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

Anno Accademico 2018-2019

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'obbiettivo 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 il ri-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

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]

Capitolo 1

Teoria

Come già anticipato, l'approccio più efficace alla risoluzione del problema della classificazione delle immagini consiste nell'impiego delle reti neurali convoluzionali (CNN, *convolutional neural networks*). In questo capitolo saranno introdotti progressivamente i presupposti teorici matematici e informatici su cui si fondano le reti neurali, partendo dalle definizioni preliminari fino a costruire il modello generale di una CNN.

1.1 Reti neurali

TODO: parlare del neurone e dell'idea di replicarlo nella funzione (dopo aver parlato dei classificatori lineari, perché bisogna parlare anche

1.2 Immagini digitali

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

Un'**immagine digitale** è una rappresentazione binaria di un'immagine (in generale a colori) a due dimensioni[1]; essa può essere definita matematicamente come un tensore $\mathcal{I} \in \mathbb{R}^{h \times w \times c}$, 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 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 $r, g, b : \{0, \dots, h\} \times \{0, \dots, w\} \to \{0, \dots, 255\}$ 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

[1] Ci riferiamo in questa sede solo alle immagini di tipo raster, tipiche ad esempio delle fotografie digitali in formato jpg.

[2] Spesso si scrive che $\mathcal{I}$ è un immagine $w \times h \times c$, o più semplicemente $w \times h$

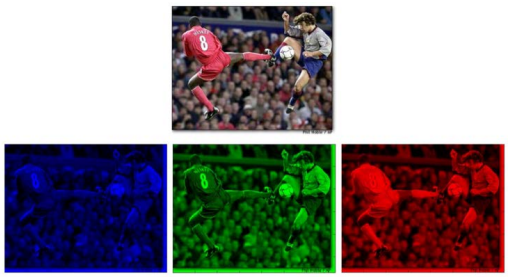

Figura 1.1: Canali RGB di un'immagine

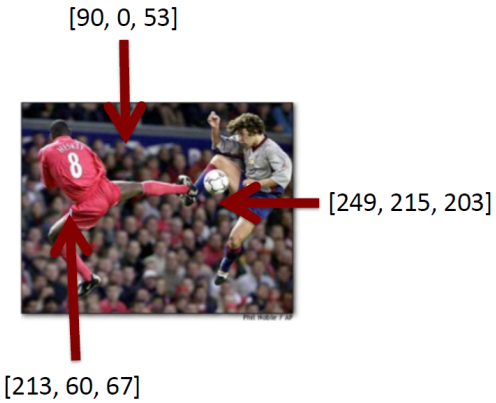

Figura 1.2: Pixel di un'immagine

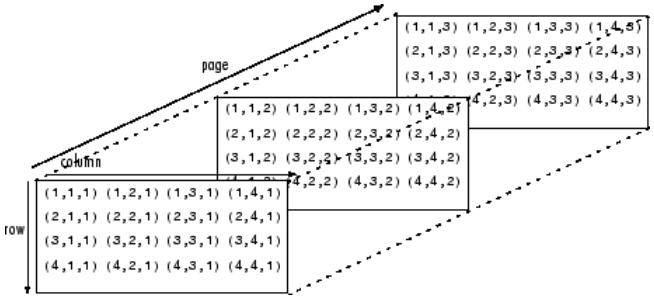

Figura 1.3: Rappresentazione grafica di un tensore tridimensionale; in ogni posizione compaiono gli indici del tensore

definisce univocamente un colore nello spazio RGB, il quale può rappresentare in tutto 256^3 colori diversi, cioè circa 17 milioni.

Si può immaginare il tensore immagine $\mathcal{I}$ come una "pila" di tre matrici, una per ogni canale di colore, come mostrato in figura 1.3.

1.2.1 Supervised Learning

Il paradigma dell'**apprendimento supervisionato** (*supervised learning*) si basa sulla creazione di un algoritmo in grado di apprendere una funzione che mappi un input all'output corretto, sulla base di una serie di esempi ideali costituiti da coppie di input e dei relativi output attesi, che gli vengono inizialmente forniti per addestrarlo [1]. Un algoritmo di apprendimento supervisionato analizza i dati di addestramento e inferisce una funzione che può essere usata per mappare nuovi input ai corretti output. 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, possono essere formulati ricorrendo ad una funzione matematica

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

che associa ad ogni elemento nello spazio degli input X (dataset) uno ed un solo elemento dello spazio degli output. 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 definizioni preliminari.

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

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

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 orientato all'*object recognition* (riconoscimento di un oggetto in un immagine), 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.[3]
Nella pratica, TODO f non può essere trovata esattamente. (vd gianvito)

TODO: scrivere ora o in un paragrafo a parte i tipi di dato per gestire le immagini messi a disposizione da matlab.

[3]Teoricamente una stessa immagine potrebbe essere descritta da più di un'etichetta o addirittura da nessuna, coerentemente col fatto che in essa potrebbero essere presenti più oggetti o nessun oggetto tra quelli previsti in Y. Tuttavia nella presente tesi questa ambiguità non può sussistere: la classificazione riduce qualsiasi immagine ad una di due categorie mutualmente esclusive e di cui almeno una deve essere ammessa, cioè la presenza o meno di una pinna nell'immagine.

1.3 Classificatore lineare

Il classificatore lineare è una tra le più semplici funzioni di classificazione.[4] 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

$$f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b} \tag{1.1}$$

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

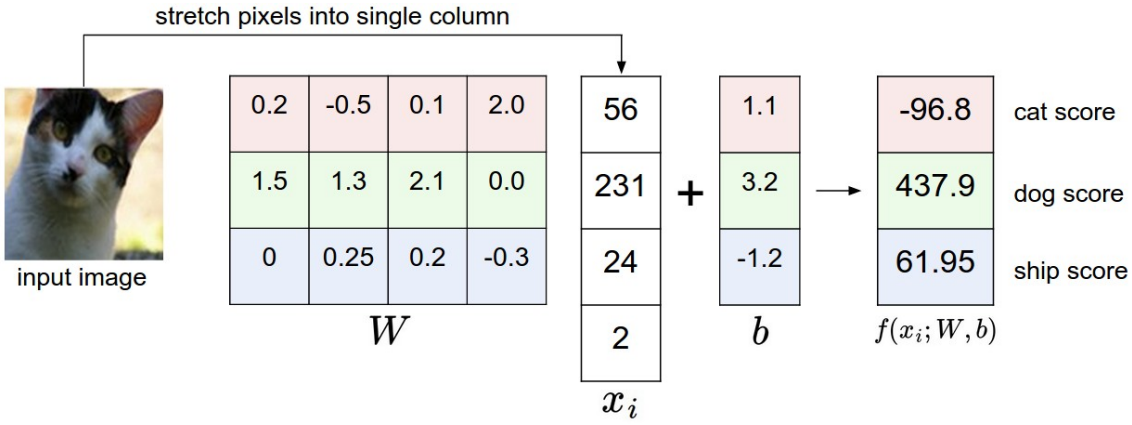

Figura 1.4: Mappatura di un'immagine ai punteggi di ogni classe mediante un classificatore lineare. Si noti che i pesi di $\mathbf{W}$ non costituiscono un buon set di parametri: il punteggio assegnato alla classe "cane" (sbagliata) è alto e quello totalizzato dalla classe "gatto" (corretta) è basso. Il classificatore "è convinto" di aver classificato l'immagine di un cane.

[4] La fonte principale per gli argomenti trattati in questo paragrafo è [2]

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 1.5.

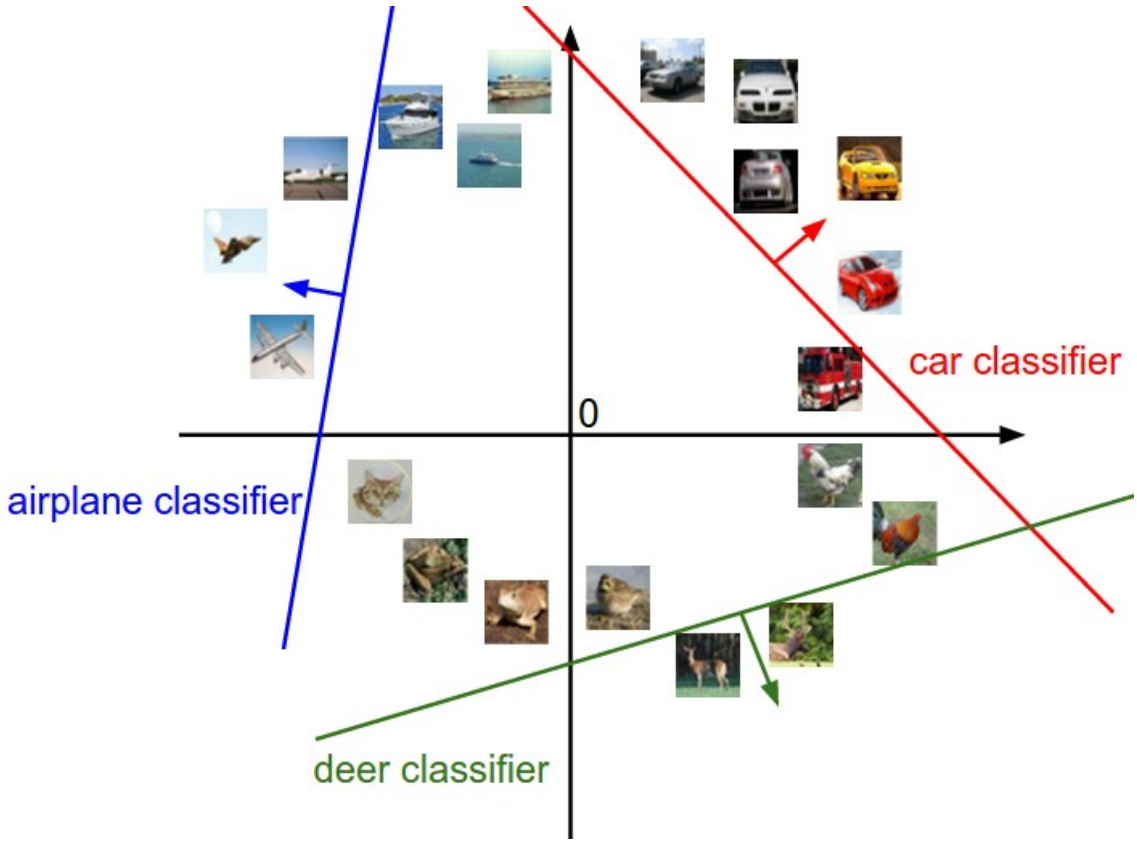

Figura 1.5: Visualizzazione di tre righe di un classificatore lineare, una per ciascuna delle classi "aereo", "auto", "cervo".

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 **b** significa regolare l'intercetta delle rette (cioè traslarle verticalmente).

Un altro modo di interpretare i pesi **W** può essere quello di far corrispondere ogni riga di **W** a un **prototipo** (in inglese **template**) per una delle classi. In questa interpretazione, il punteggio realizzato per ogni classe da un'immagine è ottenuto attraverso l'operazione di prodotto matriciale tra il prototipo della classe j ($\mathbf{w}_j$) e l'immagine da classificare ($\mathbf{x}^{(i)}$). Usando la terminologia introdotta, possiamo affermare che ciò che sta facendo il classificatore lineare è un'operazione di *template matching*, dove i *templates* sono oggetto di apprendimento da parte del classificatore[5].

Ad esempio, analizziamo il dataset *CIFAR-10* [3]. Esso contiene immagini 32×32 ciascuna appartenente ad una di 10 classi. Visualizzando[6] i pesi (e quindi i 10 templates) di un classificatore lineare addestrato su CIFAR-10 si ottengono i risultati in figura seguente:

Figura 1.6: Visualizzazione dei templates di un classificatore addestrato sul dataset CIFAR-10

Si possono fare alcune interessanti osservazioni.

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

[5]Si introdurranno gli algoritmi di apprendimento (supervisionato) nel capitolo **??**.

[6]TODO 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`.

Bias trick

Concludiamo questo capitolo menzionando un "trucco" matematico molto utilizzato per rappresentare **W** e **b** come un'unica matrice, semplificando la notazione 1.1. 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

$$f(\mathbf{x}^{(i)}; \mathbf{W}) = \mathbf{W}\mathbf{x}^{(i)} \tag{1.2}$$

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

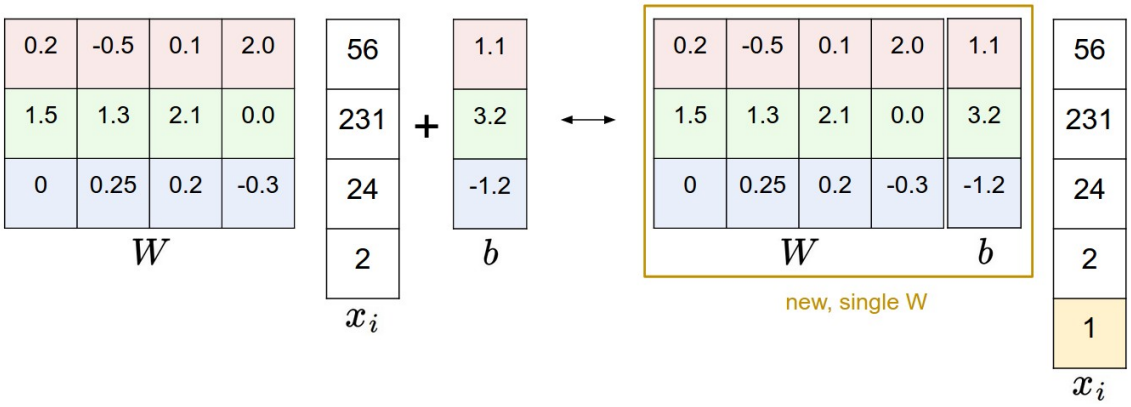

Figura 1.7: Bias trick

TODO: loss functions.

1.4 AlexNet

AlexNet è una CNN creata tra il 2011 e il 2012 da Alex Krizhevsky, in collaborazione con Ilya Sutskever e Geoffrey Hinton [4]. La vittoria di AlexNet nella ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [5], 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 [6]

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. L'alto 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.

1.4.1 Architettura di AlexNet

L'architettura di AlexNet è riportata schematicamente nella figura 1.8 e con maggiore dettaglio in tabella 1.1. 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.

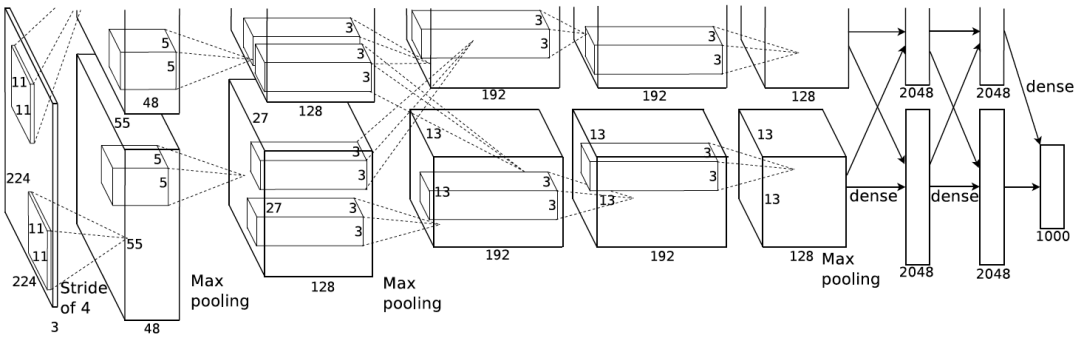

Figura 1.8: Architettura originale di AlexNet [4]

Come si evince dalla figura 1.8, 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 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*[7].

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

Funzione di attivazione ReLU

Dopo ogni strato parametrizzato, i valori delle attivazioni sono passati alla funzione attivatrice "rettificatore": $f(x) = x^+ = \max(0, x)$ [7]. 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 $f(x) = \tanh(x)$.

Local Response Normalization

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

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

dove $a^i_{x,y}$ è l'attivazione del neurone ottenuto applicando il kernel i-esimo alla posizione (x, y) e applicando in seguito la funzione ReLU, $b^i_{x,y}$ 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.

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. È stato osservato durante la fase di training che questa funzione di *max pooling con sovrapposizione* ha attenuato lievemente l'*overfitting* della rete.

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

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, quest'ultimo è stato "arricchito" con le seguenti immagini:

- Estrazione casuale di ritagli 224×224 dalle immagini
- Riflessione orizzontale ("a specchio") delle immagini
- Somma di un'immagine e le sue componenti principali (PCA)[8]

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* [9]. Questa tecnica consiste nel settare a zero l'attivazione di ciascun neurone di un layer intermedio con probabilità p (AlexNet impiega un dropout con $p = 0.5$. 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 AlexNet, il dropout dei neuroni è utilizzato nei primi due layer completamente connessi. In fase di test, i neuroni di questi due strati sono moltiplicati per 0.5 per tenere conto dell'impiego del dropout in addestramento.

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. Dettagli più specifici sulla fase di addestramento di AlexNet possono essere trovati nel paper originale [4].

[8] L'*analisi delle componenti principali* (PCA, principal component analysis) è una tecnica per la semplificazione dei dati utilizzata nell'ambito della statistica multivariata. In questa sede ci limitiamo a specificare che il suo utilizzo nell'ambito della data augmentation è di evidenziare una importante proprietà delle immagini naturali, e cioè che l'identità di un oggetto è invariante rispetto ai cambi d'intensità e di colori nella sua illuminazione. Si rimanda ad esempio a [8] per approfondimenti sulla PCA.

Tabella 1.1: Architettura originale di AlexNet

N	Layer	Attivazioni	Parametri
1	INPUT	$(227 \times 227 \times 3)$	
2	CONVOLUTION	$(55 \times 55 \times 96)$	Pesi: $(11 \times 11 \times 3) \times 96$ Bias: (96)
3	RELU	–	–
4	NORMALIZATION	–	–
5	MAX POOLING	$(27 \times 27 \times 96)$	–
6	GROUPED CONVOLUTION	$(27 \times 27 \times 256)$	Pesi: $(5 \times 5 \times 48) \times 128 \times 2$ Bias: $(128) \times 2$
7	RELU	–	–
8	NORMALIZATION	–	–
9	MAX POOLING	$(13 \times 13 \times 256)$	–
10	CONVOLUTION	$(13 \times 13 \times 384)$	Pesi: $(3 \times 3 \times 256) \times 384$ Bias: (384)
11	RELU	–	–
12	GROUPED CONVOLUTION	$(13 \times 13 \times 384)$	Pesi: $(3 \times 3 \times 192) \times 192 \times 2$ Bias: $(192) \times 2$
13	RELU	–	–
14	GROUPED CONVOLUTION	$(13 \times 13 \times 256)$	Pesi: $(3 \times 3 \times 192) \times 128 \times 2$ Bias: $(128) \times 2$
15	RELU	–	–
16	MAX POOLING	$(6 \times 6 \times 256)$	–
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	–	–

Bibliografia

[1] Stuart Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2009.

[2] Andrej Karpathy. *Lectures of Stanford class CS231n: Convolutional Neural Networks for Visual Recognition*. Appunti del corso Stanford CS class C231n. 2019.

[3] Alex Krizhevsky. «Learning multiple layers of features from tiny images». In: (2009).

[4] Alex Krizhevsky, Ilya Sutskever e Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems 25*. A cura di F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[5] *ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)*. 2012. URL: http://image-net.org/challenges/LSVRC/2012/results.

[6] Md Zahangir Alom et al. «The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches». In: (3 mar. 2018). arXiv: http://arxiv.org/abs/1803.01164v2 [cs.CV].

[7] Vinod Nair e Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». In: *ICML*. A cura di Johannes Fürnkranz e Thorsten Joachims. Omnipress, 2010, pp. 807–814. URL: http://dblp.uni-trier.de/db/conf/icml/icml2010.html#NairH10.

[8] Sergio Bolasco. *Analisi multidimensionale dei dati. Metodi, strategie e criteri d'interpretazione*. Carocci Editore, 1999. ISBN: 9788843014019. URL: https://www.amazon.com/Analisi-multidimensionale-strategie-criteri-dinterpretazione/dp/8843014013?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=8843014013.

[9] Geoffrey E. Hinton et al. «Improving neural networks by preventing co-adaptation of feature detectors». In: (3 lug. 2012). arXiv: http://arxiv.org/abs/1207.0580v1 [cs.NE].

[10] 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. ????

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