SAPIENZA
UNIVERSITÀ DI ROMA

Riconoscimento del Melanoma Cutaneo attraverso tecniche di Deep Learning

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Ingegneria Informatica e Automatica

Candidato

Giulio Tamburini
Matricola 1915793

Relatore

Prof. Thomas Alessandro Ciarfuglia

Anno Accademico 2021/2022

Riconoscimento del Melanoma Cutaneo attraverso tecniche di Deep Learning
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: tamburini.1915793@studenti.uniroma1.it

Sommario

Il documento seguente è il frutto di mesi di lavoro, nei quali ho applicato le conoscenze apprese nel mio intero percorso di studi ed in particolare dal *Laboratorio di Intelligenza Artificiale e Grafica Interattiva*, corso che ho frequentato nell'ultimo semestre del terzo Anno Accademico.

In questo elaborato verrà affrontato un problema di intelligenza artificiale applicata al campo della diagnostica dermatologica. Più precisamente, l'obiettivo è quello di riuscire a creare una rete neurale convoluzionale che sia in grado di classificare correttamente delle immagini di nei in benigni o maligni.
Per fare ciò, ho scaricato un dataset di 1279 immagini dermoscopiche etichettate con le rispettive categorie (benigno o maligno). Usando queste immagini ho quindi addestrato e successivamente testato il mio modello, ottenendo i risultati che analizzerò dettagliatamente nell'ultimo capitolo.

Questo progetto, oltre ad avermi permesso di migliorare e di mettere in pratica le mie conoscenze nel campo dell'intelligenza artificiale, ha suscitato in me un grande interesse dovuto al suo riscontro pratico e alle sue grandi potenzialità in ambito medico. Infatti l'applicazione di algoritmi di apprendimento automatico per il riconoscimento del melanoma è già una realtà, e in futuro diventerà uno strumento indispensabile per velocizzare e migliorare l'accuratezza dell'individuazione di questa ed altre malattie.

Indice

Capitolo 1

Introduzione

L'Intelligenza Artificiale (IA) è indubbiamente uno dei campi di ricerca che si è sviluppato maggiormente negli ultimi anni. Il suo obiettivo è quello di implementare sistemi informatici in grado di svolgere compiti che vengono generalmente associati all'intelligenza umana, come ad esempio la capacità di ragionare, di apprendere dalle esperienze passate o di percepire visivamente gli oggetti che ci circondano. Pur essendo presenti molte sfide e dibattiti riguardo all'etica sull'uso dell'intelligenza artificiale, è bene tenere presente che il suo obiettivo è quello di migliorare la qualità di vita degli esseri umani e aumentare l'efficienza in numerosi ambiti lavorativi.

Uno dei rami dell'IA che ha avuto maggior impatto sociale ed economico è quello del Machine Learning (ML). Traducendo una frase di uno dei più grandi pionieri dell'intelligenza artificiale:

> "[Il Machine Learning è il] campo di studi che dà ai computer la capacità di imparare senza essere esplicitamente programmati."
> -Arthur Samuel, 1959

In altre parole, il machine learning è quel sottoinsieme dell'intelligenza artificiale il quale si propone di sviluppare algoritmi che permettono ai computer di apprendere e migliorare le loro prestazioni nello svolgimento di compiti specifici, basandosi semplicemente sulle esperienze passate e sui dati, senza essere esplicitamente programmati per quei compiti.

Uno dei motivi che ha portato il machine learning a ricevere un enorme interesse negli ultimi anni è stato il Deep Learning, una sottocategoria del ML che si basa su reti neurali composte da numerosi strati, chiamate reti neurali profonde. Il deep learning porta con sé numerosi vantaggi, in particolare il fatto che riesce a gestire enormi volumi di dati sfruttando la potenza di calcolo delle moderne GPU. In questo modo riesce a svolgere compiti altamente complessi come il riconoscimento vocale, la traduzione automatica del linguaggio e il rilevamento di caratteristiche e oggetti all'interno delle immagini, usato ad esempio per la guida autonoma dei veicoli o per la diagnosi medica.

L'algoritmo che verrà introdotto in questa tesi è un chiaro esempio di come le reti neurali profonde, ed in particolare le reti neurali convoluzionali (CNN), riescano ad avere applicazioni pratiche nell'ambito medico.

Nel Capitolo 2 sarà presente un'introduzione sul melanoma cutaneo: la malattia della pelle sulla quale si concentrerà l'algoritmo proposto in questa tesi.

Nel Capitolo 3 verrà spiegato cosa è l'organizzazione ISIC e in cosa consiste la competizione da loro organizzata.

Nel Capitolo 4 verrà illustrato il concetto di deep learning e la struttura delle reti neurali convoluzionali, con un focus sull'architettura ResNet-50. Questo capitolo risulta fondamentale per acquisire una comprensione completa dell'algoritmo.

Nel Capitolo 5 verrà descritto nel dettaglio il funzionamento dell'algoritmo, a partire dalla preparazione dei dati fino ad arrivare ai risultati ottenuti.

Infine il Capitolo 6 contiene la conclusione del progetto e le mie considerazioni finali.

Capitolo 2

Melanoma cutaneo

2.1 Definizione

Il melanoma cutaneo è un tipo di cancro che si sviluppa a causa dei melanociti, alcune delle cellule che compongono la pelle. Queste cellule producono la melanina, un pigmento che protegge la cute dai danni causati dai raggi UV del sole.
In condizioni normali i melanociti possono creare ammassi scuri sulla pelle, visibili ad occhio nudo e comunemente chiamati nei (il termine medico è "nevi").
In seguito ad un'eccessiva esposizione ai raggi solari o ad altri fattori di rischio, come l'insufficienza del sistema immunitario, la pelle e gli occhi chiari, i capelli rossi o biondi, una storia familiare di melanoma o la presenza di molti nei o lentiggini, i melanociti possono crescere fuori controllo diventando maligni, cioè in grado di diffondersi in altre parti del corpo.

Esistono quattro tipi diversi di melanoma cutaneo: il melanoma a diffusione superficiale, che è il più comune (circa il 70% dei casi), il melanoma lentigginoso acrale, il lentigo maligna melanoma e il melanoma nodulare, che è il più aggressivo e rappresenta circa il 10-15% di tutti i melanomi cutanei.

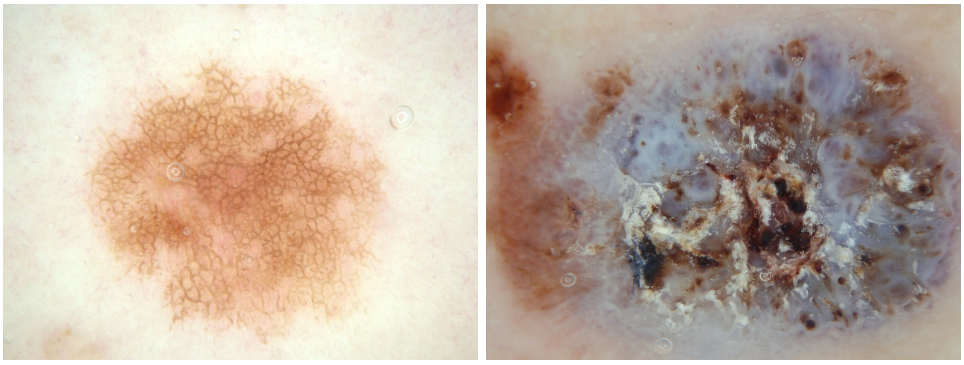
2.2 Sintomi

Questo tipo di cancro può avere origine da una pelle sana oppure da nevi già esistenti, che possono essere presenti sin dalla nascita o più comunemente possono comparire col passare del tempo. Alcuni segnali di allarme che possono indicare l'insorgenza di un melanoma sono riassunti nella regola dell'ABCDE:

- A come Asimmetria: una metà di un neo è diversa dall'altra;
- B come Bordo: i bordi del neo sono irregolari e indistinti;
- C come Colore: il colore ha sfumature diverse all'interno dello stesso neo e può includere tonalità di marrone o nero, a volte con chiazze rosa, rosse, bianche o blu;
- D come Dimensioni: il diametro del neo è più grande di 6 millimetri (anche se alcuni possono essere più piccoli);
- E come Evoluzione: il neo in tempi brevi cambia di dimensione, forma o colore.

Altri campanelli d'allarme a cui bisogna prestare attenzione possono essere il prurito, il sanguinamento o la presenza di un nodulo o di un'area arrossata attorno al neo.

In tutti questi casi è opportuno consultare un medico specializzato al più presto ed è comunque consigliato farsi visitare periodicamente da un dermatologo.



(a) Esempio di un neo benigno. **(b)** Esempio di melanoma.

Figura 2.1. Differenza tra due nei, che evidenzia i sintomi descritti in precedenza.

2.3 Evoluzione e cura

Il melanoma viene solitamente classificato in cinque stadi definiti sulla base del sistema TNM sabilito dall'American Joint Committee on Cancer (AJCC). Questo sistema si basa su: lo spessore del tumore e la presenza di ulcerazioni[1] (T); la diffusione del melanoma ai linfonodi vicini (N); la presenza di eventuali metastasi[2] (M). I diversi stadi della malattia sono:

- Stadio 0: il melanoma è confinato nella parte superficiale della pelle (l'epidermide), senza aver invaso lo strato più profondo (il derma) e senza essersi diffuso in altre parti del corpo. Questo stadio viene anche chiamato *melanoma in situ.*
- Stadio I: il melanoma ha invaso il derma, ha uno spessore minore di 2 millimetri, ma è ancora confinato nella zona in cui si è originato, senza essersi diffuso ai linfonodi vicini o in altre parti del corpo.
- Stadio II: il melanoma potrebbe essere ulcerato; è più spesso di 1 mm e può raggiungere anche più di 4 mm. Non si è diffuso ai linfonodi vicini o in altre parti del corpo.
- Stadio III: il melanoma potrebbe essere ulcerato; si è diffuso ai linfonodi vicini o ai tessuti circostanti, ma non ancora in altre parti del corpo.
- Stadio IV: il melanoma si è diffuso ad altre parti del corpo, come i polmoni, il fegato o il cervello.

Ovviamente, la prognosi e le probabilità di sopravvivenza possono essere molto diverse in base alla gravità della malattia: la percentuale di sopravvivenza dopo 5 anni per pazienti a cui hanno diagnosticato un melanoma di tipo I è circa del 95% mentre si abbassa tra l'8 e il 25% quando la malattia viene diagnosticata allo

[1] Rottura della pelle sopra il melanoma. I melanomi che sono ulcerati hanno una probabilità maggiore di diffusione.

[2] Diffusione del tumore in linfonodi o organi distanti.

stadio IV. Il trattamento per il melanoma cutaneo dipende quindi dallo spessore della lesione e dallo stadio della malattia e può includere: la rimozione chirurgica del tumore, che in genere è la prima scelta (soprattutto se il melanoma è in fase iniziale), la chemioterapia, l'immunoterapia, la terapia a bersaglio molecolare ed infine la radioterapia.

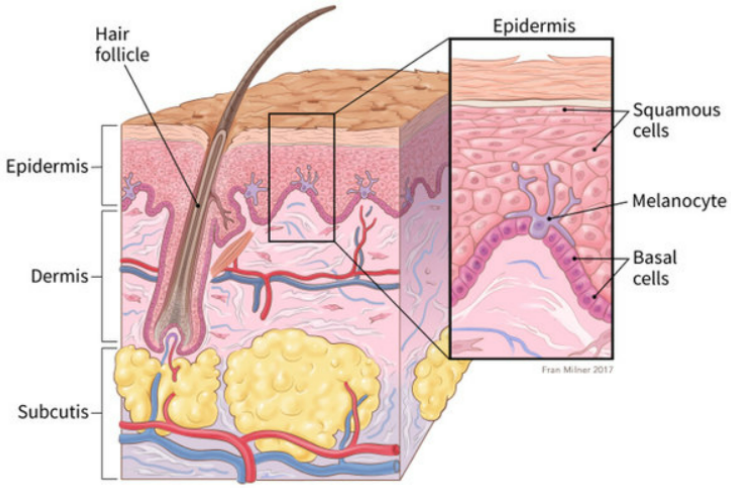


Figura 2.2. Illustrazione dello strato cutaneo e sottocutaneo.

2.4 Individuazione precoce e IA

Bisogna tenere presente che il melanoma cutaneo costituisce soltanto una frazione ridotta (circa il 5 per cento) di tutti i tumori della pelle. Nonostante ciò, è il tipo più pericoloso in quanto, come abbiamo già visto, tende a diffondersi rapidamente in altre parti del corpo e in molti casi può portare anche alla morte.
L'individuazione tempestiva del melanoma diventa dunque fondamentale per la sua cura e può essere effettuata attraverso un esame della pelle da parte di un medico specializzato.

Nella stessa direzione si sta anche muovendo il campo dell'intelligenza artificiale, che sta facendo grandi passi avanti nel campo della diagnostica del melanoma attraverso l'uso di algoritmi di apprendimento automatico che analizzano le immagini di lesioni della pelle, cercando di individuare eventuali tumori.

I primi algoritmi di apprendimento automatico basati sulle reti neurali sono stati sviluppati negli anni '90, ma non venivano usati a causa della scarsa tecnologia e della conseguente impossibilità di produrre risultati accettabili. Tuttavia negli ultimi anni, grazie ai miglioramenti portati dall'avvento delle reti neurali convoluzionali e del deep learning (che approfondiremo maggiormente nel Capitolo 4), alcuni studi hanno dimostrato che l'accuratezza di questi algoritmi nell'identificazione del melanoma può essere paragonabile a quella di dermatologi esperti. Da ciò si può dedurre che l'intelligenza artificiale sta assumendo un'ingente rilevanza, destinata in futuro ad aumentare sempre di più, come risorsa per la diagnosi precoce del melanoma cutaneo.

Per comprendere il potenziale della tecnologia in questo campo basti pensare ad un suo possibile uso in zone sottosviluppate e carenti di dermatologi esperti, o addirittura ad un uso privato per monitorare costantemente il proprio corpo senza il

bisogno continuo di fare controlli di routine. A conferma di ciò, esistono già varie aziende e piattaforme digitali accessibili a tutti in grado di svolgere questi compiti, come ad esempio MetaOptima o SkinVision, un'applicazione disponibile su App Store e Google Play che ha l'obiettivo di riconoscere la presenza di tumori della pelle semplicemente tramite la fotocamera del proprio smartphone o tablet.

Ha assunto inoltre una considerevole importanza nello sviluppo dell'automatizzazione della diagnostica dermatologica anche la ISIC Challenge, una competizione di intelligenza artificiale nel quale io stesso mi sono cimentato e il cui approfondimento avremo modo di trattare nel prossimo capitolo.

Capitolo 3

ISIC Challenge

3.1 Cosa è la ISIC Challenge

La ISIC (International Skin Imaging Collaboration) Challenge è una delle più importanti competizioni di intelligenza artificiale per la diagnostica dermatologica, il cui obiettivo principale è quello di ottimizzare la diagnosi precoce del melanoma. La sfida, sponsorizzata dall'International Society for Digital Imaging of the Skin (ISDIS), è stata avviata nel 2016 ed è stata organizzata annualmente fino al 2020 dal gruppo di ricerca ISIC. La competizione è aperta a qualsiasi partecipante interessato, dai ricercatori accademici ai professionisti dell'informatica e della sanità, e offre premi in denaro ai vincitori.

L'archivio ISIC contiene la più ampia collezione di immagini di lesioni della pelle disponibile pubblicamente. Parliamo infatti di oltre 13000 immagini dermoscopiche[1] che sono state raccolte dalle principali cliniche a livello internazionale.
I partecipanti alla sfida sono invitati a sviluppare algoritmi di intelligenza artificiale in grado di riconoscere eventuali segni di melanoma presenti in queste immagini, utilizzando tecniche di apprendimento automatico.

3.2 Challenge Tasks

La ISIC Challenge in ogni edizione propone problemi diversi, i quali possono essere suddivisi in compiti separati, anche detti "task". Ogni partecipante deve risolvere almeno uno di questi task e viene valutato singolarmente per ogni consegna.

L'obiettivo generale della competizione è sempre quello di sviluppare strumenti di analisi delle immagini per consentire la diagnosi automatizzata del melanoma. Per fare ciò, ogni partecipante deve creare un modello che viene "allenato" usando un dataset di immagini dermoscopiche fornito dalla ISIC. Questo modello viene poi testato su un dataset più piccolo (sempre fornito dall'organizzazione), producendo i risultati su cui vengono valutati i singoli progetti.

Di seguito vengono riportati più in dettaglio i diversi task delle edizioni passate della competizione:

- ISIC Challenge 2016: la sfida era divisa in tre task:
    1. Segmentazione della lesione (vedi Figura 3.1);

[1] Immagini ad alta risoluzione che permettono ai dermatologi di esaminare i dettagli della pelle che sono invisibili ad occhio nudo.

2. Rilevamento ed estrazione di caratteristiche / modelli dermoscopici delle lesioni;
3. Classificazione binaria tra lesioni maligne (melanoma) e lesioni benigne.

Per questa sfida veniva fornito un training set[2] di 900 immagini con le relative label[3], ed un test set[4] di circa 350 immagini, grazie al quale vengono valutati oggettivamente tutti i progetti.

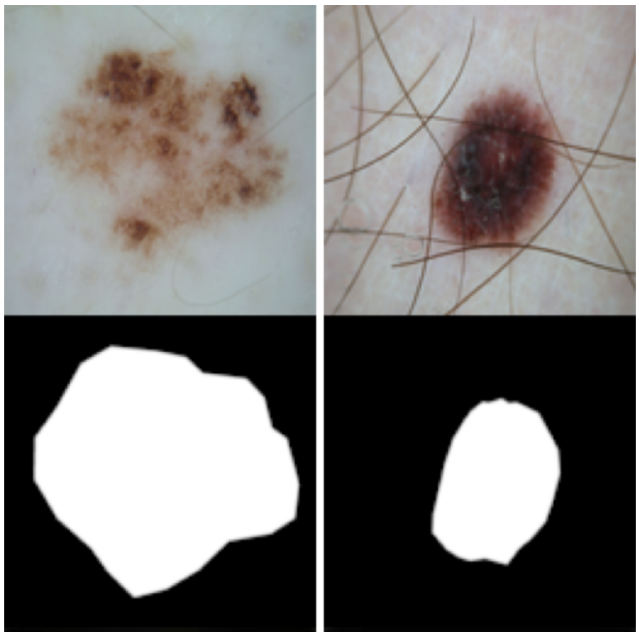


Figura 3.1. In alto due immagini originali di lesioni, in basso le rispettive segmentazioni.

- ISIC Challenge 2017: questa sfida era divisa in tre task, di cui i primi due uguali all'edizione precedente, mentre il terzo consisteva in due classificazioni binarie separate, le quali coinvolgevano tre tipi diversi di diagnosi: melanoma, nevo benigno e cheratosi seborroica.
  Per il primo compito di classificazione binaria l'obiettivo era quello di distinguere tra: (a) melanomi e (b) nevi o cheratosi seborroica.
  Per la seconda classificazione binaria, veniva invece richiesto di distinguere tra: (a) cheratosi seborroica e (b) nevi o melanomi.
  Per questa edizione veniva fornito un training set di 2000 immagini etichettate ed un test set contenente circa 600 immagini.

- ISIC Challenge 2018: la sfida era suddivisa come nel 2017, ma il terzo task consisteva nella classificazione multiclasse delle lesioni in una delle seguenti sette categorie: melanoma, nevo melanocitico, cheratosi attinica, cheratosi benigna, lesioni vascolari, dermatofibroma o carcinoma basocellulare.
  Il training set di questa edizione era composto da circa 11000 immagini etichettate, mentre il test set da circa 1500.

[2] Dataset da usare per addestrare il proprio modello.
[3] "Etichetta" che specifica a quale classe appartiene ciascuna lesione.
[4] Dataset da usare per testare il proprio modello.

- ISIC Challenge 2019: questa sfida era divisa in due task, aventi però lo stesso obiettivo di classificare le lesioni in nove categorie diverse: melanoma, nevo melanocitico, cheratosi attinica, cheratosi benigna, lesioni vascolari, dermatofibroma, carcinoma a cellule squamose, carcinoma basocellulare o nessuna delle precedenti. La differenza tra i due task era che per il primo bisognava usare un training set di 25331 ed un test set di 8238 immagini etichettate, mentre per il secondo venivano usati dei metadati[5] associati ad esse.
- ISIC Challenge 2020: il task principale di questa sfida consisteva nella classificazione binaria tra lesioni maligne (melanoma) e lesioni benigne, potendo contare su un training set di 33126 ed un test set di 10982 immagini, con label e metadati associati ad esse.

Io, avendo a disposizione delle risorse di calcolo limitate, ho deciso di lavorare sulla sfida che richiedesse un costo computazionale minore, ossia la ISIC Challenge del 2016. In particolare mi sono cimentato nel terzo task: la classificazione binaria delle lesioni in benigne e maligne.

[5] Informazioni aggiuntive sulle immagini, come ad esempio la parte del corpo dove si trova la lesione, il sesso e l'età del paziente.

Capitolo 4

Deep Learning

4.1 Cosa è il Deep Learninig

Il deep learning è un metodo di apprendimento automatico (machine learning) che si basa sulla costruzione di reti neurali profonde per apprendere ed elaborare informazioni. Una rete neurale profonda è semplicemente una rete neurale composta da molti strati (o *layer*), ciascuno dei quali ha un ruolo importante nell'estrazione e nell'elaborazione di specifiche informazioni, a partire dai dati di input.

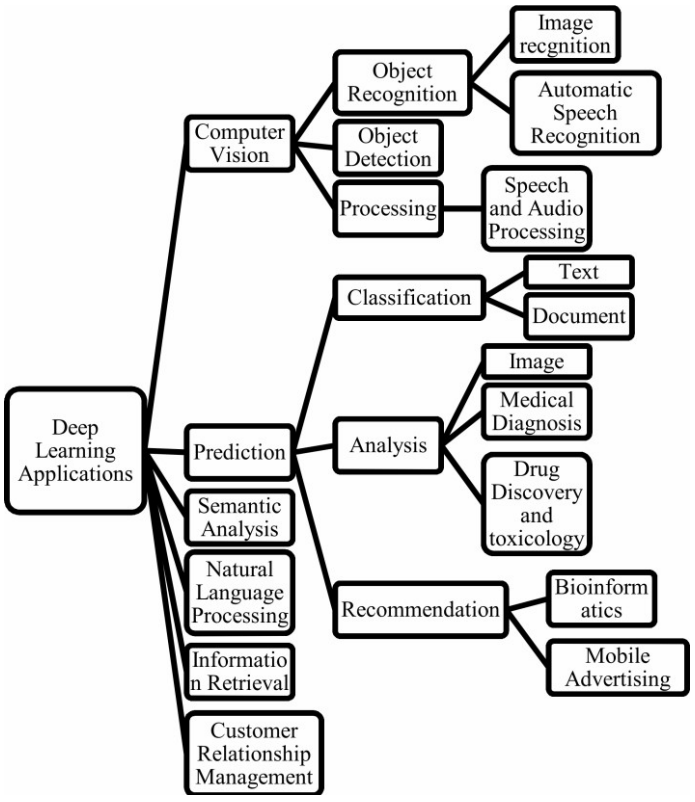


Figura 4.1. Applicazioni del deep learning.

Le tecniche di deep learning sono utilizzate in numerosi campi che necessitano l'elaborazione e l'analisi di grandi quantità di dati e informazioni complesse. Alcuni esempi di applicazioni del deep learning includono i sistemi di riconoscimento facciale,

la rilevazione di oggetti, la traduzione automatica del linguaggio naturale, la guida autonoma di veicoli, l'analisi dei mercati finanziari, l'automazione industriale e molti altri ancora.

4.2 Reti Neurali Convoluzionali

Un esempio importante di reti neurali profonde sono le **Reti Neurali Convoluzionali** (*Convolutional Neural Networks* o CNN), utilizzate principalmente per l'analisi di immagini e video. L'idea di base di queste reti è simile a quella delle reti neurali artificiali (ANN) tradizionali, in quanto sono composte da neuroni che si ottimizzano attraverso l'auto-apprendimento, ma sussistono alcune importanti differenze con queste ultime.

Una rete neurale convoluzionale è caratterizzata da diversi strati, ognuno dei quali si occupa di diverse fasi di elaborazione dell'immagine. Ecco una breve descrizione degli strati principali:

1. Strato di input: le CNN prendono in input delle immagini con la struttura di una matrice tridimensionale, chiamata "tensore". Le dimensioni del tensore rappresentano rispettivamente la larghezza, l'altezza e i canali di colore dell'immagine. Un'immagine a colori RGB (red, green, blue) ha tre canali, uno per ogni colore (rosso, verde e blu); mentre una foto in bianco e nero ne ha uno solo (scala di grigi). Ad esempio, un'immagine a colori di dimensioni 224x224 pixel avrà come formato di input (224, 224, 3); una in bianco e nero con le stesse dimensioni comparirà invece con il formato (224, 224, 1).

2. Strato convoluzionale: in questo strato le immagini vengono elaborate mediante l'uso di **filtri di convoluzione** (o *kernel*), ovvero matrici di piccole dimensioni che permettono di estrarre delle caratteristiche specifiche come bordi, linee, angoli, curve, ecc.

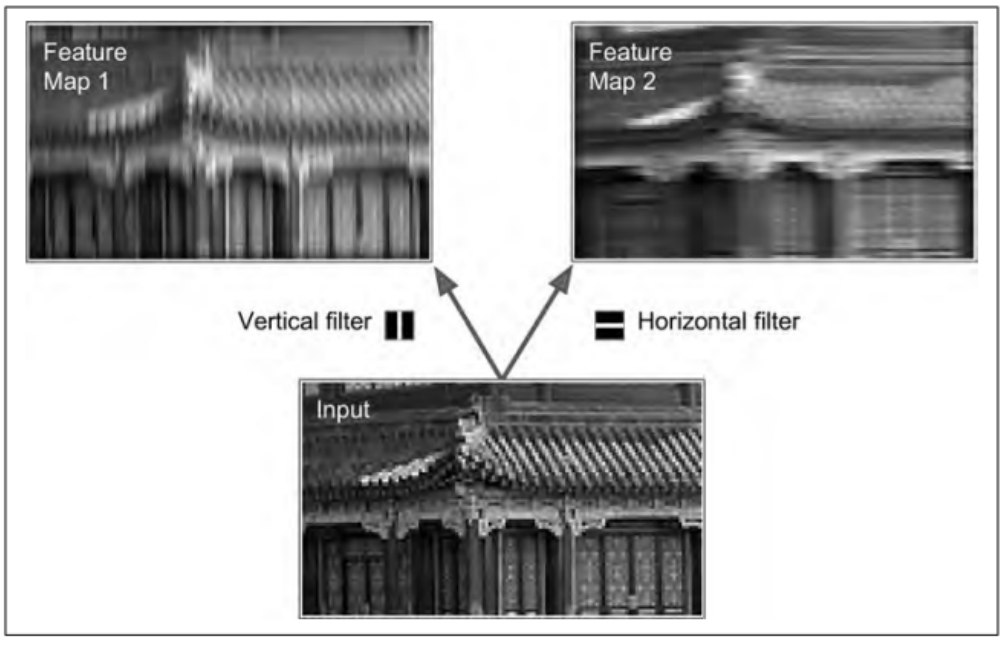


Figura 4.2. Applicazione di due diversi filtri di convoluzione.

Nella pratica, l'operazione di convoluzione consiste nel moltiplicare il valore dei pixel dell'immagine di input per i valori corrispondenti del filtro, sommare il tutto e assegnare il risultato al pixel corrispondente nella mappa delle caratteristiche (o *feature map*). Questa operazione viene poi ripetuta spostando il filtro lungo l'immagine di input con uno stride[1] specifico, ottenendo così una feature map che evidenzia le aree dell'immagine che sono più simili al filtro utilizzato.

3. Strato di **pooling**: viene usato per diminuire la dimensione dei dati di input e di conseguenza ridurre il carico computazionale, l'uso della memoria e il numero di parametri (limitando il rischio di overfitting[2]).
   Il layer di pooling seleziona una piccola regione (ad esempio 2x2 o 3x3) dell'immagine di input e ne calcola il valore massimo (*Max Pooling*) o medio (*Average Pooling*), assegnando poi questo valore all'area corrispondente della feature map. La figura 4.3 mostra un layer di pooling con kernel di dimensioni 2x2 e uno stride uguale a 2, che dunque ritornerà in output un'immagine due volte più piccola in entrambe le direzioni (ossia con un area quattro volte più piccola).

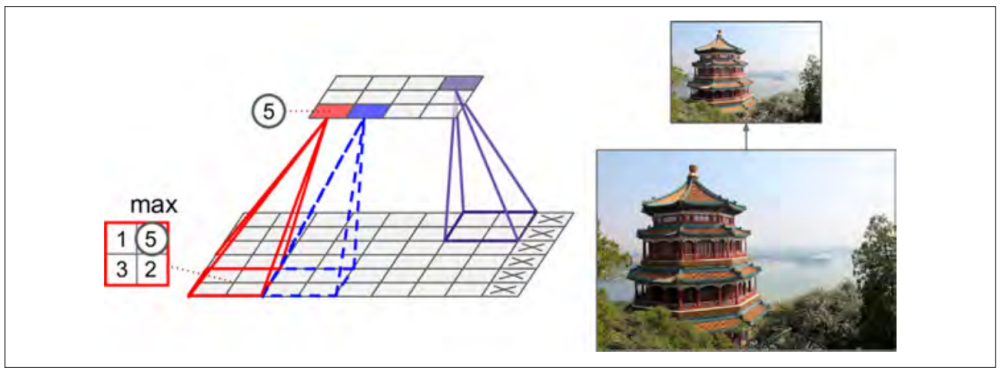


Figura 4.3. Esempio di un layer di Max Pooling (kernel 2x2 e stride 2).

4. Strato **completamente connesso** (o *fully connected*): corrisponde alla struttura delle classiche ANN, in cui ogni singolo neurone è connesso a tutti i neuroni del layer precedente e di quello successivo. Gli input che sono forniti ai neuroni di uno strato fully connected vengono moltiplicati per i relativi pesi e poi sommati insieme per ottenere il neurone di output. Questo tipo di layer, anche detto *Dense Layer*, si trova alla fine di una rete neurale convoluzionale, poiché è adatto alla classificazione delle immagini a partire dalle caratteristiche estratte nei processi di convoluzione e pooling.

5. Strato di output: contiene un numero o un vettore di numeri, il quale indica l'etichetta che identifica l'immagine di input nella propria classe di appartenenza.

In mezzo agli strati precedenti vengono spesso applicati altri due layer molto importanti: la normalizzazione batch e la funzione di attivazione.

[1] Parametro di un layer che indica di quanti passi deve spostarsi il kernel all'interno della matrice dell'immagine.

[2] La tendenza di una rete neurale ad adattarsi troppo bene ai dati di training, diventando incapace di generalizzare bene su dati che non ha mai esaminato prima.

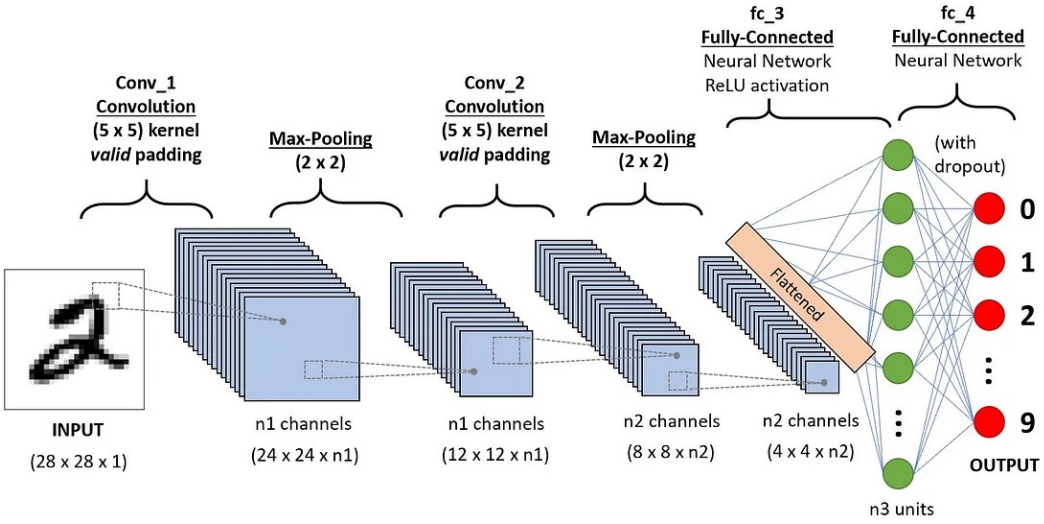


Figura 4.4. Esempio di una CNN completa per il riconoscimento di cifre scritte a mano.

La **normalizzazione batch** (in inglese *Batch Normalization*, BN) consiste in una trasformazione lineare che viene applicata ai dati di input. In pratica questo livello ha il compito di prendere dei mini-batch di dati, ossia un piccolo sottoinsieme del dataset completo, e normalizzarli in modo che siano più "bilanciati". Questo strato ha la funzione di velocizzare e stabilizzare il processo di apprendimento.

In seguito alla normalizzazione batch, viene applicata una **funzione di attivazione** (in inglese *activation function*), ossia una funzione matematica che trasforma l'output dei neuroni in forme non lineari. Per aggiungere non linearità allo strato di convoluzione viene solitamente applicata la Rectified Linear Unit (ReLU), una funzione che trasforma i valori negativi in zero e lascia i valori positivi inalterati. Altre funzioni di attivazione che vengono spesso usate nelle CNN sono: la Sigmoide, che produce un output compreso tra 0 e 1 e viene utilizzata per problemi di classificazione binaria; la tangente iperbolica (Tanh), che è simile alla sigmoide, ma produce output compresi tra -1 e 1 ed è particolarmente utile per la classificazione multi-classe; la Softmax, la quale viene spesso usata come funzione di attivazione nell'ultimo strato di una CNN, poiché trasforma un vettore di valori in un vettore di probabilità, indicando così la previsione finale.

4.3 ResNet-50

Un grande svantaggio che si presenta durante l'addestramento di reti neurali profonde è il cosiddetto "**Vanishing Gradient Problem**" (problema della scomparsa del gradiente). Questo problema si verifica quando, durante la fase di retropropagazione dell'errore (o *backpropagation*), il valore del gradiente diminuisce talmente tanto da apportare modifiche quasi nulle ai pesi.
Per essere più chiari, il gradiente viene calcolato come il prodotto delle derivate parziali delle funzioni di attivazione che si incontrano lungo il percorso di retropropagazione. Visto che gran parte delle funzioni di attivazione che vengono utilizzate hanno valori di output compresi tra 0 e 1, il prodotto di questi numeri può diventare sempre più piccolo. Per questo motivo, diventa problematico aggiornare i pesi dei

primi strati della rete, poiché il gradiente che viene usato per aggiornarli diminuisce talmente tanto da non riuscire a produrre un cambiamento significativo. Di conseguenza l'addestramento di una rete che presenta questo problema può subire grandi rallentamenti o addirittura l'arresto totale.

Per aggirare il problema della scomparsa del gradiente, nel 2015 i ricercatori Kaiming He, Xiangyu Zhang, Shaoqing Ren e Jian Sun svilupparono una delle reti neurali convoluzionali profonde più innovative ed efficaci in varie applicazioni della computer vision: l'architettura **ResNet**.
ResNet (che sta per Residual Network) è una rete residuale, ossia una rete neurale profonda che utilizza un'architettura a **blocchi residuali**, unità di elaborazione che consentono di migliorare l'apprendimento e prevenire il problema della scomparsa del gradiente. I blocchi residuali sono composti da un percorso principale e da un collegamento residuo: il percorso principale è costituito da una serie di strati che elaborano l'input del blocco, mentre il collegamento residuo, o *shortcut*, è una connessione diretta tra l'input e l'output del blocco, che permette dunque di saltare alcuni degli strati principali.

Esistono diversi tipi di blocchi residuali; in particolare nell'architettura Resnet-50 vengono usati dei blocchi detti *Bottleneck*, i quali sono costituiti da tre strati convoluzionali:

- uno strato con filtri convolutivi di dimensione 1x1, che serve a ridurre la dimensione dell'input;
- uno strato con filtri 3x3, necessario per l'estrazione delle caratteristiche;
- un altro strato con filtri 1x1, usato per espandere la dimensione dell'output, ripristinando quella originale.

Le prime due convoluzioni sono seguite da una normalizzazione del batch e una funzione di attivazione ReLU, mentre l'ultima convoluzione è seguita solo da una normalizzazione batch. All'output dell'ultimo strato convoluzionale viene sommato l'input originale del blocco tramite un collegamento residuo e viene poi applicata una funzione di attivazione ReLu (Figura 4.5 a sinistra). Inoltre, se le dimensioni dell'input e dell'output non corrispondono, sul ramo di shortcut viene utilizzata una convoluzione 1x1 per riadattare le dimensioni, seguita da una normalizzazione batch (Figura 4.5 a destra). Un blocco con una shortcut di questo tipo viene chiamato *blocco convoluzionale*.

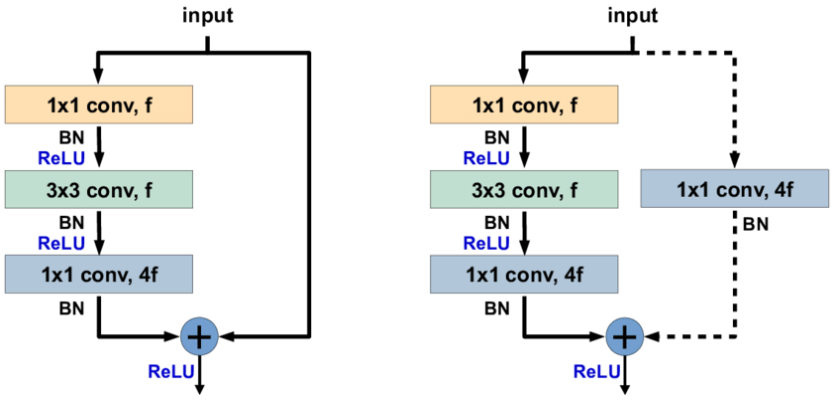


Figura 4.5. Blocco Bottleneck usato in ResNet-50.

ResNet-50 è una variante di ResNet composta da 50 layer. La sua struttura può essere suddivisa in più fasi:

1. Strato di Input: ResNet-50 prende in input immagini a colori con un formato standard di 224x224x3.

2. Tecnica di *Zero Padding*: questa tecnica consiste nell'aumentare la dimensione di una matrice di dati aggiungendo dei pixel con valore zero attorno ai bordi della matrice. Applicando questa funzione il formato delle matrici di input diventa 230x230x3.

3. conv1: un primo strato convoluzionale, composto da una convoluzione 7x7 con 64 filtri e stride 2, una normalizzazione del batch e una funzione di attivazione ReLu.

4. MaxPool: un layer di Max Pooling con un filtro 3x3 e stride 2, il quale è preceduto da un'altra funzione di Zero Padding.

5. conv2_x: un blocco residuale composto da 3 sottoblocchi "bottleneck" con 64 filtri per ciascun sottostrato convoluzionale. Il primo dei sottoblocchi (conv2_1) è un blocco convoluzionale, poiché è necessario riadattare le dimensioni dell'output del blocco residuale precedente.

6. conv3_x: un blocco residuale composto da 4 sottoblocchi "bottleneck" con 128 filtri per ciascun sottostrato convoluzionale. Il primo dei sottoblocchi (conv3_1) è un blocco convoluzionale.

7. conv4_x: un blocco residuale composto da 6 sottoblocchi "bottleneck" con 256 filtri per ciascun sottostrato convoluzionale. Il primo dei sottoblocchi (conv4_1) è un blocco convoluzionale.

8. conv5_x: un blocco residuale composto da 3 sottoblocchi "bottleneck" con 512 filtri per ciascun sottostrato convoluzionale. Il primo dei sottoblocchi (conv5_1) è un blocco convoluzionale.

9. AvgPool: un layer di Average Pooling 7x7.

10. Strato completamente connesso: un layer fully connected che produce un vettore di output di lunghezza 1000, necessario per la classificazione multiclasse delle immagini in 1000 classi diverse. Questo strato è seguito da una funzione di attivazione Softmax che converte i punteggi di classificazione dell'output in probabilità.

Il motivo per cui il layer fully connected è composto da 1000 neuroni è che ResNet-50 è stata progettata per risolvere il problema della classificazione multiclasse delle immagini su un vasto dataset chiamato ImageNet. Questo dataset contiene oltre un milione di immagini, etichettate per l'appunto in 1000 classi diverse. Il vettore di output rappresenta dunque la probabilità che l'immagine di input appartenga ad ognuna delle mille categorie del dataset ImageNet.

ResNet-50 viene molto usato in applicazioni di computer vision poiché è un modello pre-addestrato, ossia un modello che è già stato allenato su un dataset molto vasto e generico (ImageNet). Un modello pre-addestrato, avendo già imparato ad estrarre le caratteristiche generali delle immagini, si può semplicemente adattare al nostro dataset tramite la tecnica del **fine-tuning**. Questa tecnica consiste nell'addestrare ulteriormente un modello che è già stato addestrato, come ResNet-50, su un dataset

più piccolo e specifico per il problema che si vuole risolvere. In questo modo si migliora la precisione del modello e si riduce notevolmente il tempo di addestramento.

Nella tabella in Figura 4.6 viene riassunta la struttura dell'architettura ResNet in tutte le sue versioni principali. Si può facilmente intuire che le versioni di Resnet con un numero più alto di layer sono quelle più efficienti in termini di risultati e accuratezza. Tuttavia, un alto numero di layer corrisponde ad un crescente numero di parametri e, di conseguenza, ad un costo computazionale maggiore. Per questo motivo, ResNet-50 diventa una scelta molto popolare quando si tratta di applicazioni in cui è importante avere una buona predizione, tenendo però conto della velocità di addestramento della rete.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 4.6. Comparazione dei vari modelli ResNet.

Capitolo 5

Algoritmo

5.1 Ambiente di lavoro

Google Colab è un editor sviluppato da Google che permette di scrivere ed eseguire codici Python direttamente sul Cloud, sfruttando i Jupyter Notebook. Questi sono dei documenti interattivi che permettono di suddividere il nostro programma in celle di testo e celle di codice eseguibili singolarmente.

Figura 5.1. Esempio di Jupyter Notebook su Google Colab.

Colab ha varie caratteristiche che lo rendono estremamente utile e facile da usare: permette di sfruttare gratuitamente, anche se con dei limiti di utilizzo, la potenza di calcolo (CPU, GPU e TPU) fornita da Google, a prescindere dalle specifiche del proprio calcolatore; non necessita di nessuna configurazione; tutti i blocchi note Colab che vengono scritti si salvano automaticamente sull'account Drive collegato, per cui si può lavorare ai propri progetti anche cambiando dispositivo; si possono sfruttare le librerie di Python senza doverle scaricare sulla propria macchina, ma semplicemente importandole; sono presenti vari tutorial utili su come utilizzare alcune funzionalità di Colab e alcune librerie di Python.
Per tutti i motivi elencati precedentemente ho deciso di lavorare su questa comoda piattaforma, la quale viene usata da molti sviluppatori per l'esecuzione di algoritmi di addestramento complessi come quelli di deep learning.

5.2 Dataset e preparazione dati

Come già accennato nel Paragrafo 3.2, i partecipanti della ISIC Challenge devono usare un dataset di immagini che viene fornito al momento dell'iscrizione. Per il task nel quale mi sono concentrato (classificazione binaria dei nei in benigni e maligni) venivano fornite 900 immagini in formato JPEG per il training set e 379 per il test set. Venivano inoltre forniti due file csv, detti "Ground Trouth", ognuno contenente due colonne ed un numero di righe pari a quello delle immagini del rispettivo dataset. La prima colonna di ogni riga contiene una stringa nella forma *ISIC_<image_id>*, dove *<image_id>* corrisponde al nome del file JPEG. La seconda colonna invece contiene le label corrispondenti all'immagine della prima colonna. In particolare, nel Training Ground Trouth (il file csv corrispondente al training set) i possibili valori della seconda colonna sono "benign", che rappresenta una lesione benigna, e "malignant", che rappresenta una lesione maligna; mentre nel Test Ground Trouth i valori sono "0.0" e "1.0", i quali rappresentano rispettivamente una lesione benigna e una maligna.
Tutti questi dati sono disponibili pubblicamente nella sezione "Data" del sito ufficiale della ISIC Challenge (vedi Bibliografia[12]).

ISIC_0000001	benign
ISIC_0000002	malignant
ISIC_0000004	malignant
ISIC_0000006	benign

Figura 5.2. Prime quattro righe del Training Ground Trouth.

Per la preparazione dei dati, come prima cosa ho creato sul mio account Drive due cartelle, una per il training set e una per il test set, in cui ho creato due sottocartelle nominate "Maligno" e "Benigno" e salvato tutte le immagini. Successivamente ho realizzato una funzione (Figura 5.3) che prende in input due parametri: il path del file csv contenente il Ground Trouth ed il path della cartella in cui spostare le immagini. Scansionando il file csv riga per riga, la funzione ad ogni iterazione sposta un'immagine nella rispettiva sottocartella.

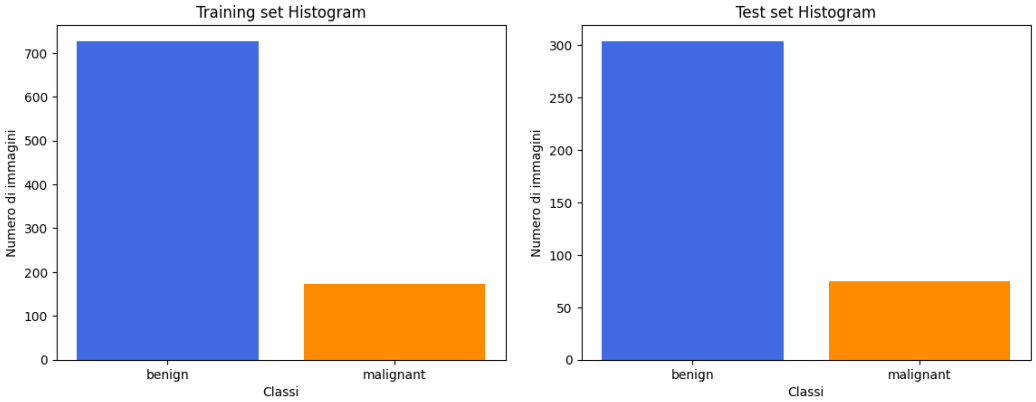
```
def move_ben_mal(file_csv, dest)
  with open(file_csv) as file_obj:
  reader = csv.reader(file_obj)

  # itero ogni riga del file csv
  for row in reader:
    if  os.path.exists(dest+row[0]+'.jpg'):
      if row[1] == 'benign' or row[1] == "0.0":
        shutil.move(dest+row[0]+'.jpg', dest+'Benigno')
      elif row[1] == 'malignant' or row[1] == "1.0":
        shutil.move(dest+row[0]+'.jpg', dest+'Maligno')
      else:
        print("errore: label non riconosciuta")

move_ben_mal(csv_dir +'ISBI2016_ISIC_Part3_Training_GroundTruth.csv', train_dir)
move_ben_mal(csv_dir+'ISBI2016_ISIC_Part3_Test_GroundTruth.csv', test_dir)
```

Figura 5.3. Funzione che sposta le immagini nelle rispettive sottocartelle.

Dopo aver scaricato e spostato le immagini nelle varie cartelle, usando la libreria *Matplotlib* ho implementato una funzione che stampa un istogramma raffigurante la distribuzione delle due classi in ciascuno dei due dataset.
Come si può vedere in Figura 5.4, il dataset è molto sbilanciato, ovvero il numero di campioni benigni è molto superiore a quello dei campioni maligni, sia nel training set che nel test set.



(a) Training Set. **(b)** Test Set.

Figura 5.4. Istogrammi che rappresentano la distribuzione delle due classi "Benigno" e "Maligno" all'interno dei dataset.

In particolare il training set contiene 727 campioni benigni e 173 maligni, mentre il test set ne contiene 304 benigni e 75 maligni. Questo sbilanciamento porta con sé delle problematiche nell'addestramento di un classificatore binario, in quanto il modello avrà grandi difficoltà nel predire correttamente la classe minoritaria (quella con meno campioni). Per cercare di ridurre gli effetti di questo sbilanciamento si possono applicare varie tecniche. Le principali sono:

- *Undersampling*: questa tecnica consiste nell'eliminare delle istanze della classe maggioritaria dai dati, in modo tale da avere un dataset più bilanciato;
- *Oversampling*: è il contrario dell'undersampling. Consiste nell'aggiungere delle istanze della classe minoritaria al dataset. Per fare ciò si possono prendere delle immagini da un dataset separato, aggiungere al dataset delle copie della classe minoritaria o applicare tecniche di *data augmentation*[1] al dataset originale in modo da bilanciarlo;
- *Class Weighting*: questo metodo regola la funzione di costo del modello in modo tale che l'errata classificazione di un campione della classe minoritaria sia penalizzata maggiormente rispetto all'errata classificazione di un campione della classe maggioritaria. In questo modo, riequilibrando la distribuzione delle classi dovrebbe migliorare l'accuratezza del modello.

Questi metodi spesso vengono usati congiuntamente. Io nel mio algoritmo ho deciso di usare la tecnica dell'oversampling, applicando data augmentation maggiormente sulla classe minoritaria, insieme alla tecnica del Class Weighting.

[1] Tecniche che ampliano il dataset senza raccogliere nuovi campioni, ma semplicemente applicando delle trasformazioni alle immagini originali, realizzandone delle copie modificate.

5.3 Data Augmentation

Il data augmentation è una tecnica usata nella computer vision per aumentare la quantità dei dati di addestramento, semplicemente applicando delle trasformazioni ai dati presenti nel training set originale, con l'obiettivo di ridurre il rischio di overfitting. Ad esempio, se partendo da un'immagine originale volessi creare delle copie modificate di quest'ultima, potrei ruotarla, ribaltarla, ingrandirla, cambiare la luminosità o la brillantezza...

Come anticipato nel paragrafo precedente, nel mio algoritmo ho sfruttato questa tecnica anche per fare un'oversampling sui dati della classe minoritaria. Ho infatti applicato delle trasformazioni al training set, creando quattro copie modificate per ogni immagine originale appartenente alla classe dei nei maligni (quella con meno campioni) e una sola per le immagini della classe benigna. In questo modo il numero di immagini appartenenti alla classe minoritaria è quintuplicato mentre quello della classe maggioritaria è semplicemente raddoppiato, rendendo il dataset meno sbilanciato.

Per applicare le tecniche di data augmentation al mio dataset ho sfruttato la funzione *ImageDataGenerator* della libreria Keras, come mostrato in Figura 5.5.

```
datagen = ImageDataGenerator(rotation_range=90,
                             shear_range=15,
                             zoom_range=0.2,
                             fill_mode='reflect',
                             vertical_flip=True,
                             horizontal_flip=True,
                             )
```

Figura 5.5. Operazioni di data augmentation.

Questa funzione prende come parametri tutte le trasformazioni con cui si vuole aumentare il proprio dataset, le quali vengono applicate casualmente, e ritorna un generatore di batch di tensori corrispondenti alle immagini modificate.
Per questo problema ho deciso di applicare delle operazioni semplici quali la rotazione tra 0 e 90 gradi, l'inclinazione (o *shearing*) tra 0 e 15 gradi, l'ingrandimento del $\pm 20\%$ e il ribaltamento sia sull'asse verticale che su quello orizzontale.

Dopo aver creato questo "generatore", ho definito due array, chiamati *train_data* e *train_target*, nei quali ho salvato rispettivamente i tensori corrispondenti alle immagini, sia quelle originali che quelle ottenute applicando la funzione *flow* del generatore, e le label corrispondenti alle classi di appartenenza (0 per la classe benigna e 1 per quella maligna). Infine ho normalizzato i valori dei pixel di ogni immagine (contenuti nell'array *train_data*) nell'intervallo compreso tra 0 e 1.

5.4 Creazione della rete

Per creare la mia rete neurale convoluzionale ho deciso di applicare la tecnica del fine-tuning a partire da ResNet-50. Ho quindi importato la struttura e i pesi di questo modello pre-addestrato tramite la funzione *Resnet50* della libreria Keras. Successivamente ho aggiunto uno strato completamente connesso (*Dense* layer) contenente un solo neurone, seguito da una sigmoide.

```python
# creo il modello
resnet = ResNet50()
x = Flatten()(resnet.output)
prediction = Dense(1, activation='sigmoid')(x)
model = Model(inputs=resnet.input, outputs=prediction)


# compilo il modello
model.compile(
  loss='binary_crossentropy',
  optimizer=Adam(learning_rate=1e-4),
  metrics=['accuracy', AUC(name='auc')]
)
```

Figura 5.6. Creazione e compilazione del modello.

Questa funzione di attivazione ha la seguente forma matematica:

$$f(x) = \frac{1}{1 + e^{-x}}$$

dove "x" corrisponde al valore contenuto nel neurone ed "e" è il numero di Eulero [2]. La proprietà della sigmoide è quella di comprimere il valore di input nell'intervallo [0,1], restituendo quindi un valore che può essere interpretato come una probabilità e che in questo caso corrisponde alla predizione finale del modello. Se la predizione è compresa tra 0 e 0.5 vorrà dire che l'immagine di input verrà classificata come appartenente alla classe 0, che corrisponde ai nei benigni. Se invece è compresa tra 0.5 e 1, vorrà dire che l'immagine di input appartiene alla classe 1, quella dei nei maligni.

Dopo aver definito l'architettura del modello, l'ho compilato mediante la funzione *compile*. Compilare un modello corrisponde a definire il suo processo di addestramento, ovvero in che modo verranno aggiornati i pesi per ottimizzare le performance del modello. In particolare, vanno specificati quattro parametri:

1. **Loss Function**: è una funzione che serve a valutare se l'algoritmo sta facendo un buon lavoro nel modellare i dati. Questa funzione calcola la "distanza" tra l'output attuale e quello atteso, usando poi questa misura come feedback per modificare il modo (o la direzione) in cui l'algoritmo regola i pesi della rete per migliorare le predizioni. Questa regolazione è proprio ciò che viene chiamato apprendimento.
   Per essere più chiari, se un valore predetto è completamente diverso dal suo valore atteso, allora l'output della loss function sarà molto alto. Se invece il valore predetto si avvicina a quello atteso, il valore della loss sarà basso. Essendo questo un problema di classificazione binaria, ho usato come funzione di loss la *Binary Cross Entropy Loss*, una scelta diffusa per problemi di questo tipo.

2. **Algoritmo di ottimizzazione**: è l'algoritmo che viene eseguito per aggiornare i pesi della rete neurale durante il suo addestramento, cercando quelli ottimi.

[2] Numero irrazionale fondamentale nei campi della matematica, della fisica e della scienza; il suo valore approssimativo è 2.718;

L'algoritmo di ottimizzazione (o *optimizer*) che ho scelto è Adam, il quale utilizza una combinazione di stime del primo e del secondo momento del gradiente per adattare il tasso di apprendimento di ogni peso in modo dinamico, rendendolo più robusto rispetto ad altri optimizer come AdaGrad o RMSP.

3. **Learning Rate**: è un iperparametro usato negli algoritmi di ottimizzazione che indica di quanto bisogna modificare i pesi ad ogni iterazione. La scelta del learning rate (o *tasso di apprendimento*) è estremamente importante e determina quanto velocemente il modello deve imparare dai dati. Questa scelta non è banale, in quanto un valore troppo piccolo potrebbe comportare una convergenza molto lenta o, in alcuni casi, potrebbe addirittura impedire al modello di convergere del tutto, interrompendo quindi il processo di addestramento. Al contrario, un valore troppo alto porterà il modello ad un apprendimento troppo veloce e oscillante, senza trovare una serie di pesi ottimale.

4. **Metriche di valutazione**: sono le metriche con le quali vengono valutate le prestazioni del modello durante il suo addestramento. In questo algoritmo ho utilizzato due metriche di valutazione, che verranno approfondite meglio nel Paragrafo 5.6:

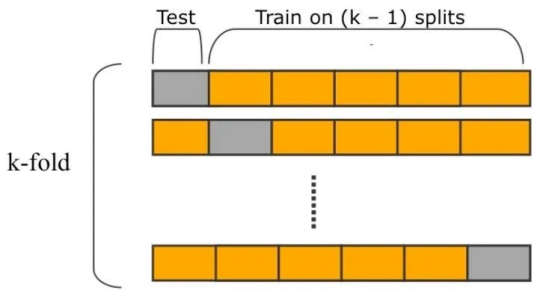
   - *Accuracy*: indica la percentuale di predizioni corrette rispetto a quelle totali.
   - *AUC (Area Under the Curve)*: rappresenta l'area al di sotto della *ROC Curve* e valuta la capacità di un modello di distinguere tra due classi.

Dopo aver creato e compilato la rete, si può finalmente passare alla parte più importante e complessa di tutto l'algoritmo: l'addestramento.

5.5 Addestramento

Per addestrare il mio modello ho applicato la tecnica del Leave-One-Out Cross Validation (LOOCV), utilizzata per valutare l'accuratezza di un modello predittivo durante il suo addestramento.

Come per il K-fold cross validation, in questa tecnica il training set viene diviso in K parti contenenti lo stesso numero di immagini, dove, in questo caso, K corrisponde anche al numero di iterazioni dell'addestramento della rete. Ad ogni iterazione, il modello viene allenato su K-1 parti del training set e testato sulla parte rimanente, che viene chiamata **validation set**.



In questo modo, ci si assicura che tutte le K parti del training set vengono utilizzate sia per testare che per allenare il modello, rendendone la valutazione più accurata e affidabile.

Per implementare questa tecnica ho usato la funzione *StratifiedKFold* della libreria Scikit-Learn (sklearn), la quale prende come parametro principale il numero k di cartelle in cui si vuole dividere il dataset e restituisce k array di indici per splittare (dividere) il dataset in training e validation set.

```
EPOCHS = 25
hist = {'loss':[], 'accuracy':[], 'auc':[], 'val_loss':[], 'val_accuracy':[], 'val_auc':[]}
yt = []   # Array che conterrà tutte le etichette corrette
yp = []   # Array che conterrà tutte le predizioni
t0 = time.time()  # Faccio partire un timer per misurare il tempo totale di training

for train,val in StratifiedKFold(8, shuffle=True).split(train_data,train_target):
  # divido il dataset in training e validation folds
  x_train, x_val, y_train, y_val = train_data[train], train_data[val], \
                                   train_target[train], train_target[val],

  history = model.fit(
    x_train,
    y_train,
    validation_data=(x_val, y_val),
    epochs=EPOCHS,
    class_weight=class_weights,
    shuffle=True,
    callbacks=[early_stop]
  )
  hist['loss'].append(np.mean(history.history['loss']))
  hist['accuracy'].append(np.mean(history.history['accuracy']))
  hist['auc'].append(np.mean(history.history['auc']))
  hist['val_loss'].append(np.mean(history.history['val_loss']))
  hist['val_accuracy'].append(np.mean(history.history['val_accuracy']))
  hist['val_auc'].append(np.mean(history.history['val_auc']))

  y_pred = model.predict(x_val)
  yt.append(y_val)
  yp.append(y_pred)


t1 = time.time()
print('Tempo di training in secondi: ', int(t1-t0))

yt = np.concatenate(yt)
yp = np.concatenate(yp)
```

Figura 5.7. Addestramento della rete.

La particolarità di questa funzione è che garantisce che la percentuale di campioni di ogni classe all'interno di ciascuna cartella creata sia uguale alla percentuale di campioni di ogni classe nell'intero dataset. Questa funzione risulta quindi particolarmente utile in presenza di dataset sbilanciati, poiché assicura che in ogni cartella sia contenuto un numero equilibrato di campioni di tutte le classi, senza rischiare di trascurare la classe minoritaria in qualche iterazione del processo di addestramento.

Ho deciso quindi di dividere il dataset in 8 fold (cartelle), così da avere un addestramento diviso in 8 iterazioni (effettuate attraverso un ciclo for), ognuna delle quali usa sette fold per allenare il modello e una per valutarlo tramite le metriche specificate precedentemente.

Ad ogni iterazione viene chiamata la funzione *fit* la quale, come si capisce dal nome, effettua il "fitting" del modello sui dati di training, ossia la ricerca dei pesi che minimizzano la loss function. Uno dei tanti parametri che ho utilizzato in questa funzione è *class_weight*, un dizionario che ha come chiavi i nomi delle varie classi e come valori i pesi da dare alle rispettive classi. In questo modo si può effettuare la tecnica del class weighting (introdotta nel Paragrafo 5.2) per contrastare gli effetti negativi dello sbilanciamento del dataset. Per trovare i pesi ottimali da applicare a ciascuna classe ho utilizzato la funzione *compute_class_weight* di Scikit-Learn.

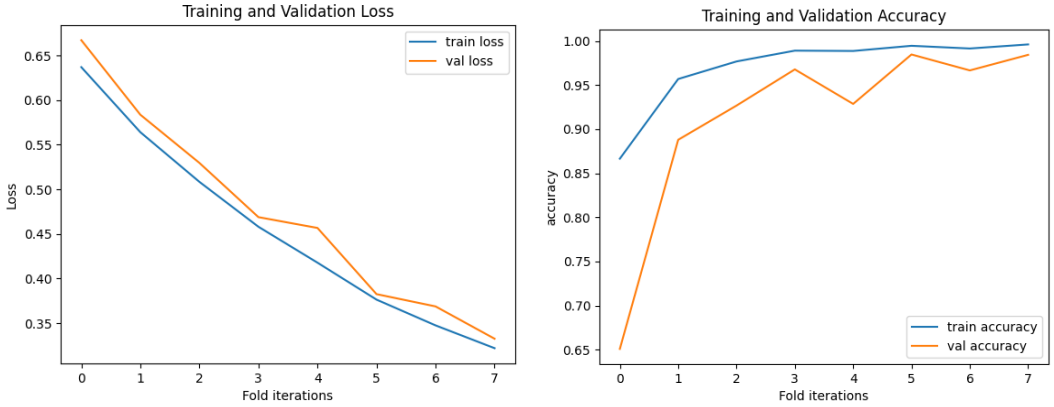
```
class_weights = compute_class_weight('balanced', classes=np.unique(train_target), y=train_target)
class_weights = {0: class_weights[0], 1: class_weights[1]}
print(class_weights)
```

```
{0: 0.797455295735901, 1: 1.3404624277456647}
```

Figura 5.8. Funzione per il Class Weighting.

Inoltre, per controllare l'andamento della fase di training, ho salvato la media di tutte le epoche di ogni iterazione, facendo poi un grafico per ogni metrica di valutazione usata. In Figura 5.9 sono presenti questi grafici (quello della AUC l'ho tralasciato poiché ha un andamento praticamente identico a quello della accuracy), indicando sull'asse delle ascisse le iterazioni, e sull'asse delle ordinate il valore rispettivamente della loss e della accuracy. Inoltre i grafici contengono due curve diverse: la curva blu si riferisce all'andamento delle due funzioni rispetto ai dati usati per addestrare il modello in ogni iterazione, mentre quella arancione rappresenta l'andamento delle due funzioni rispetto ai dati usati come validation set.

Si può facilmente notare l'assenza di grandi oscillazioni nel processo di addestramento e come entrambe le funzioni abbiano l'andamento atteso: la loss function scende durante la fase di addestramento tendendo a zero e al contrario la accuracy sale cercando di convergere a uno (overfitting).



(a) Andamento della loss **(b)** Andamento della accuracy

Figura 5.9. Grafici della loss function e della accuracy nella fase di training.

Infine, come ultimo elemento di analisi per questo addestramento ho deciso di generare una **Confusion Matrix**: una matrice che mostra il numero di predizioni corrette ed errate fatte per ciascuna classe.
Nella confusion matrix vengono riportati quattro valori diversi:

- True Negative (TN): il numero di predizioni corrette per i campioni appartenenti alla classe negativa, che in questo caso è la classe dei nei benigni.
- False Negative (FN): il numero di predizioni sbagliate per i campioni appartenenti alla classe negativa.
- True Positive (TP): il numero di predizioni corrette per i campioni appartenenti alla classe positiva, che in questo caso è la classe dei nei maligni.

- False Positive (FP): il numero di predizioni sbagliate per i campioni appartenenti alla classe positiva.

Grazie a questi quattro valori si possono calcolare diverse metriche di valutazione, tra cui la accuracy, la precision, la recall, la specificity e l'F1-score, le quali verranno approfondite nel prossimo paragrafo. Definiti i quattro valori caratteristici di una confusion matrix, quest'ultima avrà la forma seguente:

		Predicted	
		Negative (**N**) **-**	Positive (**P**) **+**
Actual	Negative **-**	True Negatives (**TN**)	False Positives (**FP**) **Type I error**
	Positive **+**	False Negatives (**FN**) **Type II error**	True Positives (**TP**)

Per visualizzare la confusion matrix, ho implementato una funzione che da uso delle librerie Scikit-Learn e MatplotLib, le quali permettono di stampare a schermo la confusion matrix e personalizzarla a proprio piacimento.
Il risultato ottenuto sui dati di addestramento è molto buono poiché, come si può vedere in Figura 5.10, il modello riesce a classificare correttamente la maggior parte dei campioni di entrambe le classi.

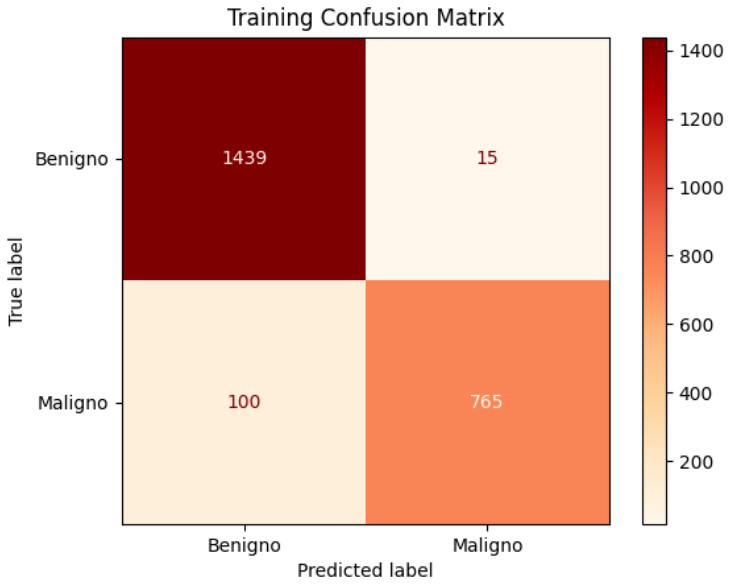


Figura 5.10. Confusion Matrix sui dati di training.

5.6 Analisi dei risultati

Una volta terminata la fase di addestramento, i pesi del modello sono ottimizzati e posso quindi valutare le sue performance predittive sul test set.
Come primo strumento di valutazione ho scelto di stampare la confusion matrix. È chiaro che la percentuale di predizioni corrette effettuate sui dati del test set sarà minore rispetto a quelle effettuate sul training set poiché, al contrario di quelli usati per l'addestramento, questi campioni non sono mai stati esaminati dalla rete.
Dalla confusion matrix in Figura 5.11 si evince che il modello riesce a classificare molto bene le immagini di lesioni benigne e leggermente peggio quelle di lesioni maligne. Questo effetto è ovviamente dovuto dallo sbilanciamento del dataset.

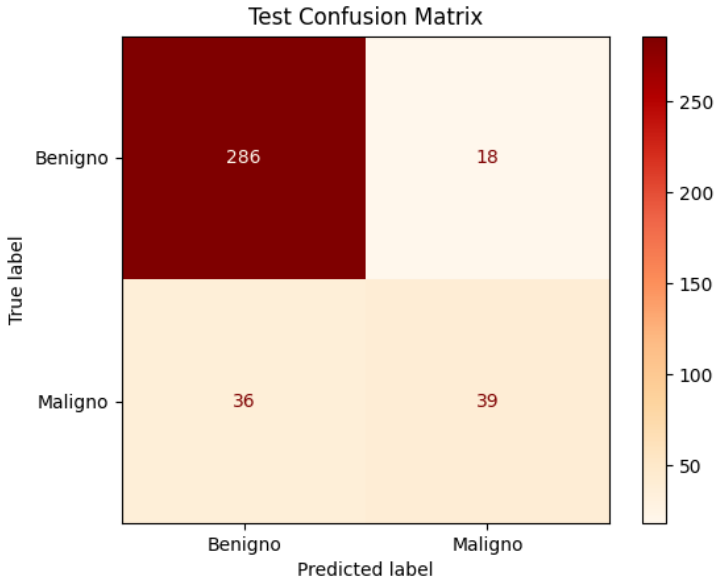


Figura 5.11. Confusion Matrix sul test set.

Successivamente, per valutare più accuratamente il mio modello, ho usato le stesse metriche che furono usate nella ISIC Challenge del 2016 per stabilire la classifica finale:

- Accuracy: come già detto in precedenza, è il numero di predizioni corrette fratto il numero di predizioni totali. La sua formula a partire dai valori della confusion matrix è dunque (TP+TN)/(TP+TN+FP+FN).
- Sensitivity (o Recall o True Positive Rate): è una metrica che misura la capacità del modello di identificare correttamente i campioni positivi. Si calcola come TP/(TP+FN).
- Specificity (o True Negative Rate): misura la capacità del modello di identificare correttamente i campioni negativi e si calcola come TN/(TN+FP).
- PPV (Positive Predictive Value o Precision): quantifica il numero di campioni positivi classificati correttamente dal modello rispetto al numero di tutti i campioni classificati come positivi. La sua formula è TP/(TP+FP).

- NPV (Negative Predictive Value): quantifica il numero di campioni negativi classificati correttamente dal modello rispetto al numero di tutti i campioni classificati come negativi e si calcola come TN/(TN+FN).
- Balanced Multiclass Accuracy: questa metrica risulta molto utile in presenza di dataset sbilanciati e corrisponde alla media aritmetica tra sensitivity e specificity.
- Dice Coefficient (o F1-score): anche questa metrica viene utilizzata in presenza di dataset sbilanciati. Si calcola come 2*PPV*sensitivity/(PPV+sensitivity)
- AUC (Area Under the Curve): corrisponde alla misura dell'area sotto la ROC Curve. Un valore di AUC uguale a 0.5 corrisponde ad un modello completamente casuale, mentre se la AUC è pari a 1, vuol dire che il classificatore è perfetto.
- AUC, Sens>80%: è l'area al di sotto della ROC Curve, calcolata solo nelle zone in cui quest'ultima ha una sensitivity che supera l'80%.
- Average Precision: corrisponde all'area sotto la Precision-Recall Curve. Questa è la metrica principale con la quale vengono valutati i partecipanti della competizione e viene decretata la classifica finale.

Nella seguente tabella vengono riportati i risultati ottenuti dalla mia rete neurale convoluzionale usando le metriche di valutazione elencate in precedenza.

Metrica	Valore
Balanced Multiclass Accuracy	0.730
AUC	0.828
AUC, Sens>80%	0.586
Average Precision	**0.623**
Accuracy	0.856
Sensitivity	0.520
Specificity	0.941
Dice Coefficient	0.591
PPV	0.684
NPV	0.888

Per avere un confronto con le performance di altri modelli, di seguito vengono invece riportati i risultati dei migliori quattro partecipanti nella classifica della ISIC Challenge del 2016, stipulata seguendo la metrica dell'Average Precision.
La classifica completa si può facilmente trovare nella sezione "Leaderboard" del sito ufficiale della competizione (Bibliografia [11]).

Metrica	1° Posto	2° Posto	3° Posto	4° Posto
Balanced Multiclass Accuracy	0.724	0.723	0.641	0.742
AUC	0.804	0.802	0.826	0.796
AUC, Sens>80%	0.568	0.542	0.647	0.500
Average Precision	**0.640**	**0.622**	**0.601**	**0.567**
Accuracy	0.855	0.813	0.834	0.786
Sensitivity	0.507	0.573	0.320	0.667
Specificity	0.941	0.872	0.961	0.816
Dice Coefficient	0.580	0.548	0.432	0.552
PPV	0.679	0.524	0.667	0.472
NPV	0.885	0.892	0.851	0.908

Le due curve citate in precedenza (ROC Curve e Precision-Recall Curve) sono degli strumenti fondamentali per la valutazione delle capacità di un classificatore binario e per questo motivo ho deciso di esaminarle più nel dettaglio graficandole entrambe, ancora con l'uso delle librerie Scikit-Learn e MatplotLib. La ROC (Receiver Operating Characteristic) Curve rappresenta la sensibility (True Positive Rate) al variare della specificity (False Positive Rate) per diversi valori di threshold[3]. Come si può verificare in Figura 5.12, il mio classificatore, pur essendo comunque distante da un classificatore perfetto, produce risultati ampiamente al di sopra di un classificatore random.

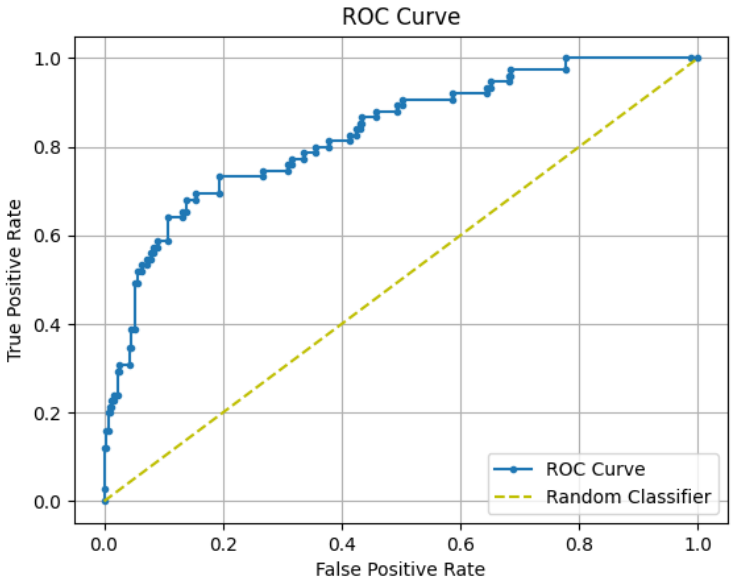


Figura 5.12. Test ROC Curve.

[3]In un modello di classificazione binaria è il valore di separazione tra le due classi. In altre parole, se un campione ha un valore di classificazione minore del valore di threshold, verrà assegnato dal modello alla classe negativa, altrimenti verrà assegnato alla classe positiva.

La Precision-Recall Curve, diversamente dalla ROC Curve, viene usata soprattutto quando le classi sono sbilanciate (come in questo caso), e rappresenta la Precision (PPV) in funzione della Recall (Sensitivity) per diversi valori di threshold. La Precision-Recall Curve ottenuta testando il mio modello è la seguente:

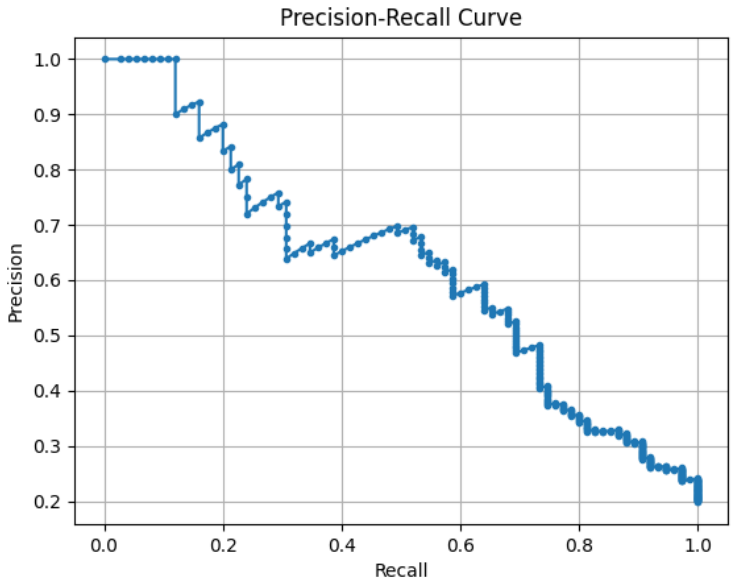


Figura 5.13. Test Precision-Recall Curve.

Con questo grafico si conclude la valutazione di questa rete neurale, la quale con dataset più grandi e una potenza di calcolo maggiore potrebbe migliorare ancora e fornire delle predizioni più accurate.

Capitolo 6

Conclusioni

La tesi che ho realizzato mi ha coinvolto particolarmente poiché mi ha fatto realmente capire quanto l'intelligenza artificiale possa effettivamente applicarsi in qualsiasi campo, avendo potenzialità tali da semplificare e migliorare la vita degli esseri umani.

Questo progetto infatti si poneva come obiettivo quello di automatizzare la fase di riconoscimento del melanoma cutaneo, una malattia della pelle potenzialmente letale. Sfruttando l'architettura di una rete neurale convoluzionale profonda, l'algoritmo è stato addestrato su un vasto insieme di immagini dermatoscopiche, già classificate e divise in lesioni benigne e maligne.

Sebbene il modello abbia ottenuto risultati molto promettenti, classificando quasi perfettamente le lesioni benigne, la sua capacità di classificazione per le lesioni maligne è risultata meno accurata. Ciò è dovuto da varie complicazioni di natura puramente tecnica che ho riscontrato durante questo progetto. Va infatti sottolineato che le performance della mia rete potrebbero essere ulteriormente migliorate tramite l'utilizzo di un dataset più ampio e bilanciato, una risoluzione più alta delle immagini e soprattutto una potenza di calcolo maggiore.
Queste limitazioni hanno influenzato le prestazioni del modello, le quali restano comunque soddisfacenti e rappresentano un ottimo punto di partenza per studi più approfonditi nel campo della diagnostica dermatologica.

Alla luce di questi risultati, si può concludere che l'utilizzo del machine learning e in generale dell'intelligenza artificiale in ambito sanitario rappresenta un'importante opportunità per velocizzare il processo di studio delle malattie, riducendo la mole di lavoro del personale medico ed aprendo la strada a nuove prospettive di ricerca e sviluppo.

Bibliografia

[1] Szeliski, R. (2019). *"Computer Vision: Algorithms and Applications"* (2nd ed.), Springer

[2] LeCun, Y., Bengio, Y., & Hinton, G. (2015). *"Deep learning"*, Nature 521:436–444.

[3] P. P. Shinde & S. Shah, (2018). *"A Review of Machine Learning and Deep Learning Applications"*, 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), p. 1-6.

[4] Ioffe, S., & Szegedy, C. (2015). *"Batch normalization: Accelerating deep network training by reducing internal covariate shift"*.

[5] He, K., Zhang, X., Ren, S. & Sun, J. (2016). *"Deep Residual Learning for Image Recognition"*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-778.

[6] Géron, A. (2017). *"Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems"*, O'Reilly.

[7] A. L. Samuel, (1959). *"Some Studies in Machine Learning Using the Game of Checkers"*, IBM Journal of Research and Development, vol. 3, no. 3, pp. 210-229, doi: 10.1147/rd.33.0210.

[8] Chollet, F. (2017). *"Deep Learning with Python"*.

[9] Associazione Italiana per la Ricerca sul Cancro, www.airc.it/cancro/informazioni-tumori/guida-ai-tumori/melanoma-cutaneo

[10] American Cancer Society, www.cancer.org/cancer/melanoma-skin-cancer.html

[11] Sito ufficiale della Isic Challenge, https://challenge.isic-archive.com/

[12] Dataset utilizzato (vedi task 3), https://challenge.isic-archive.com/data/#2016

[13] Google Colab, https://colab.research.google.com/notebooks/intro.ipynb

[14] Introduzione alle CNN, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[15] Sito web della libreria Keras, https://keras.io/

[16] Sito web della libreria Scikit-Learn, https://scikit-learn.org/stable/

[17] Algoritmo di ottimizzazione Adam, https://optimization.cbe.cornell.edu/index.php?title=Adam

[18] https://neptune.ai/blog/balanced-accuracy

Ringraziamenti

Io, come sapete, non sono una persona che tende ad esternare facilmente ciò che prova, ma in questo caso cercherò di fare una breve eccezione.
Con questa tesi si chiude il primo importante percorso della mia vita, che sono riuscito a raggiungere grazie alla presenza costante di alcune persone.

Le prime persone che vorrei ringraziare sono la mia famiglia: i miei fratelli, con i quali ho un rapporto stupendo e spero che sarà così per sempre, ma soprattutto i miei genitori. Voi mi avete permesso di vivere sempre una vita felice e serena, mandandomi in giro per il mondo, mettendo i miei interessi davanti ai vostri e non facendomi mai mancare niente. Vi sforzate tutti i giorni lavorando giorno e notte ed è soprattutto grazie al vostro esempio (e qualche rottura di palle) che riesco a raggiungere gli obiettivi della mia vita. Vi stimo tanto e vi vorrò sempre bene.

Ringrazio poi i miei amici del liceo, con i quali ho condiviso tanti bei momenti e che riesco a vedere sempre anche col passare degli anni.

Un grazie speciale va a Martina, la ragazza che più di tutti è riuscita a farmi aprire, la prima ad esserci sempre quando ho bisogno e la prima a festeggiare orgogliosamente tutti i miei traguardi negli ultimi due anni.

Un ringraziamento va anche ai miei compagni di vela, con i quali continuo da anni a vivere delle esperienze indimenticabili, girando il mondo intero e scoprendo sempre nuovi posti e persone.

Infine un'ultimo grazie va alle persone che mi hanno seguito e aiutato più da vicino in questo percorso. I miei amici di università, tra i pochi in tutto il corso in grado di trovare un equilibrio tra lo studio e la vita, quelli che mi hanno aiutato nella pratica ma soprattutto nel vivere questi tre anni (e mezzo) di studio con un po' di leggerezza.

A tutti voi, grazie.