Introduzione

Ad oggi il carcinoma orale rappresenta il più diffuso dei tumori maligni del cavo orale ed è un serio problema per la salute umana con un impatto clinico in termini di incidenza, prevalenza e tassi di mortalità che non tende a migliorare[carcinoma orale manuale di riferimento lorenzio lo muzio sandro pelo]. In Italia ogni anno vengono diagnosticati circa 4500 casi di tumore alla bocca e si registrano circa 3.000 decessi. Purtroppo questo tipo di cancro viene di solito diagnosticato in fase avanzata, quando la massa tumorale si è ingrandita al punto da richiedere interventi mutilanti, spesso con scarsi risultati[airc]. È quindi essenziale che i professionisti della salute orale come dentisti, igienisti dentali, odontoiatri e terapisti orali comprendano l’importanza di condurre uno screening orale approfondito per le patologie maligne o potenzialmente maligne nell’ambito delle loro valutazione cliniche di routine.

Appare quindi opportuno lo sviluppo di sistemi automatici in grado di supportare le figure specialistiche nella diagnosi preventiva delle patologie del cavo orale. Un valido strumento di ausilio è sicuramente l’uso di algoritmi di machine learning, nel corso degli ultimi anni abbiamo assistito all’affermazione progressiva di questa branca dell’informatica, avente come obiettivo l’elaborazione di meccanismi tali da permettere alle macchine di simulare la capacità del cervello umano di apprendere e generalizzare le informazioni. Nello specifico, la loro capacità di imparare si basa essenzialmente sulla creazione di un’esperienza a partire da migliaia di esempi, da cui successivamente si ottengono previsioni o decisioni. Al giorno d’oggi le tecniche di Intelligenza Artificiale, ovvero di “apprendimento automatico”, sono presenti in molteplici strumenti della nostra vita, sia ordinaria che lavorativa. Tra questi dispositivi il più diffuso è sicuramente lo smartphone che grazie ad algoritmi di machine learning può apprendere le abitudini degli utenti e predire i comportamenti di questi o utilizzare app facenti uso dell’intelligenza artificiale per assolvere ai problemi più disparati.

È in questo scenario che il presente elaborato si pone l’obiettivo di creare la rete neurale (nello specifico Convolutional Neural Network) da implementare all’interno di un app medica per lo screening e la diagnosi di patologie orali semplicemente analizzando la foto del cavo orale scattata da uno smartphone. In questo lavoro di tesi verranno descritte le tecniche di costruzione della rete il suo miglioramento e la relativa implementazione nel framework utilizzato per la costruzione dell’app. Per quanto riguarda lo sviluppo frontend e backend dell’applicazione, si rimanda al lavoro di tesi del dottor Alfio Aurelio D’urso [React Native per la realizzazione di un’applicazione per lo screening e la diagnosi di patologie orali]

Capitolo 1

Machine Learning e Deep Learning

Prima di descrivere il deep learning e illustrarne le caratteristiche più importanti bisogna introdurre al lettore definizioni e concetti base sul machine learning, branca principale di cui fa parte l’apprendimento approfondito. Il primo capitolo della tesi tratterà questo tema. Nella prima sezione si darà una definizione principale di apprendimento per i modelli computazionali, la seconda fornirà i dettagli sull’ approccio supervisionato che è stato utilizzato nel presente lavoro di tesi; successivamente verranno introdotte le reti neurali e le nozioni sulla metodologia che permette di addestrarle. I successivi paragrafi tratteranno la dipendenza dell’efficienza dei risultati dal numero di dati trattati dal modello e la descrizione del compito per cui sono state addestrate le reti: la classificazione.

* 1. Definizione di apprendimento per una macchina

Il machine learning è una branca della computer science che studia i sistemi e gli algoritmi che possono imparare dai dati, sintetizzando da essi nuova conoscenza. Questa branca è fondamentale nello studio e nello sviluppo delle Intelligenze Artificiali; un sistema basato sull’apprendimento automatico può migliorare la propria conoscenza del sistema da studiare dall’osservazione dei dati di input per poi fornire output più vicini a quelli desiderati.

Bisogna però dare una definizione più precisa di apprendimento; citiamo a tal proposito la definizione di machine learning data dal prof. T.M. Mitchell:1 :

*"Un programma apprende da una certa esperienza E se nel rispetto di una classe di compiti T, con una misura della prestazione P, la prestazione P misurata nello svolgere il compito T è migliorata dall’esperienza E”* [Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. Machine learning: An artificial intelligence approach. Springer Science & Business Media, 2013.*].”*

I Compiti del machine learning sono spesso descritti in termini di come il sistema possa trattare un esempio, una collezione di feature, o caratteristiche, quantitativamente misurate da alcuni oggetti o eventi che vogliamo il sistema elabori. Tipicamente l’input viene rappresentato da un vettore x Ꞓ dove ogni rappresenta una feature. Per esempio le feature di una immagine possono essere i valori dei pixel. Molti compiti possono essere appresi attraverso l’utilizzo delle tecniche di machine learning. Alcuni dei più importanti sono la regressione, trascrizione, traduzione, sintesi, stima di densità di probabilità, denoising e classificazione. Quest’ultimo assume un ruolo importante per il presente lavoro di tesi.

Gli algoritmi di machine learning possono essere categorizzati in due tipi : algoritmi supervisionati e algoritmi non supervisionati dal tipo di raccolta dati a cui sono sottoposti durante il processo di apprendimento.

* **Apprendimento Supervisionato:** “supervisionato” perché la presenza delle soluzioni, denominate anche “etichette”, è fornita nell’insieme dei dati di addestramento dal programmatore, che pertanto prende il nome di “supervisore”. Il supervisore fornisce all’algoritmo degli esempi di cui sono indicate le variabili di input con la previsione corretta e, attraverso tali esempi, l’algoritmo elabora un modello predittivo.[ A. Minini, L’apprendimento supervisionato, URL: <http://www.andreaminini.com/ai/machine-learning/apprendimentosupervisionato>.]
* **Apprendimento Non Supervisionato:** in questo caso le etichette non vengono fornite: sarà l’algoritmo, per come è strutturato, a trovare una logica di classificazione. Esempi pratici si hanno negli attuali motori di ricerca, i quali, data una parola chiave, sono in grado di creare una lista di link rimandanti alle pagine che l’algoritmo di ricerca ritiene attinenti a quella effettuata [Apprendimento non supervisionato, URL: https://it.wikibooks.org/wiki/Intelligenza\_artificiale/Apprendimento \_non\_supervisionato.].
  1. Generalità sulle tecniche di Deep Learning

Il deep learning, o apprendimento in profondità, è un’area del machine learning che si basa su un apprendimento automatico a più livelli dai dati di input. Ogni livello più profondo prende come input i dati di output del livello precedente estraendo sempre più informazioni con l’aumentare della profondità. Questo approccio permette ai modelli computazionali di apprendere rappresentazioni di dati con più livelli di astrazione. Questa intuizione sui livelli di apprendimento dà il nome all’intero ambito e si ispira al modo in cui il cervello dei mammiferi elabora le informazioni ed impara, rispondendo agli stimoli esterni. Ogni livello seguendo questo parallelismo, corrisponde ad una delle diverse aree che compongono la corteccia cerebrale. Applicando il Deep Learning, si ottiene quindi una "macchina" che riesce autonomamente a classificare i dati ed a strutturarli gerarchicamente, trovando quelli più rilevanti e utili alla risoluzione di un problema, migliorando le proprie prestazioni con l'apprendimento continuo.

Altre motivazioni che spingono ad usare il deep learning sono legate ai vantaggi dell’impiego di architetture gerarchiche multi-strato. Un’organizzazione gerarchica dei dati permettere di condividere e riusare informazioni estratte durante l’elaborazione e di selezionare e scartare dettagli inutili lungo una gerarchia. Rispetto ad una architettura semplice a tre strati (input-strato con unità nascoste-output), una architettura multi-strato permette di distribuire meglio un grande numero di nodi su più strati riducendo il costo computazionale elevato se fossero tutti localizzati su un solo strato e attenuando l’ingente utilizzo di memoria che comporterebbe una struttura meno profonda.

Capitolo 2

Reti Neurali

2.1 Neurone Biologico

Il modo migliore per capire il funzionamento di una rete neurale è capire come funziona il neurone biologico.

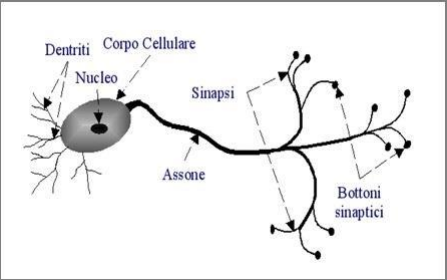


Figura 1. Rappresentazione schematica di un neurone biologico [tesi polito]

I principali componenti di un neurone biologico sono :

* **Soma**: parte centrale del neurone;
* **Assone**: un filamento che fuoriesce dal soma;
* **Sinapsi**: “ramificazione” dell’assone , ciascuna sinapsi termina con un bottone sinaptico;
* **Dendriti**: filamenti che fuoriescono dal soma.

Il neurone è capace di ricevere segnali attraverso i propri dendriti, li elabora nel soma e successivamente trasmette il segnale, tramite l’assone, al neurone successivo. L’assone non è direttamente collegato ai dendriti di altri neuroni: il punto in cui il segnale viene trasmesso da una cellula ad un’altra è un piccolo spazio denominato “fessura sinaptica”. Quando un segnale è nei pressi di una sinapsi, questa rilascia un quantitativo di sostanze chimiche chiamate “neurotrasmettitori”, i quali determinano la conduttività di una sinapsi, ovvero quanto la sinapsi attenua o enfatizza il segnale elettrico dall’assone. Nella trasmissione di un segnale, le correnti si possono sommare in spazio e tempo e se tale somma oltrepassa un certa soglia, un impulso di una certa entità e durata, denominato “potenziale di azione” , è generato. Il segnale cosi prosegue per il prossimo assone, ricominciando il processo.

* 1. Neurone Artificiale

Le reti neurali si basano sulla simulazione di neuroni artificiali opportunamente collegati, i quali ricevono in ingresso degli stimoli elaborandoli di conseguenza. Il primo modello di neurone venne introdotto da Warren McCulloch e Walter pitts nel 1943.

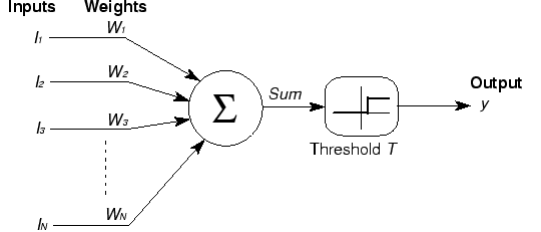


Figura 2 rappresentazione del modello McCulloch e Pitts [tesi polito]

L’elaborazione prevede, nei casi più semplici, che i singoli ingressi vengano moltiplicati per un opportuno valore, detto “peso”, e il risultato delle moltiplicazioni venga poi sommato; se tale somma supera una certa soglia, il neurone attiva la propria uscita. La distribuzione dei valori dei pesi varia in base all’importanza dell’ingresso: un ingresso importante avrà un peso elevato, a differenza di uno meno importante che avrà un valore inferiore. Tuttavia, tale modello non si rilevò molto pratico, in quanto, per avere i valori desiderati, bisognava impostare manualmente pesi e connessioni.

* 1. Percettrone

Alla fine degli anni Cinquanta, Frank Rosenblatt introdusse una rete composta di unità di perfezionate dal modello McCulloch-Pitts, il *Percettrone*, ovvero l’unione del concetto del modello precedente con la *regola di Hebbian* [tesi polito 9 ]per l’adattamento dei pesi. Inoltre, il modello percettrone aggiungeva un ulteriore valore di input che rappresenta il *bias.*

Le principali differenze con il modello McCulloch-Pitts sono:

1. I pesi e le soglie non sono tutti identici;
2. I pesi possono assumere valori positivi e negativi;
3. Non esiste una sinapsi inibitoria assoluta;
4. Vi è una regola d’apprendimento

Immagine che contiene testo, orologio

Descrizione generata automaticamente

(1.1)

Se consideriamo w e x come vettori e b (bias) come l’opposto della soglia possiamo riscrivere l’equazione nel seguente modo:

Immagine che contiene testo

Descrizione generata automaticamente

(1.2)

Dove a è detta “funzione di attivazione”. Il bias può essere interpretato come una soglia che influenza ampiamente l’output dell’unità. La novità principale del modello di Rosenblatt è la sua capacità di modificare i propri pesi adattandoli al problema dato in modo tale da non dover creare un circuito a mano [Wikibooks, Hebbian Learning, URL: https://en.wikibooks.org/wiki/Artificial\_Neural\_Networks/Hebbian\_ Learning .]

Tuttavia nel 1969, Marvin Minsky e Symour A. Papert, nel loro libro *“Perceptrons: an introducion to computational geometry”* [11 polito], misero in luce tutti i limiti delle reti a due strati basate sui percettroni e come queste non fossero capaci di poter risolvere problemi se non quelli caratterizzati da separabilità lineare delle soluzioni, dando cosi inizio ad un periodo che prese il nome di “Inverno della IA”.

Al fine di risolvere problemi più complessi, si cominciò a interconnettere gli input dei neuroni artificiali con gli output di altri neuroni artificiali, creando una rete neurale a più livelli, ovvero il **percettrone multistrato**

* 1. Percettrone Multistrato

Il percettrone multistrato (MLP) è una vera e propria rete neurale artificiale composta, come si evince dal nome, da più percettroni. Esso si compone di un livello di input, il quale riceve il segnale, e un livello di output, che esegue una previsione o prende una decisione per quanto concerne l’input e, tra questi due livelli, vi è un numero arbitrario di strati “nascosti”, il vero motore computazionale della rete. Ciascun neurone di un livello è connesso a tutti i neuroni del livello precedente, per tale motivo una rete di questo tipo è anche detta **Fully Connected**

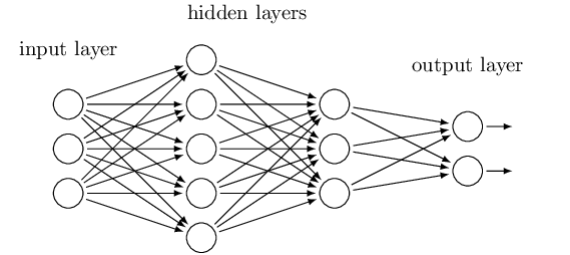


Figura 3 Esempio di MLP [polito]

* 1. Funzione di Attivazione

La funzione di attivazione di un neurone rappresenta un ulteriore passo di elaborazione, applicata prima che il risultato sia inviato ai neuroni successivi; si tratta di funzioni non lineari che applicano una precisa manipolazione matematica ai dati in input. Al giorno d’oggi le reti neurali propongono vari tipi di funzioni di attivazione.

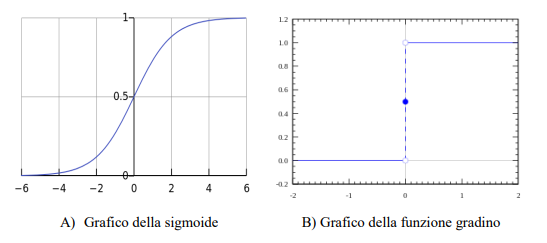
* + 1. Sigmoide

La differenza sostanziale di tale funzione rispetto alla formula (1.2) è la sua natura non lineare da cui si ottengono combinazioni non lineari.

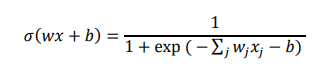


(1.3)

Di fatto, l’obiettivo del sigmoide è quello di ridurre gli effetti di piccole variazioni e di bilanciarli sull’output finale.



La nuova funzione di attivazione diverrà dunque:



(1.4)

La sigmoide può essere vista come una versione smussata della funzione di attivazione del percettrone; tuttavia, anche se è una delle funzioni più usate, non è esente da problemi: si noti come nel grafico A), verso la fine della funzione, i valori delle coordinate tendano a rispondere molto meno rispetto alle ascisse. Questo fenomeno solleva il problema della scomparsa del gradiente, ovvero il gradiente ha assunto un valore talmente basso, quasi è scomparso, che la rete rifiuta di apprendere ulteriormente.

* + 1. Tangente Iperbolica

La funzione di tangente iperbolica (tanh) è buona alternativa alla sigmoide. La sua natura è sempre non lineare ma il suo gradiente è molto più resistente della sigmoide e decidere tra le due dipenderà dalle richieste di robustezza del gradiente stesso, tuttavia anche tale funzione non è esente dal problema della scomparsa del gradiente.

La sua equazione è:



(1.5)

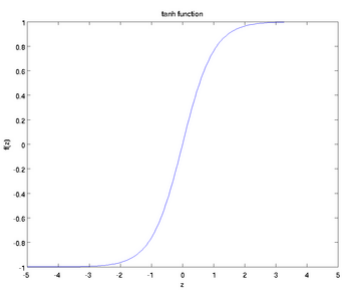


Figura 4. Grafico della funzione tangente iperbolica

* + 1. SoftMax

Funzione di attivazione usata il più delle volte, nell’output layer di una rete neurale, in particolare, nei problemi di classificazione [Wikipedia, Funzione SoftMax, URL: <https://it.wikipedia.org/wiki/Funzione_softmax>.]. Tale funzione accetta in input un vettore K di numeri reali normalizzando in una distribuzione di probabilità composta da K valori di probabilità sugli esponenziali dei valori in input. Ciò garantisce che, dopo averla applicata, i valori saranno sempre compresi nell’intervallo (0,1).



(1.6)

In pratica, si applica la funzione esponenziale per ogni elemento del vettore input z e ogni valore viene normalizzato dividendolo per la somma di tutti gli esponenziali garantendo che la somma dei singoli componenti nel vettore di output sia 1.

* + 1. ReLU

Di questa funzione di attivazione parleremo in seguito nel paragrafo. Per ora si sappia che è una funzione di attivazione dal carattere non lineare.

* 1. Addestramento di una rete neurale

Nel 1986 David E. Rumelhart, G. Hinton e R.J.Williams proposero il più conosciuto e utilizzato algoritmo per l’addestramento di una rete neurale: l’algoritmo della *retropropagazione dell’errore* (*Error* *Backpropagation*) [D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors, Nature, 1986]. Questo algoritmo consiste in una tecnica d’apprendimento tramite esempi, costituente una generalizzazione dell’algoritmo d’apprendimento del percettrone di Rosenblatt. L’algoritmo si basa sulla modifica sistematica dei pesi delle connessioni tra neuroni cosicché l’output della rete coincida sempre di più con il risultato aspettato. Distinguiamo due fasi principali nell’addestramento della rete: *forward propagation* e *backward propagation.*

Nella forward propagation i pesi assumono dei valori fissi (saranno dei valori di default alla prima iterazione) e vengono calcolate tutte le attivazioni dei neuroni della rete, dal primo layer proseguendo fino all’ultimo. Nella backward propagation, il risultato generato dalla rete viene confrontato con quello desiderato e se ne calcola l’errore. L’errore viene così propagato nel senso inverso a quello delle sinapsi, con l’intento di minimizzarlo, modificando i pesi di conseguenza. Alla fine di questa fase comincia una nuova iterazione con la forward propagation.

* + 1. Forward Propagation

Definiamo Forward Propagation il processo di calcolo dell’output di una rete dati i suoi input. Consideriamo un problema di classificazione binaria utilizzando una rete composta da un singolo layer nascosto, un output layer con un solo neurone e la sigmoide come funzione di attivazione [V. Iuhaniwal, Forward propagation in neural networks, URL 16 polito]

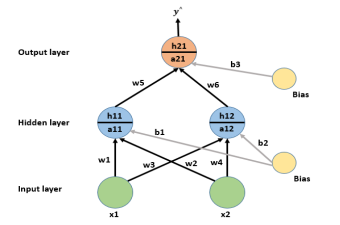


Figura 5. Rete neurale d’esempio

Durante la forward propagation, ad ogni nodo del layer nascosto dell’output layer avvengono le azioni di pre-attivazione e attivazione. Osserviamo la figura 5: ad esempio, nel primo livello vengono calcolati i valori di pre-attivazione i quali vengono passati alla relativa funzione di attivazione ottenendo . Lo stesso discorso viene fatto sul nodo dell’output layer cosi il valore di output predetto .

Definendo come i valori di output stimati e come i valori di output stimati e come le etichette reali, possiamo calcolare l’**errore** (o **perdita**) come:

Immagine che contiene testo, orologio

Descrizione generata automaticamente

(1.7)

Definiamo “**Training**” la fase di ricerca dei pesi che minimizzino l’errore quadratico dell’equazione (1.7). Data una funzione di attivazione *f* che sia differenziabile, la minimizzazione viene effettuata utilizzando il metodo della *“Discesa del gradiente”,* tale fase è detta **Back Propagation**.

* + 1. Back Propagation

Siano i valori di input, le etichette, i valori di output e l’i-esimo peso, si calcolano le derivate parziali dell’errore rispetto alle singole componenti:

Immagine che contiene testo, orologio, calibro

Descrizione generata automaticamente

(1.8)

Il gradiente è l’insieme delle derivate rispetto a tutte le n componenti:

Immagine che contiene testo

Descrizione generata automaticamente

(1.9)

A questo punto si ottiene la “*training* *rule*” che definisce come cambiare i pesi delle connessioni con lo scopo di minimizzare l’errore.



(1.10)

Dove µ è definito “*learning* *rate*”, valore positivo ma molto piccolo, in modo da non eliminare le corrette classificazioni, e specifica il grado di apprendimento dei parametri. A questo punto i pesi vengono aggiornati con al seguente formula:

Immagine che contiene testo

Descrizione generata automaticamente

Conosciuta, più comunemente, come **discesa del gradiente**.

L’applicazione di tale algoritmo a tutti i pesi delle sinapsi è definito **Retropropagazione dell’errore** e ha molteplici vantaggi:

* Esegue la discesa del gradiente a tutti i pesi dell’intera rete;
* È semplice da generalizzare per grafi diretti arbitrari;
* Riesce a trovare un minimo locale, non necessariamente uno globale.

Tuttavia:

* Il training potrebbe richiedere migliaia di operazioni

Capitolo 3

Reti Neurali Convoluzionali

Le *Reti Neurali Convoluzionali* (CNN) nascono nel 1990 dalla ricerca di Yann Le Cunn insieme al suo team basandosi sul funzionamento della corteccia visiva del cervello umano[9 tesi cnn veicoli]. Grazie alle ottime prestazioni che si sono riuscite a ricavare soprattutto in ambito del riconoscimento di immagini, ancora oggi le CNN sono considerate lo “stato dell’arte” per quanto riguarda il riconoscimento di pattern ed immagini. Le reti neurali convoluzionali sono molto simili alle reti artificiali di cui si è parlato nel capitolo precedente: sono composte da neuroni e hanno pesi e bias da apprendere. Allora una CNN può essere definita come una rete neurale che usa la convoluzione al posto del generale prodotto matriciale in almeno uno dei suoi strati.

La convoluzione è una operazione su due funzioni a valori reali; date queste due funzioni x e w, il loro prodotto di convoluzione sarà:



(2.1)

Nelle reti di convoluzione spesso la funzione x si riferisce all’input e la funzione w al kernel, che può essere visto come una funzione di peso relativa ai dati di input. Più in generale, nelle applicazioni l’input è un vettore multidimensionale dei dati e il kernel è un altro array multidimensionale di parametri che vengono adattati dall’algoritmo di apprendimento in maniera appropriata.

Per utilità pratiche conviene definire nello specifico l’operazione di *convoluzione discreta*, un prodotto di convoluzione che invece di essere implementato su un integrale esteso all’infinito è implementato su una sommatoria su un numero finito di indici riferiti agli elementi dei vettori. Per esempio, avendo come input una immagine bidimensionale *I* , si potrà usare un kernel bidimensionale W:

(2.2)



3.1 Architettura di una CNN

Nelle CNN distinguiamo tre tipi di livelli (layer): *Convolutional Layer, Pooling Layer e Fully-Connected Layer.* Le spiegazioni e le figure dei seguenti sotto-paragrafi sono tratti da un articolo dell’università di Standford [17 polito].

3.1.1 Convolutional Layer

Questi è il blocco portante di una CNN, in cui avvengono la maggior parte delle operazioni di computazione più pesanti. I suoi parametri consistono in un insieme di filtri da apprendere, ognuno molto piccolo dal punto di vista spaziale, per quanto riguarda larghezza e profondità. Un tipico esempio di filtro di un primo livello di una CNN potrebbe avere dimensioni 5x5x3 (i.e. il numero 5 sta ad indicare il numero di pixel per altezza e larghezza, mentre il numero 3 indica la profondità, poiché un’immagine ha 3 canali, RGB, per quanto riguarda i colori). Durante la forward propagation si convolve ciascun filtro lungo la larghezza e l’altezza del volume di input, producendo una *activation map* (o *feature map*) bidimensionale per quel filtro. La rete apprenderà, intuitivamente, i filtri che causano l’attivazione 15 dell’uscita del neurone quando osservano un certo tipo di tratto visuale, ad esempio una macchia di un certo colore sul primo layer. Seguendo questa procedura si finirà per ottenere un intero insieme di filtri per ogni Convolutional Layer, di cui ognuno produrrà una activation map bidimensionale, le quali verranno unite lungo la dimensione della profondità producendo il volume di output.

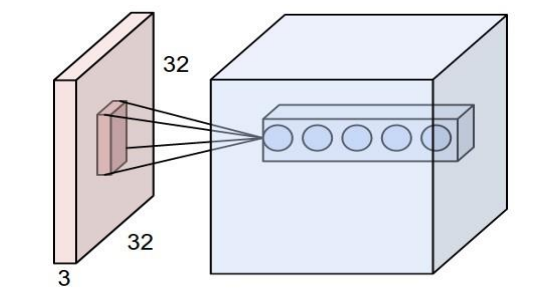


Figura 6. Esempio di convoluzione su immagine [32x32x3]

Facciamo un esempio pratico: supponiamo di avere un volume di input pari a [32x32x3] e la dimensione del filtro pari a [5x5]. Dunque, ogni neurone del Convolutional Layer avrà dei pesi per una regione del volume di input pari a [5x5x3] per un totale di75 pesi (+1 per il bias).

3.1.2 Organizzazione dello spazio

Quattro iperparametri controllano la dimensione del volume di output :

1. La **profondità** (*depth*) del volume di output: corrisponde al numero di filtri che vogliamo usare, ognuno che apprende a osservare una particolarità differente dell’input. Definiamo l’insieme di neuroni che osservano la stessa regione dell’input come *depth column*;
2. Il **passo** (*stride*) con cui il filtro trasla. Se il passo ha il valore 1 allora io filtro trasla di un pixel alla volta. All’aumentare del passo l’output avrà una dimensione spazialmente inferiore;
3. Delle volte conviene inserire degli zeri lungo il bordo dell’input, ciò che viene definito come *zero-padding.* Il valore dello zero-padding permette di avere sotto controllo la dimensione spaziale del volume di output;
4. Numero di filtri (*kernel*) da applicare.

Possiamo determinare la dimensione spaziale del volume di output come una funzione del valore del volume di input (**W**), la dimensione del filtro (**F**) dei neuroni nel Convolutional Layer, il passo che è stato applicato (**S**) e infine il valore di zero-padding usato sul bordo (**P**). la formula per calcolare quanti neuroni sono necessari è:

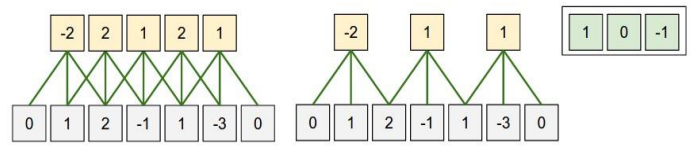


Figura 7. Illustrazione di organizzazione spaziale

Nell’esempio in figura 7 vi è una sola dimensione spaziale (asse x), un neurone con dimensione del filtro F = 3, dimensione di input W = 5 e vi è zero-padding P = 1. Guardando la figura da sinistra, nella prima immagine il neurone ha un passo S = 1, dando come output (5-3+2) /1 + 1 = 5. Nell’immagine seguente, il neurone ha un passo S = 2, restituendo come output (5-3+2) /2 + 1 = 3. Da notare come il passo S non possa assumere valore 3 in quanto non sarebbe compatibile con il volume. I pesi di questo neurone sono [-1,0,1] con bias nullo, e saranno condivisi attraverso i neuroni gialli.

3.1.3 Condivisione dei parametri

La condivisione dei parametri è usata nei Convolutional Layer per controllarne il numero. Prendiamo come esempio l’architettura che ha vinto la sfida di ImageNet, nel 2012, proposta da Krizhevsky A. et al., la AlexNet [18 polito]: vi sono 55x55x96 = 290.400 neuroni nel primo Convolutional Layer, ognuno dei quali ha 11x11x3 =363 pesi e 1 bias. Ciò porta a 290.400 \* 364 = 105.705.600 parametri solo nel primo livello.

A quanto pare, però, si può ridurre drasticamente il numero di parametri con una sola ragionevole assunzione: se una feature è utile da calcolare in una precisa posizione () allora deve essere utile da calcolare anche in una posizione () differente.

In particolare, data una sezione bidimensionale lungo l’asse di profondità del volume di output nota come depth slice (nell’esempio precedente un volume di [55x55x96] ha 96 depth slice, ognuno di dimensione [55x55]), i neuroni di ogni sezione saranno vincolati ad usare gli stessi pesi e bias. Riprendendo l’esempio, avremo solo 96 set di pesi (uno per ogni sezione) per un totale di 96\*11\*11\*3 = 34.848 pesi univoci, o 34.944 parametri (+96 bias). Durante la fase di back propagation, ogni neurone calcolerà il gradiente dei suoi pesi, ma tali gradienti saranno aggiunti attraverso i vari depth slice aggiornando un singolo insieme di pesi per ciascun depth slice.

Da notare che se tutti i neuroni in un singolo depth slice usano lo stesso vettore dei pesi, allora il passo di forward propagation del Convolutional Layer, in ogni depth slice, può essere calcolato come convoluzione dei pesi dei neuroni con il volume di input, da qui il nome Convolutional Layer. È per questo motivo che si è solito riferirsi agli insiemi dei pesi come un filtro (o kernel), il quale è convoluto con l’input.

3.1.4 ReLU layer

ReLU sta per “ Rectified Linear Unit” ed è un tipo di funzione di attivazione. Dal punto di vista matematico si definisce come e visivamente ha il seguente aspetto:

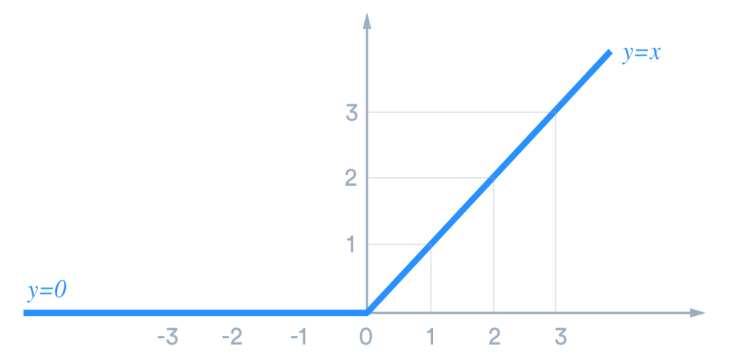


Figura 8. Diagramma della funzione ReLU [19 polito]

Essa è la funzione di attivazione più usata nelle reti neurali, specialmente nelle CNN, e assume un carattere lineare, per tutti i valori positivi, e nullo, per tutti quelli negativi. Ha molteplici vantaggi:

1. Poco costosa dal punto di vista del calcolo in quanto non vi è una logica matematica complicata dietro;
2. Converge più velocemente in quanto la linearità garantisce che la pendenza non saturi al crescere di x;
3. Non soffre il problema della scomparsa del gradiente come per la sigmoide o la tangente iperbolica;
4. Poiché la ReLU è nulla per gli input negativi, non ci sarà alcuna attivazione per tali tipi di input, comportamento spesso ricercato.

I ReLU layer si trovano dopo i Convolutional Layer e hanno la funzionalità di aumentare la proprietà di non linearità della funzione di attivazione senza modificare la dimensione del filtro.

3.1.5 Pooling Layer

È solito trovarsi, tra un *Convolutional Layer* e un altro, un layer intermedio chiamato *Pooling Layer*. La sua funzione principale è quella di ridurre progressivamente la dimensione spaziale della rappresentazione in modo tale da avere un numero inferiore di parametri e abbattere il costo di computazione della rete controllando il fenomeno del sovradattamento (*overfitting*)

Immagine che contiene testo, orologio

Descrizione generata automaticamente

Figura 9. Esempio di operazione di pooling

Il *Pooling Layer* opera separatamente e indipendentemente su ogni *depth slice* del proprio input applicando un algoritmo di selezione dei parametri riducendo la dimensione spaziale. L’algoritmo più comune è quello del *MAX pooling .*

Immagine che contiene tavolo

Descrizione generata automaticamente

Figura 10. Operazione di MAX Pooling con filtro 2x2 e passo 2 su un depth slice

Prendendo in esame la figura 10, notiamo come viene utilizzato un filtro di dimensione 2x2 con passo 2: viene sempre preso il valore massimo, considerato più importante ai fini dell’analisi, scartando quelli più bassi, diminuendo drasticamente la dimensione iniziale.

Vi è un secondo algoritmo che è quello dell’*Average Pooling*, ovvero a volte è conveniente, anziché estrarre il massimo dei valori considerati, prenderne il valor medio; questa tecnica è molto usata nelle recenti architetture in presenza dei *Fully Connected layer*.

Più in generale un Pooling Layer:

1. Accetta un volume di dimensione ;
2. Richiede due iperparametri:
   1. L’estensione spaziale F;
   2. Il passo S.
3. Produce un volume di dimensione ;
   1. ;
   2. ;
   3. ;
4. Non introduce ulteriori parametri poiché calcola una funzione fissa dell’input;

3.1.6 Fully connected Layer

Nel *Fully Connected layer*, i neuroni sono tutti interconnessi alle funzioni di attivazione del layer precedente, come visto nelle classiche reti neurali. L’output finale di *un Fully Connected layer* sarà un vettore di dimensione 1 x 1 x K, dove K rappresenta il numero di neuroni di cui è composto i layer, contente le attivazioni calcolate. È facile notare come, avendo un input organizzato in tre dimensioni e ottenendo un singolo vettore in output, non vi sia più la possibilità di applicare un altro *Convolutional Layer* a seguire. Infatti, la sua funzione principale è quella di raggruppare tutte le informazioni ottenute fino a quel momento, esprimendole con un singolo numero utile per la classificazione finale.

Possono essere presenti più *Fully Connected layer*, di cui l’ultimo avrà il parametro K pari al numero delle classi presenti nel dataset su cui sta lavorando la rete. I valori finali verranno infine passati all*’output layer* che avrà il compito di effettuare la classificazione utilizzando un’apposita funzione probabilistica.

Capitolo 4

Il progetto

In questo capitolo verrà mostrato in dettaglio la realizzazione del progetto finale ossia un’applicazione per lo screening e la diagnosi delle patologie del cavo orale.

Nel presente elaborato verrà solamente descritta la realizzazione del modulo di deep learning per l’analisi delle foto e la sua implementazione all’interno dell’app. Per quanto riguarda lo sviluppo frontend e backend, come già detto nell’introduzione, si rimanda al lavoro di tesi del dottor Alfio Aurelio D’urso [React Native per la realizzazione di un’applicazione per lo screening e la diagnosi di patologie orali]. Al lettore basterà sapere che per lo sviluppo dell’applicazione si è optato per il framework ***React Native****,* che permette lo sviluppo combinato di applicazioni native per sistemi *Android* e *iOS*, facendo al contempo uso di un linguaggio duttile e non tipizzato quale ***Javascript*** . il tutto è stato sviluppato sotto la piattaforma ***Expo*** che, tra le altre cose, permette di visualizzare direttamente sullo smartphone la propria applicazione durante lo sviluppo, fornisce tools per valutarne le prestazioni e consente di distribuirla in modo diretto.

La struttura di base dell’applicazione è molto semplice: essa deve permettere di scattare una foto (o di selezionarla dalla galleria del proprio smartphone) e di analizzarla tramite il modulo di deep learning. Una volta analizzata la foto e restituito il risultato, se l’utente non è soddisfatto o vuole approfondire, ha la possibilità di inviare la foto ad un centro medico, il quale si preoccuperà di ricontattare l’utente.

Immagine che contiene testo

Descrizione generata automaticamente Immagine che contiene testo

Descrizione generata automaticamente

Figura 13. Sezione per l’invio al centro medico

Figura 12. Schermata risultato dell’analisi

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Figura 11. Schermata home

* 1. Framework e librerie

Normalmente, sono tre i principali linguaggi di programmazione utilizzabili per implementare algoritmi di *Machine Learning*: *Python*, *C++, Matlab*. Tutti questi linguaggi permettono, attraverso l’uso apposito di librerie o plug-in, di effettuare i calcoli di interesse per le *CNN.*

* + 1. Python

È stato scelto Python come linguaggio di programmazione per un motivo: la semplicità. Python è un linguaggio di programmazione ad oggetti con la possibilità di creare classi, esattamente come in C++, tuttavia è nato con un’impostazione user-friendly che lo rende molto didattico. La libreria più interessante, appartenente sia a Python che a C++, si chiama ***Tensorflow***, una libreria che permette di velocizzare significativamente i calcoli necessari ad un algoritmo di Machine Learning, contiene una serie di funzioni che sono in grado di sfruttare i Tensori[11 tesi veicoli]. Essa possiede una API chiamata ***Keras*** molto utilizzata per la creazione di modelli CNN con poche righe di codice.

* 1. Dataset

L’obiettivo del progetto è far riconoscere al dispositivo delle patologie del cavo orale per poi poterle classificare. Per poter allenare una rete neurale artificiale, si hanno bisogno di dati da elaborare più e più volte. Uno degli elementi fondamentali per il machine learning è la disponibilità di un dataset cioè delle collezioni di dati (nel nostro caso immagini) adeguati da utilizzare per le fasi di training e testing, oltre che di valutazione finale del modello.

Per lo sviluppo del progetto è stato utilizzato un dataset reso disponibile dalla professoressa Rosalia Maria Leonardi ( professoressa ordinaria di malattia odontostomatologiche afferente al dipartimento di Chirurgia generale e specialità medico-chirurgiche dell’Università degli studi di Catania) e dal dottor Gaetano Isola (Ricercatore di Malattie odontostomatologiche dell’università degli studi di Catania)

Il dataset si compone di:

* + - **Carcinoma orale**: 69 immagini;
    - **Leucoplachia**: 96 immagini;
    - **Lichen Planus**: 71 immagini;
    - **Paziente Sano**: 44 immagini.

* + 1. Problematiche del dataset

Per riuscire ad avere una rete neurale performante bisognerebbe essere in possesso di dataset dettagliati e in particolare, per ogni categoria presente nel dataset, dovranno essere presenti un ampio numero di immagini, cercando di non avere un numero di elementi troppo diverso fra tutte le classi, al fine di evitare situazioni di sbilanciamento.  
Inoltre per ciascuna categoria del dataset, le immagini raccolte dovranno essere suddivise in un *training set* ed un *validation set*, che contiene una quantità ridotta di immagini. È importante tuttavia fare attenzione al fatto che le immagini presenti nel *validation set*, non siano presenti anche nel training set, in modo da non rendere falso il valore di accuratezza.

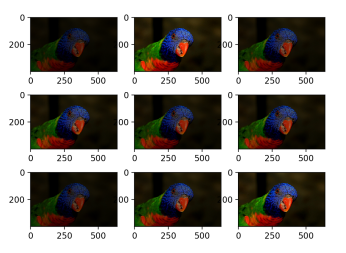
Fatta questa premessa risulta evidente come il dataset in nostro possesso sia composto da un numero di samples relativamente basso, ragion per cui, al fine di evitare tali problematiche verranno adoperate tecniche di deep learning come il “*Transfer Learning”*  e di ampliamento del dataset *“Data Augmentation”.*

* + 1. Data Augmentation

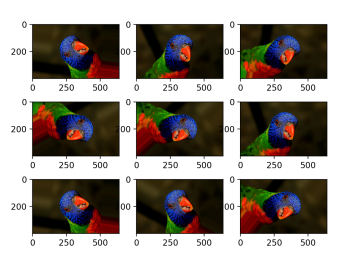
Le reti convoluzionali hanno bisogno di un gran numero di dati in ingresso durante il training per dare un’accuratezza maggiore. Poche immagini causano una più frequente analisi delle stesse durante il training, causando come è noto overfitting. Indicativamente una rete convoluzionale inizia ad avere un accuratezza accettabile a partire da 1000 immagini usate per il training. Nei casi in cui l’acquisizione di un grande dataset di training risulti complicato, esistono tecniche che permettono di estrarre variazioni mediante trasformazioni artificiali delle immagini.

Le caratteristiche visive di un oggetto in un immagine sono diverse: luminosità, messa a fuoco, rotazione (angolo), distanza dal punto di vista, sfondo, forma e colore. Esistono diverse trasformazioni applicabili alle immagini:

* + **Flip (capovolgimento) e rotazione**: il flip può essere sia orizzontale che verticale. La scelta di uno o entrambi dipende dalla caratteristica dell’oggetto. Similmente, la scelta di ruotare un oggetto o meno in fase di data augmentation è dettata da come si suppone esso sia disposto in testing. In figura 4.4a sono mostrati esempi di rotazioni di un immagine.
  + **Crop (ritagli) e ridimensionamento**: il ritaglio casuale di pezzi più piccoli a partire da un immagine consente alla rete di adattarsi al caso in cui sia richiesta la classificazione di un oggetto di cui si è acquisita solo una parte.
  + **Luminosità e contrasto**: le condizioni di luminosità possono influire anche in modo importante nel riconoscimento di un oggetto. Pertanto, oltre che acquisire immagini sul campo con diverse condizioni di luce, è possibile agire modificando in modo casuale luminosità e contrasto dell’immagine. In figura 2.18b è mostrato un esempio di immagini generate con luminosità differenti.
  + **Distorsione**: proprietà che potrebbe essere interessante considerare per oggetti sottoposti a distorsione di vario tipo (stretching) o per oggetti acquisiti da prospettive leggermente differenti.



(b) Variazioni casuali di luminosità



1. Variazioni casuali di rotazione

Figura 4.4

Per poter aumentare il numero di campioni in maniera artificiosa. La classe *ImageDataGenerator* appartenente a *Keras* permette, tramite rotazioni, tagli, zoom, ecc., di modificare le immagini presenti nel Dataset e utilizzarne anche le versioni modificate per incrementare il numero di test effettuati: non risulta molto efficace con dataset già grandi ma aiuta molto con i più piccoli come quello preso in esame