

Università degli Studi di Padova
Corso di Laurea Triennale in Ingegneria Informatica



Deep Learning

- Tecniche predittive per l'insorgenza di Alzheimer -

Laureando: *Luca Masiero*

Relatore: *Prof. Loris Nanni*

Anno Accademico 2018-2019

15 Luglio 2019

Alla mia famiglia.

Contents

1	Introduzione	2
2	Reti Neurali	4
2.1	Struttura e funzionamento del neurone biologico	4
2.2	Le sinapsi	5
2.3	I canali ionici	6
2.4	Struttura del neurone artificiale	6
2.5	Architettura di una rete neurale artificiale	7
2.6	Le funzioni di attivazione	7
3	Che cos’è il <i>Deep Learning</i>?	9
3.1	<i>Task</i> e <i>performance</i>	9
3.2	Principio di funzionamento	10
3.3	Qual è la differenza tra <i>Machine Learning</i> e <i>Deep Learning</i> ?	11
3.4	Transfer Learning	11
4	Algoritmi di ottimizzazione	13
4.1	Discesa stocastica del gradiente	13
4.1.1	Stochastic Gradient Descent con <i>momento</i>	14
4.2	<i>Adaptive moment estimation: adam</i>	15
5	Dropout e L_2 Regularization	16
6	Reti Neurali Convoluzionali	18
6.1	Convolution layer	18
6.2	Pooling layer	20
6.3	ReLU layer	21
6.4	Fully Connected layer e <i>Softmax</i>	21
7	AlexNet e diagnosi di Alzheimer: nuovi metodi	23
7.1	Struttura del codice	23
7.1.1	Metodo 1	24
7.1.2	Metodo 2	25
7.1.3	Metodo 3	27
7.2	Risultati dei test	28
7.2.1	Metodo 1	28
7.2.2	Metodo 2	31
7.2.3	Metodo 3	33
8	Conclusioni	34
9	Bibliografia	35

1 Introduzione

L'Alzheimer (AD, *Alzheimer's disease*) è una *malattia neurologica*¹ progressiva, irreversibile e multiforme, che si manifesta spesso in età avanzata; colpisce più di 47 milioni di persone in tutto il mondo ogni anno.

Questa malattia distrugge lentamente le cellule del cervello causando perdita di memoria, limitazioni nell'abilità di pensiero e nell'apprendimento, portando infine all'incapacità di eseguire le più semplici azioni quotidiane.

L'Alzheimer è una fra le malattie che non possono ancora essere rallentate o curate.² Una diagnosi tempestiva e definita è imperativa affinché i trattamenti necessari a rallentare l'incedere di questa malattia risultino davvero efficaci. L'obiettivo del raggiungimento di una diagnosi accurata e in anticipo richiede un'analisi dello stadio pre-demenza, denominato *Deterioramento Cognitivo lieve* (dall'inglese *Mild Cognitive Impairment*, MCI). Lo scopo della ricerca consiste nel determinare se l'MCI si trasformerà (MCIC) o meno (MCInc) in AD.

La diagnosi individuale clinica dell'Alzheimer è principalmente basata su di una valutazione neuropsicologica e di altri test amministrati sui pazienti. Una diagnosi definitiva è tuttavia possibile solamente tramite un'analisi post-mortem.

I criteri di diagnosi clinica dell'Alzheimer, che vennero sviluppati negli anni '80 del secolo scorso dal *National Institute of Neurologic and Communicative Disorders and Stroke and the Alzheimer's Disease and Related Disorders Association* (NINCDS-ADRDA) sono ancora in fase di definizione. I processi diagnostici si basarono principalmente sulla presenza di menomazione cognitiva. Venne allora introdotta l'informazione neuropatologica, basata sulla presenza di placche e grovigli neurofibrillari. Questi criteri sono stati recentemente revisionati dal *National Institute on Aging-Alzheimer's Association* (NIA-AA), che ha introdotto diverse nuove tecniche di supporto nella diagnosi della malattia: 1) test neurogenetico, 2) misurazione del fluido cerebrospinale (CSF)³, 3) *amiloide*⁴ e *tau*⁵, e 4) *biomarcatori*⁶ di danni neuronali che utilizzano tecniche di *neuroimaging* (*i.e.* MRI e PET, ovvero *Magnetic Resonance Imaging* e *Positron Emission Tomography*).

L'introduzione di tecniche di *imaging* nei criteri di diagnosi per l'Alzheimer, in particolare MRI e PET, è stata fondamentale per una migliore misurazione, da un lato, di atrofia, e, dall'altro, di marcatori di metabolismo/amiloide. Cambiamenti in queste *features* possono essere identificati anche prima del sopraggiungere della demenza, dal momento che rappresentano uno strumento di grande supporto per la diagnosi della malattia. Oltre a ciò, l'MRI è una tecnica vantaggiosa e soprattutto non invasiva.

¹Le malattie neurologiche comprendono essenzialmente i disturbi che colpiscono il sistema nervoso centrale (*e.g.* sclerosi multipla, malattie cerebrovascolari, Parkinson, epilessia).

²Il cervello è il centro del sistema nervoso umano, esso è composto da più di 100 miliardi di nervi che comunicano tramite trilioni di connessioni, chiamate *sinapsi*. Contiene regioni che sono spazialmente distribuite ma funzionalmente connesse che condividono continuamente stimoli e risposte le une verso le altre. Con l'avanzare dell'età il cervello sviluppa placche e grovigli che disabilitano o bloccano la comunicazione fra le cellule nervose che, impossibilitate alla loro funzione, muoiono.

³Il liquido cefalorachidiano (LCR) (denominato anche *liquor*, liquido cerebrospinale o liquido rachido-spinale, in inglese *cerebrospinal fluid* con acronimo CSF) è un fluido corporeo trasparente e incolore che si trova nel sistema nervoso centrale.

⁴Le *placche senili*, dette anche *placche amiloidi*, sono formazioni extracellulari costituite da una parte centrale in cui si accumula la proteina amiloide, e una parte periferica in cui si depositano detriti neuronali (principalmente frammenti assonali); sono riscontrabili nell'encefalo, in misura maggiore a livello dell'ippocampo, e nelle corteccce associative delle regioni frontali e temporo-parietali.

⁵La proteina *tau* si diffonde da un neurone all'altro con le stesse modalità dell'influenza: le aree cerebrali maggiormente connesse, e quindi con molti contatti con le altre, sono anche quelle che si "ammalano" di più.

⁶Un marcatore biologico, o *biomarcatore*, è qualcosa che può essere misurato e che indica la presenza di una malattia, un cambiamento fisiologico, una risposta a un trattamento o una condizione psicologica.

Per queste ragioni, negli ultimi anni la ricerca si è concentrata molto sull'implementazione e lo sviluppo di tecniche sempre più sofisticate per l'MRI che sfruttano sistemi di *machine learning* per migliorare la qualità di diagnosi dell'Alzheimer.

Sistemi automatici in grado di distinguere soggetti malati grazie agli studi dell'MRI potrebbero rappresentare passi significativi verso una rapida diagnosi di questa malattia. Il problema della classificazione automatica dell'Alzheimer è costituito tuttavia dalle grandi dimensioni dei *feature vectors*⁷ dell'MRI e dalla piccola quantità di elementi da allenare all'interno dei dataset.

La convergenza di questi due aspetti porta al *curse of dimensionality problem*: quando la dimensionalità aumenta, il volume dello spazio aumenta così rapidamente che i dati disponibili diventano sparsi e difficili da generalizzare.

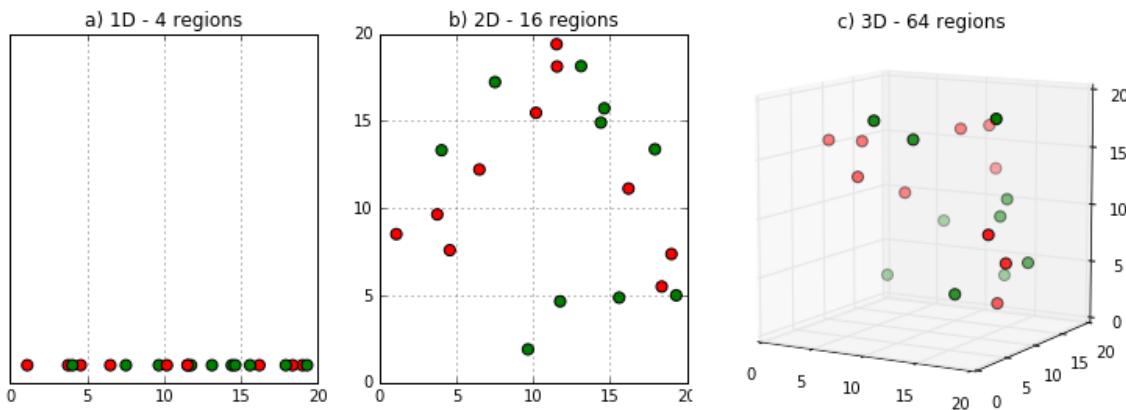


Figura 1.1: Rappresentazione in più dimensioni del *curse of dimensionality problem*.

Per progettare sistemi di classificazione automatica davvero performanti è essenziale sviluppare tecniche che permettano di estrarre un sottoinsieme (*subset*) dall'insieme (*set*) originale di *features* per le quali le performance di classificazione siano sempre ottime.

A questo punto potreste chiedervi: *"Per leggere questa tesi è quindi obbligatorio conoscere la terminologia medica?"* o addirittura *"È necessario avere una laurea in Neurologia?"*

La risposta è assolutamente **no**. Io stesso, che ho sviluppato questa tesi, non conosco il funzionamento di una risonanza magnetica eppure questa è una tesi che tratta di Alzheimer.

Ho sviluppato tre nuovi metodi che, partendo da dataset di piccole dimensioni, sono in grado di determinare (con buone prestazioni) se una persona potrebbe o meno essere affetta dalla malattia.

Gli argomenti che presenterò successivamente forniranno da base teorica per la comprensione della struttura e del funzionamento di questi metodi, cui ho riservato la sezione finale della tesi.

⁷Un *feature vector* è un vettore che contiene le informazioni importanti necessarie alla descrizione di un oggetto.

2 Reti Neurali

Questa prima sezione introduttiva ha lo scopo di delineare un interessante parallelismo: è infatti fondamentale conoscere il funzionamento del nostro cervello, la struttura dei neuroni al suo interno e come essi comunicano fra di loro, al fine di riconoscere tutte le somiglianze ed analogie con la teoria che sta alla base del *Deep Learning*.

Le **reti neurali** (in inglese *neural networks*) sono un *modello computazionale con una struttura gerarchica*, organizzata per livelli, che vede come propria unità fondamentale il **neurone artificiale**, introdotto per la prima volta nel 1943 da Warren McCulloch (1898-1969, neurofisiologo) e Walter Pitts (1923-1969, matematico)⁸.

2.1 Struttura e funzionamento del neurone biologico

La parte centrale del neurone viene denominata **soma**, o *corpo cellulare*, ed è costituita dal *pirenoforo*, in cui risiede il **nucleo**, e da altri organi dedicati alle principali funzioni cellulari (come l'apparato di Golgi, i neurofilamenti, i neurotubuli, i mitocondri⁹, e i reticoli endoplasmatici liscio e rugoso).

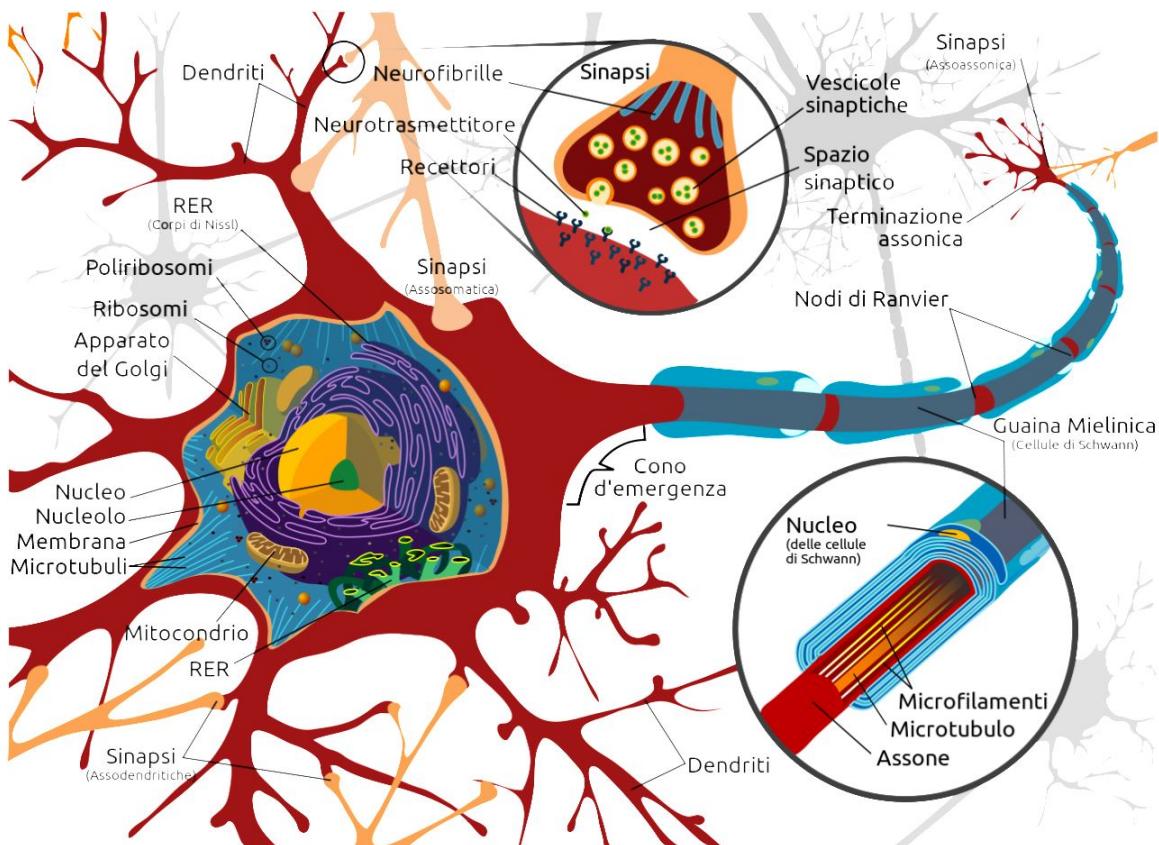


Figura 2.1: Rappresentazione schematica di un neurone biologico.

⁸Walter H. Pitts Warren S. McCulloch. *A logical calculus of the ideas immanent in nervous activity*. *Bulletin of Mathematical Biophysics*, (5):115–137, 1943. In questo articolo McCulloch e Pitts cercarono di dimostrare che una macchina di Turing poteva essere realizzata con una rete finita di neuroni, per dimostrare che il neurone era l'unità logica di base del cervello.

⁹I mitocondri sono organelli cellulari di forma generalmente allungata (reniforme o a forma di fagiolo): costituiscono la centrale energetica della cellula.

Sebastian Seung, neuroscienziato dell'università di Princeton, con quella che può sembrare una battuta, ha definito il neurone una "cellula poliamorosa", dal momento che dal corpo cellulare hanno origine prolungamenti citoplasmatici, detti *neuriti*, ovvero i **dendriti** e l'**assone**, che costituiscono un folto fascio di ramificazioni con le quali abbraccia altre migliaia di neuroni.

I dendriti, più corti e spessi, che hanno diramazioni simili ad un albero, ricevono segnali da neuroni afferenti e li propagano verso il pirenoforo (fungono dunque da *ricettori* dell'impulso nervoso). A differenza dell'assone, i dendriti non sono dei buoni conduttori dei segnali nervosi.

L'assone, lungo e sottile, conduce il segnale verso altre cellule (funge dunque da *trasmettitore* dell'impulso). Ha un diametro uniforme ed è un ottimo conduttore grazie agli strati di mielina¹⁰ (la sua velocità di trasmissione è di 120 m/s, corrispondenti a circa 432km/h). La parte terminale dell'assone è un'espansione detta **bottone sinaptico**. Attraverso questi bottoni un assone può prendere contatto con i dendriti o il corpo cellulare di altri neuroni affinché l'impulso nervoso si propaghi, con una reazione a catena, lungo un circuito neuronale.

2.2 Le sinapsi

L'impulso nervoso viaggia da un neurone all'altro, lo fa grazie alla *connessione che si stabilisce fra l'assone del neurone che trasmette il segnale e il dendrite del neurone che lo riceve*.

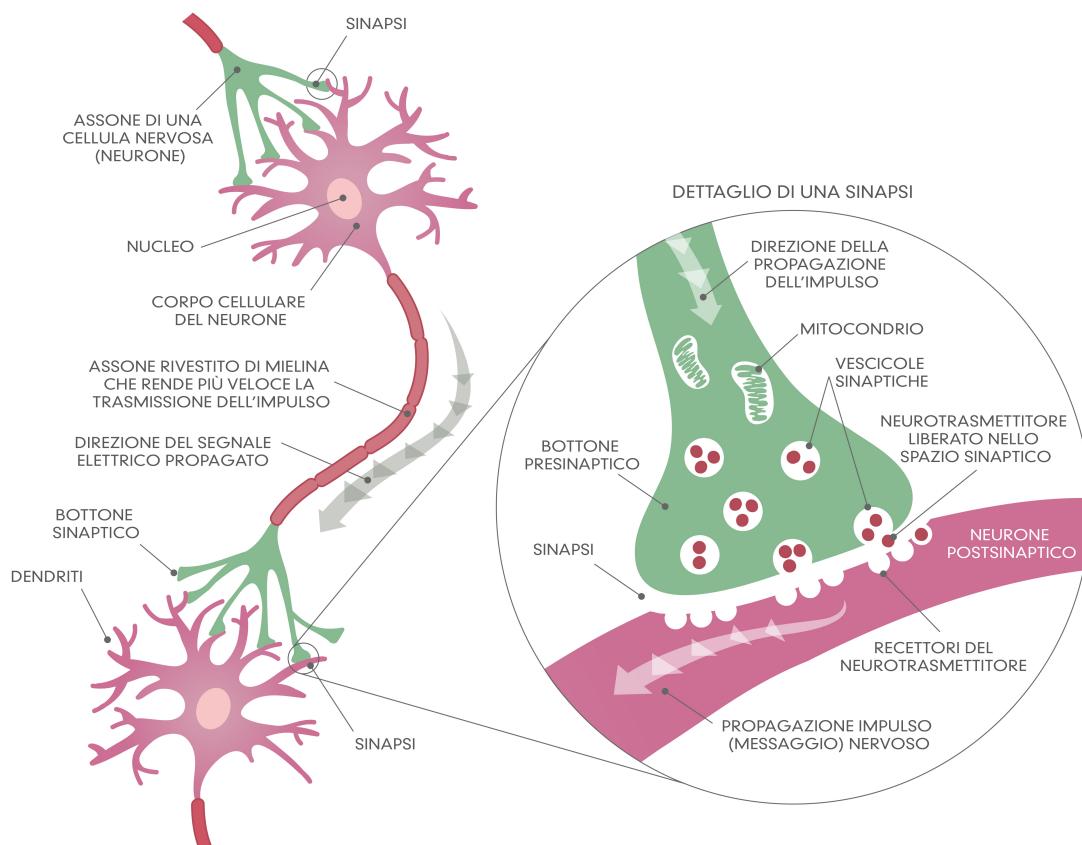


Figura 2.2: Rappresentazione schematica di una sinapsi.

¹⁰Sostanza che costituisce la guaina midollare delle fibre nervose e che ha funzione, oltre che protettiva, isolante nei riguardi della conduzione dello stimolo nervoso.

Non si tratta di una connessione diretta: fra le estremità delle due cellule si apre un piccolo spazio vuoto, detto **sinapsi**.

Come succede con i fili della corrente elettrica, se nel circuito c'è un'interruzione l'elettricità non passa. Per superare la fessura il bottone sinaptico, situato all'estremità dell'assone, libera alcune molecole, dette **neurotrasmettitori**, che attraversano lo spazio intersinaptico fino ad unirsi ai recettori situati all'"estremità" del dendrite (nota come *spina dendritica*).

2.3 I canali ionici

Quando la corrente elettrica arriva al terminale sinaptico e si produce dunque la liberazione dei neurotrasmettitori, la comparsa del *potenziale d'azione*¹¹ del neurone che riceve il segnale avviene grazie a proteine che si attivano quando il neurotrasmettore si unisce al recettore. Queste proteine sono dette **canali ionici**.

I canali ionici trasformano nuovamente il segnale chimico della sinapsi tra due neuroni in un impulso elettrico. Si tratta di proteine presenti nella membrana cellulare che agiscono come tubi in grado di aprirsi e chiudersi per lasciar passare un determinato tipo di atomo con carica elettrica (per esempio ioni sodio, potassio o calcio). In stato di riposo, l'interno della cellula ha una carica negativa rispetto all'esterno.

Quando l'azione del neurotrasmettore provoca l'apertura dei canali del sodio, tali ioni passano dall'esterno all'interno della cellula e ciò modifica il potenziale di membrana: la parte interna diventa meno negativa, o più positiva. Tale modificazione è nota come **depolarizzazione** e produce il potenziale d'azione del neurone recettore.

Questo processo, apparentemente semplice, costituisce la base della nostra attività cerebrale perché è grazie ad esso che i neuroni si connettono formando reti in grado di elaborare e trasmettere i segnali generando risposte e reazioni.

2.4 Struttura del neurone artificiale

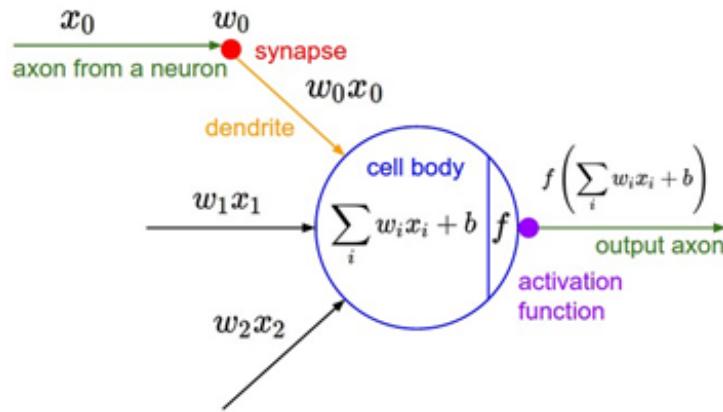


Figura 2.3: Schematizzazione di un neurone artificiale.

Seguendo lo stesso principio, un **neurone artificiale** riceve in input dei segnali x_1, x_2, \dots, x_i da i assoni di neuroni afferenti. Ogni segnale è caratterizzato da un peso w_i , che rappresenta l'efficacia

¹¹Rapido cambiamento di carica tra l'interno e l'esterno della membrana cellulare. L'esterno è caricato positivamente (+), l'interno negativamente (-). Durante un potenziale d'azione neuronale l'informazione nervosa viene trasmessa saltando da un *nodo di Ranvier* (sezione in cui la guaina mielinica che ricopre i neuroni si interrompe) all'altro: tale processo dura circa 2 ms.

della connessione sinaptica del dendrite. Oltre a questi vi è un ulteriore peso b (*bias*) che si considera collegato ad un input fittizio avente valore sempre uguale ad 1 e che serve per regolare il punto di lavoro del neurone stesso: lo scopo è dunque quello di permettere la traslazione della funzione di attivazione del neurone in modo da poter ottenere con certi input e certi pesi tutti i potenziali output desiderati.

L'output finale $out_i = f(net_i)$ è dato dalla funzione di attivazione f e dipende dal potenziale interno (detto anche *livello di eccitazione globale*) del neurone:

$$\sum_i w_i \cdot x_i + b.$$

2.5 Architettura di una rete neurale artificiale

Le reti neurali sono formate da livelli di neuroni artificiali interconnessi tra di loro ed organizzati in maniera gerarchica; la loro migliore rappresentazione si ottiene tramite *grafi orientati*: i nodi rappresentano i singoli neuroni mentre gli archi rappresentano le connessioni input/output tra i neuroni stessi.

In genere una rete neurale è composta da un livello di input, uno o più livelli intermedi (*hidden layers*) e un livello di output.

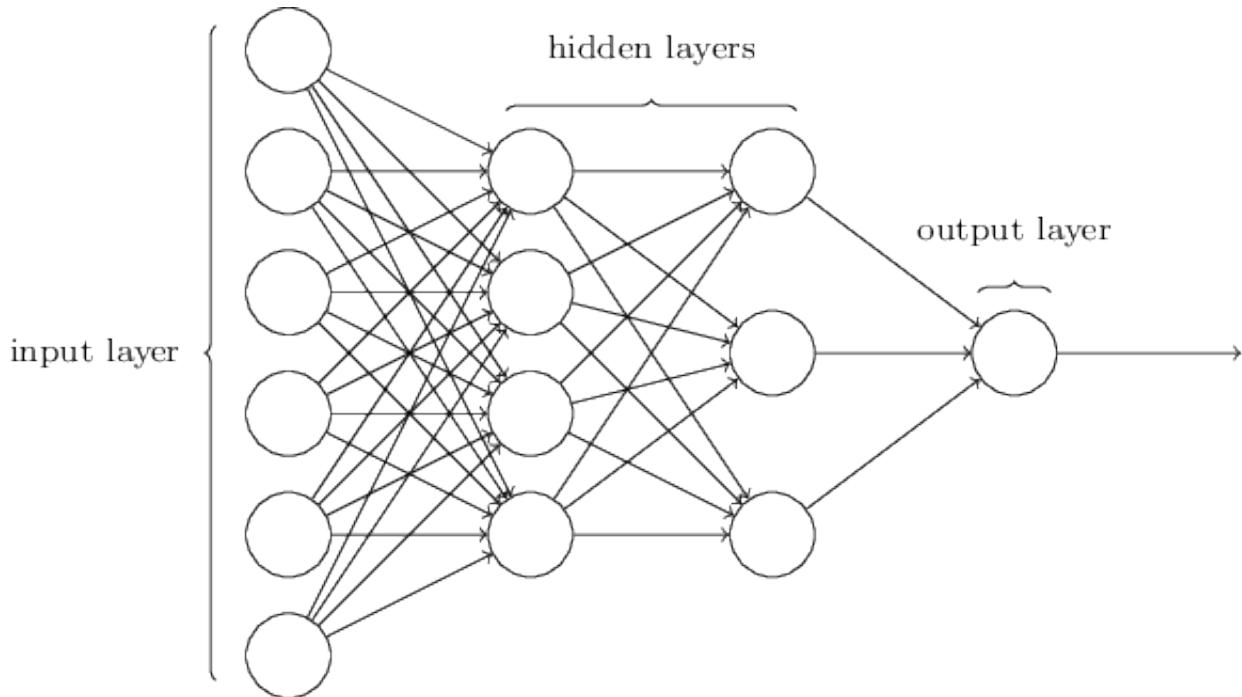


Figura 2.4: Un semplice esempio di rete neurale artificiale.

2.6 Le funzioni di attivazione

Le **funzioni di attivazione** utilizzate nella rete sono fondamentali in quanto determinano l'output dei singoli nodi e, di conseguenza, anche quali caratteristiche dei dati forniti come input verranno propagate nella rete stessa e quali filtrate.

Nel contesto dei neuroni biologici, lo scambio di informazioni avviene fisicamente tramite processi elettro-chimici: il passaggio di ioni dalle sinapsi dei dendriti porta alla creazione di una differenza

di potenziale tra il neurone e l'esterno: questa continua ad aumentare con l'accumularsi degli ioni finché non viene raggiunto un valore di soglia: il neurone "spara" un impulso elettrico lungo il proprio assone e poi torna a riposo.

Le reti neurali artificiali utilizzano *funzioni di attivazione non-lineari* in modo da poter meglio approssimare un maggior numero di funzioni e permettere un mapping (tra input e output) più complesso. Se si considera la fase di *training* della rete è importante che $f(\cdot)$ sia *continua* e *differenziabile* così da poterne calcolare il *gradiente* e determinare la *retropropagazione dell'errore*.

Un singolo neurone riceve in ingresso un valore numerico (corrispondente alla somma pesata dei diversi input) e può attivarsi (elaborando il valore ricevuto tramite una specifica funzione) oppure rimanere inattivo (a seconda che venga superata o meno la sua soglia di attivazione).

Le funzioni di attivazione più importanti sono:

- La funzione **sigmoide**, che riceve in input un qualsiasi numero reale e restituisce in output un numero compreso nell'intervallo $[0, 1]$:

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

- La funzione **tangente iperbolica**, che si può ricavare a partire dall'equazione della funzione sigmoide, applicando un cambiamento di scala di 2 e una traslazione di -1, cioè:

$$\tanh(z) = 2\sigma(2z) - 1.$$

Pur essendo più complessa rispetto alla funzione di origine, \tanh è preferibile poiché converge più velocemente grazie alla simmetria rispetto allo zero.¹²

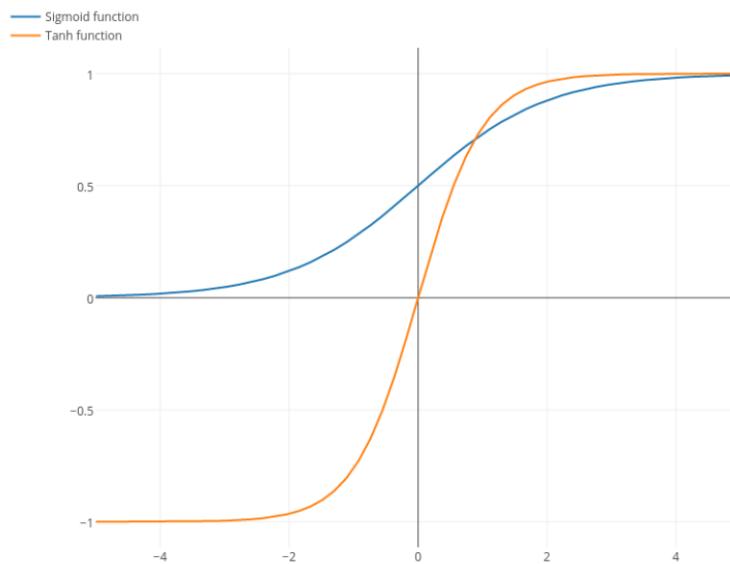


Figura 2.5: Funzioni sigmoide e tangente iperbolica a confronto.

¹² Al seguente link sarà possibile avere idee chiare sulle funzioni di attivazione in generale, nonché un efficace confronto tra le due funzioni sopra brevemente descritte:

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

A questo punto saranno abbastanza chiari i funzionamenti sia dei neuroni biologici che di quelli artificiali; forse però è il caso che io spieghi brevemente il significato delle due parole che costituiscono il titolo di questa tesi.

3 Che cos'è il *Deep Learning*?

Il **Machine Learning**¹³ (in italiano *Apprendimento Automatico*) è una branca dell'**Intelligenza Artificiale** che si basa sull'idea che una *macchina* possa **imparare** dai dati, **identificare modelli** in modo autonomo e **prendere decisioni** con un intervento umano ridotto al minimo.

A livello intuitivo è importante sapere cosa si intende per *algoritmo di machine learning*¹⁴: si tratta di un algoritmo che è capace di *imparare* dai dati: ma cosa significa "imparare"? Tom M. Mitchell (1951, computer scientist americano) ci dà una definizione succinta:

Si dice che un programma impara da un'esperienza E sulla base di una classe di task T e di una misurazione di performance P, se la sua performance nel task T, misurata da P, aumenta con esperienza E.

3.1 Task e performance

Il processo di apprendimento stesso non è il *task*. Per **apprendimento** si intende la nostra abilità di realizzare il *task* (che si può quindi tradurre con "obiettivo"). Per esempio, se vogliamo che un robot sia in grado di camminare, allora "il camminare" è il *task*. Potremmo programmare il robot in modo che impari a camminare, o potremmo provare direttamente a scrivere un programma che specifica "come" camminare.

Nel machine learning, i *task* sono spesso descritti tramite **esempi**: un esempio è una collezione di *features*, cioè di caratteristiche che sono state quantitativamente misurate da un oggetto o da un evento che vogliamo che il sistema di machine learning analizzi. In genere un esempio si rappresenta tramite un vettore \mathbf{x} dove ogni componente x_i del vettore corrisponde ad una *feature*. Per esempio, le *features* di un'immagine sono spesso i valori dei pixels che compongono l'immagine stessa.

Per valutare l'efficienza di un algoritmo di machine learning dobbiamo essere in grado di valutare *quantitativamente* la sua performance: per farlo spesso si misura l'**accuracy** (cioè l'*accuratezza*) del modello. L'*accuracy* si definisce come la proporzione di esempi per i quali il modello ha prodotto un output corretto. Si possono ottenere informazioni utili in modo equivalente misurando l'**error rate**, cioè la proporzione di esempi per i quali il modello ha fornito output non corretti.

In generale si è interessati a valutare le prestazioni di un algoritmo su dati che non sono mai stati esaminati prima, si utilizza infatti un *test set* separato dai dati forniti durante la fase di allenamento.

Il **Deep Learning** è dunque una tecnica di machine learning che insegna ai computer a svolgere un'attività naturale per l'uomo: *imparare con l'esempio*.

Nel deep learning, un modello computerizzato impara a svolgere attività di classificazione direttamente da immagini, testo o suoni, superando talvolta le prestazioni ottenute dall'uomo. L'addestramento dei modelli viene eseguito utilizzando un grande set di dati etichettati e architetture di reti neurali contenenti più layers (gli "strati" della rete).

¹³Il termine venne coniato nel 1959 da Arthur Samuel (1901-1990, informatico statunitense)

¹⁴Per una trattazione teorica completa rimando alla lettura del *Deep Learning* di Ian Goodfellow, Yoshua Bengio e Aaron Courville.

Il deep learning è stato teorizzato per la prima volta negli anni '80, ma è diventato utile soltanto di recente per due ragioni principali:

1. Richiede una *grande quantità di dati etichettati*: per esempio, per lo sviluppo delle automobili senza conducente sono necessari milioni di immagini e migliaia di ore di video.
2. Richiede una *notevole potenza elaborativa*: le GPU ad alte prestazioni sono infatti dotate di un'architettura parallela molto più che efficiente. In combinazione con i *cluster* o il *cloud computing*, i team di sviluppo sono in grado di ridurre i tempi di addestramento per una rete neurale da diverse settimane a poche ore.

3.2 Principio di funzionamento

La maggior parte dei metodi di deep learning utilizza le architetture di reti neurali, motivo per cui i modelli di deep learning sono spesso denominati "reti neurali *profonde*".

Il termine "profondo" si riferisce al numero di layer nascosti nella rete neurale. Le reti neurali tradizionali contengono solo 2-3 layer nascosti, mentre le reti profonde possono arrivare addirittura fino a 150.

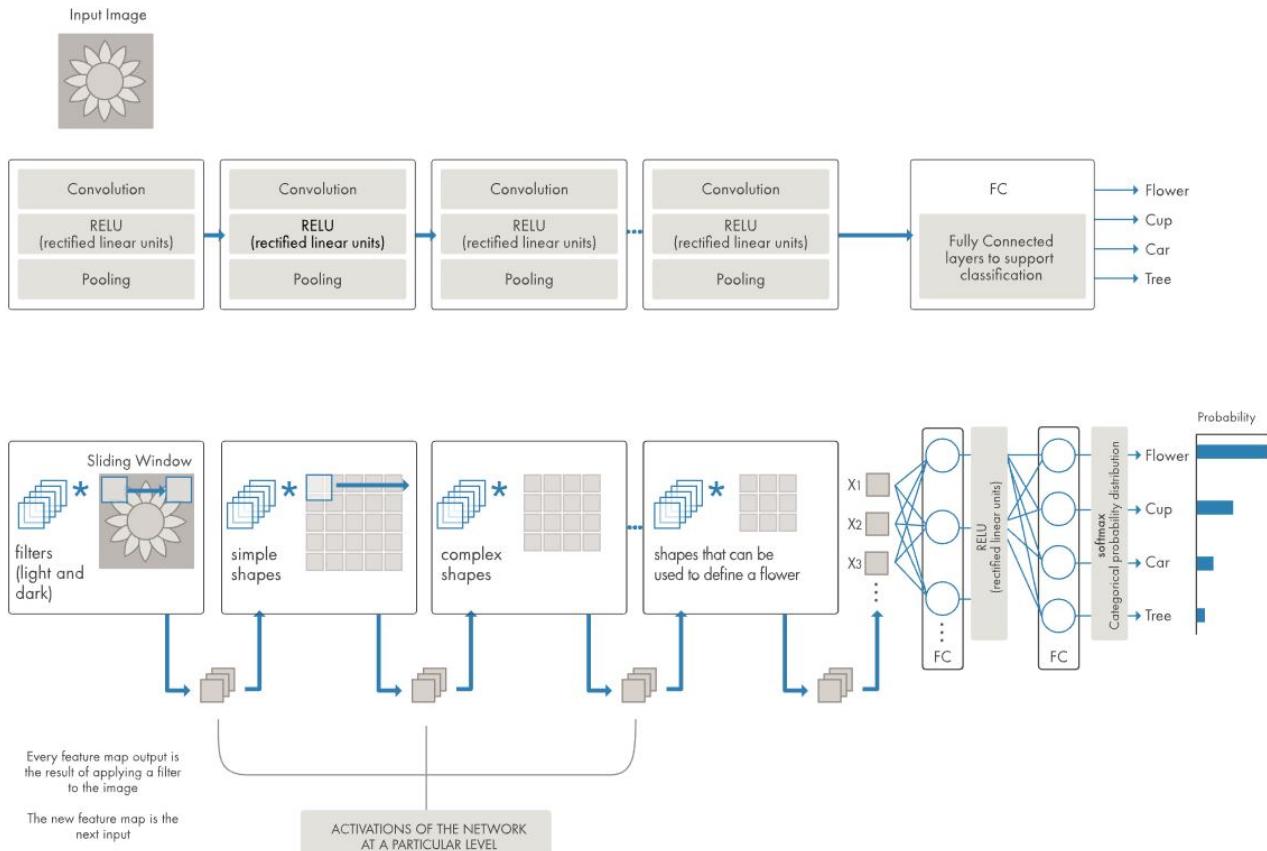


Figura 3.1: Esempio di una rete con molti layer convoluzionali. A ciascuna immagine, in diverse risoluzioni, vengono applicati dei filtri e l'output di ogni immagine convolta serve da input per il layer successivo. Una delle tipologie più comuni di reti neurali è nota come rete neurale *convoluzionale*, argomento che verrà trattato prossimamente.

L’addestramento dei modelli di deep learning viene eseguito utilizzando grandi set di dati etichettati e architetture di reti neurali in grado di apprendere le *features* direttamente dai dati senza la necessità di estrarre manualmente.

3.3 Qual è la differenza tra *Machine Learning* e *Deep Learning*?

Il deep learning è una forma specifica di machine learning. Un flusso di lavoro di machine learning inizia con l’estrazione manuale delle *features* significative dalle immagini. Le *features* vengono quindi utilizzate per creare un modello che categorizza gli oggetti nell’immagine. Con il deep learning le *features* significative vengono estratte automaticamente dalle immagini. Inoltre, il deep learning esegue un “apprendimento end-to-end” in cui una rete apprende automaticamente come elaborare dati grezzi e svolgere un’attività (per esempio una classificazione).

Un’altra differenza fondamentale è che gli algoritmi di deep learning scalano coi dati, mentre l’apprendimento superficiale utilizza la convergenza. Per *apprendimento superficiale* si intendono i metodi di machine learning che non consentono ulteriori sviluppi una volta raggiunto un certo livello di performance, quando si aggiungono esempi e dati di training alla rete.

Un vantaggio fondamentale delle reti di deep learning è la possibilità di migliorare le prestazioni con l’aumentare del formato dei dati.

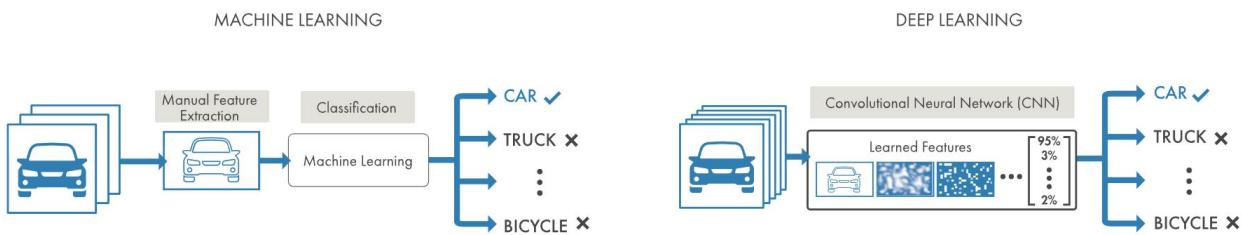


Figura 3.2: Confronto tra un approccio di *machine learning* (a sinistra) e *deep learning* (a destra) nella categorizzazione di veicoli.

Al momento di scegliere tra machine learning e deep learning occorre considerare se di dispone di una GPU ad alte prestazioni e di una grande quantità di dati etichettati. In caso contrario, è più sensato utilizzare il machine learning anziché il deep learning. Il deep learning è generalmente più complesso, per cui è necessario disporre di almeno alcune migliaia di immagini per ottenere risultati affidabili; con una GPU ad alte prestazioni il modello impiega meno tempo ad analizzare tutte queste immagini.

3.4 Transfer Learning

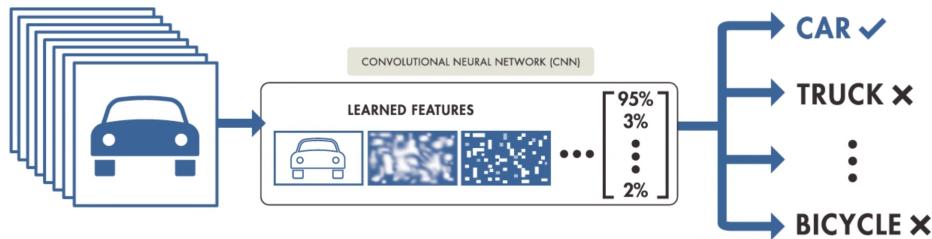
Per addestrare una rete profonda ”da zero” è necessario raccogliere un set di dati etichettati di grandi dimensioni e progettare un’architettura di rete in grado di apprendere le *features* e il modello. Si tratta di un approccio meno comune poiché, considerata la grande quantità di dati e la velocità di apprendimento, l’addestramento di queste reti richiede giorni o settimane.

La maggior parte delle applicazioni di deep learning utilizza l’approccio denominato **transfer learning**, un processo che consiste nell’affinamento di un modello precedentemente addestrato. Si parte da una rete esistente, come AlexNet (nel mio caso) o GoogLeNet, in cui si inseriscono nuovi dati contenenti classi precedentemente sconosciute.

Una volta messa a punto la rete è possibile svolgere una nuova attività con un grande vantaggio: vengono richiesti molti meno dati (infatti vengono elaborate migliaia di immagini anziché milioni) e i tempi di calcolo si riducono a pochi minuti o ad alcune ore.

Il *transfer learning* può essere dunque considerato come un'ottimizzazione che permette progressi rapidi e un miglioramento delle performance nel raggiungimento di un determinato obiettivo.

TRAINING FROM SCRATCH



TRANSFER LEARNING

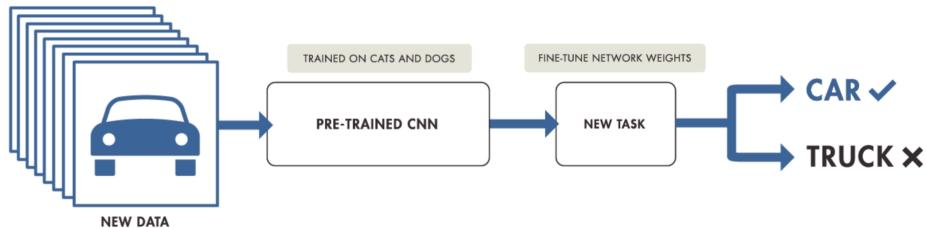


Figura 3.3: Confronto tra una rete allenata "da 0" (*from scratch*) e una *fine-tuned*.

Rimane aperta una domanda: *"Cosa possiamo fare quando la fase di allenamento (training) risulta essere lenta?"* Abbiamo bisogno di algoritmi di ottimizzazione che possano velocizzare l'apprendimento e magari portarci ad ottenere un risultato migliore per la *funzione costo*.

Nella prossima sezione vado a descrivere i due algoritmi di ottimizzazione utilizzati nel training di AlexNet: *sgdm* e *adam*.

4 Algoritmi di ottimizzazione

4.1 Discesa stocastica del gradiente

La **discesa stocastica del gradiente**¹⁵ (SGD) è un *metodo iterativo* per l'ottimizzazione di funzioni differenziabili che si utilizza quando la *cost function*¹⁶ ha la forma di una somma.

Ad ogni iterazione questo algoritmo sostituisce il valore esatto del gradiente della *cost function* con una stima ottenuta valutando il gradiente solo su di un sottoinsieme degli addendi. In generale si ha la necessità di ottimizzare una funzione del tipo:

$$Q(w) = \frac{1}{n} \sum_{i=1}^n \nabla Q_i(w)$$

dove ogni addendo Q_i rappresenta il costo calcolato sull' i -esima osservazione del dataset.

Il metodo di discesa del gradiente standard esegue iterazioni nella forma:

$$w := w - \eta \nabla Q(w) = w - \eta \frac{1}{n} \sum_{i=1}^n \nabla Q_i(w)$$

dove η è un *iperparametro*¹⁷ che controlla l'ampiezza di ogni passo e si definisce *tasso di apprendimento* (*learning rate*).

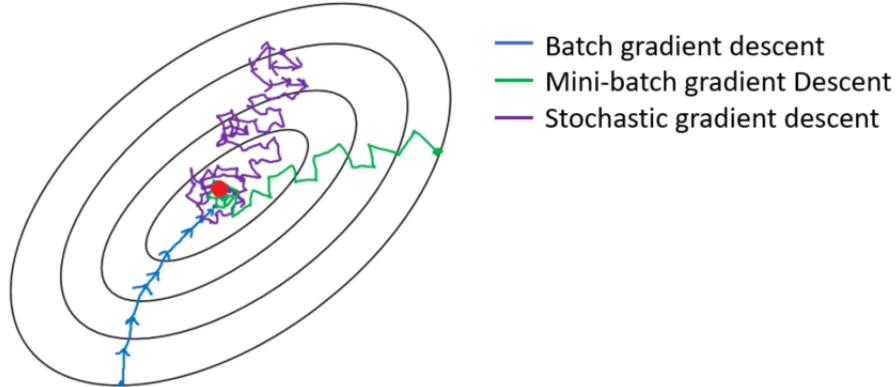


Figura 4.1: Confronto fra i vari metodi esistenti di discesa del gradiente.

Quando il dataset è particolarmente ampio e non esiste un'espressione elementare per il costo, la valutazione dell'intera somma può essere particolarmente onerosa. SGD affronta tale problema in questo modo: il gradiente di $Q(w)$ è approssimato ad ogni iterazione con il gradiente calcolato su un singolo addendo della funzione costo (corrispondente ad un elemento del dataset):

$$w := w - \eta \nabla Q_i(w).$$

¹⁵Approssimazione stocastica del metodo di **discesa del gradiente**, tecnica che consente di determinare i punti di massimo e minimo in una funzione di più variabili: è un metodo per calcolare la migliore direzione lungo la quale è necessario spostarsi per aggiornare il vettore dei pesi e trovare il valore minimo della *loss function*.

¹⁶Questa funzione serve a stimare "quanto male" il modello utilizzato sta performando.

¹⁷Parametro di addestramento.

I parametri vengono aggiornati dopo ogni valutazione e il dataset può essere attraversato diverse volte finché il metodo non converge.

Tipicamente, l'ordine degli elementi è randomizzato ad ogni attraversamento del dataset e il tasso di apprendimento η può essere variato dinamicamente riducendolo con il procedere delle iterazioni (questa procedura è conosciuta con il termine *annealing*). L'aggiornamento dei parametri implica generalmente un andamento molto irregolare della funzione costo¹⁸.

4.1.1 Stochastic Gradient Descent con *momento*

Il *momento* (**momentum**) è un *termine dipendente dalle iterazioni precedenti che viene sommato al gradiente nel tentativo di regolarizzare il movimento nello spazio dei parametri*.

L'aggiornamento dei parametri usa una combinazione lineare del gradiente nell'iterazione corrente e dell'aggiornamento nell'iterazione precedente:

$$\Delta w_{t+1} = \alpha \Delta w_t - \eta \nabla Q_i(w_t)$$

$$w_{t+1} = w_t + \Delta w_{t+1}$$

che risulta in

$$w_{t+1} = w_t - \eta \nabla Q_i(w_t) + \alpha \Delta w_t$$

Questo metodo introduce un secondo iperparametro α per controllare l'effetto del momento (cioè l'importanza dell'iterazione passata). Un valore empirico per tale parametro è solitamente fissato intorno ad $\alpha = 0.9$.

Il nome di questa variante deriva da un'analogia (non del tutto accurata) con il momento lineare in Fisica, in quanto il vettore dei parametri può essere visto come una particella che si muove nello spazio dei parametri, soggetta ad accelerazione da parte di una forza il cui potenziale è espresso dalla funzione costo.

¹⁸Una variante ampiamente utilizzata, che rappresenta un approccio intermedio tra SGD e *batch* GD, è nota come *mini-batch* e prevede di calcolare il gradiente in un sottoinsieme di elementi del dataset ad ogni iterazione, invece che su di un singolo elemento. La dimensione di ogni mini-batch è generalmente dipendente dal contesto: ciò rende l'evoluzione della funzione costo nel corso delle iterazioni meno rumorosa, in quanto il gradiente ad ogni iterazione diventa la media di ogni mini-batch, risultando in un effetto di *smoothing* e riducendo il costo computazionale.

4.2 Adaptive moment estimation: adam

Come ho mostrato nella sezione precedente, il *learning rate* è assolutamente fondamentale per l’allenamento di una rete neurale, ma è anche difficile trovarne il valore ”perfetto”. Per mitigare questo problema, negli anni, sono stati proposti diversi algoritmi: il più famoso ed utilizzato è *adam*.

Adam (che sta per *Adaptive moment estimation*) è un algoritmo di ottimizzazione che può essere utilizzato al posto di SGDM per aggiornare i pesi della rete: l’algoritmo che ho presentato precedentemente mantiene infatti il *learning rate* α costante per ogni aggiornamento dei pesi e tale valore non cambia durante tutta la fase di allenamento.

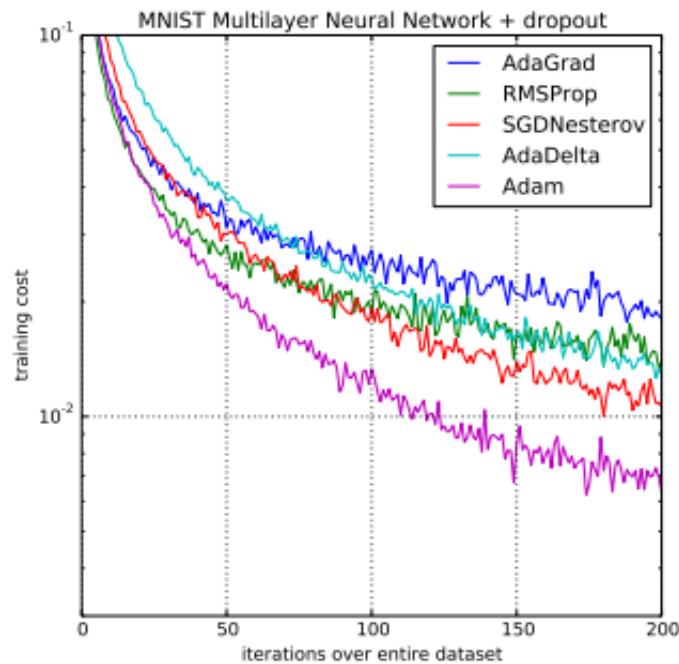


Figura 4.2: Confronto fra *adam* e altri algoritmi di ottimizzazione.

È un algoritmo molto efficiente dal punto di vista computazionale, richiede poca memoria ed è adatto a problemi che utilizzano una grande quantità di dati, in particolare in ambienti con gradienti sparsi o molto rumorosi.

A titolo informativo elenco qui di seguito i parametri che *adam* utilizza:

1. α : *learning rate* o *step size*;
2. β_1 e β_2 sono due iperparametri che controllano il decadimento esponenziale;
3. ϵ : termine di *smoothing* aggiunto ai fini della stabilità numerica¹⁹; in genere è un numero molto piccolo (*e.g.* 10^{-8}) che evita qualsiasi divisione per zero durante l’implementazione.

¹⁹Rappresenta l’accuratezza del risultato.

5 Dropout e L_2 Regularization

La **regolarizzazione** è una tecnica che aiuta a risolvere il problema dell'*overfitting* e riduce la complessità computazionale del modello: lo fa penalizzando la *loss function*.

Che cos'è dunque la *loss function*? Intuitivamente, è un metodo che serve a valutare "quanto bene sta andando il mio algoritmo su quel determinato dataset". Se le prestazioni sono pessime, la *loss function* fornirà in output un numero grande, se le prestazioni sono buone, il numero sarà più piccolo.

Le due tecniche di regolarizzazione più importanti sono:

1. **Dropout**: questa tecnica viene applicata (quando il training set è piccolo) durante la fase di allenamento e consiste nell'eliminazione casuale di unità dalla rete neurale (cioè settando a 0 un certo numero di attivazioni in un layer). La probabilità che un neurone rimanga acceso oppure spento è impostata di *default* a 0.5.

Lo svantaggio di questa tecnica è che la *cost function* non è più definita: ad ogni layer viene infatti associata una probabilità di "perdita" dei suoi nodi. L'idea alla base è che, ad ogni iterazione, viene allenato un modello sempre diverso che utilizza soltanto un sottoinsieme dei suoi neuroni: i neuroni diventano quindi "meno sensibili" alle attivazioni di un neurone specifico dal momento che potrebbe o meno essere spento.

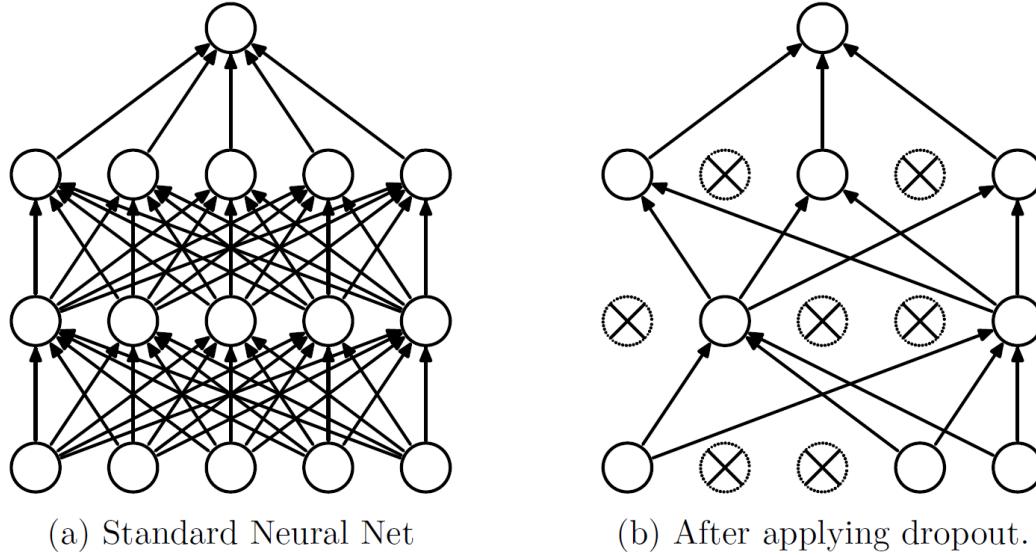


Figura 5.1: Dropout: esempio intuitivo.

2. **L_2 Regularization**: Comincio definendo la *loss function* applicata ad un solo esempio (il termine y rappresenta l'uscita che ci si aspetta, mentre \hat{y} l'uscita reale della rete):

$$\mathcal{L}(y, \hat{y}) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})).$$

La *loss function* calcola la differenza tra il valore che ci si aspetta e il valore reale (sempre per una sola immagine).

La *cost function* è semplicemente la media delle *loss functions* di tutto il dataset:

$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(y^{(i)}, \hat{y}^{(i)})$$

Che cosa fa questo tipo particolare di regolarizzazione? Alla *cost function* viene sommato un valore, cioè

$$\frac{\lambda}{2m} \sum_{i=1}^L \|\mathbf{w}^{[l]}\|_2^2$$

(dove \mathbf{w} è la matrice dei pesi del livello L); tale valore è la media di quella operazione di tutti i livelli, il tutto moltiplicato per λ che è un iperparametro con un valore prefissato (di solito varia da 10^4 a 10^5).

Cosa stiamo facendo? Stiamo normalizzando una matrice facendo la media della *norma* di tutte le matrici.

$$J(\mathbf{w}^{[i]}, b^{[i]}, \dots, \mathbf{w}^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m} \sum_{i=1}^L \|\mathbf{w}^{[l]}\|_2^2$$

dove $\|\mathbf{w}^{[l]}\|_2^2 = \mathbf{w}^{[l]T} \mathbf{w}^{[l]}$.

Riassumendo: si misurano le "lunghezze" delle matrici dei pesi di ogni livello, se ne fa la media e la si moltiplica per λ : il valore lo si somma alla *cost function*.

Risultato: ciò che si ottiene è una nuova *cost function* che penalizza di più gli errori: nella *backpropagation* i pesi verranno modificati in misura maggiore.

6 Reti Neurali Convoluzionali

Le **Reti Neurali Convoluzionali** (in inglese *Convolutional Neural Networks, CNNs*) sono un tipo particolare di reti neurali che utilizza la convezione al posto della semplice moltiplicazione tra matrici in almeno un layer²⁰; i layers sono *strutturati in modo gerarchico* ed hanno una specifica funzione a seconda del tipo (alcuni di questi hanno dei parametri allenabili, altri invece implementano semplicemente una determinata funzione).

Le CNN sono prevalentemente utilizzate per l'analisi di immagini (l'input layer viene infatti direttamente collegato ai pixels dell'immagine).

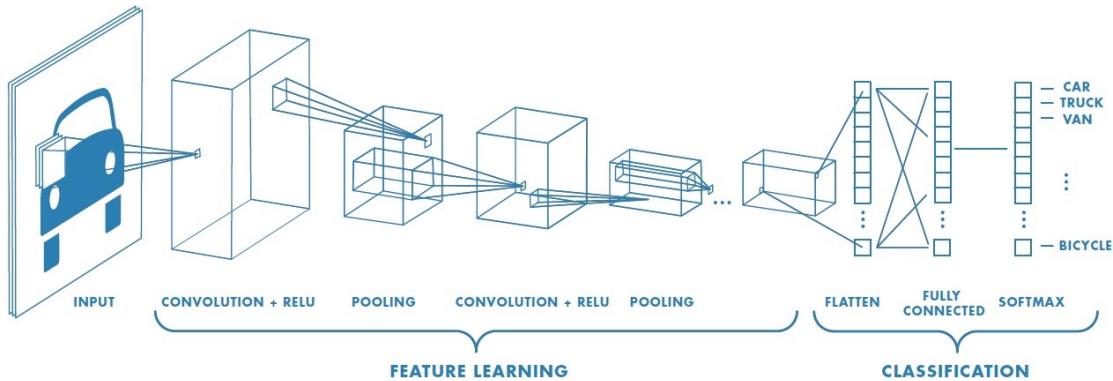


Figura 6.1: Schematizzazione di una CNN in tutta la sua interezza.

Nelle sezioni successive vado a presentare brevemente i layers più importanti di una CNN.

6.1 Convolution layer

I nostri computer "leggono" le immagini come "insieme di pixels" e le esprimono attraverso matrici $N \times N \times 3$, cioè *altezza* \times *larghezza* \times *profondità* (le immagini fanno uso di tre canali, RGB, e questo spiega perché la profondità risulti essere uguale a 3).

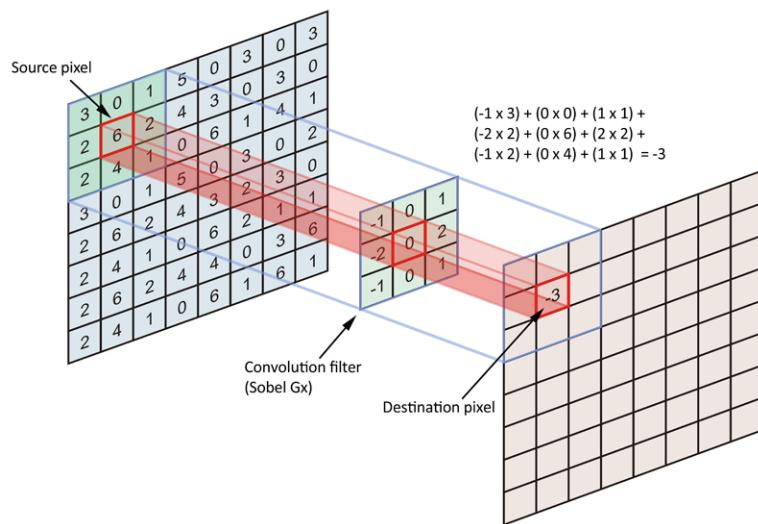


Figura 6.2: Matrice dei pixel 8x8 dell'immagine a cui si è applicato un filtro 3x3: la somma pesata risulta essere uguale a -3.

²⁰Il termine verrà abitualmente utilizzato ed indica lo "strato", che si sta considerando, della rete.

Il **convolution layer** è il principale e più importante: si applica un *filtro digitale* all'immagine tramite un'operazione di convoluzione.

Con *filtro digitale* si intende l'applicazione di una maschera 2D (matrice) di pesi fatta scorrere sulle diverse posizioni dell'input, per ogni posizione viene generato come output un prodotto scalare (cioè una somma pesata) tra la maschera e la porzione coperta dall'input.

Due sono i parametri più importanti da considerare:

1. **Stride**: significa "di quanto far *shiftare* il filtro";
2. **Padding**: per operare il *padding*²¹ è necessario stabilire la "taglia" dello stesso: zero-padding pari a 0 significa che non si ha alcun padding; zero-padding uguale ad 1 significa che tutto il volume di input avrà, per ogni dimensione, un bordo di grandezza 1 di zeri, ecc...

Il parametro di zero-padding permette di controllare anche la taglia del nostro volume di output: viene utilizzato per fare in modo che i valori assegnati ai vari parametri che specificano le dimensioni dei volumi dei layer della rete siano validi per le operazioni che si andranno ad effettuare successivamente.

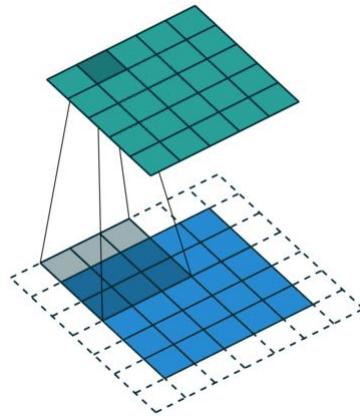


Figura 6.3: Padding con stride=1.

Durante la *forward propagation*²² si trasla, o più precisamente si convolve, ciascun filtro lungo la larghezza e l'altezza del volume di input, producendo una **feature map** bidimensionale per quel filtro. Man mano che si sposta il filtro lungo l'area dell'input si effettua un'operazione di prodotto membro a membro (cioè un prodotto scalare) fra i valori del filtro e quelli della porzione di input al quale è applicato.

²¹Indica lo spessore (in pixel) del bordo.

²²Uno dei metodi più efficaci per il training delle reti neurali è l'algoritmo di **retropropagazione dell'errore** (*error backpropagation*), che aggiorna sistematicamente i pesi delle connessioni tra i vari neuroni, di modo che la risposta della rete si avvicini sempre di più a quella desiderata. L'addestramento di una rete neurale avviene principalmente in due diversi stadi: **forward propagation** e **backward propagation**.

Nella prima fase vengono calcolate tutte le attivazioni dei neuroni della rete, partendo dal primo e procedendo fino all'ultimo layer: durante questa fase i valori dei pesi sono tutti fissati; alla prima iterazione si avranno dei valori di default assegnati. Nella seconda fase l'output reale viene confrontato con l'output desiderato ottenendo così l'errore della rete.

L'errore calcolato viene propagato nella direzione inversa rispetto a quella delle connessioni sinaptiche, ovvero in senso opposto alla prima fase. Al termine della seconda fase, sulla base degli errori appena calcolati, i pesi vengono modificati in modo da minimizzare la differenza tra l'output attuale e l'output desiderato. L'intero procedimento viene successivamente re-iterato con una nuova *forward propagation*.

L'accodamento di tutte queste *feature maps* per tutti i filtri (lungo la dimensione della profondità) forma il volume di output di un layer convoluzionale.

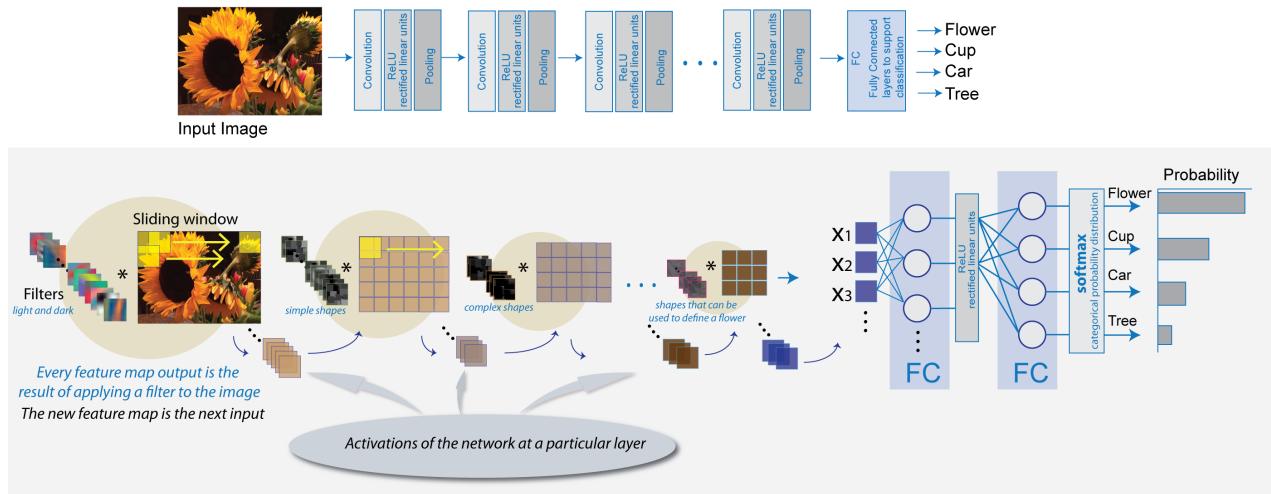


Figura 6.4: Feature map: uno schema rappresentativo.

Ciascun elemento di questo volume può essere interpretato come l'output di un neurone che osserva solo una piccola regione dell'input e che condivide i suoi parametri con gli altri neuroni nella stessa *feature map*, dato che questi valori provengono tutti dall'applicazione del medesimo filtro.

A titolo informativo inserisco qui di seguito la formula che si utilizza per calcolare la dimensione (orizzontale) W_{out} della *feature map* di output date la corrispondente dimensione dell'input W_{in} e la dimensione orizzontale del filtro F :

$$W_{out} = \frac{W_{in} - F + 2 \times Padding}{Stride} + 1.$$

6.2 Pooling layer

Un **Pooling layer** esegue un'*aggregazione* delle informazioni nel volume di input, generando *feature maps* di dimensione inferiore.

L'obiettivo è di conferire *invarianza* rispetto a semplici trasformazioni dell'input mantenendo comunque le informazioni significative ai fini della discriminazione dei pattern; il *pooling layer* è inserito fra *convolution layers*, "tiene sotto controllo" l'**overfitting**²³ e riduce progressivamente la dimensione della rete stessa.

L'aggregazione opera (in generale) nell'ambito di ciascuna *feature map*, in modo che il numero delle *feature maps* nei volumi di input e di output sia lo stesso. Questo tipo di aggregazione non ha parametri o pesi da apprendere.

²³Nei casi in cui l'apprendimento è stato effettuato troppo a lungo o dove è presente uno scarso numero di esempi per l'allenamento, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei casi; in presenza di overfitting, dunque, le prestazioni sui dati di allenamento aumenteranno mentre le prestazioni sui dati non visionati (sconosciuti) saranno peggiori.

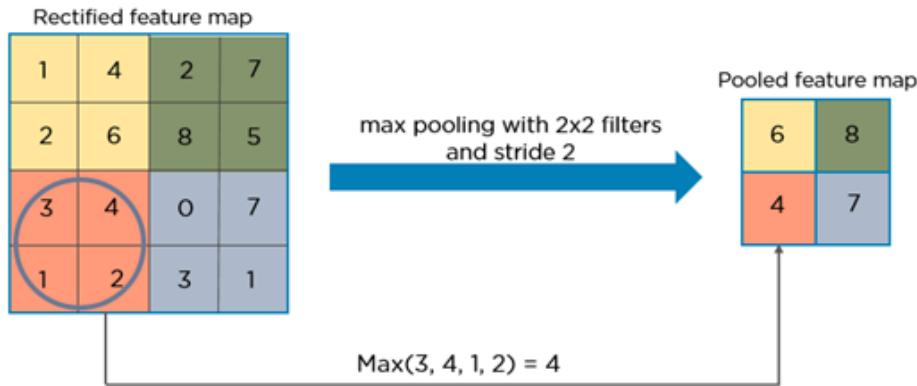


Figura 6.5: Esempio di max-pooling con stride=2.

6.3 ReLu layer

Questo tipo di layer è molto comune in una CNN e viene utilizzato più volte all'interno della stessa (di solito dopo un layer convoluzionale): permette di allenare la rete molto velocemente e fornisce buone prestazioni. Non ha alcun parametro impostabile o allenabile.

La funzione che in genere viene utilizzata è $f(\text{net}) = \max(0, \text{net})$. La derivata vale 0 per valori negativi o nulli di net e 1 per valori positivi: per valori positivi non avviene alcuna saturazione. Questo tipo di layer porta ad attivazioni sparse (significa che una certa quantità di neuroni risulterà "spenta") che conferiscono una maggiore robustezza alla rete.

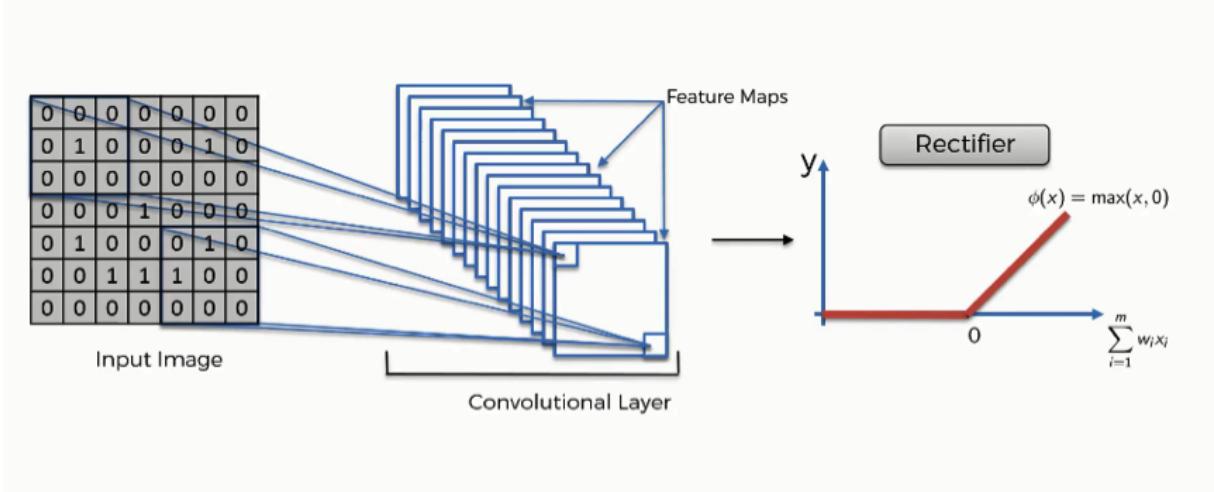


Figura 6.6: Schematizzazione del funzionamento di un ReLu Layer.

6.4 Fully Connected layer e Softmax

In questo tipo di layer ciascun neurone è connesso a tutti i neuroni del layer precedente. L'unico parametro impostabile in questo tipo di layer è il numero dei K neuroni che lo costituiscono.

A livello intuitivo, ciò che questo layer fa è collegare i suoi K neuroni con tutto il volume di input e calcolare l'attivazione di ciascuno dei suoi K neuroni: il suo output sarà un singolo vettore $1 \times 1 \times K$ contenente le attivazioni calcolate.

Il fatto che dopo l'utilizzo di un singolo FC layer si passi da un volume di input (organizzato in 3 dimensioni) ad un singolo vettore di output (in una sola dimensione), fa intuire che dopo il suo utilizzo non si potrà più disporre di alcun layer convoluzionale. La funzione principale dei FC layer

è dunque quella di effettuare un raggruppamento delle informazioni ottenute fino a quel momento, esprimendole con un singolo numero (l'attivazione di uno dei suoi neuroni) che servirà nei calcoli successivi per la classificazione finale.

Una scelta piuttosto comune nelle CNN prevede anche di impiegare un livello finale **Softmax**: è costituito da j neuroni *fully connected* rispetto ai neuroni del livello precedente. Il livello di attivazione del neurone k -esimo si calcola (anziché utilizzare la funzione sigmoide o ReLu) in questo modo:

$$z_k = f(\text{net}_k) = \frac{e^{\text{net}_k}}{\sum_{i=1}^j e^{\text{net}_i}}.$$

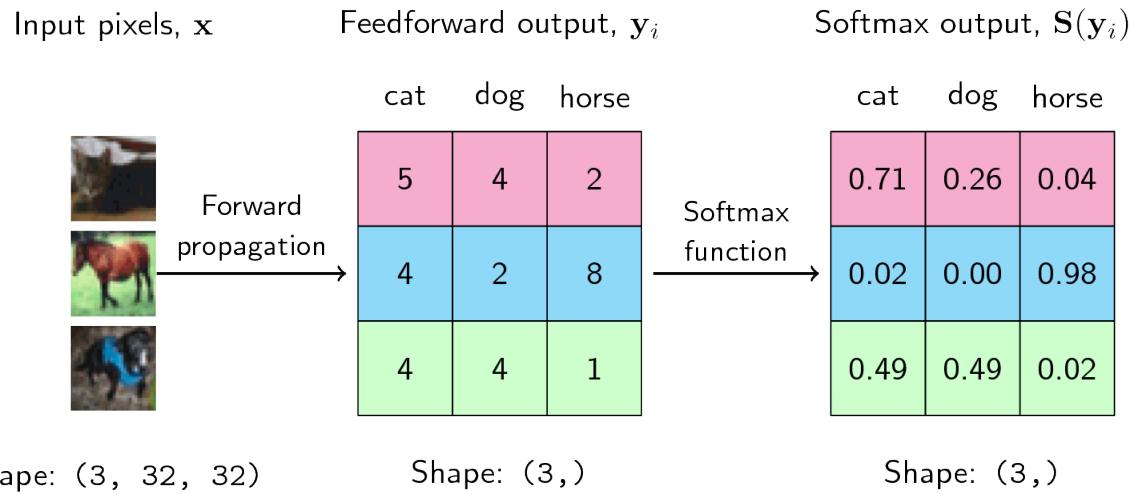


Figura 6.7: Un semplice esempio di applicazione della funzione *Softmax*.

7 AlexNet e diagnosi di Alzheimer: nuovi metodi

Tutto il codice che è stato generato ai fini di questa tesi è stato implementato in MATLAB. I test sono stati eseguiti sul mio PC portatile ASUS ROG GL703GM *Scar Edition*, con processore Intel Core i7-8750H Hexa-Core, Nvidia GeForce GTX 1060 e memoria RAM da 16GB.

Il pre-processing dei dati è stato fatto dall'*Institute of Molecular Bioimaging and Physiology* (IBFM), che ha sede a Milano. I test che ho svolto sono stati fatti su voxel²⁴ 121x145x121.

AlexNet, rete neurale convoluzionale sviluppata da Alex Krizhevsky (computer scientist ucraino), è costituita da 5 Convolution layers e da 3 Fully Connected layers. Sebbene sia stata allenata su più di un milione di immagini e sia in grado di classificare oltre 1000 classi di oggetti (come ad esempio tastiere, mouse, matite e animali), l'approccio utilizzato per lo sviluppo di questa tesi ha previsto la tecnica del transfer learning come suo punto di forza nella classificazione di dataset contenenti le risonanze magnetiche di centinaia di pazienti.

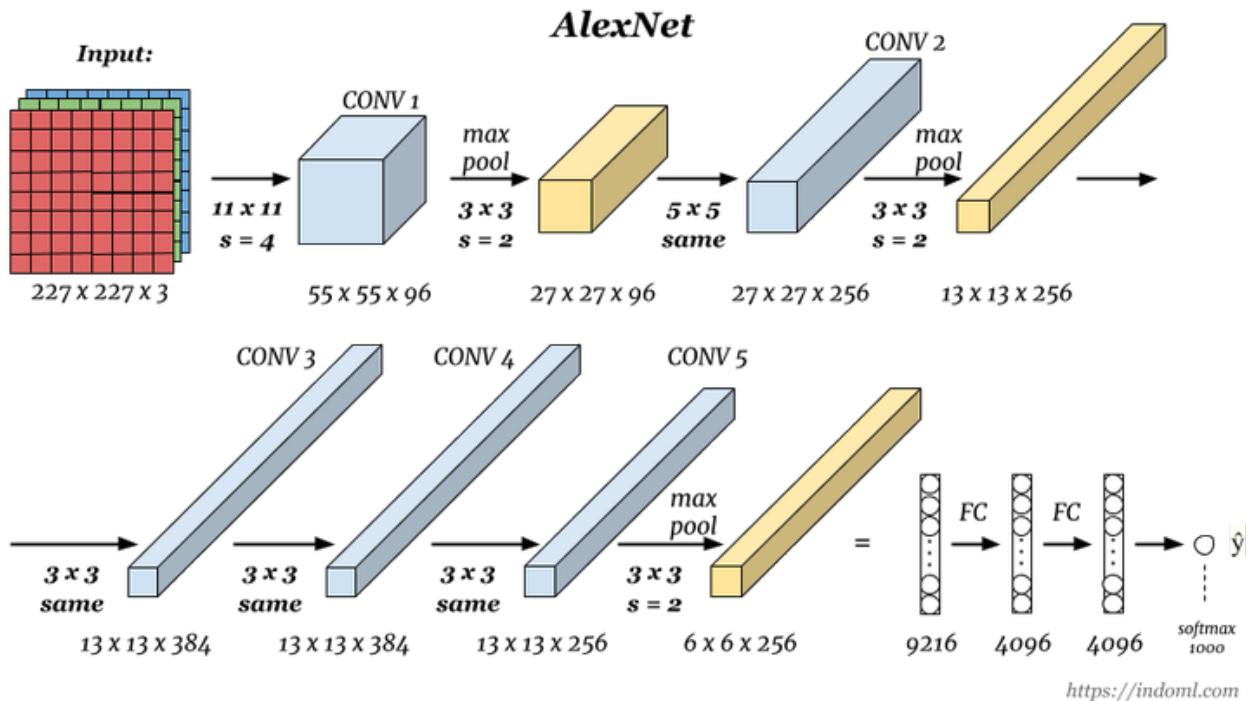


Figura 7.1: Struttura di AlexNet.

7.1 Struttura del codice

A causa della limitata memoria RAM disponibile, anzichè convertire ogni MRI in 100 immagini, ho estratto solamente 6 *slices* (termine che si può tradurre con "sezioni") per ogni MRI ed allenato AlexNet su 3 *folds*²⁵ anzichè 20.

Di seguito vado ad indicare, a grandi linee, la struttura del codice MATLAB comune ai tre metodi sviluppati:

²⁴Il voxel è la controparte tridimensionale del pixel bidimensionale.

²⁵Un fold è un insieme (spesso consecutivo) di record di un dataset.

1. **Definizione della rete:** utilizzo del transfer learning come miglior approccio per il training di AlexNet. L'*input size*²⁶ della CNN è [227 227] (valore di *default*).

Sono stati lasciati inalterati tutti i layers della rete eccetto gli ultimi 3, che sono stati sostituiti e modificati seguendo la documentazione fornita da MathWorks per i problemi di transfer learning:

- *layers(end-2) = fullyConnectedLayer(2, 'WeightLearnRateFactor', 20, 'BiasLearnRateFactor', 20);*
- *layers(end-1) = softmaxLayer;*
- *layers(end) = classificationLayer;*

2. **Istanziazione dei parametri modificabili della rete:** ho svolto diversi test (i risultati migliori sono stati riportati nella **Sottosezione 7.2**) modificando alcuni parametri, come:

- (a) *miniBatchSize*: quante immagini da utilizzare in ogni iterazione;
- (b) *MaxEpochs*: un'*epoca* è un ciclo completo di allenamento per l'intero training dataset.
Nel caso del transfer learning non è necessario aumentare molto tale valore, basta infatti fare uno '*Shuffle*'-'*every-Epoch*', cioè "rimescolare" i dati ad ogni epoca.
- (c) *optimizer*: i test hanno previsto l'utilizzo degli algoritmi di ottimizzazione presentati nella **Sezione 4**: discesa stocastica del gradiente con momento (*sgdm*) e *adam*.

3. **Caricamento dei dataset CN vs MCIC, CN vs AD e MCInc vs MCIC:** concatenazione dei quattro vettori CN-training, CN-testing, ecc... in un vettore unico, altrimenti non avrei potuto lavorarci se fossero stati separati.

Trasformazione di 238 immagini (totali) MRI in 1428 (= 238×6) immagini di dimensione $227 \times 227 \times 3$ (dimensione totale di circa 1GB).

4. **3-Fold Cross-Validation:** consiste nella suddivisione del dataset totale in k parti di uguale numerosità e, ad ogni passo, la k -esima parte del dataset diventa il validation dataset, mentre la restante parte costituisce il training dataset. Così, per ognuna delle k parti, si allena il modello evitando problemi di *overfitting*.

In altre parole, si suddivide il campione osservato in gruppi di egual numerosità, si esclude iterativamente un gruppo alla volta e lo si cerca di predire con i gruppi non esclusi. La k -Fold Cross-Validation è una delle tecniche più utilizzate al fine di verificare l'affidabilità del modello di predizione.

In seguito alla fase di allenamento, ho salvato le *metrics* per valutare la percentuale di corretto riconoscimento delle immagini da parte della rete.

7.1.1 Metodo 1

Questo primo metodo prevede innanzitutto la scelta del dataset (fra i tre sopra elencati) da utilizzare e dell'asse (x , y o z) sul quale svolgere l'analisi.

Applicazione del "punto 1": Definizione della rete.

Applicazione del "punto 2": Istanziazione dei parametri modificabili della rete.

Applicazione del "punto 3": Caricamento dei due dataset da analizzare, concatenazione in un unico vettore.

²⁶La dimensione dell'immagine processata.

A questo punto (con un ciclo *for*) si estrae una singola immagine MRI, la si trasforma in 3 immagini di dimensione $227 \times 227 \times 3$ con l'applicazione del metodo $IMG = mriToCNN(IMG, siz, axis)$.

Cosa rappresentano i parametri di questo nuovo metodo? $IMG=dataset\{i\}$ indica l'estrazione della singola MRI, $siz = [227 227]$ la dimensione dell'input richiesto da AlexNet e $axis$ l'asse che stiamo considerando per quel determinato allenamento.

I parametri scelti per questo metodo sono $nPictures=6$ (che rappresenta il numero di *slices* estratti per ogni MRI) e $gap=2$ (che rappresenta appunto il gap, il "salto", tra un'immagine e l'altra). Questi valori sono stati scelti tenendo sempre presente la limitata potenza computazionale a mia disposizione (aumentare tali valori avrebbe messo a dura prova la GPU), dopodiché si estraggono 6 *slices* in sequenza dal centro (se li avessi estratti partendo dai lati la qualità e la quantità di informazioni utili sarebbe stata assai più limitata) e si applica un *resize* all'immagine di dimensione 121x145 per portarla al valore 227x227 richiesto.

Applicazione del "punto 4": 3-Fold Cross-Validation con salvataggio di metrics e scores.

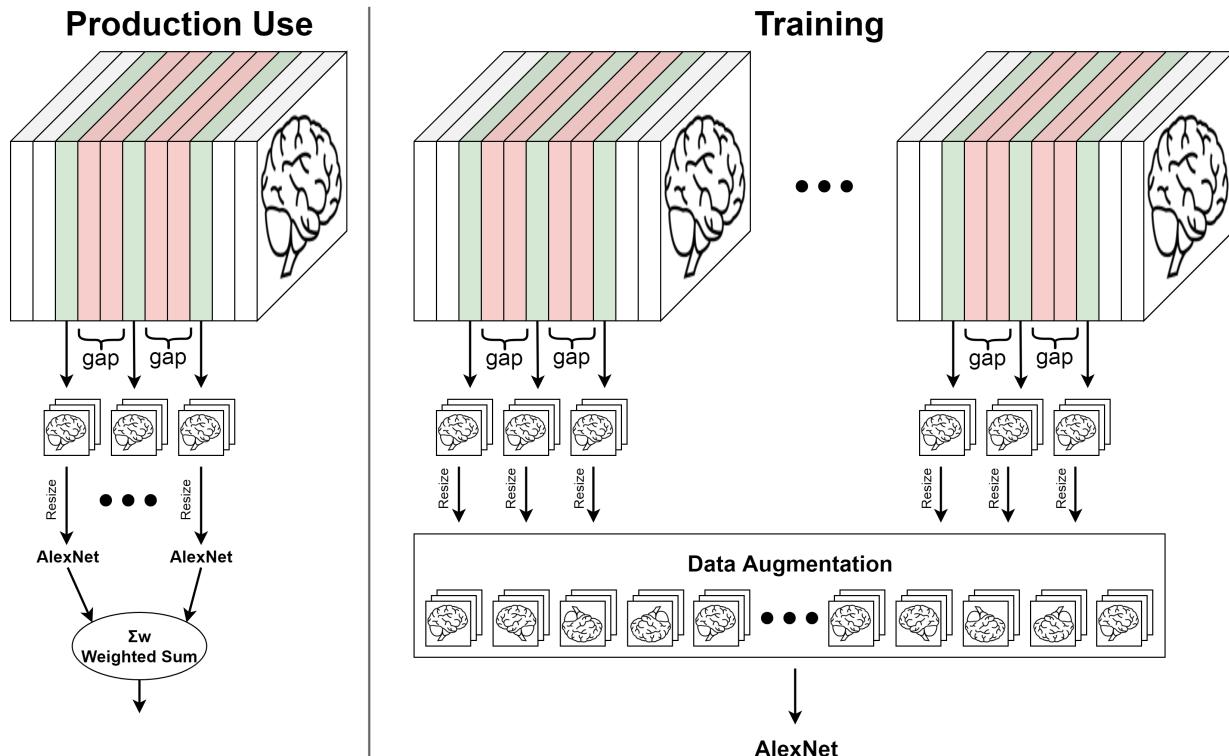


Figura 7.2: Schema del Metodo 1.

7.1.2 Metodo 2

Il secondo metodo che vado a descrivere può essere considerato come una "variante" del primo. Prevede anch'esso la scelta del dataset (fra i tre elencati in precedenza) da utilizzare e dell'asse (x , y o z) sul quale svolgere l'analisi.

Il numero degli *slices* coinvolti si calcola mediante la formula

$$totSlicesInvolved = (2^k + 1 + gap) \times nPictures - gap.$$

Si procede seguendo la struttura del "Metodo 1".

Applicazione del "punto 1": Definizione della rete.

Applicazione del "punto 2": Istanziamento dei parametri modificabili della rete.

Applicazione del "punto 3": Caricamento dei due dataset da analizzare, concatenazione in un unico vettore.

A questo punto (con un ciclo *for*) si estrae una singola immagine MRI, la si trasforma in 3 immagini di dimensione $227 \times 227 \times 3$ con l'applicazione del metodo $IMG = mriToCNN(IMG, siz, axis, nPictures, gap, k)$. I primi tre parametri sono gli stessi già visti precedentemente; *nPictures* indica il numero di sezioni da estrarre da ogni MRI, *gap* lo "spazio" tra le sezioni e *k* può variare.

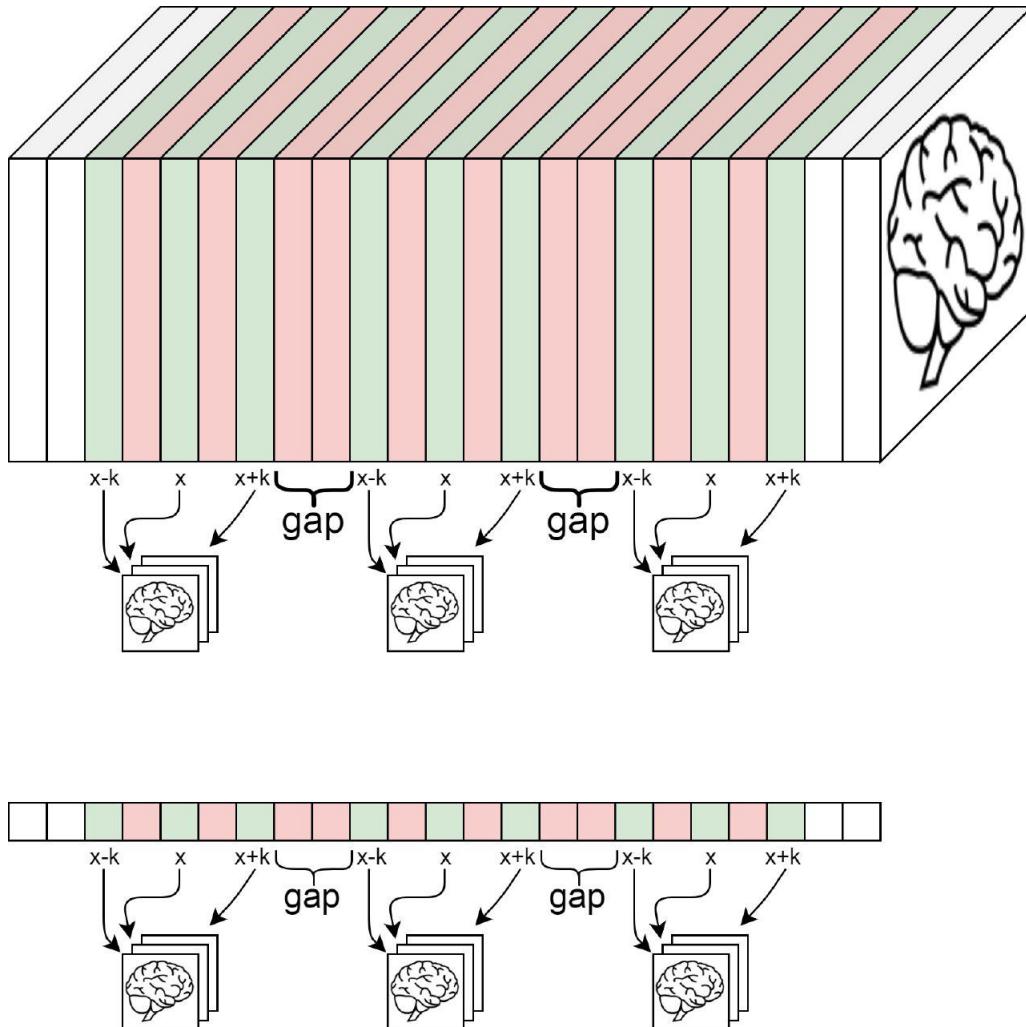


Figura 7.3: Schema del Metodo 2.

Per questo metodo i valori scelti sono: $k=1$, $gap=2$ e $nPictures=6$.

Essendo "di default" una sezione composta da 3 *slices* è possibile fare alcune considerazioni al variare di *k*: se $k=0$ significa che tutti i 3 canali della sezione sono lo stesso *slice*; se $k=1$ vuol dire che vengono presi 3 *slices* consecutivi; se infine $k=2$ vuol dire che c'è 1 *slice* fra 3.

Applicazione del "punto 4": 3-Fold Cross-Validation con salvataggio di *metrics* e *scores*.

7.1.3 Metodo 3

Il terzo ed ultimo metodo prevede un approccio abbastanza diverso da quanto visto finora.

Innanzitutto è necessario scegliere il dataset (fra i tre elencati precedentemente) sul quale performare il training.

Questo metodo lavora sui tre assi contemporaneamente.

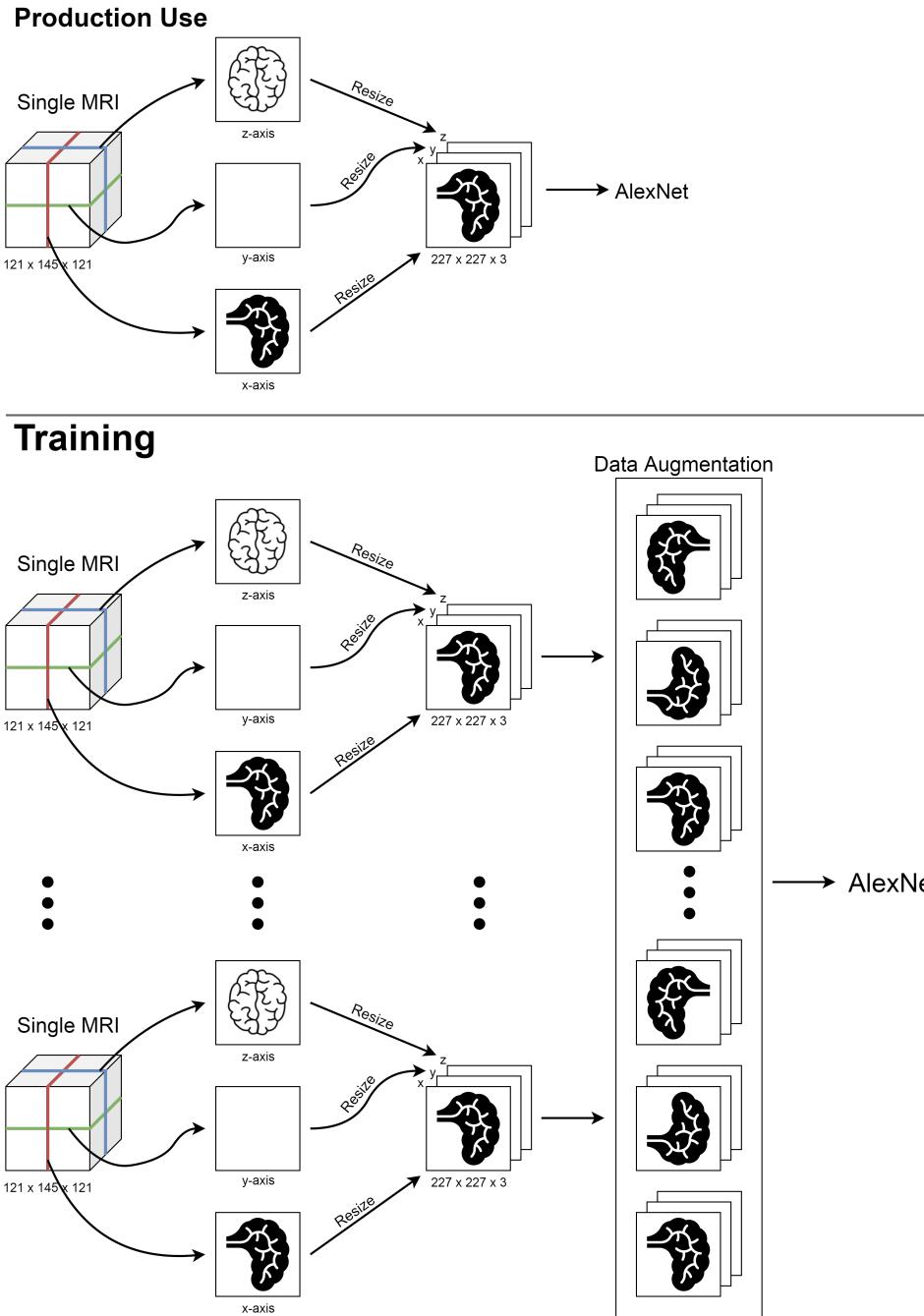


Figura 7.4: Schema del Metodo 3.

Considerando una singola MRI (di dimensioni $121 \times 145 \times 121$, sezione "Production Use" della **figura 7.4**) si estraie l'immagine centrale per ogni asse. L'immagine estratta dall'asse x appartiene al canale R, quella dall'asse y al canale G ed infine quella estratta dall'asse z al canale B.

7.2 Risultati dei test

Ho svolto numerosi test per valutare gli effettivi potenziale ed efficacia dei nuovi metodi presentati. Nel seguito vado ad elencare gli iperparametri utilizzati nelle fasi di training (indicando, di volta in volta, quali valori sono stati spesso assegnati).

Per ogni metodo di ogni dataset ho elencato, all'interno di una tabella, i valori TP (True Positive), TN (True Negative), FP (False Positive), FN (False Negative) rilevati da AlexNet, la percentuale di *accuracy* dopo ogni allenamento ed infine la *AverageAccuracy* (corrispondente alla media aritmetica dei tre risultati dei test fatti).

Osservazione: In una situazione ideale ci si aspetta che un test sia in grado di discriminare perfettamente due popolazioni (in questo caso: "sani" e "malati") non sovrapponibili. In realtà quello che avviene è che le due popolazioni si sovrappongono in parte: il test necessariamente identificherà come positivi alcuni soggetti non malati (Falsi Positivi, FP) e come negativi alcuni soggetti invece malati (Falsi Negativi, FN).

Vado ora ad elencare tutti i parametri con relativi valori assegnati nel codice MATLAB:

1. $folds = 3$;
2. $miniBatchSize$ è il parametro che ho fatto variare più spesso (assieme a $maxEpochs$): si passa da un minimo di 20 ad un massimo di 64 (sono stati eseguiti test anche con valori uguali a 24 e 30);
3. $learningRate = 10^{-4}$ oppure 10^{-5} ;
4. $maxEpochs$, a seconda delle dimensioni del dataset, varia da 15 a 20, da 30 a 50 fino addirittura a 230: a causa dell'*overfitting* l'allenamento con 230 epoche (alcune ore di training) ha portato a risultati fallimentari o comunque molto meno performanti di quelli eseguiti, sempre con gli stessi parametri, con un numero molto inferiore di epoche.
5. $optimizer = 'sgdm'$ oppure $'adam'$ (gli algoritmi di ottimizzazione presentati nella **Sezione 4**);
6. $"L2Regularization"$, 0.0001;
7. $"Momentum"$, 0.889;
8. $'Shuffle'$, $'every-epoch'$.

7.2.1 Metodo 1

1. ***CN vs AD***

Asse x	TP	17	41	11
	TN	42	30	46
	FP	24	6	14
	FN	7	13	19
	accuracy	0.6556	0.7889	0.6333

AverageAccuracy = 0.6926

<i>Asse y</i>	TP	23	36	20	<i>AverageAccuracy = 0.7407</i>
	TN	45	28	48	
	FP	21	8	12	
	FN	1	18	10	
	<i>accuracy</i>	0.7556	0.7111	0.7556	
<i>Asse z</i>	TP	17	43	9	<i>AverageAccuracy = 0.6889</i>
	TN	43	26	48	
	FP	23	10	12	
	FN	7	11	21	
	<i>accuracy</i>	0.6667	0.7667	0.6333	

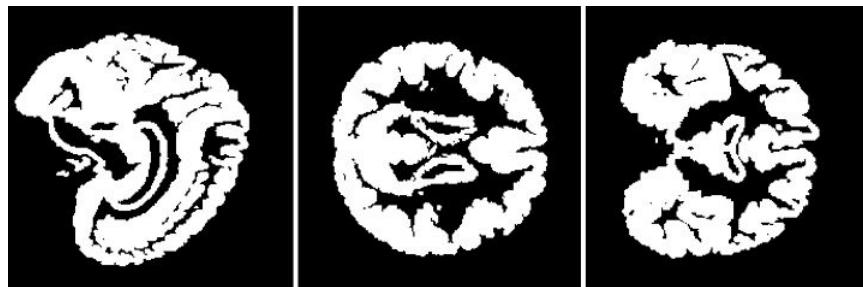


Figura 7.5: Input di AlexNet (a sinistra l'asse *x*, al centro l'asse *y* e a destra l'asse *z*) in seguito alla fase di training per il dataset ***CN vs AD***.

2. ***CN vs MCIC***

<i>Asse x</i>	TP	2	4	1	<i>AverageAccuracy = 0.6587</i>
	TN	45	42	44	
	FP	3	6	4	
	FN	16	20	23	
	<i>accuracy</i>	0.7121	0.6389	0.6250	
<i>Asse y</i>	TP	7	11	8	<i>AverageAccuracy = 0.6999</i>
	TN	39	41	41	
	FP	9	7	7	
	FN	11	13	16	
	<i>accuracy</i>	0.6970	0.7222	0.6806	
<i>Asse z</i>	TP	2	9	5	<i>AverageAccuracy = 0.6801</i>
	TN	41	46	40	
	FP	7	2	8	
	FN	16	15	19	
	<i>accuracy</i>	0.6515	0.7639	0.6250	



Figura 7.6: Input di AlexNet (a sinistra l'asse x , al centro l'asse y e a destra l'asse z) in seguito alla fase di training per il dataset **CN vs $MCIC$** .

3. $MCInc$ vs $MCIC$

<i>Asse x</i>	TP	10	3	0	<i>AverageAccuracy = 0.6278</i>
	TN	28	30	42	
	FP	14	6	0	
	FN	8	21	18	
	<i>accuracy</i>	0.6333	0.5500	0.7000	
	TP	6	5	12	
<i>Asse y</i>	TN	36	35	25	<i>AverageAccuracy = 0.6611</i>
	FP	6	1	17	
	FN	12	19	6	
	<i>accuracy</i>	0.7000	0.6667	0.6167	
	TP	12	12	12	
<i>Asse z</i>	TN	30	31	23	<i>AverageAccuracy = 0.6667</i>
	FP	12	5	19	
	FN	6	12	6	
	<i>accuracy</i>	0.7000	0.7167	0.5833	

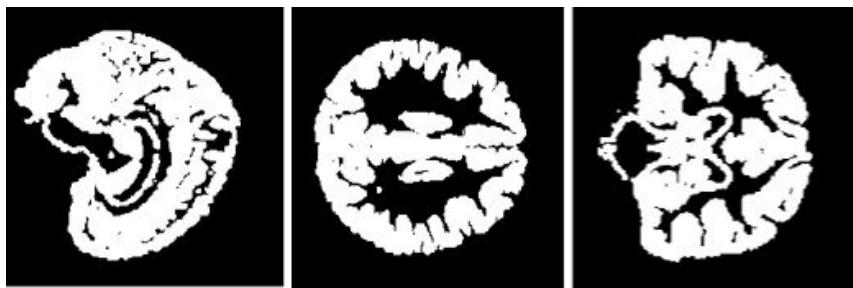


Figura 7.7: Input di AlexNet (a sinistra l'asse x , al centro l'asse y e a destra l'asse z) in seguito alla fase di training per il dataset **$MCInc$ vs $MCIC$** .

7.2.2 Metodo 2

1. *CN vs AD*

<i>Asse x</i>	TP	11	51	8	<i>AverageAccuracy = 0.7889</i>
	TN	62	23	58	
	FP	4	13	2	
	FN	13	3	22	
	<i>accuracy</i>	0.8111	0.8222	0.7333	
<i>Asse y</i>	TP	22	39	15	<i>AverageAccuracy = 0.7185</i>
	TN	42	29	47	
	FP	24	7	13	
	FN	2	15	15	
	<i>accuracy</i>	0.7111	0.7556	0.6889	
<i>Asse z</i>	TP	12	42	9	<i>AverageAccuracy = 0.7037</i>
	TN	47	28	52	
	FP	19	8	8	
	FN	12	12	21	
	<i>accuracy</i>	0.6556	0.7778	0.6778	

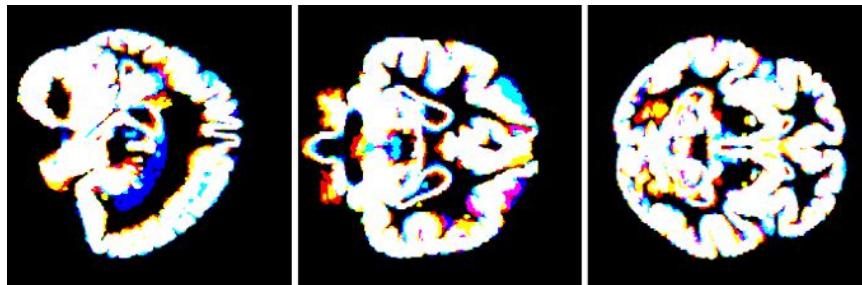


Figura 7.8: Input di AlexNet (a sinistra l'asse *x*, al centro l'asse *y* e a destra l'asse *z*) in seguito alla fase di training per il dataset *CN vs AD*.

2. *CN vs MCIc*

<i>Asse x</i>	TP	8	9	6	<i>AverageAccuracy = 0.6978</i>
	TN	44	36	43	
	FP	4	12	5	
	FN	10	15	18	
	<i>accuracy</i>	0.7879	0.6250	0.6806	
<i>Asse y</i>	TP	6	6	5	<i>AverageAccuracy = 0.6675</i>
	TN	40	40	43	
	FP	8	8	5	
	FN	12	18	19	
	<i>accuracy</i>	0.6970	0.6389	0.6667	

Asse <i>z</i>	TP	5	15	5
	TN	43	39	36
	FP	5	9	12
	FN	13	9	19
	<i>accuracy</i>	0.7273	0.7500	0.5694
	<i>AverageAccuracy</i> = 0.6822			

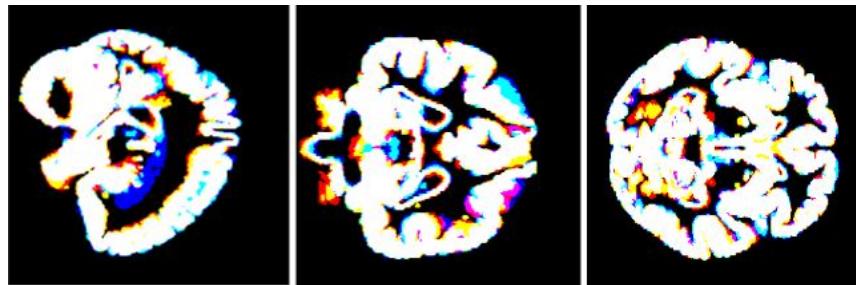


Figura 7.9: Input di AlexNet (a sinistra l'asse *x*, al centro l'asse *y* e a destra l'asse *z*) in seguito alla fase di training per il dataset ***CN vs MCIc***.

3. ***MCIInc vs MCIC***

Asse <i>x</i>	TP	8	14	5
	TN	35	15	35
	FP	7	21	7
	FN	10	10	13
	<i>accuracy</i>	0.7167	0.4833	0.6667
	<i>AverageAccuracy</i> = 0.6222			
Asse <i>y</i>	TP	3	11	4
	TN	40	19	37
	FP	2	17	5
	FN	15	13	14
	<i>accuracy</i>	0.7167	0.5000	0.6833
	<i>AverageAccuracy</i> = 0.6333			
Asse <i>z</i>	TP	8	11	14
	TN	37	20	25
	FP	5	16	17
	FN	10	13	4
	<i>accuracy</i>	0.7500	0.5167	0.6500
	<i>AverageAccuracy</i> = 0.6389			

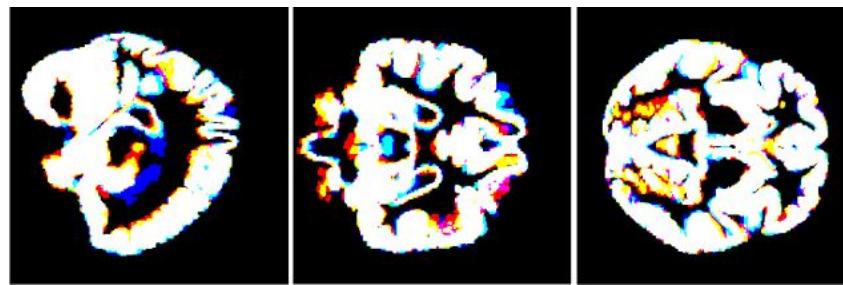


Figura 7.10: Input di AlexNet (a sinistra l'asse *x*, al centro l'asse *y* e a destra l'asse *z*) in seguito alla fase di training per il dataset ***MCIInc vs MCIC***.

7.2.3 Metodo 3

1. *CN vs AD*

TP	4	6	3
TN	7	6	9
FP	4	0	1
FN	0	3	2
<i>accuracy</i>	0.7333	0.8000	0.8000

AverageAccuracy = 0.7778

2. *CN vs MCIC*

TP	2	2	1
TN	7	7	5
FP	1	1	3
FN	1	2	3
<i>accuracy</i>	0.8182	0.7500	0.5000

AverageAccuracy = 0.6894

3. *MCInc vs MCIC*

TP	0	2	0
TN	7	4	6
FP	0	2	1
FN	3	2	3
<i>accuracy</i>	0.7000	0.6000	0.6000

AverageAccuracy = 0.6333

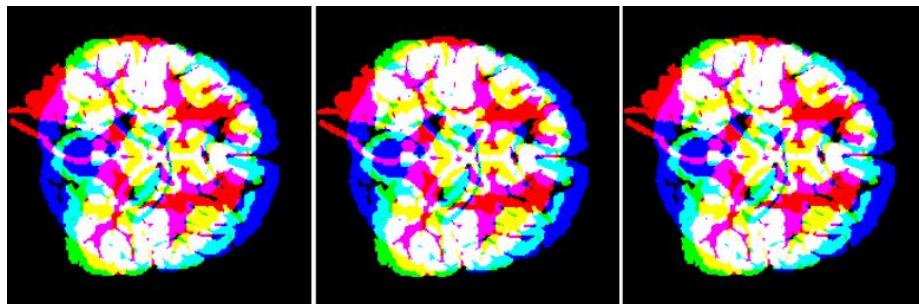


Figura 7.11: Input di AlexNet in seguito alla fase di training per il dataset *CN vs AD* a sinistra, *CN vs MCIC* al centro e *MCInc vs MCIC* a destra.

8 Conclusioni

Con questa tesi ho affrontato un problema complesso con l'ausilio del *Deep Learning*. L'obiettivo è stato riconoscere, con la maggior percentuale di *accuracy* possibile (da parte di AlexNet) un soggetto malato di Alzheimer da uno, invece, sano.

I risultati (esclusi pochi test sfortunatamente insufficienti) hanno dimostrato il grande potenziale del *transfer learning* e, se vogliamo, dell'**Intelligenza Artificiale** in generale, disciplina che, sebbene nata più di settant'anni fa grazie al fondamentale contributo di Alan Turing, si sta velocemente imponendo come fondamentale e determinante nelle nostre vite.

Sta a noi nuove generazioni studiare questa affascinante materia, imparare ad evolvere con essa ed essere in grado di superare qualsiasi sfida si presenti all'*essere umano* in quanto tale, come ad esempio la diagnosi precoce e la conseguente cura di una malattia che sarà ancora per poco inaffrontabile.

Ogni tecnologia sufficientemente avanzata è indistinguibile dalla magia.

Arthur C. Clarke

9 Bibliografia

1. Loris Nanni, Nicolò Zaffonato, et al. *An Ensemble of Classifiers for early diagnosis of Alzheimer's disease*, 2017.
2. Walter H. Pitts Warren S. McCulloch. *A logical calculus of the ideas immanent in nervous activity*. *Bulletin of Mathematical Biophysics*, (Volume 5):115–137, 1943.
3. Bengio Y., Courville A., Goodfellow I., *Deep Learning*, The MIT Press, Cambridge (MA)-Londra 2017.
4. N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, et al. *Convolutional neural networks for medical image analysis: Full training or fine tuning?* IEEE Transactions on Medical Imaging, maggio 2016.
5. H. Shin, H. Roth, M. Gao, et al. *Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning*, IEEE transactions on medical imaging (TMI), 2016.
6. Shuqiang Wang, Yanyan Shen, et al. *Automatic Recognition of Mild Cognitive Impairment from MRI Images Using Expedited Convolutional Neural Networks*, 2017.
7. Saman Sarrafa, Ghassem Tofighic. *DeepAD: Alzheimer's Disease Classification via Deep Convolutional Neural Networks using MRI and fMRI*, 18 agosto 2016.
8. Karim Aderghal, Jenny Benois-Pineau, et al. *FuseMe: Classification of sMRI images by fusion of Deep CNNs in 2D+ ϵ projections*, giugno 2017.
9. Tanya Gluzman, Orly Liba. *Hidden Cues: Deep Learning for Alzheimer's Disease Classification*, 2016.