Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in
Ingegneria Informatica

Introduzione alle reti neurali e loro applicazione nell'ambito della classificazione medica

Relatore:
Prof. Luca Schenato

Laureando:
Giacomo Masiero

Anno Accademico 2022/2023

Indice

1 Introduzione

Nel corso della storia, l'umanità ha costantemente cercato di svelare i misteri dell'intelletto e della conoscenza umana. Ciò che per gli esseri umani è sempre stato un processo naturale, come il riconoscimento degli oggetti, la lettura della scrittura a mano o la comprensione del linguaggio, ha rappresentato una nuova sfida per i computer tradizionali. Mentre noi, in frazioni di secondo, elaboriamo istintivamente queste informazioni, i computer moderni faticano a ottenere risultati simili. Per questo motivo, attraverso ricerche e sviluppo, sono state create tecnologie che emulano il comportamento del cervello umano, conosciute come *reti neurali*. In questa tesi descriveremo in maniera approfondita il funzionamento delle reti biologiche ed artificiali. In particolare, verrà discusso un tipo di rete chiamato *feedforward*, un modello, privo di cicli, dove le informazioni viaggiano seguendo un'unica direzione. Oltre a questo, verranno descritti gli algoritmi che regolano l'addestramento e l'ottimizzazione di una rete. Infine, verrà presentata un'applicazione pratica in python di una rete neurale per la classificazione tumorale.

2 Reti Neurali

2.1 Neurone biologico

Le reti neurali biologiche sono composte da miliardi di cellule nervose chiamate *neuroni*. Ogni neurone è composto da un corpo cellulare, dendriti e assone. Le dendriti ricevono i segnali da altri neuroni attraverso connessioni chiamate *sinapsi*. Quando il segnale raggiunge il corpo cellulare, questo viene trasmesso al neurone successivo attraverso l'assone. In un sistema biologico umano ogni neurone è tipicamente connesso a un migliaio di altri neuroni, Grazie a questa vasta interconnessione il cervello umano è in grado di elaborare informazioni complesse, eseguire processi di apprendimento e memoria, coordinare il movimento, percepire il mondo circostante e svolgere una vasta gamma di funzioni cognitive ed emotive.

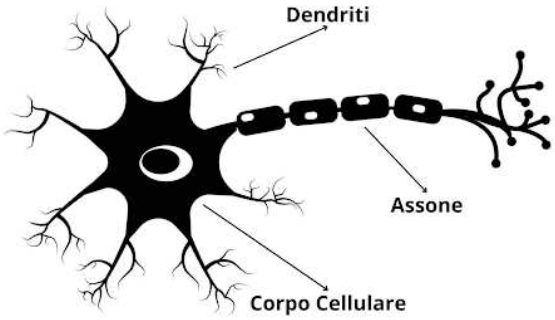

Figura 1: Struttura di un neurone biologico

Ogni neurone è dotato di due stati principali: stato di riposo, stato attivo. Nello *stato di riposo* il neurone non è coinvolto attivamente nella trasmissione di segnali. Durante questo stato, il potenziale di membrana del neurone è stabile, e ciò significa che il neurone non sta trasmettendo segnali elettrici lungo il suo assone. Nello *stato attivo*, invece, il potenziale di membrana del neurone cambia, diventando meno negativo. Se questa eccitazione raggiunge una soglia critica, il neurone genererà un impulso elettrico lungo l'assone noto come *potenziale d'azione*. Una volta arrivato alle sinapsi, il segnale provoca il rilascio di sostanze chimiche chiamate *neurotrasmettitori*, che possono inibire oppure stimolare i dendriti del neurone che le riceve. Grazie a questa capacità, di inibire o stimolare il segnale, il cervello è in grado di adattarsi e modificarsi

in base all'esperienza. Questo fenomeno viene chiamato *plasticità sinaptica*, ed è un processo essenziale che sottolinea la straordinaria flessibilità del sistema nervoso biologico. La plasticità sinaptica non solo supporta l'apprendimento e la memoria, ma anche la capacità del cervello di recuperare da lesioni e di adattarsi a nuove sfide e ambienti.

2.2 Neurone artificiale

Uno fra i modelli più famosi per la descrizione matematica di un neurone biologico è sicuramente il *perceptron*. Esso rappresenta l'unità base di elaborazione di una rete neurale artificiale ed è ampiamente noto per la sua semplicità e versatilità nel modellare relazioni tra input ed output. È importante sottolineare che il perceptron da solo ha limitazioni nell'affrontare problemi semplici; Tuttavia, se integrato in strutture multistrato, dimostra la capacità di risolvere problemi complessi, inclusi quelli correlati all'intelligenza artificiale.

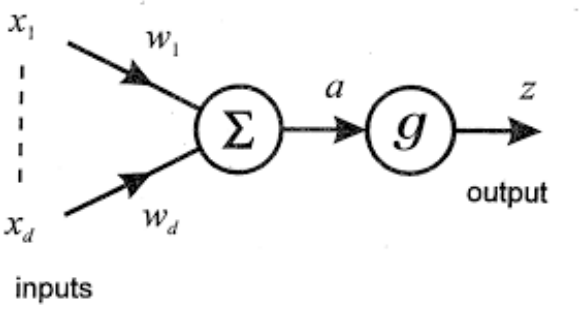

Figura 2: Modello di perceptron

Il modello si compone di due parti fondamentali: Nella prima parte, viene calcolata la somma pesata di tutti gli ingressi $(x_1, x_2, \cdots, x_d)$, utilizzando come pesi $(w_1, w_2, \cdots, w_d)$. Il risultato di questa operazione è definito come:

$$a = \sum_{i=1}^{d} w_i x_i + w_0 \tag{1}$$

dove x_i rappresentano gli input, w_i sono i pesi associati a ciascun input e il parametro w_0 rappresenta il *bias* (che corrisponde alla soglia di attivazione del neurone). E' possibile fissare il valore di $x_0 = 1$ ed ottenere una formulazione

più compatta uguale a:

$$a = \sum_{i=0}^{d} w_i x_i. \tag{2}$$

I valori dei pesi possono variare in positivo o in negativo a seconda dell'effetto desiderato, che può essere eccitatorio o inibitorio. Nella seconda parte del modello, si calcola l'uscita finale z del neurone utilizzando una *funzione di attivazione*, che trasforma la somma a in un valore. Si ottiene quindi:

$$z = g(\sum_{i=0}^{d} w_i x_i). \tag{3}$$

La funzione di attivazione è una componente fondamentale della rete neurale. Questo perché introduce all'interno del modello la non linearità, consentendo in questo modo alla rete di apprendere e generalizzare le informazioni durante il processo di apprendimento che vedremo in seguito. La funzione di attivazione può essere differente in base alla tipologia e allo strato di rete che si vuole realizzare. Fra le più comuni troviamo:

- Sigmoide
$$g(a) = \frac{1}{1 + e^{-a}} \tag{4}$$

- Tangente iperbolica
$$g(a) = \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \tag{5}$$

- ReLu
$$g(a) = max(0, a) \tag{6}$$

- Soglia
$$g(a) = \begin{cases} 1, & \text{se } a > s \\ 0, & \text{se } a \leq s \end{cases} \tag{7}$$

  dove s rappresenta un valore di soglia.

La funzione sigmoide risulta la maggiormente utilizzata nelle reti neurali grazie alla sua non linearità e al suo output limitato. Viene utilizzata soprattutto per risolvere problemi di classificazione binaria, dove è necessario restituire in output un livello di confidenza per una data classe. Viceversa , nelle reti neurali

profonde, è stato dimostrato che la funzione relu permette di ottenere un addestramento migliore anche utilizzando un insieme grande di dati complessi. Il modello *perceptron*, invece, utilizza come funzione di attivazione quella di soglia.

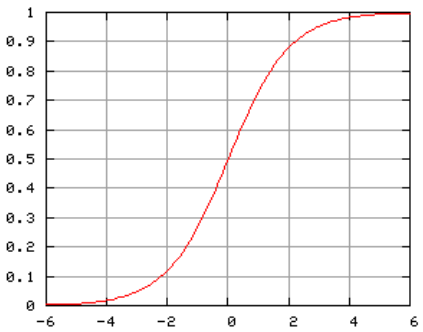

Figura 3: Grafico della funzione sigmoide

2.3 Reti neurali artificiali

Le reti neurali artificiali, dette comunemente *rete neurali*, sono modelli di calcolo ispirati al funzionamento delle reti neurali biologiche. Come specificato precedentemente, in questo capitolo analizzeremo le reti neurali feedforward, un particolare tipo di rete caratterizzato dall'assenza di cicli nelle connessioni. Questo tipo di reti accettano un insieme di parametri indipendenti $x = (x_1, x_2, \cdots, x_d)$ in ingresso, e forniscono in uscita un insieme di variabili dipendenti $y = (y_1, y_2, \cdots, y_c)$. Le uscite della rete sono modellate come funzioni, la cui struttura e comportamento sono influenzati da una serie di valori $w = (w_1, w_2, \cdots, w_d)$ noti come *pesi*. Per eseguire elaborazioni complesse o apprendere informazioni significative dai dati in input, le reti neurali devono essere organizzate in strati. Ciascun strato è composto da una serie di neuroni, interconnessi fra di loro. Generalmente questo tipo di reti sono composte da tre principali strati: strato di ingresso, strato nascosto e strato d'uscita.

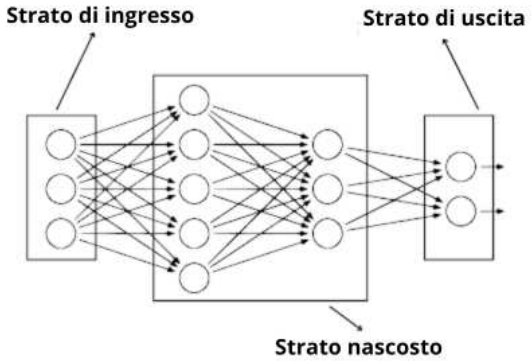

Figura 4: Modello a strati di una rete neurale

Lo *strato di ingresso* ha il compito di ricevere le variabili in input. Ad esempio, in un'applicazione di riconoscimento, ogni neurone di questo strato potrebbe rappresentare una feature dell'immagine. Lo *strato nascosto*, invece, è responsabile dell'elaborazione ed estrazione di caratteristiche significative dai dati in ingresso, catturando le relazioni complesse tra di essi. Infine lo *strato finale*, produce l'output del modello, che può essere una probabilità, una classificazione o un valore numerico a seconda del tipo di problema affrontato. Questa struttura gerarchica degli strati consente alle reti neurali di apprendere e risolvere una vasta gamma di compiti complessi.

2.4 Processo di addestramento

Per poter utilizzare un modello neurale, è necessario avviare il cosiddetto processo di addestramento. Durante questa fase la rete neurale acquisisce conoscenze dai dati passati in input, in modo da generalizzare e produrre risultati accurati su nuovi dati. In questa fase, vengono determinati i *pesi* w delle connessioni tra i neuroni con l'obiettivo di ridurre l'errore tra l'output previsto dalla rete e l'output desiderato. Per poter determinare questi valori è necessario utilizzare un'insieme di dati noto come *training set* o insieme di addestramento, i cui elementi sono coppie $(x^q, t^q) : q = 1, \cdots, n$ dove x^q rappresenta un certo input mentre t^q un certo output o *target*. Il processo di addestramento è composta da diverse fasi fondamentali: Nella prima fase si inizializzano i pesi ad un valore fisso e si propaga l'input attraverso gli strati della rete. Gli input x presi dall'insieme di addestramento o training set vengono pesati da collegamenti sinaptici w e successivamente sommati insieme. La somma risultante per un generico

neurone j è pari a:

$$y_j = g(\sum_{i=0}^{n} w_{ij} x_i) \tag{8}$$

dove g è la mia *funzione di attivazione*. Nella seconda fase avviene il calcolo dell'errore attraverso una specifica funzione di costo. Una delle funzioni di costo più comuni è *l'errore quadratico medio*(Mean Square Error, MSE) che è definito come:

$$E^{(n)} = \frac{1}{2} \sum_{j=1}^{c} (t_j - y_j)^2 \qquad E = \sum_{j=1}^{n} E^{(j)} \tag{9}$$

dove c è il numero di output della rete neurale, n è il numero del pattern, t_j è il valore desiderato dell'output j e y_j è il valore previsto dall'output j. Si può notare come il calcolo dell'errore totale E sia pari alla somma degli errori individuali relativi ai singoli pattern $E^{(n)}$ del training set. Nella terza e ultima fase del processo, si retropropaga l'errore negli stati interni della rete per modificare i pesi dei singoli collegamenti attraverso *l'algoritmo di backpropagation* e di *gradient descent*.

2.4.1 Algoritmo di backpropagation

Tra le tecniche maggiormente utilizzate per l'addestramento delle reti neurali troviamo l'*algoritmo di backpropagation*. Per comprendere appieno questo processo consideriamo una rete neurale composta da tre strati indicati come $(d : n_H : c)$. Data questa rete possiamo considerare l'output k-esimo del neurone di uscita attraverso la formula (8) per ottenere:

$$z_k = g(\sum_{j=0}^{n_H} w_{jk} y_j) \tag{10}$$

ed espandendo il calcolo di y_i abbiamo che:

$$z_k = g(\sum_{j=0}^{n_H} w_{jk} g(\sum_{i=0}^{d} w_{ij} x_i)) \tag{11}$$

dove w_{jk} sono i pesi hidden-output, mentre w_{ij} quelli input-hidden. Ora esaminiamo l'equazione (9) e determiniamo il valore minimo, considerando come variabili solo i pesi delle interconnessioni. Analizziamo per ora i pesi *hidden-output*

e calcoliamo la derivata dell'errore applicando la seguente formula:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(\frac{1}{2}\sum_{c=1}^{s}(t_c - z_c)^2). \tag{12}$$

Sviluppando la derivata dentro la sommatoria e otteniamo

$$\frac{\partial E}{\partial w_{jk}} = (t_k - z_k)\frac{\partial(-z_k)}{\partial w_{jk}} \tag{13}$$

e sapendo che $z_k = g(a_k)$ possiamo riscrivere l'equazione come

$$\frac{\partial E}{\partial w_{jk}} = (t_k - z_k)\frac{\partial(-g(a_k))}{\partial w_{jk}}. \tag{14}$$

Si nota come la sommatoria scompare dalla derivata perché l'unico termine influenzato da w_{jk} è z_k. Applichiamo ora la regola della catena all'equazione (14) e otteniamo

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - z_k)\frac{\partial(g(a_k))}{\partial a_k}\frac{\partial a_k}{\partial w_{jk}} \tag{15}$$

e definiamo

$$g'(a_k) = \frac{\partial(g(a_k))}{\partial a_k} \tag{16}$$

$$y_j = \frac{\partial \sum_{s=1}^{n_H} w_{sk} y_s}{\partial w_{jk}}. \tag{17}$$

Ora sostituiamo i risultati di (16) e (17) all'interno dell'equazione (15) per avere

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - z_k)g'(a_k)y_j. \tag{18}$$

Se a questo punto definiamo

$$\delta_k = (t_k - z_k)g'(a_k) \tag{19}$$

e sostituiamo a (18) otteniamo l'espressione

$$\frac{\partial E}{\partial w_{jk}} = -\delta_k y_j. \tag{20}$$

Consideriamo ora la modifica dei pesi fra *input-hidden*, e ripercorriamo i passaggi fondamentali. Esaminiamo la formula dell'errore (9) e calcoliamo la derivata

rispetto ai pesi w_{ij} per ottenere

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(\frac{1}{2}\sum_{c=1}^{s}(t_c - z_c)^2). \tag{21}$$

Svolgiamo i passaggi precedenti e applichiamo la regola della catena fino ad ottenere

$$\frac{\partial E}{\partial w_{ij}} = -\sum_{c=1}^{s}(t_c - z_c)\frac{\partial(g(a_c))}{\partial a_c}\frac{\partial a_c}{\partial w_{ij}}. \tag{22}$$

Dall'equazione (22) definiamo i seguenti termini

$$g'(a_c) = \frac{\partial(g(a_c))}{\partial a_c} \tag{23}$$

$$\delta_k = (t_c - z_c)g'(a_c). \tag{24}$$

Sviluppiamo ora il termine $\frac{\partial a_c}{\partial w_{ij}}$ dall'equazione (22) e troviamo che

$$\frac{\partial a_c}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(\sum_{r=1}^{n_H} w_{rc}y_r) \tag{25}$$

e visto che $y_r = g(a_r)$ è facile riscrivere

$$\frac{\partial a_c}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(\sum_{r=1}^{n_H} w_{rc}g(a_r). \tag{26}$$

Sapendo che solamente il termine a_j viene influenzato da w_{ij} è possibile riscrivere l'equazione (26) come

$$\frac{\partial a_c}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(w_{jc}g(a_j)). \tag{27}$$

Applichiamo ora la regola della derivazione a catena all'equazione (27) e otteniamo

$$\frac{\partial a_c}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(w_{jc}\frac{\partial g(a_j)}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}})). \tag{28}$$

Se sostituiamo $g'(a_j) = \frac{\partial g(a_j)}{\partial a_j}$ all'equazione (28) e sviluppiamo il termine $\frac{\partial a_j}{\partial w_{ij}}$ abbiamo che

$$\frac{\partial a_c}{\partial w_{ij}} = w_{jc}g'(a_j)\frac{\partial}{\partial w_{ij}}(\sum_{q=1}^{d} w_{qj}x_q)). \tag{29}$$

Sapendo che solamente il termine x_i viene influenzato da w_{ij} possiamo riscrivere l'equazione (29) nel seguente modo

$$\frac{\partial a_c}{\partial w_{ij}} = w_{jc} g'(a_j) x_i. \tag{30}$$

Consideriamo ora l'equazione (22) e, sostituendo le definizioni (24) e (30), è facile ottenere

$$\frac{\partial E}{\partial w_{ij}} = -x_i g'(a_j) \sum_{c=1}^{s} \delta_c w_{jc}. \tag{31}$$

A questo punto definiamo

$$\delta_j = g'(a_j) \sum_{c=1}^{s} \delta_c w_{jc} \tag{32}$$

e riscriviamo l'equazione (31) nel seguente modo

$$\frac{\partial E}{\partial w_{ij}} = -\delta_j x_i \tag{33}$$

Una volta trovati i valori delle derivate è possibile modificare i pesi all'interno di una rete neurale attraverso un algoritmo chiamato *gradient descent*.

2.4.2 Gradient descent

Il *gradient descent*, o discesa del gradient, è un metodo di ottimizzazione numerica utilizzato per trovare il minimo (o massimo) di una funzione. In questo caso specifico, l'implementazione di questo algoritmo prevede l'impiego del vettore del gradiente $\bigtriangledown E(w)$ per trovare il minimo della funzione d'errore ed aggiornare i valori dei pesi. Il processo di gradient descent si articola in diverse fasi che possono essere sintetizzate nei seguenti passaggi:

1. Si inizializzano i pesi con valori casuali o scelti in modo appropriato. I valori verranno poi modificati iterativamente durante il processo di addestramento.

2. Si definisce il vettore gradiente $\bigtriangledown E(w)$ attraverso la seguente formula:

$$\bigtriangledown E(w) = [\frac{\partial E}{\partial w_{11}}, \cdots, \frac{\partial E}{\partial w_{nm}}]. \tag{34}$$

3. Si aggiornano i pesi della rete utilizzano la seguente espressione:

$$w^{(r+1)} = w^{(r)} - \eta \bigtriangledown (E(w^{(r)})) \tag{35}$$

Si nota che, se il gradiente è *negativo*, la funzione errore è decrescente in quel punto, e quindi il valore del peso dovrà aumentare per raggiungere un punto di minimo. Al contrario, se il gradiente è *positivo*, la funzione errore è crescente in quel punto, e quindi sarà necessario diminuire il valore del peso per raggiungere un punto di minimo. Il parametro η viene definito *learning rate*. E' un iper parametro che determina la dimensione dei passi che l'algoritmo di ottimizzazione compie per minimizzare la funzione di errore. Se il valore di risulta troppo piccolo si ottiene una convergenza molto lenta alla soluzione ottimale.

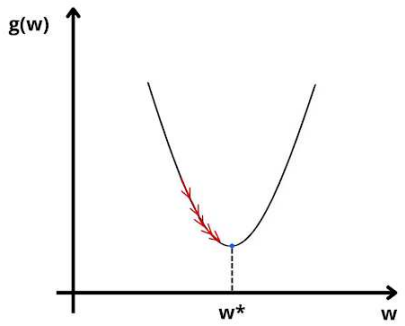

Figura 5: Grafico di ottimizzazione con η piccolo

Se invece risulta troppo grande la soluzione finale può divergere o oscillare e non arrivare mai ad un minimo.

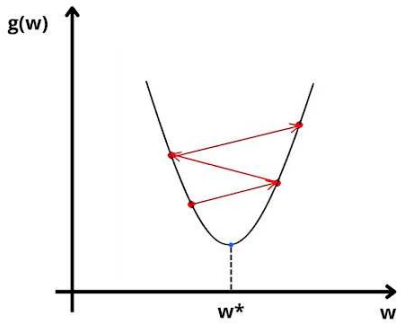

Figura 6: Grafico di ottimizzazione con η grande

4. Si ripetono in maniera iterativa i passi 2 e 3 fino a che la funzione di costo $E(w)$ raggiunge un valore minimo accettabile. È inoltre possibile stabilire un limite massimo per il numero di iterazioni da eseguire.

Sebbene siano disponibili altre strategie di addestramento, la scelta della procedura da utilizzare dipende dall'applicazione specifica e dalla complessità del problema da risolvere. Tuttavia, la descrizione di queste tecniche non verrà qui affrontata.

3 Modelli Neurali

L'uso di reti neurali ha aperto un mondo di possibilità per l'apprendimento automatico e l'elaborazione dei dati complessi. Le reti feedforward, precedentemente esposte, hanno rappresentato un tassello importante nello studio e nello sviluppo di sistemi di intelligenza artificiale. Tuttavia, l'utilizzo di un singolo modello non può essere universalmente adatto per tutte le applicazioni e strutture dati. In questo capitolo, analizzeremo altre tipologie di reti e ne discuteremo le loro caratteristiche e applicazioni specifiche.

3.1 Reti convoluzionali

Le reti convoluzionali (CNN) rappresentano una classe di reti neurali specializzate nella manipolazione di dati visivi. Questo tipo di rete trae ispirazione dal

funzionamento della corteccia visiva umana, che attraverso la sua composizione è in grado di estrarre forme e caratteristiche dai dati di ingresso. Le reti convoluzionali presentano una struttura stratificata, simile a quella delle reti feedforward esaminate precedentemente. Tuttavia, una caratteristica distintiva è la presenza dei cosiddetti *livelli di convoluzione*. Sono contenuti negli strati interni della rete e svolgono il ruolo fondamentale di estrarre le features dai dati passati in input. Per eseguire questo passaggio vengono utilizzati i *filtri*, che a seconda del tipo, sono in grado di identificare caratteristiche diverse dalle immagini. Tipicamente si passa, nei primi strati della rete, a filtri elementari, per poi arrivare,nei livelli successivi, a filtri più specifici e sofisticati. Nella pratica, un filtro è una matrice di valori, appresi durante l'addestramento, che viene fatta scorrere lungo le diverse posizioni dell'immagine di input; Per ogni posizione viene generato un valore di output calcolando il prodotto scalare tra la maschera e la porzione di input coperta. Questo genere di operazione viene chiamata *convoluzione*. Per comprendere meglio consideriamo una matrice input I ed una kernel K:

$$\mathbf{I} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} \qquad \mathbf{K} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}. \tag{36}$$

Applichiamo la convoluzione $I * K$ alle matrici ed otteniamo come risultato:

$$z = (0 \cdot -1) + (0 \cdot -1) + \cdots + (1 \cdot -1) + (2 \cdot -1) = 4 \tag{37}$$

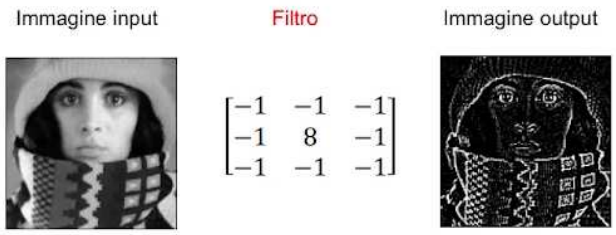

Figura 7: Esempio di filtraggio di un'immagine utilizzando il filtro K

All'interno di questa architettura, oltre ai numerosi strati di convoluzione che includono filtri di diverse intensità e dimensioni, si trovano anche gli *strati di pooling*. Questi strati, di solito alternati con gli strati convoluzionali, hanno lo

scopo di ridurre le dimensioni dell'immagine e conservare le informazioni estratte nei livelli precedenti. Durante questa operazione, una finestra rettangolare con dimensioni specifiche scorre sull'intera immagine per estrarre caratteristiche rilevanti, come il valore massimo (*max-pooling*), il valore minimo (*min-pooling*) o la media dei valori (*average-pooling*). L'integrazione di questi vari livelli forma una rete convoluzionale completa.

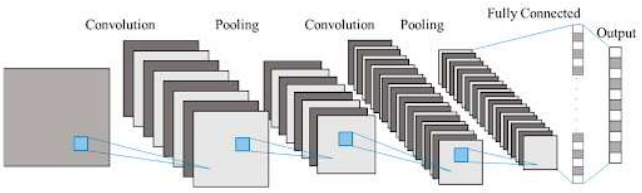

Figura 8: Rete convoluzionale completa

La funzione di attivazione più comunemente adottata per questo tipo di rete è la *relu*, che viene preferita per la sua semplicità ed efficienza durante la fase di addestramento. Essa consente anche attivazioni sparse dei neuroni, creando così una somiglianza con il comportamento osservato nel cervello umano. Negli strati finali, solitamente, troviamo una rete fully connected, capace di analizzare le caratteristiche principali estratte dai livelli precedenti e restituire un risultato finale, che può essere una previsione, una classificazione o un'ulteriore elaborazione dei dati, a seconda dell'obiettivo specifico della rete.

4 Ottimizzazione della rete

4.1 Generalizzazione

Nel secondo capitolo è stata esaminata con dettaglio la metodologia per addestrare una rete neurale in grado di rappresentare efficacemente i dati di addestramento. Tuttavia, per avere un modello utilizzabile nella pratica, è di gran lunga più importante assicurare ottime prestazioni su dati che non sono mai stati presentati durante la fase di addestramento. Il processo di ottimizzazione che rende una rete utilizzabile su un nuovo insieme di dati prende il nome di *generalizzazione*. Durante questa fase il modello viene ottimizzato, modificando il numero di unità nascoste della rete, in modo da estrarre le caratteristiche rilevanti dai dati di addestramento e garantire previsioni affidabili in risposta a nuovi dati di ingresso. La regolazione di questo parametro permette inoltre di

prevenire l'*overfitting*, un fenomeno che si verifica quando il modello si adatta eccessivamente ai dati di addestramento e agli errori in essa contenuti. In questo caso la rete perderebbe la sua capacità di generalizzare e avrebbe prestazioni scadenti in presenza di nuovi dati in ingresso. In sintesi lo scopo di questa fase è trovare un valore ottimale di unità nascoste $\hat{m}$ che permetta alla rete di avere ottime prestazioni e buone capacità di generalizzazione. Per fare questo è necessario considerare due insiemi di dati che chiamiamo *insieme di addestramento* e *insieme di test*. Utilizziamo il primo insieme per addestrare la rete e tracciamo il grafico dell'errore E al variare del numero di unità nascoste m. La funzione che ci si aspetta di trovare è monotona decrescente perché la rete ha la capacità di rappresentare relazioni più complesse nei dati. Presentiamo ora i dati di test alle varie reti addestrate al variare di m nella fase precedente e calcoliamo l'errore E. In questo caso la funzione decresce fino ad un certo valore di m, e poi, a causa dell'overfitting comincia a risalire. Il valore ottimale di unità nascoste $m = \hat{m}$ è il valore che minimizza l'andamento dell'errore rispetto all'insieme di test.

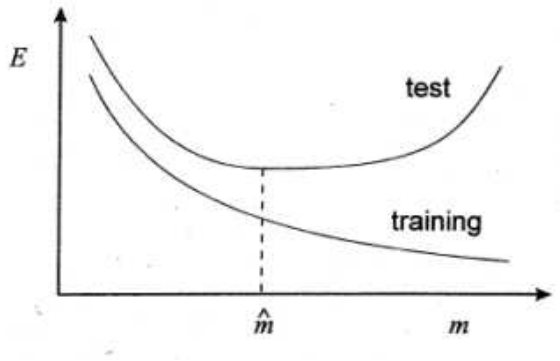

Figura 9: Grafico dell'errore rispetto agli insiemi di addestramento e di test.

Il grafico presentato nella figura (9) rappresenta un caso idealizzato. Nella pratica il grafico che si ottiene è molto meno regolare, e presenta diversi minimi locali.

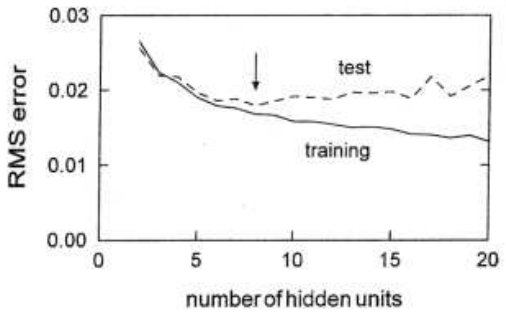

Figura 10: Grafico dell'errore quadratico medio in una applicazione reale.

La selezione dei set di addestramento e di test rappresenta un elemento cruciale per il successo della fase di generalizzazione. Uno dei metodi più ampiamente utilizzati a tale scopo è il *cross-validation*. L'idea alla base è quella di suddividere il dataset in parti più piccole chiamate *fold*, e iterativamente, utilizzare alcuni di questi fold per addestrare il modello e i rimanenti per testarlo. Alla fine, le misurazioni delle prestazioni raccolte durante ogni iterazione, vengono aggregate insieme in modo da fornire una visione più completa e affidabile dell'efficacia del modello.

4.2 Data Preprocessing

Il *pre-processing* , o pre-elaborazione, è una fase fondamentale nella preparazione dei dati destinati all'addestramento di modelli neurali. Utilizzare informazioni grezze non elaborate, porta ad avere reti poco precise e con prestazioni insufficienti. Durante questa fase i dati di input vengono elaborati attraverso una serie di trasformazioni che migliorano la qualità e la rilevanza durante la fase di addestramento. Allo stesso modo, è essenziale sottoporre i dati di output della rete a un processo di rielaborazione chiamato *post-processing*, in modo da essere coerenti con i valori presentati di ingresso.

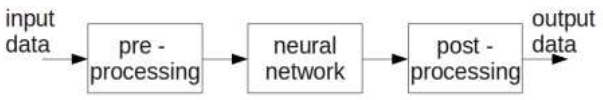

Figura 11: Processo di pre-processing e post-processing di una rete neurale.

Osserviamo che, nell'ambito dell'utilizzo della pre/post-elaborazione, è fondamentale applicare tali trasformazioni sia durante la fase di addestramento che in quella di utilizzo della rete. Analizziamo ora alcune delle tecniche di pre-processing più utilizzate.

4.2.1 Normalizzazione

La *normalizzazione* è una tecnica che uniforma le caratteristiche dei dati e le rende comparabili su una scala comune. L'obiettivo principale è quello di ricondurre tutte le features a un intervallo compreso fra 0 e 1, a1 fine di agevolare la comparazione e l'analisi dei dati stessi. Questo processo è particolarmente utile quando le feature hanno scale molto diverse, che potrebbero influenzare in maniera negativa l'addestramento dei modelli. Il calcolo del valore normalizzato segue la seguente formula:

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{38}$$

dove x è il valore da normalizzare, mentre $x_{\max}$ e $x_{\min}$ sono rispettivamente i valori massimi e minimi della feature.

4.2.2 Riduzione della dimensionalità

La *riduzione della dimensionalità* è una tecnica essenziale di pre-processing, la cui finalità è ridurre le dimensioni dei dati in modo da preservare la massima quantità di informazione possibile. Avere set di dati con alta dimensionalità porta ad una elevata complessità temporale e spaziale in fase di addestramento e un aumento complessivo del rumore. Questo fenomeno viene anche chiamato *curse of dimensionality*. Una delle tecniche più utilizzate per la riduzione della dimensionalità è la PCA (*Principal Component Analysis*). Si proiettano i dati lungo un nuovo sistema di coordinate in modo che la varianza massima sia presente nella prima dimensione, la seconda varianza massima lungo la seconda e così via. Si riduce quindi la dimensionalità, scartando le dimensioni con varianza minore. Per comprendere appieno questo processo consideriamo un insieme di dati di addestramento

$$\left\{x_i \in \mathbb{R}^d : i = 1, \ldots, n\right\} \tag{39}$$

dove d è la dimensionalità dati. Calcoliamo ora la matrice di covarianza

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12} & \cdots & \sigma_{1m} \\ \sigma_{21} & \sigma_{22}^2 & \cdots & \sigma_{2m} \\ \vdots & \vdots & \ddots & \cdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_{nm}^2 \end{bmatrix} \tag{40}$$

dove la varianza σ_{ij} è pari a:

$$\sigma_{ij} = \frac{1}{n}(x_i - \mu_i)^T(x_j - \mu_j) \tag{41}$$

$$\mu_i = \frac{1}{n}\sum_{j=1}^{n} x_j^{(i)} \tag{42}$$

Si seleziona un valore k, che rappresenta la dimensione dello spazio ridotto, e si procede con la costruzione di una matrice di proiezione $W_k \in \mathbb{R}^{d \times k}$, le cui colonne sono formate dagli autovettori di Σ associati ai k più grandi autovalori:

$$W_k = [\varphi_1, \varphi_2, \cdots, \varphi_k] \qquad \lambda_1 > \lambda_1 > \cdots > \lambda_k \tag{43}$$

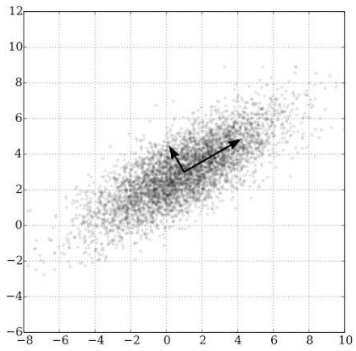

Figura 12: Esempio di riduzione di dimensionalità. I vettori indicati sono gli autovettori della matrice di covarianza.

Dopo aver calcolato la matrice di proiezione W_k, la trasformazione dei dati dallo spazio originario allo spazio PCA avviene attraverso una proiezione geometrica del dato sull'iperpiano generato dalla matrice W_k. La formula da seguire per questo procedimento risulta essere la seguente:

$$q = W_k^T(x - \mu) \qquad \mu = \frac{1}{n}\sum_{1=1}^{n} x_i \tag{44}$$

E' possibile anche effettuare la retroproiezione, che consente di riportare il dato dallo spazio dell'iperpiano PCA allo spazio di dimensione originale d, mediante l'utilizzo della seguente formula:

$$x = W_k q + \mu \tag{45}$$

E' importante sottolineare che il PCA non sempre preserva tutta l'informazione presente nei dati. La quantità di informazione preservata dipende dalla dimensione dello spazio ridotto k, ovvero dal numero di componenti principali mantenuti. Se k è uguale alla dimensione dello spazio originale d, allora l'informazione è completamente preservata. Ma se k è significativamente inferiore alla dimensione dello spazio originale, c'è inevitabilmente una perdita di informazione. Per calcolare il valore minimo di k necessario per mantenere una percentuale t di contenuto informativo basta considerare la seguente formula:

$$k = argmin_z \left\{ \frac{\sum_{i=1}^{z} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \geq t \right\} \qquad \lambda_1 > \lambda_1 > \cdots > \lambda_k \tag{46}$$

La scelta della percentuale t dipende dal tipo di rete che si vuole realizzare, ed è fortemente influenzato dalle specifiche esigenze dell'applicazione.

5 Applicazioni

Nei capitoli precedenti, abbiamo esaminato i principi teorici che governano il funzionamento delle reti neurali, dallo studio delle reti feedforward più elementari fino alla comprensione delle reti convoluzionali. In questo capitolo, ci concentreremo su un'applicazione concreta dei sistemi neurali nel campo della diagnosi medica. In particolare, procederemo alla creazione e alla dettagliata descrizione di un classificatore in grado di identificare diverse tipologie di tumore cerebrale. Questo classificatore sarà implementato utilizzando il linguaggio di programmazione Python e la libreria Tensorflow, sviluppata da Google, la quale offre strumenti avanzati per la costruzione e l'addestramento di reti neurali.

5.1 Classificatore Tumorale MRI

Negli ultimi anni, l'evoluzione delle tecnologie neurali ha aperto la strada a nuovi approcci nella diagnosi e nella cura di patologie. In particolare, nell'ambito della diagnosi tumorale, le reti neurali hanno la possibilità di rivoluzionare il modo in cui i medici identificano e valutano i tumori. Tradizionalmente, il processo di individuazione e classificazione dei tumori richiede l'esperienza di un medico, il quale analizza manualmente le immagini diagnostiche, valuta le caratteristiche e fornisce una valutazione basata sulla sua esperienza clinica. Tuttavia, questo approccio presenta limiti legati alla soggettività umana, alla variabilità e alla possibile fatica dell'operatore. Con l'utilizzo di una rete neurale, invece, è possibile riconoscere modelli e correlazioni difficilmente individuabili dall'occhio umano. Nello specifico, grazie alle reti convoluzionali CNN è possibile ottenere risultati efficaci nell'analisi di immagini mediche, come quelle ottenute da scanner o da esami di imaging ad alta risoluzione. In questo capitolo realizzeremo un classificatore tumorale MRI, discutendo in maniera approfondita la stuttura dei dati di addestramento e della struttura della rete.

5.1.1 Dataset

Per la realizzazione di un algoritmo di intelligenza artificiale è fondamentale disporre di un set di dati adeguatamente grande e vario. In questo caso specifico è stato utilizzato un dataset di dominio pubblico chiamato *Brain Tumor Classification MRI*. All'interno del dataset sono presenti immagini di risonanze magnetiche, raffiguranti sia pazienti in condizione di salute che pazienti affetti da tumore. Le immagini sono state suddivise in tre categorie, ciascuna rappresentante un tipo specifico di tumore: il *glioma*, una neoplasia caratterizzata da vari livelli di aggressività; il *meningioma*, un tumore che si sviluppa sulle membrane che rivestono cervello e midollo spinale; l'*adenoma pituitario*, un'alterazione che colpisce l'ipofisi e influisce sulla produzione di ormoni. In aggiunta, è stata inclusa una quarta categoria che rappresenta il *cervello in condizione di salute*, il quale funge da punto di riferimento anatomico per la valutazione delle diverse anomalie nelle categorie tumorali.

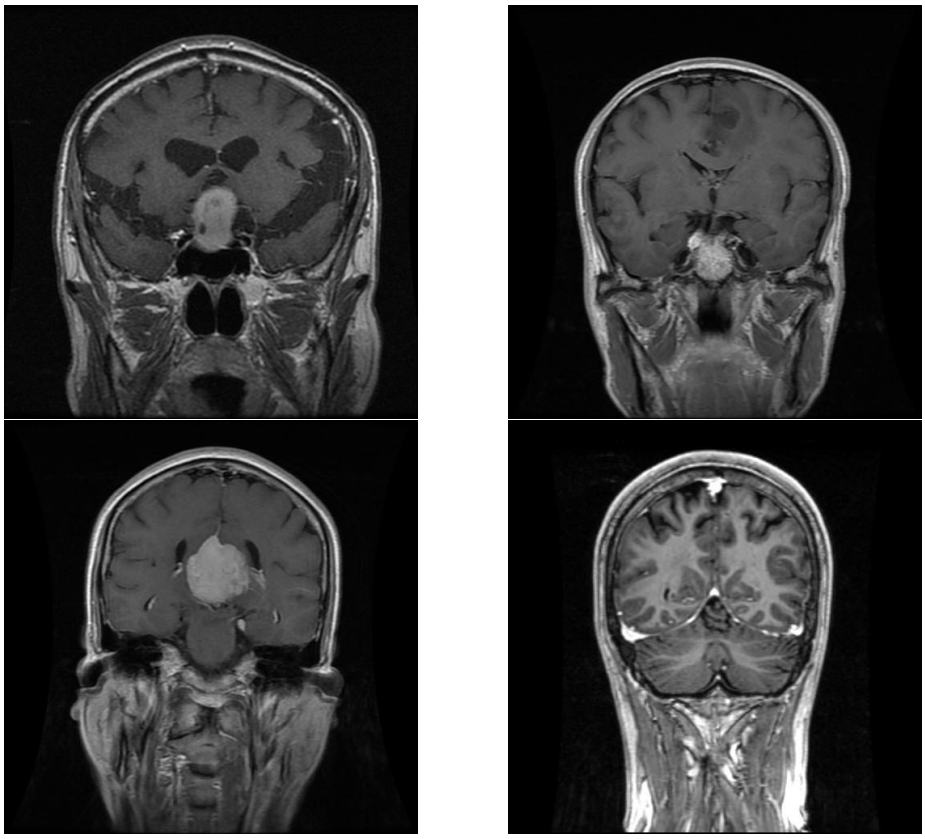

Figura 13: In alto a sinistra, un adenoma pituitario; in alto a destra, un glioma; in basso a sinistra, un meningioma; in basso a destra, un cervello sano.

Le immagini nel dataset sono acquisite da varie prospettive e organizzate in due cartelle principali: *Training*, che ospita le immagini destinate alla fase di addestramento, e *Testing*, che contiene le immagini utilizzate per la fase di test. All'interno di ciascuna cartella, sono presenti quattro sottocartella che contengono le diverse categorie di immagini presentate precedentemente.

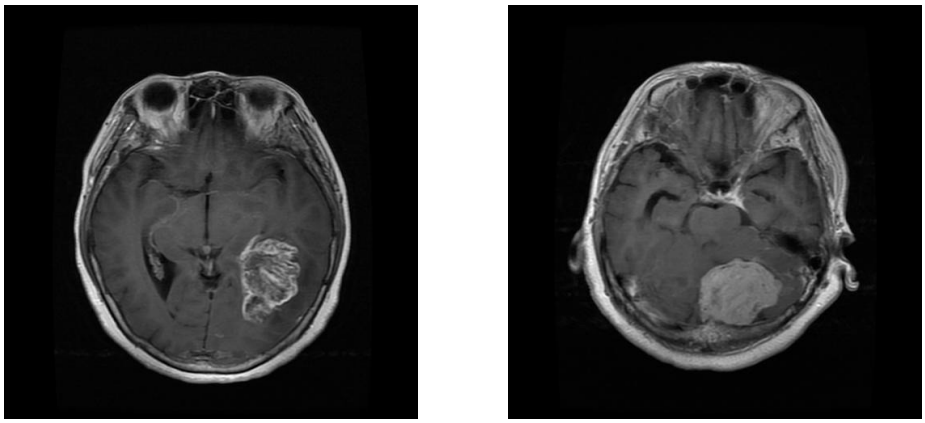

Figura 14: A sinistra un immagine dall'alto di un glioma, mentre a destra un immagine dall'alto di un meningioma.

5.1.2 Preprocessing dei dati

Per utilizzare i dati all'interno di una rete neurale, soprattutto nel caso di immagini, è necessario effettuare un'operazione di pulizia e di pre-elaborazione. Per fare questo, prendiamo in considerazione il seguente codice:

```
images=[]
labels=[]
IMAGE_SIZE=(150,150)
(for).....
label=class_names_label[folder]
img_path=os.path.join(os.path.join(path,folder),file)
image=cv2.imread(img_path)
image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
image=cv2.resize(image,IMAGE_SIZE)
images.append(image)
labels.append(label)
(end for).....
```

Le immagini vengono importate da ciascuna cartella, convertite nel formato RGB e ridimensionate a 150x150 pixel. Questa operazione consente di ridurre la complessità computazionale del modello e velocizzare l'addestramento della rete. Ciascuna immagine viene quindi inserita all'interno di un array chiamato *images*, mentre la corrispondente etichetta viene posizionata in un array separato chiamato *labels*, occupando la stessa posizione dell'immagine.

L'array *images* viene convertito in valori floating point a 32 bit, comune nell'ambito dell'elaborazione di immagini digitali. L'array è composto da un insieme di valori che rappresentano, per ogni pixel dell'immagine, l'intensità di ciascun colore nella scala RGB(da 0 a 255). Similarmente, l'array *labels* subisce una trasformazione simile, ma in questo caso i valori vengono convertiti in interi a 32 bit.

```
images=np.array(images,dtype='float32')
labels=np.array(labels,dtype='int32')
```

Si creano così due set di dati distinti, uno per i dati di allenamento e l'altro per i dati di test. Questi insiemi contengono sia le immagini che le relative etichette, e costituiscono la base per la creazione della rete neurale.

```
(train_images,train_labels),(test_images,test_labels)=
    load_data()
```

5.1.3 Addestramento della rete

Per la rete neurale, si utilizza un modello convoluzionale CNN, che elabora le informazioni e classifica il tipo di immagine passato in input. Viene creato un modello sequenziale utilizzando Keras, una libreria open source per apprendimento automatico e deep learning. L'inizio della rete presenta uno strato di convoluzione, un processo che coinvolge 32 filtri di dimensioni 3x3. Questi filtri, attivati dalla funzione *relu*, funzionano come occhi e scrutano l'immagine per rilevare specifiche caratteristiche visive come linee e bordi. A seguito di questa fase, vengono applicati strati di max pooling che riducono le dimensioni dell'immagine, mantenendo le caratteristiche più importanti. Questa sequenza di passaggi viene poi ripetuta, permettendo alla rete di catturare dettagli sempre più complessi.

```
model=tf.keras.Sequential([  tf.keras.layers.Conv2D
    (32,(3,3),activation='relu',input_shape=(150,150,3)
    ),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Conv2D(32,(3,3),activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128,activation=tf.nn.relu),
tf.keras.layers.Dense(4,activation=tf.nn.softmax)
```

```
])
model.
compile(optimizer='adam',loss='
    sparse_categorical_crossentropy',metrics=['accuracy
    '])
```

Il cuore della rete è costituito dai *livelli densi*, i quali svolgono un ruolo cruciale nell'elaborazione delle informazioni apprese nelle fasi precedenti. Questi strati, diversi dalle convoluzioni e dal max pooling, sono completamente connessi. Il primo strato denso presenta 128 unità, ognuna delle quali cerca modelli complessi nelle informazioni estratte precedentemente. L'attivazione *relu* introduce non linearità nelle analisi. L'ultimo strato denso, con 4 unità, rappresenta le categorie di classificazione desiderate. Qui entra in gioco la funzione *softmax* che converte le attività di queste unità in probabilità, indicando quale categoria è più probabile per l'immagine. In breve, questa rete sfrutta le convoluzioni iniziali per cogliere tratti visivi, mentre gli strati densi, attraverso la loro completa connessione, apprendono e classificano le immagini in modo preciso. Per il modello si utilizza la funzione di perdita *sparse categorical cross entropy*, comunemente utilizzate per i problemi di classificazione multiclasse e *adam*, una variante più efficiente dell'algoritmo di gradient descent.

Dopo aver definito il modello, è essenziale mescolare casualmente tutti gli esempi e le rispettive etichette. Questo passaggio è cruciale per garantire la robustezza della rete, evitando che essa impari pattern basati sull'ordine dei dati.

```
train_images,train_labels=shuffle(train_images,
    train_labels,random_state=25)
```

Si procede quindi con la fase di addestramento, dove vengono elaborati gruppi di 128 immagini alla volta per 5 epoche. Per la validazione del modello si utilizza un set di convalida che corrisponde a un quinto dell'insieme totale dei dati di addestramento(circa il 20% del set).

```
history=model.fit(train_images,train_labels,batch_size
    =128,epochs=5,validation_split=0.2)
```

Si traccia, per ogni epoca, il grafico dell'accuratezza sui dati di addestramento e sui dati di validazione, attraverso la funzione *plot_accuracy_loss*, realizzata utilizzando la libreria *matplotlib*. Si osserva che dopo la terza epoca, l'accuratezza sui dati di addestramento continua a aumentare, mentre sui dati di validazione

sembra stabilizzarsi e subire una lieve diminuzione. Questo fenomeno suggerisce la presenza di overfitting, in cui il modello sta imparando troppo, adattandosi ai dettagli e ai rumori dei dati. In questo caso, visto l'andamento del grafico, è preferibile utilizzare solamente tre epoche di addestramento.

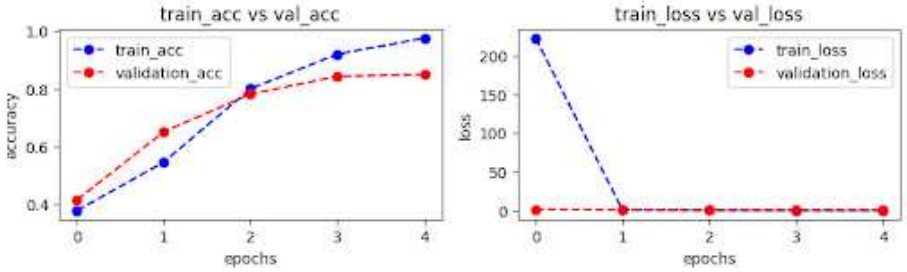

Figura 15: Sulla sinistra il grafico di accuratezza; Sulla destra il grafico della loss.

Dopo aver completato il processo di addestramento, risulta cruciale sfruttare i dati di test per valutare le prestazioni complessive di accuratezza e loss del modello. Tale approccio fornisce un quadro completo delle performance della rete neurale quando è confrontata con dati completamente nuovi.

```
test_loss=model.evaluate(test_images,test_labels)
```

Il risultato che si ottiene è un valore di loss pari a 2.65 mentre una accuracy di 0.7124. Da queste informazioni si capisce che il modello è stato in grado di predire circa il 71.24% delle immagini presenti nel test set. Questo valore offre sicuramente spazio al miglioramento, attraverso un modello più complesso, o dati più specifici. Per una valutazione più approfondita è utile considerare anche le metriche di precision e recall: La *precision* misura la proporzione di istanze classificate come positive dal modello che sono effettivamente corrette. In altre parole, rappresenta la percentuale di veri positivi rispetto a tutti gli elementi che il modello ha classificato come positivi. La *recall* misura la proporzione di veri positivi predetti correttamente dal modello rispetto a tutti gli elementi che sono effettivamente positivi. Rappresenta la percentuale di veri positivi individuati correttamente rispetto a tutti gli elementi positivi presenti nei dati. Di seguito sono presenti i dati raccolti durante questa fase.

Tipo	Precision	Recall
no_tumore(0)	0.59	0.96
glioma(1)	1.0	0.20
meningioma(2)	0.66	0.83
pituitari(3)	0.77	0.62

Notiamo che il modello ottiene un alto valore di recall per il tipo *no_tumore*, il che significa che è molto bravo a identificare i casi in cui non c'è un tumore, ma la precision è leggermente più bassa, il che indica che potrebbe classificare alcuni casi come *no_tumore* che in realtà sono tumori. Al contrario, per il tipo *glioma*, la precision è molto alta, il che indica una grande precisione nella classificazione dei casi come *glioma*, ma il recall è notevolmente più basso, suggerendo che il modello potrebbe non riuscire a individuare tutti i casi di *glioma*. Per i tipi *meningioma* e *pituitari*, il modello mostra un buon equilibrio tra precision e recall, con valori ragionevoli per entrambe le metriche.

6 Conclusioni

Nel corso di questa ricerca, abbiamo esaminato in dettaglio le fondamenta teoriche delle reti neurali feedforward, analizzando nello specifico l'architettura e il funzionamento dei neuroni artificiali e il processo di apprendimento attraverso gli algoritmi di backpropagation e gradient descent. Abbiamo sottolineato l'importanza di una corretta progettazione di rete, dalla scelta delle funzioni di attivazione alla gestione dell'overfitting e delle prestazioni. Abbiamo descritto, inoltre, le reti convoluzionali, e di come possano essere utilizzate per risolvere problemi di classificazione e riconoscimento di immagini. In particolare, abbiamo esaminato in modo approfondito come queste reti possano essere applicate in un contesto pratico di classificazione medica. Il programma in questione aveva lo scopo di identificare la tipologia tumorale di un paziente dalle immagini mri ricevute in ingresso. In conclusione possiamo affermare che le reti neurali artificiali manterranno un ruolo cruciale nell'evoluzione della tecnologia e dell'informatica. Grazie alla ricerca continua in questo campo, ci attendiamo un costante progresso nelle capacità di queste reti, aprendo la strada a soluzioni innovative e all'applicazione di intelligenza artificiale in ambiti sempre più diversificati e cruciali per il progresso scientifico e tecnologico.

Riferimenti bibliografici

[1] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[2] Sartaj. *Brain tumor classification (MRI)*. Mag. 2020. URL: `https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri`.