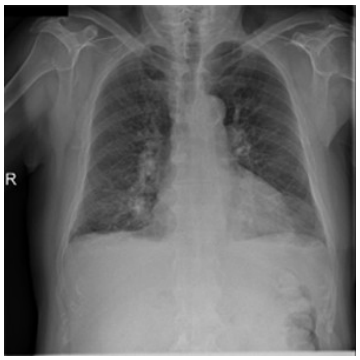
Relazione sul progetto di Machine Learning

L’obiettivo del progetto è stato quello di riuscire a classificare immagini di radiografie polmonari in modo da distinguere in modo automatico tra:

- Polmoni normali
- Polmoni di un malato affetto da covid
- Polmoni di un malato affetto da polmonite
- Polmoni di un malato affetto da opacità polmonare

Covid



Polmonite

Normale

Opacità polmonare

Analisi del dataset

Il dataset si chiama ‘COVID-19 Radiography Dataset’ ed è stato scaricato dalla piattaforma Kaggle al seguente link: https://www.kaggle.com/tawsifurrahman/covid19-radiography-database .
Il dataset contiene 21165 immagini, tutte di dimensione 299x299 pixel, organizzate in quattro cartelle rappresentanti le quattro classi di interesse ossia *Covid*, *Lung_Opacity*, *Normal* e *Viral Pneumonia*. Ogni cartella ovviamente contiene tutte le immagini della stessa classe determinata dal nome della cartella stessa, in particolare si hanno 3615 immagini appartenenti alla classe *Covid*, 10192 alla classe *Normal*, 6012 alla classe *Lung_Opacity* e 1354 alla classe *Viral Pneumonia*.

Il dataset risulta essere sbilanciato, per cui abbiamo scelto di utilizzare la tecnica di **data augmentation** durante il training per bilanciare il numero di immagini tra le varie classi e abbiamo confrontato i risultati che si ottengono utilizzando il dataset originale.
Visualizzando il dataset abbiamo scelto di utilizzare diverse tecniche di preprocessamento volte a migliorare il contrasto. In particolare, prendendo spunto dall'articolo *'Exploring the effect of image enhancement techniques on COVID-19 detection using chest X-ray images'* abbiamo scelto di utilizzare e confrontare le tecniche: **histogram equalization**, **contrast limited adaptive histogram equalization**, **image invert/complement**.

Modelli di classificazione utilizzati

MLP
Il primo modello di classificazione scelto è stata una semplice rete MLP definita da noi. A tal fine abbiamo sfruttato la libreria **Keras** che consente di creare un modello passando la lista dei layers desiderati. La nostra MLP è costituita semplicemente da tre layers fully connected, come funzione di attivazione degli hidden layers abbiamo scelto la **LeakyRelu** con parametro alfa=0.1 e dato che sperimentalmente si dimostra prona all'overfitting la accompagniamo alla tecnica del **dropout** con rate pari a 0.2, infine come optimizer abbiamo scelto di usare **adam.** Abbiamo passato in input a tale modello il dataset originale e ottenuto il 64,68% di accuracy sul test.

ResNet50
Abbiamo scelto di utilizzare la tecnica del transfer learning e successivamente del fine tuning sfruttando ResNet50, utilizzando come partenza i pesi ottenuti dall'addestramento di tale rete sul dataset ImageNet.
Per poter utilizzare ResNet50 abbiamo dovuto preprocessare opportunamente l'input in quanto le immagini devono essere di dimensione 224x224x3 e devono essere convertite da RGB a BGR. Per il ridimensionamento abbiamo definito una funzione ad hoc mentre per la conversione abbiamo sfruttato la funzione *preprocess_input* fornita dal modulo resnet50 di Keras. Il modello è stato creato a partire dal modello base resnet50 (con 175 layers in totale), fornito proprio da Keras, di cui non sono stati considerati solo gli ultimi layer poiché sono finalizzati agli obiettivi di classificazione di ImageNet, il resto dei layers invece viene congelato impedendone così l'aggiornamento durante l'addestramento e infine vengono aggiunti dei nuovi layers, ossia un layer di tipo *Flatten* per favorire la connessione con l'ultimo layer della rete preaddestrata e un layer di tipo *Dense* per eseguire la classificazione tra le 4 classi che sono di nostro interesse, come optimizer abbiamo scelto **adam** e

come metrica la **accuracy**. Per l'addestramento si considerano batch costituiti da 32 immagini e si eseguono 5 epoche. Le prestazioni sono risultate buone poiché abbiamo ottenuto un' accuracy del 96,84% nel training e dell' 87,50% nel test. Abbiamo scelto poi di procedere con la tecnica del fine tuning dove nel processo di training si aggiustano i pesi degli ultimi layer così da specializzare ancora di più il modello al nostro obiettivo. In particolare abbiamo scelto di addestrare dal centesimo layer in su mantenendo congelati i layers sottostanti, quindi si continua l'addestramento a partire da quanto ottenuto a seguito del transfer learning eseguendo ulteriori cinque epoche. Il risultato ottenuto con il fine tuning risulta peggiorato dato che abbiamo ottenuto un' accuracy del 95,11% nel training e dell' 80,98% nel test.

MobileNet

Abbiamo scelto di applicare la tecnica del transfer learning e successivamente del fine tuning anche considerando come rete di partenza MobileNet, utilizzando i pesi ottenuti dall'addestramento di tale rete sempre sul dataset ImageNet. Per poter utilizzare MobileNet è stato sempre necessario preprocessare opportunamente l'input, in particolare la rete deve ricevere come input immagini di dimensione 224x224x3 e i pixel devono assumere valori nell'intervallo [-1, +1] e non da 0 a 255 come avviene per le immagini del dataset. Il modello è stato creato a partire dal modello base MobileNetV2 (con 154 layers in totale) fornito da Keras di cui non sono stati considerati solo gli ultimi layer poiché sono finalizzati agli obiettivi di classificazione di ImageNet, il resto dei layers invece viene congelato impedendone così l'aggiornamento durante l'addestramento. Questo modello base si comporta quindi come un estrattore di featues, in particolare estrae dalle immagini 224x224x3 in blocchi 7x7x1280 di features. A partire dal blocco di features abbiamo aggiunto un layer di tipo *GlobalAveragePooling2D*, che consente di fare una media in una finestra 5x5, al fine di generare le predizioni finali poichè ci consente di convertire le features in un singolo vettore di 1280 elementi per immagine. Infine abbiamo aggiunto un layer di tipo *Dense* per convertire queste features in una singola predizione con funzione di attivazione **relu**, come optimizer abbiamo scelto **adam** e come metrica la **accuracy**. Per l'addestramento si considerano batch costituiti da 32 immagini e si eseguono 5 epoche. Passando in input a tale modello il dataset originale abbiamo ottenuto un' accuracy del 64,89% sul test. Abbiamo proseguito con la tecnica del fine tuning con lo stesso approccio utilizzato per ResNet50 e abbiamo ottenuto un'accuracy del 16,47% sul test.

Xception
Abbiamo scelto di utilizzare la tecnica del transfer learning e successivamente del fine tuning anche considerando Xception utilizzando sempre i pesi ottenuti dall'addestramento sul dataset ImageNet. Per poter utilizzare Xception è stato necessario eseguire un preprocessamento analogo a quanto descritto precedentemente per MobileNet. Il modello è stato creato a partire dal modello base Xception (con 132 layers in totale) fornito da Keras, di cui non sono stati considerati solo gli ultimi layer poiché sono finalizzati agli obiettivi di classificazione di ImageNet, il resto dei layers invece viene congelato impedendone così l'aggiornamento durante l'addestramento. Questo modello base si comporta come un estrattore di featues a seguito del quale abbiamo aggiunto un layer di tipo *Dense* per convertire queste features in una singola predizione con funzione di attivazione **relu**, come optimizer abbiamo scelto **adam** e come metrica la **accuracy**. Per l'addestramento si considerano batch costituiti da 32 immagini e si eseguono 5 epoche. Passando in input a tale modello il dataset originale abbiamo ottenuto un' accuracy del 83,06% sul test, abbiamo poi proseguito con la tecnica del fine tuning ottenendo un' accuracy del 91,87%.

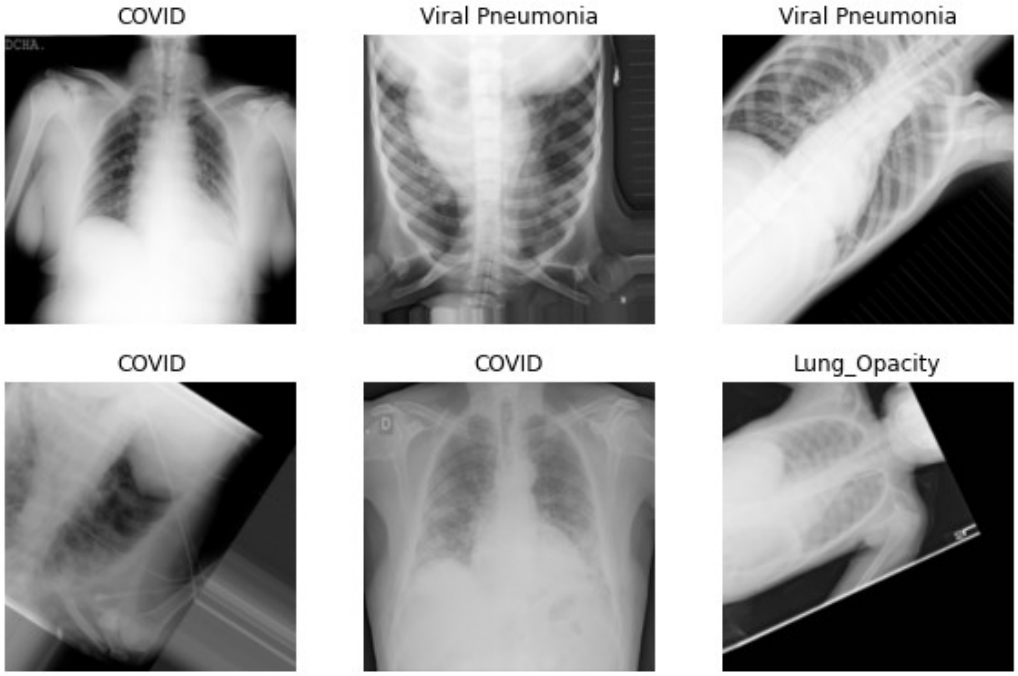
Risultati delle esecuzioni

Per ogni modello presentato precedentemente sono stati considerati separatamente cinque dataset come input:

1. dataset originale
2. dataset con data augmentation
3. dataset con HE
4. dataset con CLAHE
5. dataset con image invert/contrast

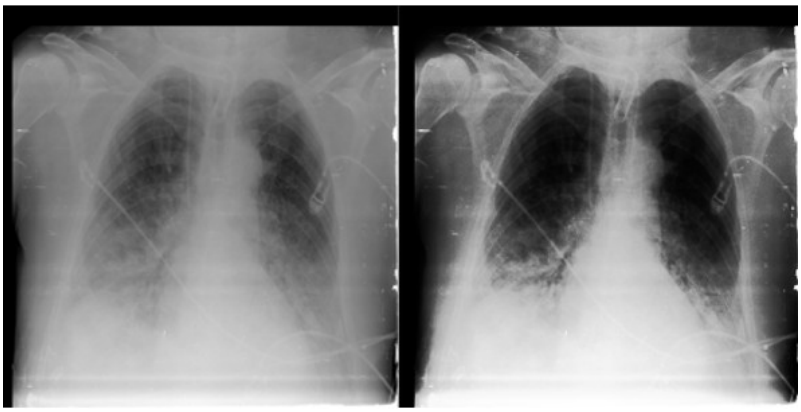
Descriviamo ora brevemente come abbiamo realizzato a partire dal dataset originario gli altri quattro dataset dopo di che riportiamo in una tabella i risultati ottenuti a seguito di tutte le esecuzioni.

Per realizzare l’aumento di dati abbiamo utilizzato **ImageDataGenerator** offerto da keras. Abbiamo scelto come trasformazioni da applicare randomicamente: la rotazione in un intervallo di 180 gradi e la traslazione sia orizzontale sia verticale di al massimo 100 pixels.
L’immagine sottostante riporta alcuni esempi di immagini ottenute applicando le trasformazioni suddette.

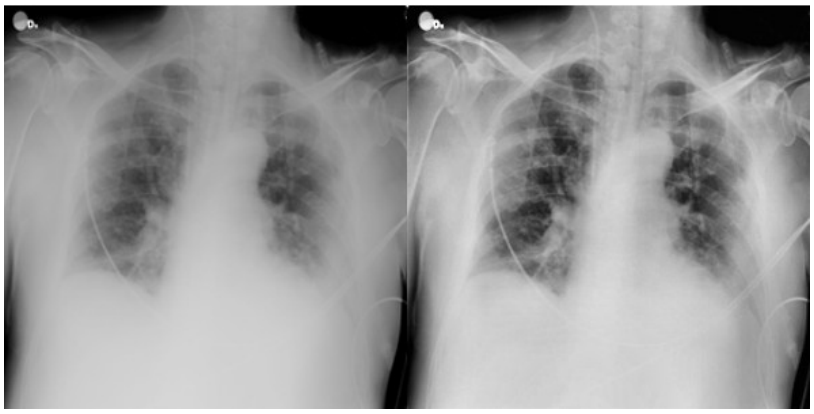


Per eseguire le tecniche di miglioramento del contrasto abbiamo sfruttato la libreria **opencv.**

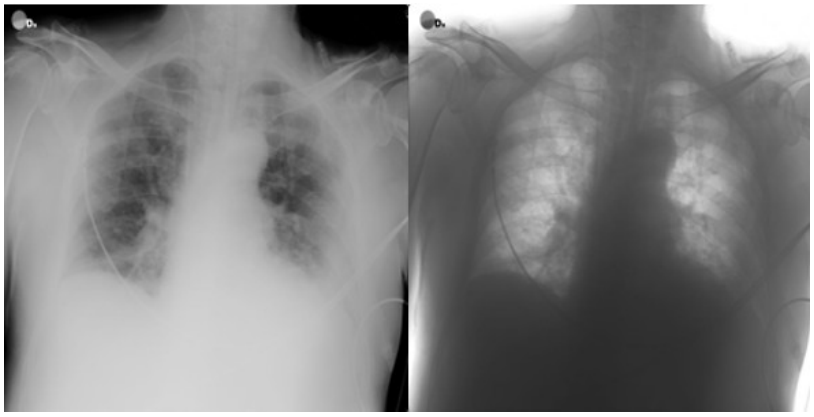
Per **HE** abbiamo utilizzato la funzione *equalizeHist.*
A sinistra è riportata un’ immagine del dataset originale e a destra la stessa immagine a cui è stata applicata la tecnica di miglioramento del contrasto HE.



Per **CLAHE** abbiamo utilizzato la funzione *createCLAHE*.
A sinistra è riportata un' immagine del dataset originale e a destra la stessa immagine a cui è stata applicata la tecnica di miglioramento del contrasto CLAHE.



Per l'esecuzione dell'**inversione delle immagini** abbiamo sfruttando la funzione *cv.bitwise_no*.
A sinistra è riportata un' immagine del dataset originale e a destra la stessa immagine invertita.



La tabella sottostante mostra le prestazioni dei vari modelli a seconda del dataset in input in termini di accuracy ottenuta nel test.

	MLP	RESNET50 TRANSFER LEARNING 5 epoche	RESNET50 FINE TUNING 100 5 epoche	MOBILENET TRANSFER LEARNING 5 epoche	MOBILENET FINE TUNING 100 5 epoche	XCEPTION TRANSFER LEARNING 5 epoche	XCEPTION FINE TUNING 100 5 epoche
ORIGINALE	64,68%	87,50%	80,98%	59,93%	16,47%	83,06%	91,87%
DATA AUGMENTATION	66,10%	89,95%	92,88%	81,99%	16,46%	85,87%	28,46%
HE	62,49%	89,27%	91,02%	79,65%	16,46%	76,73%	93,64%
CLAHE	66,99%	90,41%	91,05%	82,09%	16,47%	60,03%	16,47%
IMAGE INVERT	65,91%	88,73%	92,89%	64,89%	16,47%	38,97%	16,47%

Abbiamo notato un peggioramento dell'accuracy a seguito del fine tuning per il modello MobileNet e ciò indipendentemente dal dataset in input, dunque abbiamo deciso di indagare come cambia l'accuratezza di tale rete a seconda del numero di strati che lasciamo 'congelati'.
Con il modello Xception invece abbiamo deciso di indagare come varia l'accuracy all'aumentare del numero di epoche, confrontando i risultati ottenuti con una singola epoca da quelli ottenuti eseguendo 5 epoche.

Nell' ultima tabella è riportata l'accuracy sul test avendo lasciato congelati i primi 100, 80 e 120 layers per il modello MobileNet ed eseguendo 1 e 5 epoche per il modello Xception.

	MOBILENET FINE TUNING 100	MOBILENET FINE TUNING 80	MOBILENET FINE TUNING 120	XCEPTION TRANSFER LEARNING 1 epoca	XCEPTION TRANSFER LEARNING 5 epoche	XCEPTION FINE TUNING 1 epoca	XCEPTION FINE TUNING 5 epoche
ORIGINALE	16,46%	16,46%	16,46%	77,79%	83,06%	16,47%	91,87%
DATA AUGMENTATION	16,46%	82,11%	47,20%	83,67%	85,87%	89,96%	28,46%
HE	16,46%	64,87%	16,46%	78,14%	76,73%	89,96%	93,64%
CLAHE	16,47%	16,47%	16,47%	78,79%	60,03%	89,96%	16,47%
IMAGE INVERT	16,46%	16,46%	16,46%	72,43%	38,97%	16,47%	16,47%

Conclusioni

Miglior risultato
La migliore accuracy in assoluto (93,64%) l'abbiamo ottenuta con il modello Xception addestrato eseguendo fine tuning, in particolare mantenendo congelati i primi 100 layers ed eseguendo 5 epoche, utilizzando come input il dataset a cui è stata applicata la tecnica di miglioramento del contrasto CLAHE.

Confronto tra tecniche di miglioramento del contrasto
In generale la tecnica CLAHE si dimostra migliore rispetto alla tecnica HE.

Confronto tra immagini originali e immagini invertite
Confrontando i risultati ottenuti considerando come input il dataset originale e il dataset a cui è stata applicata l'inversione delle immagini, abbiamo notato che quest'ultimo consente di avere un' accuracy migliore ad eccezione del modello Xception con cui invece si ottengono prestazioni migliori o analoghe utilizzando il dataset originale.

Data augmentation
L'aumento dei dati di training, volto a mitigare lo sbilanciamento tra classi, consente sempre di migliorare le prestazioni tranne che con il modello Xception quando si esegue fine tuning con 5 epoche.

Problemi riscontrati con il fine tuning
Durante le varie esecuzioni abbiamo notato che si presenta ripetutamente il valore di accuracy 16,47% indipendentemente dal dataset in input, dal numero di layers che lasciamo congelati, dal numero di epoche e anche dal modello considerato.