



POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE - DEI
Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione (D.M. 270/04)

TESI DI LAUREA IN CALCOLO NUMERICO

ANALISI DI IMMAGINI CON RETI NEURALI CONVOLUZIONALI PER LA CLASSIFICAZIONE DEI CETACEI NEL GOLFO DI TARANTO

Relatore:
Chiar.mo Prof. Ing. Tiziano POLITI

Correlatore:
Dott. Ing. Vito RENÒ

Laureando:
Tommaso MONOPOLI
Matricola 568581

Sommario

Il sistema terrestre è sempre stato soggetto alle conseguenze delle attività umane, e la biodiversità degli ecosistemi acquatici e marini è fortemente a rischio. 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.

Un'attività mirata alla raccolta di informazioni rilevanti allo studio dei cetacei è la *foto-identificazione degli individui* di una specie, che prevede il riconoscimento - automatico o manuale - di uno stesso individuo in diverse immagini collezionate nel tempo, mediante l'analisi di particolari segni distintivi (*feature*) presenti nell'immagine.

Questa 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* per automatizzare tale attività. L'obiettivo del presente lavoro di tesi è la creazione di classificatori binari che ricevano in input un dataset di immagini bidimensionali collezionate nei pressi delle isole Azzorre (Oceano Atlantico settentrionale) e sappiano suddividere lo stesso dataset in due classi di immagini, a seconda che in ciascuna immagine sia rilevata o meno una *feature* utile ad una successiva foto-identificazione. Nel caso dei cetacei, il criterio di classificazione è la presenza nell'immagine della pinna dorsale dell'individuo.

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 dimostrano l'utilità di tali tecniche di Computer Vision nel campo della foto-identificazione dei cetacei.

Indice

1	Introduzione	1
1.1	Problema e obiettivi	1
2	Metodologie	3
2.1	Immagini digitali	3
2.1.1	Spazio di colore Lab	5
2.1.2	Collezioni di immagini in Matlab	5
2.2	Trasformazioni	6
2.2.1	Ridimensionamento	6
2.2.2	Rotazione	6
2.2.3	Riflessione	6
2.2.4	Traslazione	7
2.2.5	Convoluzione	7
2.3	Problemi di <i>Computer Vision</i>	9
2.3.1	Segmentazione	9
2.3.2	Object recognition	9
2.3.3	Object detection	10
2.4	Machine Learning e Deep Learning	12
2.4.1	Supervised Learning	13
2.4.2	Underfitting e overfitting	15
2.5	Classificazione	17
2.6	Classificatore lineare	17
2.7	Reti Neurali	22
2.7.1	I neuroni	23
2.7.2	Architettura delle reti neurali artificiali	24
2.8	Reti neurali convoluzionali	25
2.8.1	Input Layer	28
2.8.2	Convolution Layer	28
2.8.3	Activation layer	32
2.8.4	Pooling layer	32
2.8.5	Fully Connected Layer	33
2.8.6	Softmax layer	34
2.9	Image preprocessing e augmentation	34
2.10	Addestrare una rete neurale	38
2.10.1	Inizializzazione dei parametri	38
2.10.2	Loss function	39

2.10.3	Stochastic gradient descent	40
2.10.4	Backpropagation	43
2.11	AlexNet	45
2.11.1	Architettura di AlexNet	45
2.11.2	Funzione di attivazione ReLU	46
2.11.3	Local Response Normalization	46
2.11.4	Overlapping Max Pooling	46
2.11.5	Data Augmentation	47
2.11.6	Dropout	47
2.11.7	Addestramento di AlexNet	48
2.12	GoogLeNet	50
2.12.1	Convoluzione 1×1	51
2.12.2	Modulo <i>Inception</i>	51
2.12.3	Classificatori ausiliari	52
2.12.4	Global Average Pooling	53
2.12.5	Data Augmentation	53
2.12.6	Architettura di GoogLeNet	54
2.12.7	Addestramento di GoogLeNet	54
2.13	ResNet	56
2.13.1	<i>Batch Normalization</i>	57
2.13.2	Residual blocks	58
2.13.3	Architettura di ResNet-18	59
2.13.4	Architettura di ResNet-50	60
2.13.5	Data Augmentation	62
2.13.6	Addestramento di ResNet-18 e ResNet-50	62
3	Esperimenti e risultati	65
3.1	Descrizione dei dataset utilizzati	65
3.2	CropFin v1	66
3.2.1	Fase di ritaglio	67
3.2.2	Fase di classificazione	73
3.3	Classificazione mediante CNN e Transfer Learning	74
3.3.1	Scelta delle CNN	75
3.3.2	Addestramento	75
3.3.3	Classificatore ensemble	82
Conclusioni e sviluppi futuri	87	
Conclusioni	87	
Bibliografia	89	

Capitolo 1

Introduzione

La foto-identificazione è una tecnica largamente impiegata per l'identificazione dei singoli individui a partire da una o più immagini. Il principale vantaggio di questa tecnica è la sua non invasività che la rende particolarmente utile per studiare sia la dinamicità che i movimenti di ogni specie. Tale metodologia risulta essere uno strumento affidabile quando viene applicato nella comprensione dei comportamenti dei cetacei (migrazioni e spostamenti). Tra i delfini, vi sono due specie più adatte a tali studi. Il primo delfino riguarda la specie “*Tursiops truncatus*” (tursiope) o delfino dal naso a bottiglia, mentre la seconda specie riguarda la specie “*Grampus griseus*” (Grampo, o delfino di Risso), avente numerose cicatrici su tutto il corpo, entrambi appartenenti alla famiglia dei Delfinidi.[1]

1.1 Problema e obiettivi

L'obiettivo principale del presente lavoro di tesi è stato quello di creare un sistema per il rilevamento automatico della presenza di cetacei all'interno di uno scatto fotografico.

Lo scopo finale è facilitare lo studio dei cetacei, attorno al quale si riunisce grande interesse scientifico (par. ??), incentivando l'utilizzo di tecniche non invasive basate su algoritmi innovativi e grandi disponibilità di dati. Uno dei principali metodi di studio non-invasivi dei cetacei è la foto-identificazione degli esemplari (par. ??). Il presente lavoro di tesi rappresenta un passo avanti verso la completa automatizzazione del processo di foto-identificazione degli esemplari incontrati durante le campagne di avvistamento, fornendo un miglioramento nelle prestazioni di una routine (CropFin v1, par. 3.2) che può aiutare i biologi nel successivo lavoro di foto-identificazione.

Il problema affrontato TODO

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 presupposti teorici 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 [2].

Per i paragrafi dedicati al *machine* e *deep learning* le fonti di riferimento sono [3] e soprattutto [4].

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 dimensioni¹; 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 **altezza** e **larghezza** dell'immagine, la coppia (w, h) **risoluzione** mentre c è il numero di *canali di colore*². 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 **intensità luminosa** 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.

¹Ci 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.

²Spesso 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.

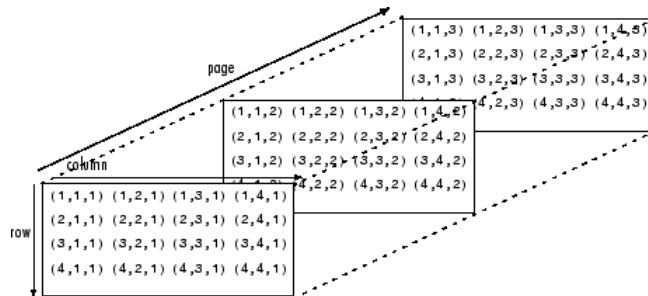


Figura 2.1: Rappresentazione grafica di un tensore tridimensionale; in ogni posizione compaiono gli indici del tensore

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

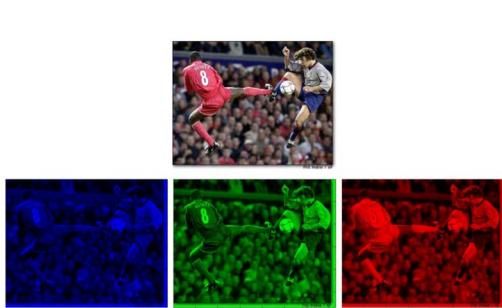


Figura 2.2: Canali RGB di un'immagine

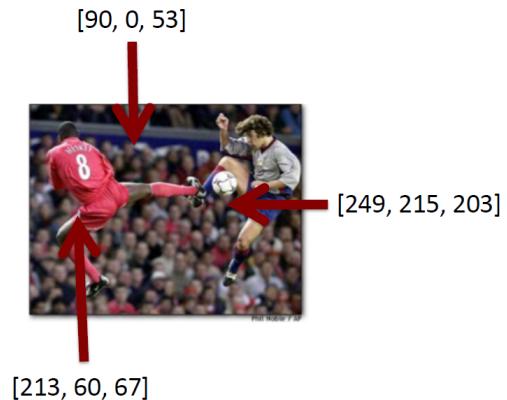


Figura 2.3: Pixel di un'immagine

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 dato³).

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)];
```

³https://it.mathworks.com/help/matlab/ref/image.html?s_tid=doc_ta#buqdlnb-C

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 triplettre 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. 2.9.

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. 2.9).

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`⁴, 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 α 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);
```

⁴La funzione `imresize` attua, per lo scopo, una tecnica avanzata di calcolo numerico: l'interpolazione bicubica. (TODO ? 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:

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

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

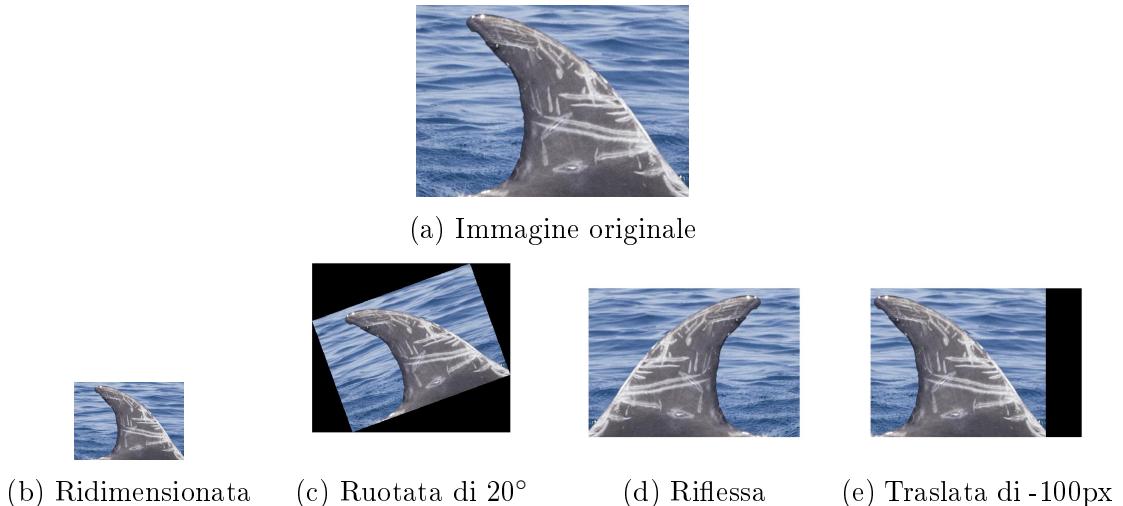


Figura 2.4: Trasformazioni applicate ad un'immagine digitale

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. 2.8), e più specificamente nei layer convoluzionali (par. ??).

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. 2.8.2).

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 $\mathcal{K} \in \mathbb{R}^{a \times b \times c}$. L'operazione viene così ridefinita.

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

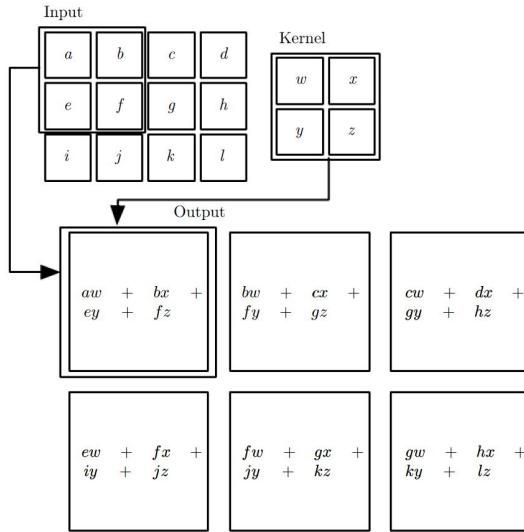


Figura 2.5: Un esempio di cross-correlazione (ovvero convoluzione discreta 2-D senza il ribaltamento del kernel)

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. ?? sul layer convoluzionale di una CNN.

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.

TODO TODO TODO vd 2.3 gianvito

2.3.1 Segmentazione

TODO

Soglia di Otsu TODO

Regioni connesse TODO

Riempimento degli holes TODO

2.3.2 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.

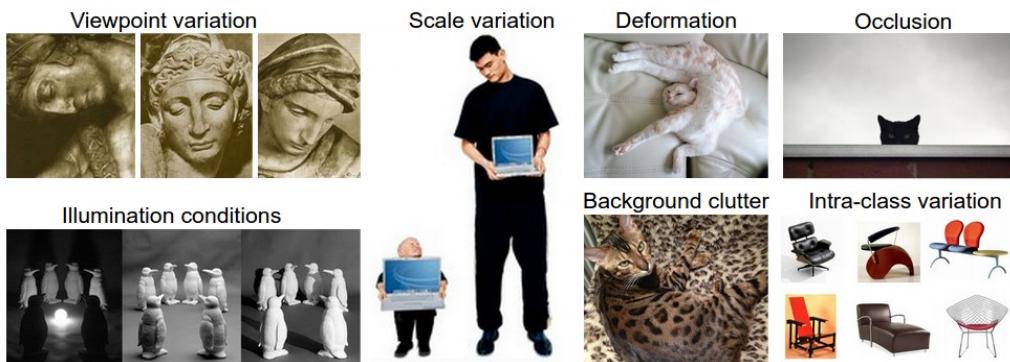


Figura 2.6: I principali "ostacoli" al riconoscimento automatico degli oggetti

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.

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.

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. 2.8) e che verranno impiegate negli esperimenti (par. 3.2.2).

2.3.3 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.

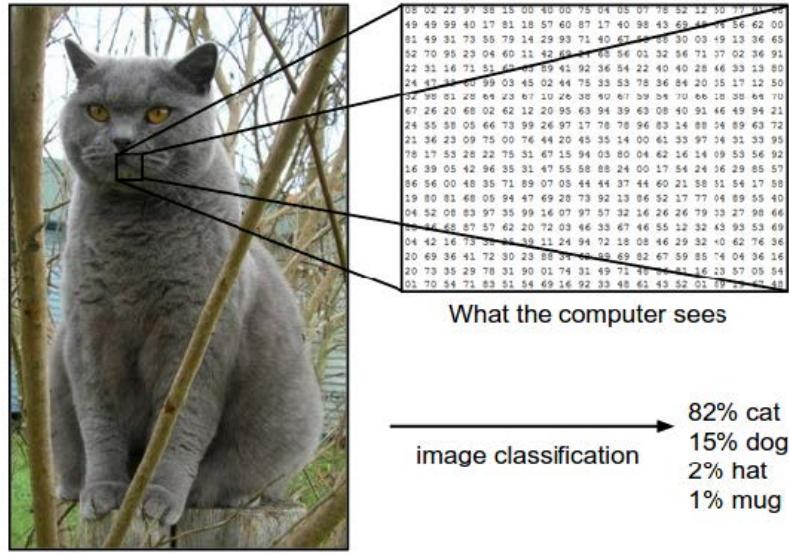


Figura 2.7: Il task della *image classification* consiste in questo caso nel calcolare una distribuzione di probabilità su quattro etichette (gatto, cane, cappello, tazza) per un’immagine digitale.

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 ritaggiano le eventuali pinne presenti nell’immagine, sfruttando una forma di conoscenza di alto livello direttamente disponibile nella rappresentazione delle immagini: il colore⁵;
- 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.

⁵L’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)

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[5], 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. 2.7)

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 [6]. 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. 2.3.2, 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

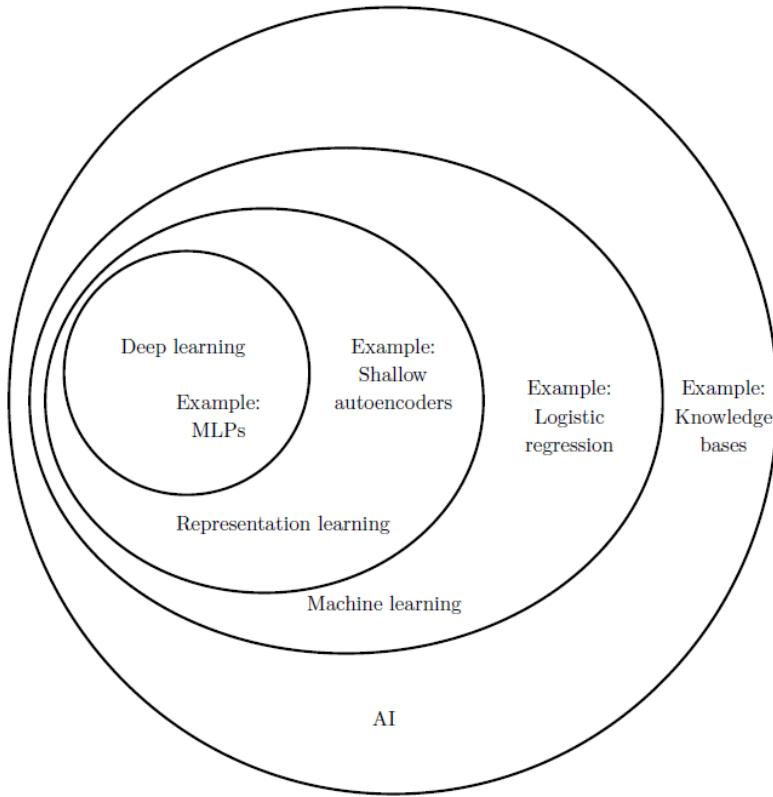


Figura 2.8: Un diagramma di Venn che mostra come il deep learning sia un tipo di representation learning, che a sua volta è un tipo di machine learning.

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 .⁶

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.

⁶Teoricamente 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.

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.⁷

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. 2.7) 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).

⁷Il 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").

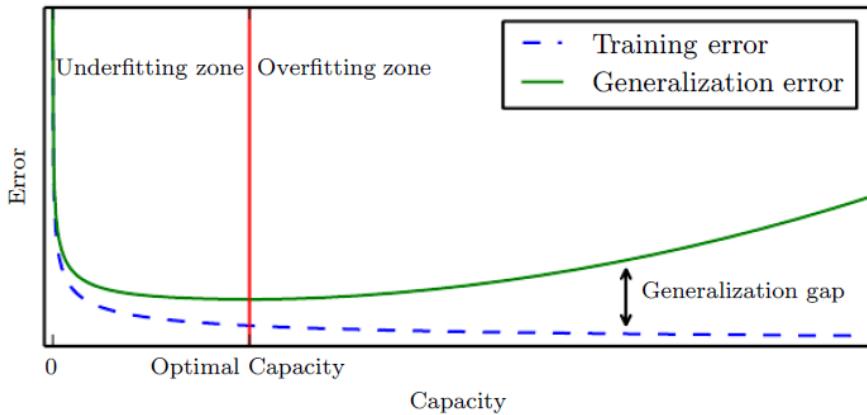


Figura 2.9: Training error e test error (asse y) al variare della capacità del modello (asse x)

2.5 Classificazione

In un problema di classificazione l'insieme delle classi Y è discreto e finito. Sulla base del numero di classi $|Y|$ si distinguono due tipi di classificazione: binaria ($|Y| = 2$) o multclasse ($|Y| > 2$).

TODO magari lo inserisco direttamente nel capitolo sulle reti neurali.

2.6 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

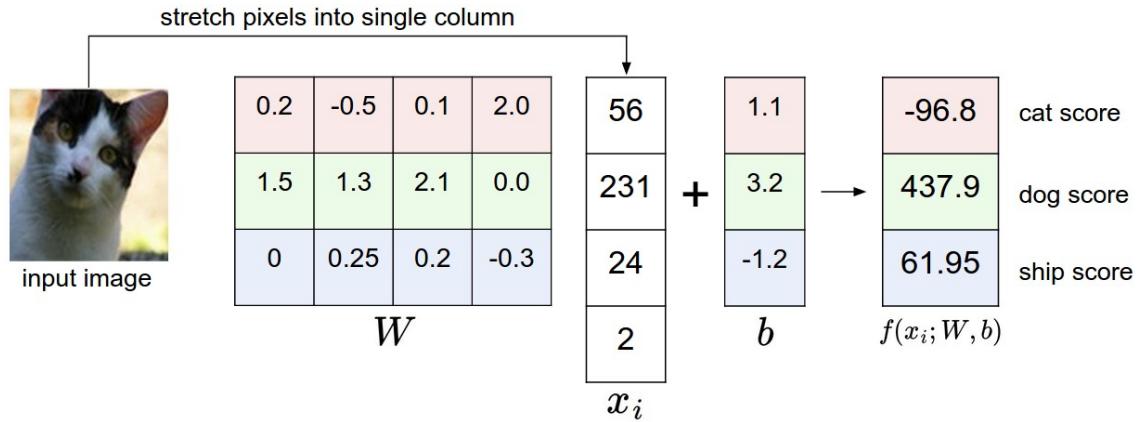


Figura 2.10: Mappatura di un’immagine ai punteggi di ogni classe mediante un classificatore lineare. Si noti che i pesi di \mathbf{W} non costituiscono un buon set di parametri: il punteggio assegnato alla classe "cane" (sbagliata) è alto e quello totalizzato dalla classe "gatto" (corretta) è basso.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

2×2 e composta da un unico canale di colore ($c = 1$) (quindi \mathbf{x} , scritta come vettore colonna, è 4×1).

Come si vedrà nel par. 2.7.1, 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} . 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 trasstrarle verticalmente).

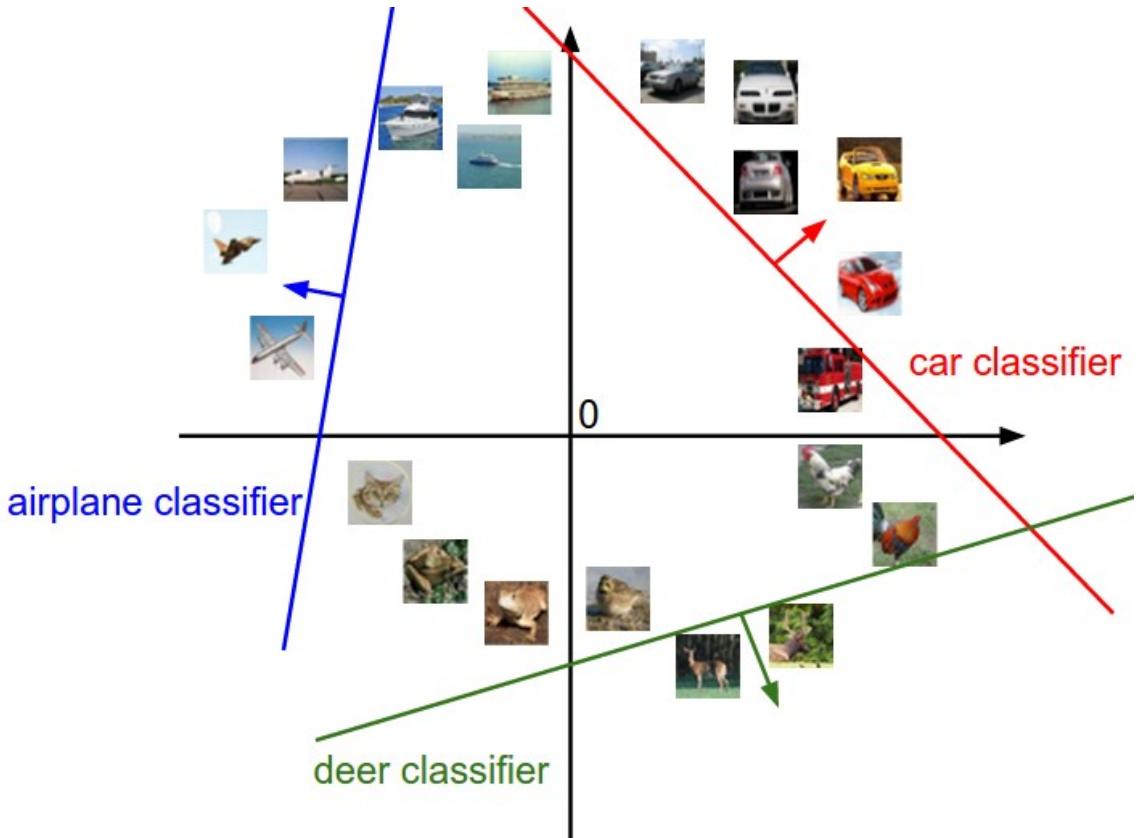


Figura 2.11: Visualizzazione di tre righe di un classificatore lineare, una per ciascuna delle classi "aereo", "auto", "cervo".

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 classificatore⁸.

Ad esempio, analizziamo il dataset *CIFAR-10* [7]. Esso contiene immagini 32×32 ciascuna appartenente ad una di 10 classi. Visualizzando⁹ i pesi (e quindi i 10 templates) di un classificatore lineare addestrato su CIFAR-10 si ottengono i risultati in figura seguente:

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 clas-

⁸Si introduggeranno gli algoritmi di apprendimento (supervisionato) nel capitolo ??.

⁹TODO 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>.



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

sificare (cioè *raffrontare al template*) è una barca in mare aperto. In altre parole, 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 "cavallo" 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 "truco" matematico 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

TODO: loss functions.

TODO: valutazione delle prestazioni di una rete neurale (matrice di confusione)

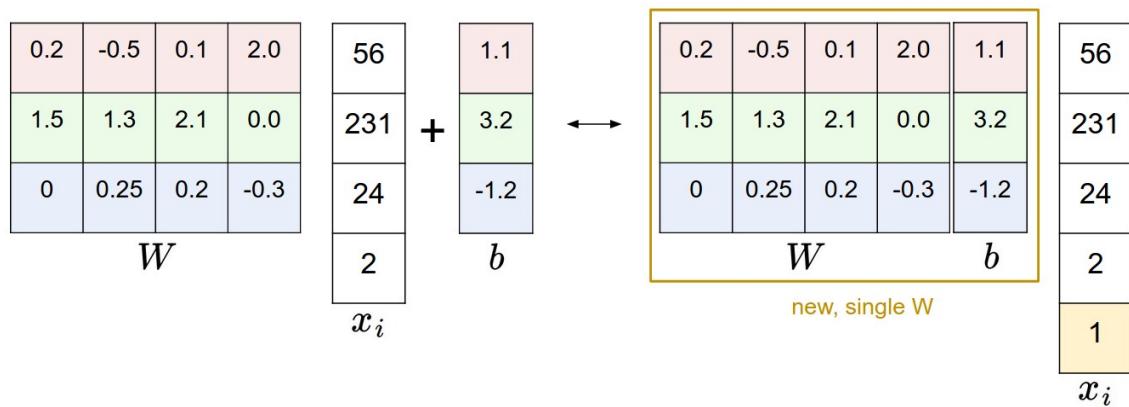


Figura 2.13: Bias trick

2.7 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

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

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:

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

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

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

Questo approccio è usato nel caso di classificazione binaria, come descritto al par. 2.5.

- 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 (par. 2.5)

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

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. 2.11, 2.12, 2.13). Altre possibilità sono la sigmoide $g(x) = \sigma(x)$ oppure la tangente iperbolica $g(x) = \tanh(x)$.

2.7.1 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"¹⁰ 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 biologico¹¹.

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. 2.6): 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.

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

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

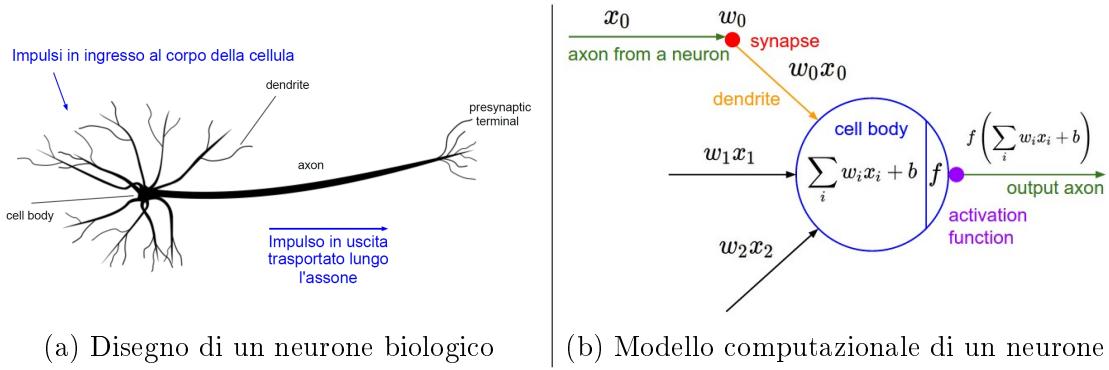


Figura 2.14: Neuroni

2.7.2 Architettura delle reti neurali artificiali

Le reti neurali sono modellate come collezioni di neuroni connessi in un grafo aciclico diretto¹², 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.

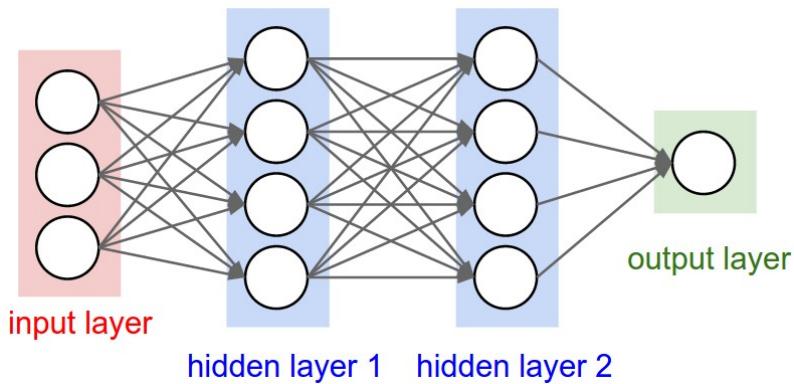


Figura 2.15: Una rete neurale con 3 strati (conventionalmente non si conta quello di input) di cui due nascosti (*hidden layers*) con 4 neuroni ciascuno e uno di output con un solo neurone.

I valori reali assunti dai neuroni sono chiamati **attivazioni** dei neuroni. Nei problemi di classificazione delle immagini (par. 2.3.2), lo strato di input contiene neuroni in numero esattamente uguale a $h \times w \times c$, 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.

¹²in 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.

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. ?? 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 [8] 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*.

2.8 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 di-

stribuzione di probabilità relativa alle categorie di oggetti classificabili.

L'architettura di una CNN può comporsi di strati di tipologie diverse¹³

- Input: generalmente un'immagine di dimensioni contenute (es. 224x224). (par. 2.8.1)
- *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. 2.8.2). 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. 2.8.3)
- *Pooling layer* (POOL): viene eseguito un sottocampionamento dell'immagine attraverso una statistica riassuntiva delle regioni. (par. 2.8.4)
- *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. 2.8.5)
- *Softmax layer*: applica la funzione di attivazione softmax alla fine della rete, per calcolare la distribuzione di probabilità delle classi da predire. (par. 2.8.6)

L'architettura di una CNN può prevedere una concatenazione di layer in serie (si parla in questo caso di *series networks*, es. AlexNet par. 2.11) ma anche una serie di modifiche che portano certi layer a funzionare in parallelo tra loro (ad es. il modulo *inception* di GoogLeNet, par. 2.12.2) o che comunque alterano il flusso lineare dell'informazione attraverso la rete mediante diramazioni (ad es. le *skip connections* in ResNet, par. 2.13.2).

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.

¹³Va 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>

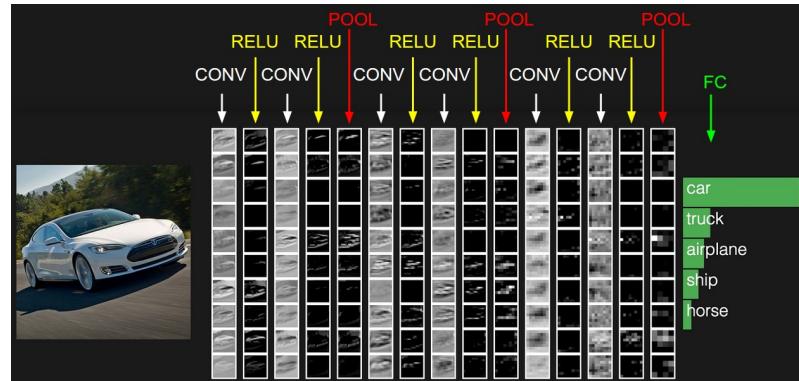


Figura 2.16: Le attivazioni di un esempio di architettura ConvNet. Il volume iniziale memorizza i pixel dell'immagine in input (a sinistra) e l'ultimo volume memorizza la probabilità per ogni classe (a destra). Ogni volume di attivazioni lungo il percorso di elaborazione è mostrato come una colonna, in cui ogni immagine è una mappa bidimensionale di attivazione (vd. par. 2.8.2).

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.

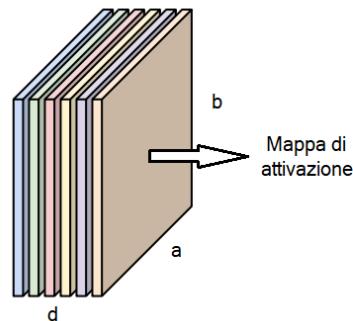


Figura 2.17: Volume di attivazioni $a \times b \times d$

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. 2.11) 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 (es. GoogLeNet, par. 2.12, ResNet-18 e 50, par. 2.13) 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.

2.8.1 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'**

2.8.2 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**:

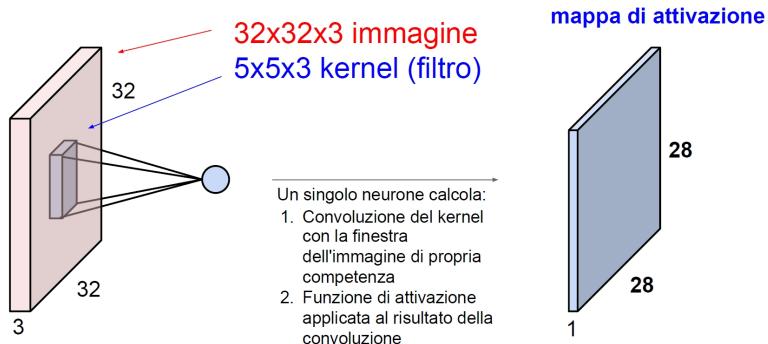


Figura 2.18: Esempio di computazione di un neurone attraverso un layer di convoluzione e di attivazione. Utilizzando una convoluzione valida con stride pari a 1 e senza input padding, la convoluzione di un'immagine $32 \times 32 \times 3$ con un filtro $5 \times 5 \times 3$ produce una matrice $28 \times 28 \times 1$, cui viene applicata la funzione d'attivazione scelta per ottenere una activation map. Tutti i 28×28 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

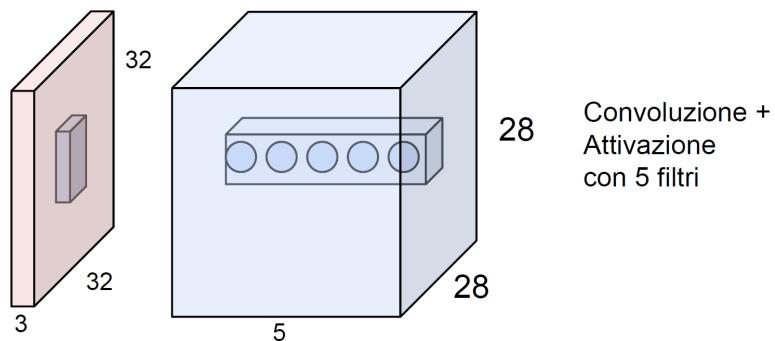


Figura 2.19: I 5 neuroni mostrati applicano un diverso filtro (un kernel con diversi parametri) alla stessa regione spaziale dell'immagine in input, in maniera indipendente l'uno dall'altro.

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)

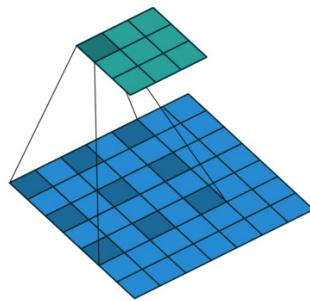


Figura 2.20: Dilation di fattore $L=2$

- **'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

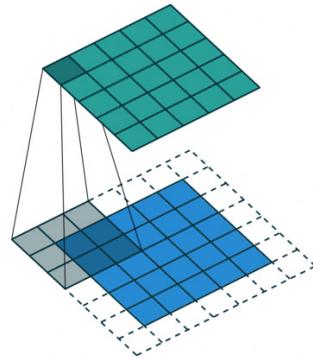
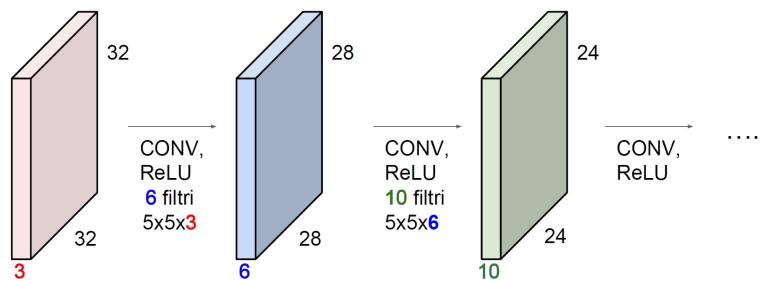

 Figura 2.21: Padding con $t=b=l=r=1$


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

2.8.3 Activation layer

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

2.8.4 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. 4×4) 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 2×2 (o anche 3×3) 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 ($\text{regione } 2 \times 2$). La

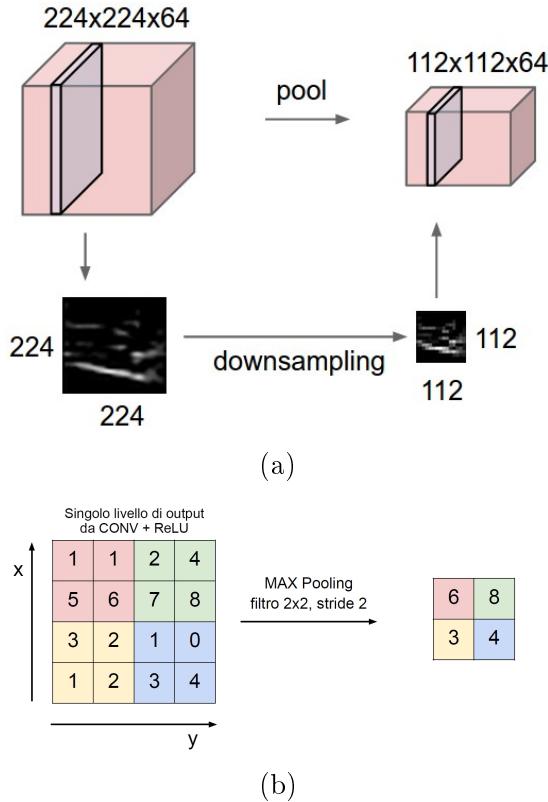


Figura 2.23: Operazione di pooling: (a) effetto di downsampling; (b) calcolo del max pooling

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'`.

2.8.5 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.

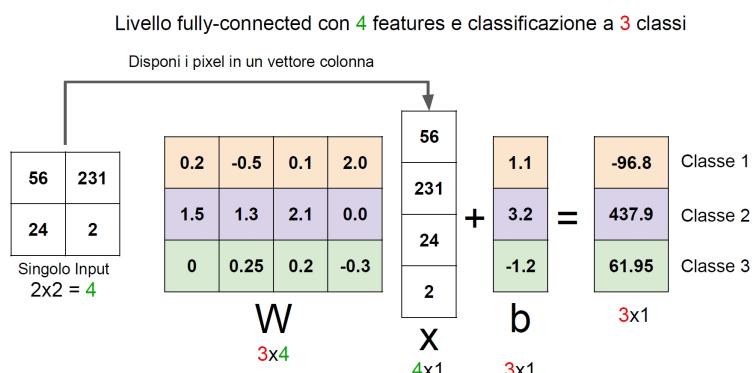


Figura 2.24: Esempio numerico di livello fully connected con un singolo input

2.8.6 Softmax layer

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

2.9 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.

- 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 aumentare un dataset con un oggetto `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 rete¹⁴, in modo che ciascuna epoch 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`.

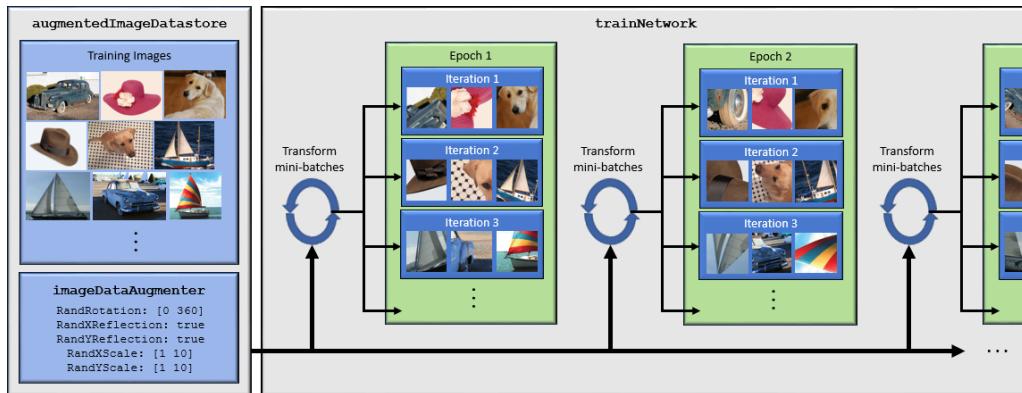


Figura 2.25: Processo di addestramento (mediante *stochastic gradient descent* con dimensione del mini-batch = 3) il cui training set è definito con un oggetto `augmentedImage datastore`

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

```
auimds = 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).

¹⁴Il 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 $[a, b]$

2.10 Addestrare una rete neurale

Per poter essere impiegata per risolvere un certo task, una rete neurale deve essere 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 permettono 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 di una 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. 2.10.2). 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. 2.10.3), che calcola il gradiente della funzione costo rispetto ad ogni parametro con l'algoritmo di backpropagation (par. 2.10.4).

2.10.1 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:

- estrazione di un valore casuale della variabile aleatoria $\varepsilon\mathcal{X}$, dove ε 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 variabili descritte al punto precedente

2.10.2 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 la "gravità" 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 noi utilizzeremo 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 λ (iperparametro) è grande. $\lambda R(\mathcal{W})$

È evidente che addestrare la rete significa allora risolvere un problema di ottimizzazione, che consiste nel trovare 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 il training set è grande. 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. 2.10.3 e fig. 2.27b).

2.10.3 Stochastic gradient descent

Si può immaginare il grafico della funzione L come una "vallata multidimensionale", differenziabile ovunque¹⁵ 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 il (un) 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 una **epoch** (epoca), cioè un ciclo di addestramento sull'intero dataset, sia in realtà diviso in $\frac{|X|}{\text{mini-batch}}$ iterazioni per epoca, in ognuna 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

¹⁵ poiché composizione di funzioni differenziabili

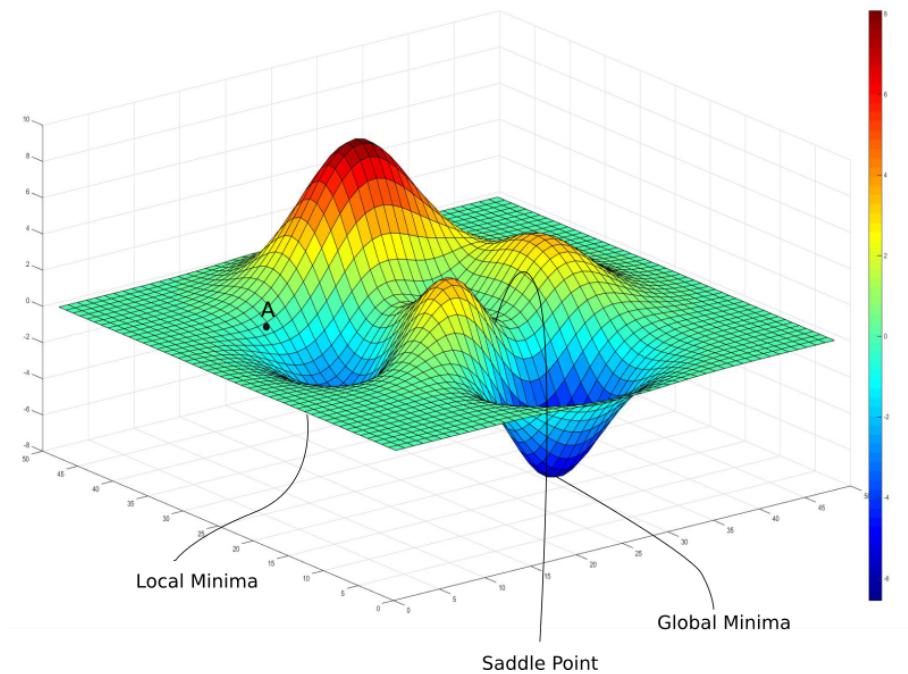


Figura 2.26: Grafico di una loss function con due sole variabili

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_l** = **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 ??.

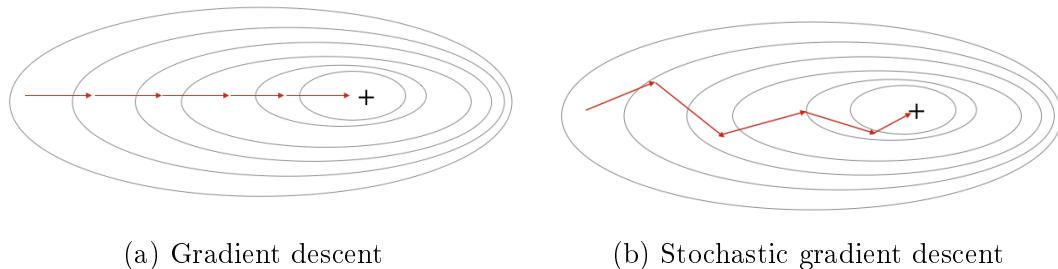


Figura 2.27

Alcune importanti osservazioni:

-
- La discesa del gradiente converge verso uno dei punti di minimo globale di L , non necessariamente in un punto di massimo 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 epoch (ad esempio si dimezza il learning rate ogni 5 epoch)
 - 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 epoch 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.

Questo metodo introduce un ulteriore iperparametro μ (compreso tra 0 e 1) per controllare l'effetto del momento, ovvero l'importanza dell'iterazione passata: più è grande μ 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

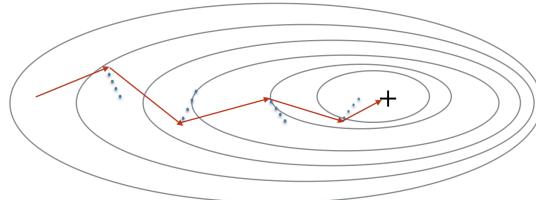


Figura 2.28: Stochastic gradient descent with momentum

2.10.4 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 [3].

2.11 AlexNet

AlexNet è una CNN creata tra il 2011 e il 2012 da Alex Krizhevsky, in collaborazione con Ilya Sutskever e Geoffrey Hinton [9]. La vittoria di AlexNet nella *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* (par. ??)¹⁶, 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 [10].

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.

2.11.1 Architettura di AlexNet

L'architettura di AlexNet è riportata schematicamente nella figura 2.29 e con maggiore dettaglio in tabella 2.1. La rete accetta in input immagini 227×227 . 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.

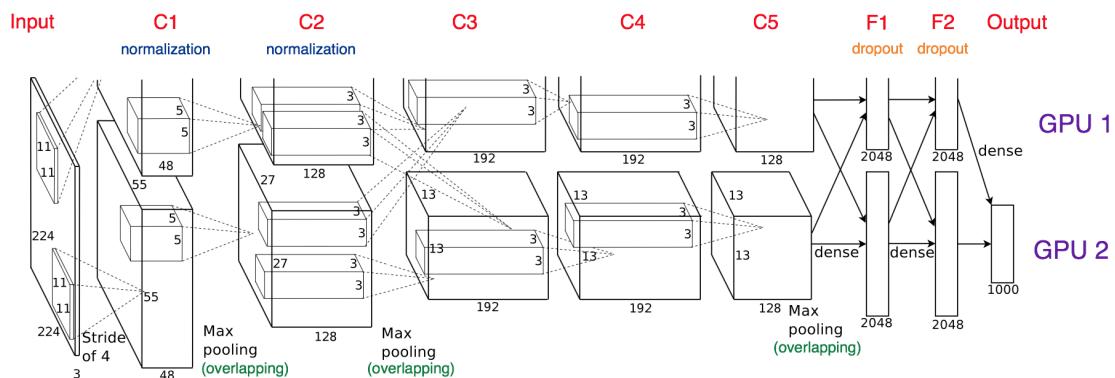


Figura 2.29: Architettura originale di AlexNet [9]

Come si evince dalla figura 2.29, 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

¹⁶<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*¹⁷).

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

2.11.2 Funzione di attivazione ReLU

Dopo ogni strato parametrizzato, i valori delle attivazioni sono passati alla funzione attivatrice "rettificatore": $f(x) = x^+ = \max(0, x)$ [11]. 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 $f(x) = \tanh(x)$.

2.11.3 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, α, β 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.

2.11.4 Overlapping Max Pooling

La funzione di max pooling in AlexNet è stata caratterizzata dalla scelta di una dimensione del filtro di pooling 3×3 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.

¹⁷La scelta di questo pattern di connettività fra le due GPU parallele è il risultato di un problema di cross-validation.

2.11.5 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 256×256 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 256×256 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 224×224 ¹⁸ dalle immagini
- Riflessione orizzontale ("a specchio") delle immagini (50% di probabilità)
- Somma di un'immagine e le sue componenti principali (PCA)¹⁹

2.11.6 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* [13]. 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. 2.30 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.

¹⁸Le immagini vengono portate a 227×227 , cioè la risoluzione di input, attraverso uno *zero-padding* (3 pixel aggiuntivi ai bordi)

¹⁹L'*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 [12] per approfondimenti sulla PCA.

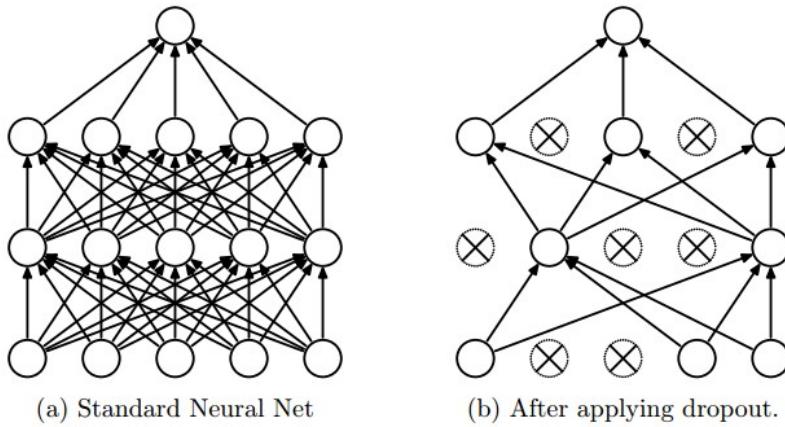


Figura 2.30: Figura tratta dal paper originale [13]. Durante l’addestramento, il dropout può essere immaginato come una “sotto-rete” neurale, che è soggetta ad addestramento al posto della rete originale.

2.11.7 Addestramento di AlexNet

Nella sua forma originale, AlexNet fu addestrato 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 [9].

2.11. ALEXNET

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

2.12 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 [14] è avvenuta poco dopo la schiacciatrice vittoria riportata da GoogLeNet nella *ILSVRC 2014*²⁰, 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. 2.12.2). In particolare, se confrontata ad AlexNet (par. 2.11), 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 1×1 (par. 2.12.1).
- 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 1×1 , all'operazione di *global average pooling* (par. 2.12.4) e all'usuale pratica della *image augmentation* (par. 2.12.5).
- Le parti salienti di un'immagine possono avere delle dimensioni estremamente variabili all'interno di essa. Si guardi ad esempio la figura seguente:

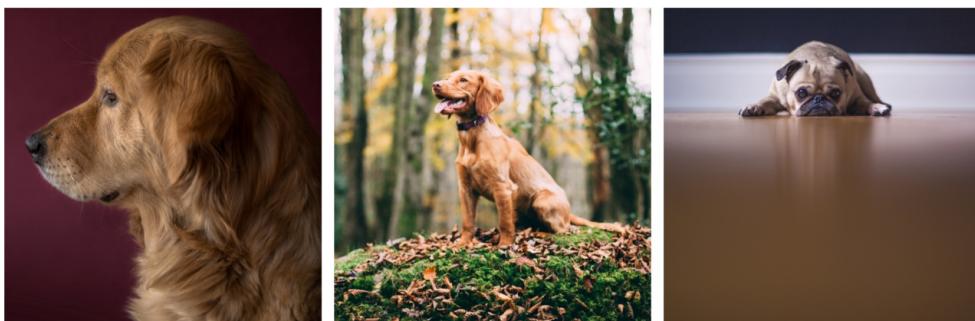


Figura 2.31: Tre immagini di cani. L'area occupata da ciascun cane è differente e via via più piccola in ogni immagine

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

²⁰<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. 2.12.2).

- 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. ??), 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. 2.12.3).

2.12.1 Convoluzione 1×1

Mutuando un'idea precedentemente esposta in [15], GoogLeNet introduce l'operazione di convoluzione 1×1 (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. 2.12.2) - più "larghe". La fig. 2.32 mostra schematicamente l'applicazione di un'operazione di convoluzione 1×1 .

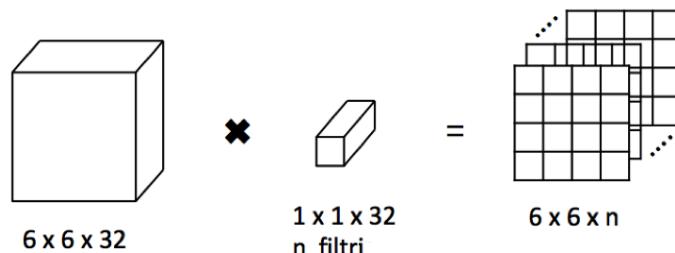


Figura 2.32: Convoluzione 1×1 applicata ad un volume di attivazioni $6 \times 6 \times 32$. La profondità del volume di output dipende dal n. di filtri 1×1 adoperati. In questo caso si ottiene una riduzione della profondità per $n < 32$.

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

2.12.2 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. 2.33 è mostrato il modulo *inception*, innestato più volte nell'architettura di GoogLeNet (par. 2.12.6)

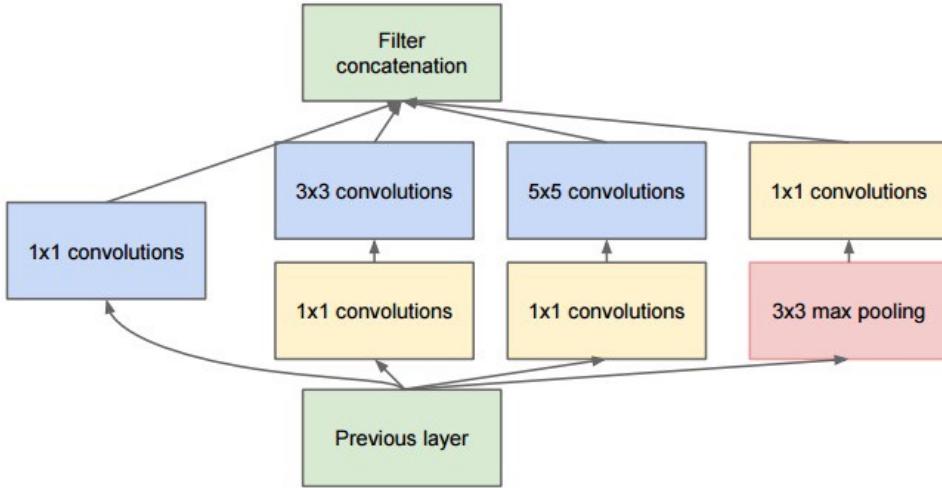


Figura 2.33: Modulo *inception*

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

2.12.3 Classificatori ausiliari

L'architettura di GoogLeNet prevede, per la sola fase di addestramento, delle "diramazioni", posizionate circa a metà della rete, evidenti in fig. 2.12.6. Questi rami della rete ospitano dei classificatori ausiliari che consistono di

- Average Pooling 5×5 (Stride 3)
- Convoluzione 1×1 (128 filtri)
- ReLU layer
- 1024 Fully Connected layer
- 1000 Fully Connected layer
- Dropout layer (70% di probabilità di dropout di ciascun output; vd. par. 2.11.6)
- Softmax layer

Questa particolare architettura è stata ideata per attenuare il problema della scomparsa del gradiente (*vanishing gradient problem*, par. ??), 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).

2.12.4 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. 2.11, che presenta tre fully connected layer finali). Mutuando un'idea esposta in [15], 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 7×7 (le dimensioni di base e altezza del volume di output precedente) e restituisce una superficie 1×1 , 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. 2.34 mostra un confronto tra il funzionamento di un FC layer e un GAP (Global Average Pooling) layer.

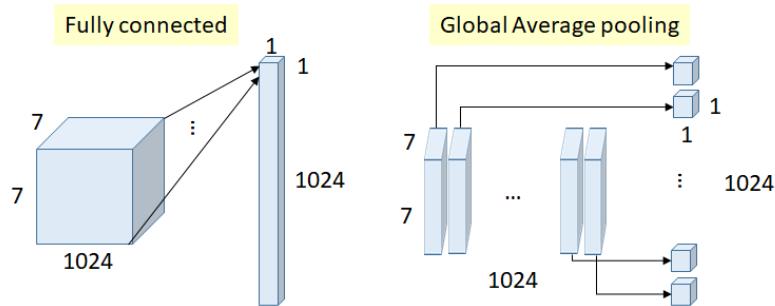


Figura 2.34: Confronto fra un FC layer e un GAP layer

2.12.5 Data Augmentation

Per ridurre ulteriormente l'*overfitting* al training set di ImageNet, GoogLeNet fa uso di una particolare tecnica di *data augmentation* (par. 2.9). 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 [16].

2.12.6 Architettura di GoogLeNet

L’architettura di GoogLeNet è riportata in forma tabellare in tab. 2.35 e in forma grafica in fig. 2.36

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	$7\times 7/2$	$112\times 112\times 64$	1							2.7K	34M
max pool	$3\times 3/2$	$56\times 56\times 64$	0								
convolution	$3\times 3/1$	$56\times 56\times 192$	2		64	192				112K	360M
max pool	$3\times 3/2$	$28\times 28\times 192$	0								
inception (3a)		$28\times 28\times 256$	2	64	96	128	16	32	32	159K	128M
inception (3b)		$28\times 28\times 480$	2	128	128	192	32	96	64	380K	304M
max pool	$3\times 3/2$	$14\times 14\times 480$	0								
inception (4a)		$14\times 14\times 512$	2	192	96	208	16	48	64	364K	73M
inception (4b)		$14\times 14\times 512$	2	160	112	224	24	64	64	437K	88M
inception (4c)		$14\times 14\times 512$	2	128	128	256	24	64	64	463K	100M
inception (4d)		$14\times 14\times 528$	2	112	144	288	32	64	64	580K	119M
inception (4e)		$14\times 14\times 832$	2	256	160	320	32	128	128	840K	170M
max pool	$3\times 3/2$	$7\times 7\times 832$	0								
inception (5a)		$7\times 7\times 832$	2	256	160	320	32	128	128	1072K	54M
inception (5b)		$7\times 7\times 1024$	2	384	192	384	48	128	128	1388K	71M
avg pool	$7\times 7/1$	$1\times 1\times 1024$	0								
dropout (40%)		$1\times 1\times 1024$	0								
linear		$1\times 1\times 1000$	1							1000K	1M
softmax		$1\times 1\times 1000$	0								

Figura 2.35: Architettura di GoogLeNet

La rete accetta in input immagini 224×224 . 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. 2.12.3, che le ramificazioni che ospitano i due classificatori ausiliari esistono solo in fase di addestramento, e vengono eliminati dopo di esso.

2.12.7 Addestramento di GoogLeNet

GoogLeNet fu addestrato sul dataset di ImageNet (par. ??) su una macchina che usava solamente le sue CPU per effettuare calcoli²¹, usando come algoritmo di ottimizzazione la discesa stocastica del gradiente con momento = 0.9, *learning rate*²² con *annealing* del 4% ogni 8 epoche. Dettagli più specifici sulla fase di addestramento di GoogLeNet possono essere trovati nel paper originale [14].

²¹Tuttavia 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).

²²Per 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

2.12. GOOGLENET

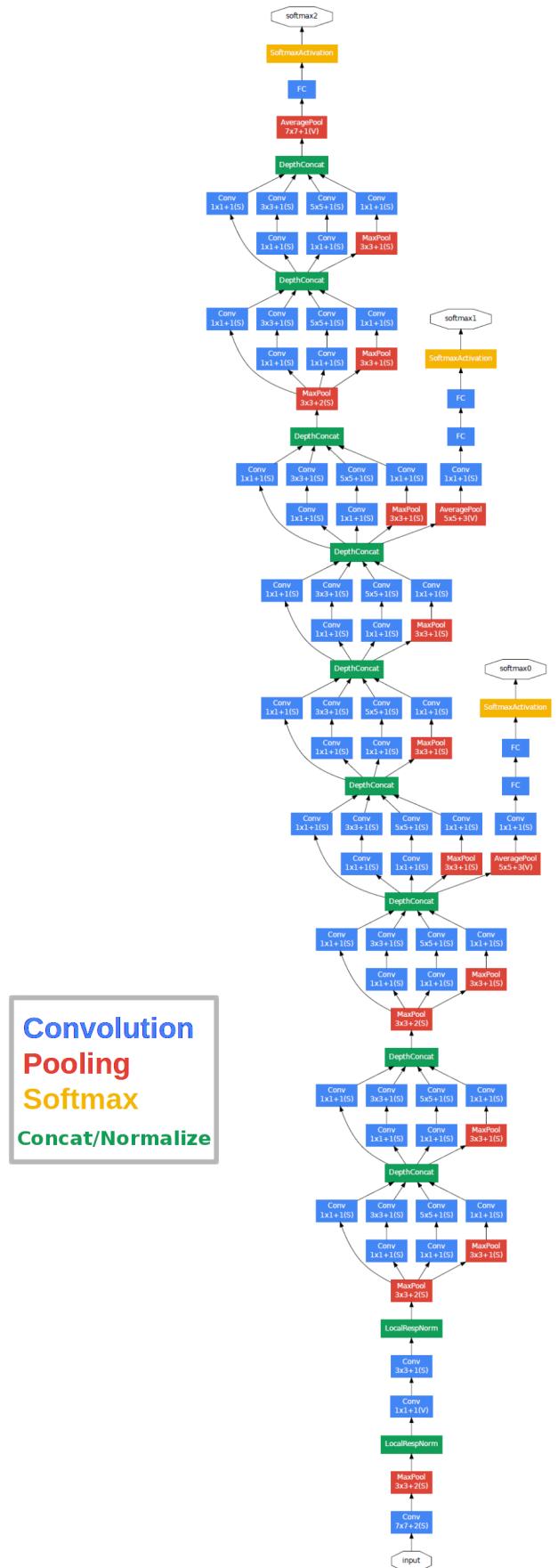


Figura 2.36: Architettura di GoogLeNet

2.13 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 [17]. Presentate nell'edizione del 2015 della *ILSVRC* 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 ([14],[18]) 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. ??). 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. ??) e il presentarsi di nuovi (un problema di degradazione esposto per la prima volta in [19], 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. 2.13.1) introdotta dal team dei creatori di GoogLeNet [20].
- 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. 2.37.

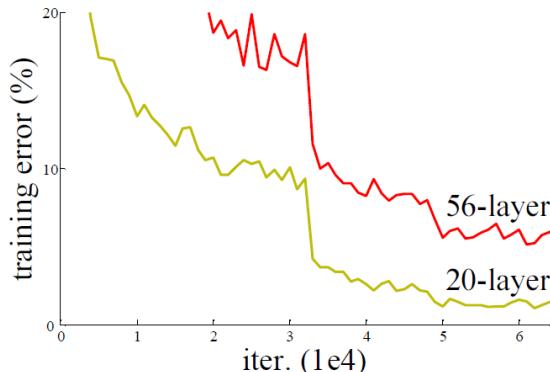


Figura 2.37: Problema di degradazione della *training accuracy* su CNN "standard" da 20 e 56 layer rispettivamente. La rete più profonda ha un *training error* più alto. [17] per più dettagli.

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. 2.13.2)

2.13.1 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 [20].

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.

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

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

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

I parametri γ e β , 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 [21] 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 [20], 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).

2.13.2 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. 2.38 è mostrato uno schema generale dell'oggetto in esame.

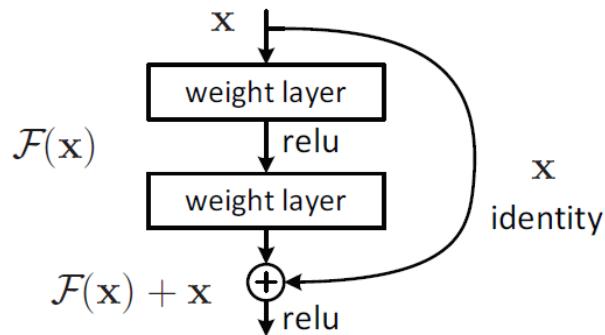


Figura 2.38: *Residual block* con *shortcut identity mapping*. Si noti che i particolari layer di questo blocco sono arbitrari.

Invece di sperare che un dato blocco di layer non lineari contigui impari ad approssimare una certa funzione $\mathcal{H}(x)$ desiderata, si decide di creare una "scorciatoia" (propriamente *shortcut connection*) che connette l'input x e l'output del blocco (vd. fig. 2.38); in questo modo il blocco deve imparare ad approssimare non più la funzione desiderata $\mathcal{H}(x)$ ma una funzione residuale $\mathcal{F}(x) := \mathcal{H}(x) - x$. In definitiva il blocco addestrato darà in output la funzione $\mathcal{F}(x) + x$.

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 ($\mathcal{F}(x) = x$). 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 [17] hanno mostrato tuttavia che non conosciamo nessun algoritmo di ottimizzazione che permetta almeno di eguagliare il *training error* della prima rete addestrando la 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. [19]), 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*.

2.13.3 Architettura di ResNet-18

ResNet-18 è una rete neurale convoluzionale profonda addestrata sul dataset ImageNet (par. ??). La rete accetta in input immagini 224×224 ed è composta da 18 layer parametrizzati, di cui un layer convoluzionale iniziale con filtro 7×7 (seguito da batch normalization, ReLU e max pooling), altri 16 layer convoluzionali 3×3 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 2.39

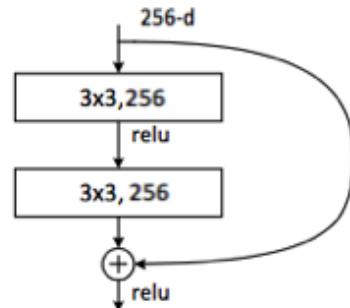


Figura 2.39: Uno degli otto *residual blocks* di ResNet-18

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

- Convoluzione 3×3 ²³
- Batch Normalization
- ReLU
- Convoluzione 3×3
- Batch Normalization

²³il numero di filtri di convoluzione, e quindi la profondità del volume di output, è riportata per ciascun *residual block* in fig. 2.42 e in fig. 2.41

Inoltre, il ramo che realizza la *shortcut connection* si compone eventualmente di una convoluzione 1×1 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. 2.43 con un arco tratteggiato.

L'architettura di ResNet-18 è riportata schematicamente nella figura 2.42 e in forma tabellare, in confronto con ResNet-50, in fig. 2.41 nel prossimo sottoparagrafo.

2.13.4 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 224×224 ; essa è composta da 50 layer parametrizzati, di cui un layer convoluzionale iniziale con filtro 7×7 (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 2.40

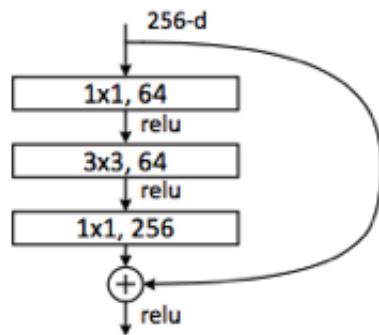


Figura 2.40: Uno dei sedici *residual blocks* di ResNet-50

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

- Convoluzione 1×1
- Batch Normalization
- ReLU
- Convoluzione 3×3
- Batch Normalization
- ReLU
- Convoluzione 1×1
- Batch Normalization

Inoltre, il ramo che realizza la *shortcut connection* si compone eventualmente di una convoluzione 1×1 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. 2.43 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 3×3 sono computazionalmente costose, pertanto in reti molto profonde non si possono prevedere tutti *residual blocks* ciascuno operante due convoluzioni 3×3 . Si decide allora di modificare il *residual block* usato: si decide di operare un'unica convoluzione 3×3 per blocco, preceduta e seguita da una convoluzione 1×1 (par. ??) per rispettivamente ridurre e ripristinare la profondità del volume di attivazioni su cui la convoluzione 3×3 lavora, abbassando così il costo computazionale associato all'addestramento di ciascun *residual block*.

L'architettura di ResNet-50 è riportata schematicamente nella figura 2.43 e in forma tabellare, in confronto con quella di ResNet-18, in fig. 2.41 seguente.

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

Figura 2.41: Le architetture di ResNet-18 e ResNet-50 a confronto

2.13.5 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) 224×224 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. 2.11.5). Il ritaglio è eventualmente capovolto orizzontalmente (50% di probabilità)

2.13.6 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 6×10^5 . I pesi sono inizializzati secondo il metodo descritto in [22], 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 [17].

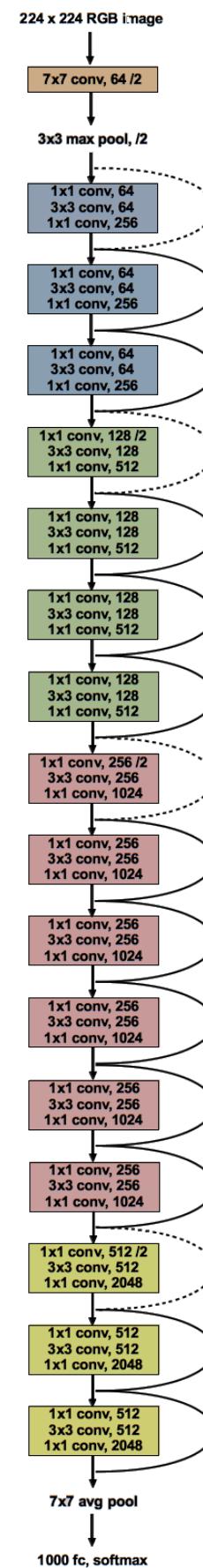
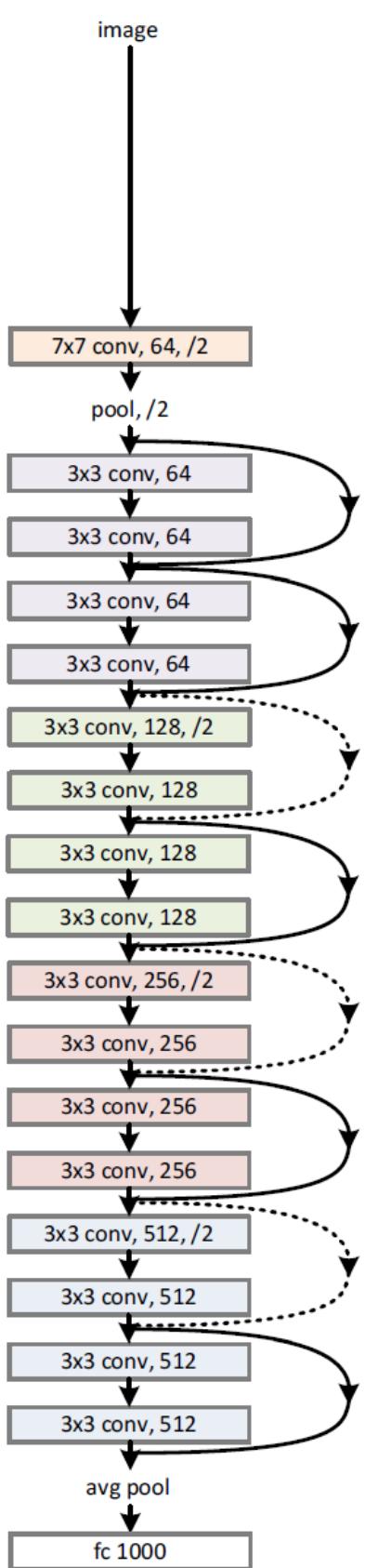


Figura 2.42: Architettura di ResNet-18

Figura 2.43: 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. ??). Il dataset contiene immagini acquisite in un'area di 14000 km^2 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. ??). 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.

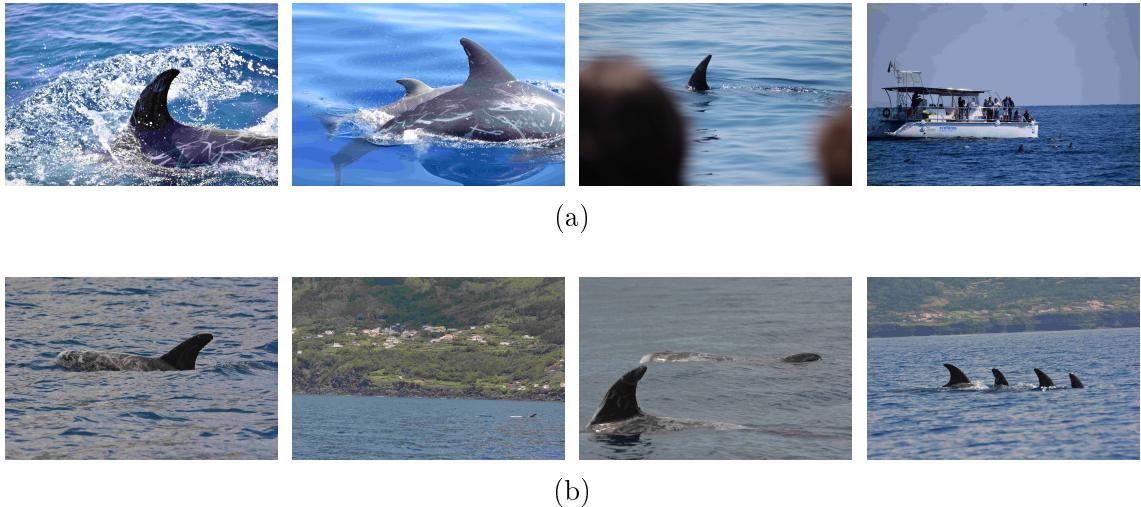


Figura 3.1: Alcune immagini estratte dai due dataset utilizzati. (a) Dataset degli scatti di Taranto, (b) Dataset degli scatti delle Azzorre. Si noti la presenza di foto con disturbi o addirittura senza alcun valore informativo ai fini dello studio dei cetacei.

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 dall’ing. Gianvito Losapio nel suo lavoro di tesi triennale [1] sulla base di un precedente lavoro di tesi dell’ing. Flavio Forenza [23]. 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 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 (1200×800). Questa operazione di preprocessing è stata adottata per diminuire il costo computazionale delle operazioni successive.¹

Il risultato dell'operazione è visualizzato in figura 3.2



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

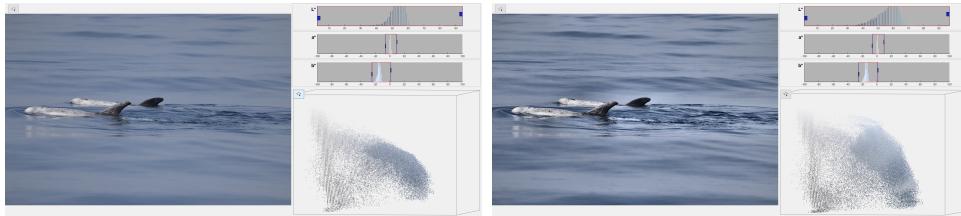
CLAHE

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

¹La risoluzione di partenza delle immagini utilizzate è stata 6000×4000 , 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 [24].

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:

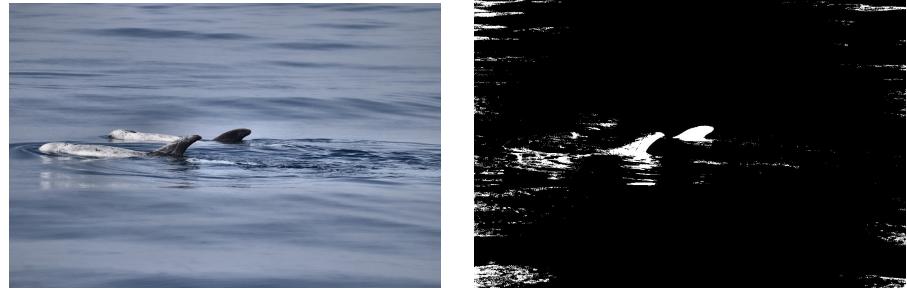
- nella regione più scura del canale L, cioè a sinistra della soglia sul canale L
- nella regione contenente il grigio del canale b, cioè a destra della soglia sul canale b

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)²

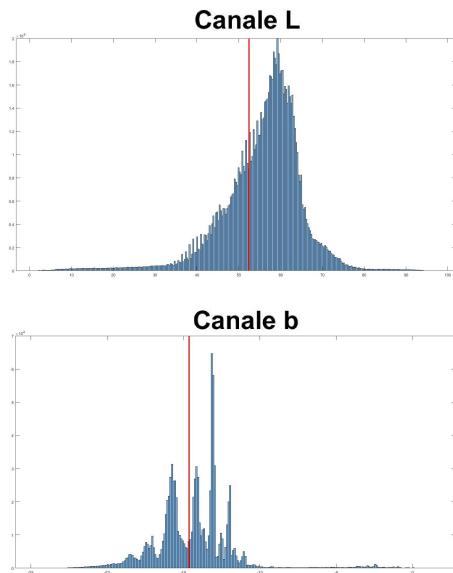
- valore della componente L minore della soglia di Otsu sul canale L
- valore della componente b maggiore della soglia di Otsu sul canale b

Il risultato dell’operazione è visualizzato in figura 3.4

²L’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.



(a) Prima della segmentazione secondo Otsu (b) Dopo la segmentazione secondo 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.

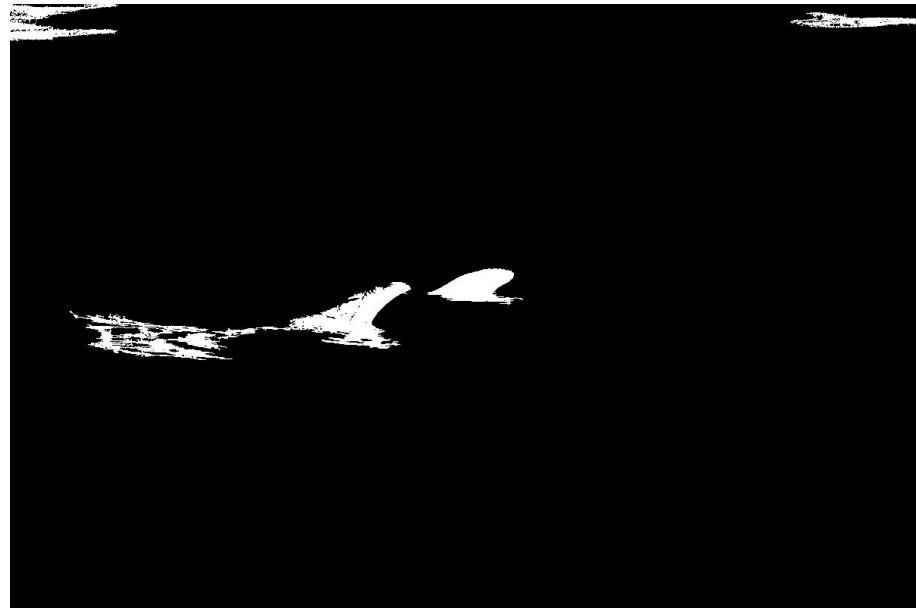
Il risultato dell'operazione è visualizzato in figura 3.5

Ritaglio adattivo

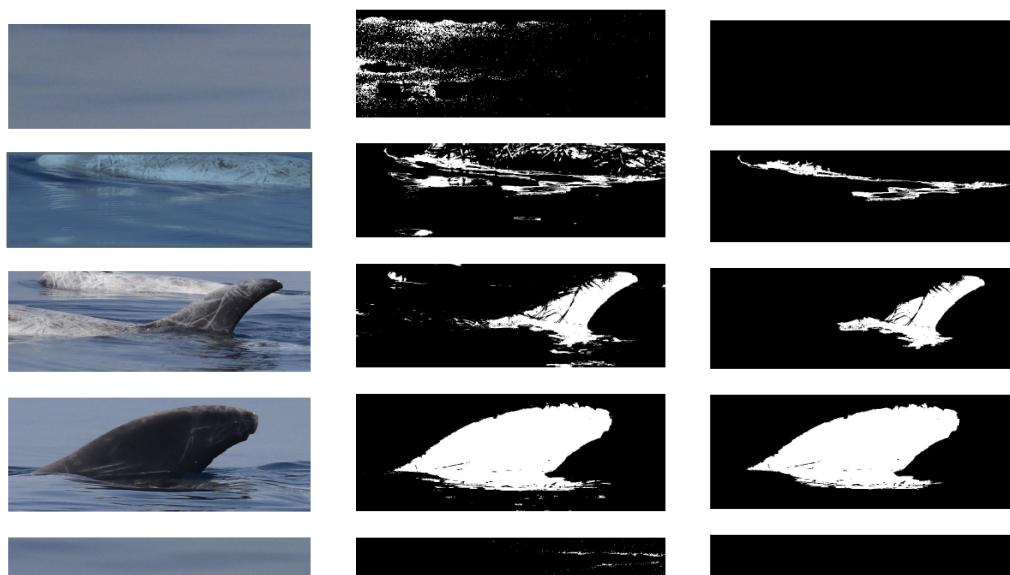
Le regioni binarie mantenute in seguito alla fase di filtraggio sono sottoposte ad un algoritmo che consente di ottenere un ritaglio preciso in corrispondenza delle pinne. Tale operazione si può definire "adattiva" nella misura in cui la regione di ritaglio è ottenuta a partire da precisi punti geometrici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

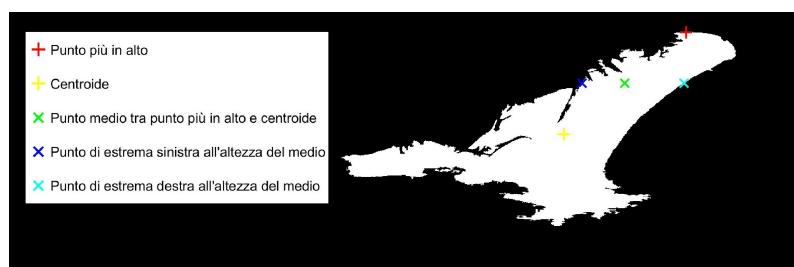
Il risultato dell'operazione è visualizzato in figura 3.6



(a) Regione binaria da sottoporre a ritaglio adattivo



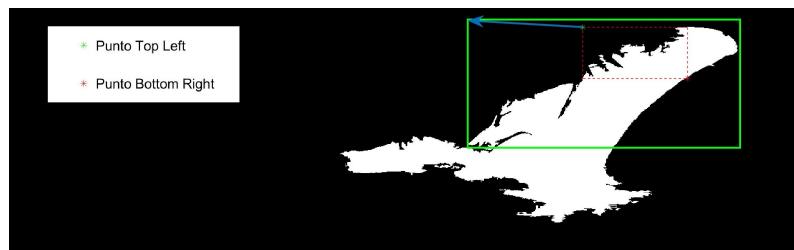
(b) Riempimento degli *holes*



(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 [25] e migliorata in [26]). 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.

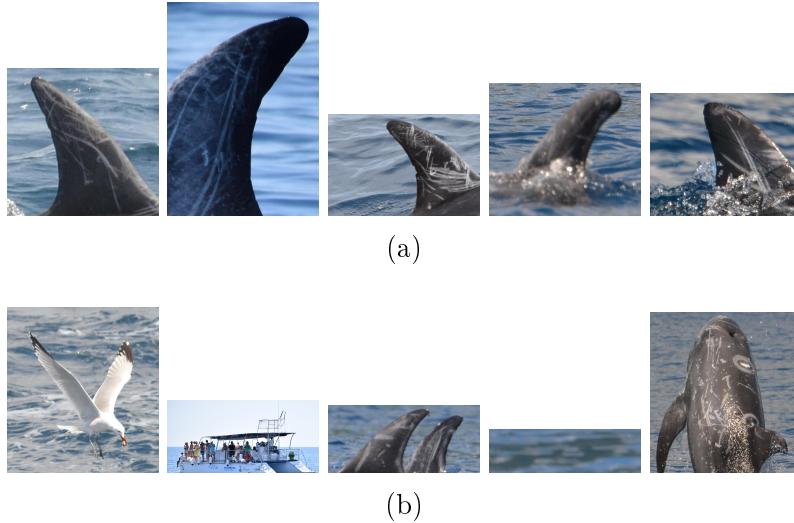


Figura 3.7: Alcuni ritagli etichettati manualmente come appartenenti alla classe (a) 'Pinna', (b) 'No Pinna'

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. ?? sono stati descritti molteplici motivazioni per le quali per risolvere un problema di classificazione (in particolare di *image captioning*) 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 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. ??). 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. 2.11)

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. 2.12)

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. 2.13)

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. 2.13)

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. ??.

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 "augmentata" 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*

Un secondo oggetto `augmentedImageDatastore` è usato per rappresentare `cropValidation`, ma in questo caso 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. ??) per classificare un'immagine scegliendo tra le mille categorie diverse del dataset. Applicare la tecnica del *transfer learning* (par. ??) 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 [27], 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. ??.

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 classificazione³; 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 letteratura⁵, 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 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

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

3.3. CLASSIFICAZIONE MEDIANTE CNN E TRANSFER LEARNING

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 nota in fig. ???. Questa mancata capacità di generalizzazione può essere conseguenza dell'eccessiva complessità della rete (n. di parametri inutilmente elevato), in

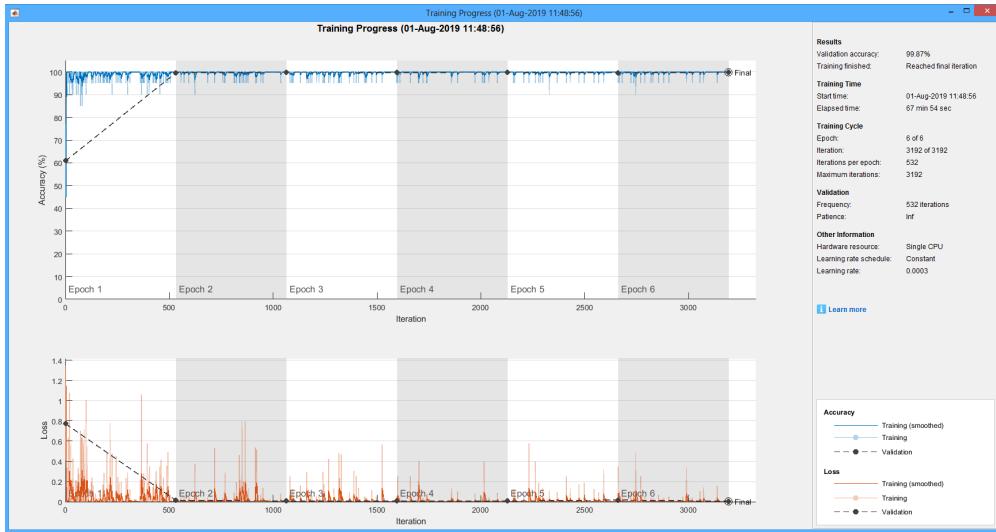


Figura 3.9: Grafico del ri-addestramento di AlexNet

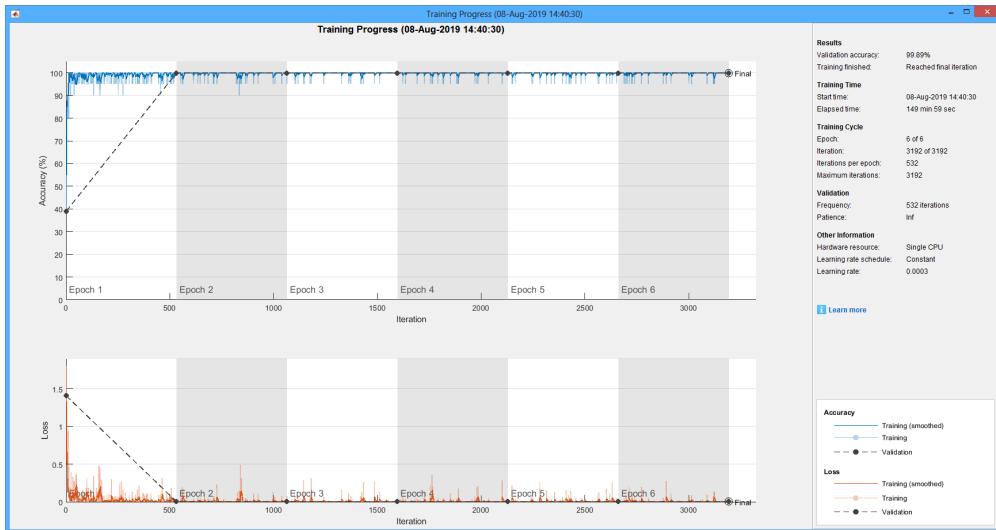


Figura 3.10: Grafico del ri-addestramento di GoogLeNet

rapporto ad un *training set* relativamente piccolo⁴.

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 '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

⁴Delle considerazioni simili sono state fatte anche nel paper originale di ResNet [17].

3.3. CLASSIFICAZIONE MEDIANTE CNN E TRANSFER LEARNING

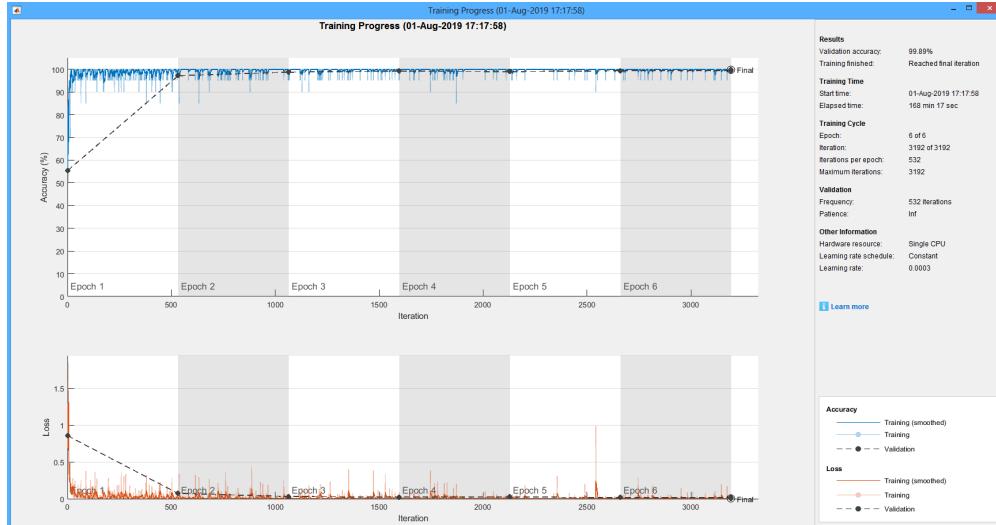


Figura 3.11: Grafico del ri-addestramento di ResNet-18

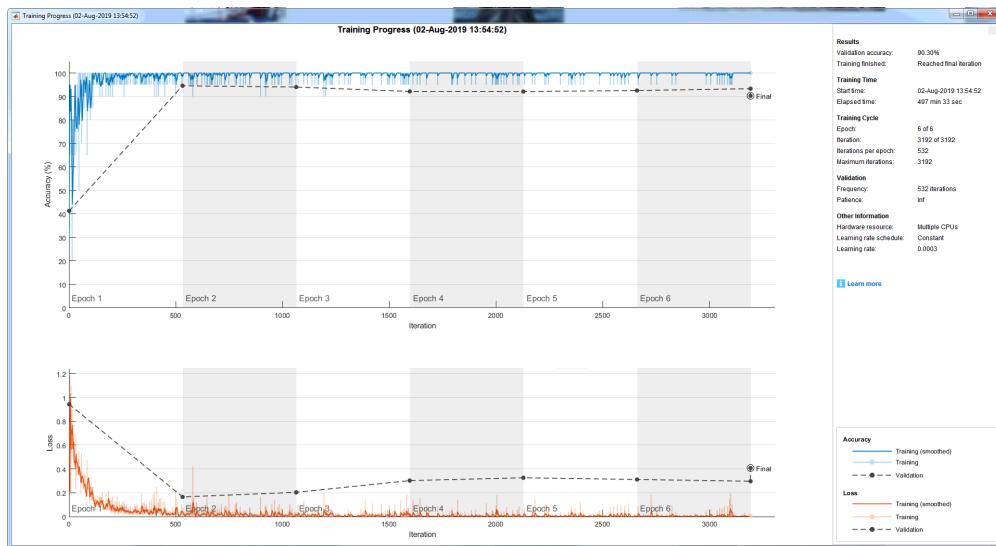


Figura 3.12: Grafico del ri-addestramento di ResNet-50. Si noti che la *training accuracy* è prossima al 100% mentre la *validation accuracy* si attesta al 90%, chiaro sintomo di *overfitting* al training set

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 [26] 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. ??). 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. ??, 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.[3][28]

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

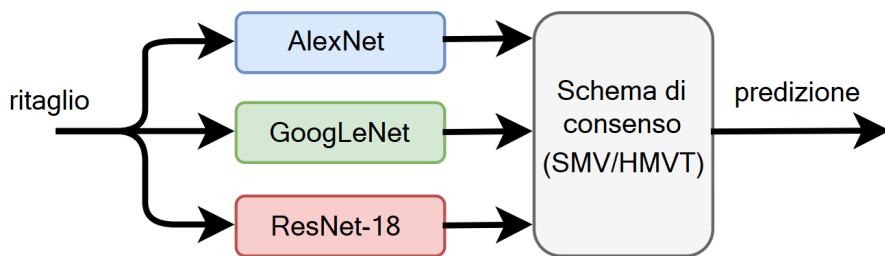


Figura 3.13: Schema del classificatore ensemble creato

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

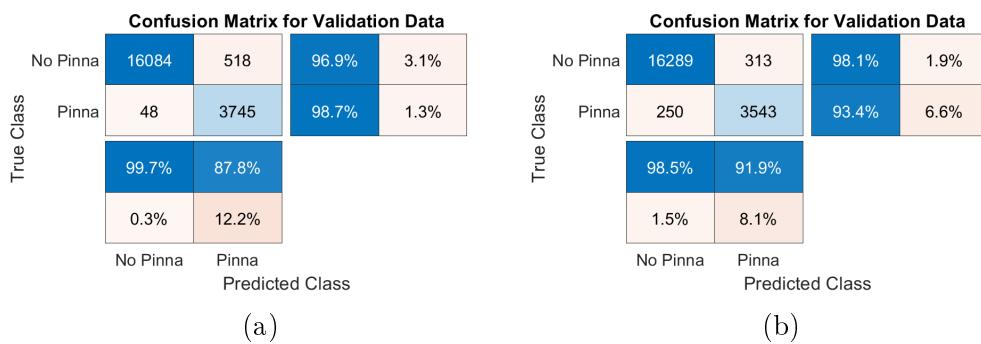


Figura 3.14: Matrici di confusione relative alle predizioni del nuovo ensemble sui ritagli delle Azzorre, con schema di consenso (a) *soft major voting*, (b) *hard major voting* con soglia.

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.

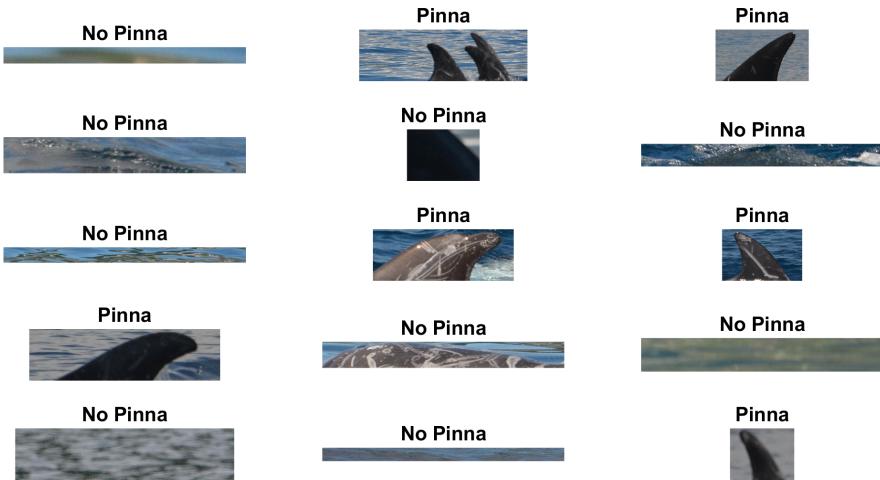


Figura 3.15: Alcune predizioni proposte dall’ensemble con schema di consenso HM-VT. Si noti ad esempio quella centrale in alto: compaiono tre pinne di cui addirittura due sovrapposte, ma il classificatore restituisce ‘Pinna’. Sebbene il classificatore sia riuscito a confermare la presenza di una pinnula, 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 precedenza⁵.

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

⁵In 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

un algoritmo di foto-identificazione delle pinne, ad esempio quello descritto in [26]. 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 (~ 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 alla predizione (parametri molto "sentiti" dall'utilizzatore finale) 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.

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 ([29], [30], [31], 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 dal nostro ensemble di reti da parte di routine che compiano appunto questo foto-riconoscimento (*SPIR* [25] e *SPIR v2*[26]) 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 sviluppi futuri.

Si può pensare di riutilizzare il classificatore mediante *transfer learning* per risolvere il task di riconoscimento delle pinne dorsali anche per 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*) [32] e squali [33].

Gli algoritmi di *photo-ID* dei cetacei a partire dalla loro pinna, come quello descritto in [26], 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) quando una pinna è 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.

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] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. URL: <http://szeliski.org/Book/>.
- [3] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [4] Andrej Karpathy. *Lectures of Stanford class CS231n: Convolutional Neural Networks for Visual Recognition*. Appunti del corso Stanford CS class C231n. 2019.
- [5] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [6] Stuart Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2009.
- [7] Alex Krizhevsky. «Learning multiple layers of features from tiny images». In: (2009).
- [8] G. Cybenko. «Approximation by superpositions of a sigmoidal function». In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314.
- [9] Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems* 25. 2012.
- [10] Md Zahangir Alom et al. «The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches». In: (2018).
- [11] Vinod Nair e Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». In: *ICML*. 2010.
- [12] Sergio Bolasco. *Analisi multidimensionale dei dati. Metodi, strategie e criteri d'interpretazione*. Carocci Editore, 1999.
- [13] Geoffrey E. Hinton et al. «Improving neural networks by preventing co-adaptation of feature detectors». In: (2012).
- [14] Christian Szegedy et al. «Going Deeper with Convolutions». In: (2014).
- [15] Min Lin, Qiang Chen e Shuicheng Yan. «Network In Network». In: (2013).

-
- [16] Andrew G. Howard. «Some Improvements on Deep Convolutional Neural Network Based Image Classification». In: (2013).
 - [17] Kaiming He et al. «Deep Residual Learning for Image Recognition». In: () .
 - [18] Karen Simonyan e Andrew Zisserman. «Very Deep Convolutional Networks for Large-Scale Image Recognition». In: () .
 - [19] Rupesh Kumar Srivastava, Klaus Greff e Jürgen Schmidhuber. «Highway Networks». In: () .
 - [20] Sergey Ioffe e Christian Szegedy. «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». In: (2015).
 - [21] Shibani Santurkar et al. «How Does Batch Normalization Help Optimization?» In: *NeurIPS* (2018).
 - [22] Kaiming He et al. «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification». In: () .
 - [23] 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.
 - [24] Nobuyuki Otsu. «A Threshold Selection Method from Gray-Level Histograms». In: *IEEE Trans. Syst. Man. Cybern.*, vol. 9, no. 1, pp. 62–66 (1979).
 - [25] 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.
 - [26] 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.
 - [27] 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).
 - [28] Lars Kai Hansen e Peter Salamon. «Neural Network Ensembles». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10 (1990).
 - [29] Mark Thomas et al. «Marine Mammal Species Classification using Convolutional Neural Networks and a Novel Acoustic Representation». In: (2019).
 - [30] Emilio Guirado et al. «Automatic whale counting in satellite images with deep learning». In: (2018).
 - [31] 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.
 - [32] 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).

BIBLIOGRAFIA

- [33] 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.
- [34] 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.
- [35] Olga Russakovsky et al. «ImageNet Large Scale Visual Recognition Challenge». In: *arXiv* (2014).