Università
Ca'Foscari
Venezia

Corso di Laurea
in Data Science

Tesi di Laurea

CONTINUAL LEARNING: ARCHITECTURAL APPROACH BY PROTOTYPE AND ELM

Relatore

Andrea Torsello

Assistente supervisore

Francesco Pelosin

Laureando

Giosuè Zannini

873810

Anno accademico 2020/2021

Indice

Capitolo 1

Introduzione

1.1 Problema

In questa tesi tratto il problema del continual learning, in quanto la caratteristica desiderabile in un AI[1] è la capacità, di essere in grado di aggiungere conoscenza senza perdere il sapere acquisito in precedenza. Il modello che viene a crearsi dovrebbe avere due caratteristiche principali: stabilità e plasticità. La prima permette al sistema di non dimenticare la conoscenza pregressa e la seconda rende il modello flessibile e quindi capace di aggiungere esperienza. Il continual learning viene utilizzato per cercare di risolvere il problema del catastrophic forgetting che affligge qualsiasi intelligenza artificiale. Infatti, un sistema che cerca di aggiungere conoscenza tende a dimenticare completamente il sapere appreso in precedenza dopo aver aggiunto nuove informazioni. Questo concetto è presente fin dalla nascita dell'intelligenza artificiale ma solo alla fine del '900 si è iniziato a studiarlo in maniera più approfondita. Uno dei primi studi risale al 1996 ed è stato compiuto da Sebastian Thrun, successivamente Ruvolo e Eaton hanno introdotto "Efficient lifelong learning algorithm" per migliorare l'apprendimento multi task basandosi sulla strategia della regularization utilizzando l'informazione di Fisher.

[1] Acronimo di Artificial Intelligence

Attualmente per mitigare questo problema esistono vari approcci, i tre maggiormente utilizzati sono:

- Regularization: prevede di aggiustare i parametri all'interno del modello per aggiungere i vari task. Questa tecnica non è molto efficace.
- Rehersal: prevede di utilizzare un unico modello, che nel caso in cui ci siano nuovi task da apprendere, riallena AI aggiungendo le istanze dei nuovi task al processo di train. Questa tecnica è molto efficace, ma ha una complessità spaziale molto esosa in quanto deve tenere in memoria tutte le istanze di train e con l'aumentare dei task appresi aumenterà anche la memoria occupata dal modello.
- Architectural: prevede di utilizzare un modello per task, in questo modo non sarà necessario mantenere in memoria le istanze di train e sarà sufficiente che il sistema scelga il modello adatto per la previsione da eseguire. Questa tecnica è molto efficace, ma avendo N task si avranno molti parametri e si avrà il problema di identificare la AI corretta da utilizzare.

Per cercare di risolvere questo problema ho utilizzato l'ultimo approccio precedentemente elencato, con l'accortezza di adoperare una tecnica ad hoc per la scelta del modello corretto da adoperare nella previsione. Prima di sviluppare a pieno il progetto, reputo necessario dare una breve ma esaustiva introduzione degli strumenti che utilizzerò, in modo da poter capire con più chiarezza il lavoro svolto.

1.2 Artificial Neural Network

Le reti neurali sono un sottoinsieme del machine learning[2] e sono il cuore degli algoritmi di deep learning[3]. Il loro nome e la loro struttura è ispirata al cervello umano, in quanto ognuna di esse è un modello matematico composto da neuroni artificiali, anche chiamati percettroni, che si ispirano a una rete neurale biologica.
Prima di esaminare nel dettaglio le caratteristiche delle reti neurali, voglio fare una piccola e semplice introduzione su come funzionino le reti biologiche, così da poter poi capire le assonanze con quelle artificiali.

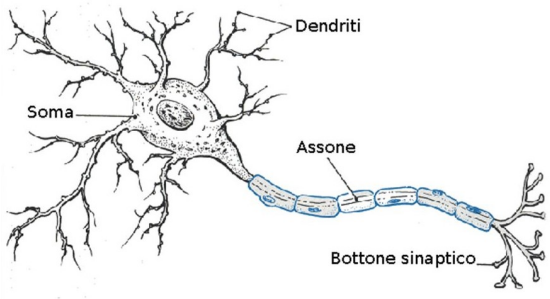

FIGURA 1.1: Neurone biologico

- Dendriti: ricevono il segnale da altri neuroni
- Soma: processa le informazioni
- Assone: trasmette l'output del neurone
- Bottone sinaptico: punto di connessione con altri neuroni

[2]E' un sottoinsieme dell'intelligenza artificiale che si occupa di creare sistemi che apprendono o migliorano le performance in base ai dati che utilizzano

[3]E' una sotto categoria del machine learning e indica quella branca dell'intelligenza artificiale che fa riferimento agli algoritmi ispirati alla struttura e alla funzione del cervello chiamate reti neurali artificiali

Di base, un neurone prende le informazioni in input, le processa e trasmette l'output ai neuroni seguenti. Questo è il funzionamento ad alto livello di un singolo percettrone, ma il cervello umano ha in media 10^{11} neuroni interconnessi tra loro, i quali sono a loro volta suddivisi in strati, dove ogni strato ha le proprie regole e il proprio compito.
Dopo questa piccola introduzione su come funziona una rete biologica posso passare alla prima rappresentazione artificiale di un percettrone.
Il primo modello di rete neurale fu sviluppato da MuCulloch e Pitts, i quali nel 1943 pubblicarono "A Logical Calculus of Ideas Immanent in Neuron Activity" nel quale indicarono un modello computazionale semplificato sul possibile funzionamento dei neuroni biologici per il calcolo di operazioni complesse usando operazioni logiche (and, or, not).

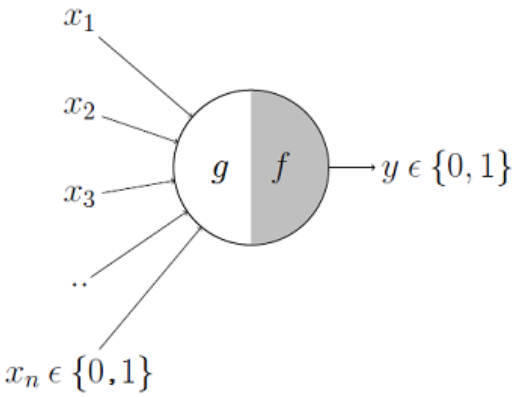

FIGURA 1.2: Neurone MuCulloch-Pitts

Esso era diviso in due parti, la prima parte g prendeva l'input il quale poteva essere un eccitatore o un inibitore, ed in seguito eseguiva un'aggregazione. Dato il valore, la seconda parte f prendeva una decisione. Gli inibitori avevano il massimo potere sul processo decisionale, quindi un solo inibitore positivo portava ad una determinata decisione, invece gli input eccitanti implicavano una scelta quando venivano combinati tra loro.

Oggi giorno le ANN (Artificial Neural Network) sono molto più complesse, infatti ne esistono di diverse tipologie, ma la più utilizzata è la feedforward[4]. Di norma sono composte da diversi layer (almeno tre, in quanto abbiamo: un layer di input, uno o più hidden layer e un layer di output) dove ognuno di essi è costituito da diversi neuroni, ed ogni percettrone assume un valore detto attivazione che ne indica l'importanza. Ogni nodo è collegato a tutti i nodi del layer successivo, e ogni collegamento ha un peso, il quale insieme al valore del neurone precedente andrà ad influenzare i neuroni negli strati seguenti. Infatti il valore di ogni percettrone (escluso il layer di input), è dettato dalla somma dei pesi moltiplicati per i valori dei neuroni nel layer precedente, ai quali viene sommata la soglia che identifica quanto alta deve essere la sommatoria dei pesi, in modo che il neurone cominci ad attivarsi. La formula che identifica il valore per ogni neurone di tutti i layer successivi a quello di input è:

$$\forall i = 0, \ldots, K : a_i^{(l+1)} = \delta(\sum_{n=1}^{N}(a_n^{(l)} w_{i,n}) + b_i^{(l+1)}) \tag{1.1}$$

La quale può essere riscritta in forma matriciale per una più facile comprensione:

$$\delta\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K,0} & w_{K,1} & \cdots & w_{K,N} \end{bmatrix} \begin{bmatrix} a_0^{(l)} \\ a_1^{(l)} \\ \vdots \\ a_N^{(l)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{bmatrix}\right) \tag{1.2}$$

[4]E' una ANN dove le connessioni tra le unità non formano cicli. In questa rete le informazioni si muovono solo in avanti, rispetto a nodi d'ingresso, attraverso nodi nascosti fino ai nodi d'uscita

Dove:

- δ indica la funzione d'attivazione utilizzata nel layer $l + 1$.
- a indica il valore di ogni neurone nel layer l.
- w è il peso dell'arco che collega $a^{(l)}$ al neurone $a^{(l+1)}$.
- N indica la quantità di neuroni che sono collegati attraverso un arco ad $a^{(l+1)}$.
- K indica la quantità di neurone presenti nel layer $l + 1$.

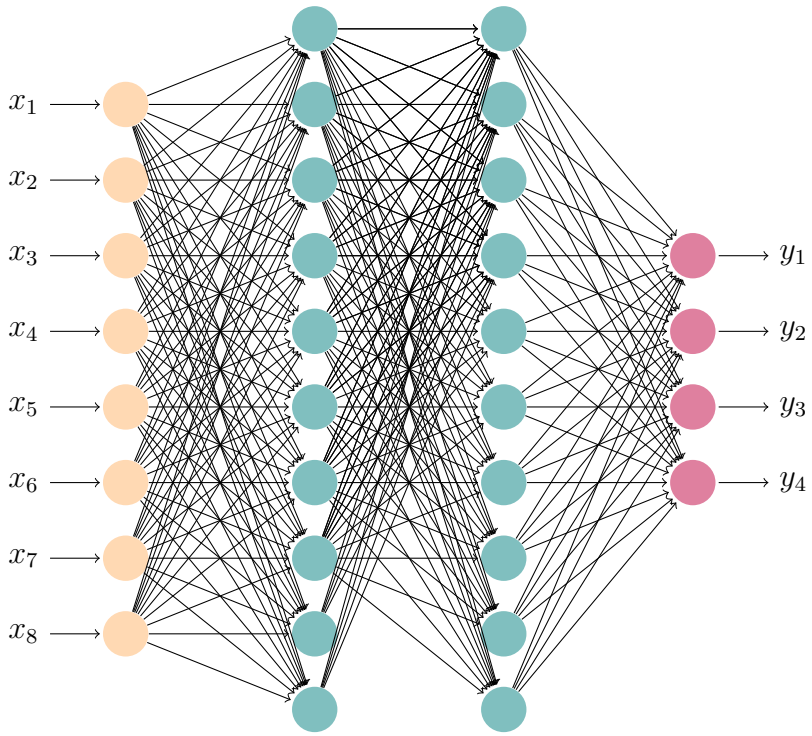

FIGURA 1.3: Esempio di rete neurale feed forward

1.2.1 Funzione d'attivazione

Una funzione d'attivazione altro non è che un'equazione matematica applicata a qualsiasi neurone appartenente ai hidden e output layer, avente il compito di definire l'attivazione di ogni nodo. Essa ha un ruolo chiave nelle reti neurali, in quanto se non fosse presente, le ANN equivarrebbero a un modello di regressione lineare[5]. Infatti, un artificial neural network composta da svariati hidden layer aventi solo funzioni d'attivazione lineari, equivarrebbe ad una rete con un singolo layer (in quanto una combinazione di funzioni lineari produce a sua volta una funzione lineare). Lo scopo delle reti neurali è quello di essere in grado di approssimare qualsiasi tipo di funzione e per farlo, è necessario introdurre un fattore di non linearità.
Le due più note funzioni d'attivazione non lineari sono:

- Funzione sigmoide:
$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.3}$$
E' una funzione molto utilizzata nel layer di output nei problemi di classificazione binaria, in quanto normalizza l'output di ogni classe in un intervallo $[0; 1]$.

- Funzione ReLU:
$$f(x) = \max(0, x) \tag{1.4}$$
E' una funzione molto utilizzata nei layer intermedi ed è molto semplice da calcolare. Questa semplicità, unita al fatto di ridurre drasticamente il problema del vanishing gradient[6] la rende un ottima funzione d'attivazione da utilizzare. Un altro beneficio non trascurabile è il fatto di ridurre il numero di neuroni attivi, in quanto tutti i valori negativi vengono portati

[5]Rappresenta un metodo di stima del valore atteso condizionato da una variabile dipendente Y, dati i valori di altre variabili indipendenti X_i
[6]E' un fenomeno che crea difficoltà nell'addestramento delle reti

a zero. Questo rende le attivazioni sparse, più efficienti e di conseguenza si avrà una rete più leggera.

1.2.2 Funzione di costo

La funzione di costo ha il compito di determinare l'adattamento del modello ai dati. La caratteristica desiderabile è trovare i giusti valori per i pesi e soglie in modo che la rete approssimi il più possibile le $y(x)$ per tutti i valori di input x. Per determinare quanto efficientemente la rete approssimi i risultati esistono svariate funzioni, tra cui:

- Mean Square Error:

$$MSE(x) = \frac{1}{2n} \sum_{x} ||y(x) - \hat{y}(x)||^2 \tag{1.5}$$

  Viene utilizzata maggiormente in problemi di regressione.

- Cross Entropy Loss:

$$H(x) = -\sum_{x} y(x) \log(\hat{y}(x)) \tag{1.6}$$

  Viene utilizzata maggiormente in problemi di classificazione multi-classe.

Quando la funzione di costo generica $f(x) \approx 0$, più precisamente quando $y(x)$ è approssimativamente uguale all'output $\hat{y}(x)$ significa che il modello sta utilizzato i corretti pesi e soglie per produrre ottime predizioni. Contrariamente se $f(x)$ dista molto dallo 0 significa che le $y(x)$ sono molto distanti dalle previsioni $\hat{y}(x)$ computate dalla rete. Quindi l'obbiettivo della ANN è quello di minimizzare la funzione di costo in modo da rendere più accurate possibili le previsioni, e per farlo deve trovare i giusti valori da inserire nella rete. Per questo compito viene utilizzato l'algoritmo di discesa del gradiente.

1.2.3 Discesa del gradiente

La discesa del gradiente è un processo iterativo utilizzato nella backpropagation per identificare minimi locali (o globali in caso la funzione sia convessa) data la funzione di costo $C : \mathbb{R}^n \to \mathbb{R}$. Questo è possibile grazie al gradiente, il quale indica direzione ed intensità di massima crescita della funzione di costo.

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w_{11}^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w_{K,N}^{(1)}} \\ \frac{\partial C}{\partial b_1^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial b_N^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w_{K,N}^{(L)}} \\ \frac{\partial C}{\partial b_N^{(L)}} \end{bmatrix} \tag{1.7}$$

Dove:

- K indica la quantità di archi entranti in ogni nodo di ogni layer.
- N indica la quantità di neuroni per ogni hidden e output layer.
- L indica la quantità di layer presenti.
- w, b indicano rispettivamente pesi e soglie presenti nella rete.

La derivata parziale di ogni peso e bias viene calcolata utilizzando la regola della catena[7]. Dove $\forall i, j$ con $i = 1, \ldots, K$ e $j = 1, \ldots, N$ risulta per i pesi la seguente derivata parziale:

$$\frac{\partial C}{\partial w_{i,j}^{(l)}} = \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial C}{\partial a_j^{(l)}} \tag{1.8}$$

e per le soglie la successiva derivata parziale:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial C}{\partial a_j^{(l)}} \tag{1.9}$$

con:

$$\frac{\partial C}{\partial a_j^{(l)}} = \sum_{j=1}^{n} \frac{\partial z_j^{(l+1)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l+1)}}{\partial z_j^{(l+1)}} \frac{\partial C}{\partial a_j^{(l+1)}} \tag{1.10}$$

Dove:

$$\begin{aligned} z_j^{(l)} &= w_1^{(l)} a_1^{(l-1)} + \cdots + w_i^{(l)} a_i^{(l-1)} + b_j^{(l)} \\ a_j^{(l)} &= f^{(l)}(z_j^{(l)}) \; con \; f^{(l)} : \mathbb{R} \to \mathbb{R} \; funzione \; d'attivazione \end{aligned} \tag{1.11}$$

Essendo che il gradiente ci indica direzione ed intensità di massima crescita della funzione C, il metodo prevede di spostarsi verso il minimo utilizzando $-\nabla C$.

[7] E' una regola di derivazione che permette di calcolare la derivata della funzione composta da due funzioni derivabili

Il funzionamento dell'algoritmo è di semplice comprensione, di seguito ne elenco i passaggi:

1. Viene scelta una posizione casuale di partenza $\vec{q_0}$ (il quale rappresenta un vettore contente tutte le variabili della rete) nella funzione di costo C e un tasso di apprendimento $l > 0$.

2. Si calcolano iterativamente i successivi punti $\vec{q_i} = \vec{q_{i-1}} - l\nabla C(\vec{q_{i-1}})$ fino a che $C(\vec{q_i})$ non converge in modo soddisfacente verso il minimo, oppure finché non si raggiunge un valore massimo di iterazioni eseguite.

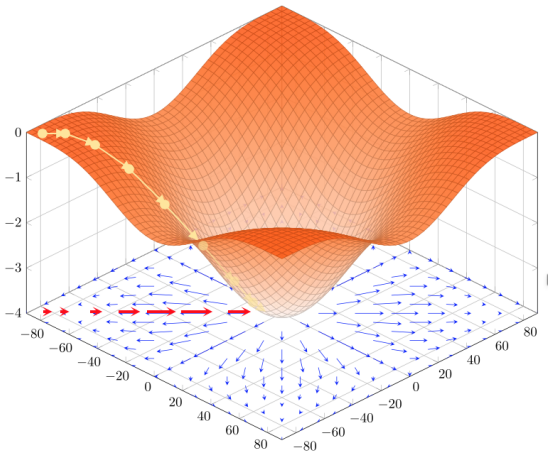

FIGURA 1.4: Esempio di discesa del gradiente

La scelta del tasso di apprendimento l è chiusa nell'intervallo $[0, 1]$ ed è fondamentale per il corretto funzionamento dell'algoritmo. Infatti, con l troppo grande il metodo divergerà, in quanto farà passi troppo grandi e si allontanerà dal minimo della funzione. Dalla controparte con un tasso d'apprendimento troppo piccolo si otterrebbe una convergenza troppo lenta. Per questo l risulta un parametro fondamentale per determinare la convergenza e la velocità della stessa.

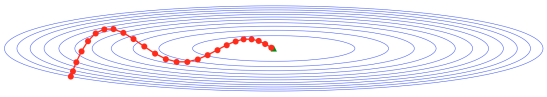

FIGURA 1.5: Esempio di convergenza

Le più note funzioni per la discesa del gradiente sono:

- Adaptive Moment Estimation (ADAM): è un metodo per l'ottimizzazione stocastica[8] che utilizza un sottoinsieme randomico del dataset in esame per creare un approssimazione stocastica. Semplice da implementare, computazionalmente efficiente, richiede l'utilizzo di poca memoria ed è indicato con dataset di grosse dimensioni.

- Stochastic Gradient Descent (SGD): è un metodo per l'ottimizzazione stocastica simile alla discesa del gradiente, in quanto ad ogni iterazione sostituisce il valore del gradiente dato dalla funzione di costo con una stima ottenuta valutando il gradiente su un sottoinsieme. Computazionalmente efficiente, richiede l'utilizzo di poca memoria ed è lo standard de facto per l'allenamento delle reti neurali.

1.2.4 Backpropagation

L'algoritmo di backpropagation è il fulcro dell'addestramento di una rete neurale, ha il compito di calcolare l'errore del sistema attraverso la funzione di costo e di utilizzare la discesa del gradiente per minimizzare questo errore retro propagandolo all'indietro per aggiornare i parametri della ANN. Questo processo viene svolto svariate volte fino ad ottenere un modello ottimale.

[8] Sono algoritmi di ottimizzazione che incorporano elementi probabilistici

1.2.5 Funzionamento

L'algoritmo ha il seguente funzionamento:

1. Assegna in modo casuale tutti i pesi e le soglie della rete.
2. Esegue le previsioni sul dataset di allenamento e calcola l'errore della rete.
3. Utilizza l'algoritmo di backpropagation per aggiornare i parametri della rete.
4. Ripete i passaggi da 2 a 3 finché l'errore della rete non è minimo o si raggiunge un limite massimo di iterazioni prestabilito.
5. Utilizza la rete appena allenata per eseguire una predizione sui nuovi dati.

1.2.6 Convolutional neural network

CNN acronimo di Convolutional Neural Network o convNet **[3]** sono tra le più famose reti feedforward nel deep learning per il riconoscimento d'immagini. Questa architettura è stata ispirata dalle ricerche biologiche di Hubel e Wiesel, i quali hanno scoperto che la corteccia visiva è costituita da cellule semplici e complesse. Le cellule semplici aiutano con il riconoscimento delle caratteristiche e le cellule complesse rilevano la posizione di un oggetto indipendentemente da dove è collocato nel campo visivo. Successivamente, le informazioni passano attraverso le varie cortecce celebrali le quali estraggono sempre più specifici pattern nell'immagine.

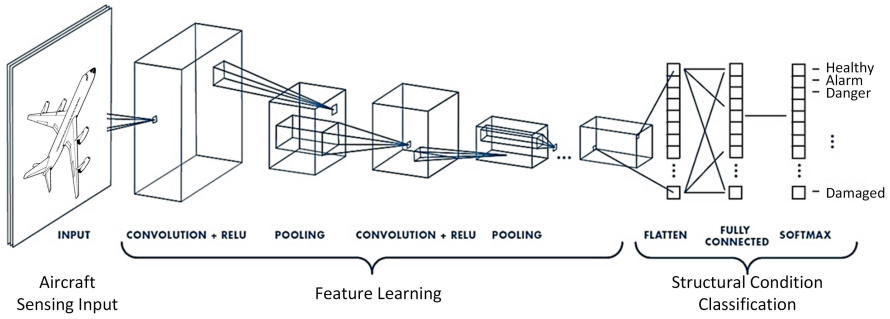

FIGURA 1.6: Convolutional neural network

La struttura di una CNN è molto simile, infatti può essere pensata come una combinazione di due componenti: da una parte un estrattore di feature e dall'altra un classificatore. La prima parte è composta da vari layer di convoluzione e pooling i quali hanno l'onere di estrarre le feature. Gli strati convoluzionari sono la parte centrale delle convNet, infatti autonomamente individuano le caratteristiche più significative. Più strati sono presenti e più feature complesse riusciranno ad essere riconosciute; ad esempio il primo layer riconosce i bordi, il secondo riconosce le figure e via dicendo. I livelli di pooling riducono drasticamente la dimensionalità spaziale del volume di input ed i requisiti computazionali per i livelli futuri. La seconda parte è composta da una rete feedforward fully connected la quale ha il compito di determinare la classe dell'immagine

avendo come input un vettore più gestibile per quanto riguarda la dimensionalità dello stesso. Il ruolo di una convNet è quello di ridurre le immagini in una forma più semplice per essere processata, senza perdere informazioni critiche per una buona previsione.

1.2.6.1 Livelli di convoluzione

Il layer di convoluzione è la componente principale della rete. Il suo obiettivo è quello di rilevare schemi, come curve, angoli, circonferenze o quadrati raffigurati in un'immagine. Ogni livello cattura una determinata caratteristica, e maggiore è il loro numero e più complesse saranno le feature che riescono ad individuare. L'estrazione avviene attraverso l'operazione di convoluzione, la quale prevede di far scorrere uno o più filtri (detti anche kernel) sulle diverse posizioni della matrice d'input (dove ogni posizione indica un pixel presente nell'immagine), e per ogni posizione viene generato un valore il quale è il risultato del prodotto scalare tra il kernel e la porzione dell'input coperta dallo stesso.

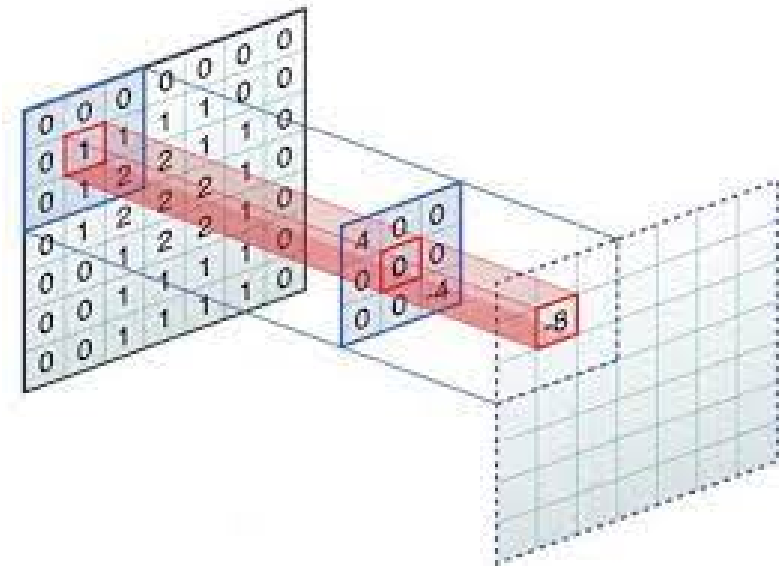

FIGURA 1.7: Esempio di convoluzione

La formula che identifica questa operazione con passo uno è la seguente:

$$G[i,j] = \sum_m \sum_n I[m+i, n+j] F[m,n] \tag{1.12}$$

Dove:

- I identifica la matrice che rappresenta l'immagine.
- F identifica la matrice filtro.
- G identifica la matrice delle caratteristiche.

Ci sono vari iperparametri da impostare in ogni livello di convoluzione:

- Dimensione del filtro F: ogni filtro cattura una determinata caratteristica in un'area ristretta dell'immagine. Di norma la sua dimensione è: 3×3, 5×5, 7×7.
- Numero di filtri K: questo valore va ad identificare la profondità della matrice di output. Infatti ogni livello andrà ad indicare l'operazione di convoluzione fatta tra l'immagine e un determinato kernel.
- Il passo S: determina di quanto si sposta il kernel dentro la matrice rappresentante l'immagine in oggetto per ogni operazione di convoluzione fatta.
- Il padding P: viene utilizzato per mantenere la stessa dimensionalità e per non perdere informazioni nei bordi dalla matrice di input in quanto i pixel nei bordi eseguiranno una sola convoluzione a differenza dei pixel al centro, i quali ne eseguiranno un numero nettamente maggiore. In questo modo l'informazione al centro verrà preservata meglio rispetto alle informazioni sui bordi. Per sopperire a questo problema è stato introdotto il padding il quale prevede di aggiungere una cornice di valori intorno all'immagine.

Questi iperparametri vanno ad identificare la dimensione della matrice in uscita dal livello di convoluzione, infatti prendendo in ingresso una matrice di dimensione $H_1 \times W_1 \times D_1$ abbiamo una feature map di dimensione:

$$\begin{aligned} H_2 &= \frac{H_1 - F + 2P}{S+1} \\ W_2 &= \frac{H_1 - F + 2P}{S+1} \\ D_2 &= K \end{aligned} \tag{1.13}$$

A differenza delle ANN le convNet riducono la quantità di connessione tra i vari nodi e la quantità di pesi da dover aggiornare con la backpropagation. Infatti nei layer di convoluzione abbiamo:

- Connettività locale: quando si parla di immagini si intende una enorme quantità di pixel e quindi non è pratico connettere i neuroni nel layer successivo a tutti i neuroni del livello precedente. Per cui ogni percettrone viene collegato solo ad una regione locale della matrice di input la quale dipende dalla dimensione del filtro utilizzato. Questo fa si che i neuroni processino nello stesso modo porzioni diverse dell'immagine. Si tratta di un comportamento desiderato, in quanto regioni diverse del campo visivo contengono lo stesso tipo di informazioni.

- Condivisione dei parametri: viene utilizzata per contenere il numero di parametri da aggiornare con la backpropagation in quanto se avessimo pesi diversi per ogni arco avremmo una quantità di parametri troppo esosa da poter calcolare in tempi ragionevoli. Questo è possibile grazie al fatto che se una caratteristica è desiderabile in una determinata posizione spaziale, allora dovrebbe essere utile anche calcolarla in una posizione diversa.

Questo permette alla rete di essere molto più snella e di poter prendere in ingresso anche immagini di grosse dimensioni. Alla fine di ogni livello di convoluzione è presente una funzione non lineare la quale ha lo stesso ruolo visto nel capitolo 1.2.1, cioè aggiungere una componenente non lineare in quanto l'operazione di convoluzione è un operazione lineare.

1.2.6.2 Matrice filtro

I filtri rappresentano i pesi da aggiornare nelle CNN. Infatti inizialmente nel algoritmo sono assegnati in maniera casuale, e vengono poi aggiornati ad ogni iterazione mediante l'algoritmo di backpropagation, visto nel capitolo 1.2.4. Così facendo la rete addestra i propri filtri ad estrarre le caratteristiche più importanti per riuscire a distinguere un'immagine da un'altra. Esistono diversi filtri ed ognuno di essi è impostato per riconoscere determinate caratteristiche.

FIGURA 1.8: Esempio applicazione filtro

1.2.6.3 Pooling

Dopo ogni livello di convoluzione usualmente viene inserito un layer di pooling il quale ha il compito di ridurre la dimensione spaziale di ogni feature map presente.

Questo permette di ridurre il numero di parametri e quindi di rendere meno onerosa la spesa computazionale e di contrastare l'overfitting[9]. Inoltre permette l'estrazione di caratteristiche dominanti, in quanto in presenza di una feature abbiamo un valore d'attivazione elevato, e questo permette l'estrazione senza rendere conto della sua posizione esatta, ma rispettando la posizione rispetto alle altre caratteristiche. Il funzionamento è molto simile a quello del livello di convoluzione, infatti viene scelto un passo e la dimensione del kernel da applicare all'immagine di input.
I tipi di pooling più utilizzati sono i seguenti:

- Max pooling : ritorna il massimo valore appartenente alla porzione occupata dal kernel. Ha il compito di eliminare le feature poco rilevanti e di ridurre la dimensionalità.
- Average pooling : ritorna il valore medio appartenente alla porzione occupata dal kernel ha il compito di ridurre la dimensionalità.

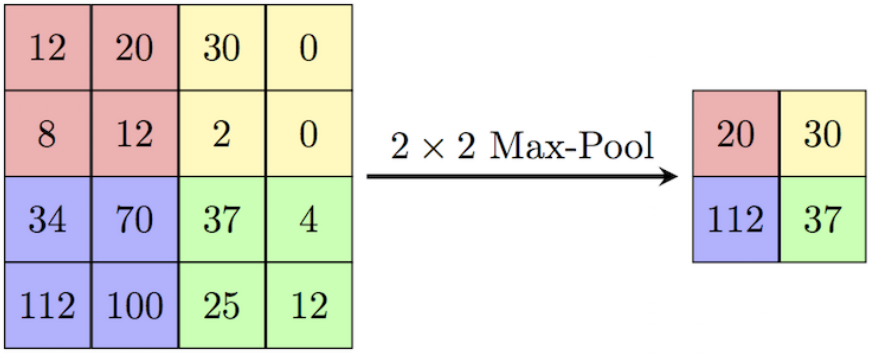

FIGURA 1.9: Esempio di max pooling

[9]E' quando un modello statistico molto complesso si adatta troppo bene ai dati osservati perché ha un numero eccessivo di parametri

1.2.6.4 Funzionamento

L'algoritmo ha il seguente funzionamento:

1. Assegna in modo casuale tutti i valori dei filtri nei vari livelli.
2. Esegue le previsioni sul dataset di allenamento e calcola l'errore della rete.
3. Utilizza l'algoritmo di backpropagation per aggiornare i parametri della rete.
4. Ripete i passaggi da 2 a 3 finché l'errore della rete non è minimo o si raggiunge un limite massimo di iterazioni prestabilito.
5. Utilizza la rete appena allenata per eseguire una predizione sui nuovi dati.

1.2.7 Vantaggi e svantaggi

I vantaggi principali di questa architettura sono:

- Memorizzazione delle informazioni su tutta la rete: le informazioni sono salvate in tutta la struttura, inoltre la scomparsa di alcune di esse non impedisce il corretto funzionamento.
- Predire con informazioni incomplete: durante la predizione la ANN può produrre risultati corretti anche con informazioni incomplete. La perdita di performance dipende dalla rilevanza dei dati mancanti.
- Attività in parallelo: le reti neurali sono in grado di eseguire più attività in parallelo.
- Approssimatore universale: le ANN sono degli approssimatori universali e questo genera un modello in grado di approssimare funzioni complesse.

E i relativi svantaggi sono:

- Dipendenza dal hardware: le ANN richiedono processori con potenza di calcolo in parallelo.
- Tempo di allenamento: le reti neurali richiedono molto tempo per allenarsi.
- Dimensione dataset: questa struttura ha bisogno di una mole di dati consistente sulla quale allenarsi per poi essere in grado di eseguire una predizione corretta.

1.3 Extreme Learning Machine

ELM acronimo di Extreme Learning Machine rappresenta un gruppo di reti neurali feedforward con un unico hidden layer (SLFN). A differenza delle classiche ANN non richiedono l'utilizzo della backpropagation per l'aggiornamento dei pesi e delle soglie nella rete, e questa differenza le rende drasticamente più veloci delle neural network durante l'apprendimento.

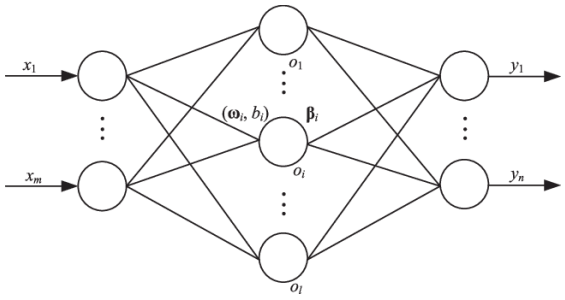

FIGURA 1.10: Extreme learning machine

Quello che le differenzia è dato dal fatto che, se la funzione d'attivazione $g : \mathbb{R} \to \mathbb{R}$ del hidden layer è infinitamente differenziabile e i pesi e le soglie del primo layer sono scelti in modo casuale, SLFN può essere considerato un sistema lineare. Quindi l'output è determinato attraverso una generalizzazione della matrice inversa tra hidden e l'output layer.
Per determinare gli output la rete utilizza la seguente formula:

$$\hat{y_j}(x) = \sum_{i=1}^{L} \vec{\beta_i} g_i(x_j) = \sum_{i=1}^{L} \vec{\beta_i} g(\vec{w_i} x_j + b_i),\ j = 1, \ldots, N \tag{1.14}$$

Dove:

- L è il numero di nodi nel hidden layer.
- N indica la cardinalità del campione in esame.
- $g : \mathbb{R} \to \mathbb{R}$ funzione d'attivazione utilizzata tra input layer e hidden layer.
- $\vec{w_i}$ indica il vettore dei pesi che connette i nodi di input con i-esimo nodo del hidden layer.
- b_i indica la soglia del i-esimo percettrone del hidden layer.
- $\vec{\beta_i}$ indica il vettore dei pesi che connette i-esimo nodo del hidden layer con i nodi dell'output layer.
- x_j indica il j-esimo vettore di input.

Grazie a Huang e Babri **[1]** i quali hanno dimostrato che una SLFN con L nodi nel hidden layer e con funzione d'attivazione non lineare g può esattamente imparare L distinte istanze di input. Da questo possiamo dedurre che la $\sum_{j=1}^{L} ||\hat{y}(x_j) - y(x_j)|| = 0$, allora esistono β_i, w_i, $bias_i$ tale che:

$$\sum_{i=1}^{L} \beta_i g(w_i x_j + bias_i) = y_j, \ con \ j = 1, \ldots, N \tag{1.15}$$

L'equazione sovrastante può essere riscritta come:

$$Y = H\beta \tag{1.16}$$

Dove:

$$H = \begin{bmatrix} g(w_1 x_1 + b_1) & \cdots & g(w_L x_1 + b_L) \\ \vdots & \ddots & \vdots \\ g(w_1 x_N + b_1) & \cdots & g(w_L x_N + b_L) \end{bmatrix}_{N \times L} \tag{1.17}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \qquad Y = \begin{bmatrix} y_1^T \\ \vdots \\ y_N^T \end{bmatrix}_{N \times m} \tag{1.18}$$

- m indica il numero di neuroni nel layer di output.
- H indica la matrice dove ogni riga determina i valori dei percettroni nel hidden layer per ogni istanza di input.
- Y indica le label del campione in esame.

Teorema 1. *Data una SLFN con N nodi nel layer nascosto e una funzione d'attivazione $g : \mathbb{R} \to \mathbb{R}$ la quale è infinitamente differenziabile in ogni punto, per N campioni arbitrariamente distinti (x_i, t_i), dove $x_i \in \mathbb{R}^n$ e $t_i \in \mathbb{R}^m$, per ogni w_i e b_i scelti in modo casuale in un qualsiasi intervallo rispettivamente di $\mathbb{R}^n$ e $\mathbb{R}$, in accordo con una qualsiasi distribuzione continua di probabilità, allora con probabilità uno, l'output della matrice del layer nascosto H sarà invertibile e $||H\beta - Y|| = 0$.* **[2]**

Teorema 2. *Dato un qualsiasi piccolo valore positivo $\epsilon > 0$ e una funzione d'attivazione $g : \mathbb{R} \to \mathbb{R}$ la quale è infinitamente differenziabile in ogni punto, allora esiste $L \leq N$ tale che per N campioni arbitrariamente distinti (x_i, y_i), dove $x_i \in \mathbb{R}^n$ e $y_i \in \mathbb{R}^m$, per ogni w_i e b_i scelti in modo casuale in un qualsiasi intervallo rispettivamente di $\mathbb{R}^n$ e $\mathbb{R}$ in accordo con una qualsiasi distribuzione di probabilità continua , allora con probabilità uno, $||H_{N \times L}\beta_{L \times m} - Y_{N \times m}|| < \epsilon$.* **[2]**

Teorema 3. *Esista una matrice G tale che Gy è la minima soluzione dei minimi quadrati del sistema $Ax = y$ allora è necessario e sufficiente che $G = A^\dagger$, che è la pseudo matrice inversa di A.* **[2]**

Grazie ai teoremi appena enunciati possiamo esporre un semplice modo per allenare le SLFN. Di norma nell'apprendimento dobbiamo identificare i corretti $\hat{w_i}$, $\hat{b_i}$, $\hat{\beta_i}$ con $i = 1, \ldots, L$ tale che:

$$||H(\hat{w_i}, \ldots, \hat{w_L}, \hat{b_i}, \ldots, \hat{b_L})\hat{\beta} - Y|| = \min_{w_i, b_i, \beta} ||H(w_i, \ldots, w_L, b_i, \ldots, b_L)\beta - Y|| \tag{1.19}$$

il quale è equivalente al minimizzare la funzione di costo. Per determinare questo minimo è sufficiente trovare la soluzione dei minimi quadrati di $\hat{\beta}$ del sistema lineare $H\beta = Y$. In accordo con il Teorema 1 se il numero di nodi nel hidden layer L è uguale al numero di campioni distinti N, la matrice H è quadrata e invertibile, quindi se i pesi w_i e le soglie b_i sono scelti in modo randomico l'errore dato dalla funzione di costo sarà zero. Invece, nella maggior parte dei casi $N \gg L$ e non esiste $H\beta = Y$. In accordo con il Teorema 3 la più piccola soluzione ai minimi quadrati del sistema lineare é $\hat{\beta} = H^{\dagger}Y$ dove $H^{\dagger}$ è la matrice inversa di Moore-Penrose.

1.3.1 Matrice pseudo inversa di Moore-Penrose

Sia $A \in \mathbb{K}^{m \times n}$ una matrice rettangolare di m righe e n colonne, con eventualmente $m = n$. Si chiama pseudo inversa di A la matrice $A^{\dagger} \in \mathbb{K}^{n \times m}$ che soddisfa le seguenti proprietà:

- $AA^{+}A = A$
- $A^{+}AA^{+} = A^{+}$
- $(AA^{+})^{T} = AA^{+}$ e $(A^{+}A)^{T} = A^{+}A$

La matrice pseudo inversa risulta indispensabile nelle ELM in quanto risolve sistemi di equazioni lineari del tipo $Ax = y$

1.3.2 Funzionamento

L'algoritmo in sé è molto semplice, infatti si compone di appena quattro passaggi:

1. Assegna valori randomici ai pesi w_i e alle soglie b_i, per $i = 1, \ldots, L$
2. Calcola la matrice del hidden layer H
3. Calcola la matrice dei pesi tra il layer nascosto e il layer d'output $\hat{\beta} = H^{\dagger}T$
4. Utilizza $\hat{\beta}$ per eseguire una predizione sui nuovi dati

1.3.3 Vantaggi e svantaggi

I vantaggi principali di questa architettura sono:

- Indipendenza dal hardware: le ELM possono non sfruttare calcoli su GPU.
- Tempo d'allenamento: il tempo di train di questa struttura è molto contenuto in quanto non necessita dell'algoritmo di backpropagation.
- Risorse: la Extreme Learning Machine non richiede una mole di dati eccessiva per eseguire il train della rete.
- Conversione: la struttura è in grado di convertire dati in formato immagine in dati di tipo vettoriale e quindi di convertire problemi difficili in problemi gestibili.

E i relativi svantaggi sono:

- Riconoscimento di pattern complessi: Extreme Learning Machine non è in grado di risolvere task la quale complessità intrinseca è elevata.
- Simultaneità dei data: questa struttura deve analizzare i dati in maniera sincrona a differenza delle classiche ANN le quali possono dividere il dataset in batch.

Capitolo 2

Analisi Sperimentale

Dopo aver descritto dettagliatamente tutti gli aspetti teorici, vado ad esporre di seguito l'approccio utilizzato per risolvere il problema del catastrophic forgetting introdotto nel capitolo 1, il quale indica la tendenza di una rete neurale a dimenticare completamente le informazioni apprese in precedenza. Come anticipato andremo ad utilizzare il terzo metodo esposto per la risoluzione del problema, inoltre metteremo a confronto l'attuale standard de facto con Extreme Learning Machine per verificare se quest'ultima risulti più efficace per la previsione nei vari dataset. I set di dati che ho visionato per questa tesi sono: MNIST, CIFAR10, CIFAR100, CORE50.
L'attuale standard de facto per quanto riguarda l'analisi e il riconoscimento d'immagini è la Convolutional Neural Network. La sua configurazione è la seguente:

- Estrattore di feature : resnet18 **[4]** pre allenata sul dataset ImageNet.
- Rete feedforward: composta da due hidden layer aventi ciascuno 400 nodi. Utilizza ReLU come funzione d'attivazione e come layer di output utilizza softmax[1].
- Ottimizzatore: viene utilizzato l'algoritmo ADAM.
- Funzione di costo: viene utilizzata la Cross Entropy Loss.

[1]Layer di output utilizzato per problemi di classificazione multi-classe

Dalla controparte ELM è composta da 400 nodi nel livello nascosto, utilizzando ReLU come funzione d'attivazione e come layer di output utilizza softmax.

2.1 Metodo per il riconoscimento della ANN

Dato che utilizzo l'approccio Architectural, ho individuato un metodo funzionale per il riconoscimento della ANN corretta da utilizzare per la previsione; infatti date M classi si avrà complessivamente N reti neurali con $N < M$, dove ognuna di esse dovrà riconoscere un numero definito di classi d'immagine. Questo metodo è stato introdotto nel paper **[5]** e l'idea che sta alla base è quella di creare una media della funzione di probabilità (in quanto la media E è uno stimatore consistente[2]) detto anche prototype per ogni neural network utilizzando i dati che sono stati sfruttati per allenarle. Questo prototipo è un vettore di lunghezza M dove per ogni posizione i viene indicata la probabilità media che la rete dia come risposta l'i-esima classe. Essendo che ogni ANN è istruita per riconoscere solo un sottoinsieme delle M classi, risulterà che il prototype d'allenamento avrà una probabilità maggiore nelle classi per le quali la rete è stata allenata.

[2]Uno stimatore $\hat{Q}$ è consistente se al crescere della dimensione campionaria n il suo errore campionario converge a zero

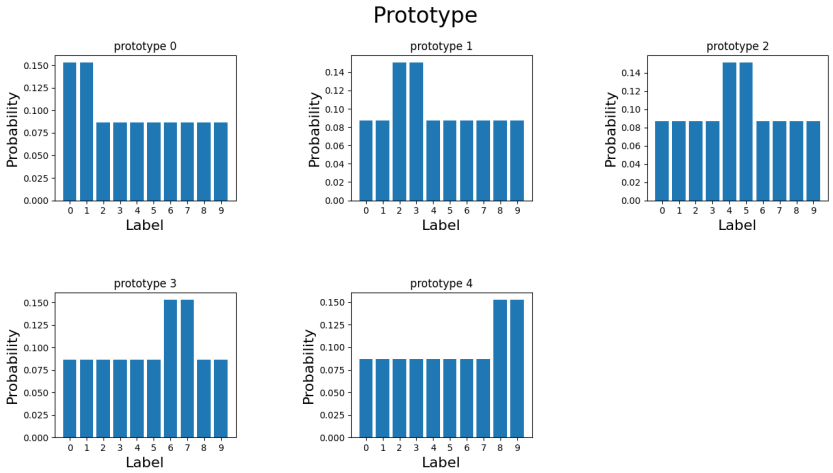

FIGURA 2.1: Esempio di prototype prodotto dal dataset CIFAR10 incr. 2

Guardando la figura 2.1 si può notare la probabilità media delle risposte date da ogni singola rete utilizzando i dati d'allenamento.

Per comprendere a pieno il processo svolto dall'algoritmo per identificare correttamente la rete, ho deciso di portare un esempio. Supponiamo di avere $M = 10$ e ogni ANN abbia il compito di distinguere tra due classi, per un totale di $N = 5$. Inoltre, immaginiamo che una rete abbia come obiettivo il riconoscere i gatti dai cani; allora esisterà una Neural Network allenata per riconoscere questo genere di animali. Sicuramente, data la funzione di probabilità generata da questa ANN la probabilità in prossimità delle labels che indica cane e gatto sarà nettamente maggiore rispetto alle altre label in quanto la struttura è allenata per individuare solo questo genere di etichette. Al momento della previsione verrà creato un altro prototipo per ogni neural network presente nel sistema utilizzando i dati da predire. Dopodiché, si verificherà la similarità tra questi prototipi e quelli creati al momento dell'allenamento il quale genererà un vettore di dimensione N dove ogni cella identificherà la distanza tra il prototipo d'allenamento e il prototipo di test della rete i con $i \in N$.

La somiglianza tra i prototype è calcolata utilizzando la distanza p-norma con $p = 8$:

$$D(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{2.1}$$

IN questo modo la AI adatta per eseguire il task è quella che darà come risultato la minima differenza tra il prototipo d'allenamento e quello di predizione. Infatti, essendo che ogni rete avrà la probabilità maggiore nelle istanze che prevede correttamente risulterà che il prototype più vicino sarà quello corretto per intraprendere una classificazione per il test.

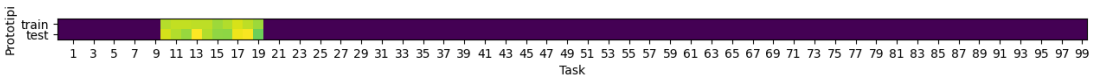

FIGURA 2.2: Esempio di differenza tra prototype di train e test ottenuto dal dataset CIFAR100 incr. 10

Questa è un'ottima tecnica per determinare la corretta ANN da utilizzare, in quanto è efficiente sia dal punto di vista computazionale che spaziale.

2.2 Risultati ottenuti con entrambi gli approcci

In questa sezione, vado a visualizzare ed analizzare i risultati ottenuti nella previsione dei dataset elencati all'inizio di questo capitolo con l'utilizzo della CNN e della ELM.
Per redarre questa tabella ho utilizzato il seguente hardware:

- Processore: Intel Xeon (2x 2.3 GHz).
- Ram: 12 GB DDR4.
- Scheda grafica: Nvidia Tesla p100-pcie (16 GB).

Problema	Algoritmo	Tempo train (s)	Tempo test (s)	Accuratezza	Errore std.
CIFAR10	CNN	697.09	2.29	0.823	0.079
incr. 2	ELM	12.59	2.37	0.653	0.032
CIFAR10	CNN	704.74	2.21	0.642	0.004
incr. 5	ELM	12.37	2.15	0.641	0.009
CIFAR100	CNN	717.13	3.55	0.544	0.014
incr. 10	ELM	14.78	2.76	0.533	0.009
CIFAR100	CNN	717.09	3.15	0.416	0.008
incr. 20	ELM	12.82	2.39	0.407	0.007
CORE50	CNN	16143.81	87.48	0.844	0.144
incr. 10	ELM	188.39	83.43	0.099	0.017
CORE50	CNN	12711.07	72.88	0.019	0.003
incr. 25	ELM	168.13	47.65	0.162	0.001
MNIST	CNN	1531.27	3.75	0.914	0.044
incr. 2	ELM	12.08	1.79	0.992	0.007
MNIST	CNN	1609.06	3.98	0.854	0.001
incr. 5	ELM	11.99	1.67	0.972	0.002

In questo summary vi sono due aspetti che sono subito evidenti. Il primo è l'estrema diversità nei tempi di train tra le due architetture. Infatti, lo standard de facto utilizza la backpropagation per aggiornare i pesi e le soglie sulla rete, e questo la rende drasticamente più lenta rispetto alla ELM. Il secondo è l'accuratezza della ELM nel dataset CORE50 la quale è drasticamente bassa in quanto questo genere di architettura non riesce a predire correttamente le istanze di un dataset cosi articolato.

2.3 Considerazioni riguardo CNN

Analizzando la Convolutional Neural Network utilizzata in questo studio, ho notato che è molto più complessa da allenare, non solo per quanto riguarda il tempo impiegato, ma anche per la quantità di iperparametri da settare per far si che la rete produca un risultato soddisfacente. Infatti, è stato necessario identificare un corretto learning rate il quale dipende dalla funzione di costo e determinare il numero di epoche esatte, per fare in modo che la ANN apprenda correttamente i dati di train senza incombere nell'overfitting. In più la bontà dell'adattamento influenza in maniera netta il prototype, in quanto una Artificial Neural Network, correttamente allenata, produrrà un prototipo più affidabile sia dal punto di vista del train che dal punto di vista del test. Dai grafici successivi si può notare come cambi l'adattamento della rete al variare delle epoche e come vada ad influenzare il prototipo.

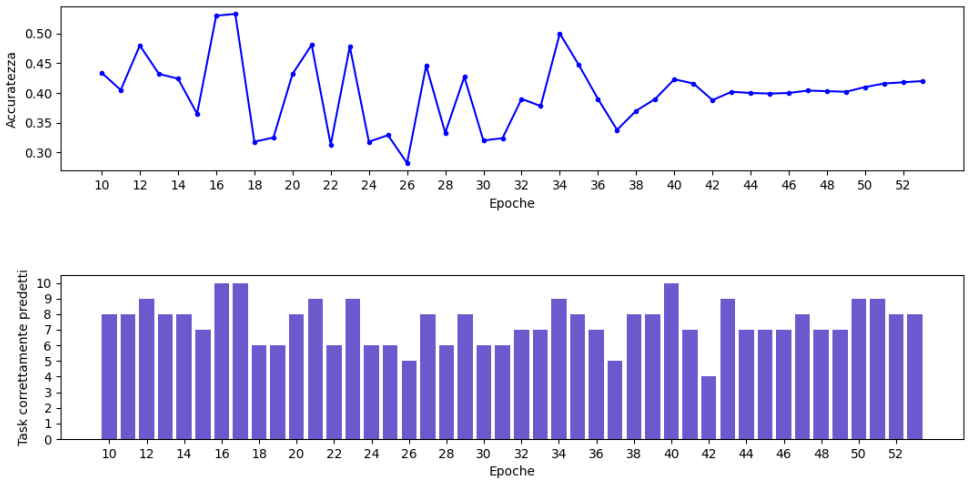

FIGURA 2.3: Esempio di adattamento del modello al variare delle epoche ottenuto dal dataset CIFAR100 incr. 10

Giustappunto da questa rappresentazione noto la correlazione presente tra l'adattamento del modello e il numero di task correttamente predetti; la quale risulta essere (prendendo in esame il campione appena esposto) pari a 0.836.

Dopodiché, voglio far notare come muta il tasso d'apprendimento della rete al variare del iperparametro learning rate tenendo il numero delle epoche costanti.

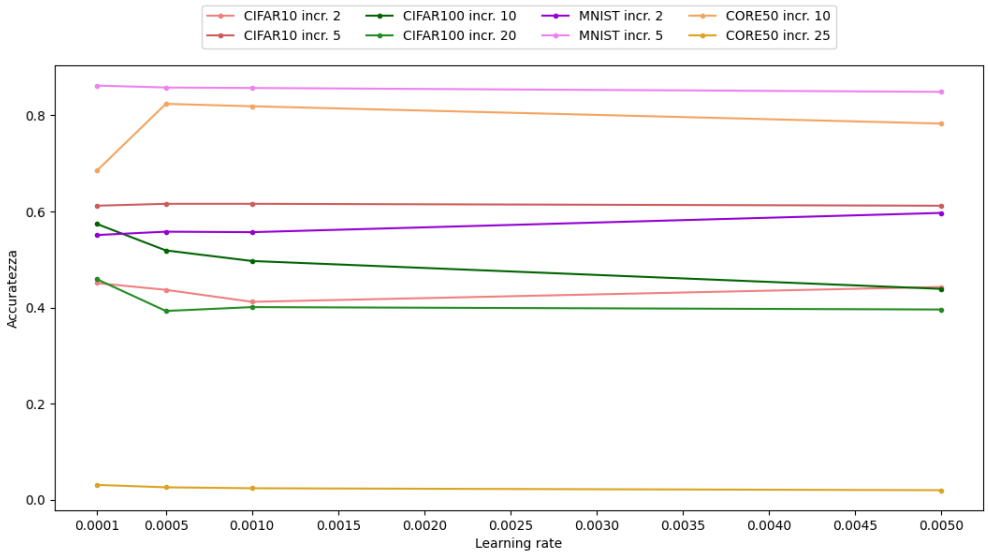

FIGURA 2.4: Esempio di adattamento dei modelli al variare del learning rate

Da questo elaborato posso notare che nella maggior parte dei dataset trattati al variare del parametro muti l'accuratezza del modello.

2.4 Considerazioni riguardo ELM

L'Extreme Learning Machine risulta efficiente per dataset dalla complessità non elevata come in questo caso CIFAR10, CIFAR100 e MNIST. Infatti in quest'ultimo dataset l'accuratezza della struttura raggiunge quasi il 100%. Il dataset CORE50 è un task contenente 50 oggetti domestici dove ogni classe ha il medesimo oggetto in differenti pose, differenti ambienti e differente luminosità e questo lo rende un dataset particolarmente difficile da predire. Per quanto riguarda gli altri tasks la differenza tra CNN e ELM non è cosi marcata, facendomi concludere che questa ANN è un'ottima alternativa per sistemi leggeri, i quali dispongono di limitate risorse di calcolo e devono risolvere problemi non eccessivamente complessi. Di seguito riporto una rappresentazione grafica che ci riporta alle affermazioni sopra fatte.

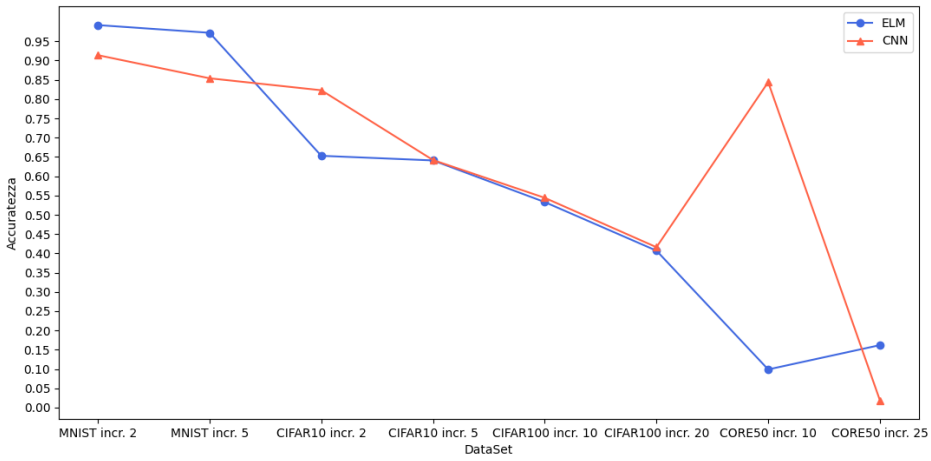

FIGURA 2.5: Differenza accuratezza tra ELM e CNN nei vari dataset trattati

Capitolo 3

Conclusioni

Concludo questo elaborato con delle riflessioni personali.
Durante questo periodo ho avuto l'opportunità di mettermi in gioco in un ambiente a me completamente sconosciuto, questo ha contribuito ad accrescere le mie conoscenze in materia di intelligenza artificiale. Inoltre ho potuto apprendere ed anche apprezzare, i principali costrutti matematici che stanno alla base di queste strutture, realizzando come un semplice cambiamento sia in grado di stravolgere il corretto funzionamento di una rete. Ma l'aspetto che più mi ha colpito durante questi mesi è la capacità di un modello artificiale di riconoscere immagini in contesti di tutti i giorni, emulando il funzionamento del cervello umano, e le molteplici applicazioni nelle quali questa tecnologia può essere utilizzata.

Bibliografia

[1] G.-B. Huang, H.A. Babri, Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions, IEEE Trans. Neural Networks 9 (1) (1998) 224–229.

[2] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: Theory and applications. (2006) 1-14.

[3] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks. (2012) 1-9.

[4] K. H. Xiangyu, Z. Shaoqing, R. J. Sun, Deep Residual Learning for Image Recognition. (2015) 1-12.

[5] D. Hendrycks, K. Gimpel, A baselin for detecting misclassified and out-of distribution examples in neural network. (2018)

[6] Michael Nielsen, "Using neural nets to recognize handwritten digits", neuralnetworksanddeeplearning.com. (2019)

[7] Chi-Feng Wang, "Finding the Cost Function of Neural Networks", towardsdatascience.com. (2018)

[8] Akshay L Chandra, "McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron", towardsdatascience.com. (2018)

[9] Eugenio Culurciello, "Continual learning", medium.com. (2018)

[10] Sebai Dorsaf, "Comprehensive synthesis of the main activation functions pros and cons", medium.com. (2020)

[11] Avinash Sharma V, "Understanding Activation Functions in Neural Networks", medium.com. (2017)

[12] Hamza Mahmood, "Activation Functions in Neural Networks", towardsdatascience.com. (2018)

[13] Ayyüce Kızrak, "Comparison of Activation Functions for Deep Neural Networks", towardsdatascience.com. (2019)

[14] Sagar Sharma, "Activation Functions in Neural Networks", towardsdatascience.com. (2017)

[15] Hamza Mahmood, "Gradient Descent", towardsdatascience.com. (2019)

[16] Simeon Kostadinov, "Understanding Backpropagation Algorithm", towardsdatascience.com. (2019)

[17] Lorenzo Govoni, "Algoritmo Discesa del Gradiente", lorrenzogovoni.com. (2021)

[18] Kemal Erdem, "Introduction to Extreme Learning Machines", towardsdatascience.com. (2020)

[19] Sumit Saha, "A Comprehensive Guide to Convolutional Neural Networks", towardsdatascience.com. (2018)

[20] "Different Kinds of Convolutional Filters", saama.com. (2017)

[21] Jefkine Kafunah, "Backpropagation In Convolutional Neural Networks", jefkine.com. (2016)

[22] Matthew Stewart, "Simple Introduction to Convolutional Neural Networks", towardsdatascience.com. (2019)

[23] Christopher Thomas, "An introduction to Convolutional Neural Networks", towardsdatascience.com. (2019)

[24] Piotr Skalski, "Gentle Dive into Math Behind Convolutional Neural Networks", towardsdatascience.com. (2019)

[25] Christian Versloot, "What is padding in a neural network?", machinecurve.com. (2020)

[26] Jason Brownlee, "How Do Convolutional Layers Work in Deep Learning Neural Networks?", machinelearningmastery.com. (2019)

[27] "Convolutional Neural Networks", cs231n.github.io

[28] Olah Chris, Mordvintsev Alexander, Schubert Ludwig, "Feature Visualization", distill.pub. (2017)

[29] 3Blue1Brown, "Backpropagation calculus", youtube.com. (2018)

[30] 3Blue1Brown, "What is Neural Network?", youtube.com. (2020)

[31] Stanford University School of Engineering, "Introduction to Convolutional Neural Networks for Visual Recognition", youtube.com

[32] Stanford University School of Engineering, "Image Classification", youtube.com

[33] Stanford University School of Engineering, "Loss Functions and Optimization", youtube.com

[34] Stanford University School of Engineering, "Introduction to Neural Networks", youtube.com

[35] Stanford University School of Engineering, "Convolutional Neural Networks", youtube.com

[36] 3Blue1Brown, "Gradient descent, how neural networks learn", youtube.com. (2018)

[37] Brandon Rohrer, "How Convolutional Neural Networks work", youtube.com. (2016)

[38] Vincenzo Lomonaco, "Why Continual Learning is the key towards Machine Intelligence", medium.com. (2017)