

XRay Classification

Robin Faro, Giulio Sichili, Alessandro Strano

1. Problema

1.1. Scelta del problema

La scelta del problema da affrontare nel nostro progetto è stata sicuramente guidata dalla volontà condivisa di realizzare un prototipo di qualcosa che potesse almeno far intravedere le immense potenzialità del machine learning nelle scienze mediche, campo nel quale l'impatto di questa disciplina è in continua crescita. [Javaid et al. \(2022\)](#) Certamente consci della nostra inesperienza, abbiamo valutato possibili applicazioni restringendo il campo a quelle che sapevamo di poter realizzare con le nostre conoscenze, perciò una prima scelta è stata quella di cercare un task di classificazione di immagini mediche. A questo punto, da alcune ricerche abbiamo constatato che per vari motivi, come la relativa disponibilità di immagini e facilità di reperimento di esperimenti pregressi, la classificazione di immagini da lastre a Raggi-X del torace per la diagnosi di malattie polmonari potesse essere un problema alla nostra portata ma allo stesso tempo avvincente (e utile!).

1.2. Formalizzazione

Tenendo conto che le specificità del Dataset utilizzato e relativi esempi saranno trattati in [Dataset](#), enunciamo formalmente il problema affrontato nella sua generalità di classificazione di immagini.

Consideriamo il dataset D costituito dagli esempi

$$X = [x^{(1)}, x^{(2)}, x^{(3)}, \dots x^{(m)}]$$

e dalle etichette di groundtruth

$$Y = [y^{(1)}, y^{(2)}, y^{(3)}, \dots y^{(m)}],$$

in cui ogni esempio $x^{(i)}$ è una matrice di dimensione $H \times L \times C$ (nel nostro caso le immagini sono quadrate e in scala di grigi, per cui $H = L$ e $C = 1$), e $y^{(i)} \in Classi = \{0, 1, 2\}$.

Nella nostra applicazione le classi 0, 1 e 2 indicano rispettivamente un soggetto sano, uno che è stato colpito dal Covid-19 e uno da una polmonite batterica.

Il modello M dovrà eseguire il task di predizione di una classe per ogni pattern di test in input e naturalmente sarà valutato in base alla sua capacità di far corrispondere la predizione all'etichetta di groundtruth secondo una o più metriche di performance P :

$$P(M(X), Y).$$

Per l'analisi dettagliata delle componenti D , M e P rimandiamo alle rispettive sezioni.

1.3. Contesti applicativi

Un classificatore come quello implementato è in grado fornire un'indicazione di massima sulla probabilità che il paziente di una radiografia presenti una data patologia, (o l'abbia sviluppata in passato), ma è compito del medico stabilire la diagnosi, nei casi in cui la predizione risulti incerta. Un'ipotesi applicativa consiste nell'ipotizzare, con l'aiuto di un comitato di esperti, una soglia di certezza p_c al di sopra della quale la predizione di soggetto sano è considerata sicura, e passare al vaglio umano solo le predizioni più incerte, con $p_{pred} < p_c$. Approcci di questo tipo possono accelerare i tempi diagnostici consentendo un'identificazione precoce di malattie e condizioni. Inoltre, l'automazione fornita dai modelli di machine learning può ridurre l'onere sui medici, consentendo loro di concentrarsi su compiti più complessi e critici e migliorando l'efficienza generale dell'assistenza sanitaria.

2. Dataset

Una volta scelto il topic che avremmo voluto affrontare, la ricerca di un dataset quanto migliore possibile è stato il primo problema con cui ci siamo dovuti interfacciare.

2.1. Why is it so difficult?

La difficoltà di trovare una banca dati ampia di immagini XRay su pazienti affetti da Polmonite, Covid o in buona salute è basata sui seguenti fattori:

1. Disponibilità limitata dei dati: l'accesso ad un ampio e diversificato dataset di immagini di raggi X può essere limitato a causa delle restrizioni sulla condivisione e l'accesso a dati medici sensibili. Gli ospedali e le strutture sanitarie devono aderire a rigorosi standard di sicurezza dei dati: le informazioni mediche sono altamente riservate e devono essere protette per garantire la privacy dei pazienti. Tutto ciò rende difficile la reperibilità di un dataset di immagini di raggi X ampio e accessibile.
2. Risorse limitate per l'etichettatura: la creazione di un dataset etichettato richiede l'intervento di medici qualificati per confermare la presenza o l'assenza di polmonite o COVID-19 nelle immagini di raggi X. Tali esperti potrebbero non essere facilmente disponibili o potrebbero richiedere un considerevole investimento di tempo e risorse per l'etichettatura di un così grande numero di immagini.

- Variabilità delle condizioni dei pazienti: ottenere un dataset che rappresenti una vasta gamma di condizioni dei pazienti può essere complesso. Le immagini di raggi X possono variare in base a diversi fattori come l'età dei pazienti, lo stadio della malattia e altre variabili cliniche. Raccogliere una varietà sufficiente di casi positivi per COVID-19 e polmonite, insieme a casi normali, richiede una difficile collaborazione tra diverse istituzioni sanitarie.
- Qualità e standardizzazione delle immagini: le immagini di raggi X possono variare notevolmente in termini di qualità, risoluzione e standard di acquisizione. La presenza di immagini di scarsa qualità o acquisite con diverse tecniche può influire sulla capacità di addestrare un modello di machine learning in modo accurato e di conseguenza sulla sua capacità di generalizzazione a nuovi dati, rendendo l'allenamento vano.

2.2. First and Second Dataset

Dopo una lunga ricerca siamo riusciti a trovare una banca dati che pensavamo potessere fare a caso nostro. Ci offriva 5861 immagini XRay, suddivise in due cartelle Train e Test, a loro volta suddivise in Normal e Pneumonia. Ciascun elemento aveva nel nome l'etichetta ed erano le seguenti: normal, bacteria e virus. Qui la distribuzione delle classi:

	Normal	Bacterial	Viral
Train	1249	2596	1256
Test	230	340	190

Table 1: Distribuzione del primo dataset

Nonostante la disposizione delle cartelle suggerisse una classificazione binaria (Normal vs Pneumonia), abbiamo tentato una classificazione a tre classi: Normals vs Bacteria vs Virus. Come mostrato in [11](#) i risultati rendevano esplicita la polarizzazione del dataset verso la classe Bacteria, che da sola occupava il 50% del totale.

Così come descritto in [Esperimenti](#), dopo i risultati ottenuti siamo andati alla ricerca di una nuova raccolta dati che potesse fare al caso nostro. Abbiamo recuperato un'unica repository di 5228 immagini, in cui anche qui ciascun elemento ha nel nome l'etichetta corrispondente: Normal, Covid o Pneumonia.

Qui la distribuzione delle classi:

	Normal	Covid	Pneumonia
	1802	1626	1800

Table 2: Distribuzione del secondo dataset

Quando le classi sono sbilanciate, la classe dominante può influenzare il processo di apprendimento, portando il modello ad essere più incline a predire quella classe con maggiore frequenza. Per questo, nonostante il secondo dataset abbia un numero minore di immagini complessive, il suo bilanciamento delle classi ha contribuito al miglioramento delle prestazioni del modello rispetto al primo dataset.

2.3. Deep into the dataset

Tutte le immagini sono state raccolte in un'unica cartella e poi suddivise a runtime con il metodo `train_test_split` della libreria `sklearn`, suddividendo l'intero set in Train(60%), Validation(20%) e Test(20%).

Qui a seguire degli esempi di immagini della banca dati:



Figure 1: Due casi di Normal



Figure 2: Due casi di COVID-19

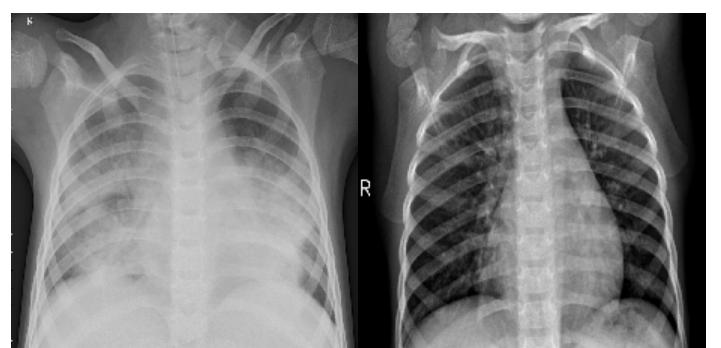


Figure 3: Due casi di Pneumonia

È importante sottolineare che, ad occhio nudo e senza l'esperienza medica necessaria, riconoscere con precisione la presenza di casi normali, COVID-19 o polmonite in immagini di raggi X può essere estremamente difficile, se non impossibile. Queste condizioni possono manifestarsi con sintomi e caratteristiche radiologiche che possono sovrapporsi

e confondersi visivamente, persino agli occhi di un osservatore esperto. Ad esempio, sia i casi di COVID-19 che di polmonite possono presentare opacità, infiltrati o consolidamenti polmonari. Queste caratteristiche visive possono essere simili, rendendo difficile una distinzione accurata solo tramite l'osservazione (Figura 2 sx - Figura 3 sx). Le immagini di raggi X possono variare notevolmente da paziente a paziente e da fase di malattia a fase di malattia. Pertanto, non è sempre garantito che i segni tipici di polmonite o COVID-19 siano presenti in modo evidente nelle immagini e potrebbero essere confusi con casi normali ad occhio nudo (Figura 2 dx - Figura 3 dx). Questa sfida nella diagnosi visiva evidenzia l'importanza dei modelli di machine learning come strumento di supporto per il processo di screening e diagnostica, poiché possono apprendere dei pattern nascosti nei dati, identificando caratteristiche sottili e complesse che potrebbero sfuggire all'occhio umano. È importante sottolineare che i modelli di machine learning non sostituiscono l'esperienza e il giudizio clinico dei medici, ma debbono agire come strumenti di supporto per migliorare la precisione e l'efficienza della diagnosi. L'approccio combinato tra l'esperienza medica e modelli di machine learning può portare a risultati più affidabili e consentire una migliore e più tempestiva assistenza sanitaria.

3. Metodi

In questa sezione mostriamo i metodi utilizzati per attaccare il problema, e specifichiamo:

1. Il modello utilizzato, le ragioni della sua scelta e le sue specifiche tecniche;
2. Il modo in cui le immagini del dataset vengono processate prima di essere fornite in input alla rete;
3. I metodi di allenamento del modello, e gli approcci di Transfer-Learning utilizzati

3.1. Scelta del modello

Il modello M da utilizzare nel contesto della classificazione di immagini, innanzitutto, doveva essere una Rete neurale convoluzionale (CNN), per diverse ragioni che la rendono ideale per questo tipo di task:

- Vantaggi intrinseci della convoluzione (o meglio, cross correlazione) rispetto al prodotto matriciale:
 - Il numero di parametri di ogni livello dipende dalla scelta della forma del kernel, e non dal numero di features in input;
 - Invarianza rispetto alla traslazione orizzontale e verticale;
 - Riduzione del numero di parametri (vedi Connessioni sparse);
- Connessioni sparse: le connessioni tra input e output di un livello, non sono del tipo fully-connected, ma rappresentano il cosiddetto receptive field di un'unità, che è limitato e riduce drasticamente il numero di parametri per livello;

- Possibilità di ottimizzare la convoluzione con tecniche di separazione lineare;
- Tecniche di pooling che riducono la dimensione preservando le caratteristiche principali delle feature e sono anch'esse robuste rispetto alla traslazione.

Nell'ambito delle CNNs, un punto di riferimento per le prestazioni raggiunte è l'architettura ResNet (Residual Network). I vantaggi e gli svantaggi della scelta di questo modello si possono riassumere dicendo che:

- La profondità della rete le permette di ottenere buone performance su svariate tipologie di dataset, generando attivazioni relative a pattern via via più complessi man mano che si scende nei layer più profondi.
- La capacità del modello può essere un arma a doppio taglio, perché ci espone al rischio di overfitting.

In questa prospettiva, un elemento determinante che ci ha spinto a scegliere ResNet è la disponibilità di diverse versioni della stessa, al variare della profondità. Abbiamo scommesso sul fatto che ResNet34 o ResNet18 fossero delle alternative che potessero offrirci un buon compromesso tra un'ottima capacità di apprendere i pattern, e una profondità limitata che arginasse il rischio di overfitting; inoltre, avere due alternative simili ci permetteva di effettuare da subito un confronto tra le due e scegliere la migliore.

Da questo esperimento preliminare, è apparso evidente che ResNet34 aveva una capacità esagerata rispetto al nostro bisogno, e che oltre a provocare overfitting aveva dei tempi di allenamento considerevolmente più lunghi.

3.2. Image preprocessing

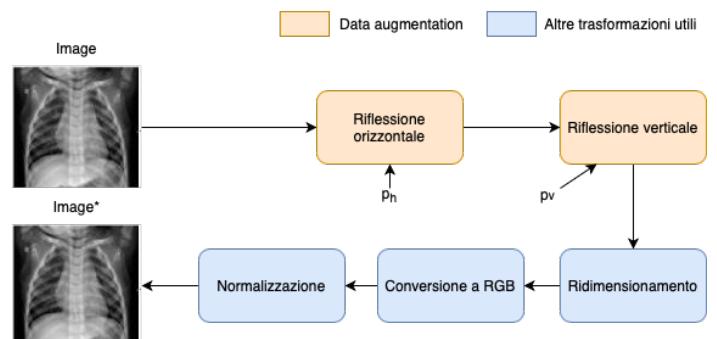


Figure 4: Schema delle trasformazioni a cui viene sottoposta l'immagine in input $Image$. Chiamiamo l'immagine trasformata $Image^*$.

Affinchè la ResNet sia in grado di apprendere efficacemente dal dataset, è necessario effettuare alcune operazioni sulle immagini:

- Trasformazioni tese alla Data Augmentation:
 - Riflessione orizzontale, con probabilità p_h ;
 - Riflessione verticale, con probabilità p_v .

Queste sono utili per aumentare la variabilità all'interno del dataset, costringendo il modello ad imparare a riconoscere anche le immagini ribaltate rispetto a quelle originali. Osserviamo inoltre che la CNN sarà in grado di riconoscere particolari pattern a prescindere dalla posizione all'interno della figura e, con queste trasformazioni, anche a prescindere dal fatto che siano ribaltati o meno.

- Trasformazioni necessarie per la corretta elaborazione:
 - Resizing, a una dimensione di 224x224 pixel;
 - Conversione da scala di grigi a RGB, perché la rete è progettata per lavorare su 3 canali;
 - Normalizzazione, utile per garantire una certa omogeneità dei pesi durante il training.

3.3. Deep into ResNet18

La caratteristica che distingue le Residual Network da quelle che vengono chiamate "Plain Networks" è la presenza di shortcuts, ovvero delle connessioni tra livelli non contigui.

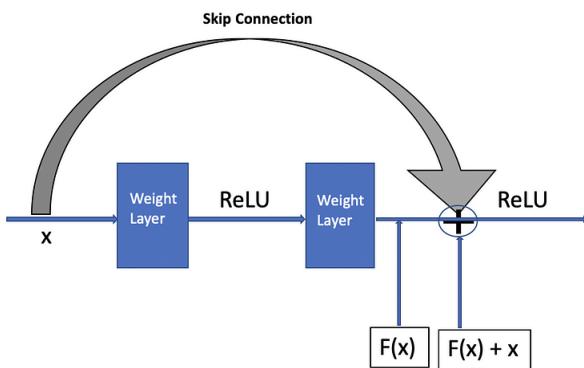


Figure 5: Blocco residuale della ResNet, con shortcut, chiamata anche *skip connection*

In figura 5 viene mostrato uno schema semplificato di una coppia di livelli $l, l+1$ costituenti di un blocco residuale della ResNet. Consideriamo l'attivazione $a^{(l)}$ in input al livello l , e teniamo presente che la funzione di attivazione presente alla fine di ogni layer è la ReLU, che indicheremo con h . La composizione di operazioni è quella standard fino al termine $z^{(l+2)}$:

$$\begin{aligned} z^{(l+1)} &= W^{(l+1)}a^{(l)} + b^{(l+1)} \\ a^{(l+1)} &= h(z^{(l+1)}) \\ z^{(l+2)} &= W^{(l+2)}a^{(l+1)} + b^{(l+2)}; \end{aligned}$$

La differenza tra un blocco residuale e un normale insieme di livelli di una CNN standard sta nel fatto che la funzione di attivazione del secondo layer di un blocco residuale viene calcolata sulla somma tra l'input del livello precedente e il termine calcolato dal livello stesso:

$$a^{(l+2)} = h(z^{(l+2)} + a^{(l)}).$$

Questo meccanismo si è dimostrato efficace nel combattere il cosiddetto vanishing dei gradienti He et al. (2016), cioè la tendenza ad annullarsi dei gradienti delle reti convoluzionali profonde, specialmente quelli dei livelli iniziali, che nella fase di backward propagation vengono aggiornati per ultimi, dopo svariati prodotti con valori spesso minori di 1.

Il concetto di residuo, infatti, sta proprio nella capacità della rete di preservare l'informazione attraverso i blocchi residuali, replicando di fatto la funzione identità piuttosto che vanificare i gradienti precedentemente calcolati (infatti la ReLU verrà applicata a un valore positivo anche quando $z^{(l+2)}$ sia prossimo a 0):

$$\lim_{z^{(l+2)} \rightarrow 0} h(z^{(l+2)} + a^{(l)}) = h(a^{(l)}) = a^{(l)}$$

In figura 6 troviamo uno schema della ResNet18. Eccetto il layer di input e il classificatore di output, tutti i livelli nascosti vengono raggruppati per formare blocchi residuali. Questi livelli sono tutti costituiti da un numero di kernel convoluzionali che raddoppia ogni 4 layer (e si mantiene dunque costante tra il livello di partenza e di arrivo di ogni shortcut). Dopo ogni macroblocco di 4 layer è presente un pooling con stride 2, che dimezza la dimensione della feature map. Il prodotto $\text{Dim}_{f.map} \times \text{Num}_\text{kernels}$ si mantiene quindi costante, scelta motivata dal voler ottenere la stessa complessità computazionale per tutti i livelli. Osserviamo inoltre che una certa omogeneità dimensionale all'interno dei singoli blocchi residuali è necessaria affinché sia possibile effettuare la somma $z^{(l+2)} + a^{(l)}$. L'architettura prevede che tutti i kernel siano di dimensione 3x3.

L'ultimo layer è un softmax, che nella nostra applicazione è stato sovrascritto affinché la classificazione avvenisse sulle 3 possibili classi, invece che su 1000 come era di default.

3.4. Metodi di training

Qui mostriamo altri due elementi fondamentali nell'allenamento del modello: il *criterion* e l'*optimizer*.

Il criterion da noi scelto è la cross entropy loss, standard *de facto* per la multi-class classification, riportata in Valutazione. L'algoritmo di ottimizzazione che abbiamo usato è lo Stochastic Gradient Descent (SGD), che effettua la discesa del gradiente su un sottoinsieme randomico del dataset. Osserviamo che la componente stocastica dell'algoritmo non è compresa nel modulo di PyTorch torch.optim.SGD, a dispetto del nome, ma è compito nostro implementare un sistema di mini batch tramite loaders, affinché la discesa del gradiente (GD) risulti effettivamente una SGD.

Infine, restava da valutare l'utilizzo di eventuali approcci di transfer learning per sfruttare modelli preallenati. Il modulo resnet18 di TorchVision offre la possibilità di caricare i pesi derivanti dall'allenamento su Dataset pubblici di test per il riconoscimento di immagini, come ImageNet. Abbiamo effettuato una prova di allenamento della rete lasciando fissi i parametri di tutti i livelli eccetto l'ultimo, ovvero usando la rete come feature extractor. I risultati ottenuti però non sono stati quelli sperati, e abbiamo dedotto che il nostro Dataset fosse troppo diverso da quello usato per il preallenamento. Proseguendo nella valutazione con degli allenamenti sull'intera

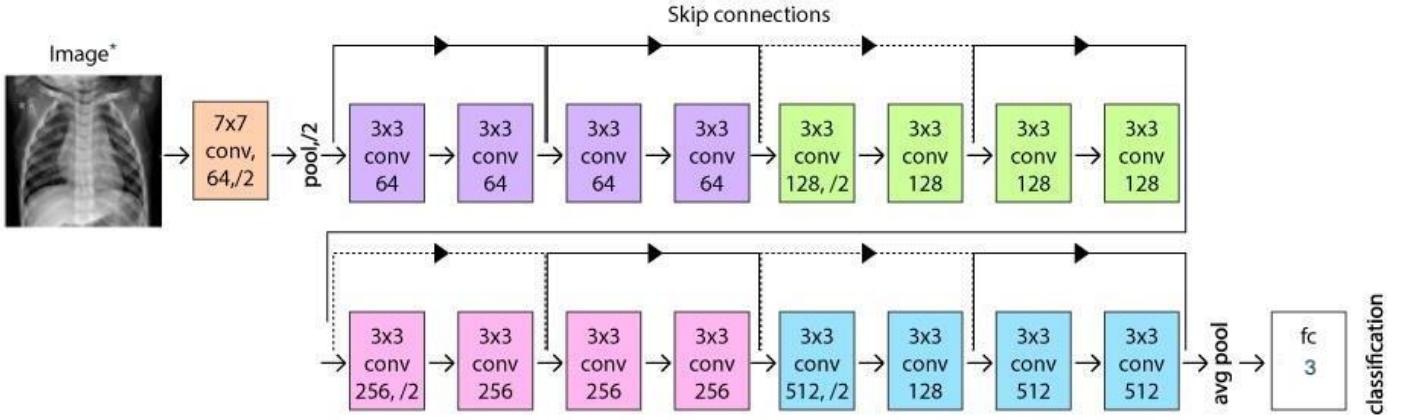


Figure 6: Architettura della ResNet18, credits to Mustafa Khan, Muhammad et al. ‘Dual Deterministic Model Based on Deep Neural Network for the Classification of Pneumonia’. 1 Jan. 2023 : 1 – 14.

rete, abbiamo riscontrato una buona risposta del modello e dunque abbiamo insistito su questa strada, senza utilizzare il Finetuning.

4. Valutazione

Una volta scelto il modello da allenare, abbiamo dovuto proseguire con la selezione dei parametri da visionare al fine di una valutazione delle prestazioni di quest’ultimo. Alla fine la scelta è ricaduta sulle seguenti performance measures:

1. Train and Validation Loss: in generale la funzione di loss è una componente chiave in un algoritmo di apprendimento, difatti il diminuire del valore di loss vuol dire che il modello sta imparando. La verifica sul validation set è necessaria sia per assicurarsi di non essere in presenza di overfitting, sia per comunque, più in generale, scegliere la corretta configurazione degli iperparametri, specialmente il learning rate. In linea con quanto appreso in teoria, la funzione di loss adeguata ad un problema di classificazione è la cross entropy loss, che ricordiamo essere definita come segue:

$$J(\theta) = -\frac{1}{M} \sum_{i=1}^M \sum_{c=1}^3 1\{y^{(i)} = c\} \log(h_\theta(x^{(i)})) \quad (1)$$

Infatti come ci aspettiamo questa funzione restituirà valori alti in caso di predizioni errate, e bassi nel caso di classificazione corretta.

2. Train and Validation Accuracy: chiaramente il nostro classificatore funziona bene se predice correttamente le classi di ogni immagine. Anche in questo caso abbiamo deciso di monitorare il parametro sia sul training set che sul validation set, per motivi analoghi a quelli sopra. Formalmente l’accuracy si può esprimere come:

$$Accuracy = \frac{\#\hat{y}_{correct}}{\#y} \quad (2)$$

3. Matrici di confusione: nel contesto di classificazione multi classe, la matrice di confusione è molto utile per evidenziare alcune possibili criticità del modello, ad esempio la difficoltà nel distinguere 2 classi in particolare. Per ogni epoca abbiamo attenzionato dunque i valori di True Positive, False Positive, True Negative e False Negative per ciascuna delle 3 classi.
4. Recall and F1 Score: avendo a disposizione la matrice di confusione abbiamo deciso di valutare questo parametro, che ricordiamo essere così definito

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

Essendo il nostro un caso medico, riteniamo quello della recall un parametro rilevante, visto che una recall che tende ad 1, vuol dire avere un basso numero di falsi negativi, il che è molto importante. In ogni caso abbiamo deciso di monitorare anche il parametro F1 Score, che tiene conto sia della recall che della precision, nella seguente maniera

$$F1 = 2 \frac{Recall * Precision}{Recall + Precision} \quad (4)$$

Dove ricordiamo che la precision è definita come

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

5. Esperimenti

- Come discusso nella sezione **Dataset**, i primi allenamenti del nostro modello sono stati su un dataset iniziale, che per comodità chiameremo D1. Inoltre ci riferiremo ai vari allenamenti con TX dove X è l'indice cronologico dell' allenamento da noi effettuato. Il primo training in assoluto non è stato molto positivo, come questi valori possono dimostrare:

Dataset	Epochs	Learning Rate	Best Val Accuracy
D1	50	0.01	0.7952

Table 3: Parametri e risultati del training T1

Ma se a livello di best accuracy le prestazioni non erano molto elevate, ancor più preoccupante per noi era l'andamento dei grafici di loss e accuracy

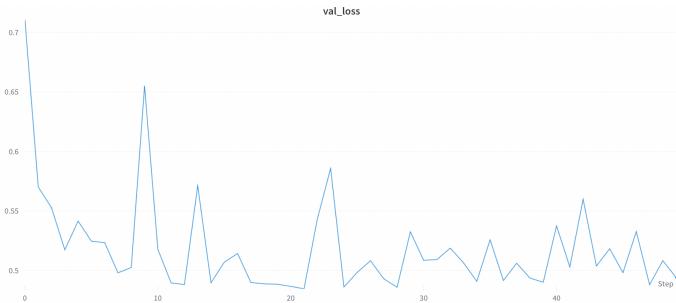


Figure 7: T1 Validation Loss

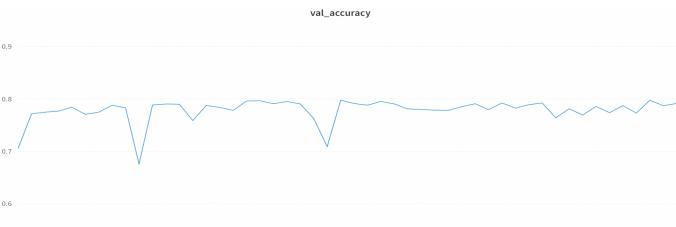


Figure 8: T1 Validation Accuracy

Infatti l'andamento a 'zig zag' della funzione di loss è un qualcosa che fondamentalmente vogliamo sempre evitare.

- Una possibile causa di queste oscillazioni può in genere essere un learning rate troppo alto, pertanto il primo pensiero è stato quello di diminuire questo iperparametro. Inoltre abbiamo aumentato il numero delle epoche da 50 a 100. Dunque il secondo loop di allenamento ha avuto quest'esito:

Dataset	Epochs	Learning Rate	Best Val Accuracy
D1	100	0.005	0.7962

Table 4: Parametri e risultati del training T2

Nonostante le modifiche apportate, gli effetti sortiti non sono stati quelli sperati come si vede dai seguenti grafici:

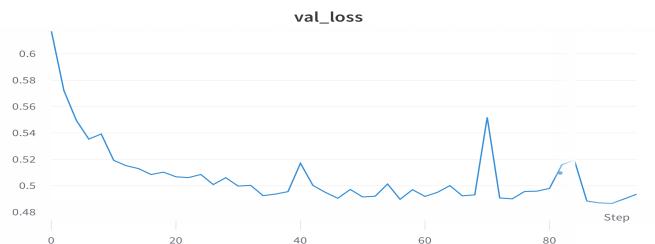


Figure 9: T2 Validation Loss



Figure 10: T2 Validation Accuracy

Infatti l'andamento della loss continua ad essere oscillante, inoltre il fatto che il miglior valore di accuracy nel validation set sia stato prima della ventesima epoca, ci ha dato ulteriore conferma del fatto che il modello non stesse imparando. Visto l'andamento del primo training, da questo in poi abbiamo deciso di plottare pure le matrici di confusione, in modo tale da poter prestare attenzione alle possibili difficoltà di classificazione. Il risultato, visibile in 11 fa notare come il problema sia la corretta classificazione degli elementi di classe 2, ovvero polmonite virale, che viene per il 40% delle volte classificata come polmonite batterica.

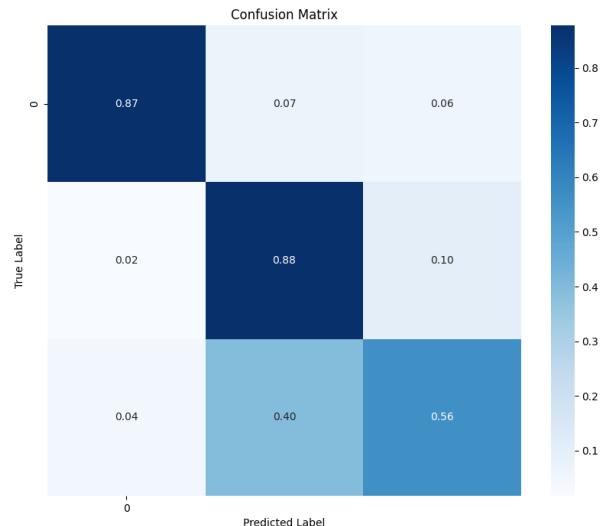


Figure 11: T2 Confusion Matrix

3. Preso atto di ciò abbiamo provato a implementare un meccanismo a cascata, partendo da una prima classificazione binaria tra radiografia con presenza di polmonite(unendo dunque i casi di batterica e virale) e radiografia normale, con l'obiettivo di, successivamente, passare le immagini predette come polmoniti ad un secondo classificatore, che doveva semplicemente concentrarsi sulla distinzione tra classe batterica e classe virale. Il primo allenamento del primo classificatore è stato il seguente:

Dataset	Epochs	Learning Rate	Best Val Accuracy
D1	100	0.01	0.9058

Table 5: Parametri e risultati del training T3

Seppur a primo impatto il risultato fosse discreto la matrice di confusione ha evidenziato delle criticità

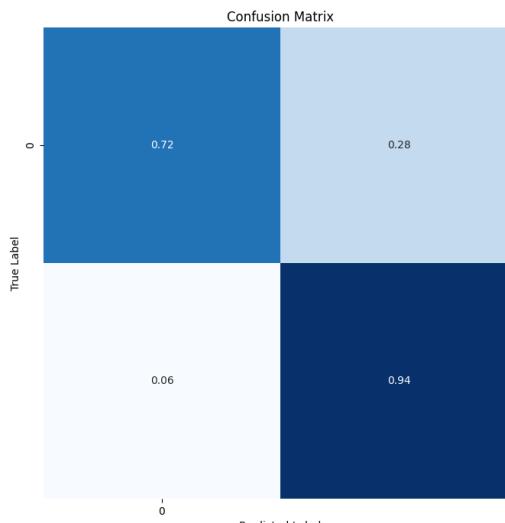


Figure 12: T3 Confusion Matrix

4. Infatti la difficoltà nel riconoscere i casi normali era evidente, ed effettivamente anche immaginabile visto lo sbilanciamento delle classi abbastanza netto, con 1479 esempi normali, e un totale di 4382 casi di polmonite. Per prima cosa abbiamo provato dunque ad aumentare il weight decay del nostro modello, precedentemente impostato a 0.001, a 0.005, per una maggiore regolarizzazione. Di questa run ci basta analizzare la matrice di confusione 13.

Come possiamo notare la situazione è migliorata, ma non di quanto volessimo, visto che aspettandoci un risultato peggiore sul secondo classificatore, volevamo risultati quasi impeccabili sul primo.

5. A questo punto abbiamo cercato qualche altro esempio di immagini di radiografie 'normali' in rete, imbastendoci nel dataset D2. Il nostro primo step è stato però aggiungere le 1802 immagini normali di D2 a D1, per ribilanciare le

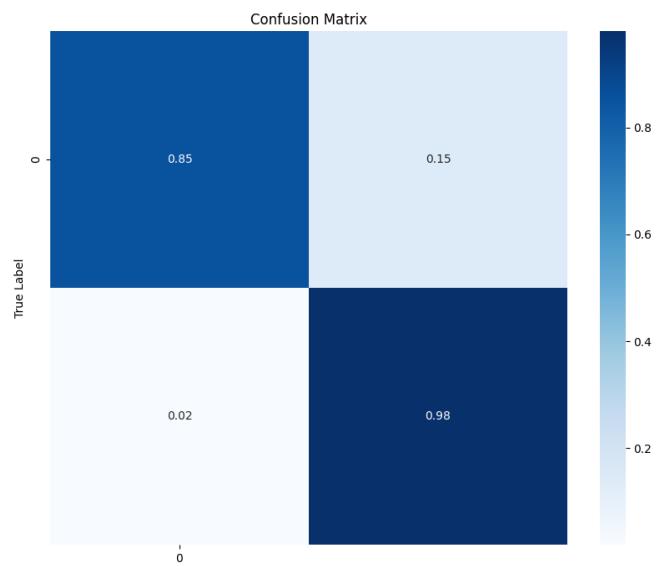


Figure 13: T4 Confusion Matrix

classi. Dunque, con questi presupposti il risultato è stato il seguente:

Dataset	Epochs	Learning Rate	Best Val Accuracy
D1 modified	100	0.005	0.8987

Table 6: Parametri e risultati del training T5



Figure 14: T5 Validation Loss

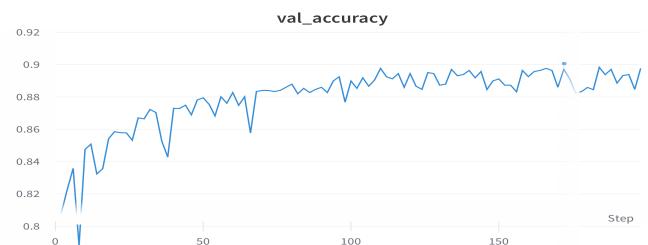


Figure 15: T5 Validation Accuracy

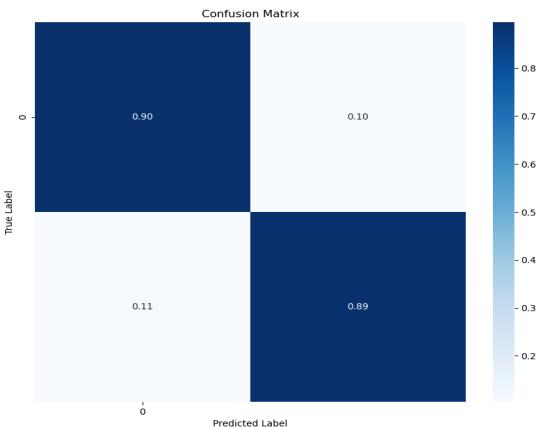


Figure 16: T5 Confusion Matrix

I risultati erano quindi per noi soddisfacenti, dunque si è potuto proseguire con l’allenamento del modello atto a distinguere tra polmonite virale e batterica.

6. Purtroppo, però come visibile dai risultati e dai grafici, la distinzione tra le due tipologie di polmonite risultava difficile per il modello, probabilmente sempre a causa di un eccessivo sbilanciamento tra le classi.

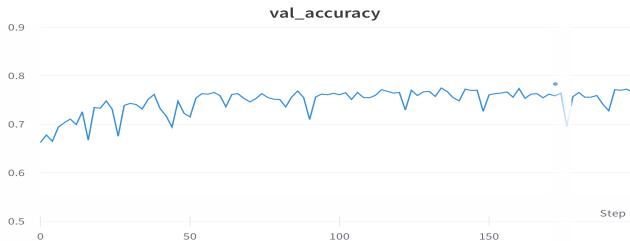


Figure 17: T6 Validation Accuracy

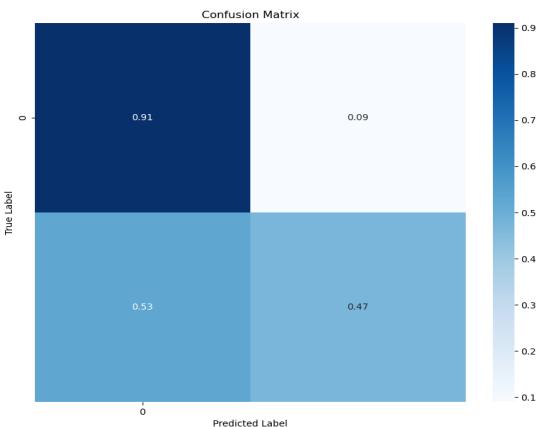


Figure 18: T6 Confusion Matrix

7. Alla luce di queste considerazioni, abbiamo deciso di ritornare sui nostri passi, affrontando il problema con una classificazione a 3 classi, andando a però ad utilizzare il dataset D2, che presenta, come evidenziato in precedenza una distribuzione più uniforme rispetto a D1. Questa volta i risultati sono stati soddisfacenti, tanto è vero che gli andamenti grafici erano di continua discesa della loss e graduale salita dell’accuracy.

Dataset	Epochs	Learning Rate	Best Val Accuracy
D2	200	0.005	0.8765

Table 7: Parametri e risultati del training T7

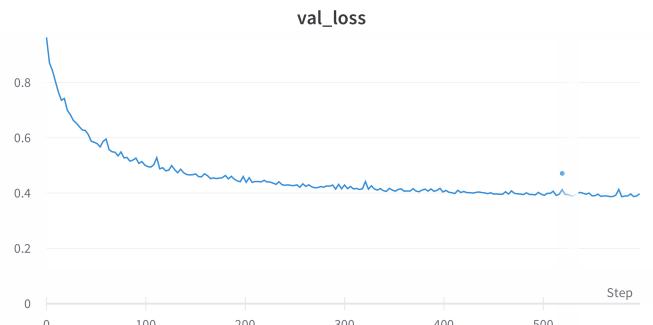


Figure 19: T7 Validation Loss

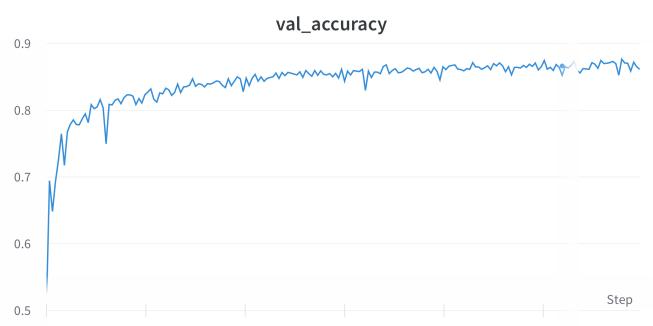


Figure 20: T7 Validation Accuracy

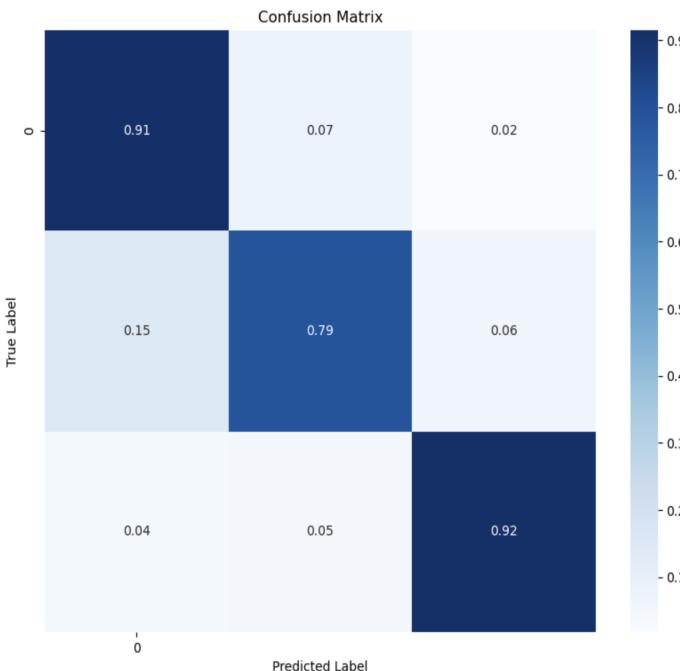


Figure 21: T7 Confusion Matrix

Stabiliti quindi i parametri del modello, lo abbiamo utilizzato per fare inferenza sul test set in modo tale da valutare le performance su un insieme di elementi mai visti prima. A seguire i risultati:

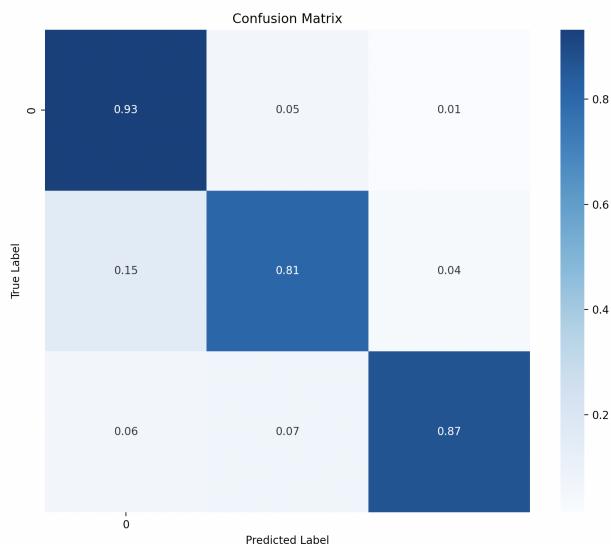


Figure 22: T7 Test Confusion Matrix

Loss	Accuracy	Recall	F1
0.3964	0.8704	0.8702	0.8707

Table 8: Risultati del Test finale

6. Demo

Dopo aver importato le librerie necessarie al funzionamento del codice, viene caricato il dataset creando un’istanza della classe `ChestXrayDataset`, specificando la directory delle immagini del dataset ed imponiamo che non venga applicata nessuna trasformazione alle immagini. Dopo la definizione di alcune variabili ed il caricamento del modello già allenato, verrà visualizzata la seguente interfaccia:

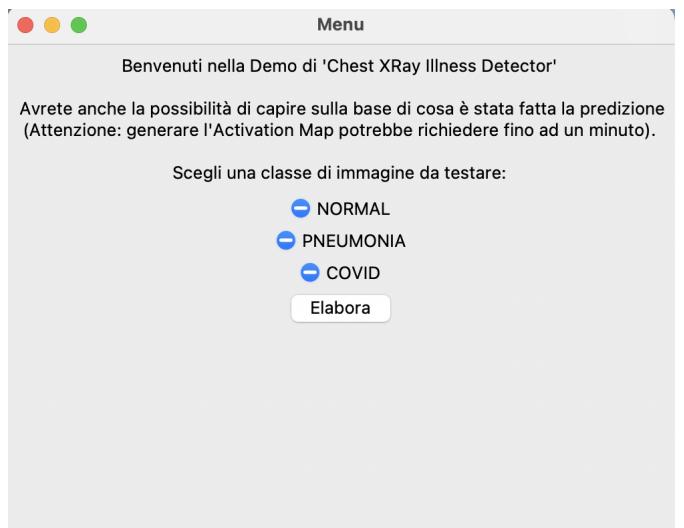


Figure 23: Demo welcome page

Per proseguire basterà cliccare una delle tre opzioni e successivamente il tasto ”Elabora”. Un esempio di risultato è il seguente:

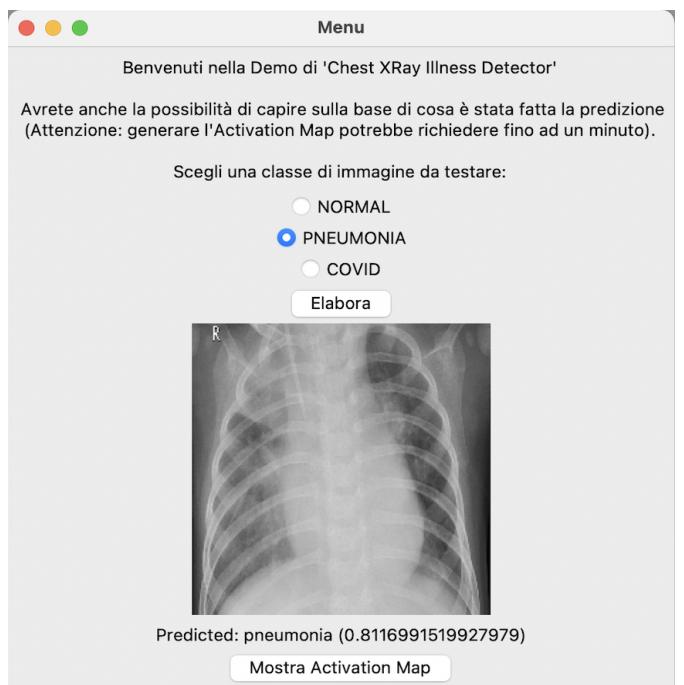


Figure 24: Prediction and score

Viene selezionata un'immagine casuale dal set di test e viene fatta un'inferenza su di essa utilizzando il modello allenato. Il nome della classe predetta seguito dalla probabilità di predizione verranno visualizzati nell'interfaccia grafica. Qui sarà possibile cliccare su "Mostra Activation Map" per visualizzare la mappa di attivazione. Come scritto, la generazione potrebbe richiedere fino ad un minuto per la sua computazione. L'output sarà come il seguente:

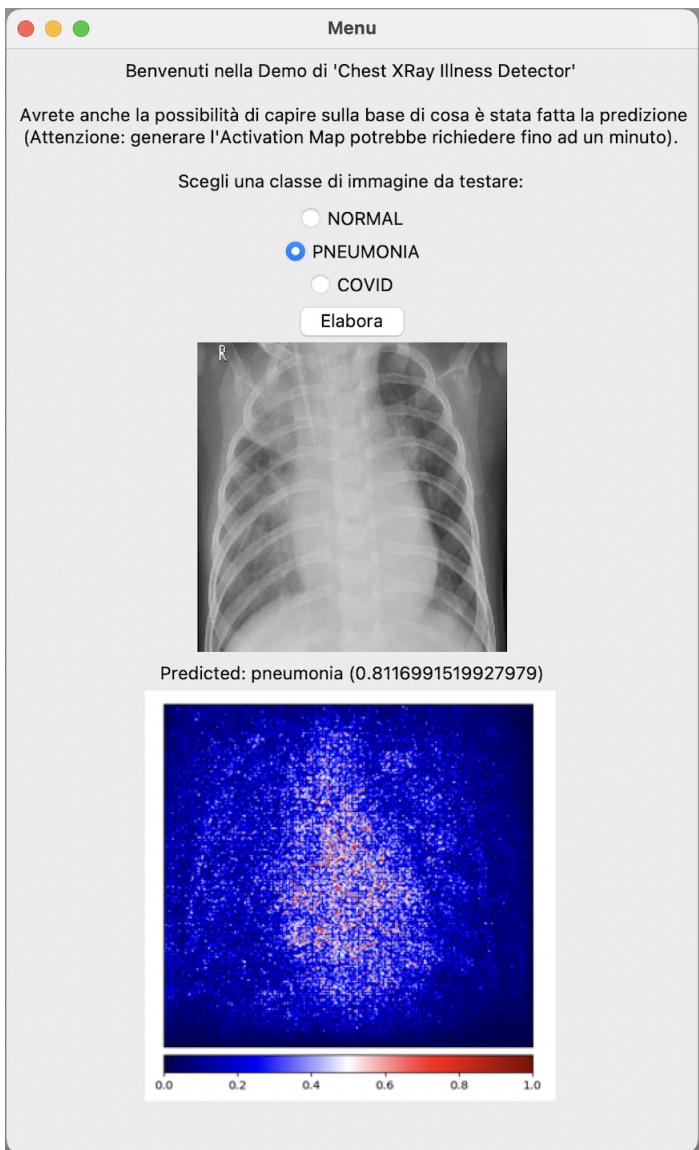


Figure 25: Activation Map

La mappa di attivazione riveste un'importanza significativa nell'analisi delle reti neurali convoluzionali e nella comprensione dei processi decisionali di tali modelli. Fornisce una rappresentazione visuale delle regioni dell'immagine che hanno avuto un impatto maggiore sulla predizione effettuata dal modello. L'analisi dell'activation map consente di ottenere una forma di interpretabilità per i modelli di machine learning, rendendo il processo decisionale delle reti neurali più trasparente e comprensibile. Inoltre, attraverso la visualizzazione delle re-

gioni attivate, gli esperti del settore interessato possono analizzare e valutare la coerenza delle predizioni del modello rispetto alle loro conoscenze. Ciò può contribuire a identificare potenziali errori o punti deboli del modello, migliorando l'affidabilità delle sue predizioni.

7. Codice

Il codice è organizzato in 3 file principali: `MyModel.py`, `ChestXRayDataset.py` e `main.py`.

7.1. `MyModel.py`

Contiene la definizione della classe `MyModel`, eredita dalla classe `nn.Module` di PyTorch e viene utilizzata per creare la nostra rete neurale. Nel metodo `__init__` inizializziamo il modello utilizzando `resnet18` come base e sostituiamo l'ultimo strato, il classificatore, per adattarlo al numero di classi specificato. Se il parametro `Trained` è impostato su `True`, il modello viene caricato da un file di checkpoint preallenato chiamato `"my_model_best_val.pth"` (servirà nella fase di test). Sono presenti i metodi `train` e `test`: `train` consente di allenare il modello utilizzando i dati di addestramento, mentre `test` viene utilizzato per valutare le prestazioni del modello usando i dati di test, ottenendo alla fine la matrice di confusione, `Recall` e `F1 score`.

7.2. `ChestXRayDataset.py`

In `ChestXRayDataset.py` definiamo la classe `ChestXRayDataset`, che rappresenta un dataset personalizzato per le immagini di raggi X del torace. Il costruttore specifica la directory che ospita le immagini e se le queste devono subire o meno delle trasformazioni. Quando si applicano le trasformazioni, viene utilizzata una combinazione di riflessioni orizzontali e verticali e un ridimensionamento a `224x224` pixel. Inoltre, le immagini vengono poi convertite in tensori e normalizzate. Sempre qui dividiamo il dataset in tre sottoinsiemi tramite la funzione `train_test_split` di `sklearn` ottenendo Train set (60%) Validation set (20%) e Test set (20%). Oltre ai metodi privati `__len__`, `__getlabel__` e `__getitem__` per ottenere la lunghezza del dataset, l'etichetta di un'immagine specifica (in cui sfruttiamo la formattazione `"CLASS_x.png"`) e per ottenere un campione di dati specifico, è presente `get_loaders` per farsi restituire i `dataloader` per i dati di `train`, `validation` e `test`.

7.3. `main.py`

Dopo aver importato le librerie necessarie al funzionamento del codice, viene caricato il dataset creando un'istanza della classe `ChestXRayDataset`, specificando la directory delle immagini del dataset. Di default `transform=True` quindi stiamo applicando le trasformazioni alle immagini. Dopo aver mostrato delle informazioni sul numero di immagini di training, validation e test, si entra nel vivo del programma: basterà scegliere "1" per allenare il modello o "2" per testarlo. Se viene scelto "1", verrà caricato il modello non allenato, definiti un optimizer (con un certo learning rate e weight_decay) e una funzione di loss e poi fatto partire il metodo di `train`. Alla pressione del "2", invece, viene caricato il modello già allenato, dunque

con i pesi corrispondenti alla miglior performance di validation ottenuta nei nostri **esperimenti**, e viene avviata la procedura di test: verranno mostrati i valori di Loss, Accuracy, Recall, F1 score e la Matrice di Confusione.

8. Conclusione

Il lavoro che abbiamo svolto ci ha consegnato un modello capace di classificare una radiografia come normale, covid-19 o polmonite batterica, il tutto con un'accuratezza dell'87% circa. Per arrivare a ciò, come si può vedere dai paragrafi precedenti, è stato necessario affrontare diverse problematiche che ci hanno portato alle seguenti riflessioni:

1. In primis abbiamo potuto toccare con mano come il dataset, e quindi i dati, siano il cuore del processo di apprendimento di un modello di machine learning, visto ad esempio l'impatto che un dataset sbilanciato ha avuto sui nostri esperimenti iniziali. Da ciò traiamo quanto possa essere di fondamentale importanza investire abbastanza tempo in un'accurata ricerca del dataset stesso, o comunque ad suo adattamento in funzione del task da svolgere.
2. In secondo luogo, la scelta del modello riveste un ruolo di primo rilievo, visto che esperimenti intermedi da noi svolti evidenziavano come modelli troppo complessi portassero ad una situazione di overfitting, sicuramente accentuata dallo sbilanciamento iniziale del dataset. Faremo tesoro dell'importanza del trade-off tra bias e variance del modello.
3. In fine, come già detto, abbiamo appurato come l'analisi delle mappe di attivazione possa contribuire a rendere la nostra classificazione sempre più "explainable", dando così più valore ed utilità alla predizione effettuata.

Alla luce di quanto detto, immaginiamo che un miglioramento delle prestazioni del nostro modello si possa raggiungere attraverso un dataset più ampio, ma ugualmente bilanciato, ottenuto sia da altre repository che attraverso tecniche di data augmentation e dunque l'inserimento tra i dati di immagini create a partire da quelle presenti nel dataset.

References

- He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778. doi:[10.1109/CVPR.2016.90](https://doi.org/10.1109/CVPR.2016.90).
- Javid, M., Haleem, A., Pratap Singh, R., Suman, R., Rab, S., 2022. Significance of machine learning in healthcare: Features, pillars and applications. International Journal of Intelligent Networks 3, 58–73. URL: <https://www.sciencedirect.com/science/article/pii/S2666603022000069>, doi:<https://doi.org/10.1016/j.ijin.2022.05.002>.