# CT COVID-19 DETECTOR

Progetto per l’esame di Sistemi Digitali M

Autori: **Manfreda Filippo** e **Andrini Enrico**

# Sommario

# Introduzione

Nel mese di Dicembre 2019 nella città di Wuhan, in Cina, iniziano a riscontrarsi polmoniti atipiche. Già il 31 Dicembre le autorità cinesi comunicano il diffondersi di una nuova malattia all'organizzazione mondiale della sanità e pochi giorni dopo ufficializzano che tale malattia è veicolata da un nuovo virus, definito SARS-CoV-2, appartente alla famiglia dei coronavirus.

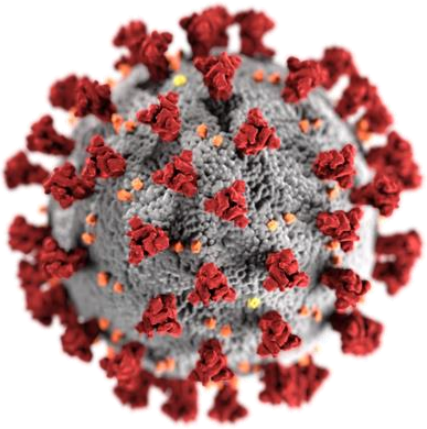

*Figura 1 - Rappresentazione grafica del coronavirus SARS-CoV-2*

Questo nuovo virus nell'arco di pochi mesi si diffonde in tutto il pianeta, portando l'OMS a dichiarare la situazione di pandemia globale. La malattia causata dal SARS-CoV-2 prende il nome di COVID-19, acronimo dell'inglese COronaVIrus Disease 19.

Caratteristica distintiva di questo virus è l'elevata contagiosità e a livello di malattia colpisce principalmente il tratto respiratorio superiore e inferiore ma può provocare sintomi che riguardano tutti gli organi e apparati. Nei casi più gravi che necessitano di cure ospedaliere, il tratto distintivo del virus è la presenza di una polmonite virale interstiziale.

Questo tipo polmonite era già presente prima del COVID-19, ma quella causata da quest'ultimo ha caratteristiche particolari poiché spesso si presenta in forma bilaterale (presente in entrambi i polmoni) e con focolai multipli (segni di infezione in diverse zone del polmone), condizione che prima era molto rara.

Lo strumento migliore per diagnosticare questo tipo di polmonite è la tomografia computerizzata (TC), una tecnica diagnostica per immagini con la quale si raccolgono dati dal passaggio di vari fasci di raggi X nell'area interessata e successivamente tali dati vengono rielaborati da un computer, il quale ricostruisce un'immagine tridimensionale della zona esaminata. Solitamente queste immagini tridimensionali vengono scomposte in una sequenza di immagini sul piano assiale per essere esaminate.

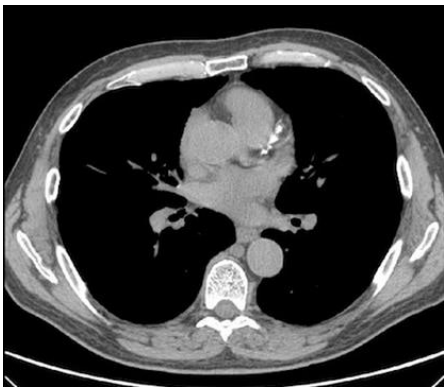

*Figura 2 - Immagine sul piano assiale di un esame toracico di TC*

La possibilità di utilizzare tecnologie con intelligenza artificiale a supporto della tradizionale medicina per velocizzare e migliorare la diagnostica di determinate patologie sta diventando negli ultimi anni un tema di grande importanza e in forte sviluppo.

Partendo da questo presupposto e per la situazione che stiamo vivendo attualmente, abbiamo deciso di sviluppare un rete neurale, utilizzando le API Keras, che permetta di rilevare i segni caratteristici del COVID-19 in immagini di tomografia computerizzata. La rete è stata definita utilizzando il modello EfficientNetB1 con l'aggiunta di nuovi layer e adottando la tecnica di transfer learning. E' stata addestrata utilizzando un determinato dataset trovato in rete e testata su diverse immagini, alcune di queste fornite da una conoscente che lavora come tecnico di radiologia in un ospedale pubblico della zona. Successivamente abbiamo sviluppato un'applicazione per smartphone e tablet Android, con la quale poter utilizzare la rete neurale creata sotto forma di modello TensorFlow Lite.

# 1 Richiami teorici

## 1.1 Keras

Keras è una libreria open-source realizzata in Python volta allo sviluppo di reti neurali. E' sviluppata come interfaccia di alto livello e si basa sulla piattaforma di machine-learning TensorFlow 2. Fornisce un'astrazione semplice ed intuitiva per lo sviluppo di applicazioni di reti neurali con un particolare focus sul deep learning.

I layer costituiscono i blocchi di base per lo sviluppo di reti neurali in Keras. Un layer consiste in una funzione che accetta un tensore come input e fornisce in uscita un altro tensore e qualche valore di stato raccolto in variabili TensorFlow. I principali layers messi a disposizione da Keras sono input layer (tensore base di TensorFlow), dense layer rappresentante un layer fully connected e activation layer contenente la funzione di attivazione dei neuroni del layer associato.

Un modello invece è un oggetto costituito da un gruppo di layers i quali formano la rete neurale da allenare, validare e testare. La costruzione di una rete efficace ed efficiente richiede generalmente tempo e costi non indifferenti, di conseguenza è comune utilizzare reti pre-allenate. Invece di ricostruire il problema dai minimi termini vengono riutilizzate le conoscenze pre-acquisite delle reti, generalmente con reti specifiche del problema.

Una volta creato il modello, Keras mette a disposizioni molte funzionalità per l'allenamento, la validazione e il testing della rete. In particolare:

- *Model.compile*: imposta la funzione di loss dei layer della rete e l'ottimizzatore, definibile a priori con parametri come il learning rate.
- *Model.fit*: permette di impostare i parametri dell'allenamento come il numero di *epoch* e con il quale vengono caricati il set di dati per il training e per la validation.
- *Model.predict*: effettua il testing della rete restituendo un Numpy Array con i risultati delle predizioni.

Una volta completato l'allenamento della rete, Keras permette di salvare il modello nella propria directory. Inoltre consente di caricare modelli e quindi di utilizzare reti già allenate precedentemente.

## 1.2 Data augmentation, transfer learning e fine-tuning

Per incrementare l'efficienza della rete sono utilizzate tecniche influenti sul dataset di immagini e sulla capacità di allenamento dei layers della rete:

1) *Data augmentation*: tecnica usata per creare versioni modificate delle immagini del dataset aumentando la capacità della rete di generalizzare anche in situazioni diverse e realistiche. Mette a disposizione operazioni come la rotazione, il ribaltamento o l'aumento di contrasto dell'immagine.

2) *Transfer learning*: tecnica che sfrutta capacità di riconoscimento ottenute precedentemente da una rete su un precedente problema. I layers già allenati vengono "congelati"(non sono allenabili), mentre al di sopra ne sono aggiunti di nuovi che sfruttano le caratteristiche e le potenzialità della rete pre-allenata.

3) *Fine-tuning*: opzione successiva al transfer learning, consiste nello scongelamento di layers precedentemente congelati e al loro allenamento sfruttando un learning rate generalmente basso in modo da perfezionare le precedenti capacità della rete sui nuovi dati del problema.

## 1.3 EfficientNet

Le reti neurali convoluzionali (CNN) sono comunemente sviluppate a un costo di risorse fisso e quindi ridimensionate per ottenere nuove reti simili ma con una migliore precisione all'aumentare delle risorse disponibili. Nata come evoluzione delle ConvNet, la famiglia delle EfficientNet scala uniformemente su tutte le principali dimensioni della rete come profondità, larghezza e risoluzione utilizzando un cosidetto "coefficiente composto", derivante da determinate considerazioni matematiche. Il risultato è una famiglia di reti che guadagna di circa dieci volte l'efficienza complessiva rispetto al proprio predecessore. Nella figura sottostante si può constatare l'aumentare dell'efficacia delle rete sul dataset ImageNet all'aumentare del numero di parametri in confronto ad altre reti.

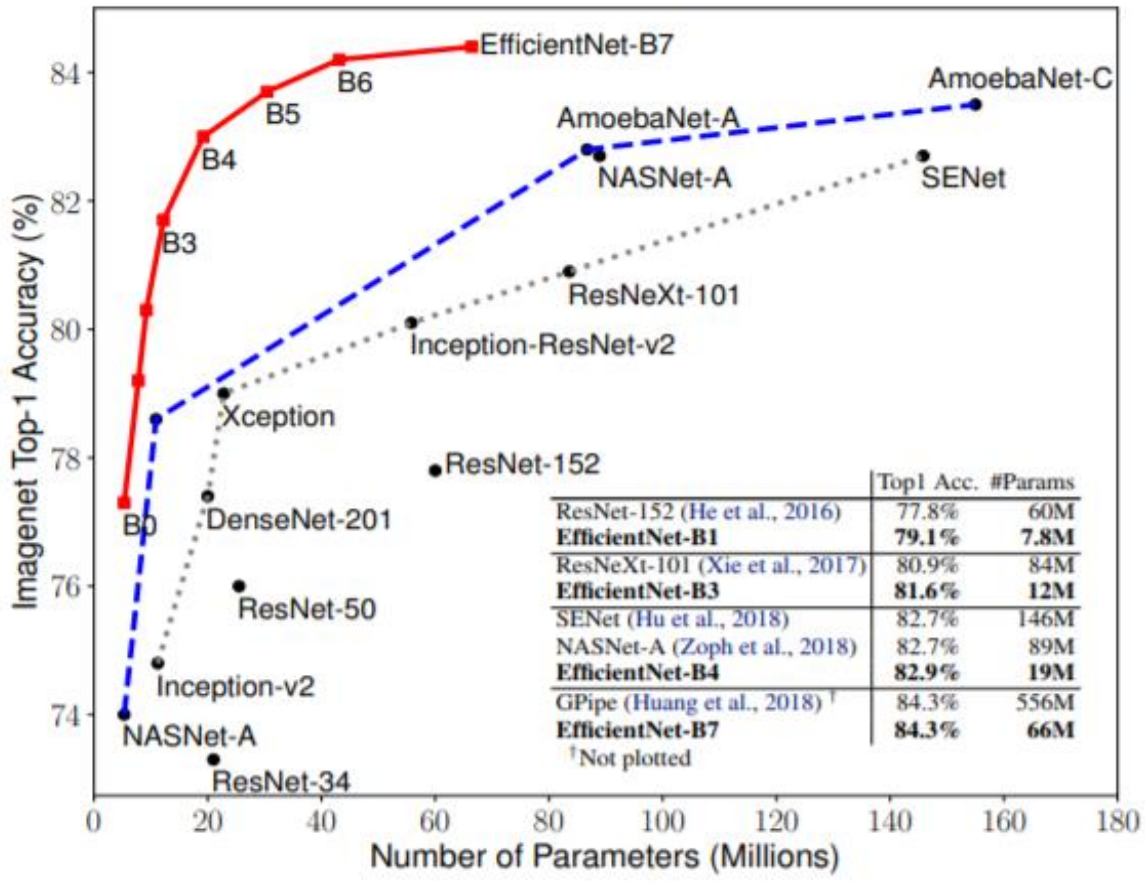


*Figura 3 - Confronto accuracy di reti diverse*

## 1.4 TensorFlow Lite

TensorFlow Lite è un insieme di strumenti che permette agli sviluppatori di rendere utilizzabile un modello TensorFlow su dispositivi mobili, embedded o IoT. Permette di utilizzare on-device il modello creato con TensorFlow a bassa latenza e con dimensioni ridotte. I componenti principali sono:

- *TensorFlowLite converter*: converte modelli TensorFlow per l'utilizzo su dispositivi mobile.

- *TensorFlowLite interpreter*: si occupa del modello TensorFlowLite generato dal convertitore ed è in grado di eseguirlo nelle applicazioni mobile come visibile in figura.

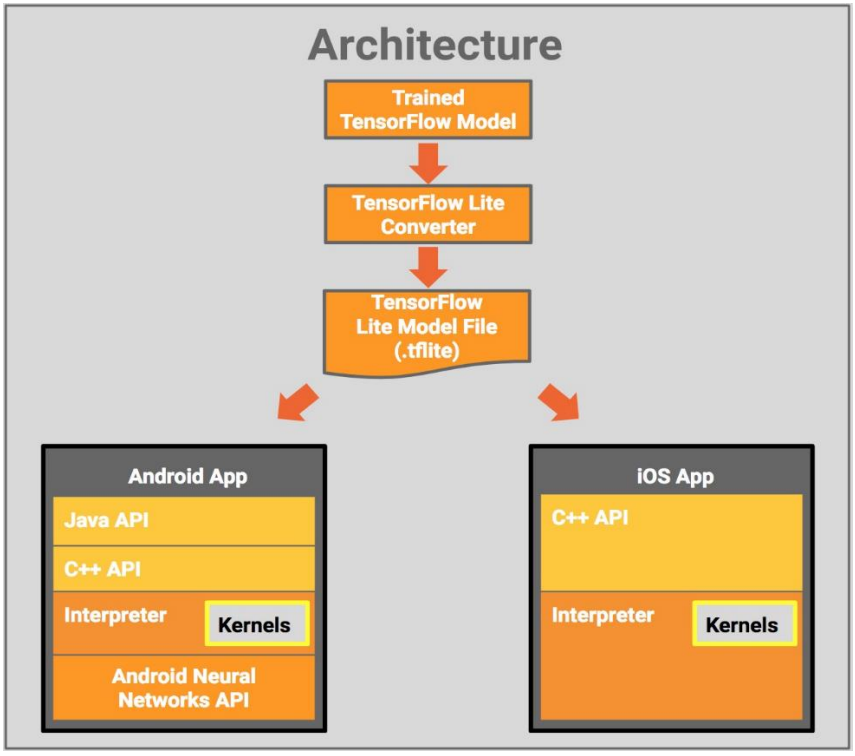


*Figura 4 – Architettura di TensorFlow Lite*

# 2 Dataset

La scelta di un determinato dataset risulta essere, ovviamente, di estrema importanza per ottenere buoni risultati per quanto riguarda la validità di una rete neurale in un contesto reale.

Per scegliere il dataset più adatto è necessario in primo luogo analizzare il problema di classificazione che ci si trova davanti : nel nostro caso la classificazione deve tener conto di “elementi” sparsi in diverse zone dell’immagine.

In rete sono presenti diversi dataset di immagini sul piano assiale di tomografia computerizzata relativi a pazienti affetti dal virus SARS-CoV-2. Molti di questi li abbiamo “testati”, anche con reti diverse, ma la maggior parte non hanno portato a risultati soddifacenti per quanto concerne l’accuratezza in immagini provenienti da dataset diversi da quello utilizzato per l’addestramento.

Partendo da questo presupposto il dataset più appropriato per addestrare la rete definitiva è risultato essere il “SARS-COV-2 Ct-Scan Dataset” poiché le immagini presenti in esso mostrano quasi esclusivamente la parte interna dei polmoni, parte che contiene le informazioni necessarie per la classificazione.

Il “SARS-COV-2 Ct-Scan Dataset” è un dataset pubblico contenente 2482 immagini di tomografia computerizzata, di cui 1252 immagini relative a pazienti positivi al virus SARS-CoV-2 e 1230 immagini CT di pazienti non infettati dal virus SARS-CoV-2. Tali dati sono stati raccolti da 120 pazienti in diversi ospedali della città di San Paolo, in Brasile.

Le immagini sono in formato PNG, non hanno una dimensione standard e anche il livello di contrasto è variabile.

A lato è possibile vedere, in alto due immagini relative a polmoni che presentano segni della malattia in atto, mentre in basso due immagini di TC di pazienti sani.

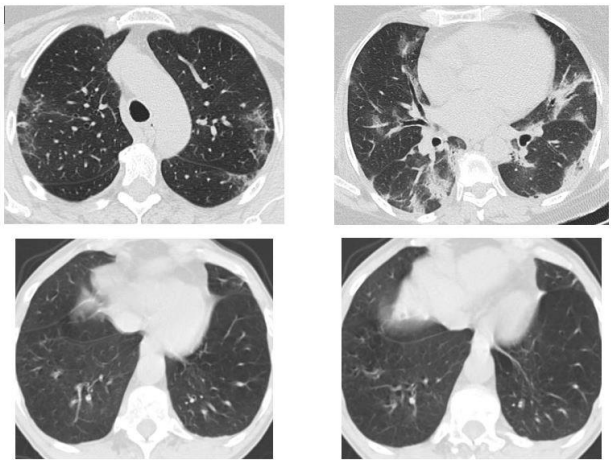

*Figura 5 - esempio di immagini del dataset "SARS-COV-2 CT Scan Dataset"*

# 3 Implementazione della rete

Per l'implementazione della rete sono state utilizzate le API Keras, le quali sono costrutite sopra quelle TensorFlow. La scelta è dovuta ad una maggiore semplicità di utilizzo. Per la scrittura e l'esecuzione del codice sono stati utilizzati Jupyter notebook e Colab per poter eseguire più prove simultaneamente.

Sono state effettuate svariate sperimentazioni modificando la EfficientNetB1, adottando diverse strategie di costruzione, inoltre come già spiegato nel secondo capitolo sono stati sperimentati diversi dataset che hanno fornito risultati interessanti ma non completamente accettabili: seppure le reti risultavano efficienti nel riconoscimento di immagini appartenenti allo stesso dataset, non fornivano risultati soddisfacenti su immagini appartenenti ad altri dataset o sui campioni reali citati nel capitolo di introduzione. Il risultato migliore è stato ottenuto applicando la tecnica di transfer learning e fine-tuning alla EfficientNetB1.

## 3.1 Preparazione dei dati

Prima di poter addestrare la rete è necessario preparare i dati da utilizzare. Il dataset viene prima suddiviso in tre sottocartelle corrispondenti a train,validation e test con relative percentuali di 90%, 5% e 5%. Successivamente le immagini vengono processate da un determinata funzione che effettua la normalizzazione delle immagini tra 0 e 1 e viene effettuato un ridimensionamento a 240 x 240 pixel, poiché la EfficientNetB1 era stata preaddestrata con tali specifiche. Successivamente si definisce la data augmentation che effettua una rotazione casuale tra 0 e 10 gradi e una modifica casuale del contrasto, questo perché mediante un smartphone si potrebbero fare foto che risultano inclinate e perchè le immagini di tomografia computerizzata hanno un livello di contrasto variabile in funzione di come vengono processate dalla macchina che le esegue. Questa operazione viene eseguita sulle immagini in uscita dal primo layer della rete.

## 3.2 Sviluppo della rete

Come accennato nei richiami teorici è comune riutilizzare reti pre-allenate per sfruttarne le capacità pregresse, adattandole al problema in questione. Nell'elaborato è stata scelta la EfficientNet-B1 come intermezzo tra dimensione ed efficienza tra un EfficientNet-B0 e le successive alla EfficientNet-B1. Viene quindi caricata la rete, preallenata sul dataset ImageNet, disponibile su Keras.

È stato poi modificato il "blocco" di output aggiungendo diversi layer in sequenza:

- Layer di *GlobalAveragePooling* che ottiene un valore medio del valore dei pool.
- Layer di *BatchNormalization* che standardizza i valori di input per un apprendimento più veloce in meno epochs.
- Layer di *Dropout* che riduce la possibilità di overfitting.
- Layer *Dense* finale con due classi, le quali corrispondono alle classi COVID e NON_COVID, utlizzando come funzione di attivazione la "softmax", tipicamente usata in problemi di classificazione.

Sono state sperimentate varie combinazioni ed inoltre cambiato l'ordine dei layer soprastanti, escluso il layer Dense. I migliori risultati sono stati ottenuti adottando la sequenza indicata sopra.

L'addestramento consiste nell'utilizzo della coppia transfer learning e fine-tuning, il quale corrisponde in prima battuta al solo allenamento dei soli nuovi layer aggiunti e successivamente allo "scongelamento" di un determinato numero di layer della rete base e al loro allenamento. Nel caso in esame è stato deciso di impostare a $e^{-2}$ il learning rate per il primo allenamento di 25 epochs e poi diminuirlo fino a $e^{-4}$ per allenare per 10 epochs, oltre ai nuovi layer, anche gli ultimi 182 della rete base(circa metà della rete).

Le immagini nella successiva pagina mostrano l'addestramento della rete, la prima immagine relativa alla prima fase di addestramento e la successiva immagine relativa alla seconda fase di addestramento. Come è possibile vedere al termine dell'addestramento si è raggiunta una accuratezza di 97,28% sulle immagini di training e 97,00% sulle immagini di validation.

```
Epoch 5/15
229/229 [==============================] - 77s 337ms/step - loss: 0.1515 - accuracy: 0.9457 - val_loss: 0.1140 - val_
accuracy: 0.9600
Epoch 6/15
229/229 [==============================] - 77s 337ms/step - loss: 0.1290 - accuracy: 0.9491 - val_loss: 0.4240 - val_
accuracy: 0.8600
Epoch 7/15
229/229 [==============================] - 77s 337ms/step - loss: 0.1152 - accuracy: 0.9617 - val_loss: 0.0641 - val_
accuracy: 0.9700
Epoch 8/15
229/229 [==============================] - 77s 337ms/step - loss: 0.1276 - accuracy: 0.9500 - val_loss: 0.0487 - val_
accuracy: 0.9900
Epoch 9/15
229/229 [==============================] - 77s 337ms/step - loss: 0.1085 - accuracy: 0.9576 - val_loss: 0.0865 - val_
accuracy: 0.9800
Epoch 10/15
229/229 [==============================] - 77s 337ms/step - loss: 0.0770 - accuracy: 0.9665 - val_loss: 0.4149 - val_
accuracy: 0.8200
Epoch 11/15
229/229 [==============================] - 78s 340ms/step - loss: 0.0855 - accuracy: 0.9658 - val_loss: 0.1637 - val_
accuracy: 0.9500
Epoch 12/15
229/229 [==============================] - 78s 340ms/step - loss: 0.0819 - accuracy: 0.9672 - val_loss: 0.0473 - val_
accuracy: 0.9800
Epoch 13/15
229/229 [==============================] - 78s 340ms/step - loss: 0.0817 - accuracy: 0.9678 - val_loss: 0.2972 - val_
accuracy: 0.9300
Epoch 14/15
229/229 [==============================] - 78s 341ms/step - loss: 0.0676 - accuracy: 0.9784 - val_loss: 0.0716 - val_
accuracy: 0.9900
Epoch 15/15
229/229 [==============================] - 78s 340ms/step - loss: 0.0719 - accuracy: 0.9728 - val_loss: 0.0892 - val_
accuracy: 0.9700
```

*Figura 6 - Prima fase di addestramento*

```
Epoch 15/25
229/229 [==============================] - 35s 153ms/step - loss: 0.4032 - accuracy: 0.8355 - val_loss: 0.4440 - val_
accuracy: 0.8300
Epoch 16/25
229/229 [==============================] - 35s 153ms/step - loss: 0.4285 - accuracy: 0.8216 - val_loss: 0.3876 - val_
accuracy: 0.8300
Epoch 17/25
229/229 [==============================] - 35s 153ms/step - loss: 0.4153 - accuracy: 0.8297 - val_loss: 0.3188 - val_
accuracy: 0.8200
Epoch 18/25
229/229 [==============================] - 35s 153ms/step - loss: 0.3839 - accuracy: 0.8362 - val_loss: 0.2740 - val_
accuracy: 0.9000
Epoch 19/25
229/229 [==============================] - 35s 152ms/step - loss: 0.3579 - accuracy: 0.8534 - val_loss: 0.3496 - val_
accuracy: 0.8400
Epoch 20/25
229/229 [==============================] - 35s 153ms/step - loss: 0.4011 - accuracy: 0.8249 - val_loss: 0.2450 - val_
accuracy: 0.9200
Epoch 21/25
229/229 [==============================] - 35s 153ms/step - loss: 0.3711 - accuracy: 0.8393 - val_loss: 0.4979 - val_
accuracy: 0.7800
Epoch 22/25
229/229 [==============================] - 35s 152ms/step - loss: 0.3684 - accuracy: 0.8450 - val_loss: 0.3997 - val_
accuracy: 0.8300
Epoch 23/25
229/229 [==============================] - 35s 154ms/step - loss: 0.3954 - accuracy: 0.8307 - val_loss: 0.5993 - val_
accuracy: 0.7800
Epoch 24/25
229/229 [==============================] - 36s 157ms/step - loss: 0.4201 - accuracy: 0.8268 - val_loss: 0.3002 - val_
accuracy: 0.8700
Epoch 25/25
229/229 [==============================] - 36s 157ms/step - loss: 0.4126 - accuracy: 0.8236 - val_loss: 0.3026 - val_
accuracy: 0.8500
```

*Figura 7 - Addestramento finale*

## 3.3 Test

Per verificare la validità della rete creata sono stati eseguiti tre test:

1. Test sulle immagini contenute nel dataset iniziale, ossia il 5%(100 immagini) ottenuto dalla divisione del dataset in fase di processamento.
2. Test su immagini provenienti da un diverso dataset, da cui sono state selezionate casualmente 25 immagini relative a pazienti COVID e 25 immagini relative a pazienti NON_COVID.
3. Test sulle immagini fornite dalla conoscente che lavora in un ospedale della zona, le quali sono 15 immagini di pazienti COVID e 13 di pazienti NON_COVID.

I risutati di tali test sono visibili nelle immagini sottostanti, le quali corrispondono alle "confusion matrix" relative ad ogni test.

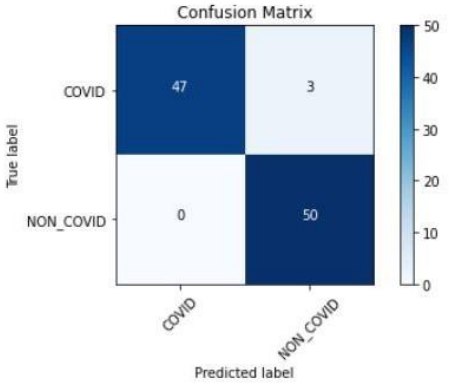


*Figura 9 - Confusion Matrix Test 1*

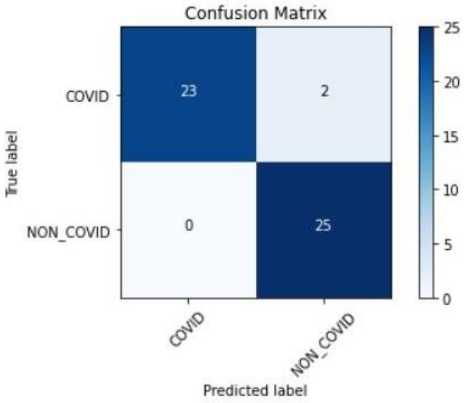


*Figura 8 - Confusion Matrix Test 2*

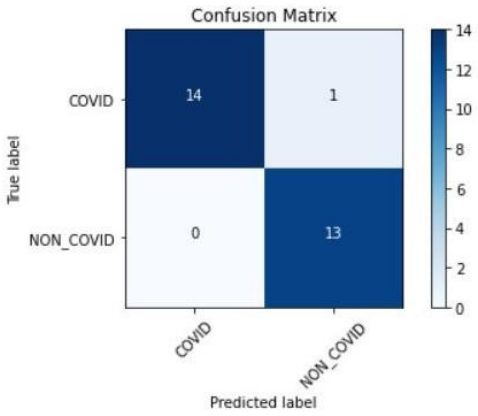


*Figura 10 - Confusion Matrix Test 3*

Dalle immagini si possono riassumere tali risultati : 97% di accuratezza nel test 1 (risultato prevedibile vista la "val_accuracy" ottenuta in fase di addestramento), 96% di accuratezza nel test 2 e 96,43% di accuratezza nel test 3.

# 4 Deploy su smartphone

Successivamente alla definizione e all'addestramento della rete siamo passati allo sviluppo di un'applicazione per smartphone e tablet Android, con la quale poter utilizzare la rete in questione per individuare la presenza del COVID-19 in immagini di tomografia computerizzata.

Prima di passare all'implementazione dell'app abbiamo eseguito la conversione e la quantizzazione della rete neuarale. La prima operazione serve per trasformare il modello addestrato in modello TensorFlow Lite utilizzabile nell'ambiente Android, la seconda operazione dimuisce significativamente le dimensioni del modello e la latenza nella sua esecuzione. La quantizzazione porta ovvi benifici con un piccolo degrado nella precisione del modello. In particolare abbiamo eseguito la quantizzazione di default di TensorFlow Lite, la quale quantizza i pesi da floating point a integer rendendo il modello 4 volte più piccolo in termini di dimensioni.

Passando all'applicazione, questa mette a disposizione dell'utente due modalità diverse per utilizzare la rete:

- Scattare la foto dell'immagine da dare in input alla rete tramite la fotocamera presente nel dispositivo
- Caricando l'immagine da dare in input alla rete direttamente dalla memoria del dispositivo

Queste due funzionalità sono entrambe utlizzabili in seguito alla pressione di un determinato pulsante presente nella schermata principale.

Oltre a ciò abbiamo implementato un semplice storico nel quale sono visibili i risultati della classificazione delle immagini caricate dalla memoria del dispostivo. All'utente è mostrato il nome del file comprensivo del percorso assoluto relativo alla sua posizione al momento del caricamento e il risultato della classificazione dell'immagine. Nuove classificazioni effettutate su immagini già presenti nello storico sovrascrivono il risultato precedente. L'utente può inoltre eliminare le classificazioni che non vuole mantenere nello storico.

L’implementazione è stata fatta utilizzando il linguaggio Kotlin e l’ambiente di sviluppo Android Studio, il quale è basato sul software IntelliJ IDEA dell’azienda JetBrains. L’applicazione è stata sviluppata per funzionare su dispositivi che adottano un livello di API Android uguale o maggiore al livello 28, il quale corrisponde ad una versione Android sucessiva alla 8.


*Figura 11 - Logo Android Studio*

## 4.1 Struttura e interfaccia grafica

Quando si parla di struttura di un’applicazione bisogna essere a conoscenza di quello che è il componente basico di un’applicazione, l’activity.

L’activity (o attività) è il componente fondamentale per l'interazione con l'utente, rappresenta una singola schermata con un'interfaccia utente. Ogni volta che si usa un’applicazione generalmente si interagisce con una o più “pagine” mediante le quali si consultano dati o si immettono input. Le activity nel loro complesso costituiscono il flusso in cui l’utente si inoltra per sfruttare le funzionalità messe a disposizione dall’applicazione.

La nostra applicazione presenta una struttura molto semplice, poiché è composta da due sole activity:

- MainActivity: mostra all’utente i pulsanti con i quali poter scattare una foto o caricare l’immagine dalla memoria e i risultati della classificazione

- HistoryActivity: presenta all’utente la lista delle classificazioni già effettuate sulle immagini presenti in memoria indicando per ognuna il percorso assoluto comprensivo di nome dell’immagine e risultato ottenuto

Abbiamo infine provato l’applicazione per verificarne il corretto funzionamento. Utilizzando il caricamento dell’immagine dalla memoria abbiamo osservato una degradazione non elevata dell’accuratezza, considerata la quantizzazione e il numero non elevato di immagini di testing. Mentre per quanto riguarda l’inferenza mediante un’immagine scattata sul momento dal device, l’accuratezza dipende molto da come viene eseguito lo scatto e dal sensore presente nello smartphone/tablet.

Le immagini sottostanti rappresentano le due activity e ciò che viene mostrato all'utente.


*Figura 12 - Schermata iniziale*

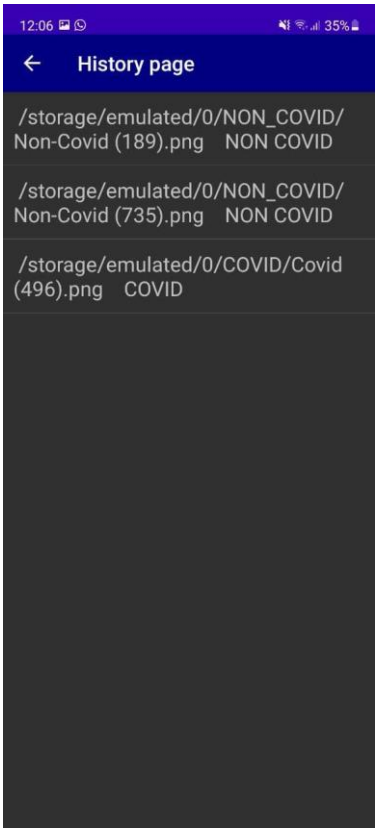


*Figura 13 - Schermata di storico*

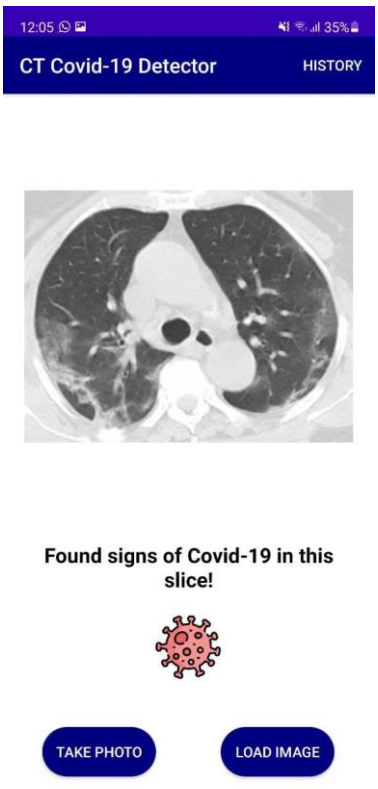


*Figura 15 - Risultato di una classificazione "COVID"*

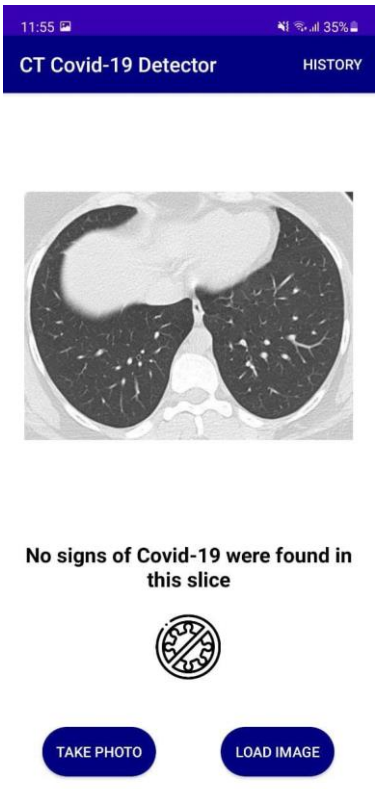


*Figura 14 - Risultato di una classificazione "NON COVID"*

# 5 Conclusioni

In questo progetto è stata definita e sviluppata una rete neurale, utilizzando il modello EfficientNetB1, per il riconoscimento di immagini provenienti da tomografie computerizzate di pazienti sani e di pazienti affetti dal virus SARS-CoV-2. Visto il numero non elevato di immagini presenti nel dataset sono state utilizzate le tecniche di transfer learning e fine-tuning sul modello, addestrando oltre a nuovi layer aggiunti, anche una parte della rete originale. La rete addestrata ha raggiunto ottimi risultati per quanto concerne l'accuratezza in fase di test, anche su immagini di origine diversa, attestandosi in media al 96,5%. Poiché un esame TC di un paziente produce un insieme di immagini sul piano assiale, l'approccio migliore in un constesto reale è utilizzare la rete non su un'unica immagine ma su tutte le immagini dell'insieme.

Successivamente è stata sviluppata un'applicazione per smartphone Android per l'utilizzo della rete neaurale creata, opportuamente convertita e quantizzata. L'applicazione sviluppata permette di scattare foto alle immagini TC, caricare immagini presenti in memoria e mantiene uno storico delle classificazioni fatte. L'opzione che offre risultati migliori nella classificazione è quella di caricare le immagini dalla memoria del dispositivo.

Il lavoro fatto mostra il potenziale dell'utilizzo del Deep Learning in ambito medico, in particolare nel riconoscimento dei segni caratteristici della malattia COVID-19 nei polmoni del paziente.