# Team Hilo – Modulo Ryan

In questo documento, il team HILO andrà a presentare il modulo di intelligenza artificiale progettato congiuntamente al progetto di Ingegneria del Software nell'anno accademico 2020/2021. Il suddetto modulo consiste in un agente in grado di processare delle immagini con lo scopo di aiutare gli operatori sanitari nell'identificazione veloce ed efficiente dei casi di Covid-19 e, di conseguenza, di organizzare nel modo migliore possibile la coda dei tamponi da effettuare dando precedenza ai casi più urgenti e probabili.

Di seguito è riportata la definizione formale del problema secondo i criteri PEAS:

- **Performance**: Percentuale di probabilità di positività al Covid-19 di un paziente data la sua radiografia
- **Environment**: l'ambiente dell'agente sarà composto da un'immagine rappresentante la radiografia al torace di un paziente. Parliamo quindi di un ambiente completamente osservabile, a singolo agente, discreto, noto, statico, sequenziale, deterministico
- **Actuators**: mostra a video il risultato
- **Sensors**: immagini date in input

## Discussione sul dominio del problema

Il problema preso in esame consiste nell'analizzare delle immagini rappresentanti le radiografie al torace di pazienti ospitati nella struttura ospedaliera che ha adottato la piattaforma di gestione integrata del team Hilo. Risulta conveniente analizzare queste radiografie poiché chi è affetto da Covid-19 presenta delle macchie diffuse sulla quasi totalità della superficie polmonare che man mano si sviluppano seguendo i vasi sanguigni presenti in quel luogo; differisce dalla normale polmonite interstiziale poiché in quel caso caso la presenza è localizzata in uno o più punti ben circoscritti all'interno del polmone (si parla di veri e propri focolai). Sfruttando la caratteristica dell'estensione delle suddette macchine è possibile addestrare un algoritmo di intelligenza artificiale per stabilire se un paziente sia affetto da Covid-19 o meno e quindi fornire una stima di quanto possa essere urgente (e utile) effettuare un tampone molecolare, fornendo un'ulteriore metrica di priorità al gestionale. Così facendo si può gestire una coda di tamponi da effettuare in maniera efficiente e coerente con i veri bisogni dei pazienti.

Da questa definizione si evince che ci si trova davanti ad un problema di apprendimento perché lo scopo è quello di addestrare un agente artificiale a riconoscere la presenza di una infezione da Covid-19 all'interno dei polmoni differenziandola da quella che può essere una normale polmonite generica o, banalmente, dall'assenza di infezioni.

## Definizione della "natura" del problema

Definita quindi la branca del problema, un nodo da sciogliere è rappresentato dalla natura di quest'ultimo. Per "natura del problema" ovviamente si intende se esso consista in un problema di Classificazione, di Regressione o di Clustering. Dal momento che è stato possibile reperire un dataset di radiografie toraciche provviste di etichette che distinguono i vari casi presi in considerazione dal problema (ovvero assenza di polmonite, presenza di polmonite generica e presenza di polmonite da Covid-19) in automatico è possibile escludere il Clustering come strada da intraprendere. È inoltre pacifico stabilire che non ci si trova in una situazione di Regressione in quanto lo scopo non è fornire una percentuale di positività al Covid-19 ma una "risposta secca" del tipo "Covid SI", "Covid No" e "Covid NO con Polmonite". Da quest'ultima considerazione (e per esclusione) si può dedurre che ci si trova di fronte ad un problema di Classificazione.

## Scelta dell'algoritmo da utilizzare

Per l'implementazione si è deciso di optare per una rete neurale in quanto, a differenza di random forest e alberi di decisione, è l'unica struttura che è in grado di lavorare con delle immagini. Adesso però il problema è decidere quale tipologia di rete neurale adottare. In primo luogo, abbiamo pensato ad una rete neurale a singolo livello (percettrone) però, a seguito della lezione a proposito, lavora solo con problemi linearmente separabili e quindi l'abbiamo scartato. Dopodiché abbiamo pensato ad una rete neurale multilivello o MLP (Multi Level Perceptron), ovvero una struttura composta da un livello di input, un numero arbitrario di livelli intermedi chiamati "livelli nascosti" e un livello di output, che è in grado di risolvere anche problemi non linearmente separabili però a seguito di ricerche su internet, e sulla base dei consigli del prof, abbiamo scartato anche questa tipologia poiché nonostante la sua potenza, in particolare con le immagini non lavora bene e quindi non è in grado di fornirci i risultati che ci aspettiamo. Quindi abbiamo infine deciso di utilizzare come struttura di riferimento una rete neurale convoluzionale (CNN o Convolutional Neural Network), che è definita in un modo molto simile alla MLP però con una sostanziale differenza nella strutturazione dei livelli. Nella CNN abbiamo un livello di Convoluzione, un livello ReLu e un livello di pooling. Il livello di convoluzione si basa su un procedimento simile al passaggio di una lente di ingrandimento sull'input in modo da riconoscere componenti particolari via via meno elementari con l'aggiunta di ulteriori livelli di convoluzione e una importante caratteristica di questo livello è che i neuroni all'interno di questo livello condividono sia pesi che bias in input, simulando una visione sovrapposta dell'immagine. Successivamente al livello di convoluzione troviamo di solito un livello ReLu, che ha lo scopo di applicare la funzione ReLu ( $f(x) = \max(0, x)$) che ci permetterà di eliminare valori negativi dalla propagazione in avanti in quanto questi possono creare problemi come lo "spegnimento di neuroni" nel processo di backpropagation. Successivamente a questo livello possiamo trovare anche un livello di pooling il quale è responsabile di un ulteriore filtraggio del risultato proveniente dal livello di convoluzione. Questa operazione consiste nel creare un filtro di una determinata dimensione e nell'estrarre i valori (solitamente i massimi) della zona analizzata dal filtro. Il risultato sarà quindi un ulteriore riduzione dell'output però con un minor rischio di portare la rete in overfit. Infine, troviamo il livello di output che è un livello fully connected (ovvero che ha i neuroni collegati completamente a tutti gli output del livello precedente) e ha semplicemente lo scopo di eseguire la classificazione finale.
Un'altra peculiarità sta nel fatto che non si è vincolati al numero di livelli utilizzabili ma se ne possono usare quanti ne vogliamo e soprattutto l'ordine di posizionamento di questi non è del tutto rilevante, ma solitamente si preferisce impostare un ordinamento del tipo In, Conv, ReLu, Pool, Out, ecc. .

## Discussione sull'implementazione

Per sviluppare questo modello abbiamo scelto di usare Python visto che la stragrande maggioranza delle API in materia sono in questo linguaggio. In particolare, abbiamo scelto come libreria di riferimento "keras" con "tensorflow". Come prima operazione, abbiamo definito una dimensione standard per le immagini che utilizzeremo per fare preprocessing, al fine di renderle tutte uniformi e utilizziamo un solo canale di colori sfruttando il fatto che radiografie sono in bianco e nero. Oltre a questo, vengono effettuate anche delle operazioni di manipolazione sulle radiografie per rendere le immagini il più differenti possibili al fine di rendere la rete più robusta su radiografie randomicamente modificate (zoom, tagli vari, immagini sottosopra, ecc.). Come condizione di stop dell'addestramento abbiamo scelto il fatto che non si presenta un miglioramento del valore loss (che rappresenta il valore della precisione del modello, ovvero il grado di convergenza dei dati rilevati individualmente rispetto al valore medio della serie a cui appartengono) per tre epoche di fila e come funzione di attivazione per i neuroni abbiamo scelto di usare la funzione ReLu che è la più utilizzata al momento ed è anche quella che si comporta meglio nelle CNN. Di seguito verranno esposte le scelte a livello di API che abbiamo fatto:

- Come modello, abbiamo scelto quello sequenziale offerto da "keras", poiché ci permette di aggiungere livelli come se fosse una coda LIFO;

- Abbiamo alternato per tre volte la convoluzione 2D con un max pooling 2D (secondo lo schema citato in precedenza Conv, ReLu, Pool, Conv, ecc.);
- Abbiamo aggiunto un livello "flatten" (un livello che ha lo scopo di ridurre le dimensioni dell'input), con lo scopo di appiattire l'output senza modificare la batch size (ovvero il numero di campioni presi per volta prima di fare la validazione);
- A seguito del livello "flatten" abbiamo aggiunto un livello "Dense" con attivazione ReLu con lo scopo di convertire il vettore ottenuto con il livello flatten di numeri reali in un vettore di probabilità di categoria (banalmente quindi R -> [0,1]);
- Successivamente c'è un livello "Dropout" che randomicamente setta unità di input a zero (nel nostro caso con una frequenza di 0.5) e gli input non settati a zero sono scalati di $\frac{1}{1-0,5}$ con lo scopo di non variare la media di ogni input;
- Di seguito viene posto un livello "Dense" con attivazione SoftMax, la quale permette di rendere evidenti i valori con un peso maggiore rispetto a quelli con peso minore. Questo viene usato per permettere la classificazione finale nelle tre classi che abbiamo individuato;

Una volta stabiliti tutti i parametri della CNN, abbiamo fatto partire l'addestramento della rete e, dai primi risultati, ci siamo accorti che le prestazioni dell'agente si sono rivelate sorprendentemente alte (si parla da subito di un'accuratezza attorno al 90%). Il primo agente che abbiamo addestrato aveva una stopping condition di 2 epoche ed è arrivato ad un'accuratezza di circa il 91.65% dopodiché, non contenti, abbiamo riaddestrato la rete con una stopping condition di 3 epoche e siamo arrivati ad un'accuratezza di circa il 92.5%. Successivamente abbiamo provato ad aumentare nuovamente la stopping condition a quattro aspettandoci un ulteriore miglioramento ma purtroppo l'accuratezza dell'agente è scesa al di sotto del 90% e, in virtù dell'accaduto, abbiamo deciso di rimanere con una stopping condition di 3 epoche consecutive.

## Nota sul funzionamento dello script

*(Queste informazioni sono presenti anche nel file README della cartella "Modulo IA" presente sul repository di Github)*

In questa cartella compressa sono presenti (quasi) tutti i file necessari per il funzionamento del modulo di IA:

- oracle.py: realizza effettivamente le predizioni data una radiografia in input
- Ryan92,5 è un file binario che contiene la rete neurale già addestrata (con un'accuratezza del 92,5%)
- evaluate.py: effettua il testing della rete addestrata
- train.py: realizza l'allenamento della rete.

**N.B.** - nella presente consegna non è stato possibile includere il dataset in quanto di dimensioni troppo elevate: per questo motivo non è possibile eseguire ulteriori sessioni di training o di test a partire da questa consegna.

## Risultati dei test sulla rete addestrata

| | Covid | Non Covid | Polmonite |
|---|---|---|---|
| Covid | 100 | 4 | 12 |
| Non Covid | 3 | 280 | 34 |
| Polmonite | 5 | 60 | 790 |

## *Esecuzione della Rete*

L'effettiva classificazione delle radiografie si effettua inserendo queste ultime nella cartella *valuta/* e lanciando lo script "oracle.py" con il comando $python3\ oracle.py$. (Nella cartella *valuta/* sono presenti tre campioni appartenenti alle tre categorie possibili prese in esame (Covid, No Covid e Polmonite "standard" con assenza di Covid) per dimostrare il corretto funzionamento dell'agente)

Team Hilo "ristretto" :)

*- Hermann Senatore - matricola 0512105743*

*- Ivan Carmine Adamo - matricola 0512105755*

*- Lorenzo Criscuolo - matricola 0512105737*