Alma Mater Studiorum Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA
DIPARTIMENTO DI INGEGNERIA DELL'ENERGIA ELETTRICA
E DELL'INFORMAZIONE
CORSO DI LAUREA IN INGEGNERIA ELETTRONICA E TELECOMUNICAZIONI

Tesi di Laurea
in
Calcolatori Elettronici T

Approcci di Machine Learning Per il Riconoscimento di Immagini

CANDIDATO:
AYMEN SAYED

RELATORE:
Prof. Luigi Di Stefano

Anno Accademico 2015/2016

Abstract

Oggigiorno, la quantità di dati che viene collezionata e archiviata nei server e data center di internet, è enorme, e sta crescendo esponenzialmente basti pensare a quante foto e video vengono caricati in rete ogni giorno, infatti in un solo minuto vengono caricati su Youtube circa 400 ore di video. Nasce da qui il bisogno di sistemi e algoritmi che sono in grado di elaborare questi dati per estrarre informazioni, si pensi a quante informazione possono essere ottenute analizzando una singola immagine presa da un video. Se per dati ad esempio intendiamo immagini, il primo passo in questo lavoro è quello di realizzare sistemi in grado di riconoscere e classificare con ottima accuratezza un dataset con un numero elevato di classi. I sistemi di Machine Learning imparano automaticamente a classificare dai dati, che ha portato negli ultimi anni la loro rapida diffusione, oggi questi sistemi gli troviamo nella ricerca web, filtri di spam, trading di azioni, riconoscimento frodi, preparazione farmaci e molte altre. Secondo un recente rapporto della McKinsey Global Institute, sostiene che il machine Learning sarà la tecnologia che porterà alla prossima grande onda di innovazione[4]. In questa tesi si tenterà di spiegare i principi di funzionamento dei principali approcci di machine Learning per la classificazione delle immagini, e di valutare ciascuno di questi approcci su un dataset di 60000 esempi, verrà poi effettuato un confronto tra le varie tecniche al fine di comprendere altri aspetti importanti nel compito di classificazione.

Indice

Introduzione

Il neurone artificiale è un modello computazionale ispirato al neurone biologico che è invece più complesso, questo modello fu proposto la prima volta nel 1943 da McCulloch e Pitts, il primo un neurofisiologo e il secondo è stato un matematico, però il modello non aveva ancora un utilizzo poiché mancava un aspetto fondamentale, cioè la possibilità di un apprendimento, che nel tempo si è cercato di risolvere con ipotesi collegate al funzionamento del sistema nervoso, cosi nel 1986 fu pubblicato un metodo di addestramento di reti di neuroni artificiali detto algoritmo di Backpropagation, che inizialmente ha succitato molto interesse tra gli studiosi nel campo del apprendimento automatico.

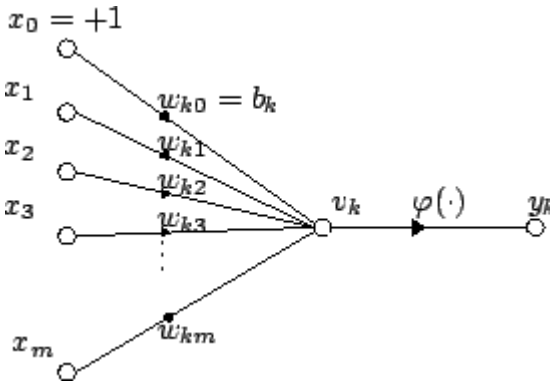


Figura 1: Modello di un neurone artificiale con funzione di attivazione della combinazione lineare degli ingressi pesati.

L'unita elementare computazionale del nostro cervello è il neurone, in cui ci sono all'incirca 86 miliardi di queste unita connessi tra di loro con circa 10^{15}sinapsi. Se volessimo confrontare il neurone biologico con il modello computazionale tipo quello in figura 1, potremo dire che i segnali d'ingresso (x_i) che viaggiano attraverso l'assone di un secondo neurone, agiscono attraverso una moltiplicazione con i pesi (w_i) dei dendriti del primo neurone, questi pesi rappresentano la forza di collegamento nella sinapsi . Se la somma finale di questi prodotti supera una certa soglia, il neurone scarica trasmettendo un impulso, la frequenza di scarica viene descritta nel modello con una funzione di attivazione φ.

Le Reti Neurali, cioè reti di neuroni artificiali, sono un modello computazionale diverso da quello dell'architettura di Von Neumann, con la passibilità di risolvere differenti tipi problemi altrimenti irrisolvibili o risolvibili in un tempo molto lungo. Nei computer tradizionali i passi computazionali sono sequenziali deterministici e logici in cui abbiamo un unico processore che lavora in modo seriale, nelle reti neurali invece non abbiamo una sola unità centrale ma tanti semplici unità di elaborazione, organizzati e connessi per svolgere molto bene attività che richiedono il calcolo parallelo.

Le reti neurali insieme ad un altro tipo di reti di neuroni artificiali detti Reti Neurali Convoluzionali, abbreviati con CNN, fanno parte di quella branca della computer science detta Machine Learning che si occupa di algoritmi per l'apprendimento automatico, cioè algoritmi che presentano un processo di addestramento e ottimizzazione per migliorare le proprie performance nello svolgere un determinato compito. Possiamo dire che ci sono diversi tipi di machine Learning, ma per questo elaborato mi concentrerò su quello più largamente utilizzato, la classificazione, per classificare i dati bisogna essere in grado di riconoscerli per poi poterli assegnare alle varie classi. Il compito di riconoscere le immagini al giorno d'oggi viene spesso assegnato agli algoritmi di machine Learning, soprattutto con le CNN per la loro proprietà di poter estrarre automaticamente le caratteristiche necessarie ad ottenere l'accuratezza più alta, infatti negli ultimi anni ha portato un grande miglioramento di performance nel suo impiego per attività di riconoscimento e localizzazione degli oggetti nelle immagini o anche dai video , così da vincere la maggior parte delle recenti competizioni per la classificazione delle immagini.

Capitolo 1

Studio Teorico dei concetti di classificazione con il Machine Learning

È molto difficile scrivere un programma per risolvere problemi come il riconoscimento di oggetti tridimensionali da differenti punti di vista e con variazioni della luminosità, anche se noi riusciamo a farlo, non sappiamo che programma scrivere perché non conosciamo come il nostro cervello è in grado di farlo e comunque scrivere questo programma può rivelarsi molto complicato.

L'approccio del machine Learning ci dice, invece di scrivere un programma per ogni specifico task di riconoscimento, collezioniamo molti esempi che specificano il corretto output per un dato input e addestriamo attraverso gli algoritmi di machine Learning un unico modello in grado di riconoscere gli oggetti sotto le varie condizioni.

Gli algoritmi di machine Learning possono capire come eseguire importanti attività generalizzando dagli esempi e sulla base dell'osservazione dei dati imparare e quindi migliorare la propria accuratezza nello svolgere quella particolare attività. Principalmente ci sono tre modalità di apprendimento automatico:

- Supervised Learning.
- Reinforcement Learning.
- Unsupervised Learning.

Nel Supervised Learning o apprendimento supervisionato, attualmente la forma più utilizzata di apprendimento automatico, si ha una funzione f che utilizza dei parametri W per mappare il vettore d'ingresso x nell'uscita y da predire.

Con questa forma di apprendimento si sottopongono in input al sistema, durante la fase di addestramento, un insieme di esempi, detto training set, ognuno dei quali è etichettato con il rispettivo valore di output desiderato; addestrare il sistema di solito significa aggiustare i parametri per ridurre la discrepanza tra l'uscita voluta su ogni esempio dei dati di addestramento e la reale uscita y, prodotta dal modello.

1.1 Classificazione delle immagini

La classificazione di immagini consiste nell'assegnare ad un immagine in ingresso una etichetta appartenente ad un numero fissato di classi.

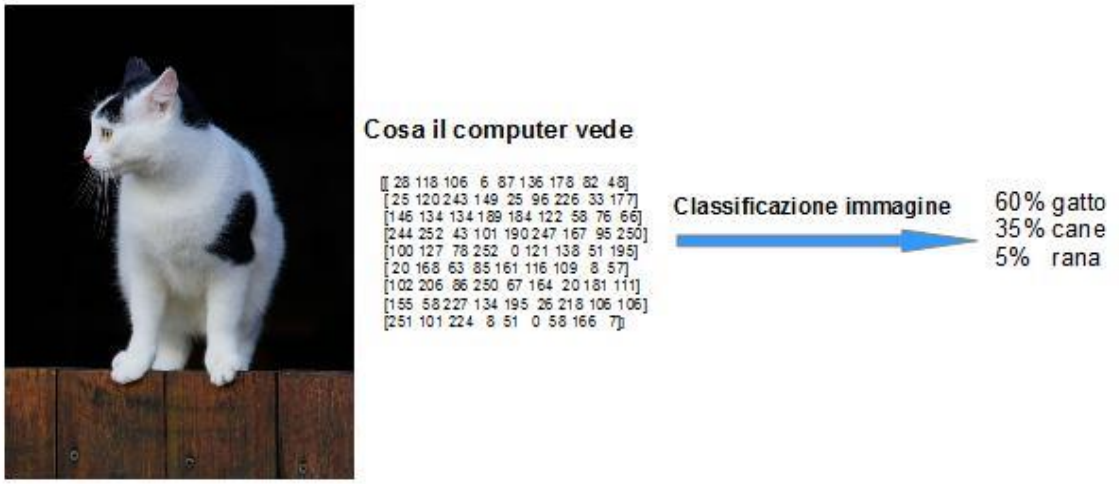


Figura 1.1: Esempio di classificazione del immagine di un gatto, assegnando ad essa le percentuali di appartenenza alle varie classi.

L'immagine dell'gatto come quella nella figura 1.1, per il computer è una matrice di numeri, ad esempio con 32 pixel in orizzontale e 32 pixel in verticale e in più da tener conto dei tre canali di colore RGB, la matrice assumerebbe la forma (32 x 32 x 3).

Anche se per noi sono banali, ci sono diverse situazioni che per il computer diventano un problema nel classificare le immagini, come il cambiamento del punto di vista, variazioni nella scala o deformazioni; ad esempio in riferimento alla figura 1.1, se nello stesso sfondo riduciamo le dimensioni del gatto, lo ruotiamo di lato e variamo leggermente la luminosità allora c'è la possibilità che il modello di classificazione non lo riconosca più nella classe gatto.

Gli algoritmi di machine Learning devono creare modelli con una buona accuratezza anche sotto queste condizioni. Nel processo iniziale per la loro creazione si hanno:

Input: consiste in un insieme di immagini ciascuna etichettata con una delle k differenti classi e che prende il nome di training set.

Addestramento: usando il training set addestriamo il modello su ogni classe.

Valutazione: valutiamo il classificatore facendogli predire le classi su un nuovo insieme di immagini che il modello non ha mai visto, ed in seguito compariamo le predizioni con le reali etichette delle immagini.

Per valutare le diverse tecniche di classificazione studiate in questa tesi è stato utilizzato come dataset il CIFAR-10 [1] che consiste in 60.000 immagini di 10 differenti classi, come si vede nella figura sotto, 50.000 di questi sono per il Training set e 10.000 per il Test set.

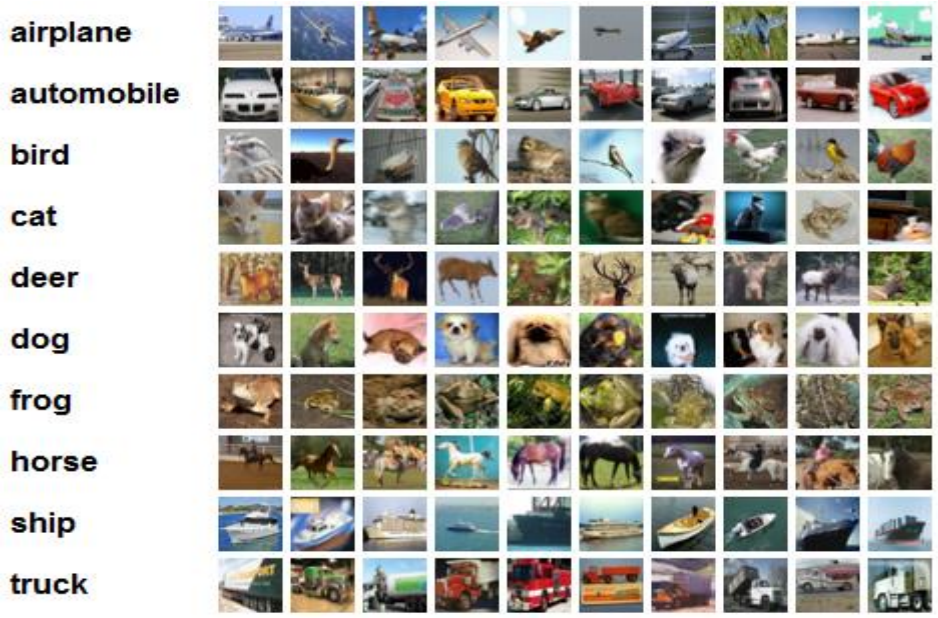


Figura 1.2: [3] Visualizzazione di un piccolo campione di 10 classi preso dal dataset CIFAR-10 [1].

1.2 Classificatore Lineare

Due componenti essenziali nella classificazione sono una **score function,** cioè una funzione che mappa i dati ai voti per le classi come vedremo in seguito, in più ci serve una **loss function** cioè una **funzione costo** che quantifica quanto siamo contenti del risultato delle predizioni da parte del modello, verrà anche aggiunta una procedura di ottimizzazione che tenterà di minimizzare la funzione costo.

Assumiamo un dataset per l'addestramento, di immagini $\boldsymbol{x_i} \in \boldsymbol{R^D}$, a ciascuna associata un etichetta $\boldsymbol{y_i}$ $con\ i = 1 \dots \dots . . N$ e $y_i \in 1 \dots \dots \dots K$. Riassumendo ciò, vuole dire che abbiamo N esempi ciascuno con dimensione D, su K classi distinte. Ad esempio in CIFAR10 [1] abbiamo 50.000 esempi per l'addestramento ciascuno con dimensione D= (32x32x3) = 3072 pixel e K=10 dato che ci sono dieci classi distinte (gatti, auto ecc.).

Adesso scriviamo la funzione "score function" per l'assegnazione dei voti alle classi riferite all'immagine in ingresso,

$$\boldsymbol{f}: \boldsymbol{R^D} \rightarrow \boldsymbol{R^K}$$

$$\boldsymbol{f}(\boldsymbol{x_i}, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\,\boldsymbol{x_i} + \boldsymbol{b} \qquad (1.1)$$

x_i è una matrice (D x 1) abbiamo allungato la matrice trasformandola in un vettore colonna, W è una matrice (K x D) e b (K x 1) che è un vettore **biases** (o di offset)**.**

Notare che ogni riga di $\boldsymbol{W}$ rappresenta un classificatore per una certa classe delle K che ci sono, quindi quando eseguiamo la moltiplicazione $\boldsymbol{W}\,\boldsymbol{x_i}$ stiamo valutando in parallelo K classificatori. A questo punto dato che gli ingressi (x_i , y_i) sono fissati, per poter migliorare l'accuratezza del modello possiamo agire solo su W e b che sono i parametri del modello, ciò che andremo a costruire, è un sistema in grado di imparare e migliorare dai propri errori durante la fase di training.

Proviamo a dare un interpretazione al classificatore lineare che nel capitolo 2 ci servirà per compararlo con un classificatore a rete neurale.
Alla fine dell'addestramento del modello sul training set del dataset CIFAR10 [1] ogni riga di $\boldsymbol{W}$, che è un classificatore, assume l'aspetto mostrato nella figura 1.3 dopo averla rimessa nella forma (32 x 32 x 3):

Figura 1.3: [4] Visualizzazione dei parametri del classificatore lineare alla fine della fase di addestramento sul dataset CIFAR-10 [1].

Facendo quindi il prodotto $\boldsymbol{W}\ \boldsymbol{x_i}$, quello che si sta facendo è confrontare i valori dei pixel dell'immagine esempio x_i con i parametri di ciascuna riga di W , producendo in uscita un vettore di voti. Si veda nella figura 1.3 come il classificatore per la classe cavallo presenta due teste una rivolta verso destra e l'altra verso sinistra ciò accade perché nel dataset ci sono cavalli rivolti sia in un senso che in quello opposto.

Per calcolare l'accuratezza sul training set, per ogni esempio di questo set di immagini, viene presa l'etichetta della classe di cui l'esempio ha ottenuto il voto più alto, si effettua poi un confronto tra le etichette corrette e quelle predette, ottenendo così il valore dell'accuratezza come il rapporto tra il numero delle predizione corrette e il numero degli esempi del training set, alcune volte questo valore può essere espresso in percentuale, facendo il prodotto con 100.

1.3 Funzione costo

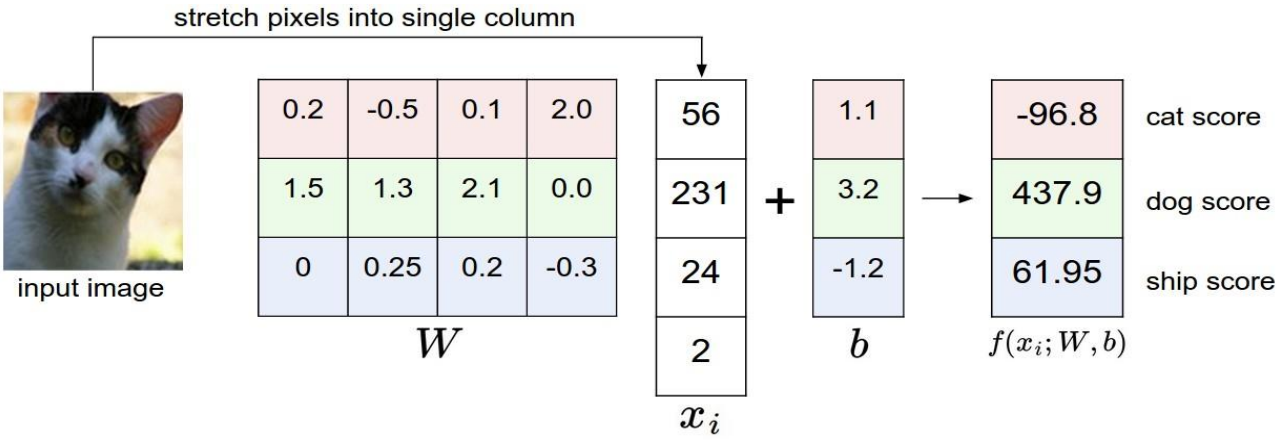


Figura 1.4: Esempio del operazione eseguita dalla score function sull'immagine in ingresso utilizzando una moltiplicazione per la matrice W ed una somma con il vettore di offset [4]

Nella figura 1.4 con ingresso l'immagine di un gatto e con tre classi (gatti, cani, navi), attraverso la score function discussa prima abbiamo ottenuto in uscita i voti per le varie categorie di immagini e che il voto per la classe gatti vale -96.8, confrontandolo con gli altri voti capiamo che il modello ha sbagliato a predire; però non siamo in grado di misurare di quanto il modello di classificazione abbia sbagliato, per poterlo fare possiamo usare una loss function cioè una funzione costo che ha come variabili i voti su tutte le classi .

Per quello che dobbiamo fare ci sono due funzioni costo molto usate, la **hinge loss** che è la funzione costo utilizzata nelle macchine a vettori di supporto SVM, una metodologia di apprendimento supervisionato, mi potrei riferire a questa funzione costo con il termine **SVM loss**; invece la **cross-entropy loss** è la funzione costo del **classificatore softmax**.

1.3.1 SVM loss

$$L_i = \sum_{j \neq y_i}^{k} \max(0, s_j - s_{y_i} + \Delta) \qquad (1.2)$$

L_i è il valore d'uscita della funzione detto anche **Data loss** ed è riferito al immagine di indice i del training set, $s_j = f(W, x_i)_j$ è il voto associato alla j-esima classe, s_{y_i} invece è il voto associato alla classe a cui appartiene l'immagine x_i , la funzione (1.2) ci sta dicendo che per avere un valore basso di L_i è necessario che il voto associato alla j-esima classe sia più basso di almeno un margine Δ rispetto al voto della classe corretta.

$$L_i = \sum_{j \neq y_i}^{k} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \qquad (1.3)$$

Questa forma dell'equazione 1.3 evidenzia come sono calcolati s_j , s_{y_i}.

1.3.2 Softmax loss

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad (1.4) \qquad f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \qquad (1.5)$$ Funzione softmax

L_i ha lo stesso significato di prima ma questa volta calcolata utilizzando una funzione costo diversa (1.4), $f_j = f(W, x_i)_j$ indicata con una notazione differente poiché secondo una interpretazione probabilistica i voti associati alle classi dalla score function vengono interpretati come logaritmi di probabilità non normalizzate ed attraverso la funzione softmax vengono normalizzate.

Con questa funzione costo (1.4) per avere il valore di L_i il più basso possibile, quindi avere un buon classificatore il voto associato alla classe corretta deve essere il più alto possibile rispetto ai voti delle altre classi, questo a differenza dell'SVM loss in cui bastava essere sopra un certo margine, come si può vedere nella figura 1.5:

Figura 1.5: Visualizzazione del margine sull'asse dei voti in riferimento alla hinge loss del SVM, i voti sono rappresentati sull'asse dai punti neri.

Ciò influisce anche nella procedura di ottimizzazione di cui ne parlerò nel paragrafo 1.5 dove l'SVM loss è meno rigido nell'ottimizzare, invece se si utilizza il softmax loss il classificatore verrà ottimizzato anche quando i voti tra la classe corretta e le altre si discostano con un valore superiore al margine Δ.

1.4 Regolarizzazione

Un altro aspetto importante per la classificazione con il machine Learning sono i concetti di overfitting (o adattamento eccessivo) e underfitting che è l'opposto del primo. L'underfitting può presentarsi per diverse ragione spesso legate alla capacità del modello, come ad esempio se alla fine della fase di addestramento notiamo che il modello pur essendo correttamente implementato non in grado di convergere sui dati di addestramento.

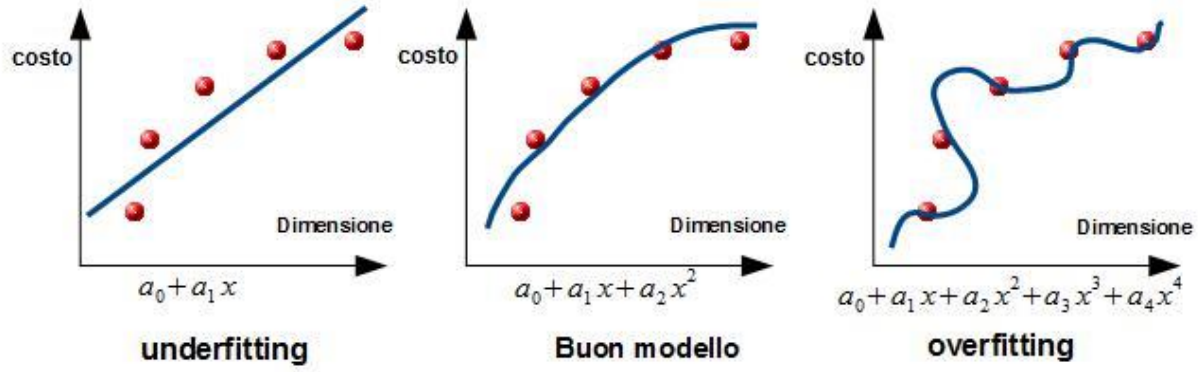


Figura 1.6: Concetto di overfitting di un modello applicato su dati d'esempio, sulle ascisse si ha la dimensione di un casa ed il suo costo sull'asse delle ordinate.

Se proviamo nella figura 1.6 a predire quanto è il prezzo di una casa in base alle dimensioni di questa, notando che c'è una relazione quadratica nei dati, se utilizziamo un modello lineare con un polinomio di primo grado verrà fuori che il modello ha una bassa capacità per il compito dato, con un polinomio di grado molto maggiore invece il modello con capacità eccessiva si adatta più del necessario sui dati di training in questo caso si dice che il modello ha una alta varianza, con la conseguenza che non è in grado generalizzare e quindi è più probabile a non avere una buona accuratezza nel test set, da qui deduciamo che un buon modello di classificazione si trova nel giusto equilibrio tra i due casi, per questo motivo inseriamo un altro componente nel sistema che agirà sulla capacità del modello in modo da avere un classificatore in grado sia ad avere delle buoni performance sul training set che di generalizzare su dati mai visti prima.

Un modo per ridurre il fenomeno dell'overfitting è quello di limitare le dimensioni dei valori dei parametri (o pesi) che si trovano nella matrice W della forma (k, D), attraverso la seguente funzione:

$$R(W) = \sum_k \sum_l W_{k,l}^2 \qquad (1.6)$$

Dato che il valore della funzione (1.6) detto **Regularization loss** (o costo della regolarizzazione) viene sommato al data loss, allora vogliamo che R(W) sia il più basso possibile, per esempio si preferisce W1 = [0.25, 0.25, 0.25, 0.25] invece che W2 = [1, 0, 0, 0], poiché i primi hanno un regularization loss più piccolo, in effetti se x = [1 ,1,1,1] allora $W_1^T x = W_2^T x = 1$ ciò vuol dire che hanno lo stesso data loss, ma R(W_1) > R(W_2); Un analogia che può aiutare a capire la preferenza di W_1, è che se volessimo predire a quale persona appartiene l'immagine d'esempio di un viso e abbiamo la possibilità di tenere conto di un certo numero di caratteristiche con l'intensità che vogliamo, in un caso decidiamo di tener conto solo del colore degli occhi con un valore dei parametri associati a questa caratteristica alti ma intanto trascurare le altre caratteristiche, non avremo in pratica un buon classificatore rispetto al caso in cui teniamo conto di molte altre caratteristiche (colore pelle , distanza tra gli occhi ecc.) e con valori dei parametri associati più piccolo .

Come accennato prima, il valore della funzione costo finale è dato dalla somma del Regularization loss e data loss come si può vedere nell'equazione (1.7).

$$L = \frac{1}{N}\sum_i L_i + \gamma\, R(W) \qquad (1.7)$$

γ detto regularization strength, indica di quanto vogliamo penalizzare W in modo che i parametri sono più piccoli e distribuiti, γ non è un parametro ma un **Iperparametro** del modello, poiché spesso non è ovvio sapere a priori quale valore assegnargli ma dobbiamo testare alcuni valori su un piccolo sottoinsieme di dati preso dal training set, chiamato **Validation set** e in seguito scegliere il valore che lavora meglio.

1.5 Ottimizzazione

Fin ad ora non sappiamo come settare i parametri con i giusti valori per poter riconoscere gli esempi delle vari classi, il processo che si occupa di questo si chiama ottimizzazione e ha il compito di trovare l'insieme di parametri W che minimizzano la funzione costo. Può essere d'aiuto poter visualizzare la funzione costo rispetto ai parametri della matrice W, dato che però questa matrice è multidimensionale con [10 x 3073] per un totale di 30,730 parametri, non riusciamo a rappresentarla nello spazio, proviamo allora ad usare W a due dimensioni.

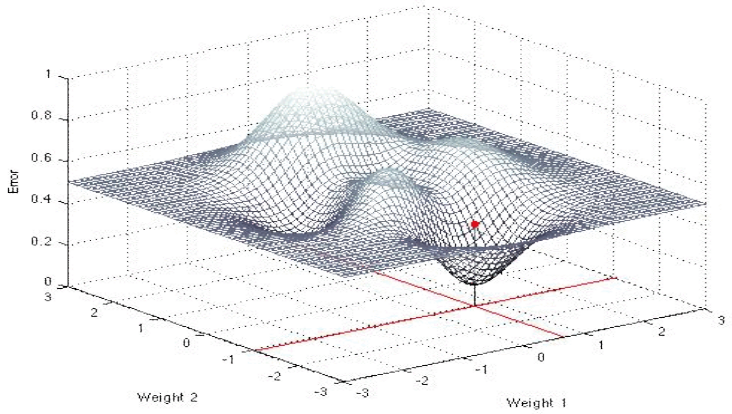


Figura 1.7: Grafico di un esempio di funzione costo rappresentata sull'asse z (verticale) rispetto ai due parametri w_1 , w_2.

Un primo approccio che ci introduce nel problema è quello di settare W= [w_1 , w_2] in modo casuale sperando di trovarci nel punto migliore cioè quello con l'errore più basso, ripetendo cosi fino ad ottenere un valore desiderato, ma si intuisce subito che non è per niente un buon modo nella pratica.
Osservando la figura 1.7, possiamo decidere di partire da un punto iniziale casuale e utilizzando un importante strumento di Calcolo che è la derivata, calcolare la pendenza in quel punto e aggiustare i parametri in modo da ridurre l'errore;

Nota: nel nostro caso avendo anche più dimensioni dobbiamo parlare di gradiente e vettori di derivate parziali.

Per poter calcolare le derivate parziali su ogni dimensione di W, possiamo usare principalmente due modi, con il gradiente numerico (1.8) o con il calcolo del gradiente analitico.

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \qquad (1.8)$$

Con il primo modo, lungo ogni dimensione, quindi parametro di W, introduciamo una piccola variazione Δw e calcoliamo la derivata parziale lungo quella dimensione utilizzando l'equazione (1.8), dove f è la funzione costo, si capisce che il calcolo numerico del gradiente è approssimativo e lento poiché con W di forma (10x3073) a cui è concatenato il vettore di offset, dovremo calcolare 30730 derivate parziali e questo è per un solo aggiornamento di W.

Con il gradiente analitico invece una volta che abbiamo l'espressione corretta del gradiente, il calcolo computazionale delle derivate parziali è molto più veloce e preciso, con lo svantaggio però che ci possono essere degli errori nel determinare l'espressione, per questo motivo si introduce una procedura detta **Gradient check** che ci dà la possibilità di utilizzare il gradiente analitico nella fase di addestramento dopo aver verificato la sua correttezza comparandolo con il gradiente numerico. Per poter a questo punto aggiornare i parametri con l'intenzione di ridurre l'errore, utilizzeremo la versione **vanilla** della discesa del gradiente o **Gradient Descent** che è molto conosciuta per questo compito poiché è la più semplice da implementare.

$$newW = W - step_{size} * dW \qquad (1.9)$$

Nell'equazione (1.9) $newW$ sono i parametri aggiornati, la presenza del segno negativo è data dal fatto che il gradiente dW ha il verso diretto lungo pendenze positive, che è l'opposto al verso con cui W deve essere aggiornato, $step_{size}$ detto anche **Learning rate** è un iperparametro ed indica di quanto deve influire il gradiente nel aggiornamento in un solo passo, durante la fase di addestramento per poter raggiungere un minimo della funzione costo, sono necessari N passi e scegliere la lunghezza del passo con il learning rate nella realizzazione dei modelli di classificazione non è facile.

In analogia per intuire meglio, è possibile immaginare di trovarsi in un tratto pendente di una collina con gli occhi chiusi e con l'obbiettivo di raggiungere il punto più basso, se quindi viene fatto un piccolo passo (20 cm) nella direzione in cui si percepice la pendenza, si avrà maggiore probabilità di raggiungere un punto più basso rispetto al caso in cui il passo è molto grande(10 m anche se non fattibile da una persona) poiché magari ci potrebbe essere una risalita e quindi ritrovarsi in un punto più alto rispetto alla partenza.

1.6 Backpropagation

Un altro modo per il calcolo del gradiente di un'espressione che risulta molto comodo utilizzarlo nell'implementazioni delle reti neurali, è utilizzando la Backpropagation che consiste nell'utilizzare ripetutamente la regola della catena lungo tutto il percorso partendo dall'uscita della funzione costo L e andando indietro verso gli ingressi; la regola della catena in analisi matematica è una regola di derivazione di funzioni composte che a noi sarà molto utili poiché ci permetterà di derivare più facilmente funzioni composte molto lunghe che ci ritroviamo con reti neurali profonde e le reti neurali convoluzionali.

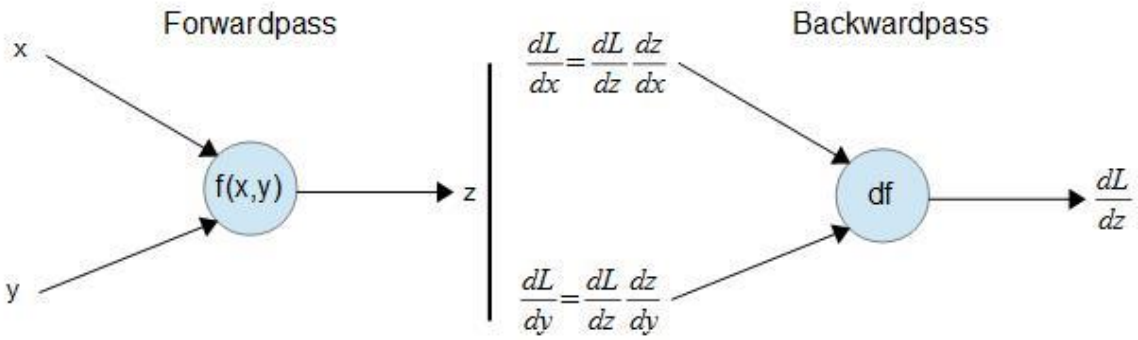


Figura 1.8: Schema del forwardpass e del backwardpass, rappresentati con funzioni e variabili.

Per poter ottenere i valori delle derivate parziali nel calcolo del gradiente all'indietro dobbiamo passare per due fasi, la prima la chiamiamo **forwardpass** in cui viene calcolato il valore finale della funzione costo e tutti i valori delle funzioni intermedie, nella seconda fase, abbiamo la **backwardpass** in cui partendo dalla derivata sulla funzione costo, applichiamo la regola della catena all'indietro fino ad ottenere tutte le derivate parziali sugli ingressi, una possibile rappresentazione è mostrata in figura 1.8.

$$f(x,w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \qquad (1.10)$$

Come vedremo più avanti, l'espressione (1.10) rappresenta un neurone a due dimensioni (due ingressi) che utilizza una funzione di attivazione sigmoidale, nella figura 1.9 (sotto), la stessa espressione è scomposta in una sequenza di funzioni elementari collegati in un grafo, in questo modo per calcolare le derivate parziali dell'uscita rispetto agli ingressi basta conoscere le derivate locali delle funzioni intermedie nel grafo e retro propagare la derivata con la regola della catena.

I valori numerici in colore verde sono stati calcolati nel forwardpass, quelli in rosso calcolati nel backwardpass, dopo quindi aver calcolato tutti i valori in verde, partendo da un gradiente sull'uscita di 1.00 (colore rosso) con $f(x) = \frac{1}{x}$; $\frac{df}{dx} = -\frac{1}{x^2}$ e x=1.37, avremo che la derivata locale sul nodo $\frac{1}{x}$ è -0.53 moltiplicata per la derivata a valle 1.00 rimane -0.53 e così via fino agli ingressi.

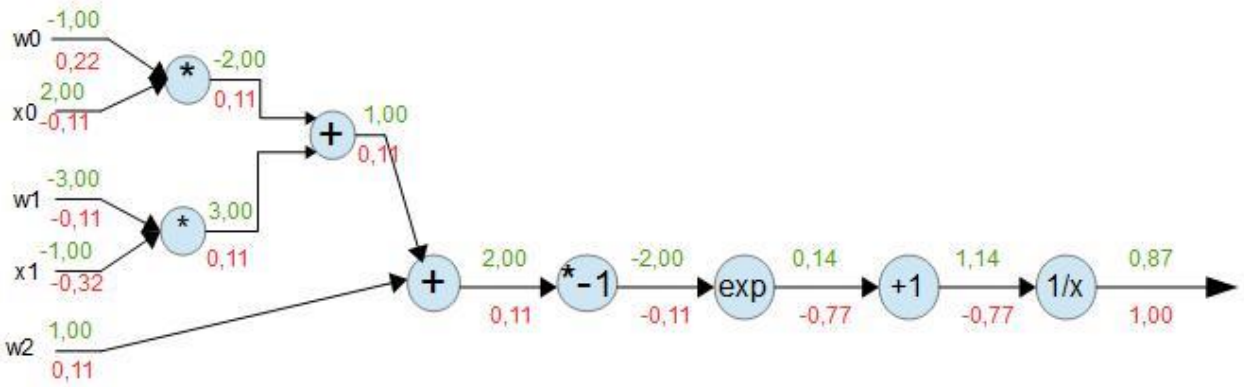


Figura 1.9: Rappresentazione con un grafo dell'equazione 1.10, scomposta nelle operazione elementari rappresentati sui nodi.

Capitolo 2

Valutazione dei vari Approcci di Machine Learning

Fino a pochi anni fa, chi si occupava di classificare un certo di tipo di dati, come immagini o suoni, doveva conoscere bene le sue caratteristiche e sapere fare le analisi necessarie per identificare quel particolare dato. Se stiamo parlando di immagini, alcune caratteristiche che si possono estrarre sono un istogramma dei colori o un istogramma dei gradienti orientati utilizzato nel computer vision come descrittore di caratteristiche che estrae contorni e bordi presenti nell'immagine; per i suoni può essere molto utile un'analisi nel dominio delle frequenze utilizzando ad esempio una "Short Time Fourier Transform". Quindi se si dovevano riconoscere le persone partendo dall'immagine del viso bisognava trovare le caratteristiche adatte da usare negli algoritmi per la classificazione e poter ottenere dei buoni risultati, il problema era che cambiando dataset in cui ad esempio bisognava riconoscere diversi animali, spesso è necessario riscrivere alcune parti dell'algoritmo e individuare nuove caratteristiche in comune utili ad distinguere tra i vari esempi del nuovo dataset, soprattutto se abbiamo altri tipi di dato, questa abilità piuttosto impegnativa diventa molto importante per avere delle buone performance che però si rischia di non avere più su un dataset diverso, serviva appunto un nuovo approccio con algoritmi che si possono implementare con poco impegno, e si possono generalizzare a molti tipi di dato e magari ottenere ottimi risultati nel riconoscimento e nella classificazione. Uno dei primi passi verso il nuovo approccio è un algoritmo chiamato "perceptron" inventato nel 1957.

2.1 Perceptron

Il perceptron inizialmente fu realizzato su Hardware, in effetti era una vera e propria macchina di grandi dimensioni rispetto ai computer che conosciamo oggi, comunque successivamente è nata anche una versione software con la stessa funzione:

$$f = \begin{cases} 1 \; se \sum_{j=1}^{n} w_j x_j + b > 0 \\ -1 \; se \sum_{j=1}^{n} w_j x_j + b < 0 \end{cases} \qquad (1.1)$$

Si noti che come funzione di attivazione si ha la funzione segno (1.1), e che i parametri w_j vengono anche detti pesi del perceptron.

Nei primi anni di questa invenzione l'entusiasmo era enorme poiché si pensava di poter risolvere nuovi tipi di problemi con più facilità, però in quegli anni è stato dimostrato che l'algoritmo dopo averlo addestrato era in grado di riconoscere solo funzioni linearmente separabili, da lì l'interesse scese. Un funzione non lineare per cui l'algoritmo era limitato è la funzione booleana XOR, che più avanti nel teorema di approssimazioni universali delle reti MLP (multilayer perceptron), si dimostrò ad esempio che il problema dell'XOR si poteva risolvere aggiungendo almeno un livello in più nella rete, come si può vedere nella figura 2.1, ottenendo così un perceptron multi livello, questa soluzione ha aumentato la capacità del modello permettendogli appunto di approssimare anche funzioni non lineari sotto alcune condizioni, ovvero il fatto di avere un numero sufficiente di unità (o perceptron) nel livello nascosto aggiunto, anche detto "hidden layer".

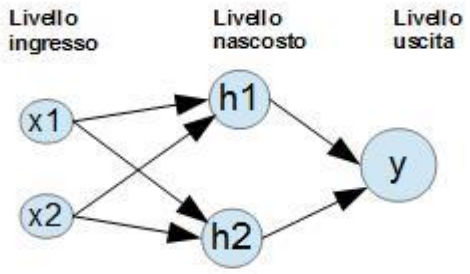
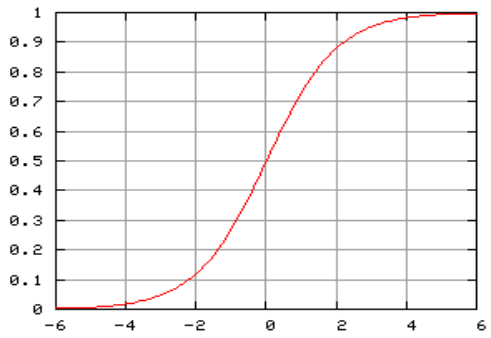


Figura 2.1: Rappresentazione di un perceptron a due livelli (livello di ingresso non viene contato).

In seguito all'algoritmo del perceptron, si inizio a parlare di reti neurali come un approccio computazionale basato su una larga collezione di unità neurali ispirati al cervello biologico. Queste unità interconnesse e organizzate su più livelli erano in grado risolvere compiti nel computer vision e speech recognition molto difficili se venisse usata una programmazione basata su regole e istruzioni eseguite in serie.

Due modi in cui queste reti possono essere interconnesse sono, **feedforward** e **recurrent**, nel primo modo le unità di un livello possono essere collegate solo verso le unità del livello successivo a differenza del recurrent in cui possono esserci anche connessioni tra unità dello stesso livello o verso livelli precedenti.

Nell'uscita di ogni unità (o neurone) è presente una funzione di attivazione, una di queste è la funzione sigmoidea figura 2.2, inizialmente introdotta poiché è in grado di descrivere bene il modello biologico del neurone ma non più usata in pratica dato che presenta diversi svantaggi tra cui l'uscita non centrata sullo zero e l'uscita che satura, da cui si hanno problemi nel backward pass, questo oltre al costo computazionale della funzione sigmoidea, cosi nel tempo è stata sostituita ad esempio dalla funzione **ReLU** (Rectified Linear Unit) figura 2.3, che utilizzerò nella rete neurale a due livelli.



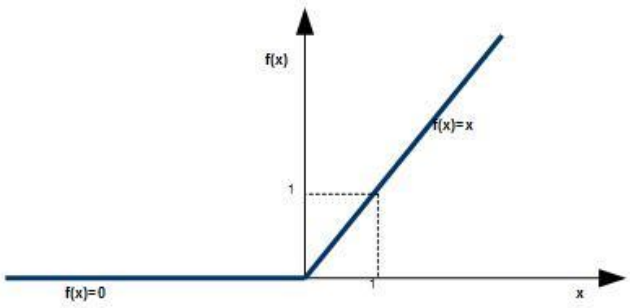


Figura 2.2: (Sinistra) Grafico della funzione sigmoidea preso da Wikipedia. Figura 2.3 (Destra) Grafico approssimativo della funzione Rectified Linear Unit.

2.2 Classificatori Lineari

Basandomi sui compiti assegnati dal corso [2] del università di Stanford, ho potuto valutare diverse tecniche nella classificazione degli immagini del Dataset CIFAR-10 [1] , in particolare partendo dal classificatore lineare e poi passando per le reti neurali fino ad introdurre un classificatore con le reti convoluzionali, in questo modo sono riuscito a notare diversi aspetti chiave molto importanti per ottenere un buon classificatore. In genere nella rappresentazione a blocchi del sistema di classificazione si ha un primo blocco di estrazione delle caratteristiche dai dati, la cui uscita è un vettore, ed è l'ingresso al blocco classificatore, nei compiti che ho fatto per le varie tecniche, inizialmente non è stato utilizzato il primo blocco ma ho lavorato direttamente sui pixel dell'immagine, in seguito ho fatto notare la differenza lavorando sui vettori di caratteristiche.

Per classificare ho utilizzato un classificatore lineare multi-classe prima con la **hinge loss** e poi con la **cross entropy loss**, osservando la score function di questo, si può notare che non è altro che un perceptron multi-classe senza funzione di attivazione; In ingresso, alla matrice x di numeri che vanno [0 – 255] viene sottratta la media effettuata su ogni dimensione per tutto il training set, ottenendo così immagini con media centrata sullo zero, in seguito la matrice di parametri W viene inizializzata con una distribuzione normale standardizzata, ricordo che nella matrice W si hanno 10 classificatori che durante l'addestramento vengono passo per passo settati per poter riconoscere le varie classi, ciò lo facciamo con il metodo della discesa del gradiente che però necessita appunto del gradiente della funzione costo rispetto ai parametri. Per il gradiente della hinge loss ho utilizzato l'espressione fornita dal sito del corso [2].

2.2.1 Multi-class Support Vector Machine

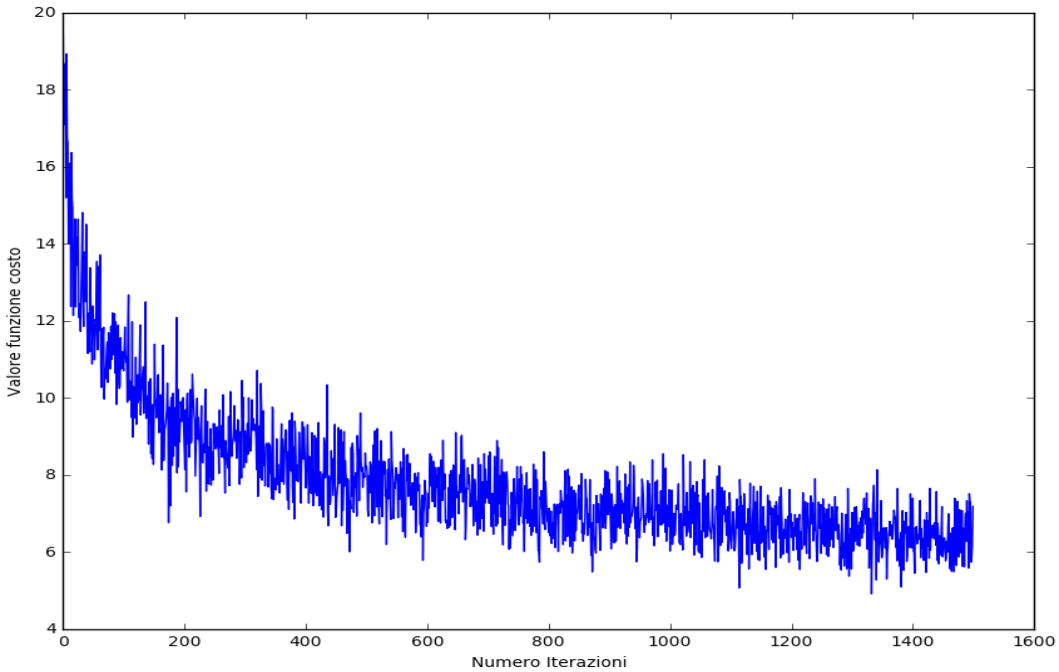


Figura 2.4: Grafico che presenta l'andamento della funzione costo durante l'addestramento sul training set compiuto in 1500 iterazioni.

Nella figura 2.4 vediamo come in questo esempio siamo riusciti a ridurre il valore della hinge loss del multi-class support vector machine, convergendo al valore di 7.391086 dopo 1500 iterazioni, utilizzando un learning rate di 10^{-7}, un regularization strenght di $5 * 10^4$ e un batch-size = 200, quest'ultimo è un iperparametro come gli altri due, ed indica quanti esempi del sottoinsieme vengono presi in modo casuale dal training set, l'andamento rumoroso è dovuto al fatto di utilizzare il metodo della discesa del gradiente a mini-batch, ciò vuol dire che ad ogni iterazione per poter effettuare un passo e quindi aggiornare la matrice W, viene calcolato il gradiente su un mini-batch di 200 esempi, quindi di piccola dimensione rispetto al training set che è di 49000 esempi, da cui si ha l'aspetto rumoroso.

Hyperparameter tuning

Spesso accade nell'implementare questi algoritmi di machine learning di non sapere quale valore attribuire agli iperparametri, per questo esiste una procedura chiamata "hyperparameter tuning" che ci aiuta a ricercare i valori che ci portano a convergere sui minimi della funzione costo, questa procedura ci servirà nelle diverse tecniche descritte in questo elaborato.

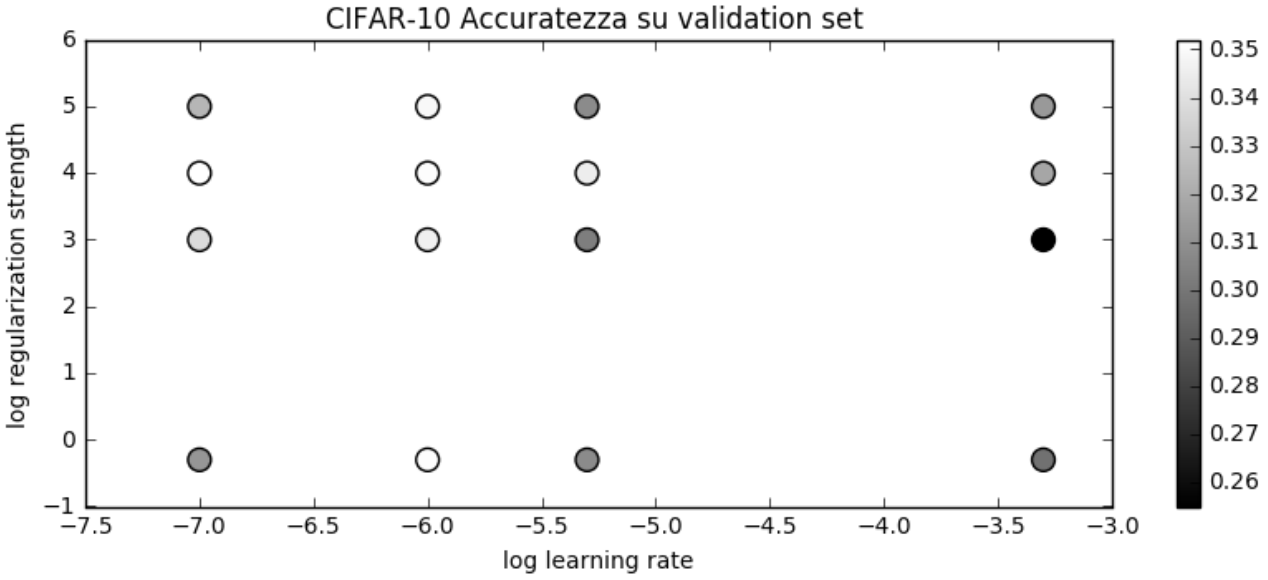


Figura 2.5: Visualizzazione dei diversi valori dell'accuratezza sul validation set in basa al colore scuro, sulle diverse combinazioni dei due iperparametri.

Questa procedura consiste nell'usare alcuni valori per ciascun iperparametro e per ogni combinazione di questi viene fatto l'addestramento ottenendo così la matrice W con cui andiamo a valutare le performance sul validation set, nel frattempo si tiene traccia della migliore combinazione e nell'intorno di questa vengono scelti altri valori da usare di nuovo nella procedura, ripetendo cosi fino a convergere verso un certo valore dell'accuratezza, nel mio caso si può intuire dalla figura 2.5 che la combinazione migliore è data da : learning_rate=10^{-6} , regularization_strenght= 10^{4} , con un numero di iterazioni pari a 2500, si ottiene una accuratezza di 0.367 sul validation set , a questi valori degli iperparametri è associata una matrice W con cui utilizzando il Linear SVM sui pixel si ottiene una **accuratezza sul test set di 0.352** .

2.2.2 Softmax

Utilizziamo ancora il classificatore lineare ma con il cross entropy loss, di cui ho dovuto determinare l'espressione del gradiente per poi implementarla nel codice, ricordo come si può vedere nell'espressione (2.2) che il loss finale è dato dalla somma del data loss e il regularizattion loss, quindi pure il gradiente finale sarà una somma, in seguito mostrerò solo il calcolo del gradiente relativo al data loss, per l'altra componente il calcolo consiste nella moltiplicazione per un coefficiente.

$$L = \frac{1}{N}\sum_i L_i + \gamma\, R(W) \qquad (2.2)$$

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) = \log\left(\sum_j e^{f_j}\right) - \log\left(e^{f_{y_i}}\right) \qquad (2.3)$$

$$L_i = \log\left(\sum_j e^{w_j^T x_i}\right) - \log\left(e^{w_{y_i}^T x_i}\right) \qquad (2.4)$$

Nel calcolo del gradiente dell'espressione (2.4), dobbiamo calcolare sia rispetto a $w_{y_i}^T$ che sono i parametri di W relativi alla classe corretta del esempio di indice i (parte1) ,sia rispetto a w_j^T che sono i parametri relativi alle altre classi (parte2).

Part1:

$$\nabla * L_{i_{w_{y_i}^T}} = \frac{\frac{\partial\left(\sum_j e^{w_j^T x_i}\right)}{\partial w_{y_i}^T}}{\sum_j e^{w_j^T x_i}} - \frac{\frac{\partial\left(e^{w_{y_i}^T x_i}\right)}{\partial w_{y_i}^T}}{e^{w_{y_i}^T x_i}} = \frac{\frac{\partial\left(e^{w_{y_i}^T x_i}\right)}{\partial w_{y_i}^T}}{\sum_j e^{w_j^T x_i}} - \frac{e^{w_{y_i}^T x_i}}{e^{w_{y_i}^T x_i}}\, x_i \tag{2.5}$$

$$\nabla * L_{i_{w_{y_i}^T}} = x_i\left(\frac{e^{w_{y_i}^T x_i}}{\sum_j e^{w_j^T x_i}} - 1\right) = x_i\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_{y_i}}} - 1\right) \tag{2.6}$$

Parte2:

$$\nabla * L_{i_{w_j^T}} = \frac{\frac{\partial\left(\sum_j e^{w_j^T x_i}\right)}{\partial w_j^T}}{\sum_j e^{w_j^T x_i}} - \frac{\frac{\partial\left(e^{w_{y_i}^T x_i}\right)}{\partial w_j^T}}{e^{w_{y_i}^T x_i}} = x_i \frac{e^{w_j^T x_i}}{\sum_j e^{w_j^T x_i}} = x_i \frac{e^{s_j}}{\sum_j e^{s_j}} \tag{2.7}$$

A questo punto come si può vedere i gradienti nelle espressioni 2.6, 2.7 sono relativi ad un solo esempio del training set, quindi bisogna poi divedere per il numero di esempi su cui si intende calcolare il gradiente, questo numero l'avevo introdotto come batch-size, che anche questa volta per la classificazione su CIFAR-10 [1] con softmax gli ho assegnato il valore 200, per quanto riguarda gli altri due iperparametri ho usato la stessa procedura descritta prima, del hyperparameter tuning, ottenendo come combinazione migliore la seguente: learning_rate= 10^{-6} e regularizattion_strenght= 10^4 e con un numero di iterazioni pari a 2500, ho ottenuto la matrice W che mi ha dato la migliore accuratezza sul validation set, nel test set usando softmax sui pixel ho ottenuto **una accuratezza di 0.382** .

2.3 Classificatore con Rete Neurale

Proviamo a classificare le immagini del dataset CIFAR-10 [1], usando un modello più complesso rispetto a quello visto fino ad ora, nel mio caso per classificare le immagini del test set, ho utilizzato una rete neurale a due livelli, con un livello di input da 3072 (che sono i pixel) questo livello solitamente non viene contato, un livello nascosto da 500 unità (o neuroni) ed un livello di output di 10 uguale al numero delle classi, in quest'ultimo livello non è presente la funzione di attivazione che invece è presente nel livello nascosto in cui viene usata la funzione ReLu. La rete neurale usata è sia feedforward che fully-connected, cioè ogni unità di un livello è collegata a tutte le unità del livello successivo.

$$s = W_2 \max(0, W_1 x + b_1) + b_2 \qquad (2.8)$$

La funzione score (2.8) utilizzata in questa rete è più complessa rispetto all'espressione (1.1) per il linear classifier, questa volta abbiamo due matrici dei parametri W_1 e W_2 e due vettori colonna di biases b_1 e b_2 . Per semplicità nei calcoli si è fatto in modo che ogni colonna di W_1 rappresenti i parametri (o pesi) di un unità del livello nascosto e così anche per W_2 dove le sue colonne sono i parametri di una delle unità del livello d'uscita, in questo modo per ogni livello della rete abbiamo un'unica matrice dei parametri a cui vengono concatenati i vettori di bias offset.

Backpropagation

$$\nabla * L_{i_{s_{y_i}}} = \frac{e^{s_{y_i}}}{\sum_j e^{s_{y_i}}} - 1 \tag{2.9}$$

$$\nabla * L_{i_{s_j}} = \frac{e^{s_j}}{\sum_j e^{s_j}} \tag{2.10}$$

Nel calcolo del gradiente della funzione costo rispetto a tutti i parametri della rete, data la lunghezza della rete è necessario usare la procedura del backpropagation, in questo calcolo per facilitare la comprensione della procedura avevo disegnato un grafo, tipo quello in figura 1.9, che mi aiutava a vedere tutti i calcoli necessari nel forwardpass e quali risultati in questo passo devono essere raccolti e memorizzati per poterli poi usare nel backwardpass; è stata utilizzata la cross entropy loss e di questa funzione serviva il gradiente rispetto ai voti, ma avendo già determinato l'espressione del gradiente analitico rispetto ai parametri, è bastato rivedere la soluzione finale ed ottenere i gradienti nell'espressioni (2.9) e (2.10) , usando poi la regola della catena avevo determinato tutte le espressioni dei gradienti lungo la rete necessarie per aggiornare i parametri, il passo più difficile è stato trasformare ciò in codice efficiente usando un implementazione vettorizzata, grazie però alla procedura del gradient check che utilizzando il gradiente numerico mi ha permesso di verificare la correttezza dell'implementazione; questa volta per aggiornare i parametri non è stato usato il gradient descent che presentava alcuni svantaggi ma è stato utilizzato un metodo chiamato "momentum update".
Dopo avere controllato anche che non ci siano errori nella fase di addestramento, ho scritto il codice per la procedura dell'hyperpameter tuning che mi ha consentito di trovare la combinazione migliore dei quattro iperparametri.

In questa procedura appunto, sono stati utilizzati altri due iperparametri, di cui il numero di unità nel livello nascosto e il numero di iterazioni che nei classificatori precedenti ho tenuto fissato al valore di 2500 iterazioni, alcune volte però potrebbe essere utilizzato il numero di epoche di addestramento, un epoca consiste in un addestramento completo sul intero training set; in fine usando un learning_rate= 10^{-3}, regularization strenght= 0.5, n_hiddenLayer=500, n_iterazioni= 2500 ho ottenuto un **accuratezza sul test set di 0.535** .

2.4 Comparazione e analisi dei parametri

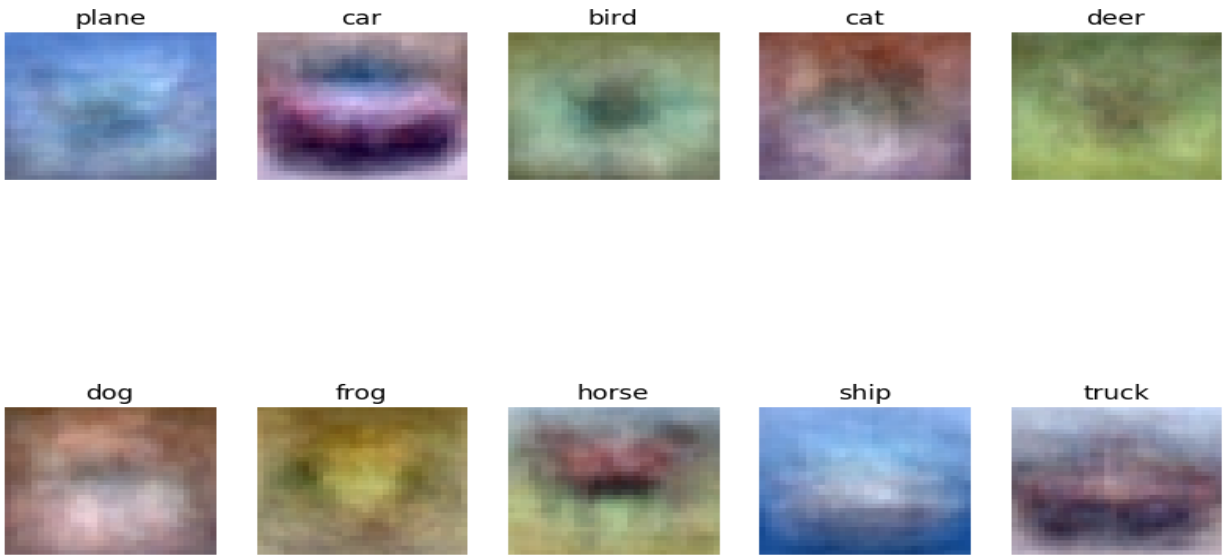


Figura 2.6: Visualizzazione della matrice dei parametri W del linear classifier con softmax, di cui sono stati mostrati i risultati nella classificazione; ogni riga della matrice messa nella forma (32*32*3) presenta l'aspetto mostrato.

Figura 2.7: Visualizzazione dei parametri della matrice W_1 , cioè i parametri del livello nascosto della rete neurale a due livelli che è stata realizzata, ogni cella nella figura rappresenta i parametri di un'unità tra le 100 presenti in questo livello.

Un modo per rendere più intuitiva la comprensione del funzionamento di questi modelli di classificazione delle immagini, è di visualizzare i parametri del modello ottenuti alla fine della fase di addestramento, ad esempio nella figura 2.6 ogni classificatore con soli 3073 parametri (3072 di w_i + 1 di offset) sta tentando di classificare 10 classi di cui ciascuna con 5000 esempi nel training set, quindi per poter realizzare questo compito, è come se il classificatore lineare fa in modo che ogni classificatore associato ad un certa classe sia la media di tutti gli esempi per quella classe, anche se in realtà non è una media ma si può vedere nel classificatore della classe cavallo la presenza di due teste una rivolta verso destra o una verso sinistra, questo accade perché i due casi sono anche presenti negli esempi del training set, si capisce che la capacità del modello non è sufficiente e con un numero maggiori di classi avremo avuto performance peggiori.

Con le reti neurali invece si ha maggiore capacità, data per esempio da un maggiore numero di parametri che nel caso studiato sono disposti su due livelli e in ciascun livello nelle varie unità, con questa architettura il compito di riconoscere un immagine tra migliaia di esempi viene suddiviso e distribuito sulle varie unità, un mio modo di interpretare questo funzionamento, è vedere ad esempio che il classificatore della classe auto, che è un'unità presente nel livello d'uscita per decidere se l'immagine in ingresso è appartenente alla classe o meno, non deve lavorare sull'intero spazio multi dimensionale dei pixel ma esso lavora solo su alcune caratteristiche estratte dall' immagine nel livello nascosto che invece lavora sui pixel, si può ad esempio vedere nella figura 2.7 , in cui sono presenti 100 unità ciascuno con il compito di classificare una caratteristica; dato che la rete usata è fully-connected il classificatore dell'auto nel livello d'uscita possiede (100 + 1) parametri, il valore di ciascun di questi parametri (escluso quello di offset) misura in un certo senso quanto viene gradito l'ingresso a cui è collegato, se ad esempio l'ingresso è dato dall'uscita di un classificatore presente nel livello nascosto per le auto rivolte verso destra allora il valore dell'parametro associato a quell'uscita sarà alto, ho disegnato la figura 2.8 per mostrare questo ragionamento, le reti con più di un livello nascosto vengono chiamate reti neurali profonde o "**deep neural network**" questi generalmente hanno maggiore capacità e sono maggiormente in grado di estrarre caratteristiche specifiche dai pixel dell'immagine.

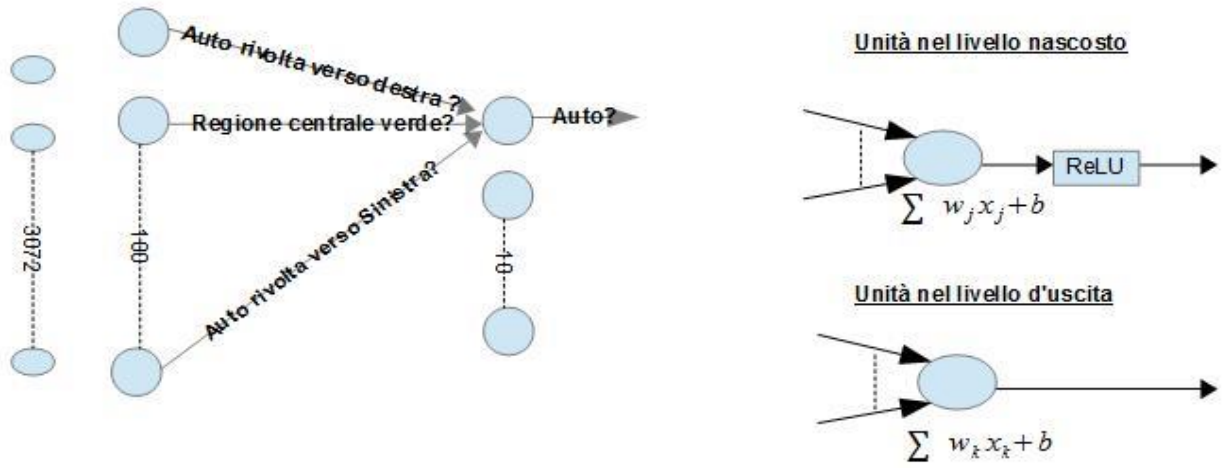


Figura 2.8: Rappresentazione approssimativa della rete neurale a due livelli, i due livelli sono fully-connected anche se per semplicità non è mostrato.

2.5 Classificazione con estrazione di Caratteristiche

Nel paragrafo 2.4, ho interpretato la rete neurale a due livelli come un classificatore lineare dato dall'livello d'uscita che ha in ingresso un vettore di caratteristiche estratte dai pixel nel livello nascosto, ottenendo un valore dell'accuratezza maggiore rispetto al classificatore lineare presentato nel paragrafo 2.2. Verrebbe quindi da chiedersi se utilizzando un classificatore lineare su un vettore ottenuto da un estrattore di caratteristiche che dà in un uscita un istogramma dei gradienti orientati (HOG) o un istogramma dei colori, sia possibile ottenere performance simile a quelli della rete neurale a due livelli.

Usando il codice fornito dal corso [2] per estrarre questi istogrammi da ciascuna immagine del dataset CIFAR10 [1], ottenendo in uscita per ogni esempio un vettore di dimensione 155 contenente i due istogrammi, ho addestrato il classificatore lineare con la hinge loss su questi vettori in modo da ottenere i migliori risultati e in fine ho avuto un accuratezza sul **test set di 0.48** , per poter rendere più simile il confronto ho utilizzato poi una rete neurale a due livelli con un livello nascosto da 155 unità, ottenendo una accuratezza di 0.509 , al di là della piccola differenza tra i due valori penso che l'approccio con le reti neurali è migliore poiché le caratteristiche estratte sono più adatte per il motivo che sono il risultato della fase di addestramento in cui l'obbiettivo è di minimizzare la funzione costo e quindi anche massimizzare l'accuratezza su quel tipo di dataset.

Avendo la possibilità, ho anche scritto il codice per addestrare una rete neurale a due livelli sui vettori di caratteristiche dei due istogrammi raggiungendo un accuratezza sul **test set di 0.585** contro quella che avevo ottenuto sui pixel di 0.535.

2.6 Rete neurale Convoluzionale

Abbiamo visto che l'approccio con le reti neurali che presentano almeno un livello nascosto è piuttosto innovativo rispetto alle tecniche tradizionali di classificazione e riconoscimento dei pattern, poiché questi modelli permettono di estrarre automaticamente le caratteristiche che il modello stesso ritiene più adatte per svolgere il compito assegnato. Quindi nella classificazione delle immagini per migliorare le performance non è necessario aggiungere una componente esterna e forzare il modello all'utilizzo delle caratteristiche estratte da questa, che magari non sono le migliori per quel tipo di dataset, aggiungendo poi altri livelli nascosti alla rete si ottengono maggiori performance con lo svantaggio però di ritrovarsi con un numero molto grande di parametri, che con l'architettura della rete a livelli fully-connected, spesso ci si ritrova nel problema del overfitting, oltre al maggiore costo computazionale, soprattutto quando si hanno reti molto profonde con i livelli fully-connected, questo è uno dei motivi per cui serviva un nuovo approccio, sempre però con il modello del neurone artificiale, questo approccio prende il nome di **rete neurale convoluzionale,** questo tipo di rete fa parte delle reti neurali feedforward ma in cui lo schema di connessione tra i suoi neuroni è ispirato all'organizzazione della corteccia visiva animale, dove si osservò che gruppi di neuroni della corteccia rispondevano a stimolazioni visive solo verso una regione ristretta dello spazio visivo detta dall'inglese 'receptive field'.

Il nome attribuito a queste reti neurali è dovuto all'utilizzo dell'operazione matematica di convoluzione che opera su due funzioni (f e g) e produce in uscita una terza funzione z, nel nostro caso (f e g e z) sono matrice e hanno dimensioni diverse.

Introduciamo un nuovo tipo di livello chiamato livello convoluzionale o dall'inglese 'convolutional layer' ottenuto usando l'operazione di convoluzione con diverse matrici detti filtri (o kernel) sulla matrice d'ingresso, in questo livello i parametri che si possono aggiustare durante l'addestramento si trovano in matrici detti appunto filtri che in un solo livello ce ne possono essere più di uno, più avanti nel paragrafo 2.8 vengono visualizzati per interpretare la loro funzione sull'immagine d'ingresso; eseguendo l'operazione di convoluzione con n filtri sull'ingresso si ottengono n mappe di attivazione (o future maps) che diventano l'ingresso per il livello successivo, quindi la possibilità di usare solo un piccolo numero di parametri per ciascun filtro e facendolo scorrere lungo tutto l'ingresso cambiando regione percettiva ma operando nelle varie regioni condividendo gli stessi parametri, riduce di molto il numero di parametri ottenendo un modello più efficiente.

Un altro livello introdotto ma non usato nelle reti neurali ordinarie, è il livello di pooling che data la grandezza che possiedono questo tipo di reti, esso viene usato per prevenire l'overfitting poiché ha la funzione di sotto campionare l'ingresso riducendolo anche del 50/75% e riducendo pure il costo computazionale.

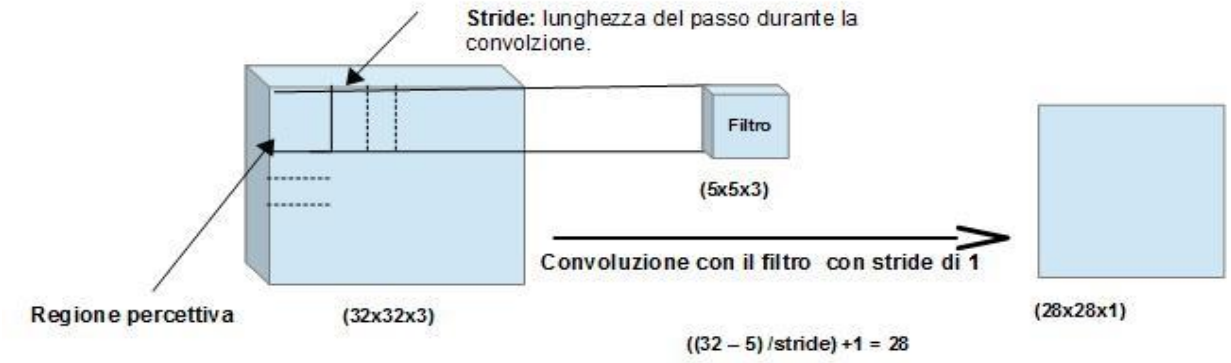


Figura 2.9: Rappresentazione di un esempio di filtro che operando con un convoluzione su un ingresso ottiene in uscita una matrice bidimensionale.

2.7 Classificazione con Rete neurale Convoluzionale

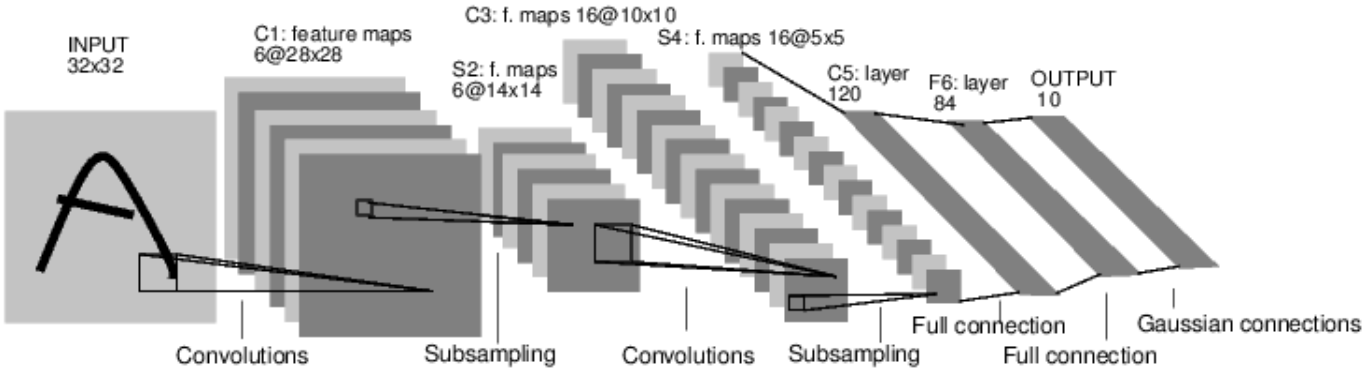


Figura 2.10: Architettura di LeNet-5, è una rete neurale convoluzionale presa dal articolo di Yann LeCun et al [5].

Nella figura 2.10 vediamo un esempio di una rete neurale convoluzionale utilizzata per riconoscere immagini di cifre manoscritte, la rete ha un architettura del tipo [Convolutional layer -> Pooling layer -> Convolutional layer -> Pooling layer -> Fully connected -> fully-connected], ed utilizza filtri 5x5 con stride di 1 e sotto campionamento con max-pooling con regioni percettive da 2x2 applicati sugli ingressi con stride di 2.

Oltre agli iperparametri introdotti nelle altre tecniche di classificazione, abbiamo anche il numero di filtri da usare in ciascun livello convoluzionale e lo stride che viene applicato per essi durante l'operazione di convoluzione, in più abbiamo le dimensioni del filtro che consiste in un solo numero applicato ugualmente per la dimensione orizzontale e verticale, il filtro ha anche una terza dimensione (profondità) che deve essere obbligatoriamente uguale alla terza dimensione dell'ingresso.

Per poter confrontare il nuovo approccio con le tecniche studiate in questo elaborato, supportandosi da parti di codice fornite dal corso [2], ho implementato una semplice architettura che aggiunge alla rete neurale a due livelli studiata nel paragrafo 2.4 solamente altri due livelli, uno di convoluzione e l'altro di pooling, giungendo a questa architettura: [Conv - relu - Pooling – F.Connected - relu – F.Connected – softmax] usando poi la procedura del hyperparameter tuning, ho scelto come iperparametri un learning_rate= 10^{-4} , regularization_strenght= 0.001, 500 unità nel livello nascosto della rete neurale a due livelli, 20 filtri di dimensione 3x3 nel livello convoluzionale applicati con stride pari a 1, addestrando poi la rete con questi valori, ho impiegato circa 30 minuti con il mio notebook utilizzando un numero di epoche pari a 3, ho utilizzato poi il modello con i parametri aggiornati per classificare sul test set di CIFAR-10 [1] ottenendo un valore **dell'accuratezza di 0.632 .**

2.8 Visualizzazione dei parametri e Confronto

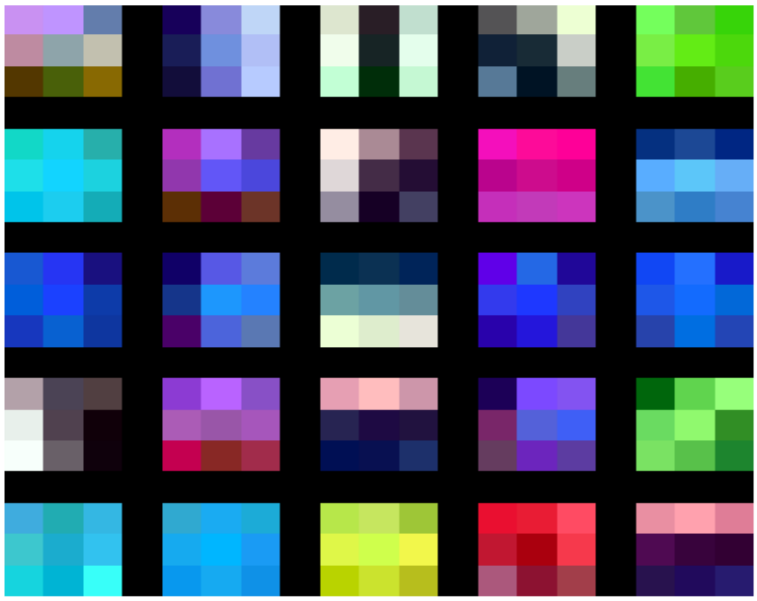


Figura 2.11: Visualizzazione dei 20 filtri (3x3), alla fine della fase di addestramento, ottenuti dal primo livello della rete convoluzionale di cui sono stati mostrati le performance nel paragrafo 2.8.

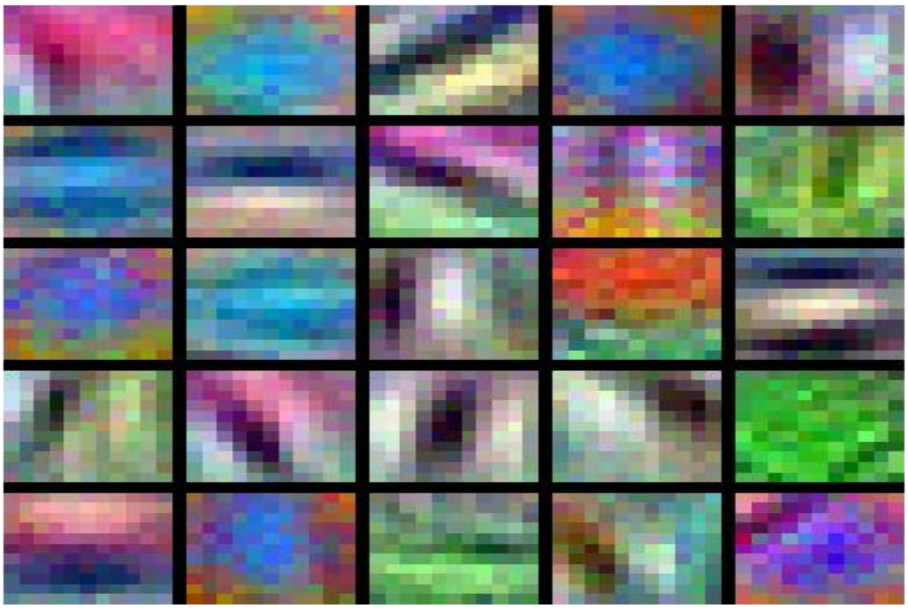


Figura 2.12: 25 filtri da (11x11) ottenuti dal primo livello di una rete convoluzionale.

Vediamo come in questa architettura i parametri del modello connessi direttamente ai pixel, messi in una forma visualizzabile cioè con matrici bidimensionali sui tre canali di colore, hanno un aspetto meno intuitivo rispetto ai parametri visualizzati nel paragrafo 2.5 del classificatore lineare e rete neurale, questo perché ogni filtro che si vede nella figura 2.12 rappresenta una caratteristica molto specifica che l'operazione di convoluzione utilizza per dare in uscita un specie di mappa che quantifica la presenza di questa caratteristica in ciascuna delle regioni percettive che si possono ottenere lungo l'intera immagine, ad esempio con un immagine di (32x32x3) e un filtro (5x5x3) utilizzando uno stride di 1 si hanno (28x28) regioni percettive; quindi se si vuole per esempio riconoscere l'immagine di un auto, nel primo livello, un filtro ricerca bordi orientati diagonalmente, l'altro bordi verticali, l'altro ancora ricerca lungo l'immagine regioni della grandezza del filtro con del blu, se ci fosse poi un livello successivo, convoluzionale o fully-connected verrebbero ricercate caratteristiche più complesse come la presenza di ruote o fanali, e nell'ultimo livello viene assemblato il tutto per decidere se l'immagine è un auto o meno dando un voto alla classe auto che poi verrà confrontato con il voto dato alle altre classi.

L’architettura di rete convoluzionale che ho utilizzato nel paragrafo 2.8 per classificare, viene chiamata rete neurale a tre livelli e possiede appunto tre livelli in cui sono presenti i parametri, se alla fine della fase di addestramento supponiamo che il primo livello sia un blocco separato, ci verrebbe in mente il caso visto nel paragrafo 2.6 in cui c’era un blocco che estraeva istogrammi di gradienti orientati e istogrammi di colore, e venivano utilizzati dalla rete neurale a due livelli, ottenendo però un accuratezza minore rispetto a quella del nuovo approccio pur essendo equivalenti da un certo punto di vista, la ragione di questo è che il primo livello della rete convoluzionale ha estratto dall’immagine caratteristiche più adatte per la classificazione sul dataset CIFAR-10 [1], rispetto alle caratteristiche dei due istogrammi.

La tecnica di classificazione con rete neurale convoluzionale, è stata quella ad ottenere il valore più alto di accuratezza rispetto alle altre tecniche studiate in questo elaborato.

Capitolo 3

Conclusioni e lavori futuri

Questa tesi ha avuto come obbiettivo la valutazione di vari modelli di machine Learning e la comprensione di base dei principi del loro funzionamento nell'attività di riconoscere e classificare le immagini.

A tale scopo, si è partiti da un modello semplice di classificatore lineare, costituito da una funzione score calcolata principalmente con una singola moltiplicazione tra la matrice dei parametri e quella dell'immagine d'ingresso, seguita da una funzione costo, per poter poi aggiornare i parametri è stato sufficiente utilizzare l'espressione del gradiente della funzione composta; nel secondo modello di rete neurale si è utilizzato le stesse funzioni costo del modello iniziale, ma con una funzione score che ci è richiesto più di una moltiplicazione per la presenza di due matrici dei parametri, nel aggiornamento dei parametri è stato necessario l'introduzione e l'uso di un nuovo metodo di backpropagation che richiede la determinazione del gradiente analitico di espressioni elementari e concatenarle lungo la rete con la regola della catena, nel terzo modello di rete neurale convoluzionale l'implementazione delle funzioni costo è rimasta invariata, ma è stata concatenata alla rete neurale a due livelli l'operazione di convoluzione, questa operazione è la parte principale del nuovo approccio che attraverso la condivisione dei parametri, migliora l'efficienza del modello.

Ogni volta dopo aver implementato ciascuno di questi modelli, vengono valutati le loro performance sul dataset CIFAR-10 [1], con il classificatore lineare insieme alla cross entropy loss, è stata ottenuta un accuratezza sul test set leggermente maggiore a quella ottenuta con la hinge loss, questa differenza non ha un significato evidente,

Poiché dipende da alcuni dettagli non considerate in questo elaborato, nella pratica di solito la differenza di performance tra i due è piccola, e varia a seconda del compito in cui vengono impiegati, la rete neurale a due livelli ha ottenuto un valore dell'accuratezza sempre lavorando sui pixel abbastanza maggiore rispetto al modello iniziale, questa rete essendo solo di due livelli, durante l'addestramento si sono riscontrati problemi di convergenza sui minimi locali della funzione costo quando veniva utilizzato il gradient descent per aggiornare i parametri, per cui è stato usato momentum update notando anche un miglioramento, la capacità di questo modello varia ad esempio con il numero di livelli nascosti e il numero di unità per ciascuno di questi livelli, quindi per limitare l'overfitting non è necessario limitare la capacità del modello, ma è sufficiente impiegare delle tecniche di regolarizzazione alla rete, nel mio caso date le dimensioni della rete è stata impiegata un'unica tecnica che limitava le dimensioni parametri, ho notato poi che utilizzando un numero di unità nel livello nascosto pari a 2000 si osservava una differenza del 25% nel accuratezza calcolata sul training set e il validation set, contro il 10% usando 500 unità, il fatto poi che la rete abbia i due livelli fully-connected porta ad un maggiore costo computazionale nell'classificare immagine ad alta risoluzione, un problema che il terzo modello non possiede, il maggiore numero di iperparametri che la rete neurale convoluzionale implementata possedeva, ha reso un po' più difficile la ricerca della combinazione migliore, ma alla fine è stata attenuta una accuratezza nell'intorno del valore del 65%, che il corso [2] considera buono per il dataset CIFAR-10 [1].

In conclusione per poter migliorare il modello nel compito di classificazione delle immagini, sarebbe necessario l'utilizzo di più livelli convoluzionali alternati a livelli di pooling, con il potenziale di ottenere una accuratezza facilmente maggiore del 70% sul dataset CIFAR-10 [1], però con architetture di quelle dimensione è necessario impiegare altre tecniche di regolarizzazione e sarebbe anche meglio l'impiego di tecniche per la corretta inizializzazione dei parametri. Nonostante ciò, il nuovo approccio è sotto ricerca molto attiva, e alcuni studi stanno tentando di minimizzare il costo computazionale per poterlo impiegare nei sistemi embedded, comunque è di gran lunga migliore nel riconoscimento dei pattern rispetto alle tecniche tradizionali utilizzate fino a poco tempo fa.

Bibliografia

[1] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton, Learning Multiple Layers of Features from Tiny Images, Capitolo 3, pp. 32-36, Alex Krizhevsky, Università di Toronto, 2009.
[2] Andrej Karpathy, Justin Johnson, Fei-Fei Li, (2015/2016), CS231n, da http://cs231n.github.io/
[3] Andrej Karpathy, (2011), Andrej Karpathy blog, da http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/
[4] Andrej Karpathy, Justin Johnson, Fei-Fei Li, (2015/2016), CS231n, da http://cs231n.github.io/linear-classify/
[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition.*Proceedings of the IEEE*, november 1998.
[6] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. Byers. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, 2011.