Francesca Nocentini

Tecniche di deep learning per la classificazione di immagini biomedicali

Relatore:
Prof. Gianluca Reali

Perugia, Anno Accademico 2020/2021
Università degli Studi di Perugia
Corso di laurea triennale in Ingegneria Informatica ed Elettronica
Dipartimento di Ingegneria

0. Indice

1 Introduzione **4**

2 Machine learning e reti neurali **5**

2.1 L'importanza dell'apprendimento 5

2.2 Neuroni artificiali e deep learning 8

2.3 Addestramento di una rete 13

2.4 Overfitting e underfitting 17

2.5 Classificazione 19

3 Reti neurali convoluzionali **21**

3.1 Funzionamento generale 21

3.2 Architettura generale di una CNN 22

3.2.1 Convoluzione 22

3.2.2 Padding 25

3.2.3 Funzioni di attivazione 25

3.2.4 Strati di subsampling: Pooling Layers 26

3.2.5 Fully connected layers 27

4 Ambiente di lavoro **29**

4.1 Python 29

4.2 Tensorflow e Keras 29

5 Implementazione delle reti e prove sperimentali **30**

5.1 Obiettivo 30

5.2 CNN per la rilevazione della polmonite 31

5.2.1 Il dataset 31
5.2.2 Setup iniziale 32
5.2.3 Definizione degli iperparametri 33
5.2.4 Definizione e compilazione del modello 34
5.2.5 Creazione dei set di training e validation traminte l'uso dell'image flowing 38
5.2.6 Fase di fitting 40
5.2.7 Predizioni 41
5.2.8 Considerazioni finali 44
5.3 CNN per la classificazione di risonanze magnetiche cerebrali 46

6 Conclusioni **47**

7 Codice **48**

1. Introduzione

Al giorno d'oggi, con l'avvento delle tecnologie di imaging biomedico, il numero di immagini che sono state catturate ed archiviate giorno dopo giorno negli ospedali e nei laboratori sta crescendo sempre di più. Con questo però, di pari passo all'avanzamento tecnologico, che preveda sistemi più robusti e all'avanguardia affinchè vengano raggiunti gli obiettivi di diagnosi e classificazione di vari tipi di patologie. Sulla scia di ciò, per assistere medici e specialisti, queste immagini possono essere usate ed utilizzate per allenare sistemi intelligenti. Pertanto in virtù di questa grande quantità di immagini mediche (ecografie, mammografie, MRI...), l'uso di metodi basate sulle *big data technologies*, come il machine learning (ML) e l'Intelligenza Artificiale è diventato fondamentale, anche e soprattutto come supporto all'equipe medica. Sono stati dunque proposti negli anni dei metodi per automatizzare il processo di analisi di immagine medica. Nel presente lavoro di tesi saranno analizzati ed utilizzati metodi di classificazione di immagini usando metodi di deep learning. Questi ultimi non sono altro che un'evoluzione dei metodi di ML atti a trattare dati di grande cardinalità e complessità.
In particolare in una prima parte verrà fatta una discriminazione tra soggetti affetti da polmonite e soggetti sani tramite l'utilizzo di radiografie al petto; in una seconda parte invece il sistema viene allenato in modo tale da poter classificare immagini di risonanze magnetiche cerebrali tra 4 diverse diagnosi per il soggetto nell'immagine: glioma, meningioma, tumore ipofisario e soggetto sano.

2. Machine learning e reti neurali

2.1 L'importanza dell'apprendimento

Le reti neurali sono alla base del deep learning, che è un sottocampo del machine learning. Questa ultima branca è fondamentale nello studio e nello sviluppo delle Intelligenze Artificiali, sistemi basati sull'apprendimento automatico partendo dallo studiare i dati in input per fornire dati più vicini possibile a quelli desiderati. Sostanzialmente un algoritmo di machine learning è un algoritmo capace di apprendere dai dati.

"Si dice che un programma apprende dall'esperienza E con riferimento a alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E."

Tom M. Mitchell[1]

Il compito principale del machine learning è che una macchina sia in grado di generalizzare dalla propria esperienza. Per generalizzazione si intende l'abilità di una macchina di portare a termine in maniera accurata esempi o compiti nuovi, che non ha mai affrontato, dopo aver fatto esperienza su un insieme di dati di apprendimento.
La macchina ha il compito di costruire un modello probabilistico generale dello spazio delle occorrenze di un determinato fenomeno da apprendere tramite dei *training examples*, in maniera tale da essere in grado di produrre previsioni sufficientemente accurate

[1]Tom Michael Mitchell è un computer scientist americano e professore universitario. É conosciuto per aver contribuito attivamente allo sviluppo degli studi sul machine learining e le intelligenze artificiali.

quando sottoposta a nuovi casi. Proprio a questo proposito, al fine di valutare la bontà di un algoritmo di machine learning, dobbiamo stimare un andamento qualitativo della sua performance P, la quale molto spesso è specifica per un determinato compito T che il sistema deve compiere.

I compiti dell'apprendimento automatico vengono tipicamente classificati in tre ampie categorie, sulla base dell'esperienza E a cui sono sottoposti nel processo di apprendimento, dunque a seconda della natura del "segnale" utilizzato per l'apprendimento o del "feedback" disponibile al sistema di apprendimento. Queste categorie, anche dette paradigmi, sono:

- **apprendimento supervisionato**, in cui al modello vengono forniti degli esempi nella forma di possibili input e output desiderati e l'obiettivo è quello di estrarre una regola generale che associ l'input all'output corretto;
- **apprendimento non supervisionato**, in cui il modello ha lo scopo di trovare una struttura negli input forniti, senza che gli input vengano etichettati in alcun modo;
- apprendimento per rinforzo, il quale punta a realizzare agenti autonomi in grado di scegliere azioni da compiere per il conseguimento di determinati obiettivi tramite interazione con l'ambiente in cui sono immersi.

Il tipo di apprendimento di interesse per il lavoro di tesi è quello supervisionato, in quanto si utilizzano coppie input-output rappresentate da immagini e label che le identificano.
Tra i compiti più importanti dell'apprendimento supervisionato vi sono la classificazione e la regressione, i quali si distinguono a seconda di come viene considerato l'output del sistema di apprendimento. Nella prima gli output sono divisi in due o più classi e il sistema di apprendimento deve produrre un modello che assegni gli input non ancora visti a una o più di queste. Si dice che i valori assunti dall'ouput sono *qualitativi*. Nella seconda invece si può dire che a differenza della prima i dati in output sono continui, cioè non riguardano una categorizzazione. Si dice i valori assunti dall'output sono *quantitativi*, in quanto restituiscono la stima di una misura.
La classificazione è il compito su cui verte il presente lavoro di tesi. Per tale scopo, la performance P dell'algoritmo di machine learning si valuta misurando l'accuratezza del modello, la percentuale di esempi per i quali il modello elabora un output corretto sul totale delle osservazioni. Un altro parametro di riferimento può essere il rate di errore, definito invece come la proporzione di esempi per cui il sistema elabora un output

sbagliato. E' importante verificare come l'algoritmo riesca a valutare dati che non abbia mai visto. Misure di performance vengono effettuate usando un insieme di dati chiamato insieme di *test*. Questo insieme di dati è, di solito, diverso dall'insieme di caratteristiche usate per allenare il sistema (insieme di *addestramento* o *training*) affinchè si possano fare valutazioni più precise ed indipendenti su quanto sia stato efficace l'addestramento. Di solito l'esperienza E corrisponde ad un intero *dataset*, una collezione di dati (che possono essere sotto forma di vettori, immagini, suoni ecc.) suddiviso in classi con ogni elemento associato ad una ed una sola di queste. Tali dati, essendo utilizzati in fase di addestramento, è estremamente importante che siano costruito in maniera corretta. Dunque il punto di partenza per un buon addestramento è sicuramente un buon dataset.

Come accennato sopra, l'apprendimento supervisionato è una forma di apprendimento automatico in cui al sistema da allenare si sottopone in input un insieme di addestramento formato da esempi etichettati con il rispettivo valore di output desiderato, e tra gli input e output si vuol trovare una corrispondenza. Questo processo fa tipicamente riferimento a tecniche di tipo statistico. Esso consiste nell'osservazione di una quantità di esempi di un vettore casuale $\mathbf{x}$ a cui è associato un vettore $\mathbf{y}$ per poi apprendere come predire $\mathbf{y}$ da $\mathbf{x}$, stimando una distribuzione di probabilità condizionata $p(\mathbf{y}|\mathbf{x})$[2] per l'output, che consenta di ottenere anche valori di confidenza per l'algoritmo. I valori di $\mathbf{y}$ fungono da guida per indirizzare e migliorare l'apprendimento di contro al caso non supervisionato in cui l'algoritmo deve trattare i dati senza l'ausilio di questa guida. Lo scopo degli algoritmi di machine learning è di derivare, a partire dal training set, una distribuzione di probabilità $p(\mathbf{y}|\mathbf{x})$ per l'output, che consenta ad esempio di ottenere anche dei valori di confidenza.
Un esempio di apprendimento supervisionato è il training di un sistema col compito di riconoscimento delle immagini. In questo caso dobbiamo specificare quale oggetto appare in ogni foto. Possiamo fare questo con un codice numerico che caratterizzi le varie classi, per esempio usando 0 per le persone, 1 per le macchine, 2 per i gatti, e così via, a seconda di quello che è il caso di interesse.

[2]Date due variabili aleatorie X e Y, la distribuzione condizionata di Y dato X è la probabilità di Y quando è conosciuto il valore assunto da X

2.2 Neuroni artificiali e deep learning

Le *reti neurali* si riferiscono storicamente alla rete di neuroni che si trovano nel cervello dei mammiferi, e la loro struttura ne ha ispirato gli algoritmi. I neuroni sono le unità fondamentali di calcolo, e sono connessi tra loro in reti per processare dati. Essi rispondono a stimoli esterni, così come le reti neurali prendono dati in input, si allenano nel riconoscere dei pattern ripetuti all'interno dei dati, e sono in grado di predire l'output per un set di dati simili. La corteccia cerebrale umana contiene circa 1010 neuroni. Sono collegati tra loro da filamenti nervosi (assoni) che si diramano e terminano in sinapsi, connessioni verso altri neuroni. Le sinapsi connettono verso i dendriti, diramazioni che si estendono dal corpo della cellula neurale e sono atti a ricevere impulsi da altri neuroni nella forma di segnali elettrici. Nelle reti neurali biologiche lo spessore dei dendriti definisce il peso ad esso associato. La rete risultante di neuroni tra loro connessi nella corteccia cerebrale è responsabile del processare immagini, audio e dati sensoriali. In presenza di una differenza di potenziale fra esterno e l'interno della cellula, il neurone si attiva trasmettendo un impulso elettrico attraverso il suo assone che provoca la liberazione di un neurotrasmettitore.
Nelle reti neurali artificiali, il modo in cui le informazioni sono processate e i segnali sono trasferiti sono ampiamente idealizzati da tali concetti.

Il primo modello computazionale di neurone è stato proposto nel 1943 da Warren MuCulloch e Walter Pitts [3]. McCulloch and Pitts hanno pensato ad un modello di neurone come un'unità di *threshold* (soglia) binaria. Essa possiede due possibili stati: attivo o inattivo.
Per ottenere il segnale di output il neurone elabora una somma pesata degli input: se la somma eccede un certo valore di threshold, lo stato del neurone risulta attivo, altrimenti inattivo. Il modello, in intervalli discreti di tempo t = 0, 1, 2, 3… , svolge del calcolo computazionale.
Lo stato del neurone numero j al passo t è identificato come:

$$s_j(t) = \begin{cases} 1 & \text{attivo} \\ -1 & \text{non attivo} \end{cases}$$

[3] McCulloch e Pitts sono un neurofisiologo e un logico americani

Ricevuti in ingresso N stati $s_j(t)$, il neurone numero i calcola

$$s_i(t+1) = sgn(\sum_{j=1}^{N} w_{ij}s_j(t) - \theta_i) = sgn(b_i(t)) \tag{2.1}$$

Infatti, dato un neurone avente N canali di ingresso s_1, ···, s_N ad ognuno dei quali è associato un peso w_ij, il cui primo indice i si riferisce al neurone che fa il calcolo, invece l'indice j rappresenta tutti i neuroni connessi al neurone i. Il valore del peso è rappresentato da un numero reale, che riproduce lo spessore e la conducibilità delle sinapsi neurali. Il segnale di uscita al passo $t + 1$ è ricavato attraverso la la somma pesata degli ingressi, detta livello di attivazione o *local field* $b_i(t)$. Alla somma pesata vediamo nella [1] essere sottratto il valore di threshold, denotato come θ_i. Nel campo del deep learning di cui si parlerà a breve esso è conosciuto anche come *bias*, ed è necessario per la robustezza della rete neurale. Può essere identificato come un ulteriore canale di ingresso s_0 che serve per alzare o abbassare la soglia di attivazione del neurone (threshold), dove per soglia indichiamo il livello oltre al quale il neurone risulta attivo. La somma pesata e shiftata viene poi passata per una determinata funzione f, che nel caso del neurone di McCulloch e Pitts è la funzione segno *sgn()*. Nel deep learning la funzione segno è molto spesso sostituita da una funzione f chiamata *funzione di attivazione*. La funzione di attivazione definisce l'uscita di un neurone in funzione del suo livello di attivazione $b_i(t)$. Una scelta comunemente utilizzata è la sigmoide:

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

Un'altra molto importante e anch'essa utilizzata è la ReLU (Rectified Linear Unit):

$$f(x) = x^+ = max(0, x)$$

Tale funzione è una delle più utilizzate negli algoritmi di deep learning per la rapidità di apprendimento per reti formate da vari strati di neuroni e altri fattori che saranno approfonditi in seguito. Un'altra funzione utilizzata è la tangente iperbolica.
La formula [1] è anche la formula che definisce la *regressione logistica*, l'algoritmo d'apprendimento base di cui si compongono le reti neurali. La computazione fatta nel singolo neurone viene poi eseguita in maniera parallela per tutti i neuroni e gli output di

s_i diventano gli input per i neuroni dello strato successivo a quello i-esimo. Lo scopo quindi di Mc-Culloghs è stato quello di costruire un modello che simulasse le dinamiche delle reti neurali reali.
Dato che una rete neurale efficiente deve essere in grado di apprendere e fornire uscite a fronte di input anche sconosciuti, occorre allenare la rete con degli algoritmi. L'addestramento della rete neurale si basa sulla presentazione al suo ingresso di una serie di esempi, la cui natura la rete è in grado di riconoscere, che costituiscono il dataset di training. Le risposte che la rete fornisce vengono confrontate con quelle attese e si calcola l'errore tra le due. Sulla base di tale errore i pesi vengono modificati istante per istante, e la procedura viene ripetuta fino a che la rete non fornisce un errore che sia al di sotto di una determinata soglia prestabilita.

Una rete neurale di dice stratificata quando è formata da vari strati di neuroni, tali che ognuno di loro sia connesso con quelli dello strato successivo, ma senza che ci siano connessioni tra neuroni del medesimo strato. Il neurone pensato da Mc-Culloghs è chiamato anche Percettrone e possiede solamente i solito si lavora con reti in cui vi sono uno o più hidden layers, i quali non comunicano direttamente con l'esterno. Queste reti sono chiamate MPL (Multi-Layer-Perceptron). uno strato di input e uno strato di output. Una rete neurale artificiale in cui per ogni strato il segnale di ingresso viaggia sempre in avanti dall'ingresso all'uscita, senza creazione di cicli, si chiama *feedforward*.

D Il primo vero miglioramento rispetto al Perceptron è stato l'Adaline (ADAptive LInear NEuron), in quanto utilizza proprio una funzione di attivazione lineare per regolare il vettore dei pesi al posto della funzione a gradino.

L'*apprendimento profondo*, in inglese Deep Learning (DL) è quel campo di ricerca del ML e dell'Intelligenza Artificiale che studia le tecniche e i sistemi per consentire alle macchine di apprendere in autonomia in modo profondo. Nello specifico, questa branca viene considerata una categoria sottostante al machine learning, in quanto si tratta di un approccio per l'apprendimento automatico dei sistemi informatici. Tuttavia, rispetto ad altri metodi il deep learning prevede un addestramento diverso delle macchine e più complesso. In altre parole, per apprendimento profondo si intende un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre

più completa con l'aumentare della profondità. Lo sviluppo del deep learning è motivato in parte dal fallimento degli algoritmi tradizionali nel tentare di svolgere compiti delle AI, come ad esempio speech recognition e object recognition. Il principio di funzionamento è lo stesso delle reti neurali classiche, con la differenza che risiede nel numero molto elevato di livelli nascosti di neuroni intermedi. Altre sfide che hanno portato allo sviluppo dell'apprendimento approfondito è la difficoltà degli algoritmi classici nel generalizzare su problemi con dati multi-dimensionali. Il fenomeno legato alla grande dimensionalità dei dati è conosciuto come *course of dimensionality*: questo fenomeno si presenta spesso in molti campi della computer science, in particolare nel machine learning. É stato introdotto questo problema per la prima volta dal matematico Richard Bellman e indica che il numero di campioni necessari per stimare una funzione arbitraria con un dato livello di accuratezza cresce esponenzialmente rispetto al numero di variabili di input (cioè dimensionalità) della funzione. All'aumentare del numero dei dati aumenta anche il numero delle *features*, ma all'aumentare delle features diminuisce anche la capacità predittiva del sistema con un conseguente peggioramento delle prestazioni. Come già detto nel precedente capitolo, con i metodi convenzionali di machine learning bisogna costruire gli estrattori delle feature dei dati di input. L'accuratezza e la precisione delle predizioni sono fortemente influenzate dall'abilità di colui che realizza il modulo di estrazione, che richiede inoltre un forte sforzo ingegneristico.
Un vantaggio fondamentale delle reti di deep learning è la possibilità di migliorare le prestazioni con l'aumentare del formato dei dati.

Un'organizzazione gerarchica dei dati permette di condividere e riutilizzare informazioni estratte durante l'elaborazione e di selezionare e scartare dettagli inutili lungo una gerarchia. Rispetto ad una architettura semplice a tre strati (input-strato con unità nascoste-output), una architettura multi-strato permette di distribuire meglio un grande numero di nodi su più strati riducendo il costo computazionale elevato se fossero tutti localizzati su un solo strato e attenuando l'ingente utilizzo di memoria che comporterebbe una struttura meno profonda.
In generale le tecniche di deep learning stanno diventando sempre più utili a nche in quello che è il campo della computer vision, cioè l'area di studio delle AI che si occupa della ricerca aull' automatizzazione dell'informazione visuale, e in particolare anche dell'analisi di dati biomedici, come nel caso del presente lavoro di tesi.

Le Deep Feedforward Networks, chiamate anche multilayer perceptrons (MLPs), sono degli strumenti molto importanti usati nel deep learning. L'obiettivo delle MLPs è approssimare una funzione f^*. Ad esempio, se si vuol costruire un classificatore, y = f^x esegue una mappatura dell'input x su una categoria y. Una rete di tipo feedforward definisce una mappatura y = f (x, θ) e cerca il parametro $theta$ che renda migliore l'approssimazione della funzione. Questi modelli sono chiamati feedforward perché l'informazione fluisce prima attraverso la funzione che deve valutare x, poi lungo le elaborazioni intermedie usate per definire f fino all'output y. Questo tipo di reti sono di solito rappresentate dalla composizione di più funzioni. Per esempio, possono essere formate da tre funzioni f^1, f^2 e f^3 connesse da una catena:

$$f(x) = f^3(f^2(f^1(x)))$$

n questo caso f^1 sarà chiamato primo strato (first layer), f^2 secondo strato e così via. La lunghezza della catena indica la *profondità* del modello. Lo strato finale della rete è chiamato strato di output. Durante l'addestramento della rete si cerca di approssimare $f(x)$ ad una funzione $f^*(x)$ valutata su differenti esempi di addestramento.
Gli esempi usati durante l'addestramento specificano direttamente cosa deve fare lo strato di output ad ogni punto x: deve produrre un valore il più vicino possibile alla label di target y. Il comportamento degli altri strati non è direttamente specificato dai dati di addestramento; l'algoritmo di apprendimento deve decidere come usarli per produrre l'output desiderato per implementare al meglio una approssimazione di f^*. Siccome i dati di addestramento non mostrano l'output desiderato per ognuno di questi strati vengono chiamati *strati nascosti*. La dimensione degli strati nascosti determina la larghezza del modello. Lo strato rappresenta una funzione a valori vettoriali formata da molte *unità* che agiscono in parallelo, ognuna rappresentante una funzione a valori vettoriali che restituisce un valore scalare. Ogni unità è assimilabile ad un neurone nel senso che essa riceve l'input da altre unità ed elabora un output che verrà mandato a sua volta come input ad altre unità, il valore di output viene chiamato valore di attivazione, che è analogo a quello delle reti neurali non profonde.

2.3 Addestramento di una rete

Come è stato già detto in precedenza, il punto di partenza per l'addestramento in generale, ma ancora di più per addestrare una rete neurale, è avere a disposizione un dataset formato da un insieme di training che verrà somministrato in input alla rete per estrarne automaticamente le features, e un insieme di test con dati completamente nuovi al sistema per verificare l'efficacia dell'addestramento stesso.
Per semplicità, assumiamo che il problema che vogliamo risolvere sia un pro- blema di classificazione binaria. Per esempio vogliamo decidere se, data una certa immagine, questa rappresenti o meno un gatto.
Rappresentiamo la nostra immagine come un vettore x unidimensionale di byte. Fissiamo y = 1 nel caso in cui l'immagine rappresenti un gatto, y = 0 nel caso in cui nell'immagine non ci sia un gatto. La nostra unità di regressione logistica dovrà dunque calcolare un'approssimazione di y; più precisamente calcolerà le probabilità che nell'immagine ci sia un gatto. Dato x, vogliamo dunque calcolare l'approssimazione $\hat{y}$ = P(y = 1|x). Perché ciò avvenga, serve che i parametri W e b siano opportunamente confi- gurati. Ed è qui che entra in gioco il concetto di apprendimento. Durante l'addestramento, dopo aver fornito l'input, la rete produce un output in forma di vettore di risultati. Bisogna calcolare una funzione che misuri l'errore (o la distanza) tra i risultati di output e i pattern desiderati dei risultati, cioè i valori con cui vengono etichettati i dati nell'apprendimento supervisionato. La funzione che ci permette di farlo è chiamata *funzione errore* o di perdita. Una delle funzioni di errore utilizzata nell'apprendimento supervisionato è la MSE (Mean Squared Error):

$$MSE = \frac{1}{2}\sum(out - target)^2 \tag{2.2}$$

Nel deep leaning è molto utilizzata *funzione di entropia incrociata*, funzione di perdita che è usata per quantificare la differenza tra due distribuzioni di probabilità: quella che vorremmo il nostro sistema raggiungesse, e quella che il nostro sistema ha elaborato statisticamente fino a quel momento. Viene descritta dalla seguente formula:

$$H(h, q) = -\sum_{x} p(x)\log(q(x))$$

dove p è la funzione di distribuzione da raggiungere su un insieme di eventi x, mentre q è quella fino ad ora elaborata. Ogni volta questa funzione ci indica quanto la nostra stima fatta si discosti dalla realtà.

La rete in fase di addestramento modifica i suoi parametri interni variabili in modo da minimizzare la funzione di costo. Questi parametri sono chiamati pesi e spesso sono numeri reali che possono essere visti come delle "manopole" che definiscono la funzione di input-output della rete. Per modificare in maniera appropriata il vettore dei pesi, l'algoritmo di apprendimento calcola un vettore gradiente che, per ogni peso, indica di quanto l'errore cresce o decresce se i pesi vengono incrementati di una quantità infinitesima. Il vettore gradiente, calcolato sulla funzione di costo indica la direzione più ripida e veloce su come andare a aggiornare i pesi stessi, portandosi sempre più vicino a dove l'errore di stima dell'output è minore in media.
L'approccio più utilizzato al fine di minimizzare l'errore è la procedura chiamata *gradiente stocastico decrescente*. Esso consiste nel mandare in input alla rete alcuni esempi, calcolandone l'output con la (2.1), l'errore associato (ad esempio con la (2.1)) e il gradiente medio di questa ultima variando i pesi in maniera infinitesima. La procedura è iterata per molti set piccoli di esempi presi dall'insieme di addestramento finchè la media della funzione oggetto non smette di decrescere. Si parla di gradiente stocastico perchè ogni piccolo insieme di esempi fornisce una stima del gradiente medio su tutti gli altri.
A titolo di esempio, si mostra successivamente come funziona l'algoritmo del gradiente discendente per un semplice percettrone.

1. La rete prende in ingresso dei valori randomici iniziali per i pesi W e un valore costante, il learning rate. Tale valore determina lo step da fare ad ogni iterazione nella direzione indicata dal gradiente.

2. Si calcola la funzione di costo con i parametri attuali;

3. Si aggiornano i valori dei pesi usando la funzione di aggiornamento dei pesi W che è:
$$W^{(\rho)} = W^{(\rho-1)} - \eta \frac{\partial E}{\partial W} \tag{2.3}$$
dove ρ indica il valore dei pesi al passo precedente e η il learining rate.

4. Alla fine l'algoritmo ritorna il costo minimizzato per la funzione di errore.

   Occorre dunque vedere come il valore dei pesi ha effetto sulla funzione di errore. Si applica la regola della catena di derivazione:

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial out}\frac{\partial out}{\partial in}\frac{\partial in}{\partial W}$$

   dove *in* è la funzione che è combinazione lineare di input e pesi, mentre *out* è l'output che si ottiene facendo passare *in* per la funzione di attivazione, per esempio la sigmoide. Nel caso in cui si abbia una rete senza hidden layers la regola della catena si applica in questo modo, se vi fossere più strati però la procedura è analoga andando ad applicare tale regola per ognuno degli strati. Se si vanno a valutare le nostre derivate per il caso del percettrone, si può facilmente dimostrare che:

$$\frac{\partial E}{\partial out} = (out - target),$$
$$\frac{\partial out}{\partial in} = out(1 - out),$$
$$\frac{\partial in}{\partial W} = in$$

   Si moltiplicano tra loro dunque tutti i termini derivativi e poi si applicano all'equazione principale (2.3) e si aggiornano i valori dei pesi di conseguenza, fintanto che la media dell'errore converge, ovvero smette di decrescere.

Tale algoritmo nelle prove sperimentali ha effettiva realizzazione in quello che è chiamato l'*ottimizzatore* (uno dei più è efficienti è proprio SGD (Stochastic Gradient Descent), ma vi sono anche Adam, ADAline... Tutti però seguono l'idea di base del gradiente discendente.
Infine terminata questa procedura si passa alle valutazioni sulla performance della rete; essa viene misurata con gli esempi dell'insieme di test, che serve a testare la capacità di generalizzazione della rete.

Il metodo migliore per minimizzare la funzione di errore è proprio quello di usare il gradiente. Nelle reti MLP (Multi-Layer-Perceptron) si effettuano una serie di derivate successive e ogni volta si trova il punto che annulla la derivata della funzione di errore. Questo procedimento è piuttosto semplice nelle reti neurali più basilari, mentre inizia ad essere più complicato nelle reti neurali con più strati. In questo caso si gioca sul valore del learning rate. Un valore del learning rate (di solito compreso tra 0 e 1) troppo alto farà fare un salto troppo grande verso il minimo della funzione di costo, facendo perdere alla rete capacità di convergenza (rischiando in alcuni casi anche la divergenza) mentre un learning rate troppo basso rallenta notevolmente il training e può comportare il fatto che l'algoritmo rimanga fermo nel calcolo di un minimo locale indesiderato per la funzione di errore.

Dunque, il ciclo di apprendimento di una rete neurale feedforward può essere sintetizzato in due fasi:

- fase di FEEDFORWARD: si fanno passare i pattern dei dati di addestramento (input features) attraverso la rete per ottenere un output;

- fase di BACKPROPAGATION: si calcola la differenza tra l'output e il valore di target (di base si calcola l'errore di predizione) e si utilizza così l'algoritmo Gradient Descent per aggiornare i valori dei pesi. In pratica si riporta il valore dell'errore all'ingresso, ne si calcola la sua derivata rispetto ad ogni peso della rete e ogni volta si aggiornano i valori dei pesi del modello.

Dopo varie iterazioni dei due passi si ottiene una maggiore accuratezza di predizione per la rete.
Nelle reti neurali profonde si utilizzano gli algoritmi di back-propagation per facilitare il procedimento di calcolo del gradiente. In questi algoritmi i pesi sono calcolati partendo dall'ultimo strato di neuroni procedendo all'indietro. Un ciclo di presentazione alla rete di tutti gli eventi appartenenti all'insieme di addestramento è detto *epoca*.

La procedura di propagazione all'indietro calcola il gradiente di una funzione oggetto in funzione dei pesi associati ai propri parametri proprio applicando semplicemente la

regola di derivazione della catena: la derivata, o proprio il gradiente, di una funzione rispetto ai valori di input può essere calcolata propagando all'indietro, dall'output del modello fino ad arrivare al primo strato. Partendo dall'output, cioè lo strato dove la rete produce le sue predizioni, il gradiente fluisce attraverso tutto il modello fino ad arrivare all'input, dove sono forniti i dati su cui sono fatte le predizioni.
Una volta calcolati questi gradienti è facile trovarne l'espressione dipendente dai pesi associati ai parametri della funzione oggetto.

2.4 Overfitting e underfitting

L'abilità di un algoritmo di performare bene su input mai osservati precedentemente si chiama generalizzazione. Un buon algoritmo di machine learning è un algoritmo che possiede errore di generalizzazione minimo.
I termini *overfitting* e *underfitting* sono nati proprio come risposta alle deficienze di cui il modello può soffrire. Pertanto sapere quanto il modello commette errori, significa sapere quanto questo va in overfitting o underfitting. Come è possibile influenzare l'errore sull'insieme di test quando questo insieme non si conosce a priori?
Il campo della teoria dell'apprendimento statistico fornisce alcune risposte. Attraverso delle assunzioni riguardanti la maniera in cui sono stati raccolti i dati si possono studiare matematicamente le relazioni che intercorrono tra l'errore dell'insieme di addestramento e l'errore sull'insieme del test.

Quindi, dopo aver generato l'insieme di addestramento, si scelgono e si variano i parametri per ridurre l'errore sull'insieme, che dovrebbe corrispondere anche all'errore sull'insieme di test. I fattori che determinano quanto sia efficiente un algoritmo di apprendimento sono: 1. la sua abilità nel minimizzare l'errore di addestramento; 2. la sua abilità nel minimizzare il gap tra errore di addestramento e l'errore di test.

L'**underfitting** è quando il modello non riesce a rendere l'errore di addestramento sufficientemente piccolo. Questo significa che il modello è troppo semplice e non si riescono a identificare un numero sufficiente di features, il che rende impossibile al modello di apprendere da quel dataset. In termini di machine learning significa che c'è stato un focus troppo basso sul set di training, dunque il modello non è nemmeno in grado

di testare correttamente. L'**overfitting** è il caso in cui il modello ha imparato molto bene dal suo dataset di training, ma non è in grado di generalizzare bene. Questo significa che c'è un forte gap tra errore di addestramento ed errore di test. In termini di machine learning vuol dire che che c'è stato un eccessivo focus nel training set, e che le relazioni stabilite tra i neuroni non sono valide per valutare correttamente nuovi dati. Generalmente gli algoritmi di machine learning performano meglio quando la capacità dei dati è appropriata per la reale complessità del compito che devono svolgere e la quantità di dati dell'insieme di addestramento forniti. Modelli con capacità insufficiente non sono capaci di svolgere compiti complessi. Modelli con grande capacità possono risolvere compiti difficili, ma nel caso in cui la capacità sia più grande di quella richiesta del compito da svolgere potrebbero andare in overfitting. In effetti nel machine learning i modelli sono soggetti al problema del *trade-off varianza-bias*. Questo nasce come conflitto nel provare a minimizzare simultaneamente le due fonti di errore che impediscono al modello di generalizzare:

- errore di bias, che è consequenziale ad errate assunzioni fatte dall'algoritmo di learning, per rendere il dataset più facilmente allenabile (underfitting).
- errore di varianza, dato dalla forte sensitività del modello a fronte di piccoli cambiamenti del training set, con conseguente aumento di rumore (overfitting).

E' impossibile minimizzare entrambe, ma è possibile raggiungere un buon compromesso tra le due.
Un modo per evitare underfitting (bias alto) è quello di utilizzare una maggior quantità di dati ma questo non sempre funziona. Altrimenti si può incrementare la complessità del modello, riducendo il rumore dei dati oppure aumentando il numero di epochs e quindi incrementando la durata del training.
Tutto ciò però nella misura tale per cui non si ricada nell'overfitting (alta varianza), che può essere evitato ad esempio riducendo la complessità del modello, in quanto dati troppo 'rumorosi' possono portare a valutazioni erronee e poco generalizzabili. Un altro modo è quello di ridurre il numero di parametri o come si vedrà in seguito nelle CNN, usando dei Dropout. Un'altra tecnica è quella di utilizzare alcune funzioni, come EarlyStopping in Keras che valutano passo passo l'andamento del training e lo fermano non appena si nota che il modello non è più in grado di generalizzare, cioè prima che cada in overfitting.

2.5 Classificazione

Come detto in precedenza, in questo lavoro di tesi le reti saranno addestrate al fine della classificazione delle immagini di risonanza magnetica cerebrale in una prima esperienza e altre immagini di raggi-X pettorali in una seconda esperienza. Riuscire ad estrarre informazioni utili dalle immagini non è affatto facile e spesso la possibilità di codificare al meglio l'informazione è inibita da fattori quali un errore sul modello o una scorretta acquisizione dell'immagine. Ciò che permette di estrarre le informazioni dalle immagini sono le feature delle stesse, identificando oggetti nell'immagine che risultano avere caratteristiche comuni. Schemi ricorrenti nei soggetti malati (o anche nei soggetti sani) permettono di sviluppare criteri oggettivi per determinare la diagnosi dei pazienti ricercando tali schemi. Come nel caso del dataset del Brain Tumor utilizzato nella tesi, le diverse categorie di tumore è sicuramente necessario riconoscere l'adenoma a seconda della sua forma e della variazione di colorazione. Dal riconoscimento di determinate caratteristiche è possibile classificare un oggetto mai visto prima dal sistema attribuendogli un classe di appartenenza o label definita a priori. In questo lavoro di tesi il processo di classificazione si è basato su un approccio supervisionato definito in un paragrafo precedente. Occorre andare ad attribuire all'oggetto in esame, nel nostro caso a immagini di risonanze magnetiche, un certo numero di caratteristiche che ci permettano di identificarlo univocamente.
Nei problemi di classificazione supervisionata come quello di interesse occorre costruire un classificatore in grado di apprendere dai dati in ingresso e riuscire così a classificare anche un dato mai visto e del quale si vuol conoscere la classe di appartenenza. Lo scopo principale delle nostre reti usate per la classificazione mediante un approccio di apprendimento supervisionato è quello di definire delle buone regole di decisione per l'assegnazione delle etichette ad oggetti sconosciuti, sulla base delle conoscenze che il modello ha acquisito. Per una generica osservazione y, ed un numero di classi k, il classificatore può essere definito dalla funzione:

$$C(y) : R^n \rightarrow 1 \ldots k$$

Se y appartiene alla classe j e C(y) $\neq$ j, si dice che il classificatore C commette un errore nella classificazione di y. Chiamati I l'insieme dei dati di input e O l'insieme dei dati di

output, si definisce una funzione h tale che associ ad ogni dato di ingresso I la sua risposta corretta. Questa funzione h non si conosce ed è quella che va appresa. L'accuratezza di un classificatore C indica la porzione di elementi correttamente classificati su un gruppo di dati N_t con classe nota rapportata ai totali classificati. Dunque la porzione di dati non correttamente classificati sarà: $\delta E = \frac{N_{err}}{N_t}$.

Abbiamo quindi descritto le reti neurali come una semplice catena di strati caratterizzata dalla profondità del modello e dalla larghezza di ogni strato. Variando questi parametri si possono ottenere un numero considerevole di architetture diverse. Molte architetture sono state sviluppate per compiti specifici e nel presente lavoro di tesi approfondite quelle che sono state utilizzate nella fase sperimentale per la classificazione: le CNN (Convolutional Neural Networks). Nel capitolo successivo viene una piccola digressione su modello matematico di tali reti.

3. Reti neurali convoluzionali

3.1 Funzionamento generale

Le Convolutional Neural Networks, o reti di convoluzione, sono reti specializzate nell'elaborazione di dati che hanno la forma di vettori multipli con una topologia nota a forma di griglia. Esse nascono nel 1990 dalla ricerca di Yann LeCun insieme al suo team basandosi sul funzionamento della corteccia visiva del cervello umano. Grazie alle ottime prestazioni che si sono riuscite a ricavare soprattutto in ambito del riconoscimento di immagini, ancora oggi le CNN sono considerate lo "stato dell'arte" per quanto riguarda riconoscimento di pattern ed immagini. Le CNN possono essere addestrate nuovamente per nuove attività di riconoscimento, consentendo agli utenti di basarsi sulle reti preesistenti.
Un'immagine può essere vista come una griglia di due dimensioni di pixel contenente i tre valori di intensità per i tre canali del colore (RGB). Il nome reti di convoluzione è dato appunto dal fatto che durante il suo funzionamento esegue un'operazione matematica chiamata convoluzione.
Un aspetto chiave di ogni algoritmo di Machine Learning è quello di riuscire ad estrarre i tratti più importanti dai dati in ingresso. Le CNN sono in grado di capire in maniera automatica i tratti più rilevanti e di apprenderne i pattern; per questo motivo di norma gli strati delle CNN vengono considerati come dei blocchi per rilevare i tratti: i primi strati (quelli subito dopo lo strato di ingresso) sono considerati "low-level features extractors", mentre gli ultimi strati (di solito completamente connessi come quelli delle Reti Neurali non convolutive discusse all'inizio) sono considerati "high-level features extractors".

A ciascuna immagine di addestramento vengono applicati dei filtri a diverse risoluzioni e l'output di ciascuna immagine convoluta viene utilizzato come input per il layer successivo. I filtri possono essere inizialmente feature molto semplici, ad esempio la luminosità o i bordi (low level features), e diventare sempre più complessi fino a includere feature che definiscono in modo univoco l'oggetto (high level features). Le CNN elaborano delle *mappe dei tratti* (o *feture maps* dove ogni elemento corrisponde a dei pixel nell'immagine originale. Per ottenere questo risultato è necessario effettuare appunto l'operazione di convoluzione.

3.2 Architettura generale di una CNN

3.2.1 Convoluzione

Una CNN usa la convoluzione al posto del generale prodotto matriciale in almeno uno dei suoi strati. La convoluzione è una operazione su due funzioni a valori reali; dette queste due funzioni *x* e *w*, il loro prodotto di convoluzione sarà

$$s(t) = \int x(a)w(t-a)da = (x * w)(t).$$

Nelle reti di convoluzione spesso la funzione *x* si riferisce alla funzione di ingresso e la funzione *w* al kernel, che può essere visto come una funzione di peso relativa ai dati di input. Più in generale, nelle applicazioni l'input è un vettore multidimensionale dei dati e il kernel è un altro array multidimensionale di parametri che vengono adattati dall'algoritmo di apprendimento in maniera appropriata. Per utilità pratiche conviene definire nello specifico l'operazione di convoluzione discreta, un prodotto di convoluzione che invece di essere implementato su un integrale esteso all'infinito è implementato su una sommatoria su un numero finito di indici riferiti agli elementi dei vettori. Per esempio, avendo come input una immagine bidimensionale I, si potrà usare un kernel bidimensionale W tale che:

$$S(i,j) = (I * W)(i,j) = \sum_m \sum_n I(m,n)W(i-m, j-n).$$

Spesso nelle implementazioni software si preferisce utilizzare una funzione che si basa su quella sopra, chiamata cross-correlation, che è la stessa della convoluzione ma con il kernel capovolto.

$$S(i,j) = (I * W)(i,j) = \sum_m \sum_n I(i+m, j+n) W(m,n).$$

Vi sono due concetti alla base del funzionamento delle CNN :

1. *Sparse Connectivity* (connessioni locali).
   L'idea è in un'immagine i pixel che sono spazialmente più vicini sono più propensi a rappresentare qualcosa di correlato tra loro. Per esempio, nell'analisi di una immagine di input che abbia centinaia o milioni di pixel siamo interessati a individuare feature significative o dettagli importanti molto piccoli rispetto alla dimensione totale dell'immagine, contenuti per esempio in una decina di pixel come ordine di grandezza. Gli strati tradizionali delle reti neurali usano il prodotto matriciale tra matrici di parametri che descrivono l'interazione tra ogni unità di input con ogni singolo output; questo significa che ogni unità di output interagisce con ogni unità di input. Le reti di convoluzione invece possono avere connessioni locali tra i neuroni che generano interazioni sparse nella rete. Questa connessione locale può essere formalizzata impostando il kernel più piccolo rispetto all'input. Dati m input e n output, un prodotto matriciale richiederebbe m·n parametri e l'algoritmo una complessità temporale O(mn) di elaborazione. Limitando il numero di connessioni per ogni output ad un numero k più piccolo di m sarebbero richiesti solo k ·n parametri e un tempo di elaborazione pari a O(k ·n). Il guadagno in efficienza diventa estremamente importante con k più piccoli di m di molti ordini di grandezza.
   Dunque l'idea della convoluzione è di preservare le relazioni spaziali tra i pixel. Ridurre il numero di connessioni regolarizza la rete e riduce il rischio di overfitting.

2. *Parameter-sharing* (parametri condivisi): l'idea di base è che gli stessi pesi vengano utilizzati per diversi gruppi di pixel dell'immagine principale.
   In una rete tradizionale, ogni elemento della matrice dei pesi è usato esattamente una volta durante il calcolo di un output dello strato e poi non viene più riutilizzato.

> Qui gli stessi pesi vengono utilizzati per diversi gruppi di pixel dell'immagine principale. Il valore di un peso applicato ad un input è collegato al valore di un peso applicato in un altro punto della rete. In una CNN ogni kernel utilizzato è fatto in modo tale da essere invariante alle traslazioni, quindi è usato in tutte le posizioni dell'input (eccetto che per alcune particolari condizioni al contorno). La rete apprende quindi solo da un determinato set di parametri invece che da tanti set separati per ogni sezione dell'input, il che riduce significativamente il numero di accessi in memoria. Inoltre si sa che così come i neuroni della corteccia cerebrale sono designati a riconoscere feature locali dell'immagine (come angoli o bordi), si utilizzano gli stessi kernel (o filtri) per differenti parti dell'immagine, sempre con gli stessi pesi e valori di attivazione.

I neuroni degli strati di convoluzione sono organizzati nelle cosiddette *feature maps*, nelle quali ogni unità è connessa ad una porzione locale della mappa allo strato seguente attraverso un insieme di pesi. Ogni feature map corrisponde ad un determinato *kernel* che ha il fine di ricercare una certa feature in determinati punti dell'immagine. Data la definizione di convoluzione discreta descritta sopra, dato un filtro di dimensione k x k, con dei pesi associati atti a ricercare una determinata feature, e chiamato θ il valore del bias si fa traslare il filtro lungo l'immagine di input e si fa una somma pesata e shiftata rispetto a θ salvando il risultato ottenuto nelle fearure maps stesse, che saranno tante quanti sono il numero di filtri applicato allo strato di input per rilevare le features.
IMMAGIENE Il kernel viene shiftato ogni volta lungo la mappa dello strato precendente in orizzontale e verticale di un valore s, detto *stride*, che è un iperparametro molto importante nella fase di convoluzione.

La formula sopra è utilizzata per un array di due dimensioni. Per immagini colorate di solito si hanno 3 canali per colore, dunque in tale caso l'array in input sarebbe tridimensionale. Si ottengono quindi array di dimensioni maggiori, chiamati *tensori*. Tutti i neuroni di uno stesso strato convoluzionale hanno lo stesso valore del bias. Il package software Tensorflow utilizzato nella fase sperimentale è atto a eseguire efficientemente tutte le operazioni con i tensori.

3.2.2 Padding

Se si accoppiano più strati di convoluzione l'uno dopo l'altro, a mano a mano che ci si muove verso destra il numero di neuroni decresce molto velocemente. E' per questo che è necessario il padding, che permette di aggiungere righe o colonne di peso pari a 0, e che permettono l'utilizzo di filtri non troppo piccoli senza dare problemi con le dimensioni. Esistono 3 tipi di padding ma quello utilizzato di più nell'implementazione di una CNN è il *same padding*, il quale fa in modo di preservare le dimensioni dell'input originale.
Il sistema si calcola automaticamente il valore p di padding da adottare affinchè ciò possa essere rispettato. La dimensione della feature map risultato di una convoluzione è data da:

$$o = \frac{(n + 2p - m)}{s} + 1$$

dove:

- *n* è la dimensione del vettore di ingresso (se è un'immagine si fa sempre in modo di renderla quadrata *n* x *n*);
- *p*, come visto in precedenza è il valore di padding;
- *m* è la dimensione del filtro;
- *s* è lo stride.

Da qui è semplice ricavare *p* per il same padding e/o *s* per avere *o* del valore desiderato.

3.2.3 Funzioni di attivazione

Come succede nelle reti neurali artificiali dense, anche nelle convoluzionali è necessario che, una volta fatta la convoluzione e generate le feature maps, i neuroni vengano passati attraverso una funzione di attivazione. Quella più utilizzata negli strati nascosti dei modelli nelle prove sperimentali è la ReLu (Rectified Linear Unit). Con questo tipo di funzione le feature maps sono passate attraverso una funzione che ritorna zero se il valore in input è inferiore a 0, mentre ritorna il valore dell'input stesso se questo è maggiore di zero.
Ciò è utile in quanto lo scopo di una CNN è quello di incrementare sempre di più la non

linearità del dataset. Quando si guarda ad un immagine in effetti si nota come essa contenga molte zone non lineari come bordi, angoli, transizioni di colore…
Lo scopo è quello di eliminare i pixel neri così da mantenere solo i grigi e i bianchi rendendo la color transition più bruta ma più efficace al fine di identificare feature.
La ReLu permette dunque di non attivare tutti i neuroni allo stesso momento, riducendo il costo del processo di backpropagation necessario ad una corretta determinazione dei pesi durante il processo di training della rete, che è sicuramente più veloce se si procede in questo modo. La convergenza della rete viene velocizzata di circa 6 volte di più rispetto a utilizzare una funzione di attivazione come la sigmoide o la tangente iperbolica.
La ReLu verrà utilizzata anche perchè risolve in parte il problema della *scomparsa del gradiente*, uno dei principali problemi del deep learning. Durante la fase di backpropagation i pesi degli strati in prossimità dell'input restano costanti o si aggiornano molto lentamente al contrario di quanto accade per gli strati vicini all'output. Questo può provocare un rallentamento della rete ed è dovuto alle funzioni di attivazione. Funzioni di attivazione come la sigmoide infatti sono funzioni a codominio limitato e hanno una derivata che presenta una regione di significatività (ovvero la regione del dominio dove la funzione derivata assume i valori più significativi) piuttosto piccola, oltre la quale il valore della derivata stessa è molto piccolo. Per la regola della catena nella fase di backpropagation è necessario andare a fare un prodotto di derivate che è pari al numero di strati della rete neurale, come detto nella sezione 2.3. A mano a mano che però si torna indietro dall'output verso l'input però, moltiplicando valori molto vicini allo zero tra di loro si ottengono valori di aggiornamento dei pesi infinitesimi che comportano l'impossibilità per la rete di apprendere bene, soprattutto quando si vanno ad utilizzare reti molto profonde, e quindi quelle tipiche del DL.

3.2.4 Strati di subsampling: Pooling Layers

Mentre il ruolo dello strato di convoluzione è quello di trovare congiunzioni locali tra feature dello strato precedente, il ruolo dello strato di pooling è quello di fondere semanticamente feature simili in uno solo riducendo la dimensione dei dati. Questo strato di solito è applicato subito dopo quello di convoluzione perchè lavora sull'output processato da questo strato e riprende il concetto di sparse connectivity.
Un neurone di uno strato di pooling prende gli input di diverse feature maps vicine tra

loro e li riduce ad un singolo output. Esistono due tipi di subsampling layers (strati di sottocampionamento) di questo tipo:

1. *Max Pooling*: le unità di questo strato calcolano il massimo di una porzione locale (definita a seconda della dimensione stabilita dal codice, ad esempio porzioni 2x2) in una mappa di feature. Allora una funzione di pooling sostituisce l'output di una rete ad una certa locazione con un risultato che riassume quelli ottenuti dagli output vicini, riducendo la dimensione della rappresentazione e focalizzando l'analisi solo sui punti salienti dell'oggetto analizzato.

2. *Mean Pooling*: in questo caso ogni neurone prende il valore medio dei valori della porzione di input.

In sintesi, un livello di pooling esegue un'aggregazione delle informazioni nel volume di input, generando feature maps di dimensione inferiore, conferendo invarianza rispetto a semplici trasformazioni dell'input, mantenendo al tempo stesso le informazioni significative ai fini della discriminazione dei pattern contenuti in essi.
Se si ha una feature map di dimensione W x W x D, un kernel di pooling di dimensione F e un valore di stride pari a S, allora la grandezza dell'output sarà determinato dalla formula

$$\frac{(W - F)}{S} + 1$$

Il pooling funziona sull'idea di base per cui piccoli cambiamenti non cambieranno il risultato finale, perciò aggiunge robustezza ai dati.

3.2.5 Fully connected layers

Dopo gli strati di convoluzione/pooling si inseriscono uno o più strati di neuroni completamente connessi (Dense Layers), dopo aver compito un'operazione di Flattening della feature map bidimensionale dello strato precedente. I FC layers connettono tutti i neuroni del livello precedente al fine di stabilire le varie classi identificative visualizzate nei precedenti livelli secondo una determinata probabilità. É grazie a questi strati che è possibile effettivamente poi ottenere un vettore in uscita che produca una stima

probabilistica e che vada ad avere dimensione c x 1, dove c è il numero delle unità di classificazione.

Ricapitolando, l'architettura di una CNN consiste generalmente di questi strati:

-Input

-Convoluzione

-Attivazione non lineare

-Pooling

-Flattening

-Dense layers

-Output

4. Ambiente di lavoro

4.1 Python

4.2 Tensorflow e Keras

5. Implementazione delle reti e prove sperimentali

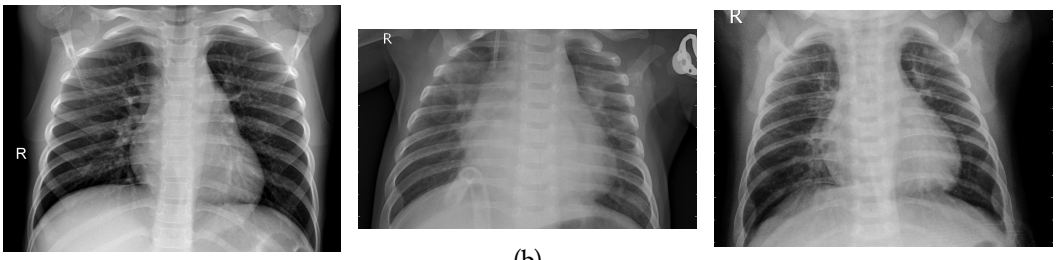
5.1 Obiettivo

L'obiettivo del lavoro svolto è stato quello di utilizzare dei Dataset di immagini biomedicali e sfruttarli per allenare una rete neurale convoluzionale il cui scopo è la classificazione.
Nel dettaglio, in una prima parte dell'esperienza è stato utilizzato un dataset di radiografie pettorali per addestrare un modello che riconoscesse quali tra i soggetti presentassero la polmonite e quali no.
Nella seconda parte si utilizza un dataset di risonanze magnetiche cerebrali per stimare una diagnosi dello stato del paziente tra 4 possibili situazioni: paziente sano, paziente con glioma, meningioma o tumore ipofisario.

5.2 CNN per la rilevazione della polmonite

5.2.1 Il dataset

Il dataset utilizzato, pubblicato sulla piattaforma Kaggle [1] da Paulo Breviglieri, è costituito da 5856 radiografie. É una versione rivisitata di un altro dataset le cui immagini sono state selezionate da uno studio di coorte retrospettivo [2] di pazienti pediatrici di età che va da 1 a 5 anni di un centro medico di Canton (Hong Kong). Le radiografie sono state realizzate come quadro clinico di routine per i pazienti. Le immagini stesse sono state sottoposte a screening per il controllo della qualità e tutte quelle illeggibili sono state rimosse. Le diagnosi sono poi state confermate da due fisici esperti e successivamente da un terzo specialista prima di essere utilizzate per allenare sistemi di AI.
Il modello di CNN utilizzato dovrà essere in grado di riconoscere i pattern di immagine tipici della malattia. L'identificazione è infatti talvolta complicata anche per i medici più esperti.
Sotto: esempio illustrativo di raggi-X in pazienti con polmonite.



(a) (b) (c)

Figura 5.1: Si può notare come un soggetto sano (a) mostri polmoni senza aree di anormale opacità. L'immagine (b) è invece un caso di polmonite batterica che presenta il tipico consolidamento. (c) è un caso di polmonite virale che presenta un pattern interstiziale più diffuso in entrambi i polmoni.

[1] Kaggle è una piattaforma online per competizioni di modelli predittivi e analitici fondata nel 2010. Ad oggi conta più di 500.000 utenti

[2] Uno studio di coorte prende in considerazione un gruppo di individui che presentano caratteristiche comuni (sesso, età, etnia…) e che hanno come unica differenza tra loro l'esposizione o meno al fattore di rischio. Questo tipo di studio identifica persone con una malattia (o altre variabili di interesse) e le paragona ad un gruppo di controllo appropriato che non presenti la patologia.
Questo gruppo viene osservato per un periodo di tempo prestabilito, al termine del quale si analizzerà la presenza o meno dell'esito atteso.

Il dataset è organizzato in 3 cartelle: una di Training (per l'addestramento stesso della rete), una di Testing (per testare le capacità di generalizzazione della rete) e una di Validation (per testare la bontà della rete e rilevare overfitting e/o underfitting in fase di addestramento). Ognuna di esse è organizzata così:

- 4,192 immagini per il training (1,082 casi normali, 3,110 di polmonite)
- 1,040 immagini di validation (267 casi normali, 773 di polmonite)
- 624 immagini per il testing (234 casi normali, 390 di polmonite)

Il numero di immagini è sufficientemente grande per poter essere utilizzato per gli obiettivi preposti ricorrendo alle tecniche di deep learning per la computer vision di ambito biomedico.
Le proporzioni del dataset rispettano ampiamente gli standard che di solito si utilizzano per addestrare un modello di classificazione: un set di testing e uno di validation che è sono circa il (10-20)% del totale, il resto è lasciato al training. Si va adesso a sintetizzare quali sono stati i principali passaggi per l'implementazione della CNN.

5.2.2 Setup iniziale

Il primo step per realizzare il classificatore è l'importazione delle librerie necessarie per la manipolazione di tensori, per le operazioni matematiche e per la relizzazione di grafici (Pandas, NumPy, MatPlotLib), così come librerie di Keras per l'image preprocessing e oggetti di Deep Learning, tra cui:

- `Sequential`, classe di Keras che permette di raggruppare layers, che non sono altro che i blocchi di base per la costruzione delle reti neurali, in uno stack lineare e realizzare così un oggetto di classe `Model`, che è implementato in modo tale da riuscire a elaborare deduzioni una volta superata la fase di training;
- `Conv2D`, classe che realizza la convoluzione spaziale sulle immagini 2D. Questo strato sfrutta un kernel di convoluzione il quale è coinvolto con lo strato in input per produrre un tensore in output. Se questo strato è usato per primo nel modello è necessario inserire come parametro la tupla rappresentante le dimensioni delle immagini in input. In questo strato è possibile scegliere il numero e le dimensioni

del filtro, il valore dello stride (s=1 di default), il tipo di padding e altri parametri del caso;

- `MaxPool2D`, realizza l'operazione di pooling, dunque è possibile anche qui scegliere la dimensione della pool, il valore dello stride, il tipo di padding...;
- `Flatten`, classe che permette di trasformare la feature map allo strato precedente in un vettore che viene dato in input alla ANN in seguito;
- `Dense`, classe che implementa un regolare strato di una NN. In sostanza è questo lo strato che implementa la somma pesata degli input con i pesi e il bias di cui si parlava nel capitolo 1.

5.2.3 Definizione degli iperparametri

Successivamente sono stati fissati dei valori per gli iperparametri, ovvero quelle variabili poste all'inizio del codice, prima che il processo di apprendimento cominci. I valori degli iperparametri sono davvero importanti per l'accuratezza del modello e delle predizioni, infatti sono stati fatti variare alcune volte prima di arrivare alla migliore soluzione. I principali sono:

- Le dimensioni di input dell'immagine, perchè è necessario che durante la fase di training vi sia un'uniformità delle dimensioni degli input;
- il numero di epoche, cioè il numero di volte in cui l'algoritmo di apprendimento lavora sull'intero dataset prima di procedere con l'aggiornamento dei pesi. Sulla base di queste varia anche la durata dell'apprendimento. Infatti gli step di apprendimento per epoca sono pari alla divisione tra il numero di istanze del set di trainig e la batch size;
- la batch size: è un iperparametro dell'algoritmo del gradiente discendente che indica il numero di campioni di training sui quali lavorare prima che venga fatto un aggiornamento dei pesi. Sia in questa esperienza sia in quella successiva si utilizza un algoritmo di apprendimento in modalità *mini-batch* in quanto il dataset è abbastanza numeroso, ovvero si utilizza un valore di batch più grande rispetto a 1, così che non si vada ad aggiornare i valori dei pesi ogni volta che il sistema riceve un

campione in input, quindi più del necessario, ottimizzando la durata dell'epoca e quindi del training. Di norma la batch size deve essere un valore che sia divisore del numero totale di immagini del dataset e nelle folders.

- il numero di feature maps: più che ne sono più caratteristiche differenti si possono andare a scovare ma allo stesso tempo aumenta anche il numero di parametri e se si vuole evitare il rischio di overfitting occorre aumentare la complessità del modello.

- il numero di canali d'immagine utilizzati nel processo di learning: per le immagini RGB colorate tale parametro è pari a 3, mentre nel caso di immagini in scala di grigi vale 1. In tale caso le immagini sono digitali RGB ma vedremo che utilizzare solo un canale è la soluzione che porta ad una maggiore accuratezza.

Facendo riferimento agli iperparametri utilizzati per la costruzione del classificatore sono stati scelti tali iperparametri come i migliori per questa esperienza:

```
img_height = img_width = 600
epochs = 10
batch_size = 8
hyper_featuremaps = 32
hyper_channels = 1
hyper_mode = 'grayscale'
```

E' stata utilizzata una dimensione per le immagini di 600 x 600 pixel associata a una batch size di 8 affinchè lo spazio occupato in RAM non sia eccessivamente alto.

5.2.4 Definizione e compilazione del modello

Per questa esperienza è stato utilizzato un modello di rete neurale convoluzionale semplice, che consiste di questi strati:

- 5 strati di Convoluzione/pooling: i primi 3 producono ognuno 32 feature maps con la convoluzione, generanti ognuna una feature map di pooling, per gli ultimi 2 sono stati invece utilizzati 64 filtri, così da produrre 64 feature maps anch'esse seguite da

un'operazione di pooling. La convoluzione è stata performata con dei kernel di dimensione 3x3 e con una funzione di attivazione non lineare, cioè la ReLu. Nel capitolo precedente è stato indicato in parte il perchè questo tipo di funzione è quella maggiormente utilizzata in ambito di classificazione d'immagine. L'operazione di Max-pooling viene fatta con delle pool di dimensione 2x2. Lo stride è stato lasciato quello di default a 1 e il padding scelto è il 'same', affinchè la dimensione dell'input sia uguale alla dimensione dell'output.

- Strato di flattening: necessario per introdurre la successiva ANN, connessa all'ultimo strato di pooling tramite un unico tensore unidimensionale.
- 3 di strati completamente connessi rispettivamente di 128, 64 e 1 neuroni. L'ultimo strato consta di un solo neurone in quanto la classificazione è binaria.

Tipicamente si parte sempre dall'utilizzare un numero di filtri più piccolo, come in questo caso 32, per poi andare ad aumentare a multipli di questa dimensione. Il modello è infine compilato usando il metodo `compile()` usando la funzione di ottimizzazione Adam, algoritmo che si basa sull'idea del gradiente discendente ma che però elabora una stima adattativa dei momenti del primo e del secondo ordine, permettendo una maggiore efficienza rispetto agli altri algoritmi a livello di costo computazionale per il training, a discapito di una conseguente minor capacità di generalizzazione. Inoltre Adam adatta il learning rate a seconda dei vari strati anche sulla base di quello che è il problema della scomparsa del gradiente di cui si è parlato sopra.
In fase di compilazione si sceglie anche la metrica, che permette di calcolare quanto spesso le label effettive coincidono con le predizioni fatte ed è dunque un parametro necessario per monitorare l'accuratezza e l'errore del modello. Una metrica di questo tipo viene settata col nome di `'accuracy'`. Per il calcolo dell'errore si usa una funzione di costo che in questo caso è chiamata *binary crossentropy*, (funzione di entropia incrociata binaria) poichè si va a fare una classificazione binaria.

```
cnn.add(Conv2D(hyper_featuremaps, (3, 3), activation="relu",
                                  input_shape=(img_width,
                                  img_height, hyper_channels)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))
```

```
cnn.add(Conv2D(hyper_featuremaps, (3, 3), activation="relu",
                                  input_shape=(img_width,
                                  img_height, hyper_channels)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))
cnn.add(Conv2D(hyper_featuremaps, (3, 3), activation="relu",
                                  input_shape=(img_width,
                                  img_height, hyper_channels)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))
cnn.add(Conv2D(hyper_featuremaps * 2, (3, 3), activation="relu",
                                  input_shape=(img_width,
                                  img_height, hyper_channels)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))
cnn.add(Conv2D(hyper_featuremaps * 2, (3, 3), activation="relu",
                                  input_shape=(img_width,
                                  img_height, hyper_channels)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))
cnn.add(Flatten())
cnn.add(Dense(activation = 'relu', units = 128))
cnn.add(Dense(activation = 'relu', units = 64))
cnn.add(Dense(activation = 'sigmoid', units = 1))
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy',
                                  metrics = ['accuracy'])
cnn.summary()
```

Code Listing 5.1: Modello in Python utilizzato

```
    Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 498, 498, 32)      320
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 249, 249, 32)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 247, 247, 32)      9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 123, 123, 32)      0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 121, 121, 32)      9248
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 60, 60, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 58, 58, 64)        18496
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 29, 29, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 27, 27, 64)        36928
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 10816)             0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               1384576
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,467,137
Trainable params: 1,467,137
Non-trainable params: 0
```

Code Listing 5.2: Riepilogo del modello. La colonna `layer` indica il tipo di strato di cui si tratta. La colonna output mostra la tupla con le dimensioni dello strato in uscita, con una dimensione in più `None` che è aggiunta per ospitare la batch size. La terza colonna `Param #` indica il numero di pesi all'interno della rete, i quali possono essere distinti in addestrabili, cioè quelli che vengono aggiornati durante la fase di backpropagation, e quelli per cui questo non vale per motivi di regolarizzazione della rete. Il numero totale di parametri si trova `(kernel_height*kernel_width*input_filters*output_filters) + output_filters`. Ad esempio nel primo strato si avra 3*3*32*1+32=320.

5.2.5 Creazione dei set di training e validation traminte l'uso dell'image flowing

Tramite l'utilizzo della classe di Keras `ImageDataGenerator()` è possibile andare a caricare le immagini di training, testing e validation andando a richiamare su tali generatori il metodo `flowFromDirectory()` che è in grado di ritornare un oggetto di tipo DataFrameGenerator costituito da una tupla (X , y) dove X è un NumPy array contenente un numero di immagini pari alla `batch size` e della dimensione pari a `image_widthximage_height`indicata e y è anch'esso un NumPy array che però contiene le label corrispondenti alle immagini prodotte. L'utilizzo di questa funzione è possibie perchè il dataset è strutturato in partenza in un dataset di trainining, validation e testing. Inoltre tale funzione genera i set provvedendo a fare uno shuffle di tutte le immagini sia di training sia di validation, mentre per il set di testing è stato inserito `Shuffle = False` così da poter testare la capacità di generalizzazione della rete e confrontare le predizioni con i risultati effettivi.
`ImageDataGenarator` è una classe che permette anche di utilizzare tecniche di *augmentation* per ampliare il dataset. Tali tecniche prevedono di andare a costruire versioni modificate delle immagini stesse. Infatti il dataset, pur essendo numeroso, non lo è al punto tale da rendere sufficientemente buona l'esperienza di image processing. Questa mancanza può essere pertanto colmata andando ad ampliare il dataset con immagini che possano arricchire l'esperienza di training del modello. Tali immagini possono essere modificate in vari modi, ma non tutti questi sono utili nel caso di interesse: infatti alcune tecniche possono generare rumore aggiuntivo e "confondere" il sistema nella

ricerca dei pattern per la rilevazione della polmonite. Infatti è differente classificare immagini biomedicali come i raggi X per capire se vi è o meno una polmonite rispetto ad una classificazione di immagini come quelli di cani o di gatti. Infatti, mentre un gatto può essere visto da varie angolazioni e ciò può essere utile al fine di distinguerlo da un cane, operare una rotazione troppo elevata per andare a riconoscere l'opacità dei polmoni in un'immagine a raggi X, non va ad arricchire il dataset, ma può portare addirittura a un peggioramento dell'accuratezza del modello. Nell'esperienza è stato fatto un training del modello dapprima senza l'utilizzo di tecniche di image augmentation, per poi confrontarlo con un altro training in cui è stato ampliato il dataset.

Occorre pertanto scegliere accuratamente quali parametri inserire. Le tecniche che sono risultate essere utili in questo caso sono:

- **Rescaling**: dato che è stata definita la modalità di colorazione 'grayscale' ogni pixel di ogni immagine avrà un valore che sta nel range [0,255] che con questa tecnica diviene compreso tra 0 e 1. Un primo beneficio è che così tutte le immagini vengono trattate alla stessa maniera. Infatti è possibile che alcune immagini abbiano un range alto di valori dei pixel, altre più basso, ma entrambe condividono poi lo stesso modello e learning rate per il training. In generale è bene fare in modo che il range di valori sia compreso tra 0 e 1 così che ogni immagine contribuisca nella maniera più uniforme possibile per il calcolo della funzione di perdita totale e quindi per l'aggiornamento dei pesi.

- **Rotazione**: è stato appurato che un piccolo range di rotazione per le immagini possa essere utile a migliorare le capacità di generalizzazione del sistema, proprio perchè molto spesso nella pratica clinica le radiografie possono essere lievemente ruotate a causa dei movimenti del paziente. Il range utilizzato in questo caso è tra i -5° e i 5°.

- **Zoom**: come nel caso della rotazione, è prassi che vi siano immagini in cui il torace del paziente sia posizionato più o meno vicino al dispositivo che elabora l'immagine. Quindi un piccolo range di variazione dello zoom delle immagini (in questo caso è stato scelto 0.1) può essere utile.

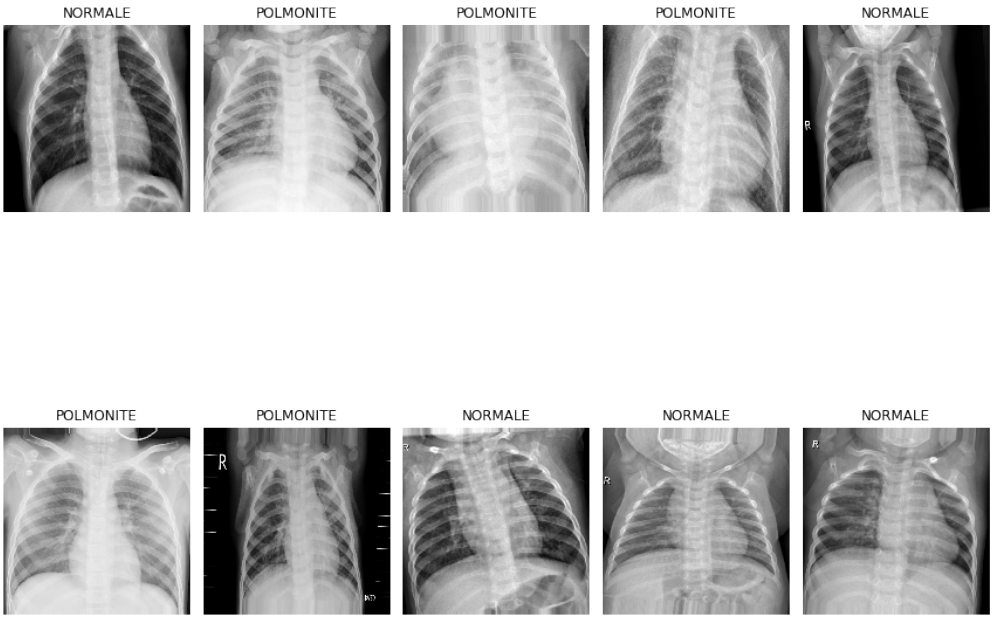
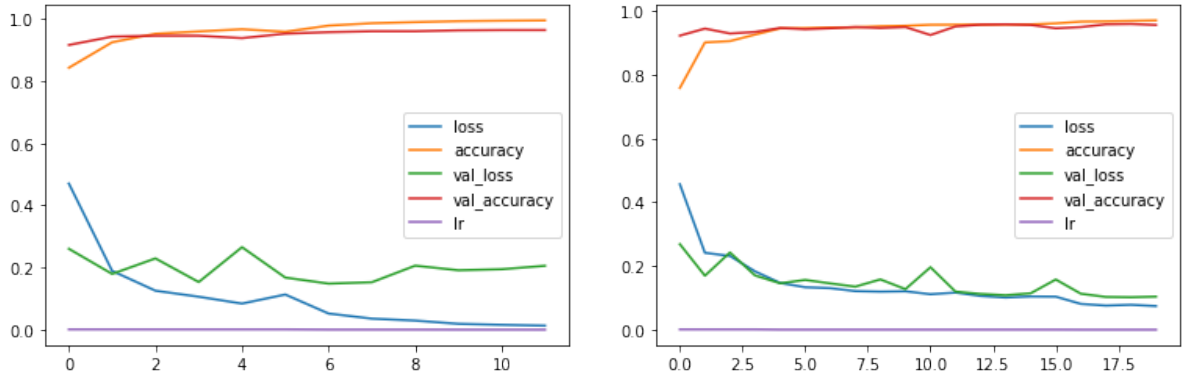


Figura 5.2: Campioni di immagini del dataset di training sui quali sono state aggiunte le tecniche sopra citate.

5.2.6 Fase di fitting

Dopo aver compilato e configurato il modello si passa alla fase di *fitting*, cioè la fase di addestramento vera e propria. Il modello viene dunque allenato tramite le immagini del set di training, con un aggiornamento dei pesi che viene fatto per ogni gruppo di campioni pari alla `batchsize`. Il set di training viene rivisto per un totale di 20 epoche, anche se dopo 10 la capacità di training risulta essere stagnante. Una volta captato questo è stata usata la funzione di callback `EarlyStopping()` che ha fermato il training non appena risultava esserci overfitting. Il metodo `fit()` ci permette anche di scegliere l'insieme di validation su cui la rete può generalizzare. Ciò fa sì che non solo si possa monitorare l'accuratezza nel training, ma anche la capacità di generalizzazione della rete, cercando di capire se questa sta andando in overfitting, underfitting o se il trade-off tra le due è accettabile, così da agire di conseguenza per poterla migliorare. Si può anche definire una lista di chiamate (callbacks) per customizzare il training, come definire un checkpoint dove salvare il modello così da poterlo utilizzare successivamente per elaborare nuove predizioni, senza necessità di allenarlo nuovamente. Inoltre è stata utilizzata una funzione che riduce il valore del learning rate del 30% automaticamente ogni due epoche in quanto

molto spesso il modello beneficia di ciò se l'apprendimento stagna. In questo caso il modello è stato allenato sull'insieme di training e validation definito dal dataset stesso. Il set di testing è l'insieme di immagini sulle quali si va a fare la valutazione finale.



(a) (b)

Figura 5.3: In entrambe le figure è possibile osservare le curve che rappresentano accuracy e loss rispettivamente per training e sul set di validation.

Alla fine si ottengono questi risultati andando a richiamare `cnn.evaluate(test)`:

- Per il caso senza modifiche al set di training risulta che l'accuracy sul training è pari al $(99.61 \pm 0.01)\%$ e quella sull'insieme di validation è del $(94.44 \pm 0.01)\%$.
- Per il caso con le modifiche al set di training risulta che l'accuracy sul training è pari al $(96.99 \pm 0.01)\%$ e quella sull'insieme di validation è del $(94.61 \pm 0.01)\%$.

5.2.7 Predizioni

Infine occorre osservare come il sistema riesce a generalizzare sull'insieme di test. Dato che la funzione di attivazione dello strato finale del modello è la sigmoide, il valore dell'output starà in un range compreso tra 0 e 1 corrispondente alla probabilità che l'immagine su cui è stata fatta la predizione presenti polmonite o meno. Se si vogliono quantificare i falsi positivi e i falsi negativi occorre vedere se l'output della rete è superiore o inferiore a 0.5 e di conseguenza predire se si tratta di polmonite o meno. É possibile visualizzare la matrice di

confusione[3]. Le tabelle 5.1 e 5.2 rappresentano dei report di classificazione in cui è possibile osservare 3 diverse voci:

- Precision = (Predizioni corrette) / (Predizioni corrette + Falsi positivi)
- Recall = Predizioni corrette / (Predizioni corrette + Falsi negativi)
- F1 = (2 * Precision * Recall) / (Precision + Recall)
- support indica il numero di immagini utilizzate come supporto al calcolo delle previsioni.

	precision	**recall**	**f1-score**	**support**
NORMALE	0.99	0.36	0.53	234
POLMONITE	0.72	1.00	0.84	390
accuracy			0.76	624
macro avg	0.85	0.68	0.68	624
weighted avg	0.82	0.76	0.72	624

Tabella 5.1

	precision	**recall**	**f1-score**	**support**
NORMALE	0.97	0.85	0.90	234
POLMONITE	0.91	0.98	0.95	390
accuracy			0.93	624
macro avg	0.94	0.92	0.93	624
weighted avg	0.94	0.93	0.93	624

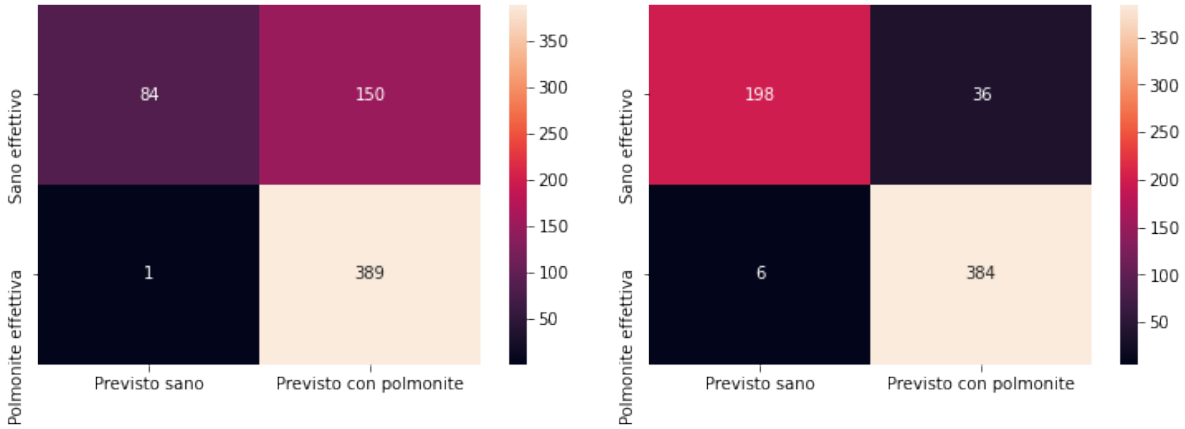
Tabella 5.2

In sintesi l'accuratezza del modello sull'insieme di test vale:

- $(75.80 \pm 0.01)\%$ nella prima esperienza;
- $(93.27 \pm 0.01)\%$ nella seconda.

[3]É un modo per visualizzare in maniera diretta il numero di predizioni corrette/falsi-negativi/falsi-positivi

Dunque sembra che, nonostante nella prima esperienza si abbia ottenuto un'accuratezza migliore sul training, poi le capacità di generalizzazione sono migliori quelle ottenute con la seconda. Ciò significa che nel primo training si è avuto un problema di overfitting.



(a) (b)

Figura 5.4: Matrici di confusione ottenute. (a) è la matrice di confusione del modello con il set di training lasciato come l'originale. Si può vedere che vi sono molti errori nel prevedere il soggetto sano.
(b) è la matrice di confusione con il set di training modificato come nella Figura 5.2. In questo caso vi sono pochissimi errori di predizione.

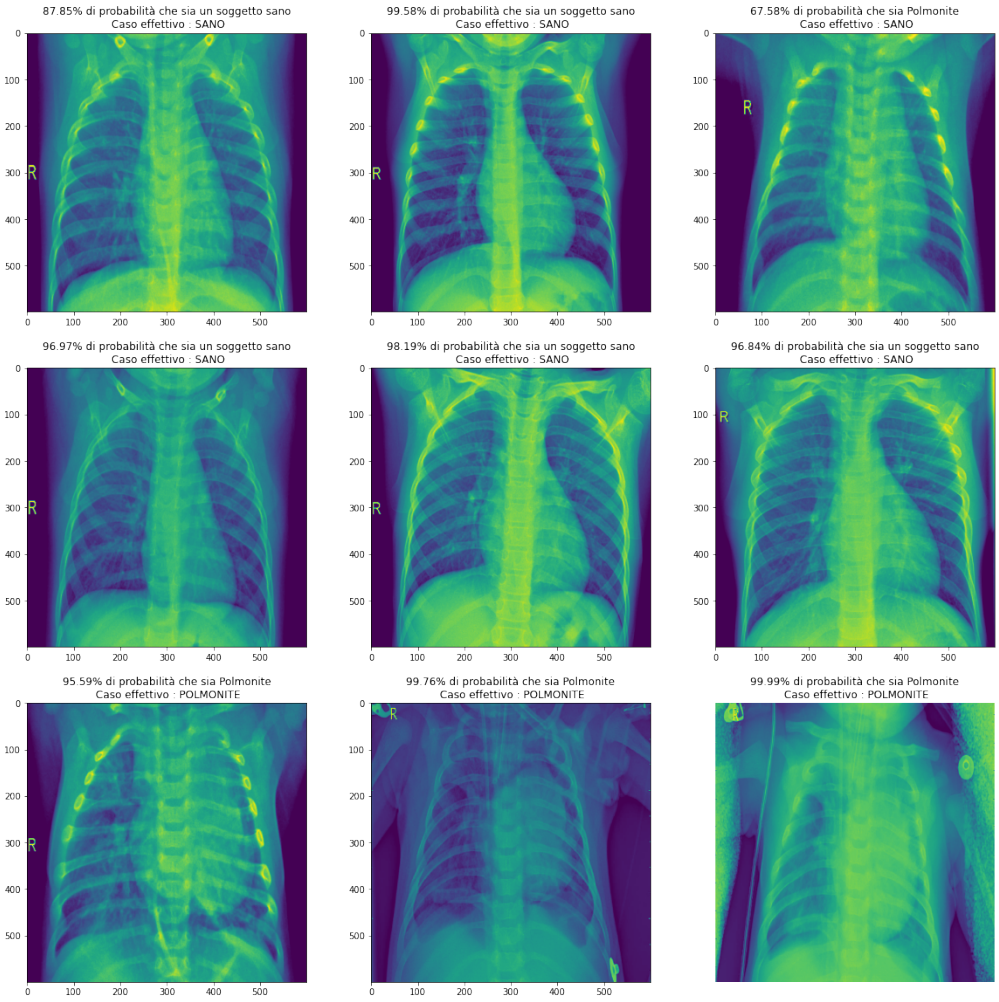


Figura 5.5: Dimostrazione di alcune previsioni.

5.2.8 Considerazioni finali

La rete addestrata con il dataset ampliato risulta produrre buoni risultati. In prima istanza sicuramente grazie all'accuratezza del dataset, ma anche grazie a degli iperparametri ben scelti. Sono state in effetti fatte delle prove con image size di dimensioni 400x400, 500x500, 600x600 e batch size = 8, 16, 32 , ma la migliore accuratezza di generalizzazione si ha con gli iperparametri della sezione 5.2.3. Risultati ragionevoli sono stati trovati dopo 10 epoche, anche se è stato fatto un addestramento anche con 25 e 50 epoche , ma ciò risultava inutile in quanto dalla decima epoca in poi l'accuratezza del modello rimane costante e non migliora. Per evitare ciò è stato utilizzato l'Early Stopping che ha appunto

fermato il modello intorno alla decima epoca. Inoltre è stato provato anche ad aumentare il numero dei canali d'immagine da 1 a 3 ma ciò non ha fatto altro che peggiorare le prestazioni. Probabilmente ciò è dovuto al fatto che per rilevare una polmonite la scala di grigi evidenzia maggiormente l'opacità del polmone. Sicuramente se il classificatore dovesse essere utilizzato in uno studio radiologico, sarebbe un buon mezzo di supporto in quanto il numero di falsi negativi è minore rispetto a quello dei falsi positivi ed è molto piccolo (clinicamente è meglio che sia diagnosticato un falso positivo piuttosto che un falso negativo!). Sotto si mostrano i risultati ottenuti con iperparametri differenti (eccetto il numero di epoche in tutti i casi pari a 20), con le modifiche apportate al set di training della sezione 5.2.5 e che hanno portato alla scelta di quelli alla sezione 5.2.3.

	training accuracy	**validation accuracy**	**test accuracy**
batch = 16 **image size = 500x500** **channels = 1**	95.99%	93.65%	91.98%
batch = 32 **image size = 400x400** **channels = 1**	95.94%	95.00%	91.5%
batch = 8 **image size = 600 x 600** **channels = 1**	96.49%	94.61%	93.26%
batch = 8 **image size = 600 x 600** **channels = 3**	96.32%	93.94%	92.62%

5.3 CNN per la classificazione di risonanze magnetiche cerebrali

6. Conclusioni

7. Codice