Immagine che contiene oggetto

Descrizione generata automaticamente

Università

degli Studi

di Catania

Dipartimento di Ingegneria Elettrica Elettronica Informatica

**Corso di Laurea in Ingegneria Informatica**

**Classificazione di patologie del cavo orale con architettura CNN**

Anno accademico [2020-2021]

Candidato: Alessio Calini

Relatore: Prof. Daniela Giordano

Correlatore:

INDICE

[INTRODUZIONE 1](#_Toc95813166)

[1. MACHINE LEARNING E DEEP LEARNING 3](#_Toc95813167)

[*1.1.* Definizione di apprendimento per una macchina 3](#_Toc95813168)

[*1.2.* Generalità sulle tecniche di Deep Learning 5](#_Toc95813169)

[2. RETI NEURALI 6](#_Toc95813170)

[*2.1.* Neurone Biologico 6](#_Toc95813171)

[*2.2.* Neurone Artificiale 7](#_Toc95813172)

[*2.3.* Percettrone 8](#_Toc95813173)

[*2.4.* Percettrone Multistrato 10](#_Toc95813174)

[*2.5.* Funzione di attivazione 10](#_Toc95813175)

[2.5.1. Sigmoide 11](#_Toc95813176)

[2.5.2. Tangente iperbolica 12](#_Toc95813177)

[2.5.3. SoftMax 12](#_Toc95813178)

[2.5.4. ReLU 13](#_Toc95813179)

[*2.6.* Addestramento di una rete neurale 13](#_Toc95813180)

[2.6.1. Forward Propagation 14](#_Toc95813181)

[2.6.2. Back Propagation 15](#_Toc95813182)

[3. Reti neurali Convoluzionali 17](#_Toc95813183)

[*3.1.* Architettura di una CNN 18](#_Toc95813184)

[3.1.1 Convolutional Layer 18](#_Toc95813185)

[3.1.2 Organizzazione dello spazio 19](#_Toc95813186)

[3.1.3 Condivisione dei parametri 20](#_Toc95813187)

[3.1.4 ReLU Layer 22](#_Toc95813188)

[3.1.5 Pooling Layer 23](#_Toc95813189)

[3.1.6 Fully Connected Layer 24](#_Toc95813190)

[4. Il Progetto 26](#_Toc95813191)

[*4.1.* Framework e librerie 27](#_Toc95813192)

[4.1.1 Pyhton 27](#_Toc95813193)

[4.1.2 Tensorflow 27](#_Toc95813194)

[4.1.3 Keras 28](#_Toc95813195)

[*4.2.* Dataset 28](#_Toc95813196)

[4.2.1 Problematiche del dataset 29](#_Toc95813197)

[4.2.2 Data Augmentation 30](#_Toc95813198)

[4.2.3 Transfer Learning 31](#_Toc95813199)

[*4.3.* Aumento del dataset 32](#_Toc95813200)

[*4.4.* Creazione della rete neurale 35](#_Toc95813201)

[4.4.1 Preprocessing dei dati 37](#_Toc95813202)

[4.4.2 Train Test Split 38](#_Toc95813203)

[4.4.3 Struttura della rete 38](#_Toc95813204)

[4.4.4 Training della rete 40](#_Toc95813205)

[4.4.5 Valutazione della rete 42](#_Toc95813206)

[*4.5.* Implementazione della rete in React Native 45](#_Toc95813207)

[4.5.1 Struttura del Progetto 45](#_Toc95813208)

[4.5.2 Funzionamento dell’applicazione 46](#_Toc95813209)

[4.5.3 Analyse Photo 48](#_Toc95813210)

[CONCLUSIONI 51](#_Toc95813211)

[RINGRAZIAMENTI 53](#_Toc95813212)

[INDICE DELLE FIGURE 54](#_Toc95813213)

[INDICE DEI GRAFICI 55](#_Toc95813214)

[INDICE DELLE TABELLE 56](#_Toc95813215)

[BIBLIOGRAFIA 57](#_Toc95813216)

# 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]

# 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.

## 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.].

## 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.

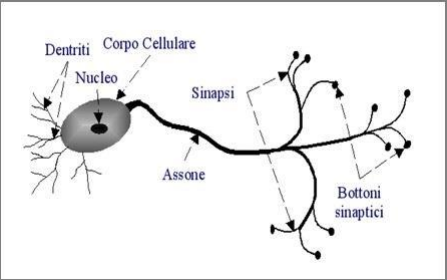
Altre motivazioni che spingono ad usare il deep learning sono legate ai vantaggi dell’impiego di architetture gerarchiche multi-strato. Un’organizzazione gerarchica dei dati permettere di condividere e riusare informazioni estratte durante l’elaborazione e di selezionare e scartare dettagli inutili lungo 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.

# RETI NEURALI

Le reti neurali artificiali (ANN: Artificial Neural Network) [8], come già anticipato, sono modelli matematici che simulano i comportamenti del cervello umano. ll cervello è un calcolatore straordinario: interpreta a velocità elevatissime informazioni fornite dai sensi e impara autonomamente a creare le rappresentazioni interne, che gli permettono di svolgere questa funzionalità. Già a partire dagli anni 90’ con i primi approcci si è cercato di riprodurre in maniera artificiale il comportamento del cervello progettando delle sorte di reti (ancora ai tempi embrionali ma oggi molto performanti), costruite tentando prima di individuare le caratteristiche essenziali dei neuroni e delle loro interconnessioni e poi programmando un calcolatore che simulasse tale comportamento.

## Neurone Biologico

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



**Figura 2.1 Rappresentazione schematica di un neurone biologico**

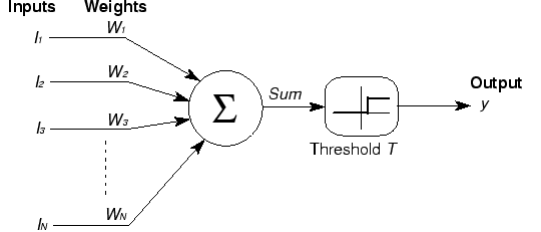
I principali componenti di un neurone biologico sono:

* **Soma**: parte centrale del neurone, in cui riesiedono il nucleo e gli altri organelli deputati alle principali funzioni cellulari;
* **Assone**: è un conduttore di impulsi, esso si origina a partire dal soma cellulare
* **Sinapsi**: “ramificazioni” dell’assone,che consente la comunicazione delle cellule del tessuto nervoso tra loro o con altre cellule, ciascuna sinapsi termina con un bottone sinaptico;
* **Dendriti**: filamenti che si ramificano a partire dal neurone, che trasportano il segnale nervoso verso il 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.

## 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 2.2 Rappresentazione del modello McCulloch e Pitts**

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.

## 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

**Equazione 2.1 Output**

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

**Equazione 2.2 Output con vettori**

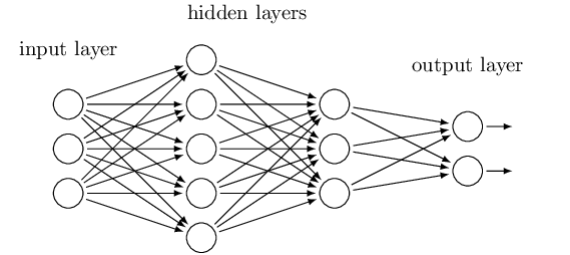
Dove a è detta “funzione di attivazione”. Il bias può essere interpretato come una soglia che influenza ampiamente l’output dell’unità. La novità principale del modello di Rosenblatt è la sua capacità di modificare i propri pesi adattandoli al problema dato in modo tale da non dover creare un circuito 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**

## 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 2.3 Esempio MLP**

## 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.

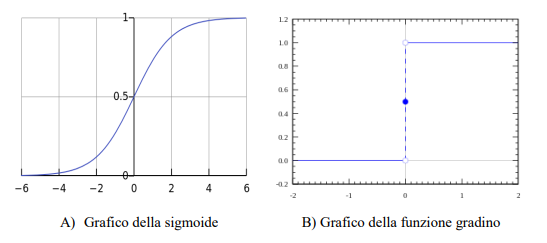
### 2.5.1. Sigmoide

La differenza sostanziale di tale funzione rispetto alla formula (2.2) è la sua natura non lineare da cui si ottengono combinazioni non lineari.



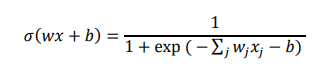
**Equazione 2.3 Sigmoide**

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



**Grafico 2.1.1 Sigmoide Grafico 2.1.2 Funzione gradino**

La nuova funzione di attivazione diverrà dunque:



**Equazione 2.4 Sigmoide con valore sostituito**

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 2.1.1, 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.

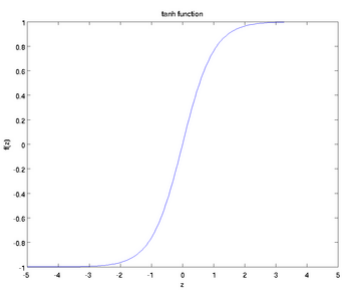
### 2.5.2. Tangente iperbolica

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:



**Equazione 2.5 Tangente iperbolica**



**Grafico 2.2 Funzione tangente iperbolica**

### 2.5.3. 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).



**Equazione 2.6 SoftMax**

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.

### 2.5.4. ReLU

Di questa funzione di attivazione parleremo in seguito nel capitolo 3. Per ora si sappia che è una funzione di attivazione dal carattere non lineare.

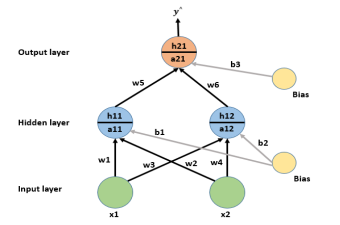
## Addestramento di una rete neurale

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.

### 2.6.1. 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 2.4 Rete Nurale 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

**Equazione 2.7 Errore**

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**.

### 2.6.2. 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

**Equazione 2.8 Back Propagation**

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

Immagine che contiene testo

Descrizione generata automaticamente

**Equazione 2.9 Gradiente**

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



**Equazione 2.10 Training rule**

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

**Equazione 2.11 Aggiornamento dei pesi**

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

# 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:



**Equazione 3.1 Convoluzione**

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:



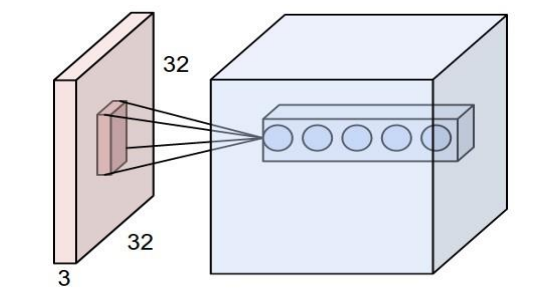
**Equazione 3.2 Convoluzione discreta**

## 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 3.1 Esempio di convoluzione su immagine [32x32x3]**

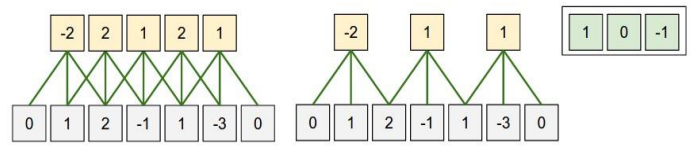
Facciamo un esempio pratico: supponiamo di avere un volume di input pari a [32x32x3] e la dimensione del filtro pari a [5x5]. Dunque, ogni neurone del Convolutional Layer avrà dei pesi per una regione del volume di input pari a [5x5x3] per un totale di75 pesi (+1 per il bias).

### 3.1.2 Organizzazione dello spazio

Quattro iper-parametri controllano la dimensione del volime 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 3.2 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.

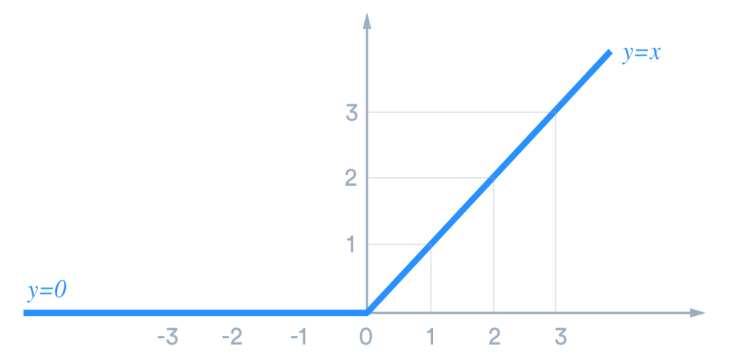
A quanto pare, però, si può ridurre drasticamente il numero di parametri con una sola ragionevole assunzione: se una feature è utile da calcolare in una precisa posizione () allora deve essere utile da calcolare anche in una posizione () differente.

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:



**Grafico 3.1 Funzione ReLU**

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 3.3 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 3.4 Operazione di MAX Pooling con filtro 2x2 e passo 2 su depth slice**

Prendendo in esame la figura 3.4, 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*.

Piu in generale un Pooling Layer:

1. Accetta un volume di dimensioni ;
2. Richiede due iper-paramentri:
   1. L’estensione spaziale di F;
   2. Il passo S.
3. Produce un volume di dimensioni ;
   1. ;
   2. ;
   3. ;
4. Non introduce ulteriori parametri poiché calcolo 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.

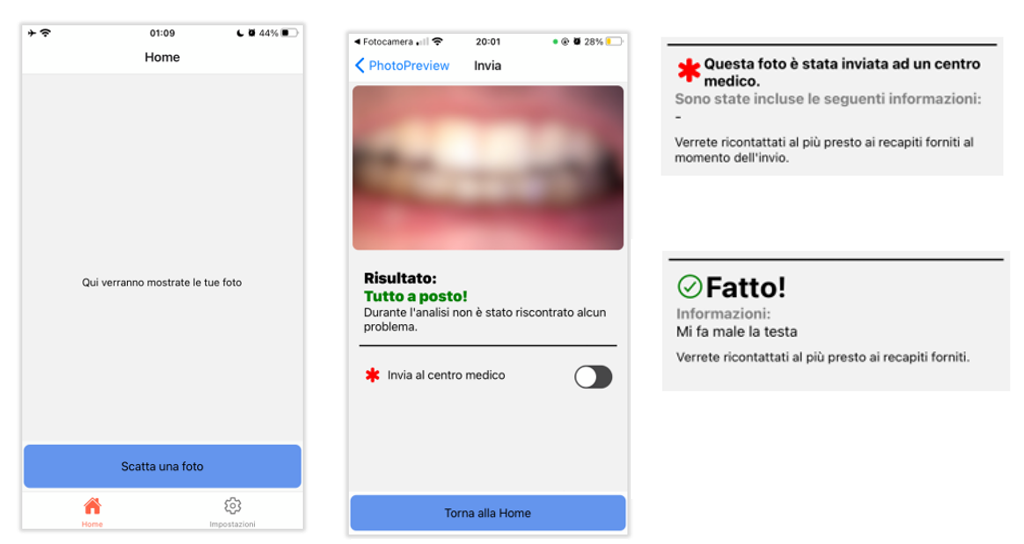
Possono essere presenti più *Fully Connected layer*, di cui l’ultimo avrà il parametro K pari al numero delle classi presenti nel dataset su cui sta lavorando la rete. I valori finali verranno infine passati all*’output layer* che avrà il compito di effettuare la classificazione utilizzando un’apposita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

# 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***

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.



**Figura 4.1 esempio di schermata Home, analisi ed invio al centro medico**

## 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.*

### 4.1.1 Pyhton

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.

### 4.1.2 Tensorflow

Tensorflow è una libreria software open source sviluppata da Google Brain e rilasciata sotto licenza Apache 2.0 per applicazioni di machine learning e, in particolar modo, modelli di deep learning come reti neurali. Definita come una “libreria di machine learning di seconda generazione”, Tensorflow si basa su Python ma mette a disposizione varie API native di alto e basso livello in anche linguaggio C/C++, Java e R. Alla base del framework Tensorflow vi è il concetto di flow graph. Un flow graph è un grafo diretto aciclico che rappresenta una computazione numerica. I nodi del grafo rappresentano delle operazioni matematiche, gli archi tra i nodi invece sono dei cosiddetti tensori, ovvero vettori multidimensionali di dati. Questo approccio offre la possibilità di operare ad alti livelli di astrazione per l’esecuzione di task numerici di basso livello. [github.com/tensorflow/tensorflow]

Le principali qualità di Tensorflow sono rappresentate dalle elevate performance, dalla possibilità di monitorare la fase di training del modello secondo un’ampia gamma di metriche e dall’ottimo supporto fornito da una community sempre più vasta. Infatti, Tensorflow è oggi una delle librerie di machine learning più diffuse ed è impiegata in numerosi servizi di Google, come Google Maps, Gmail e Ricerca.

### 4.1.3 Keras

Per quanto Tensorflow risulti essere un’ottimo framework per applicazioni di deep learning, la creazione di modelli rimane un processo complesso, in particolar modo per utenti poco esperti. Il framework open source Keras risponde all’esigenza di definire modelli di deep learning in manera intuitiva, modulare e con un numero contenuto di righe di codice. La principale peculiarità di questa libreria è la capacità di funzionare come interfaccia di librerie di più basso livello come Tensorflow o Theano. Keras fornisce tutte le più comuni implementazioni dei modelli di deep learning con una perdita di prestazioni contenuta rispetto a Tensorflow. A partire dal 2017, il team di Tensorflow ha deciso di aggiungere dei moduli per il supporto di Keras nella propria libreria. [keras.io]

## 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;
* **Linchen Planus**: 71 immagini;
* **Paziente Sano**: 44 immagini.

### 4.2.1 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.

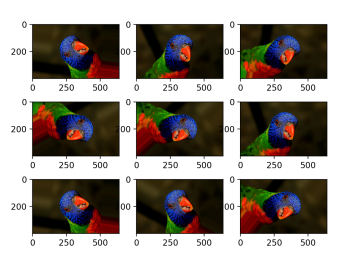
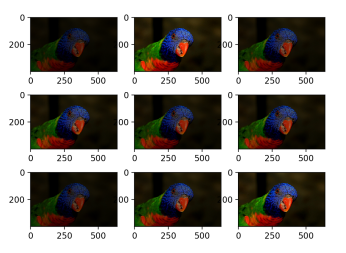
Fatta questa premessa risulta evidente come il dataset in nostro possesso sia composto da un numero di samples relativamente basso, ragion per cui, al fine di evitare tali problematiche verranno adoperate tecniche di deep learning come il “*Transfer Learning”*  e di ampliamento del dataset *“Data Augmentation”.*

### 4.2.2 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 4.4b 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.

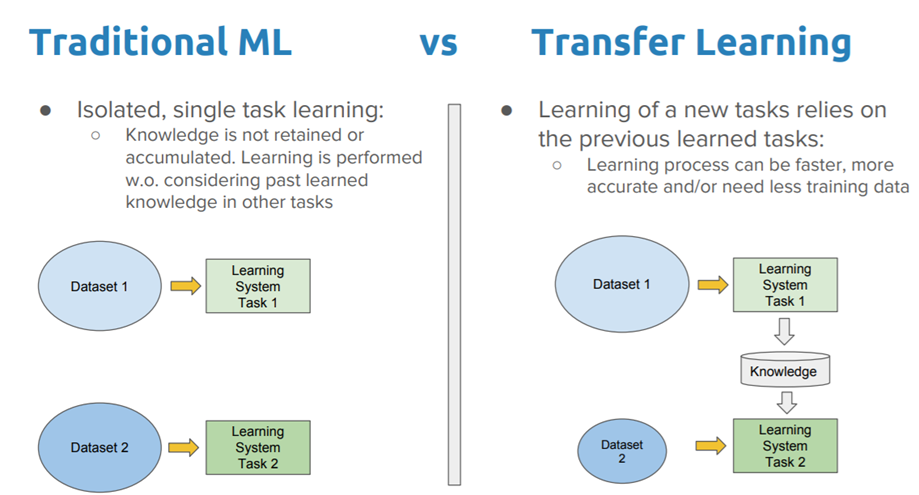
**Figura 4.2 Variazione casuale di rotazione Figura 4.3 Variazione casuale di luminosità**

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

### 4.2.3 Transfer Learning

È pratica commune sfruttare una rete già addestrata (tra quelle canoniche: AlexNet, ResNet, GoogleNet) e utilizzarla per un secondo o successivo tast. In particolare, vengono preservati i layer iniziali ( capaci di riconoscere in maniera performante le feature essenziali) e trasferiti per testare una rete o riallenare un modello simile: la rete può partire quindi con dei pesi già pre-allenati. A sua volta vi sono due sottoutilizzi: **Feature extractor** nel quale si rimuove l’ultimo layer (fully-connected) e si usa il resto della rete come feature extractor su un nuovo set di dati su cui lavorare oppure **il Fine-tuning** nel quale vengono ottimizzati i pesi della rete preaddestrata In questo modo le features generiche della rete diventano piano piano sempre piu’ specifiche per la nuova attività

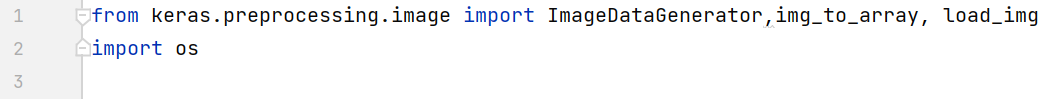
Con questa tecnica, non solo diminuisce il tempo di training ma si ha la possibilità di utilizzare un dataset costituito da un numero relativamente basso di sample (rispetto alla dimensione del dataset con cui è stata allenata la rete canonica senza Transfer Learning). A livello pratico, operare il TL significa sostituire i due layer finali delle reti già allenate: il fully connected e quello di classificazione. Il primo va sostituito con un fully connected layer avente il numero di neuroni corrispondente al numero di classi di immagini del nuovo dataset, mentre l’altro va sostituito con un layer di classificazione generico che imparerà a classificare i sample del nuovo dataset.



**Figura 4.4 Differenza tra Transfer Learning e Machine Learning tradizionale**

## Aumento del dataset

Come prima cosa per ovviare alla problematica relativa alla ridotta dimensione del dataset creiamo uno script Python per andare ad incrementare i nostri samples.



Per prima cosa importiamo dalla libreria di Tensorflow e dall’API Keras la classe ImageDataGenerator la quale genera batches di tensori di immagini per la data augmentation. Dalla stessa libreria importiamo la funzione img\_to\_array, per convertire le immagini in array, e la funzione load\_img per caricare un immagine in formato PIL. Infine verrà utilizzata la libreria OS per utilizzare le funzioni che ci permettono di interagire con il sistema operativo.

Immagine che contiene testo

Descrizione generata automaticamente

Dopo aver importato le librerie di cui abbiamo bisogno andiamo a creare il nostro generatore di immagini con i seguenti parametri:

* **rotation\_range**: valore in gradi (da 0 a 180), range entro cui ruotiamo le foto in maniera random
* **width\_shift\_range** e **height\_shift\_range**: range (definite come frazione della larghezza o altezza totale) entro cui trasliamo in modo random la figura in orizzontale o verticale
* **shear\_range**: applica casualmente una distorsione all’immagine lungo un asse
* **zoom\_range**: applica uno zoom in maniera random all’immagine
* **horizontal\_flip:** capovolge randomicamente le immagini
* **fill\_mode**: quando l’immagine viene ruotata, alcuni pixel andranno fuori lasciando un area vuota che deve essere riempita, tramite fill\_mode=’nearest’ andiamo a riempire gli spazi vuoti con i pixel piu’ vicini
* **brightness\_range**: con questo valore andiamo a controllare la luminosità dell’immgine

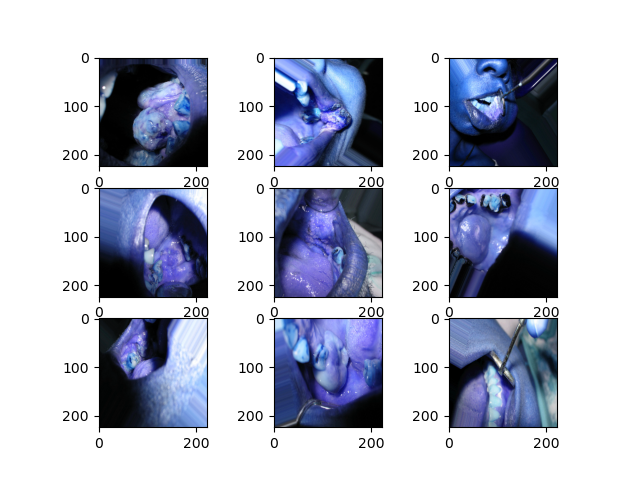
Immagine che contiene testo

Descrizione generata automaticamente

Creato il nostro generatore di dati andiamo a creare un nuovo dataset e tramite un ciclo for per ogni foto di ogni classe ne andiamo a generare altre sei nuove tramite il metodo flow dell’ImageDataGenerator.

Fatto ciò riusciamo a passare da un dataset iniziale di 280 immagini la cui suddivisione è stata definita al paragrafo §4.2 a un nuovo dataset di 1647 sample cosi suddivisi:

* **Carcinoma orale**: 404 immagini;
* **Leucoplachia**: 567 immagini;
* **Linchen Planus**: 417 immagini;
* **Paziente Sano**: 259 immagini.



**Figura 4.5 Esempio di immagini del nuovo dataset aumentato**

## Creazione della rete neurale

Risolto il problema della ridotta dimensione del dataset, siamo pronti a costruire la rete neurale convoluzionale che andrà a riconoscere le potologie del cavo orale a partire da un’immagine.

Immagine che contiene testo

Descrizione generata automaticamente

Inanzitutto importiamo le librerie utilizzate per lo sviluppo della rete, in cui oltre alle già citate tensorflow e os, troviamo: **Numpy,** per poter operare efficientemente con grandi matrici e array multidimensionali; **Pathlib** che include classi per gestire percorsi di filesystem; **OpenCV** per la gestione delle immagini (Computer Vision)

Immagine che contiene testo

Descrizione generata automaticamente

Andiamo inizialmente a definire le variabile che ci serviranno per realizzare la nostra rete neurale. Partiamo dagli Hyperparameters:

* **INIT\_LR**: learning rate che controlla la velocità di apprendimento, cioè stabilisce l’errore ripartito sui nodi per l’aggiornamento dei rispettivi pesi
* **EPOCHS**: definisce il numero di volte in cui l’algoritmo di learning lavorerà attraverso l’intero training dataset
* **BS**: batch size, definisce il numero di samples da elaborare prima di aggiornare i paramentri del modello

Dopodiché definiamo due dizionari, uno che contiene la lista delle immagini di ogni classe (**injury\_images\_dict**) e uno che assegna ad ogni classe un etichetta (**injury\_labels\_dict**). Definiamo infine due liste vuote,X e y, in cui andremo ad inserire rispettivamente tutte le immagini e tutte le labels.

### 4.4.1 Preprocessing dei dati

Per caricare le immagini è stato necessario utilizzare la libreria *Open CV:* le immagini vengono infatti caricate e ridimensionate per essere concordi con la dimensione della CNN

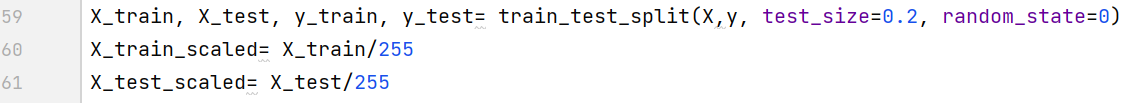
Immagine che contiene testo

Descrizione generata automaticamente

Tramite un ciclo for, ogni immagine viene ridimensionata ad un formato 224 x 224 pixel ed inserita all’interno della lista X vista in precedenza, ed il nome della corrispettiva classe è invece inserito nella lista y.

### 4.4.2 Train Test Split

La funzione “train\_test\_split” è stata utilizzata per dividere il Dataset.



Questa funzione infatti prende come ingressi gli elenchi (X e y) contenenti immagini e labels, e un “test\_size”: quest’ultimo parametro rappresenta una percentuale (espressa come numero tra 0 e 1) che indica la quantità di dati del Dataset che verrà utilizzata per la fase di test, per ricavare quindi la parte che verrà utilizzata durante l’addestramento sarà sufficiente sottrarre il valore test\_size da 1 (1-test\_size). Durante la fase di addestramento viene elaborato il vettore X\_train, per verificare se un risultato è o meno corretto, si compara il dato in uscita dalla rete con quello di y\_train: se sono uguali allora il risultato è corretto, altrimenti il risultato necessita di correzioni. Stessa cosa viene fatta nella fase di test. Infine come si può vedere dal codice si vanno a normalizzare le immagini ( sia in train che in test) per farsi che i valori dei singoli pixel sia compresi tra 0 e 1 anziché tra 0 e 255.

### 4.4.3 Struttura della rete

Immagine che contiene testo

Descrizione generata automaticamente

Come spiegato in precedenza per andare a migliorare la performance della rete neurale si è optato per l’utilizzo del Transfer Learning, cioè andare ad effettuare il *training* su reti già pre-allenate con ImageNet. Questa operazione è stata svolta allenando solo l’ultimo *layer,* mantenendo quindi costanti i pesi dei livelli precedenti. La rete *pre-trained* utilizzata è stata la MobileNetV2.

Keras mette a disposizione le *Keras application,* cioè modelli pre-allenati di Deep Learning resi disponibili insieme ai loro pesi. Questi modelli possono essere utilizzati per la classificazione, il *feature extraction* e il *fine-tuning.* Tra questi troviamo MobileNet e MobileNetV2, questi due modelli fanno parte di una classe di reti neurali efficienti, utilizzate frequentemente per applicazioni di **mobile vision** ed **embedded.** E’ stata dimostrata quindi l’efficacia delle MobileNets in una vasta gamma di applicazioni tra cui: rilevamento di oggetti, classificazione, *pattern matching* di volti e geolocalizzazione. MobileNetV2 offre una classificazione immediata delle immagini quando la categoria che si desidera prevedere è disponibile già nelle classi di ImageNet. Altrimenti grazie al Fine-Tuning, si può ottimizzare la rete per il set di dati e le classi di cui siamo a disposizione.

Come possiamo vedere dal codice, chiamiamo la funzione keras che istanzia l’architettura MobileNetV2 a cui passiamo i seguenti parametri:

* **weights=’imagenet’**: in questo modo andremo ad utilizzare i pesi pre-allenati su ImageNet;
* **include\_top=False**: impostando questo parametro a “*False*” non includiamo i layer *“Fully-Connected”;*
* **input\_Tensor**: infine definiamo il formato che dovranno avere le immagini in input.

Dopo aver definito la base del nostro modello, andiamo ad aggiungere i seguenti layer per completare la rete:

* **Flatten**: aggiungendo questo layer andiamo a rimuovere tutte le dimensioni di un tensore eccetto una che avrà un formato pari al numero di elementi contenuti nel tensore. In questo caso si passa quindi da un tensore di dimensioni (224 x 224 x 3) a un tensore 1-D di (150.528 x 1);
* **Dropout**: questo livello fa si che alcune unità neurali, durante le fasi di allenamento, vengano ignorate. Appare come una forma di regolarizzazione che agisce “sganciando” alcuni nodi dalla rete. Tutto ciò viene fatto per ridurre l’*overfitting;[21 tesi.pdf cartella tesi]*
* **Dense**: layer finale per creare lo strato *“fully-connected”*  con parametri:
  + **4:** intero che definisce il numero di output che sarà pari alle classi da predirre;
  + **activation=’softmax’:**definiamo la funzione di attivazione da utilizzare che in questo caso sarà la *softmax.*

Per concludere con l’utilizzo di un ciclo for settiamo l’attributo *trainable* dei layer della rete pre-allenata a *false*, per far sì che in fase di training essi non vengano modificati.

### 4.4.4 Training della rete

Immagine che contiene testo

Descrizione generata automaticamente

Prima di passare all’allenamento della rete andiamo a definire un paramentro fondamentale per compilare la nostra rete ovvero l’*optimizer* cioè una funzione o un’algoritmo che modifica gli attributi della rete neurale come pesi e learning rate. Essa aiuta a ridurre la *loss*  e migliorano l’*accuracy.*

Come ottimizzatore è stato scelto **Adam** che si basa su un algoritmo chiamato *“Stochastic Gradient Descend”* e sull’unione di due ottimizzatore preesistenti: **AdaGrad** (*Adaptive Gradient*) e **RMSProp** (*Root Mean Square Propagation*). Ad oggi Adam viene considerato lo stato dell’arte per quanto riguarda i neuroni nella Computer Vision e nel riconoscimento di linguaggi “naturali”[5,6 implementazione di una rete neuale ccn custom.pdf]. Al metodo Adam messo a disposizione da keras passiamo il learning rate precedentemente definito e il decadimento pari al *learning rate* fratto il numero di epoche.

Definito l’ottimizzazione possiamo compilare la rete chiamando il metodo *compile*  a cui passiamo l’ottimizzatore e la funzione di *loss,* utilizzata per decodificare l’output delle rete. Come funzione di loss viene utilizzata *Sparse Categorical Crossentropy* utilizzata per la classificazione multiclasse che differisce dalla *Categorical Crossentropy* per la rappresentazione dell’output, che in quest’ultimo caso è un vettore con un “1” nella posizione del risultato piu’ probabile mentre nella *Sparse Categorical Crossentropy* viene restituito un numero corrispondente alla classe piu’ probabile. Come ultimo paramentro del metodo compile abbiamo *metrics*  che poniamo uguale ad *accuracy*, sarà quindi quest’ultima il valore che verrà valutato durante il *training* e il *testing*.

Fatto ciò si può passare al training tramite il metodo *fit* a cui passiamo:

* **ImageDataGenerator**: discusso al paragrafo § 4.3 a cui passiamo le immagini normalizzate presenti nel *numpy array* (X\_train\_scaled) e le label (y\_train);
* **steps\_per\_epoch**: numero totale di step da effettuare prima che un epoca si può considerare conclusa. Variabile posta pari al numero di samples nell’*array* di train diviso la *batch size;*
* **validation\_data**: dati attraverso i quali andiamo a valutare la *loss* e *l’accuracy* che nel nostro caso sarà (X\_test\_scaled e y\_test) ottenuti dal dataset iniziali tramite il Train Test Split(sottoparagrafo § 4.4.2);
* **validation\_steps**: numero totale di step da validare prima di stoppare l’esecuzione;
* **epochs**: per concludere passiamo anche il numero di epoche prestabilito.

In conclusione andiamo a salvare il nostro modello

Immagine che contiene testo

Descrizione generata automaticamente

Andiamo a definire il path del nostro modello, che passeremo alla funzione *save* di tensorflow insieme all’estensione( .h5) che successivamente ci servirà per andare ad implementare la rete nurale in *React Native.*

### 4.4.5 Valutazione della rete

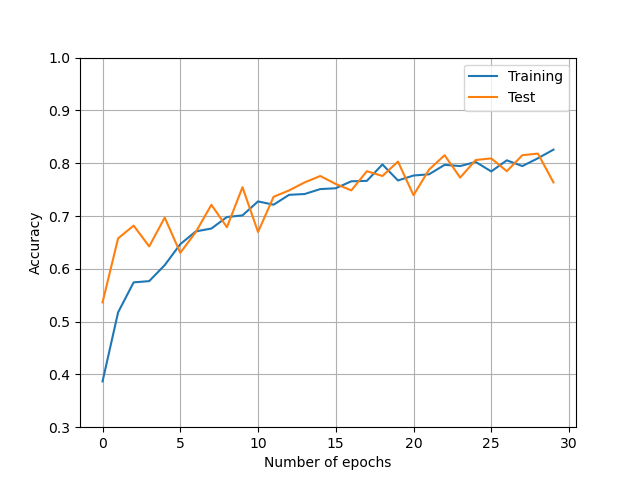
Dopo che l’allenamento della rete è andato a buon fine bisogna andare a valutare la rete, cioè verificare se l’algoritmo utilizzato ha prodotto una rete neurale in grade di ottenere buone performance in termini di accuracy, sia in *training* che in *testing.*

Per fare ciò ci avvaliamo di un’altra libreria python che ci permette di andare a grafica l’andamento dei due valori di accuracy. La libreria in questione è Matplotlib.

Immagine che contiene testo

Descrizione generata automaticamente

Da cui otteniamo:



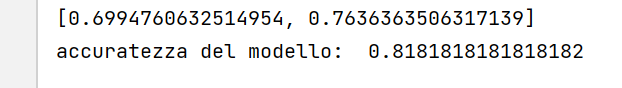
**Grafico 4.1 Andamento dell’accuracy per Training e Test al variare delle epoche**

Come si può notare l’accuracy della rete in fase di test non si discosta da quella in fase di training ciò quindi dimostra come si è riusciti ad evitare l’overfitting andando a generalizzare in maniera adeguata la CNN e ad ottenere un valore di questa grandezza abbastanza discreto. Attorno all’81%, nello specifico eseguendo questo comando:

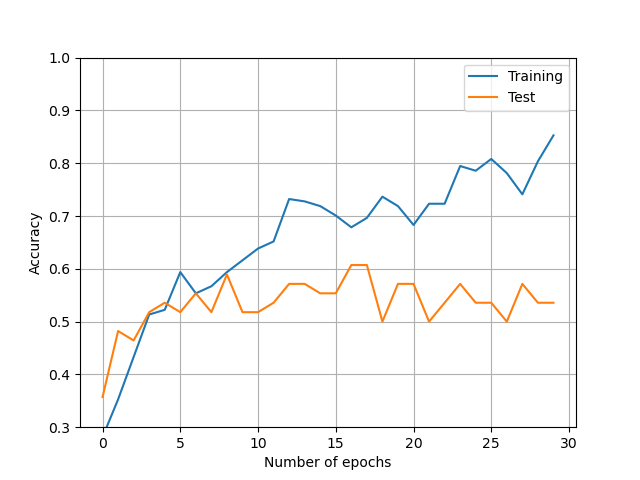
Immagine che contiene testo

Descrizione generata automaticamente

Otteniamo:



Per riuscire a capire l’importanza di avere un grande numero di dati con cui allenare la nostra rete e quindi la scelta di andare ad aumentare il numero di samples, andiamo a valutare la stessa rete con il dataset iniziale (composto da 280 immagini) e otteniamo:



**Grafico 4.2 Andamento dell’accuracy per Training e Test al variare delle epoche (caso dataset iniziale)**

Guardando il grafico si riesce subito a capire come ci sia un discostamento non indifferente tra l’andamento dell’accuracy in training rispetto a quello in testing, di fatto risulta evidente una situazione di overfitting con la rete che in allenamento raggiunge picchi di accuratezza vicini al 90% (valore persino maggiore della rete con dataset aumentato) ma che in testing si assesta attorno ad un valore medio del 50%, ottenendo quindi una CNN non utilizzabile.

Immagine che contiene testo

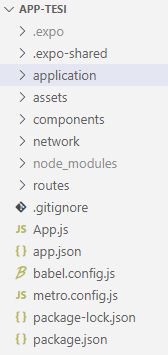
Descrizione generata automaticamente

## Implementazione della rete in React Native

Come già detto in precedenza, l’obiettivo finale del seguente lavoro di tesi è quello di andare a realizzare un’applicativo per smartphone che riesce a classificare alcune lacerazioni della cavità orale (cancerose e precancerose), mediante l’utilizzo di una rete neurale. Nei paragrafi precedenti abbiamo discusso in merito alla realizzazione del modello, ci manca quindi la sua implementazione all’interno dell’applicazione. Per lo sviluppo di quest’ultima si è optato per il framework React Native, che permette la creazione combinata 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 ttols per valutarne le prestazioni e consente di distribuirla in modo diretto[tesi aurelio d’urso paragrafo 2.1]

### 4.5.1 Struttura del Progetto

Diamo al lettore una piccola descrizione generale sul progetto dell’applicazione partendo dalla sua *directory :*



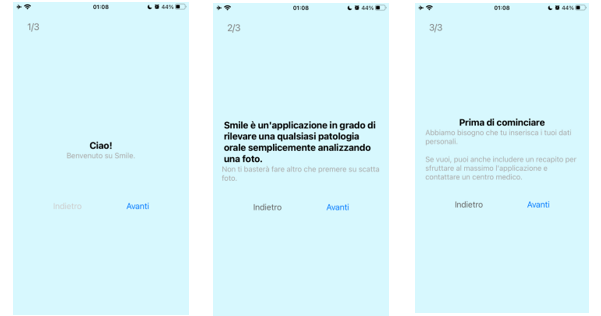
**Figura 4.6 Struttura della directory**

Le cartelle principali sono:

* **assets**: contiene le immagini (ed eventualmente altri tipi di file multimediali) necessati all’applicazione, come l’icone, il logo e lo splash screen, oltre che il modello della rete neurale che avevamo precedentemente salvato in fase di creazione (paragrafo §4.4.4);
* **application**: contiene le funzioni aggiuntive dell’applicazione. In questo caso, contiene il modulo deep learning che analizza la foto;
* **components**: contiene le varie schermate dell’applicazione: Home, Settings, Details, ecc;
* **routes**: rappresenta le varie routes dell’applicazione, ovvero come le varie pagine sono interconnesse tra di loro.

### 4.5.2 Funzionamento dell’applicazione

Ogni volta che viene effettuato l’accesso all’applicazione, viene restituita la schermata home che tramite una funzione ( *getFA()* ) rileva se è la prima volta che si accede all’app, in caso affermativo si aggiunge in cima allo stack di navigazione la pagina *FirstAccess* che consiste in tre schede introduttive



**Figura 4.7 Le tre schede introduttive**

Dopo le tre schermate si passa ad una nuova pagina di navigazione, ovvero quella relativa all’iscrizione: *Subscription*

Immagine che contiene testo

Descrizione generata automaticamente

**Figura 4.8 Schermata di iscrizione**

Una volta inseriti i dati, si può premere sul tasto *“iscriviti e accedi all’app”*. I dati verranno salvati localmente nel dispositivo e verrà visualizzata la schermata home. In quest’ultima schermata vengono visualizzate tutte le foto scattate ed analizzate. Inizialmente è vuota:

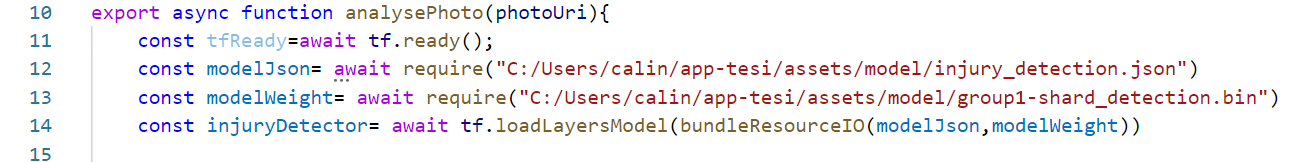


**Figura 4.9 Schermata home**

Da questa schermata è possibile iniziare l’analisi premendo sul tasto *“ Scatta una foto”* che carica la schermata della fotocamera che oltre a permettere di scattare una foto dà la possibilità di accedere alle foto del rullino fotografico e prelevare una foto già presente in memoria. Confermata la foto per l’analisi, si viene portati alla schermata *photoForm.js,* la quale come prima cosa richiama *analysePhoto.js* dov’è contenuta la funzione che permette di implementare la rete neurale nell’applicazione.

### 4.5.3 Analyse Photo

Alla funzione verrà passato l’URI dell’immagine prelevata

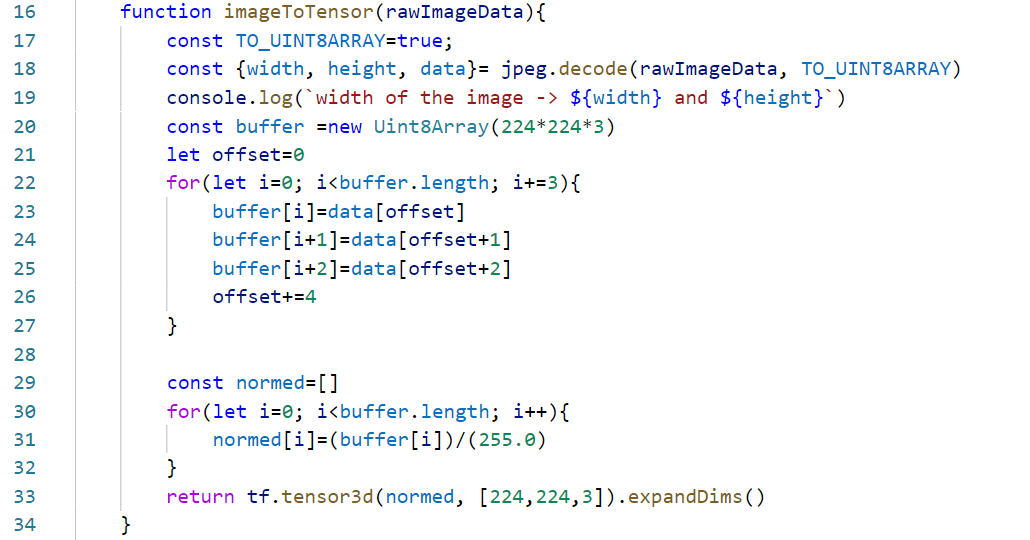


Successivamente chiamiamo la funzione *tf.ready(),* tramite la libreria messa a disposizione da JavaScrip per il machine learning (TensorFlow.js), la quale ritorna una promise che determina quando tensorflow.js è pronta per essere usata. Fatto ciò richiamo il modello (precedentemento convertito in .json) e i pesi, e carico il modello tramite la funzione *loadLayersModel* a cui passiamo i due path del modello e dei pesi.

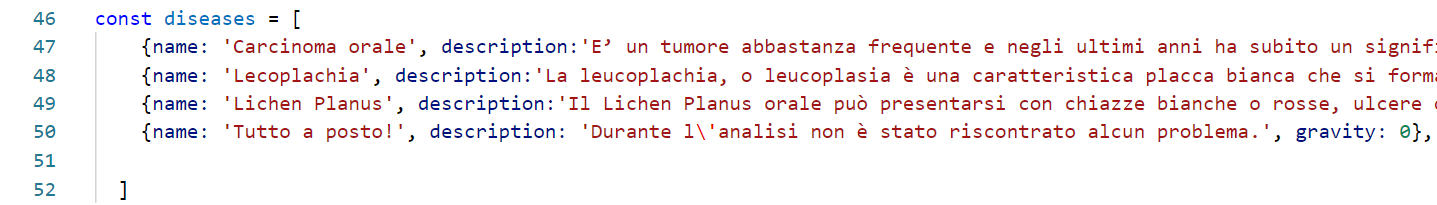
Immagine che contiene testo

Descrizione generata automaticamente

Inizializzata la nostra rete, andiamo innanzitutto a decodificare il nostro URI in *Base64,* creiamo un buffer e infine trasformiamo la nostra immagine in un tensore tramite questa funzione:



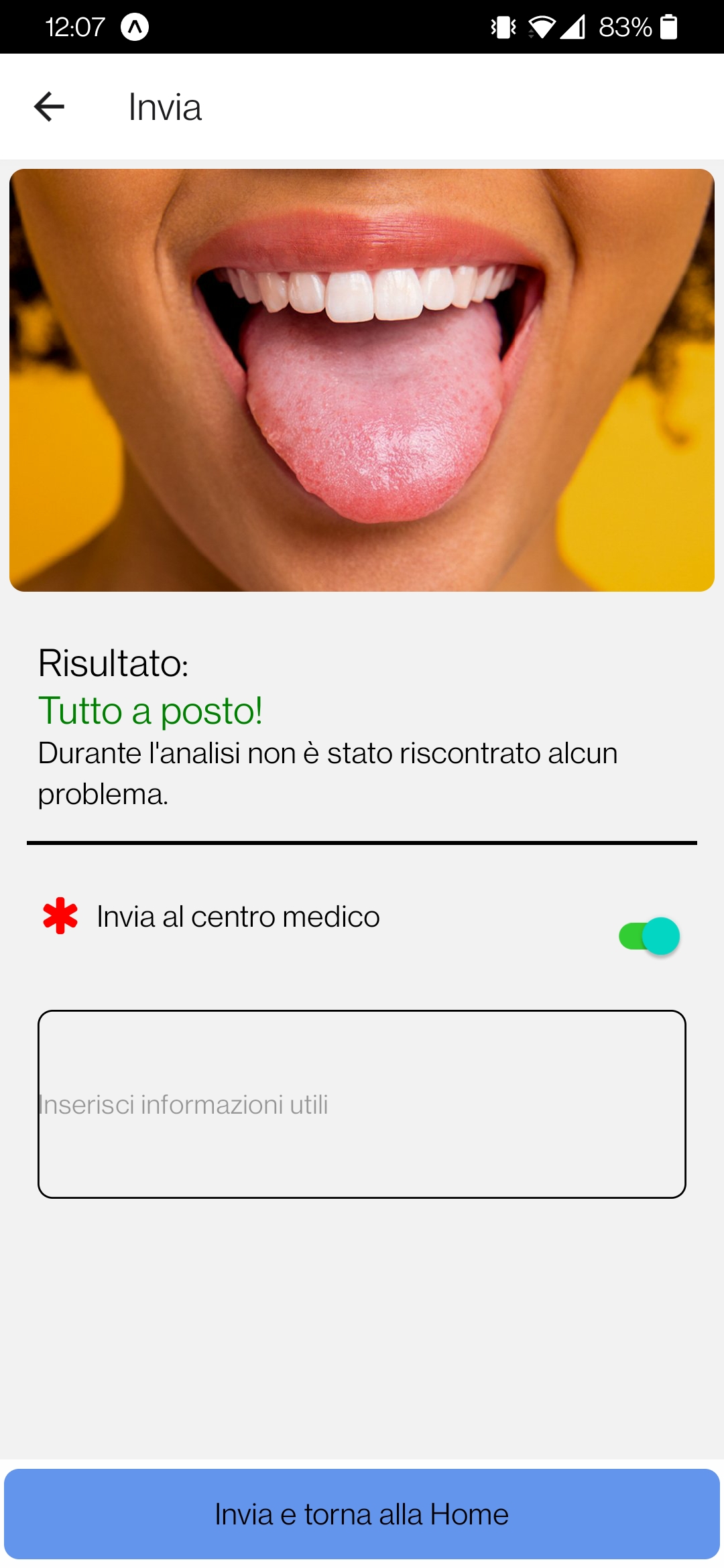
Trasformata l’immagine in tensore lo vado a ridimensionare (224 x 224) per poi chiamare la funzione *predict* del modello, la quale mi ritornerà un array con 4 valori, ognuno dei quali rappresenta la probabilità di ogni classe, da questo array, utilizzando la funzione *argMax* ottengo l’indice relativo alla classe predetta. Questo indice verrà utilizzato per estrarre un elemento dalla lista (che sarà ciò che verrà restituito dalla funzione) *diseases* cosi fatta*:*



Ogni oggetto della lista avrà le seguenti proprietà:

* **name**: il nome dell’eventuale patologia;
* **description**: la descrizione dell’eventuale patologia;
* **gravity**: numero intero [0-2] indicante la gravità della patologia.

Vediamo alcuni esempi di analisi:

Immagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamente

**Figura 4.10 Esempi di analisi**

A questo punto l’utente, se per qualche motivo non è soddisfatto del risultato dell’analisi, oppure desidera approfondire, può decidere di inviare la foto e il relativo risultato ad un centro medico toccando l’apposito switch presente in fondo alla schermata.

# CONCLUSIONI

All’inizio del progetto è stato posto l’obiettivo di implementare una rete neurale artificiale con lo scopo di classificare le patologie della cavità orale, analizzando immagini scattate in tempo reale o esistenti in memoria, mediante l’utilizzo di un’applicazione smartphone.

Sebbene possa sembrare un compito non particolarmente complesso, è stato necessario partire dalle conoscenze teoriche alla base del *Machine Learning* e delle reti neurali convoluzionali, perché attualmente risultano essere il modello di calcolo che permette risultati migliori nella Visione Artificiale. Si è quindi discusso del framework utilizzato (Python) e di come sfruttare il *Transfer Learning* allo scopo di riutilizzare la “conoscenza” di una rete neurale allo stato dell’arte e di come è stato possibile specializzarla sul dominio di interesse.

In seguito è stato trattato l’ambito *mobile,* mostrando brevemente alcuni dei meccanismi che permettono l’implementazione di una rete neurale all’interno di un’applicazione. Riuscendo ad ottenere una modello di rete con buone prestazioni

Per eventuali sviluppi futuri di questo lavoro di tesi sarebbe conveniente aumentare il valore delle epoche, migliorare il dataset sia dal punto di vista della quantità delle immagini sia dal punto di vista della distribuzione per evitare di avere uno sbilanciamento delle classi, ed essere provvisti di una GPU performante. Così facendo si raggiungerebbero prestazioni migliori.

Considerando invece un punto di vista piu’ ampio si nota l’importanza di applicare questo tipo di tecnologia in ambito medico. Numerosi sono i vantaggi che si prospettano dall’applicazione del machine learning alla medicina nel rendere la pratica clinica piu’ efficiente, conveniente, personalizzata ed efficace. Da questo semplice lavoro di tesi ci si può già rendere conto di come l’esperienze pregressa e i dati di milioni di pazienti possano aiutare i medici a fare diagnosi piu’ rapide e accurate, facilitare i ricercatori a comprendere i meccanismi genetici alla base delle malattie e predirre il rischio di una patologia in tempo utile per prevenirla. Le nuove tcnologie stanno trasformando la medicina e questa rivoluzione parte dai dati. Dati sanitari, immagini cliniche, sequenze del genoma, dati sulle tarapie prescritte e sui risultati ottenuti: dati che ciascuno di noi ha contribuito a creare.

# RINGRAZIAMENTI

Maecenas eget venenatis sapien. Fusce accumsan porta tincidunt. Nulla tempus sapien posuere dui venenatis hendrerit. Sed est justo, porttitor vitae nibh accumsan, feugiat blandit ante. Aenean sed dui commodo, faucibus lectus ac, faucibus nunc. Integer sed lorem posuere, rutrum nisi eu, ultricies quam. In egestas urna et massa rutrum eleifend. Sed malesuada lacus ac urna laoreet, nec mollis dui hendrerit. Phasellus neque nisl, scelerisque sed ipsum scelerisque, elementum consectetur sapien. Sed ut varius lectus, nec tempor mauris. Quisque sit amet placerat ante. Integer eget nisi tristique, consectetur ante nec, sagittis odio. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

# INDICE DELLE FIGURE

Figura 2.1 Rappresentazione schematica di un neurone biologico ………………6  
Figura 2.2 Rappresentazione del modello McCulloch e Pitts ……………………..8  
Figura 2.3 Esempio MLP ………………………………………………………..10  
Figura 2.4 Rete Neurale d’esempio ………………………………………………11  
Figura 3.1 Esempio di convoluzione su immagine [32x32x3] …………………..19  
Figura 3.2 Illustrazione di organizzazione spaziale ……………………………...20  
Figura 3.3 Esempio di operazione di pooling ……………………………………23  
Figura 3.4 Operazione di MAX Pooling con filtro 2x2 e passo 2 su deepth slice..23  
Figura 4.1 Esempio di schermata Home, analisi ed invio al centro medico………26  
Figura 4.2