenente i ritagli delle Azzorre da parte del classificatore ensemble. Questi esperimenti hanno come scopo la misurazione delle prestazioni del classificatore ensemble, e consentono il confronto di prestazioni tra questo nuovo modello e quello di CropFin v1, descritto in [1] (e provato sullo stesso *test set*). I risultati dei due esperimenti sono riportati in tabella 3.4, assieme a quello relativo al classificatore nativo in CropFin v1. Gli stessi dati riportati nella tabella 3.4 sono mostrati sotto forma di matrici di confusione in fig. 3.14. In figura 3.15 si riportano infine alcuni campioni di ritagli classificati dall'ensemble.

Modello	Accuracy	Sensitivity	Specificity	TP	FP	TN	FN
Ensemble (SMV)	97.2%	98.7%	96.9%	3745	518	16084	48
Ensemble (HMVT)	97.2%	93.4%	98.1%	3543	313	16289	250
CropFin v1	92%	85%	95%	—	—	—	—

Tabella 3.4: Prestazioni di differenti modelli di classificazione, valutate sul dataset dei ritagli delle Azzorre. Abbreviazioni: SMV = *soft major voting*, HMVT = *hard major voting* con soglia (*threshold*) sulla probabilità media

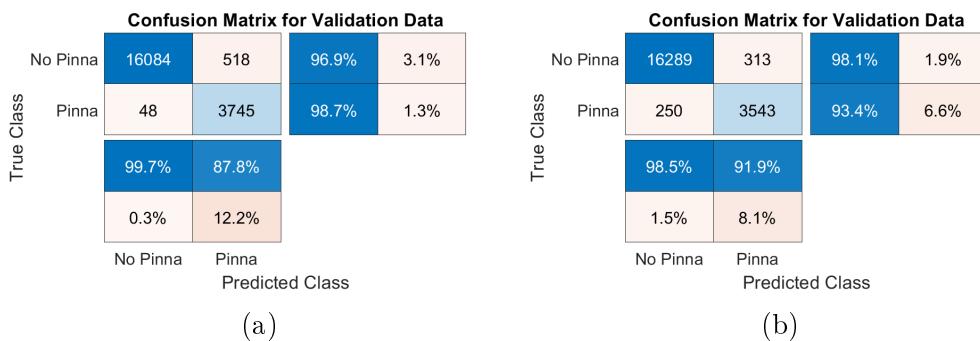


Figura 3.14: Matrici di confusione relative alle predizioni del nuovo ensemble sui ritagli delle Azzorre, con schema di consenso (a) *soft major voting*, (b) *hard major voting* con soglia.

Confrontando le prestazioni ottenute dal classificatore di CropFin v1 e i due classificatori ensemble creati sfruttando il principio del *transfer learning* è evidente la maggiore efficienza di questi ultimi, in tutti i parametri messi a confronto.

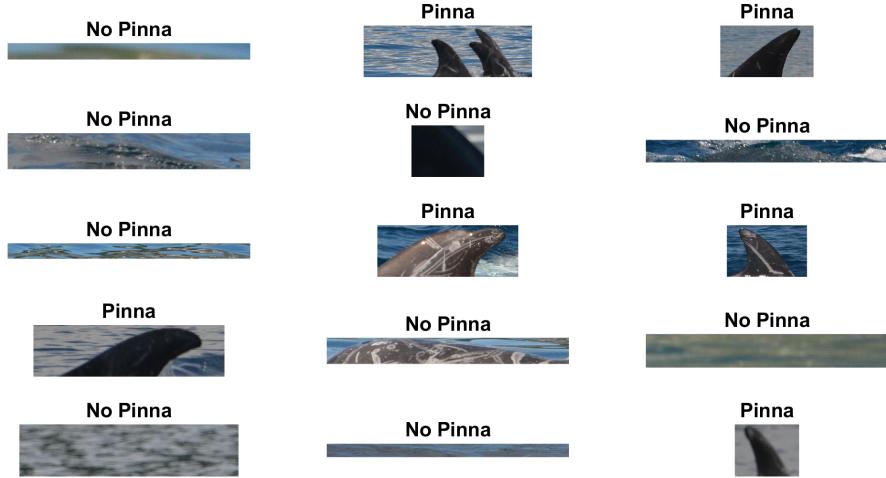


Figura 3.15: Alcune predizioni proposte dall’ensemble con schema di consenso HM-VT. Si noti ad esempio quella centrale in alto: compaiono tre pinne di cui addirittura due sovrapposte, ma il classificatore restituisce ‘Pinna’. Sebbene il classificatore sia riuscito a confermare la presenza di una pinnula, l’etichetta corretta era ‘No Pinna’ (per i criteri di etichettatura adottati sui ritagli con pinne multiple). Evidentemente nel training set con i ritagli di Taranto non erano presenti molti ritagli con pinne multiple, quindi il classificatore ha avuto difficoltà ad attribuire la classe ‘No Pinna’ a questo tipo di ritagli.

Si fa notare infine che lo schema di consenso di tipo HMVT ha fatto registrare un miglioramento del parametro di *specificity* che, come spiegato in precedenza, è un parametro fondamentale in quanto dà una misura di quanto i ritagli classificati come ‘Pinna’ sono “sporcati” da ritagli ‘No Pinna’ erroneamente classificati come ‘Pinna’, un errore che l’utente finale vorrebbe evitare. Si può verificare che alzando la soglia sopra il 97% la *specificity* aumenta ancora, ma a scapito della *sensitivity*, che decresce con una velocità purtroppo maggiore della *specificity*. Tuttavia questo non costituisce un grosso problema nel caso in cui, come spesso avviene, ci sia una discreta ridondanza nella presenza di uno stesso esemplare in più foto di una stessa spedizione in mare, come descritto in precedenza<sup>5</sup>.

In conclusione, quindi, si può affermare che adattare al nostro task di classificazione di ritagli di pinne un’opportuna rete neurale convoluzionale pre-addestrata con il metodo del *transfer learning* piuttosto che costruirne una *from scratch* è risultato vantaggioso in termini di prestazioni offerte ed è quindi servito a migliorare la fase di classificazione di CropFin v1. Inoltre, si è verificato che le prestazioni aumentano ulteriormente costruendo un classificatore di tipo *ensemble* che raccolga diverse reti ri-addestrate che lavorino in sinergia attraverso uno schema di consenso di *major voting*. Il miglioramento della fase di classificazione, e soprattutto del parametro della *specificity*, è fondamentale in vista di una successiva applicazione di un algoritmo

<sup>5</sup>In questo caso, infatti, è meno probabile che in un dataset di ritagli da scatti avvenuti in una specifica spedizione vengano scartati *tutti* i ritagli relativi alla pinnula di un certo esemplare

### 3.3. CLASSIFICAZIONE MEDIANTE CNN E TRANSFER LEARNING

di foto-identificazione delle pinne, ad esempio quello descritto in [3]. Per i motivi descritti, questo nuovo classificatore è preferibile a quello di CropFin v1.

Il tempo di addestramento è risultato molto più alto di quello del classificatore di CropFin v1, pur avendo disposto di capacità di calcolo di gran lunga superiore. Inoltre, l'occupazione di memoria del nuovo ensemble ( $\sim 263$  MB) è più di 8 volte superiore a quella del classificatore *ad-hoc* di CropFin v1 ( $\sim 31,2$  MB). Il tempo di addestramento e l'occupazione di memoria elevati sono comunque "ripagati" dalle alte prestazioni dell'ensemble di reti. Tuttavia non c'è un criterio oggettivo per valutare quantitativamente l'utilità di questo miglioramento di prestazioni in rapporto all'allungamento del tempo di addestramento e all'aumento di occupazione di memoria del classificatore.

Si è infine potuto verificare che il tempo che l'ensemble impiega per effettuare una predizione su un ritaglio è leggermente più alto a quello impiegato dal classificatore di CropFin v1.

I problemi evidenziati, soprattutto per quanto concerne le dimensioni del classificatore e il tempo necessario ad effettuare una predizione (requisiti fondamentali per agevolare il lavoro del biologo) sono però quasi ininfluenti su una macchina dal discreto potere computazionale e memoria disponibile.

I motivi sopracitati rendono il nuovo classificatore ensemble preferibile a quello nativo di CropFin v1.



# Capitolo 4

## Conclusioni e sviluppi futuri

Nel presente lavoro di tesi è stato affrontato un problema di *object detection* con oggetto il rilevamento di pinne dorsali di cetacei in una collezione di immagini, adoperando tecniche di *computer vision* e di *deep learning*. In particolare, si è deciso di utilizzare il metodo del *transfer learning* per migliorare le prestazioni registrate dagli algoritmi CropFin v1 e v2 sviluppati in [1] che per primi hanno risolto il task in esame, e che costituiscono il punto di partenza degli esperimenti effettuati in questa tesi.

L'output degli esperimenti è stato un nuovo classificatore *ensemble*, composto da tre reti neurali convoluzionali pre-addestrate di diversa profondità e capacità, adattate a risolvere il problema della classificazione binaria 'Pinna'/'No Pinna', a partire dai ritagli delle eventuali pinne di un'immagine prodotti dalla prima parte della routine CropFin v1. Questo classificatore si innesta nella routine CropFin v1 andando a sostituire il classificatore nativo, basato sull'utilizzo di una CNN *from scratch*, registrando un netto miglioramento di prestazioni nella fase di classificazione di ritagli (test error 97.2% e specificity 98.1% su una collezione di ritagli mai vista dall'*ensemble*, contro il 92% e 95% del classificatore nativo). Il raggiungimento di tali prestazioni sono il risultato di diversi fattori: in primo luogo, l'alta capacità di rappresentazione e astrazione dei concetti raggiunte dalle tre reti utilizzate; inoltre, il dominio ristretto del problema in esame e la disponibilità di immagini per l'addestramento in numerose condizioni di scatto, che garantiscono buona generalizzazione al nostro modello di classificazione.

Il tentativo di un nuovo approccio al problema con la tecnica del *transfer learning*, suggerito dai recenti successi registrati grazie all'utilizzo di questa tecnica ([35], [36], [37], tra quelli più vicini al problema in esame), è pienamente giustificato dagli ottimi risultati ottenuti in termini di prestazioni.

Il problema di rilevamento delle pinne di cetacei all'interno di un'immagine si inserisce in un più ampio problema di *object detection*, che ha come oggetto la foto-identificazione automatica dei delfini avvistati durante le campagne di avvistamento in mare aperto. In previsione di un utilizzo dei ritagli classificati dall'*ensemble* di reti creato da parte di routine che compiano appunto questo foto-riconoscimento (*SPIR* [2] e *SPIR v2*[3]) è stato fondamentale ridurre quanto più possibile il valore

---

di specificity (che dà una misura di quanto i falsi positivi "inquinano" i ritagli invece correttamente classificati come 'Pinna') per dare in input a queste routine solo ritagli che raffigurino effettivamente una pinna.

La metodologia di *object detection* introdotta può essere il punto di partenza per interessanti lavori simili e sviluppi futuri.

Nell'ambito dei lavori affini a quello presentato in questo elaborato, si può pensare di riutilizzare il classificatore per risolvere il task di riconoscimento delle pinne dorsali anche di quelle specie marine il cui studio da parte dei biologi preveda il foto-riconoscimento degli esemplari a partire dalla loro pinna dorsale, quali orche (*Orcinus orca*) [38] e squali [39]. Il ri-adattamento del classificatore è operato, ancora una volta, mediante la tecnica del *transfer learning* applicata su di esso.

Un interessante sviluppo futuro del lavoro appena concluso può riguardare un ulteriore filtraggio del dataset dei ritagli da presentare all'algoritmo di foto-identificazione. Gli algoritmi di *photo-ID* dei cetacei a partire dalla loro pinna, come quello descritto in [3], hanno il difetto di assegnare *sempre* una "identità" probabile ad un ritaglio (prodotto ad esempio come output di CropFin v1), anche nel caso in cui esso non rappresenti davvero una pinna. In questo senso, cercare di migliorare il più possibile la specificity del classificatore su un generico test set diventa di primaria importanza. Ma non solo: anche quei ritagli che sono correttamente etichettati come pinne possono in vario modo essere non idonei al foto-riconoscimento automatizzato per mezzo di un algoritmo (ad esempio perché la pinna risulta troppo piccola, sfocata, disturbata da schizzi d'acqua: in generale sono problematici tutti quei ritagli in cui le *feature* che permettono un'identificazione univoca dell'esemplare, come i noti graffi dei grampi o il bordo delle pinne dei tursiopi, sono scarsamente evidenti e pertanto inaffidabili).

Per risolvere questa criticità si può pensare di agire ulteriormente sul set dei ritagli da classificare, escludendo quelli ritenuti dalla macchina "non idonei" al successivo foto-riconoscimento, magari mediante l'uso di tecniche di *computer vision* che permettano di filtrare ulteriormente i ritagli. Ad esempio, sarebbe utile riconoscere (ed escludere) quei ritagli in cui la pinna appare troppo sfocata (la macchina può verificare che il gradiente ai bordi della pinna è poco "ripido" e porta dal grigio della pinna al blu-verde dell'acqua senza significativa soluzione di continuità). In alternativa, si può tentare un approccio che prevede il ri-addestramento del classificatore ensemble proposto in questa tesi (ancora una volta con la tecnica del *transfer learning*) presentandogli come training set una collezione di ritagli divisi tra 'Idoneo' e 'Non idoneo' al foto-riconoscimento.

# Capitolo 5

## Listato del codice sorgente

### 5.1 Funzioni utili per il transfer learning

#### 5.1.1 createLgraphUsingConnections.m

```
1 % lgraph = createLgraphUsingConnections(layers , connections) crea il
2 % grafo
3 % di una rete con strati specificati nel layer array 'layers' e
4 % connessi
5 % con le connessioni specificate nella tabella 'connections'.
6
7 function lgraph = createLgraphUsingConnections(layers , connections)
8
9 lgraph = layerGraph();
10 for i = 1:numel(layers)
11     lgraph = addLayers(lgraph , layers(i));
12 end
13
14 for c = 1:size(connections ,1)
15     lgraph = connectLayers(lgraph , connections . Source{c} , connections .
16     Destination{c});
17 end
18
19 end
```

#### 5.1.2 findLayersToReplace.m

```
1 % findLayersToReplace(lgraph) trova il layer di classificazione e il
2 % layer addestrabile ad esso precedente (FC o CONV) nel grafo specificato da
3 % 'lgraph'
4
5 function [ learnableLayer , classLayer ] = findLayersToReplace(lgraph)
6
7 if ~isa(lgraph , 'nnet.cnn.LayerGraph')
8     error('Argument must be a LayerGraph object.')
9 end
10
11 src = string(lgraph . Connections . Source);
12 dst = string(lgraph . Connections . Destination);
```

---

```

13 layerNames = string({lgraph.Layers.Name}) ;
14
15 % Trova il layer di classificazione (unico in un grafo)
16 % Nel caso di GoogLeNet, isClassificationLayer e' un logical array 144
17 % x1 con
18 % tutti 0 tranne che l'1 in posizione 144.
19 isClassificationLayer = arrayfun(@(l) ...
20     (isa(l, 'nnet.cnn.layer.ClassificationOutputLayer') | ...
21      isa(l, 'nnet.layer.ClassificationLayer')), lgraph.Layers);
22
23 if sum(isClassificationLayer) ~= 1
24     error('Il grafo della rete deve avere un unico layer di
25         classificazione')
26 end
27
28 % Il classification layer cosi' trovato
29 classLayer = lgraph.Layers(isClassificationLayer);
30
31 while true
32
33     if numel(classLayer) ~= 1
34         message = "Il grafo della rete deve avere un unico learnable
35             " + ...
36             "layer prima del layer di classificazione";
37         error(message)
38     end
39
40     currentLayerType = class(lgraph.Layers(classLayer));
41     isLearnableLayer = ismember(currentLayerType, ...
42         [ 'nnet.cnn.layer.FullyConnectedLayer', 'nnet.cnn.layer.
43             Convolution2DLayer' ]);
44
45     if isLearnableLayer
46         learnableLayer = lgraph.Layers(classLayer);
47         return
48     end
49
50 end
51
52 end

```

### 5.1.3 freezeWeights.m

```

1 % layers = freezeWeights(layers) setta il learning rate dei layer
2 % specificati nel layer array 'layers' a 0.
3
4 function layers = freezeWeights(layers)
5
6 for ii = 1:numel(layers)    %Cicla da 1 al num. di layer da freezare
7     props = properties(layers(ii));
8     for p = 1:numel(props)

```

## 5.2. RI-ADDESTRAMENTO CNN

---

```
9     propName = props{p};  
10    if ~isempty(regexp(propName, 'LearnRateFactor$', 'once'))  
11        %se presenti, setta a 0 i campi WeighLearnRateFactor e  
12        %BiasLearnRateFactor  
13        layers(ii).(propName) = 0;  
14    end  
15 end  
16  
17 end
```

### 5.1.4 plotConfusionMatrix.m

```
1 function plotConfusionMatrix(predictedLabels, trueLabels)  
2 % Costruisce e plotta la matrice di confusione per i dati specificati  
3 % predictedLabels: risultati della classificazione (predizioni)  
4 % trueLabels: classi effettive delle immagini  
5  
6 figure('Units','normalized','Position',[0.2 0.2 0.4 0.4]);  
7 cm = confusionchart(trueLabels, predictedLabels, 'FontSize', 15);  
8 cm.Title = 'Confusion Matrix for Validation Data';  
9 cm.ColumnSummary = 'column-normalized';  
10 cm.RowSummary = 'row-normalized';  
11  
12 end
```

## 5.2 Ri-addestramento CNN

### 5.2.1 AlexNet

```
1 %AlexNet  
2  
3 clear all  
4 clc  
5  
6 %% Preparazione dataset  
7  
8 % Metto dataset con i crop in un oggetto di tipo datastore  
9 datasetPath = 'Dataset Taranto';  
10 cropDS = imageDatastore(datasetPath, ...  
11     'IncludeSubfolders', true, ...  
12     'LabelSource', 'foldernames');  
13  
14 % Split in datastore di train e validation  
15 [cropTrain, cropValidation] = splitEachLabel(cropDS, 0.7, 'randomized');  
16  
17  
18 %% Inizializzazione CNN  
19  
20 % Carico la rete AlexNet  
21 net = alexnet;  
22 % Dimensioni immagine di input  
23 inputSize = net.Layers(1).InputSize;  
24 %numClasses: numero di categorie di classificazione (2)
```

---

```

25 numClasses = numel(categories(cropTrain.Labels));
26
27 % ESTRAZIONE DEI LAYER DA TRASFERIRE
28 % Gli ultimi tre layer della AlexNet pre-addestrata sono configurati
29 % per
30 % 1000 classi. Questi tre layer devono essere ri-configurati per il
31 % nuovo
32 % problema di classificazione (=> per il nuovo dataset). Questi layer
33 % sono:
34 % 1) 'fc8' FC layer —> last learnable layer (=ultimo layer provvisto
35 % di
36 % parametri addestrabili, e quindi di learning rate)
37 % 2) 'prob' Softmax layer (sta per "probabilities"; applica la softmax
38 % function)
39 % 3) 'output' Classification Output Layer (calcola la cross-entropy
40 % loss)
41
42 % Estraе tutti i layer tranne gli ultimi tre (che saranno da sostituire
43 %),
44 layersTransfer = net.Layers(1:end-3);
45
46 %% Freeze dei weights dei primi 2 layer convoluzionali
47 layersTransfer(2).WeightLearnRateFactor = 0;
48 layersTransfer(2).BiasLearnRateFactor = 0;
49
50 %% Ricostruzione della rete
51
52 %layers e' il nuovo "grafo" della rete, da addestrare
53 layers = [
54
55     layersTransfer %i primi 22
56
57     fullyConnectedLayer(numClasses, 'WeightLearnRateFactor', 10, %
58                         'BiasLearnRateFactor', 10, 'Name', 'new_fc8'), %nota il lr
59                         %abbastanza alto
60     softmaxLayer('Name', 'new_prob'), %calcola la probabilita
61                         %per ogni classe
62     classificationLayer('Name', 'new_output')]; %calcola la cross-
63                         %entropy loss Li
64
65 %%NB: i nuovi layer introdotti inferiscono il numero di classi (
66 %OutputSize) dall'OutputSize del layer precedente (il. For example,
67 %to specify the number of classes K of the
68 %network, include a fully connected layer with output size K and a
69 %softmax
70
71 %%layer before the classification layer.
72
73 %% NB:
74 %il nuovo Fully Connected Layer ha OutputSize=numClasses=2 e

```

## 5.2. RI-ADDESTRAMENTO CNN

---

```
68 % InputSize='auto', cioe' in fase di trainNetwork legge l'OutputSize
       del
69 % Fully Connected Layer a lui precedente (ci sono in questo caso i
       layer
70 % ReLU e Dropout tra i due Fully Connected in questione), che nel caso
       di
71 % AlexNet e' 4096.
72 % -il nuovo Classification Layer ha OutputSize='auto', letto in fase di
73 % trainNetwork dall'OutputSize=numClasses=2 del Fully Connected Layer a
       lui
74 % precedente, e anche Classes='auto', impostato a [ 'No Pinna' , 'Pinna' ]
       dal
75 % cropAugmentedValidation (in qualche modo...)
76
77
78 %% Image pre-processing e augmentation
79
80 pixelRange = [-60 60];
81 angleRange = [-20,20];
82 augmenter = imageDataAugmenter( ...
83     'RandXReflection',true, ...
84     'RandRotation',angleRange, ...
85     'RandXTranslation',pixelRange);
86
87 %Training set aumentato e ridimensionato 227x227
88 cropAugmentedTrain = augmentedImageDatastore(inputSize(1:2),cropTrain
       ,...
       'DataAugmentation',augmenter);
89 %Validation set ridimensionato 227x227
90 cropAugmentedValidation = augmentedImageDatastore(...
91     inputSize(1:2),cropValidation);
92
93
94
95 %% TRAINING
96
97 miniBatchSize = 20;
98 valFrequency = floor(numel(cropAugmentedTrain.Files)/miniBatchSize);
99 options = trainingOptions('sgdm', ...
100    'MiniBatchSize',miniBatchSize, ...
101    'MaxEpochs',6, ...
102    'InitialLearnRate',3e-4, ...
103    'Shuffle','every-epoch', ...
104    'ValidationData', cropAugmentedValidation, ...
105    'ValidationFrequency',valFrequency, ...
106    'Verbose',true, ...
107    'Plots','training-progress', ...
108    'ExecutionEnvironment','parallel', ...
109    'CheckpointPath','.\Checkpoint AlexNet');
110
111 TL_net = trainNetwork(cropAugmentedTrain, layers, options);
112
113
114 %% Classificazione immagini del validation set
115
116 [prediction,probs] = classify(TL_net,cropAugmentedValidation);
117 accuracy = mean(prediction == cropValidation.Labels)
```

---

```

118
119
120 %% Matrice di confusione
121
122 plotConfusionMatrix(prediction , cropValidation . Labels )
123 saveas(gcf , 'confMat_AlexNet.jpg' );
124
125
126 %% Salvataggio
127
128 %Salvataggio workspace
129 save( 'workspace_alexnet.mat' );
130 %Salvataggio della rete addestrata
131 save( 'TL_alexnet.mat' , 'TL_net' );

```

## 5.2.2 GoogLeNet

```

1 %GoogLeNet
2
3 clear all
4 clc
5
6 %% Preparazione dataset
7
8 % Metto dataset in un oggetto di tipo datastore
9 datasetPath = 'Dataset Taranto';
10 cropDS = imageDatastore(datasetPath , ...
11     'IncludeSubfolders' , true , ...
12     'LabelSource' , 'foldernames' );
13
14 % Split in datastore di train e validation
15 [cropTrain , cropValidation ] = splitEachLabel(cropDS , 0.7 , 'randomized' );
16
17
18 %% Inizializzazione CNN
19
20 % Carico la rete GoogLeNet
21 net = googlenet;
22 % Dimensioni immagine di input
23 inputSize = net.Layers(1).InputSize;
24 %numClasses: numero di categorie di classificazione (2)
25 numClasses = numel(categories(cropTrain.Labels));
26
27 %% Sostituzione last learnable layer e classification layer
28
29 %Grafo della rete originale
30 lgraph = layerGraph(net);
31
32 % Trova automaticamente i layer da rimpiazzare
33 [learnableLayer , classLayer] = findLayersToReplace(lgraph);
34
35 if isa(learnableLayer , 'nnet.cnn.layer.FullyConnectedLayer')
36     newLearnableLayer = fullyConnectedLayer(numClasses , ...
37         'Name' , 'new_fc' , ...
38         'WeightLearnRateFactor' , 10 , ...

```

## 5.2. RI-ADDESTRAMENTO CNN

---

```
39      'BiasLearnRateFactor',10);
40
41 elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
42     newLearnableLayer = convolution2dLayer(1,numClasses, ...
43         'Name','new_conv', ...
44         'WeightLearnRateFactor',10, ...
45         'BiasLearnRateFactor',10);
46 end
47
48 %Rimpiazza il learnable layer nel grafo
49 lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);
50
51 % Crea un nuovo classification layer, che avra' 0 classi (le classi
52 % saranno automaticamente associate a questo layer in fase di trainNetwork)
53 newClassLayer = classificationLayer('Name','new_classoutput');
54
55 %Rimpiazza il learnable layer nel grafo
56 lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);
57
58
59 %% Freeze dei weights dei primi 3 layer convoluzionali
60
61 %Strati e connessioni della rete originale
62 layers = lgraph.Layers;
63 connections = lgraph.Connections;
64
65 % Blocco il learning rate di pesi e bias
66 layers(1:10) = freezeWeights(layers(1:8));
67 % Riconnetto tutti i layer nell'ordine originario
68 lgraph = createLgraphUsingConnections(layers,connections);
69
70
71 %% Image pre-processing e augmentation
72
73 pixelRange = [-60 60];
74 angleRange = [-20,20];
75 augmenter = imageDataAugmenter( ...
76     'RandXReflection',true, ...
77     'RandRotation',angleRange, ...
78     'RandXTranslation',pixelRange);
79
80 % Training set aumentato e ridimensionato 224x224
81 cropAugmentedTrain = augmentedImageDatastore(inputSize(1:2),cropTrain
82     ,...
83     'DataAugmentation',augmenter);
84 % Validation set ridimensionato 224x224
85 cropAugmentedValidation = augmentedImageDatastore(...
86     inputSize(1:2),cropValidation);
87
88 %% TRAINING
89 miniBatchSize = 20;
90 valFrequency = floor(numel(cropAugmentedTrain.Files)/miniBatchSize);
91 options = trainingOptions('sgdm',...
92     'MiniBatchSize',miniBatchSize,...
```

---

```

93    'MaxEpochs',6, ...
94    'InitialLearnRate',3e-4, ...
95    'Shuffle','every-epoch', ...
96    'ValidationData', cropAugmentedValidation, ...
97    'ValidationFrequency',valFrequency, ...
98    'Verbose',true, ...
99    'Plots','training-progress', ...
100   'CheckpointPath','.\Checkpoint GoogLeNet');

101 TL_net = trainNetwork(cropAugmentedTrain,lgraph,options);
103
104
105 %% Classificazione immagini del validation set
106
107 [prediction,probs] = classify(TL_net,cropAugmentedValidation);
108 accuracy = mean(prediction == cropValidation.Labels)
109
110
111 %% Matrice di confusione
112
113 plotConfusionMatrix(prediction,cropValidation.Labels)
114 saveas(gcf,'confMat_GoogLeNet.jpg');
115
116
117 %% Salvataggio
118
119 %Salvataggio workspace
120 save('workspace_goolenet.mat');
121 %Salvataggio della rete addestrata
122 save('TL_goolenet.mat','TL_net');

```

### 5.2.3 ResNet-18

```

1 %ResNet-18
2
3 clear all
4 clc
5
6 %% Preparazione dataset
7
8 %Metto dataset in un oggetto di tipo datastore
9 datasetPath = 'Dataset Taranto';
10 cropDS = imageDatastore(datasetPath, ...
11     'IncludeSubfolders',true, ...
12     'LabelSource','foldernames');
13
14 % Split in datastore di train e validation
15 [cropTrain,cropValidation] = splitEachLabel(cropDS,0.7,'randomized');
16
17
18 %% Inizializzazione CNN
19
20 % Carico la rete ResNet-18
21 net = resnet18;
22 % Dimensioni immagine di input

```

## 5.2. RI-ADDESTRAMENTO CNN

---

```
23 inputSize = net.Layers(1).InputSize;
24 %numClasses: numero di categorie di classificazione (2)
25 numClasses = numel(categories(cropTrain.Labels));
26
27 %Grafo della rete originale
28 lgraph = layerGraph(net);
29
30 % learnableLayer = lgraph.Layers(70);
31 % softmaxLayer = lgraph.Layers(71);
32 % classLayer = lgraph.Layers(72);
33
34 % Layer rimpiazzanti
35 newLearnableLayer = fullyConnectedLayer(numClasses, 'Name', 'new_fc1000',
36     ...
37     'WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10);
38 newSoftmaxLayer = softmaxLayer('Name', 'new_prob');
39 newClassLayer = classificationLayer(...,
40     'Name', 'new_ClassificationLayer_predictions');
41
42 % Rimpiazza degli ultimi 3 layer
43 lgraph = replaceLayer(lgraph, lgraph.Layers(70).Name, newLearnableLayer);
44 lgraph = replaceLayer(lgraph, lgraph.Layers(71).Name, newSoftmaxLayer);
45 lgraph = replaceLayer(lgraph, lgraph.Layers(72).Name, newClassLayer);
46
47 %% Freeze dei weights del primo layer convoluzionale
48
49 %Strati e connessioni della rete originale
50 layers = lgraph.Layers;
51 connections = lgraph.Connections;
52
53 % Blocca il learning rate di pesi e bias
54 layers(1:3) = freezeWeights(layers(1:3));
55 % Riconnetti tutti i layer nell'ordine originario
56 lgraph = createLgraphUsingConnections(layers, connections);
57
58
59 %% Image pre-processing e augmentation
60
61 pixelRange = [-60 60];
62 angleRange = [-20, 20];
63 augmenteer = imageDataAugmenter( ...
64     'RandXReflection', true, ...
65     'RandRotation', angleRange, ...
66     'RandXTranslation', pixelRange);
67
68 % Training set aumentato e ridimensionato 224x224
69 cropAugmentedTrain = augmentedImageDatastore(inputSize(1:2), cropTrain, ...
    DataAugmentation', augmenteer);
70 % Validation set ridimensionato 224x224
71 cropAugmentedValidation = augmentedImageDatastore(inputSize(1:2), ...
    cropValidation);
72
73
74 %% TRAINING
75
```

---

```

76 miniBatchSize = 20;
77 valFrequency = floor(numel(cropAugmentedTrain.Files)/miniBatchSize);
78 options = trainingOptions('sgdm', ...
79     'MiniBatchSize',miniBatchSize, ...
80     'MaxEpochs',6, ...
81     'InitialLearnRate',3e-4, ...
82     'Shuffle','every-epoch', ...
83     'ValidationData', cropAugmentedValidation, ...
84     'ValidationFrequency',valFrequency, ...
85     'Verbose',true, ...
86     'Plots','training-progress', ...
87     'ExecutionEnvironment','parallel', ...
88     'CheckpointPath','.\Checkpoint ResNet-50');

89
90 TL_net = trainNetwork(cropAugmentedTrain,lgraph,options);
91
92
93 %% Classificazione immagini del validation set
94
95 [prediction,probs] = classify(TL_net,cropAugmentedValidation);
96 accuracy = mean(prediction == cropValidation.Labels)
97
98
99 %% Matrice di confusione
100
101 plotConfusionMatrix(prediction,cropValidation.Labels)
102 saveas(gcf,'confMat ResNet18.jpg');
103
104
105 %% Salvataggio
106
107 %Salvataggio workspace
108 save('workspace_resnet18.mat');
109 %Salvataggio della rete addestrata
110 save('TL_resnet18.mat','TL_net');

```

## 5.2.4 ResNet-50

```

1 %ResNet-50
2
3 clear all
4 clc
5
6 %% Preparazione dataset
7
8 %Metto dataset in un oggetto di tipo datastore
9 datasetPath = 'Dataset Taranto';
10 cropDS = imageDatastore(datasetPath, ...
11     'IncludeSubfolders',true, ...
12     'LabelSource','foldernames');

13
14 % Split in datastore di train e validation
15 [cropTrain,cropValidation] = splitEachLabel(cropDS,0.7,'randomized');
16
17

```

## 5.2. RI-ADDESTRAMENTO CNN

---

```
18 %% Inizializzazione CNN
19
20 %Carico la rete ResNet-50
21 net = resnet50;
22 % Dimensioni immagine di input
23 inputSize = net.Layers(1).InputSize;
24 %numClasses: numero di categorie di classificazione (2)
25 numClasses = numel(categories(cropTrain.Labels));
26
27 %Grafo della rete originale
28 lgraph = layerGraph(net);
29
30 % learnableLayer = lgraph.Layers(175);
31 % softmaxLayer = lgraph.Layers(175);
32 % classLayer = lgraph.Layers(177);
33
34 % Layer rimpiazzanti
35 newLearnableLayer = fullyConnectedLayer(numClasses, 'Name', 'new_fc1000',
    ...
    'WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10);
36 newSoftmaxLayer = softmaxLayer('Name', 'new_fc1000_softmax');
37 newClassLayer = classificationLayer(...,
    'Name', 'new_ClassificationLayer_fc1000');
38
39 % Rimpiazza degli ultimi 3 layer
40 lgraph = replaceLayer(lgraph, lgraph.Layers(175).Name, newLearnableLayer)
    ;
41 lgraph = replaceLayer(lgraph, lgraph.Layers(176).Name, newSoftmaxLayer);
42 lgraph = replaceLayer(lgraph, lgraph.Layers(177).Name, newClassLayer);
43
44 % Plotta il nuovo grafo ottenuto
45 plot(lgraph); ylim([0,10]);
46
47
48
49
50 %% Freeze dei weights del primo layer convoluzionale
51
52 %Strati e connessioni della rete originale
53 layers = lgraph.Layers;
54 connections = lgraph.Connections;
55
56 % Blocca il learning rate di pesi e bias nei primi 10 layer (the
      initial 'stem' of the ResNet network)
57 layers(1:2) = freezeWeights(layers(1:2));
58 % Riconnetti tutti i layer nell'ordine originario
59 lgraph = createLgraphUsingConnections(layers, connections); %PERMETTE
      DI AGGIRARE LA SOLA LETTURA DEGLI ATTRIBUTI
60
61
62 %% Re-addestramento ResNet-50 – Image pre-processing e augmentation
63
64 pixelRange = [-60 60];
65 angleRange = [-20, 20];
66 augmenteer = imageDataAugmenter( ...
    'RandXReflection', true, ...
    'RandRotation', angleRange, ...
    'RandXTranslation', pixelRange);
```

---

```

70
71 %Training set aumentato e ridimensionato 224x224
72 cropAugmentedTrain = augmentedImageDatastore(inputSize(1:2),cropTrain,
    DataAugmentation',augmenter);
73 %Validation set ridimensionato 224x224
74 cropAugmentedValidation = augmentedImageDatastore(inputSize(1:2),
    cropValidation);

75
76
77 %% TRAINING
78
79 miniBatchSize = 20;
80 valFrequency = floor(numel(cropAugmentedTrain.Files)/miniBatchSize);
81 options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData', cropAugmentedValidation, ...
    'ValidationFrequency',valFrequency, ...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'ExecutionEnvironment','parallel', ...
    'CheckpointPath','.\Checkpoint ResNet-50');

82
83
84
85
86
87
88
89
90
91
92 TL_net = trainNetwork(cropAugmentedTrain,lgraph,options);
93
94
95
96 %% Classificazione immagini del validation set
97
98 [prediction,probs] = classify(TL_net,cropAugmentedValidation);
99 accuracy = mean(prediction == cropValidation.Labels)

100
101
102 %% Matrice di confusione
103
104 plotConfusionMatrix(prediction,cropValidation.Labels)
105 saveas(gcf,'confMat ResNet50.jpg');
106
107 %% Salvataggio
108
109 %Salvataggio workspace
110 save('workspace_resnet50.mat');
111 %Salvataggio della rete addestrata
112 save('TL_resnet50.mat','TL_net');

```

## 5.3 Test sul dataset delle Azzorre

### 5.3.1 Singoli classificatori

```

1 clear
2 clc
3
4 %% DESCRIZIONE
5 % Questo programma effettua le seguenti operazioni:

```

### 5.3. TEST SUL DATASET DELLE AZZORRE

---

```
6 % 1. test sul dataset delle azzorre con AlexNet, GoogLeNet, ResNet18
7 % 2. creazione di un file excel con due colonne, contenenti ciascuna
8 % gli indirizzi delle immagini classificate come 'Pinna' e come 'No
9 % Pinna'
10
11 %% INIZIALIZZAZIONE DATASET
12
13 datasetPath = 'D:\Dati utente\Desktop\Tesi\Lavoro\Dataset Azzorre';
14 datasetDS = imageDatastore(datasetPath, ...
15     'IncludeSubfolders',true, ...
16     'LabelSource','foldernames');
17
18 %% CARICAMENTO RETI + TEST + OUTPUT FILE EXCEL
19
20 netsList_temp = dir('TL_*');
21 % Escludi le reti da non usare nel major voting (in questo caso ResNet
22 % -50)
22 j=1;
23 for i=1:numel(netsList_temp)
24     if string(netsList_temp(i).name)~="TL_resnet50.mat"
25         netsList(j) = netsList_temp(i);
26         j=j+1;
27     end
28 end
29
30 n = numel(netsList);
31
32 for i=1:n
33     %Caricamento della rete i-esima
34     net_toExtract = load(netsList(i).name);
35     name_asCellArray = fieldnames(net_toExtract);
36     name = name_asCellArray{1};
37     net = net_toExtract.(name);
38
39     %Ridimensionamento del dataset all'inputSize della rete i-esima
40     inputSize = net.Layers(1).InputSize;
41     datasetDS_resize = augmentedImageDatastore(inputSize(1:2),datasetDS
42 );
43
44     %Classificazione con rete i-esima
45     [prediction,probs] = classify(net,datasetDS_resize);
46     %predictionProbs = (max(probs'));
47     accuracy = mean(prediction == datasetDS.Labels)
48
49     %Salvataggio risultati della rete i-esima in un file excel
50     results = table(datasetDS.Files,datasetDS.Labels,prediction,probs
51     (:,1),probs(:,2), ...
52     'VariableNames',{ 'Crop', 'TrueClass', 'Prediction', 'Prob_No_Pinna
53     ', 'Prob_Pinna'});
54     netName = netsList(i).name(4:end-4);
55     outputExcel = ['Risultati Azzorre ',netName,'.xls'];
56     writetable(results,outputExcel);
57
58     %Stampa matrice di confusione relativa alla rete i-esima
```

---

```

56 plotConfusionMatrix( prediction , datasetDS . Labels )
57 saveas( gcf , [ 'confMat_Azzorre ' , netName , '.png' ] )
58 end

```

### 5.3.2 Classificatore ensemble

```

1 clear
2 clc
3
4 %% DESCRIZIONE – MAJOR VOTING
5 % Reti nell 'ensemble: AlexNet , GoogLeNet , ResNet18
6 % Ogni rete ha eguale peso nello schema di consenso che serve a
    prendere la
7 % decisione finale
8
9
10 %% LETTURA FILE EXCEL
11 T1 = readtable( 'Risultati Azzorre alexnet.xls' );
12 T2 = readtable( 'Risultati Azzorre googlenet.xls' );
13 T3 = readtable( 'Risultati Azzorre resnet18.xls' );
14
15 trueClass = categorical( table2array( T1(:,2) ) ); % vere etichette
16 v1 = categorical( table2array( T1(:,3) ) );           % etichette date da
    alexnet
17 v2 = categorical( table2array( T2(:,3) ) );           % etichette date da
    googlenet
18 v3 = categorical( table2array( T3(:,3) ) );           % etichette date da
    resnet18
19 p1_np = double( table2array( T1(:,4) ) );           % probabilita 'No
    pinna' alexnet
20 p1_p = double( table2array( T1(:,5) ) );           % probabilita 'Pinna'
    alexnet
21 p2_np = double( table2array( T2(:,4) ) );
22 p2_p = double( table2array( T2(:,5) ) );
23 p3_np = double( table2array( T3(:,4) ) );
24 p3_p = double( table2array( T3(:,5) ) );
25
26 %Probabilita' medie (tra le 3 reti) delle classi 'Pinna' e 'No Pinna'
27 %(model averaging)
28 probsPinna = mean([p1_p,p2_p,p3_p] )';
29
30 % tol: tolleranza , cioe' minima probabilita' accettata per 'Pinna',
    nell 'HMV
31 % 0.0 < tol < 1.0
32 tol = 0.97;
33
34 % votes: voti per 'Pinna' (servono per l 'HMV)
35 votes = sum(([v1,v2,v3]== 'Pinna' ) )';
36
37
38 %% VOTAZIONE
39 for i=1:size(votes,1)
40
41     % soft major voting classico
42     if probsPinna(i)>0.5

```

### 5.3. TEST SUL DATASET DELLE AZZORRE

---

```
43     predictionSoft{i} = 'Pinna';
44     probsSoft(i) = probsPinna(i);
45 else
46     predictionSoft{i} = 'No Pinna';
47     probsSoft(i) = 1-probsPinna(i);
48 end
49
50 % hard major voting con minima tolleranza per la probabilita' media
51 di
52 % 'Pinna'
53 if votes(i)>=2 & probsPinna(i)>tol
54     predictionHard{i} = 'Pinna';
55     probsHard(i) = probsPinna(i);
56 else
57     predictionHard{i} = 'No Pinna';
58     probsHard(i) = 1-probsPinna(i);
59 end
60
61 predictionSoft = categorical(predictionSoft');
62 predictionHard = categorical(predictionHard');
63 probsSoft = probsSoft';
64 probsHard = probsHard';
65
66
67 %% SALVATAGGIO RISULTATI IN FILE EXCEL
68
69 resultsSoft = table(predictionSoft,probsSoft,'VariableNames',...
70     {'Prediction','Accuracy'});
71 writetable(resultsSoft,'Risultati Azzorre soft major voting.xls');
72
73 resultsHard = table(predictionHard,probsHard,'VariableNames',...
74     {'Prediction','Accuracy'});
75 writetable(resultsHard,'Risultati Azzorre hard major voting.xls');
76
77
78 %% MATRICE DI CONFUSIONE
79
80 plotConfusionMatrix(predictionSoft,trueClass)
81 saveas(gcf,'confMat Azzorre soft major voting.png')
82
83 plotConfusionMatrix(predictionHard,trueClass)
84 saveas(gcf,'confMat Azzorre hard major voting.png')
```



# Bibliografia

- [1] Gianvito Losapio. *Tecniche di Deep Learning e Object Detection per la foto-identificazione dei cetacei*. Tesi di laurea triennale, a.a. 2018/2019, rel. prof. Tiziano Politi, correl. dr. Vito Renò.
- [2] Rosalia Maglietta et al. «DolFin: an innovative digital platform for studying Risso's dolphins in the Northern Ionian Sea (North-eastern Central Mediterranean)». In: *Scientific Reports* 8.1 (nov. 2018), p. 17185. ISSN: 2045-2322.
- [3] Emanuele Seller. *Pipeline innovativa per la foto-identificazione del Grampus Griseus*. Tesi di laurea triennale, a.a. 2018/2019, rel. prof. Giovanni Dimauro, correl. dr.ssa Rosalia Maglietta.
- [4] Flavio Forenza. *Tecniche innovative di Computer Vision per la Foto-Identificazione dei cetacei*. Tesi di laurea triennale, a.a. 2017/2018, rel. prof. Giovanni Dimauro, correl. dr. Vito Renò. 2017.
- [5] Ankita Shukla et al. «A Hybrid Approach to Tiger Re-Identification». In: *The IEEE International Conference on Computer Vision (ICCV) Workshops*. Ott. 2019.
- [6] Gao Huang et al. «Densely Connected Convolutional Networks». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, lug. 2017.
- [7] Dmitry A. Konovalov et al. «Individual Minke Whale Recognition Using Deep Learning Convolutional Neural Networks». In: *Journal of Geoscience and Environment Protection* 06.05 (2018), pp. 25–36.
- [8] Olga Moskvyak et al. «Robust Re-identification of Manta Rays from Natural Markings by Learning Pose Invariant Embeddings». In: (feb. 2019).
- [9] Florian Schroff, Dmitry Kalenichenko e James Philbin. «FaceNet: A unified embedding for face recognition and clustering». In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, giu. 2015.
- [10] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. URL: <http://szeliski.org/Book/>.
- [11] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [12] Andrej Karpathy. *Lectures of Stanford class CS231n: Convolutional Neural Networks for Visual Recognition*. Appunti del corso Stanford CS class C231n. 2019.

- 
- [13] Olga Russakovsky et al. «ImageNet Large Scale Visual Recognition Challenge». In: *arXiv* (2014).
  - [14] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
  - [15] Stuart Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2009.
  - [16] Alex Krizhevsky. «Learning multiple layers of features from tiny images». In: (2009).
  - [17] G. Cybenko. «Approximation by superpositions of a sigmoidal function». In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314.
  - [18] Jason Yosinski et al. «How transferable are features in deep neural networks?» In: *Advances in Neural Information Processing Systems 27 (NIPS '14)*, Nips Foundation (2014).
  - [19] Lars Kai Hansen e Peter Salamon. «Neural Network Ensembles». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10 (1990).
  - [20] Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems 25*. 2012.
  - [21] Md Zahangir Alom et al. «The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches». In: (2018).
  - [22] Vinod Nair e Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». In: *ICML*. 2010.
  - [23] Sergio Bolasco. *Analisi multidimensionale dei dati. Metodi, strategie e criteri d'interpretazione*. Carocci Editore, 1999.
  - [24] Geoffrey E. Hinton et al. «Improving neural networks by preventing co-adaptation of feature detectors». In: (2012).
  - [25] Christian Szegedy et al. «Going Deeper with Convolutions». In: (2014).
  - [26] Min Lin, Qiang Chen e Shuicheng Yan. «Network In Network». In: (2013).
  - [27] Andrew G. Howard. «Some Improvements on Deep Convolutional Neural Network Based Image Classification». In: (2013).
  - [28] Kaiming He et al. «Deep Residual Learning for Image Recognition». In: () .
  - [29] Karen Simonyan e Andrew Zisserman. «Very Deep Convolutional Networks for Large-Scale Image Recognition». In: () .
  - [30] Rupesh Kumar Srivastava, Klaus Greff e Jürgen Schmidhuber. «Highway Networks». In: () .
  - [31] Sergey Ioffe e Christian Szegedy. «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». In: (2015).
  - [32] Shibani Santurkar et al. «How Does Batch Normalization Help Optimization?» In: *NeurIPS* (2018).

## BIBLIOGRAFIA

---

- [33] Kaiming He et al. «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification». In: ().
- [34] Nobuyuki Otsu. «A Threshold Selection Method from Gray-Level Histograms». In: *IEEE Trans. Syst. Man. Cybern.*, vol. 9, no. 1, pp. 62–66 (1979).
- [35] Mark Thomas et al. «Marine Mammal Species Classification using Convolutional Neural Networks and a Novel Acoustic Representation». In: (2019).
- [36] Emilio Guirado et al. «Automatic whale counting in satellite images with deep learning». In: (2018).
- [37] Hung Nguyen et al. «Animal Recognition and Identification with Deep Convolutional Neural Networks for Automated Wildlife Monitoring». In: *International Conference on Data Science and Advanced Analytics*. IEEE, 2017.
- [38] Brent G. Young, Jeff W. Higdon e Steven H. Ferguson. «Killer whale (*Orcinus orca*) photo-identification in the eastern Canadian Arctic». In: *Polar Research* 30.1 (2011).
- [39] Benjamin Hughes e Tilo Burghardt. «Automated Visual Fin Identification of Individual Great White Sharks». In: *International Journal of Computer Vision* 122.3 (2016), pp. 542–557.
- [40] Geok See Ng e Harcharan Singh. «Democracy in pattern classifications: combinations of votes from various pattern classifiers». In: *Artificial Intelligence in Engineering* 12 (1998), pp. 189–204.
- [41] Olga Moskvyak et al. «Robust Re-identification of Manta Rays from Natural Markings by Learning Pose Invariant Embeddings». In: *CoRR* abs/1902.10847 (2019). arXiv: 1902.10847. URL: <http://arxiv.org/abs/1902.10847>.
- [42] Douglas Adams. *Guida galattica per gli autostoppisti*. A cura di Mondadori. 1999.