POLITECNICO DI TORINO

Ingegneria Biomedica Strumentazione Biomedicale

Tesi di Laurea Magistrale

Classificazione automatica delle lesioni cutanee

Relatrice	Candidato
Prof. Kristen MEIBURGER	**Matteo DILIBERTO**

Azienda

TESI S.R.L

07 2020

Sommario

Il presente lavoro si incentra sull'analisi e classificazione delle lesioni cutanee attraverso l'uso della rete neurale Inception V3, sulla quale si è intervenuto con opportuni adattamenti.

Tra le diverse tipologie di lesioni cutanee, sono state individuate sette classi, valutate le più rilevanti a livello di prognosi medica.

Partendo da altri due dataset preesistenti (HAM 10000 e ISIC), la prima fase del lavoro è stata incentrata sulla creazione di un dataset bilanciato, con l'obbiettivo specifico di ottenere prestazioni elevate e un alto tasso di apprendimento delle features specifiche di ogni classe. Non è stato però possibile raggiungere tale obbiettivo per mancanza di un sufficiente numero di immagini per classe. Nonostante l'elevato numero di istanze presenti nei due dataset, le classi meno popolose sono rimaste tali, pur aumentando in modo significativo la loro popolazione.

Pertanto, successivamente si è implementato un processo di Data Augmentation al fine di aumentare il numero di esempi da fornire al modello.

Il modello è stato addestrato con batch da 32 immagini dermatoscopiche ciascuno, per un totale di 5469 immagini di training. È stato validato usando batch da 32 immagini per un totale di 1367 immagini, mentre il test set è composto da 1710 immagini.

I risultati della fase di test sono risultati ottimali per quasi tutte le classi tranne due; dermatofibromi e cheratosi attiniche. Escluse le due classi sopracitate, si ha un'accuracy media per classe dell'88,6%. La stessa per le classi di nevo e melanoma è rispettivamente del 98,86% e 85,01%, ottenendo un F1-score del 96% e dell'89%. Si è ritrovato che la causa principale del calo di prestazioni derivasse dai pochi esempi presenti nelle classi di cheratosi attinica e dermatofibromi.

Infine come ultimo processo di verifica, è stata svolta un'ulteriore classificazione dei risultati ottenuti nelle macro categorie di lesioni maligne e lesioni benigne. Tale risultato finale presenta una misclassificazione dell'8,89% e un F1-score globale dell'88,12%.

Parole chiave: Machine Learning, InceptionV3, Lesioni cutanee, Melanoma, Classificazione, Deep Learning, Classificazione Multiclasse.

Ringraziamenti

Grazie.

Indice

Indice delle tabelle

Indice delle figure

Capitolo 1

Introduzione generale

1.1 La cute

Il tessuto cutaneo è l'organo principale dell'apparato tegumentario che ricopre tutto il corpo umano. Presenta funzioni sia protettive che metaboliche, infatti la cute funge da prima barriera contro eventuali patogeni esterni, protegge gli organi e i tessuti sottostanti da abrasione diretta e, soprattutto, funge da protettore contro i dannosi raggi UV.

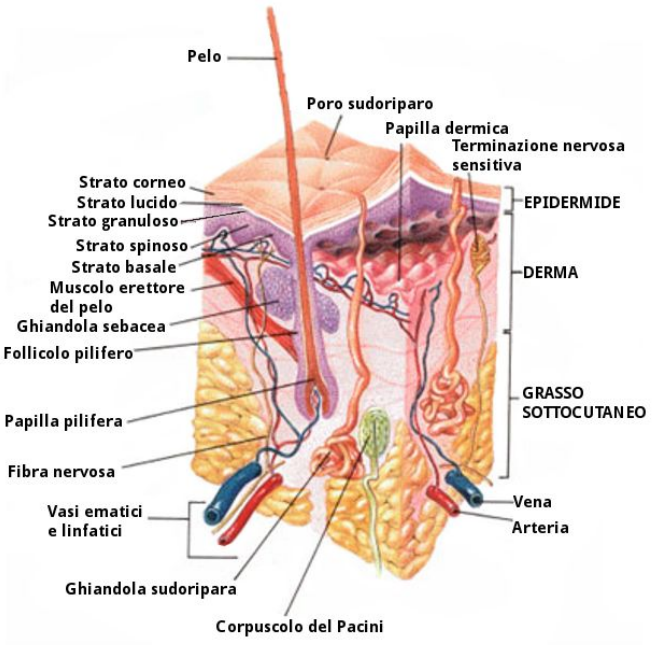


Figura 1.1: Struttura del tessuto cutaneo

Un'altra funzione molto importante e non meno essenziale, è quella termoregolatrice: grazie alle svariate ghiandole sudoripare coaudiuva il meccanismo di espulsione del calore, mentre lo strato di cellule adipose funge da barriera isolante contro una dispersione esagerata del calore. Questi processi di controllo termico e della microcircolazione nel derma sono controllati da parte del sistema nervoso simpatico.[1] La pelle è un tessuto che si compone *(Fig1.1)* principalmente di tre strati, di cui, i primi due sono quelli che identificano la cute:

- **Epidermide**
- **Derma**
- **Ipoderma**

1.1.1 Funzione e costituzione dei vari strati

Epidermide: Lo strato dell'epidermide è formato da tessuto epiteliale,[1] presenta uno spessore variabile da individuo ad individuo da 27 a 300 µm e si compone di cinque layer che sono:

- Strato corneo
- Strato lucido
- Strato granuloso
- Strato spinoso
- Strato basale

L'individuazione di questi cinque strati è possibile distinguendo le varie fasi di maturazione dei cheranociti[2]. Infatti essi partono dallo strato più profondo, quello basale, e migrano verso lo strato più esterno, il corneo, lungo un percorso di maturazione, risultando quindi sovrapposti uno sopra l'altro, formando dei substrati. Lo strato corneo più esterno si compone essenzialmente di cellule morte che prendono il nome di corneociti. Essi vengono inglobati in una particolare matrice lipidica.

> " Questa parte dello strato corneo ha un influenza di circa l'8%, che in termini quantitativi è trascurabile.[1] ".

[1] Ciò costituisce una differenza sostanziale a livello clinico e istologico con il derma che è formato da tessuto connettivo

[2] Tipo cellulare più abbondante nell'epidermide

Infatti la quasi totalità della luce che penetra all'interno di questo strato viene poi assorbita dalle cellule melaniniche epidermiche. I melanociti sono localizzati nella parte bassa del layer basale e producono un pigmento di colorazione scura chiamato melanina. Questo pigmento aiuta nella difesa dai raggi UV.
Le cellule basali si trovano alla fine dello strato epidermico. In questa porzione di cute si ha una forte proliferazione cellulare, ciononostante la popolazione cellulare viene tenuta costante da un continuo movimento in ingresso e in uscita di queste cellule.

Derma: Lo strato del derma è molto più spesso di quello epidermico, infatti ha uno spessore variabile da 0.6 a 3 mm e i primi 350 µm contengono vasi sanguigni che servono per trasportare i nutrimenti allo strato epidermico. Il derma è formato principalmente da tessuto connettivo, che presenta vasi sanguigni al suo interno, ghiandole sebacee, collagene, follicoli piliferi, ghiandole sudoripare, terminazioni nervose e cellule lipidiche.
All'interno del derma si individuano due tipologie di strati diversi:

- uno strato detto papillare
- un secondo strato detto reticolare

Le cellule sebacee sono piccole ghiandole che producono sebo, un idratante naturale. I tessuti del derma sono composti principalmente da fibre di collagene.

1.2 Le lesioni cutanee

Le lesioni cutanee sono delle anomalie circoscritte a livello della cute. Un esempio sono i nevi che sono delle lesioni presenti nella maggior parte della popolazione, dovute a un processo malformativo della cute, più propriamente detti amartomi[3]. Possono essere presenti alla nascita o svilupparsi dopo, sino a circa 40 anni. Di solito appaiono come macchie di forma e dimensioni variabili, piane o in rilievo, pigmentate o no.
Le tecniche dermoscopiche attuali presentano una sensibilità dell'83 % e una specificità del 69 % nel riconoscimento del melanoma. La dermatoscopia è un esame diagnostico non invasivo che consente di osservare strutture morfologiche localizzate nell'epidermide, nella giunzione dermo-epidermica e nel derma superficiale. Si utilizza nella valutazione clinica delle lesioni pigmentate, in particolare quelle melanocitarie, e rappresenta un valido strumento nella valutazione clinica di lesioni

[3] Alteerazioni circoscritte della cute.

sospette, ed è lo step che precede la decisione di un eventuale biopsia o escissione chirurgica.
Le percentuali di questo esame sono alte, ma non altissime, e servono per un' analisi e un'azione preventiva, che aumenterebbe drasticamente il tasso di sopravvivenza del paziente affetto da una patologia come il melanoma.
È essenziale la detezione preventiva del melanoma in situ (MIS) e delle altre lesioni, per questo lo scopo della tesi è lo sviluppo di un software che classifichi le foto in una tra 7 classi:

- nevo
- melanoma
- cheratosi attiniche
- cheratosi seborretiche
- basaliomi
- dermatofibromi
- lesioni vascolari

1.2.1 Nevi

I nevi sono macule, papule o noduli pigmentati acquisiti, circoscritti e di piccole dimensioni, costituiti da gruppi di cellule neviche melanocitiche situate nell'epidermide, nel derma e, più raramente, nel tessuto sottocutaneo.

Figura 1.2: Esempio Neo

I nevi nevocellulari melanocitici[2] possono essere classificati sulla base della localizzazione dei cluster delle cellule neviche in:

- Nevi melanocitici giunzionali, le cui cellule neviche sono situate lungo la porzione epidermica della membrana basale. Vengono definiti intra-epidermici.
- Nevi melanocitici composti. Queste cellule neviche vengono localizzate sia nella porzione intra-epidermica sia in quella dermica.

- Nevi melanocitici dermici, le loro cellule si trovano a livello dermico.

Le cellule neviche presentano una maggiore capacità di produrre melanina quando si trovano nel livello intraepidermico (nella giunzione), più le cellule neviche sprofondano meno pigmento riescono a produrre. Quindi l'aspetto clinico derivante è il seguente:

- I nevi giunzionali sono piani e scuri;
- I nevi composti sono rilevati sul piano cutaneo e scuri;
- I nevi dermici sono rilevati sul piano cutaneo e chiari.

È un fenomeno comune che i nevi giunzionali evolvano prima in composti e poi in dermici, ciò spiega perchè i nevi giunzionali e composti si riscontrano frequentemente durante tutte le fasi pre adoscelenziali e quella adolescenziale. I nevi dermici invece iniziano a comparire tra i trenta e quaranta anni.

1.2.2 Le altre classi di lesioni cutanee

I tumori della cute rappresentano in media il 10-15% di tutti i tumori maligni. Questo gruppo eterogeneo ma comune di neoplasie primitive è costituito per il 90-95% circa dagli epiteliomi, per il 5% dai melanomi e nell'1% circa dei casi dai tumori cutanei più rari. Gli epiteliomi, che originano dai cheratinociti dell'epidermide, sono rappresentati rispettivamente nel 75% e nel 15% dai due istotipi basocellulare e squamocellulare, con un rapporto tra maschi e femmine di 2-3/1.[3]

Melanoma [4] Il melanoma è il più temuto dei tumori cutanei per aggressività. Si tratta di un tumore maligno che prende di mira i melanociti, soprattutto quelli localizzati sulla pelle. Tra le caratteristiche principali di questo tumore ci sono l'elevato grado di metastatizzazione e la scarsa risposta ai trattamenti. Ciononostante, se asportato in tempi brevi la sopravvivenza è molto alta. Il melanoma è leggermente più frequente nelle donne ad eccezione per il melanoma nodulare che è più frequente negli uomini. La prognosi è peggiore per gli uomini. Alcuni fattori di rischio per il melanoma sono: radiazioni ultraviolette, avere più di 50 100 nei comuni, avere nevi atipici o nevi congeniti (soprattutto quelli giganti) e marcatori genetici.
Il melanoma presenta due fasi di crescita, una radiale limitata all'epidermide o focalmente nel derma papillare e un'altra verticale e profonda, nel corso della quale la neoplasia diventa in grado di superare la membrana basale e dunque metastatizzare. Il melanoma nodulare è una variante di melanoma caratterizzata dalla precocità della fase di crescita verticale: la lesione si sviluppa rapidamente, soprattutto in profondità ed è in grado di metastatizzare in breve tempo, peggiorando la prognosi.

Figura 1.3: Esempio melanoma

Il melanoma cutaneo primario si classifica in quattro gruppi istologici principali. Talvolta, un melanoma può essere amelanotico, ovvero non presentare pigmentazione: la lesione può essere facilmente confusa con una condizione infiammatoria, con conseguente ritardo nella diagnosi e peggioramento della prognosi. Il melanoma desmoplastico è una rara variante istopatologica (<5%) caratterizzata da melanociti di aspetto fusiforme dispersi in una densa matrice di collagene. Il melanoma può metastatizzare per via linfatica o ematica. La disseminazione linfatica può produrre satellitosi (lesioni dermiche o sottocutanee attorno alla cicatrice del melanoma primario), lesioni di transito (nella direzione del drenaggio linfatico della zona cutanea dove si trovava il melanoma) e adenopatie nei linfonodi regionali. La disseminazione ematica, associata a prognosi infausta, può produrre metastasi non viscerali così come viscerali.
Negli ultimi decenni l'incidenza dei melanomi [5] maligni ha presentato la tendenza ad aumentare. In Italia, l'incidenza del melanoma maligno è da anni in costante ascesa, e gran parte di questo fenomeno (comune a tutti i Paesi sviluppati) è attribuibile:

- All'effettivo aumento del ruolo eziopatogenetico da parte delle radiazioni UV.
- Uno stato di immunodeficienza.
- Caratteristiche fenotipiche (colorazione della pelle, colorazione capelli, presenza di lentigini, alto numero di nevi e nevi atipici).
- Nel 5-10 % dei pazienti è identificabile una familiarità per melanoma. Si ritiene che nella storia familiare possano essere implicate mutazioni ereditarie di un gene oncosoppressore (CDKN2A) e alterazioni genetiche che determinano una minor attività dei fattori oncosoppressori.

Cheratosi seborroica È il tumore epiteliale benigno più frequente (*Fig. 1.4*) ed è caratteristico dell'età avanzata e della razza bianca. Clinicamente si presenta come una macula iperpigmentata di colorito marrone-neastro. La superficie ha un tipico aspetto verrucoso(da qui il termine alternativo 'verruca seborroica'), ed al tatto ha una caratteristica consistenza 'untuosa'. Reperto caratteristico è l'evidenza degli sbocchi follicolari repleti di cheratina. [6]

In generale è asintomatica, ma talvolta può essere pruriginosa. Può comparire in qualsiasi area fatta eccezione per mucose, regione palmare e plantare. Le sedi più frequenti sono il viso, il tronco e il dorso delle mani. Non è necessario nessun trattamento. Si può ricorrere ad asportazione chirurgica in caso di dubbi diagnostici con processi maligni. La crioterapia è il trattamento d'elezione nei casi sintomatici o che provocano problemi estetici.

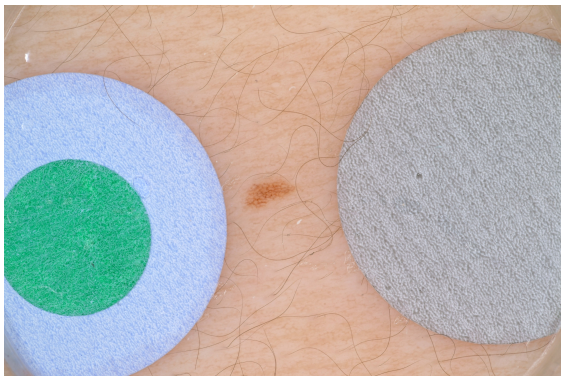


Figura 1.4: Esempio Cheratosi seborroica

Cheratosi attinica La cheratosi attinica è una proliferazione cheratinocitaria caratterizzata da una displasia che non arriva ad occupare tutto lo spessore dell'epidermide ed è un sensibile indicatore di intensa esposizione al sole. Essendo lesioni caratterizzanti dell'esposizione solare di solito sono locate sui siti esposti cronicamente come la testa e il collo, il dorso delle mani e degli avambracci[3].
Da un punto di vista clinico, risulta spesso impossibile distinguere una cheratosi attinica da un carcinoma squamoso invasivo iniziale. Questa proliferazione cheratinocitaria viene considerata una lesione precancerosa con percentuali evolutive che vanno dallo 0,1 al 10%, presenta inoltre la possibilità che si evolva verso il basalioma o un carcinoma squamoso invasivo (SCC) del 5 e del 20% rispettivamente. L'ispessimento e l'indurimento laterale alla palpazione rappresentano i segni della trasformazione verso un SCC. I carcinomi squamosi derivanti da cheratosi attiniche presentano una prognosi migliore rispetto a quelli di origine diversa, e hanno una probabilità di produrre metastasi dello 0,3-5%.
Si manifesta sotto forma di lesione desquamativa asintomatica con alone eritematoso e le varianti istologiche più frequenti sono: la forma atrofica in cui lo strato spinoso si presenta assottigliato, e quella ipertrofica con strato spinoso espanso e ipercheratosi; questa è la più utile caratteristica distintiva di questa malattia.
La sintomatologia tipica di queste lesioni sono prurito e bruciore pungente, che possono essere avvertiti con l'esposizione al sole o con la sudorazione.

Nella cheratosi attinica pigmentata la diagnosi differenziale deve essere effettuata con le lentiggini solari e la lentigo maligna. Le cheratosi generalmente sono più piccole e meno definite alla periferia che la malattia di Bowen e la cheratosi seborroica.

Basalioma Il carcinoma basocellulare è un tumore epiteliale maligno derivante dalla trasformazione delle cellule basali. Non è una neoplasia maligna particolarmente aggressiva, presenta una crescita abbastanza lenta e difficilmente crea metastasi a distanza. Tuttavia, se non vengono adottate delle misure di cura e prevenzione, si può estendere superficialmente e può arrivare dalla cute fino all'osso, creando gravi danni a tutte le strutture e tessuti circostanti.

> *L'American Cancer Society riporta che i carcinomi della pelle sono i più comuni tumori negli Stati Uniti. Il carcinoma basocellulare, in particolare, è il tumore maligno più comune negli Stati Uniti. Si tratta del carcinoma della pelle più frequente nei soggetti di razza bianca, è invece rarissimo nella razza nera e poco comune nei soggetti di razza gialla, asiatica.[2]*

Nonostante come tutte le lesioni della pelle trattate in questa tesi, la sua incidenza sia in aumento, il tasso di mortalità derivato da questo carcinoma è molto basso e comunque in calo. Presenta una maggior incidenza negli individui di sesso maschile che rientrano nella fascia d'età sopra i 50 anni.

> *Il rischio stimato di sviluppare un carcinoma basocellulare durante la vita nei soggetti di razza bianca è del 33-39% nei maschi e del 23-28% nelle femmine.[2]*

La aree di maggior sviluppo di questa lesione sono collo e viso, ma non è insolito che si sviluppino anche nel tronco. La neoplasia può essere solitaria, ma spesso è multipla.
Il basalioma presenta diverse forme cliniche nelle quali si presenta:

- Il basalioma nodulare
- Il basalioma superficiale
- Il basalioma morfeiforme
- Il basalioma infiltrativo
- Il basalioma fibroepiteliale

Dermatofibromi Un dermatofibroma è un tumore benigno della pelle, frutto di una proliferazione anomala dei fibroblasti dei tessuti connettivi fibrosi del derma. L'insorgenza di un dermatofibroma comporta la formazione, in genere sulle gambe e o sulle braccia, di uno più noduli di consistenza dura, rosso-marroni e dalle dimensioni comprese tra 0,5 e 1,5 cm. Un tumore benigno come il dermatofibroma non può tramutare in una neoplasia maligna. Il suo trattamento chirurgico è previsto solamente quando i noduli sono sintomatici. A livello epidemologico sono leggermente più interessati i soggetti adulti della popolazione femminile.(*Fig. 1.5*)

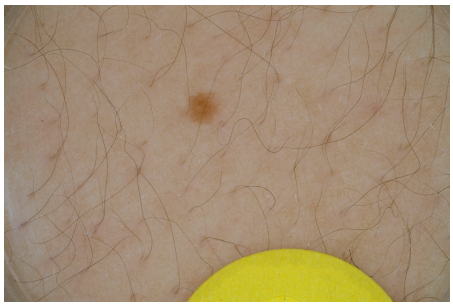


Figura 1.5: Esempio Dermatofibroma

Lesioni vascolari Sono un tipo di lesione che comprende lesioni acquisite (es granuloma piogenico) e quelle che sono presenti o insorgono subito dopo la nascita (voglie, esempio in *1.6*). Le voglie comprendono tumori vascolari (es emangiomi infantili) e malformazioni vascolari. Le malformazioni vascolari sono difetti congeniti della morfogenesi vascolare localizzati, permanenti e comprendono malformazioni capillari (es nevo flammeo), venose e arterovenose (aneurisma ciroide) e linfatiche.

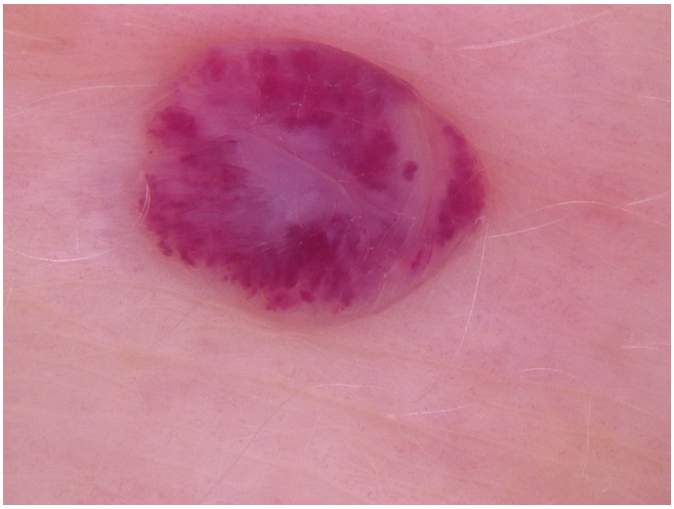


Figura 1.6: Esempio Lesione Vascolare

1.3 Le immagini per un computer

Per poter capire al meglio tutto ciò che verrà dopo, è necessario specificare cosa è un'immagine dal punto di vista di un computer, e come viene rappresentata.

(a) *Immagine per un umano* **(b)** *Immagine per un computer*

Figura 1.7

Quella rappresentata in *Fig. 1.7a* è la rappresentazione di un'immagine per un umano, che è totalmente diversa da come la vede un computer.
Un computer non ha idea di cosa sia un'immagine, esso vede unicamente la matrice mostrata in *Fig. 1.7b*. A seconda che l'immagine sia a colori o in bianco e nero allora la matrice che vede è tridimensionale piuttosto che singola, però il fatto che veda una matrice non cambia.
Ogni elemento della matrice rappresenta un pixel. Un pixel è il più piccolo elemento di rappresentazione grafica di un'immagine, e il valore all'interno si riferisce al livello di intensità di colore. Normalmente i livelli di colore vengono codificati in 8 bit, quindi in maniera discreta, vengono creati 255 livelli di intensità di colore, e il loro valore varia da zero a 255.
A seconda del tipo di codifica dell'immagine le differenze sono:

- Nel caso di codifica RGB ogni pixel di ogni layer (R, G, B) avrà la propria intensità di colore e, sovrapponendoli, si ottiene l'immagine in *Fig. 1.7a*.
- Nel caso di codifica in bianco e nero si ha una sola matrice contente il livello di sfumatura di bianco o nero per quel determinato pixel.

Per la codifica in bianco e nero lo zero rappresenta il nero, il valore 255 il bianco, e nel mezzo ci sono 253 livelli di grigio che variando intensità permettono di rappresentare tutti i toni da un estremo all'altro. Per l'RGB il discorso è lo stesso, solo che anzichè variare in toni di grigio vareirà in toni del colore del layer. Una volta sovrapposti tutti e tre i layer si ottiene l'immagine a colori.
Oltre questi due spazi di rappresentazione ne esistono tanti altri, però il concetto fondamentale da ricordare è la loro rappresentazione per un computer.

1.4 Elaborazione e classificazione manuale

Alcune tecniche vengono definite manuali in quanto è l'operatore che, grazie alla sua esperienza, decide quale sequenza di elaborazione dell'immagine applicare, attraverso quali processi e come impostare i parametri dei vari operatori che andrà ad usare.
Le tecniche manuali possono essere divise in due grandi categorie, quelle di pre-processing e quelle di post-processing. Le prime prevedono di applicare degli operatori che vanno a modificare l'immagine su cui si sta lavorando, le seconde invece, servono per trovare delle relazioni all'interno dell'immagine.

1.4.1 Preprocessing

Il preprocessing è la prima fase di elaborazione di un'immagine. Solitamente prevede vari step; rimozione del rumore, operazioni di chiusura e apertura, aumento o diminuzione del livello di contrasto e saturazione, etc.
Le tecniche che rappresentano lo stato dell'arte sono quelle classiche. Le modifiche che vengono effettuate tramite le tecniche di preprocessing servono per poter migliorare la qualità dell'immagine. Il termine qualità, in ambito di immagini mediche, assume un significato ben diverso da quello che ha per un'immagine in generale. Un'immagine medica è di qualità quando si riescono a distinguere in modo chiaro gli elementi di indagine, a volte ciò si traduce in un'immagine *'brutta'* a livello visivo, ma, funzionale per la diagnosi da parte del medico.
Analizzando più nello specifico queste tecniche, troviamo:

Operatori puntuali

Gli operatori puntuali sono gli strumenti che applicano una trasformazione a un pixel per cambiarne il suo valore, senza che esso subisca dei cambi di posizione o senza che queste trasformazioni siano influenzate dai valori dei pixel circostanti. Una tipica operazione puntuale è quella del tresholding, ad esempio: attraverso una funzione

$$\text{f(x,y)}=\begin{cases}0, & \text{se } f(x,y) \leq 10\\ f(x,y), & \text{se } f(x,y) > 10\end{cases}$$

questo operatore va ad analizzare il valore di ogni pixel in ogni posizione e, se minore di un valore preimpostato, lo imposta a 0. Esistono poi tanti altri operatori puntuali che, comunque, all'atto pratico hanno per lo più un effetto di modifica dei livelli di intensità dei grigi o di luminanza.

Operatori locali

Sono filtri che sfruttano, per determinare il valore del pixel di destinazione, non solo il valore del pixel stesso nell'immagine originale, ma anche il valore di tutti i pixel presenti all'interno di un determinato "vicinato". Il vicinato è l'insieme di pixel che racchiude il pixel di interesse. La dimensione di tale vicinanza è determinata

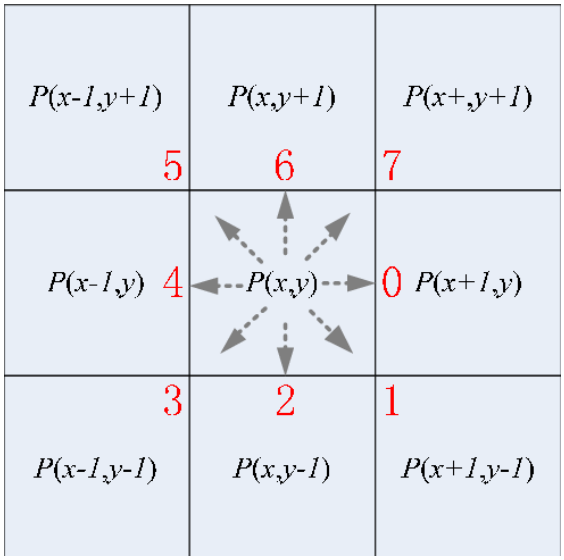


Figura 1.8: Vicinato di un pixel

dall'utente che imposta la "finestra" su cui l'operatore deve lavorare.
Nella *Fig. 1.8*, il vicinato viene definito otto connesso: è rappresentato dagli 8 pixel adiacenti al nostro pixel. È un vicinato che prende tutti i pixel distanti al massimo un pixel dal pixel di interesse. Le dimensioni del vicinato sono a discrezione del programmatore.
A seconda delle operazioni che si decide di fare all'interno di questo vicinato, per decidere il valore del pixel centrale, si implementa un tipo operatore locale piuttosto che un altro.
A livello di elaborazione di immagini mediche gli operatori che vengono usati maggiormente durante la fase di preprocessing sono:

- **Operatori di apertura e chiusura:** Gli operatori di apertura e chiusura sono due operatori morfologici: cioè vanno a modificare la morfologia degli elementi presenti all'interno dell'immagine. L'apertura e la chiusura hanno effetti opposti, ma utilizzano le stesse operazioni di filtraggio morfologico, dilatazione e erosione, ma con una diversa sequenza di esecuzione di questi.

  La dilatazione è il processo attraverso il quale si 'riempiono' le forme, e ha un effetto filtrante passa basso.

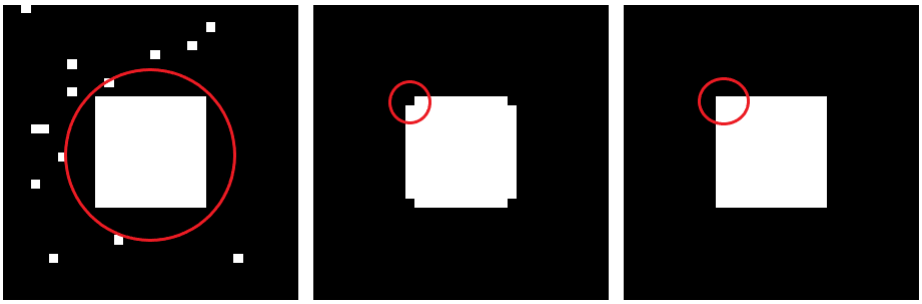


Figura 1.9: Operatori morfologici

L'erosione invece è il processo attraverso il quale si 'assottigliano' le forme.

L'operatore di apertura serve per eliminare delle interruzioni all'interno di un'immagine o dei piccoli elementi che escono fuori. Per fare un'apertura bisogna prima applicare un'erosione seguita da una dilatazione. L'operatore di chiusura invece serve per rendere più omogeneo il contorno dell'oggetto eliminando i vuoti all'interno di una figura. Per avere un effetto di chiusura invece bisogna prima applicare una dilatazione e poi un'erosione. L'immagine in *Fig. 1.9* mostra gli effetti di questi due operatori.

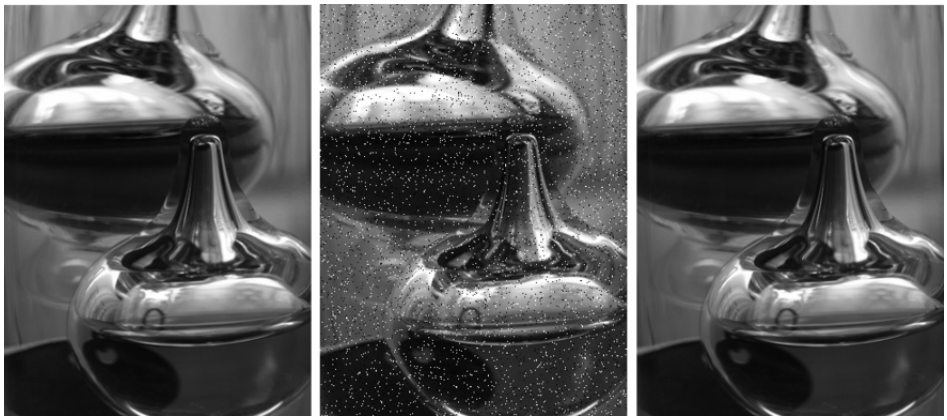


Figura 1.10: Aggiunta di rumore sale e pepe e rimozione del rumore tramite filtro mediano

- **Filtro mediano e filtro gaussiano:** Il filtro mediano è una tipologia di filtro digitale che serve per ripulire l'immagine dal rumore: un tipo di rumore su cui è molto efficace è quello 'sale e pepe'[4]. Lavora tramite un kernel[5], e

[4]Forma di rumore che consiste nella presenza di tanti pixel bianchi e neri.

[5]Matrice che definisce il vicinato su cui lavorare.

cambia il valore del pixel su cui è centrato con il valore mediano dei valori presenti all'interno del kernel. Un esempio pratico del lavoro che svolge è mostrato in *Fig. 1.10*

Il filtro gaussiano è sempre una tipologia di filtro per la rimozione del rumore. Il nuovo valore del pixel, su cui è centrato il filtro, sarà la media pesata dei valori dei pixel presenti all'interno del vicinato[6]. I pesi seguono l'andamento di una gaussiana.

- **Operatori di gradiente:** Gli operatori di gradiente non sono altro che dei filtri derivativi, attraverso i quali si studiano i cambiamenti dell'intensità di luminosità (solitamente) e tramite i quali, quindi, è possibile trovare i controrni[7] dell'immagine: il gradiente di un'immagine 2D si calcola tramite le derivate parziali, in quanto sono presenti due assi x, y. È possibile trovare i bordi poichè sono i punti dove si verifica la massima variazione di intensità luminosa, e pertanto, attraverso degli operatori differenziali si riescono a mettere in evidenza.

  I diversi operatori di gradiente esistenti presentano tutti questa caratteristica.

Operatori globali

Forniscono il valore di un pixel utilizzando i valori di luminanza[8] di tutti i pixel dell'immagine originale; il livello di grigio dei pixel è modificato con una funzione di trasformazione basata sull'istogramma dei livelli di luminanza (o di grigio).

1.4.2 Postprocessing

Segmentazione dell'immagine

La segmentazione è uno degli step fondamentali per poter riconoscere un oggetto all'interno di un'immagine. Questa procedura serve per individuare delle regioni all'interno dell'immagine, quindi, identificare delle relazioni tra i vari pixel e dare un senso a livello insiemistico di tutti i pixel all'interno di una regione. Lo scopo finale di questo step è quello di cambiare il significato delle immagini, matrici di pixel, in un'insieme di regioni. Ciò facilita poi l'analisi dell'immagine.
Per identificare una regione è necessario che i pixel all'interno della regione abbiano delle caratteristiche in comune, che saranno differenti dalle caratteristiche dei pixel appartenenti a una regione contigua.

[6] Insieme di pixel, escluso quello centrale, che stanno dentro il kernel

[7] Edge

[8] Luminosità di colore

Esitono varie tecniche per segmentare un'immagine, tra le più utilizzate si citano:

- **Region growing**

  Il region growing presenta un'approccio di lavoro abbastanza semplice: si parte da alcuni seeds[9](*Fig.1.11b*) che vengono scelti manualmente dall'operatore tramite ispezione visiva.

(a) *Immagine originale* **(b)** *Seeds* **(c)** *Region Growing*

Figura 1.11: Step del region growing

  I seeds sono dei punti appartenenti alla regione che deve essere identificata dall'algoritmo. La tecnica prende il nome di region growing poichè, inizialmente il numero di questi seeds è limitato e non individuano una regione, ma un insieme di pixel sconnessi che presentano la proprietà comune di appartenere alla stessa regione. Attraverso delle regole di espansione si fa "crescere" la regione: cioè tramite una soglia si identificano i pixel che hanno un valore superiore a un determinato valore e si includono nella regione. È una tecnica di base abbastanza dispendiosa, e potrebbe non identificare le zone d'ombra all'interno della regione che si sta cercando, inoltre, la presenza di rumore all'interno della regione potrebbe causare dei buchi all'interno dell'immagine.

- **Split**

  In modo improprio, si potrebbe considerare la tecnica di split come tecnica complementare a quella del region growing. Il concetto di partenza per questa tecnica di segmentazione è che l'immagine sia completamente omogenea; andando ad analizzarla, ogni volta che all'interno dell'immagine viene trovata una regione non omogenea, si divide l'immagine in regioni diverse. Questo processo poi viene iterato sino a che l'immagine non è completamente divisa in sole regioni omogenee.

[9]Semi. Particolari pixel da cui poi si sviluppa la regione.

- **Matching**

  La segmentazione basata sul matching è, a livello concettuale, la tecnica più semplice tra quelle presentate. Ci sono due possibili implementazioni di questa tecnica; quella basata sul riscontro di un modello e quella basata sul riscontro di determinate caratteristiche. L'implementazione del matching basata su riscontro del modello prevede l'uso dell'intero modello, e, attraverso il suo confronto con l'immagine, vengono calcolate determinate metriche per determinare la posizione ottimale del modello all'interno dell'immagine 1.12, se presente. Il secondo metodo si sviluppa allo stesso modo del primo, però, anzichè usare l'intera immagine modello, esegue il confronto tra determinate caratteristiche presenti nel modello e quelle, se presenti, nell'immagine.

Figura 1.12: Immagine originale, modello, immagine originale dove è stato individuato il modello.

1.4.3 Metodi di classificazione

La classificazione di un oggetto o di un'immagine è il processo tramite il quale si assegna all'oggetto/immagine un'etichetta che corrisponde alla classe dell'oggetto. Ci sono due tipi di classificazione: la classificazione esclusiva, cioè quando l'oggetto può appartenere solo a una classe, e la classificazione fuzzy, cioè quando l'oggetto possiede una percentuale di appartenenza ad ogni possibile classe.
I classificatori vanno addestrati, in modo da ottenere le performance migliori. L'addestramento, in genere, è supervisionato o non supervisionato. Cioè "si danno in pasto" le caratteristiche dell'input e si passa come secondo parametro la classe dell'oggetto da classificare. Durante questa fase il classificatore apprende le relazioni nascoste che sono presenti all'interno dell'oggetto. Può succedere che non venga passata la classe, in questo caso si ottiene un addestramento non supervisionato. Il classificatore non sa quale sia la classe reale dell'oggetto, conosce unicamente il numero delle classi.

Due esempi di classificatori sono il KNN e il classificatore bayesiano. Essi seguono rispettivamente un addestramento supervisionato e uno non supervisionato. Verranno poi descritti meglio nel capitolo 2.

1.5 Machine Learning: tecniche automatizzate

L'approccio umano per poter spiegare e capire un determinato fenomeno si basa sull'osservazione, estrazione di features di quel fenomeno, astrazione e deduzione; un processo che cerca di legare, tramite un approccio empirico, le cause del fenomeno all'effetto. La generalizzazione di queste leggi, che vengono trovate per spiegare il fenomeno, costituiscono poi quello che è un modello:

> " Un modello matematico è una rappresentazione quantitativa di un fenomeno naturale. Come tutti gli altri modelli usati nella scienza, il suo scopo è quello di rappresentare il più incisivamente possibile un determinato oggetto, un fenomeno reale o un insieme di fenomeni (modello matematico di un sistema fisico, sistema chimico o sistema biologico). Spesso il modello è una rappresentazione della realtà non perfetta, ma comunque fedele, ovvero significativa all'analisi o prognosi a cui si vuole arrivare."[10]

L'identificazione di un modello diventa sempre più complessa man mano che si complicano i fenomeni da descrivere. Un classico esempio è quello delle previsioni del meteo:

- Si osserva che tutte le volte che alla sera sono presenti nuvoloni il giorno dopo piove.
- Si misurano features di quel fenomeno: grandezza dei banchi di nuvole, colore, altezza...
- Si cerca il modello matematico che leghi l'input (le features) agli output (pioggia il giorno dopo).

[10] *https://it.wikipedia.org/wiki/Modello_matematico*

Prendendo un esempio più semplice, *"data l'immagine di una macchina individuare le ruote"*, è evidente che se per un essere umano è scontato trovare le ruote, per un computer non lo è, in quanto non possiede nozioni aprioristiche quali:

- Le ruote sono tonde
- Le ruote sono quattro e stanno sotto la macchina
- Le ruote stanno a contatto con l'asfalto
- Le ruote presentano un supporto attorno al quale sono montate

Questi concetti vanno modellizzati tramite il feature engineering che, nel campo dell'elaborazione delle immagini, rappresenta un task di estrema complessità.

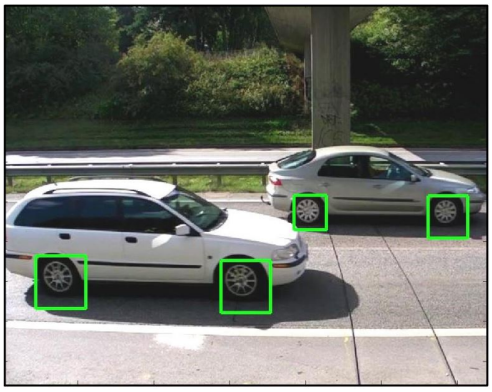


Figura 1.13: Rilevamento delle ruote

Il feature engineering è un approccio computerizzato di estrazione manuale delle caratteristiche di interesse.

Alcune delle tecniche che vengono usate dentro questa branca sono: l'enanchment delle immagini (aumentare la saturazione, la luminosità, applicare dei gradienti) e la segmentazione (trovare i contorni di un elemento d'interesse, la sua colorazione, la forma etc).

Quali sono le caratteristiche universali, indipendenti dal punto di vista e oggettive per estrarre queste feature per quanto riguarda una ruota? La forma cambia a seconda del punto di vista, le ruote non sono a contatto con l'asfalto se la macchina è capovolta, magari la macchina ha perso una ruota e ne restano solo tre. Appare chiaro come ci siano tantissimi fattori che possono influenzare la creazione di tale modello. Il Machine Learning è il processo per trovare un modello in maniera automatica partendo da osservazione fatte dall'utente, quindi è quest'ultimo a selezionare ed estrarre le features da dare in pasto alla famiglia di modello prescelta. Esistono diverse famiglie di modelli: i più noti lavorano per clusterizzazione, per separazioni lineari, per separazioni non lineari etc. Una volta scelte le features più significative e la famiglia di modello che meglio potrebbe rappresentare il nostro fenomeno, si va a verificare il suo corretto funzionamento o meno.

Il Deep Learning è una branca del machine learning che sfrutta le reti neurali per creare modelli basati su reti neurali profonde[11].

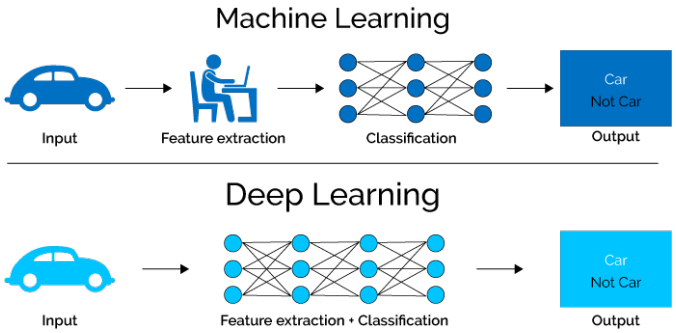


Figura 1.14: Due approcci per la creazione del modello: ML o DL

Indicativamente, una rete neurale viene definita profonda quando i layer interni alla rete sono superiori a circa quindici/venti. L'utente non sceglie più quali sono le features da estrarre e dare in pasto al classificatore piuttosto che non a una SVM. Il ruolo del programmatore ora è unicamente quello di creare l'architettura che la rete deve avere per poter permettergli di apprendere al meglio i concetti. Questa architettura è rappresentata dalle interconnessioni, mostrate in *Fig 1.14*, tra i neuroni dei vari strati della rete.

1.5.1 Reti neurali

Una rete neurale artificiale è un modello di calcolo matematico-informatico ispirato al funzionamento delle reti neurali biologiche. Il loro elemento strutturale, mostrato in *Fig. 1.15*, è il neurone; esso può essere considerato come un'unità computazionale che riceve in ingresso un numero k di input $(x_1, x_2, \ldots, x_k)$ e produce in uscita un output chiamato **attivazione** del neurone. Ogni neurone è caratterizzato da tre elementi:

- Una funzione di attivazione.
- Un coefficiente b.
- L'attivazione del neurone.

La funzione di attivazione è una qualsiasi funzione che definisce il comportamento computazionale del neurone. Viene accompagnata da una soglia che serve per definire i due possibili stati del neurone: attivo, nel caso in cui venga superata la soglia, spento nel caso in cui non si superi la soglia. Il coefficiente b, che prende il nome di **bias**, serve poichè non sempre la combinazione degli input, moltiplicati per dei determinati coefficienti chiamati pesi, produce il superamento della soglia

[11] Per la definizione di rete neurale si rimanda al paragrafo 1.3.1

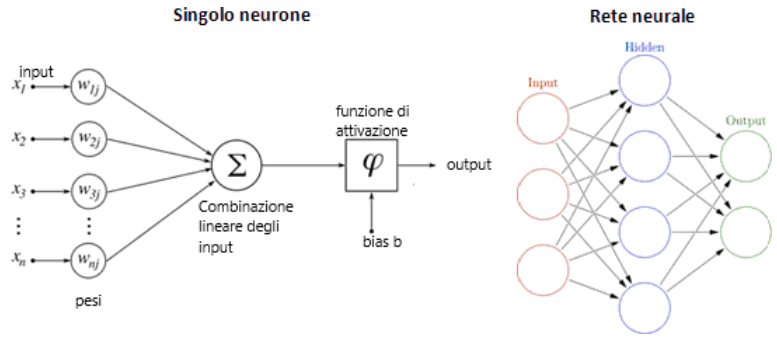


Figura 1.15: Struttura di un neurone singolo e quella di una rete.

minima della funzione di attivazione. Lo scopo del bias è quello di produrre una traslazione della funzione di attivazione.
Le reti neurali sono un modello costituito da un gruppo di interconnessioni tra neuroni artificiali che, tramite questi legami, modificano i loro pesi e si scambiano informazioni. Tali interconnessioni sono quelle che definiscono la cosiddetta architettura di una rete neurale, cioè il flow sequenziale che le informazioni seguono nel corso delle loro trasformazioni matematiche preimpostate all'interno dei neuroni d'arrivo. Tutte le reti neurali presentano 3 sezioni:

- Il blocco di input
- Il blocco di layer nascosti
- Il blocco d'output

Il blocco di input costituisce l'ingresso degli input della reta neurale. Il blocco costituito dai layer nascosti, luogo dove vengono svolte le principali operazioni matematiche, ha dimensioni variabili scelte dal programmatore e, più si aggiungono strati più la rete diventa *deep*. Il blocco di output è dove si raccolgono i risultati di tutte le operazioni precedenti, ed è lo strato che genera poi l'output.
Nella maggior parte dei casi una rete neurale è un sistema di tipo adattivo, cioè essa riesce ad aggiustare i coefficienti moltiplicativi di ogni neurone attraverso il calcolo dell'errore. Esso viene calcolato grazie a una metrica che prende in considerazione lo scarto tra la classe predetta e la classe reale dell'input. Successivamente i coefficienti vengono corretti portando indietro questo scarto attraverso il processo di **backpropagation**[12].
In termini pratici le reti neurali sono strutture non-lineari di dati statistici organizzate come strumenti di modellazione. Esse possono essere utilizzate per simulare

[12]Processo non sempre presente, tramite il quale si penalizzano i pesi.

relazioni complesse tra ingressi e uscite che altre funzioni analitiche non riescono a rappresentare.
Per poter usare una rete neurale, prima bisogna seguire sequenzialmente i seguenti step:

- Divisione del Dateset in maniera opportuna.
- Addestramento.
- Verifica sul test.
- Eventuale tuning degli iperparametri[13].
- Ripetizione delle fasi successive alla prima.
- Salvataggio della rete.

Una volta seguiti questi passaggi è possibile riusare la rete a proprio piacimento.

Divisione del Dataset

Il primissimo passaggio per la corretta costruzione e uso di una rete neurale, è quello che prende il nome di **Data Analysis**. Essenzialmente è uno studio dei dati di cui si dispone: si va a studiare se sono presenti tutti i dati o se ci sono dati che mancano, la loro distribuzione, si vede se ci sono degli outliers etc.
Dopo questa primissima analisi del dataset, bisogna scomporlo in due/tre sotto gruppi: il **Training Set** e il **Test Set** sono i due sotto gruppi obbligatori. Volendo, dal training set è possibile creare il terzo sottogruppo, quello del **Validation Set**. È una divisione importante in quanto i dati del training set verranno usati unicamente per allenare la rete, quelli del validation, se presente, per andare a fare una prima verifica sulle prestazioni dell'allenamento ogni volta che il training set è completato, quelli del test set infine verranno usati durante la fase di verifica, come ultimo passaggio di validazione in quanto dati mai visti prima.
Le proporzioni che si cerca di tenere sono tipicamente 60%-20%-20% del dataset, rispettivamente per il training, validation e test set. In questo caso, sono riuscito ad ottenere come miglior divisione possibile quella di *Fig 1.16*.

[13]Parametri che permettono di impostare al meglio determinati comportamenti della rete.

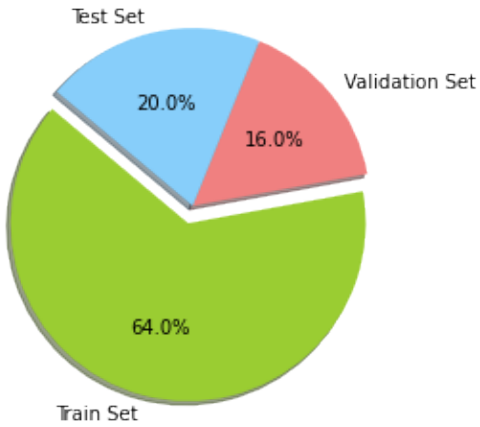


Figura 1.16: Divisione del dataset

Addestramento

Affinchè queste reti riescano nel loro scopo, dopo avere diviso i dati e creato l'architettura, bisogna iniziare ad allenarle.
Per poter allenare una rete si usano i dati presenti nel training set. La fase di allenamento è essenziale nonchè di cruciale importanza all'interno del DL[14]; le motivazioni principali sono due:

- Bisogna che la rete impari i concetti che poi dovrà essere in grado di distinguere da sola.
- È uno step necessario per verificare la correttezza dell'architettura.

Esistono diverse tipologie di addestramento/allenamento ma, sostanzialmente, tutte le metodologie presentano come punto in comune la ripetizione sequenziale dell'input di ingresso, che potrebbe essere una serie di immagini mediche, per un determinato numero di epoche[15]. Solitamente si divide l'input in ingresso in tanti *gruppetti* per poter permettere una differenziazione maggiore delle singole caratteristiche estratte dall'input: questi gruppi prendono il nome di **batch**. Una volta diviso il dataset nei diversi batch è importante fare la **batch normalization**: processo di standardizzazione di tutti i diversi batch. Tale processo assume un ruolo di importanza fondamentale in quanto:

[14]Deep Learning

[15]Numero di cicli di ripetizione dell'input.

- I batch potrebbero avere delle differenze a livello di valori presenti nelle immagini, (es. batch ha delle features che variano tra 0 e 5, il batch dopo presenta le stesse features che variano tra 7 e 14 e così via.) nel caso, vanno normalizzate tra 0 e 1.
- Velocizza tutta la fase di allenamento.
- Rende capaci i singoli layer di apprendere maggiormente.

L'allenamento è la fase dove viene generata la conoscenza, quindi dove la rete apprende.
Le metodologie di allenamento più famose per quanto riguarda le reti neurali sono:

- **Apprendimento supervisionato:** In questa modalità al computer vengono passati sia il set di dati come input sia le informazioni relative ai risultati desiderati. Viene fatto ciò con l'obbiettivo che il sistema identifichi una regola generale che colleghi i dati in ingresso con quelli in uscita (gli vengono cioè forniti degli esempi di input e di output in modo che impari il nesso tra loro), in modo da poter poi riutilizzare tale regola per altri compiti simili.
- **Apprendimento non supervisionato:** In quest'altra categoria di apprendimento non vengono forniti al calcolatore input e output ma unicamente l'input e il numero di classi in cui vanno divisi i dati in input. Sarà compito del calcolatore riuscire ad astrarre i concetti fondamentali, a trovare gli schemi o le relazioni matematiche nascoste in grado di costruire un modello che classifichi in modo corretto.
- **Apprendimento semi-supervisionato:** Quest'ultima tipologia di apprendimento è di tipo ibrido; come suggerisce il nome, fornisce una parte di dati con input e output, e un'altra parte con il solo input.

Verifica

La verifica è un passaggio essenziale per poter essere sicuri che la rete abbia appreso i concetti in modo adeguato: sappia trovare le relazioni necessarie a classificare l'oggetto di indagine (una serie di dati numerici, una foto, etc) in modo corretto, ma, soprattutto per verificare che la rete non vada in overfitting. [16]
Per fare ciò si usa quello che viene chiamato il **test set**, cioè una serie di dati su cui la rete non si è allenata e che non ha mai visto prima. Di questo set è essenziale conoscere la reale classificazione dei dati, in modo da poterla confrontare dopo con quella predetta dalla rete.

[16] Quando la rete impara *"a memoria"* le caratteristiche dei dati del training set.

Dopo il confronto si valutano le prestazioni della rete, e in base a queste si può decidere:

- La rete non raggiunge le prestazioni volute:
  - Cambio l'architettura della rete
  - La rete necessita di altre epoche di addestramento
  - La rete sta overfittando, devo allenarla di meno
  - Il dataset non è stato diviso in maniera consona e la rete non riesce a generalizzare.
- La rete ottiene le prestazioni volute.

Una volta finita questa fase in modo positivo, il modello è pronto per essere usato.

Capitolo 2

Lo stato dell'arte

2.1 Feature extraction e classificazione

Esistono alcune procedure di feature extraction e classificazione che, combinate assieme, permettono una classificazione automatica ottenendo prestazioni molto elevate, senza dovere ricorrere all'uso delle DNN.
Queste procedure sfruttano precise teorie matematiche di ottimizzazione per arrivare al risultato. All'interno della classificazione di lesioni cutanee, alcune delle migliori tecniche sono:

- Feature extraction:
  - Segmentazione tramite triangolazione di Delaunay.
  - Campi di Markov e classificazione bayesiana.
- Classificazione:
  - Adaboost
  - Classificatore Bayesiano

2.1.1 Feature extraction

Segmentazione tramite triangolazione di Delaunay

Dato come vero il teorema di Kneser, allora è vero che:

Data una superficie normale qualunque è sempre possibile triangolarla.

Questa operazione potrebbe però richiedere un numero di triangoli infiniti. Un esempio di triangolazione è mostrato in *Fig. 2.1.*
Sono mostrati tre diversi esempi di triangolazione di cui, solo i primi due, sono corretti.

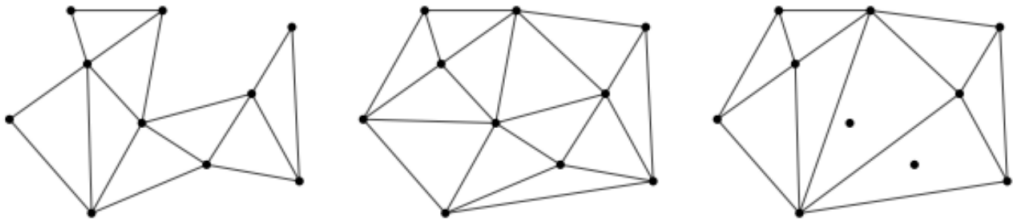


Figura 2.1: A sinistra: triangolazione di un poligono. Al centro: triangolazione di un insieme di punti. A destra: triangolazione non valida.

Teorema di Delauny. *Una triangolazione di un insieme di punti $P \subset \mathbb{R}^2$ viene detta di Delaunay se il cerchio circoscritto ad ogni triangolo è vuoto, ovvero nessun punto di P vi giace all'interno.*

Da cui deriva che:

- Ogni insieme di punti (non tutti collineari tra loro) ha una sola triangolazione di Delaunay.
- Ogni triangolazione di Delaunay massimizza il più piccolo angolo interno tra tutte le triangolazioni possibili.

Questa tecnica è stata usata in un lavoro di D. Devescovi[7] per la segmentazione e classificazione delle lesioni cutanee. In questo lavoro è stato preso il dataset PH2 contenente un totale di 200 immagini divise in 80 nevi, 80 nevi displastici e 40 melanomi. Prima di applicare la triangolazione, è stato fatto un preprocessing consistente in:

- Filtro gaussiano con un kernel σ=5 su ogni layer preso singolarmente (RGB).
- Cambio di rappresentazione in scala di grigi ed edge detection attraverso l'uso dell'algoritmo Canny.

I bordi trovati tramite Canny sono stati vettorizzati in segmenti connessi e questo è il parametro di ingresso che è stato passato all'algoritmo per la triangolazione di Delaunay
Lo step in *Fig. 2.2b* mostra il risultato della triangolazione. Si evidenzia come i triangoli più grandi che vengono creati sono locati in zone omogenee, mentre quelli più piccoli vengono localizzati nei bordi, dove la colorazione non è più omogenea e varia.

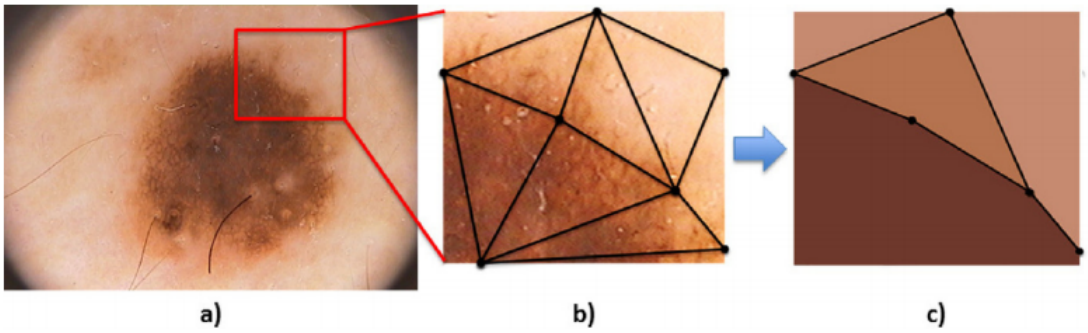


Figura 2.2: Step eseguiti durante la triangolazione.

Questo step di triangolazione non risulta però sufficiente in quanto si divide il neo in tanti triangoli diversi. Per segmentare in modo ottimale è stato eseguito un ulteriore passaggio, per cercare di uniformare la ROI[17]. Per poter unire i triangoli che presentano proprietà simili, a livello di intensità di colorazione è stato usato l'algoritmo di *Region Association*: vengono unite iterativamente le due regioni contigue che presentano il minor costo di normalizzazione rispetto a un treshold fissato. Il risultato è mostrato in *Fig. 2.2c*

Tabella 2.1: Tipi di segmentazioni.

Metodo di segmentazione	Totalmente automatico	Categoria di segmentazione
JSEG	No	Identificazione della regione
SRM	No	Identificazione della regione
KPP	No	Thresholding
K-means	No	Thresholding
Otsu	Si	Thresholding
Level-Set	Si	Contour
ASLM	Si	Identificazione della regione

[17]Regione di indagine.

Per poter fare un'analisi quantitativa sull'accuratezza della segmentazione fatta da questo algoritmo, ALSM, il risultato è stato messo a confronto con altri sei metodi di segmentazione noti. Questi metodi si differenziano per essere automatici o meno, e nel tipo di segmentazione che applicano, come elencato in *Tab. 2.1.* Le metriche usate per poter confrontare i risultati sono sensibilità, specificità, accuratezza e F1:

Tabella 2.2: Risultati di diverse segmentazioni.

Metodo di segmentazione	Sensibilità	Specificità	Accuratezza	F1
JSEG	0.7108	0.9714	0.8947 ±0.0176	0.7554
SRM	0.1035	0.8757	0.6766 ±0.0346	0.1218
KPP	0.4147	0.9581	0.7815 ±0.0356	0.5457
K-means	0.7291	0.8430	0.8249 ±0.0107	0.6677
Otsu	0.5221	0.7064	0.6518 ±0.0203	0.4293
Level-Set	0.7188	0.8003	0.7842 ±0.0295	0.6456
ASLM	0.8024	0.9722	0.8966 ±0.0276	0.8257

Come si vede dai risultati in *Tab. 2.2* il metodo implementato nel paper[7] ottiene delle prestazioni nettamente sopra la media rispetto agli altri metodi di segmentazione.

Features trovate:

Le features che si sono usate per allenare i classificatori, oltre la maschera di segmentazione trovata grazie alla segmentazione di Delaunay, sono state:

- Geometriche:
  - Area convessa: un valore scalare che è la misura del numero di pixel che racchiudono l'area della maschera di segmentazione.
  - Pienezza dell'area: un valore scalare che specifica il numero di pixel della lesione nella maschera di segmentazione con i buchi riempiti.
  - Solidità: valore scalare che fornisce il rapporto Area/Area Convessa.

- Colorazione:
  - Un istogramma per il layer H
  - Un istogramma per il layer S
  - Un istogramma per il layer V

Tutti e tre gli istogrammi sono stati plottati con sedici bins e rappresentano i valori normalizzati dei pixel colorati estratti dall'immagine originale tramite sovrapposizione della maschera di segmentazione. Le caratteristiche geometriche sono state scelte per avere dei parametri solidi per descrivere l'irregolarità dei bordi. Dall'immagine in *Fig. 2.3* si può intuire come riescano ad applicare una prima separazione in modo abbastanza netto:

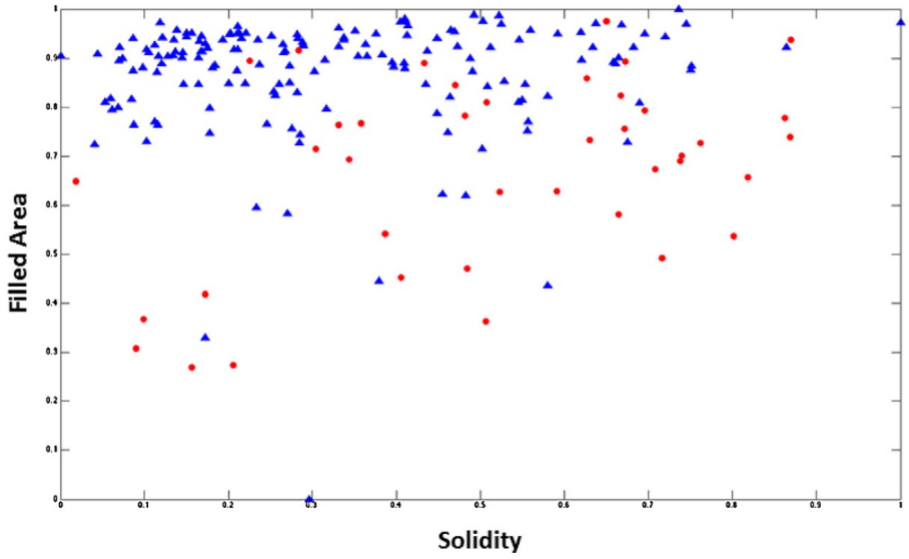


Figura 2.3

Campi di Markov e classificazione bayesiana

Campi casuali di Markov

I campi casuali di Markov sono un modello aleatorio che è stato usato anche nel campo dell'image processing, come metodo di segmentazione. Perchè il risultato dell'applicazione di questo metodo abbia senso, bisogna che venga usato in maniera congiunta a un classificatore bayesiano.
Nell'ambito della segmentazione, i MRF[18] cercano di massimizzare la probabilità di

[18]Campi casuali di Markov

identificare un particolare sistema di etichettatura data un immagine con diverse caratteristiche. I MRF sono completamente caratterizzati da distribuzioni di probabilità a priori, probabilità marginali e dalle cricche.
Il teorema di Bayes è importante in quanto, il solo lavorare con conoscenze a posteriori è praticamente impossibile, e, grazie a questo teorema si possono dividere le conoscenze a priori da quelle a posteriori. L'introduzione degli a priori ha alcuni vantaggi, quali:

- Cambia l'occorrenza delle classi.
- Esistono delle dipendenze spaziali di cui tener conto.
- L'immagine da segmentare presenta strutture geometriche ricorrenti.
- Bisogna che i contorni abbiano un certo grado di regolarità.

I campi di Markov costituiscono quindi l'a priori che viene introdotto nella classificazione bayesiana per la segmentazione dell'immagine:

> *Un campo di Markov è un processo bidimensionale in cui un determinato valore x_s di un dato elemento s(di un pixel in questo caso) è indipendente dagli altri elementi dato lo stato di un opportuno insieme di vicini.*

Per poter definire un MRF è necessario introdurre la nozione di cricca, in quanto, quella di vicinato è stata già spiegata.
Cricca: si definisce una cricca $c \in C$ un punto o un insieme di punti mutuamente vicini in accordo al tipo di vicinato $\eta(s)$ aadottato. Una cricca è quindi una sottoporzione del vicinato, può essere monodimensionale, bidimensionale, etc. Le applicazioni per le cricche con dimensionalità superiore a 2 sono limitate.

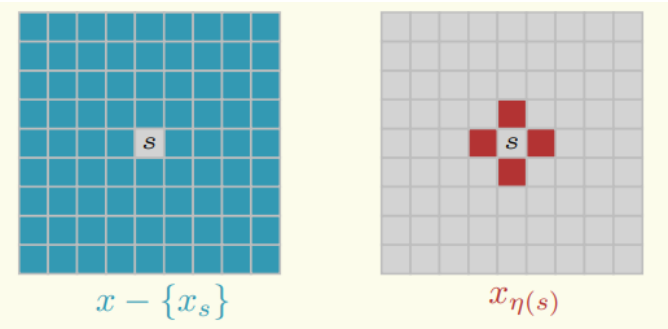


Figura 2.4: Vicinato monodimensionale

Un campo aleatorio X è un campo di Markov se vale la seguente proprietà:

$$p(x_s) > 0$$
$$p(x_s \mid x - \{x_s\}) = p(x_s \mid x_{\eta(s)})$$

Questa proprietà viene definita *markovianità*, e significa che; la probabilità che un pixel assuma il valore x_s è indipendente dall'intero insieme di pixel, e dipende unicamente da quelli identificati dal vicinato.
Per poter calcolare $p(x_s)$ a partire da $p(x_s \mid x_{\eta(s)})$, si ricorre ai GRF[19][8].
Un campo casuale F è definibile come campo casuale di Gibbs se:

$$p(x_s) = Z^{-1} * e^{-\frac{1}{T}U_{(xs)}}$$

Dove;

- $Z = \sum_{x_s \in F} e^{-\frac{1}{T}U_{(xs)}}$ è una costante normalizzata calcolata su tutte le possibili combinazioni dei pixel.
- T è un parametro che serve per controllare la distribuzione dei pattern di x_s, solitamente la si mette a 1.
- $V_c(x_s)$ è il valore di energia potenziale che assume la cricca. In genere ogni cricca assume un valore diverso.
- $U(x_s) = \sum_{c \in C} V_c(x_s)$ è la funzione energia per il pattern x_s considerato; è dato dalla somma di tutte le energie potenziali di tutte le cricche presenti nell'insieme dei siti del vicinato.

Ne deriva che minore è l'energia misurata per un determinato pattern, maggiore è la probabilità che venga riscontrato quel pattern. L'energia può essere scritta come:

$$U(x_s) = \sum_{s \in S} V_1 x_s \sum_{s \in S} \sum_{s'_i \in \eta_{s_i}} V_2(x_s, x_{s'_i})$$

Si può notare come MRF calcoli la probabilità condizionata mentre GRF calcola la probabilità globale. Ed è il calcolo di quest'ultima che interessa nell'ambito dell'elaborazione di immagini mediche poichè, non interressa il valore del singolo pixel, bensì di tutta l'immagine.

[19]Campi casuali di Gibbs.

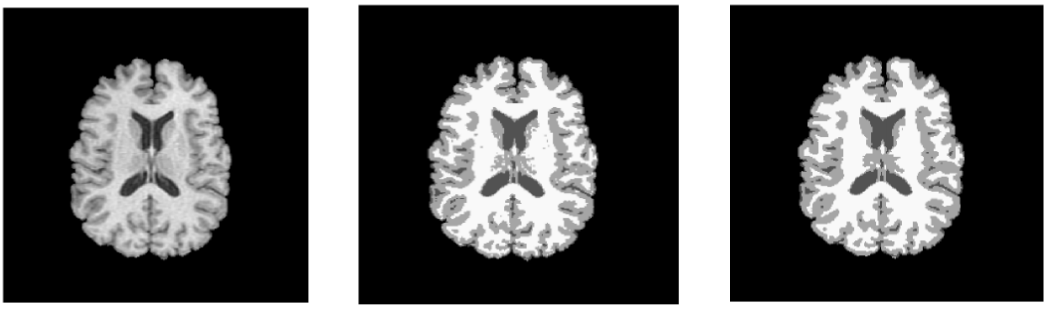
Grazie al teorema presente in[9] è stata validata l'equivalenza tra i due campi; ciò consente di calcolare la markovianità a partire dai potenziali di cricca, che rappresentano i vincoli locali. Essi sono misurabili da $p(x_s \mid x_{\eta(s)})$; infatti l'equivalenza stabilisce che l'RMF, per avere le proprietà di positività e markovianità, deve avere una distribuzione di Gibbs.
Quindi la prima parte dell'equivalenza diventa:

$$p(x_{s_i} \mid x_{\eta_{s_i}}) = \frac{e^{-(V_1(x_{s_i}) + \sum_{s'_i \in \eta_{s_i}} V_2(x_{s_i}, x_{s'_i}))}}{\sum_{x_{s_i} \in L} e^{-(V_1(x_{s_i}) + \sum_{s'_i \in \eta_{s_i}} V_2(x_{s_i}, x_{s'_i}))}}$$

La probabiltà globale invece $p(x_s)$ è data da:

$$p(x_s) = \frac{e^{-\frac{1}{T}[\sum_{s \in S} V_1(x_s) + \sum_{s \in S} \sum_{s' \in \eta_{s_i}} V_2(x_s, x'_s)]}}{\sum_{x \in F} e^{-\frac{1}{T}[\sum_{s \in S} V_1(x_s) + \sum_{s \in S} \sum_{s' \in \eta_{s_i}} V_2(x_s, x'_s)]}}$$

Pertanto $p(x_s)$ è la probabilità a priori, che si può calcolare fornendo dei precisi valori alle funzioni potenziale $V_c(x_s)$, così che vi sia un comportamento "voluto" per i pixel dell'immagine. È importante sottolineare che un a priori non ha un vero legame con l'immagine da segmentare; adottandolo all'interno della stima bayesiana e scegliendo un criterio di massimizzazione (MAP per esempio), la segmentazione si traduce nella ricerca del massimo di una distribuzioni a posteriori, dove l'MRF è usato come a priori. Un esempio di come sia un algoritmo robusto e poco sensibile alle piccole variazioni locali dovute al rumore è mostrato in *Fig. 2.5c.* In tale lavoro [10] non è stato usato un classificatore bayesiano ma un KNN.



(a) *Immagine originale* **(b)** *KNN* **(c)** *RMF + KNN*

Figura 2.5: 2.5a) Immagine originale, 2.5b) Segmentazione ottenuta tramite KNN, 2.5c) Segmentazione ottenuta con RMF e KNN

2.1.2 Classificazione

AdaBoost

AdaBoost è un classificatore ensemble. Un classificatore ensemble è un classifcatore che si compone di diversi algoritmi di classificazione deboli, i cui output, se combinati tra loro, danno luogo a un classificatore forte.
Un classificatore viene detto debole se riesce a classificare il $50\% + 1$, quindi con una scarsa accuratezza. Mettendo insieme tanti modelli di questo tipo, l'AdaBoost riesce a generare un modello che complessivamente è migliore dei singoli classificatori deboli presi singolarmente.
Come classificatore debole da addestrare, l'Adaboost si avvale di tanti alberi decisionali con un solo livello di profondità. Questi alberi vengono definiti *decision stumps*, e sono tanti quanti sono le caratteristiche del modello. Ad ogni iterazione viene introdotto in sequenza un nuovo classificatore debole con lo scopo di compensare gli errori dei modelli precedenti per creare un classificatore forte, che è un classificatore con un'alta accuratezza e specificità. L'obiettivo generale di questa azione è quello di creare nuovi modelli che, iterazione dopo iterazione, si adattano ai miglioramenti per fornire stime più accurate.

Funzionamento di AdaBoost:
Lo scopo finale di questo algoritmo è quello di trovare i giusti pesi per ogni classificatore e addestrarli. Supponiamo di analizzare un problema di classificazione binaria nelle classi uomo o donna: gli step sequenziali che l'algoritmo segue possono essere divisi in sette brevi punti[20]:

- **Calcolare i pesi dei dati:**

  Dato un set di dati di lunghezza n, vale la seguente relazione:

  $$\forall X_i \in \mathbb{R}^d, y_i \in \{-1,1\}$$

  dove -1 rappresenta la donna e 1 l'uomo.

  I pesi $w(x_i, y_i)$ vengono inizializzati come

  $$w(x_i, y_i) = \frac{1}{n}, i = 1, ..., n$$

  Supponiamo che i dati abbiano una forma come in *Tab. 2.3*, e che la loro lunghezza totale sia di sette elementi.

[20] https://lorenzogovoni.com/cose-lalgoritmo-adaptive-boosting-adaboost/

Tabella 2.3

Altezza	Peso	Età	Classe
≥ 180	≥ 75	≥ 30	Uomo
< 180	< 75	< 30	Uomo
≥ 180	< 75	< 30	Donna
< 180	≥ 75	≥ 30	Uomo
< 180	≥ 75	≥ 30	Donna
< 180	≥ 75	≥ 30	Uomo
≥ 180	≥ 75	≥ 30	Uomo

Quindi i campioni saranno inizializzati tutti con pesi pari a $\frac{1}{7}$

- **Scelta del classificatore debole:**

  Si possono scegliere diversi tipi di classificatori deboli, supponiamo di scegliere i decision stumps; per ogni carateristica verrà quindi creato un decision stump e AdaBoost sceglie quello a minor entropia, che in questo caso si calcola dalle frequenze degli attributi. Calcolando l'entropia E per ogni attributo si vede che:

  E = 0,8571, prendendo altezza e classe.

  E = 0,3936, prendendo peso e classe.

  E = 0,8014, prendendo età e classe.

  Quindi verrà scelto il classificatore peso e classe, in quanto quello ad entropia minore, che in generale è quello che fa meno errori tra tutti gli altri.

- **Calcolo delle performances:**

  Per capire come calcolare le performances del classificatore, anche chiamate amount of saying, bisogna prima calcolare l'errore totale **TE**. Supponendo di aver ottenuto queste previsioni:

Tabella 2.4

Altezza	Peso	Età	Classe	Previsione
≥ 180	≥ 75	≥ 30	Uomo	Uomo
< 180	< 75	< 30	Uomo	Uomo
≥ 180	< 75	< 30	Donna	Uomo
< 180	≥ 75	≥ 30	Uomo	Uomo
< 180	≥ 75	≥ 30	Donna	Donna
< 180	≥ 75	≥ 30	Uomo	Uomo
≥ 180	≥ 75	≥ 30	Uomo	Uomo

Le performance ottenute vengono divise in tre sottoclassi:

- Positivo: per valori di TE > di 0,5;
- Nullo: per valore di TE = 0,5;
- Negativo: per valori di TE < di 0,5;

Si vede che l'errore totale è pari a $\frac{1}{7}$. Una volta calcolato il TE, le performance PS del decision stump si calcolano attraverso la seguente formula:

$$PS = \frac{1}{2} ln(\frac{(1-TE)}{TE}) = \frac{1}{2} ln(6) = 0{,}896$$

- **Calcolo dei nuovi pesi:**

  Per l'AdaBoost, le classificazioni sbagliate avranno un peso maggiore rispetto a quelle giuste. Infatti i pesi per le classificazioni corrette vale la seguente equazione:

$$w_{nuovo} = \frac{w_{vecchio}}{2 * (1-TE)} = \frac{1}{7} * \frac{7}{12} = 0{,}0833$$

  Mentre i pesi delle classificazioni sbagliate vengono impostati pari a 0,5.

- **Creazione del nuovo dataset sulla base dei nuovi pesi:**

  Il dataset viene diviso in una distribuzione cumulativa con pesi che variano tra 0 e 1 utilizzando i pesi appena creati. Ciò significa che la prima istanza ricade all'interno dell'intervallo $(0 - 0{,}083)$, la seconda in $(0{,}083 - 0{,}1646)$ e così via, sino a raggiungere il valore 1.

Questi valori servono per decidere il nuovo ordine del dataset; infatti estraendo n valori da 0 a 1 si decide il nuovo ordine del dataset a seconda di dove ricade il valore estratto. Supponendo le seguenti estrazioni si avrà il dataset:

Tabella 2.5

Classe	Previsione	Nuovi Pesi	Divisione Dataset	Estrazione	Nuovo Dataset
Uomo	Uomo	0,083	0,03	0,25	Donna
Uomo	Uomo	0,083	0,167	0,4	Donna
Donna	Uomo	0,5	0,667	0,87	Uomo
Uomo	Uomo	0,083	0,750	0,62	Donna
Donna	Donna	0,083	0,833	0,14	Uomo
Uomo	Uomo	0,083	0,917	0,34	Donna
Uomo	Uomo	0,083	1,00	0,71	Uomo

Come si vede confrontando la seconda colonna con l'ultima di *Tab. 2.5*, le previsioni uno, due, quattro, cinque e sei sono sbagliate. Questo nuovo dataset verrà testato e verificato da un nuovo decision stump.

- **Ripetizione:** Si ripetono i punti precedenti ad esclusione del primo, fino a quando i dati di addestramento non ottengono più miglioramenti o fino al raggiungimento massimo di classificatori (n). Ad ogni ripetizione vengono aggiornati i pesi, quindi cambia il TE e influenzerà l'iterazione successiva.

- **Classificatore finale:** Una volta completate tutte le iterazioni, tutti i classificatori deboli vengono combinati tra loro per formare un classificatore forte. Questo classificatore esegue la previsione, calcolando la somma delle performance degli stump precedenti, divise per classe, e sceglie la classe che presenta la somma più alta.

I risultati ottenuti con la triangolazione di Delanuay, l'estrazione di features da Devescovi [7] sono stati testati su quattro diversi classificatori per capire quale avesse i risultati migliori:

Tabella 2.6: Classificatori messi a confronto.

Classificatore	Sensibilità	Specificità	Precisione	F1
KNN	0.875	0.706	0.872	0.873
Bayes	0.825	0.806	0.863	0.836
AdaBoost	0.935	0.871	0.936	0.935
Random-Trees	0.890	0.804	0.895	0.892

Da questo lavoro è evidente come l'estrazione di features particolarmente significative, sia in grado di portare risultati alti in tutti i classificatori testati. AdaBoost ottiene risultati che sono comparabili per qualità a quelli delle reti neurali.
È importante sottolineare però che dei risultati così alti in tutti i classificatori sono stati possibili unicamente grazie alla combinazione di features particolarmente significative per la descrizione di ogni classe e al fatto che ci fossero solo tre classi di indagine.

Classificatore Bayesiano

Il problema di segmentazione delle immagini può anche essere interpretato come un problema di tipo Bayesiano[11]. Data un'immagine y, sia $P(\Omega)$ una partizione dell'immagine y, quindi la segmentazione della stessa. Il modello bayesiano definisce la probabiltà con cui questa partizione è una "corretta" segmentazione dell'immagine [12].
Per trovare quindi la segmentazione migliore si deve trovare la segmetazione che massimizza a posteriori la probabilità:

$$max\{p(P(\Omega) \mid y)\}$$

Massimizzare questa probabilità è uguale a minimizzare l'opposto del logaritmo di quella probabilità:

$$max\{p(P(\Omega) \mid y) = -log\ p(P(\Omega) \mid y)\}$$

Ora, visto che si sta minimizzando una probabilità condizionata, è possibile applicare il teorema di Bayes per separare le informazioni che conosciamo a priori da quelle che si possono conoscere solo a posteriori:

$$p(P(\Omega) \mid y) = \frac{p(y \mid P(\Omega))p(P(\Omega))}{p(y)}$$

dove $p(P(\Omega))$ è la probabilità di avere una determinata segmentazione a prescindere dall'immagine. All'interno di questa probabilità va inserito un termine di regolarizzazione TV_g, che serve per penalizzare i bordi meno uniformi e lunghi.

$$p(P(\Omega)) \propto \ e^{\ -(\nu TV_g(\phi))}$$

Il termine TV indica la *Total Variation*, e il temrine $p(y \mid P(\Omega))$ è la probabilità di avere una determinata immagine data una segmentazione a priori. Supponendo che non ci sia nessuna relazione tra le due regioni identificate dalla segmentazione, parte esterna Ω_E e la parte intern Ω_I, allora è vero che:

$$p(y \mid P(\Omega)) = p(y \mid \{\Omega_E, \Omega_I\}) = p(y \mid \Omega_E)p(y \mid \Omega_I)$$

Infine, supponendo che i pixel all'interno della stessa regione siano delle realizzazioni indipendenti e identicamente distribuite di una stessa variabile aleatoria con densità di probabilità $f_i(x)$ con i= I, E, si ottiene che il metodo bayesiano consiste nella minimizzazione della seguente quantità:

$$- \prod_{x \in \Omega_I} f_I(y(x)) \prod_{x \in \Omega_E} f_E(y(x)) \ e^{\ -(\nu TV_g(\phi))}$$

Usando i logaritmi, si ottiene la scrittura sotto forma di integrali, dove il primo termine è chiamato il termine di fedeltà della regione interna, il secondo della regione esterna e il terzo, derivante dalla moltiplicazione dell'esponenziale, è il termine di penalizzazione per avere dei bordi regolari.

$$F_y(P(\Omega)) = - \int_{\Omega_I} \phi(x) \log f_I(y(x))dx + \int_{\Omega_E} \phi(x) \log f_E(y(x))dx + (\nu TV_g(\nabla\phi(x)))dx$$

L'ulitma cosa che manca da definire è come trovare la densità di probabilità interna ed esterna, e le soluzioni sono due. La prima prevede di stimare la densità di probabilità tramite un metodo non parametrico; una buona soluzione potrebbe essere usare l'istogramma delle luminosità, in quanto si ha un elevato numero di pixel e potrebbe essere rappresentativo. Così facendo però si resta sensibili al rumore e alle piccole fluttuazioni luminose. La seconda soluzione prevede di usare una densità parametrica: si definisce a priori la forma della densità e poi si calcolano i parametri. Per esempio si può supporrre che la distribuzione di probabilità assuma la forma di una gaussiana:

$$f_i(s) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{\ \frac{|\ x - \mu_i\ |^2}{2\sigma_i^2}}$$

Con i che indica la probabilità interna I o quella esterna E. Risolta la penultima equazione, si ottiene una segmentazione che massimizza le probabilità che le condizioni imposte dalla densità interna e dalla esterna siano rispettate, e che questa segmentazione abbia i bordi i più regolari possibili.

2.2 DNN

Attualmente, per quanto riguarda le tecniche di riconoscimento, di clusterizzazione, e non solo queste, lo stato dell'arte è rappresentato dalle DNN[21].
Una DNN è una rete neurale in grado di apprendere l'estrazione di features a partire da dati complessi al fine di estrarre le feature stesse, o per eseguire una classificazione basandosi su di esse. Il concetto di apprendimento per una rete neurale è strettamente legato al concetto matematico di minimizzazione di una funzione, $J(\phi)$ ad esempio, che prende il nome di **loss function**; una rete apprende quando minimizza o massimizza la sua loss function; affinchè si riesca a minimizzarla si segue un gradiente nel suo verso di decrescita.
Il costo computazionale di una backpropagation su tanti neuroni concatenati è notevole, ben di più della creazione di un modello non basato su reti neurali profonde. In compenso il vantaggio dato è la possibilità di automatizzare il processo di feature extraction, a patto di avere sufficienti dati ed una architettura capace di apprendere i "concettetti" necessari a generalizzare il fenomeno in analisi. Le features apprese da una rete neurale non necessariamente hanno senso per un essere umano, però lo hanno per il computer.
Le operazioni più comuni che vengono fatte dentro queste reti sono quelle di:

- **Convoluzione:** è il layer principale, in quanto in questo strato si crea un kernel convoluzionale che permette alla rete di fare diverse operazioni, tra le quali:
  - Operazioni di blurring.
  - Operazioni di sharpening.
  - Operazioni di edge detection.

  Di questo parametro è importante specificare le sue dimensioni, che variano a seconda del compito che la rete deve svolgere e a seconda delle dimensioni dell'input; solitamente non sono molto estesi in altezza e larghezza, però si estendono per tutta la profondità del volume di input. Inoltre si potrebbero impostare anche altri iperparametri come il Kernel-Regularizer che va a

[21] Deep Neural Network

modificare i pesi del Kernel stesso, l'Activity-Regularizer che impone una funzione di regolarizzazione sull'output di questo strato, etc. Durante il processo di forwardpropagation[22] si convolve il kernel (che rappresenta il filtro) lungo la larghezza e l'altezza del volume d'input. Ogni filtro produce un **Activation map** bidimensionale. La rete avrà come obbiettivo generale quello di apprendere dei filtri che si attivino in corrispondenza di una determinata caratteristica presente in una porzione dello spazio di indagine dell'input. La sovrapposizione di tutte le activation map lungo la direzione della profondità, crea il volume di output di un layer convoluzionale. I layer convoluzionali non devono per forza essere tridimensionali. Lo stesso processo è applicabile anche in 2D.

- **Average/Max Pooling**: questo layer attua una riduzione dei parametri presenti; solitamente viene fatto dopo lo strato convoluzionale in quanto, quest'ultimo, aumenta il numero di parametri da gestire.

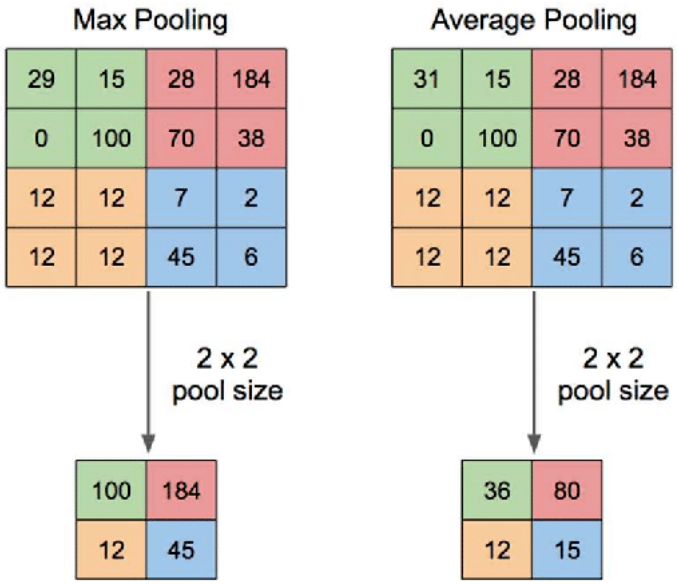


Figura 2.6: Differenze nel max pooling e average pooling

Come si vede dalla *Fig. 2.6*, il parametro principale che va specificato è la dimensione della matrice di pooling, che sarà la matrice d'output. La sostanziale differenza tra le due modalità è che, il max pooling salva unicamente (nel caso in figura) solo un quarto dei parametri (come specificato dalle dimensioni d'output), prendendo il massimo di ogni quadrante. L'average

[22]Normale andamento delle informazioni; dall'ingresso all'uscita.

pooling, al contrario, compie un' operazione di media all'interno di ogni quadrante, e in questo modo, a parità di numero di parametri d'output, porta un maggior numero di informazioni[23]

- **Dropout:** questo layer viene usato per implementare l'omonima funzione di Dropout: questa funzione consente di settare a 0 una determinata percentuale dei neuroni, quindi di 'spegnerli'. È spesso posizionata dopo uno strato fully connected in quanto, superato il layer, ne deriva un grande numero di parametri allenabili per la rete. Da un lato, aver tanti parametri allenabili è un evento positivo, dall'altro, se il numero è troppo elevato, si rischia di generare overfitting. I neuroni 'spenti' non andranno ad influire nè durante il processo di forwardpropagation, nè durante il processo di backpropagation. Le dimensioni della rete non vengono influenzate da questo strato, cambia unicamente che alcuni neuroni che contribuiscono al processo di apprendimento vengono settati a 0 e i neuroni spenti sono diversi epoca per epoca.

- **Concatenation:** raramente viene usato, in quanto, se la rete è grande e non presenta un'architettura al limite della perfezione, la presenza di questi layer genera grossi problemi in termini di quantità di dati da elaborare. La funzione di questi layer comunque è abbastanza semplice: prendono in ingresso un certo numero di blob[24] e producono in output una blob unica avente come dimensioni la somma delle dimensioni delle blob di ingresso.

- **Fully-Connected:** all'interno di questo strato tutti i neuroni sono connessi a tutti i neuroni dello strato precedente, più precisamente, sono connessi alle loro funzioni di attivazione.

  L'unico parametro che si può impostare all'interno di questo strato è il numero K di neuroni che lo costituiscono. Il lavoro che svolge è connettere tutti i K neuroni agli input e calcolare il coefficiente di attivazione di ciascuno dei K neuroni. L'output che lo strato genera è un vettore 1 x 1 x K. Il fatto che dopo l'applicazione di uno strato FC[25] si passi da un volume a un vettore in una singola dimensione, fa intuire che dopo questo layer non è possibile implementare uno strato convoluzionale. Lo scopo principale di questa parte della rete è di raggruppare tutte le informazioni ottenute sino a quel momento ed esprimerle con un numero.

---

[23] Non significa però che esse siano più rilevanti. L'uso di una modalità piuttosto che un'altra dipende dallo scopo di quel determinato layer che si sta implementando.

[24] Array standard che costituiscono l'interfaccia di memoria usata dal framework per organizzare i dati

[25] Fully Connected

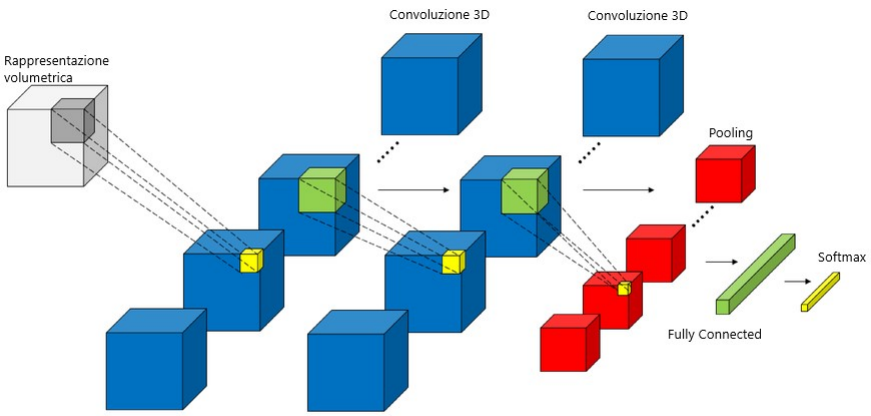


Figura 2.7: Evoluzione della rappresentazione delle informazioni per una rete neurale.

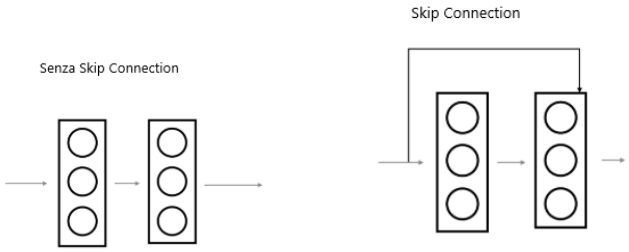
- **Classificazione:** solitamente questo strato si compone di un numero J di strati FC messi in cascata, con un numero di neuroni via via decrescente, sino ad arrivare al numero W, che è il numero di possibili classi d'output. Per fare un esempio: dovendo fare una classificazione uomo/donna, il numero W di possibili classi è 2. I valori dell'ultimo strato FC vengono dati come input a una determinata funzione, che è scelta dall'utente (in *Fig. 2.7* è stata scelta Softmax), che si occuperà di fornire la classe dell'oggetto mandato come input.

Attraverso il sequenziamento di tutte queste operazioni, ognuna svolta all'interno del proprio layer, si crea l'architettura della rete ed essa estrae le informazioni necessarie.
Il processo di minimizzazione della loss function è di tipo iterativo e viene regolato da dei modelli di ottimizzazione. Esistono due tipologie di ottimizzazione: quelle del primo ordine, rappresentati dal metodo del gradiente, e quelle del secondo ordine. Normalmente è più che sufficiente un ottimizzatore del primo ordine.
Al giorno d'oggi esistono diversi modelli che rappresentano lo stato dell'arte di cui, tra i più noti si evidenziano: ResNet e InceptionV3.

2.2.1 ResNet

ResNet è la prima rete che introduce l'operazione di skip connection[26]. La *Fig. 2.8a* mostra il normale flow dei dati, mentre la *Fig. 2.8b* mostra il processo di skip connection.

[26]Saltare il o i layers successivi.



(a) *Processo senza Skip Connection*

(b) *Processo con Skip Connection*

Figura 2.8: Differenze nel flow dei dati.

Per essere precisi; l'input si sdoppia e, una parte segue il normale sequenziamento dei layers, mentre l'altro si va a sommare all'output dell'ultimo layer saltato. È stato dimostrato [13] che l'operazione di somma, se posizionata prima del blocco di ReLu[27], produce risultati migliori. Affinchè questa operazione sia possibile è importante che il processo di batch normalization venga svolto in modo consono; se la matrice d'output del layer saltato presenta dimensioni diverse dalla matrice che si vuole sommare, l'operazione non è possibile. Nel caso in cui non si riesca ad ottenere le stesse dimensioni della matrice d'output è possibile fare passare l'input sdoppiato attraverso un layer convoluzionale separato dalla rete, come mostrato in *Fig. 2.9*

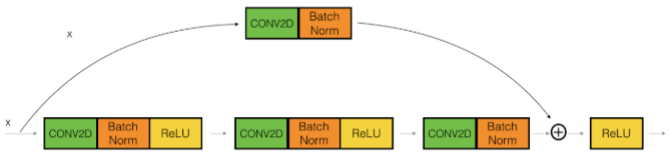


Figura 2.9

Il meccanismo di skip connection è nato come risposta al fatto che il processo di backpropagation non è in grado di trovare una soluzione 'semplice' a problemi

[27]Funzione di attivazione definita come positiva lineare per x in ingresso positive, zero altrove.

semplici. Un esempio chiarificatore può essere il seguente: ho una rete neurale e una serie di dati da analizzare $x_1, x_2, \ldots, x_n$. Voglio che questa rete mi dia come output dei dati che siano uguali agli input. Il passaggio matematico logico per un umano sarebbe quello di usare la funzione identità $f(x) = x$ ogni qual volta si deve moltiplicare l'input. Si è dimostrato che il processo di backpropagation non è in grado di trovare una soluzione così semplice; trova una soluzione complessissima che adempie allo scopo ma ciò si traduce in potenza computazionale persa e scarsa efficienza.
L'intuizione che gli sviluppatori di ResNet hanno avuto è stata quella di fornire a determinati strati non solo i dati elaborati, che potrebbero aver perso senso rispetto al dato originale, ma anche l'input stesso, così da creare una più forte correlazione tra i dati che vengono analizzati ed elaborati e l'input.
Questo modello, grazie alla sua particolare architettura, segue 5 fasi. In figura 2.10 è mostrato una versione di ResNet che prende il nome di Resnet34; il numero indica il numero di layer presenti. Il modello più grande è quello con 152 layer[28]
È inoltre possibile osservare nel dettaglio come venga usato il processo di skip connection. In questa architettura i layer che predominano sono quelli di convoluzione con un kernel 3x3, di sottocampionamento con stride [29] pari a 2. Il flusso dei dati termina con un average pooling globale e uno strato totalmente connesso con 1000 features.
Questa rete è stata comparata con altre in diversi lavori, uno di questi lavori è [14]; le prestazioni vengono confrontate con quelle di VGG16[30] e con un modello creato da loro. Gli step che sono stati seguiti sono stati:

- Analisi del dataset.
- Preprocessing
- Transfer Learning
- Addestramento e valutazione delle prestazioni.

Ovviamente, tutti questi step sono stati ripetuti nella stessa modalità per tutte le reti confrontate. Il dataset scelto è l'HAM10000, che presentava come problematica principale quella di presentare più del 50% di foto di nevi melanocitici; più di metà del dataset era costituito da una sola classe di indagine. Tra i passaggi svolti nel preprocessing quello fondamentale è stato di Data augmentation, per cercare di

[28]Si ricorda che aumentare in maniera considerevole il numero di layer non è sempre un'operazione positiva.

[29]Numero di pixel di quanto il kernel trasli sull'immagine analizzata.

[30]Un'altra DNN.

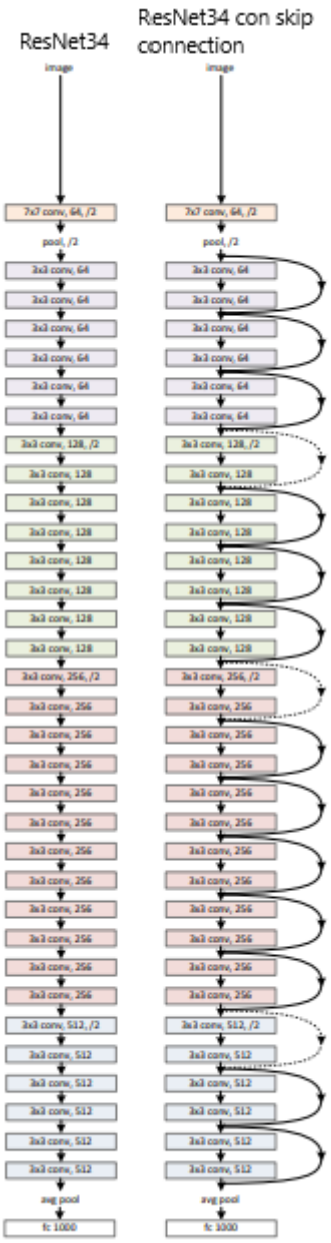


Figura 2.10: Architettura ResNet34 e ResNet34 con skip connections

aumentare la grandezza del dataset e per cercare di bilanciare le percentuali di distribuzione delle classi.

Il transfer learning è il processo attraverso il quale viene importata una rete esterna, già allenata su un diverso dataset, e riallenata sul dataset che si sta usando; la rete,

spesso e volentieri, subisce anche delle piccole modifiche ad hoc per adattarla agli scopi. Le modifiche in questione possono essere la rimozione dei primi layer, degli ultimi o di entrambi: ciò è dovuto al fatto che se il dataset di partenza, su cui si è allenata la rete importata, presenta una diverso numero di classi, si creerebbe un'inconsistenza fra la forma dei dati e la forma di input e output della rete. Spesso si segue questa strada per risparmiare tempo computazionale, in quanto i pesi dei vari neuroni non sono inizializzati a zero, e, soprattutto perchè non è sempre possibile avere un dataset che sia ben bilanciato, e con un numero sufficiente di esempi per classe; serve, ed è essenziale, che i concetti appresi in prima istanza dalla rete siano simili a quelli che deve continuare ad apprendere.
È possibile fare transfer learning in due modalità differenti: una con *fine tuning*[31] e una senza; alle volte bloccare la modifica dei pesi della rete importata può essere controproducente. Il punto fondamentale è che attraverso il processo di transfer learning si va a sommare la conoscenza dell'allenamento alle conoscenze, precedentemente acquisite, da un secondo classificatore.
Le classi di indagine sono state le stesse sette presentate come classi di analisi nel sezione **1.2 Lesioni cutanee**, e i risultati sono riassunti in *Tab. 2.7*.

Tabella 2.7: Risultati ResNet e VGG16.

Classificatore	Accuracy
ResNet	0.905
VGG16	0.780

2.2.2 Inception

Inception è l'architettura della DNN sviluppata da GoogLeNet, sviluppata con l'intento di poter usare queste architetture anche in condizioni 'limite': immagini di indagine notevolmente ridotte, un considerevole numero in meno di parametri rispetto alle reti concorrenti, una maggiore velocità di elaborazione etc.
La *Fig. 2.11* mostra l'architettura della rete in modo semplificato, i vari layer usati, le dimensioni dell'input e quelle dell'output.
Un aspetto negativo di questa architettura è rappresentato dalla complessità di fare delle modifiche e personalizzazioni che non causino gravi perdite a livello di efficienza. I creatori di Inception hanno seguito determinati obbiettivi[15] per ottimizzarla:

[31] Il fine tuning è l'atto attraverso il quale si specifica che i pesi della rete importata possono essere modificabili nel nuovo addestramento.

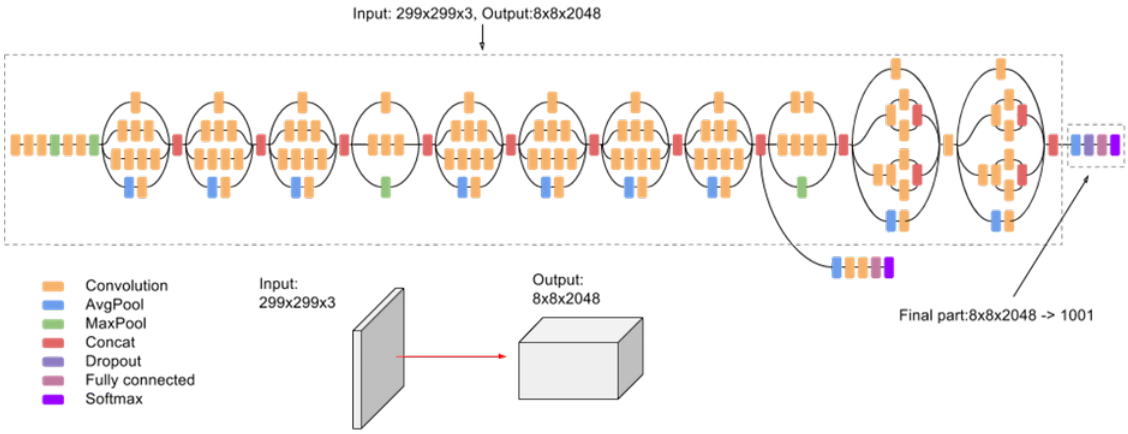


Figura 2.11: Architettura di Inception V3

- Evitare l'uso di layer che potrebbero rallentare la rete; evitando di creare colli di bottiglia. Un collo di bottiglia rappresenta un layer che, presentando un numero di neuroni notevolmente inferiore rispetto ai precedenti strati, rallenta il flusso di informazioni per poter permettere l'elaborazione, da parte di pochi neuroni, di una considerevole quantità di dati.

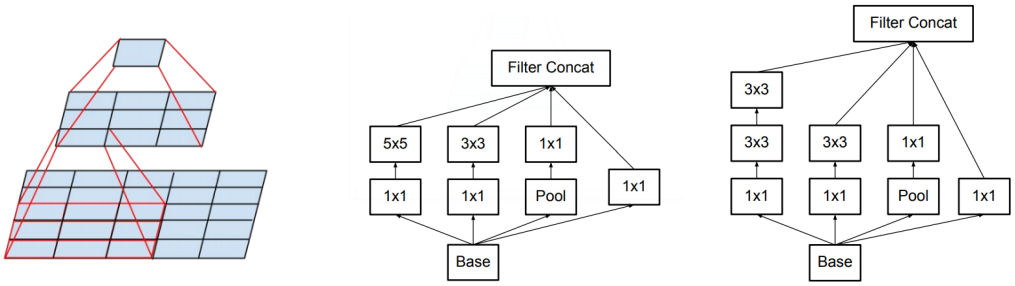
  Hanno cercato di evitarli soprattutto all'inizio del network in quanto, una rete di tipo feedforward, può essere rappresentata da un grafico aciclico che parte dall'inizio sino al classificatore.

- Avere una rappresentazione dimensionale del dato maggiore rende più facile l'elaborazione del dato stesso da parte della rete. Aumentando quindi il numero di attivazioni per strato è stato possibile ottenere delle features con un grado di indipendenza maggiore; il che si traduce in un addestramento più rapido.

- Si può creare un'efficace aggregazione spaziale attraverso una riduzione dimensionale se fatto per layer consecutivi, per esempio: prima di procedere con una convoluzione spaziale più grande (5x5), è possibile ridurre le dimensioni dell'input senza che questa azione generi una perdita di informazioni. Si è ipotizzato che la ragione di ciò sia la forte correlazione tra i risultati dei layer adiacenti, se gli output sono usati in un contesto di aggregazione spaziale. Anche questo processo produce un apprendimento più rapido.

- Bilanciare la larghezza e la profondità del network creato: si possono ottenere delle prestazioni notevoli partendo dal solo bilanciamento del numero di filtri per stage e la profondità totale del network. Aumentare entrambi, produce da un lato risultati di qualità superiore, dall'altro, invece, per riuscire a creare

questo aumento è necessario che sia fatto in modo proporzionale e che si sviluppino un maggior numero di operazioni in parallelo. Tale fatto si traduce in un elevato costo a livello computazionale, quindi si è tenuto conto di questo fattore per non creare un sovraccarico computazionale pur tenendo prestazioni ottimali.



(a) *Griglia del filtro 5x5.* **(b)** *Struttura a blocchi.* **(c)** *Sostituzione della vecchia struttura con filtri più piccoli.*

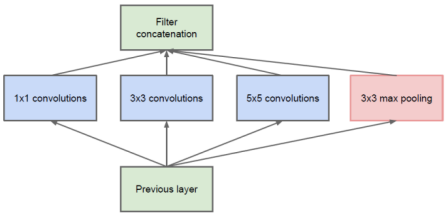
Figura 2.12: Riduzione della dimensione dei filtri.

La gran parte dei risultati ottenuti è stato grazie alla riduzione di dimensionalità; operazione che può essere vista come fattorizzazione della convoluzione.
Tanto più grande è il filtro di un layer convoluzionale, tante più informazioni è possibile estrarre dalla convoluzione. In particolare, poter riassumere le informazioni di una superficie più grande permette di mettere in relazione più dati e trovare pattern in pixel distanti fra di loro.
Il rovescio della medaglia di un filtro di grande dimensioni, è il numero di calcoli necessario da applicare durante l'addestramento: un filtro 5x5 ha 25 pesi da apprendere, un filtro 3x3 ne ha 9; infatti 25/9 mostra come una filtro 5x5 è 2.7 volte più grande di un 3x3.
Il team di inception ha intuito che due layer convoluzionali in serie con filtri di dimensione 3x3 è capace di cogliere relazione nei dati in maniera simile ad un filtro 5x5 ma ad un costo inferiore ad esso. Infatti, se si guarda attentamente l'immagine *2.12a* si vede che in realtà ogni output del filtro 5x5 può essere visto come un piccolo network totalmente connesso che scorre su gli input per 5x5 pixel. Sfruttando l'invarianza di traslazione, il filtro 5x5 totalmente connesso è stato sostituito da due strati convoluzionali più piccoli; i due strati sono filtri 3x3 e, l'output del secondo filtro, presenta un ulteriore strato totalmente connesso (*Fig. 2.12c*).
È stato poi dimostrato che questo ragionamento perde il senso se appliccato in maniera iterativa; dividere anche il filtro 3x3 in altri due filtri 3x1 e 1x3, non avrebbe senso poichè si aumenta la profondità della rete per un risparmio di appena

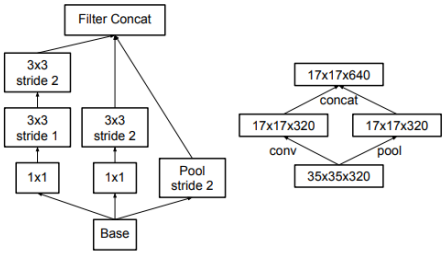
l'11% del costo computazionale rispetto al filtro 3x3. Inoltre, è stato riscontrato che questo ragionamento produce buoni risultati se non viene applicato negli stadi iniziali del network e se applicato a una griglia di medie dimensioni m x m (con $12 < m < 20$).
Una seconda motivazione che giustifica il suo ingresso nello stato dell'arte è stato il parallelizzare convoluzioni con kernel di diverse dimensioni. L'aggiunta di più branch convoluzionali in parallelo è dovuto al fatto che una singola convoluzione, per quanto accuratamente siano stati studiati i suoi parametri, non è sempre in grado di catturare tutte le features utili alla classificazione. Allargare lo spettro d'indagine allarga le possibilità di catturarle.

Figura 2.13

(a) *Esempio di un layer convoluzionale di Inception naive.*



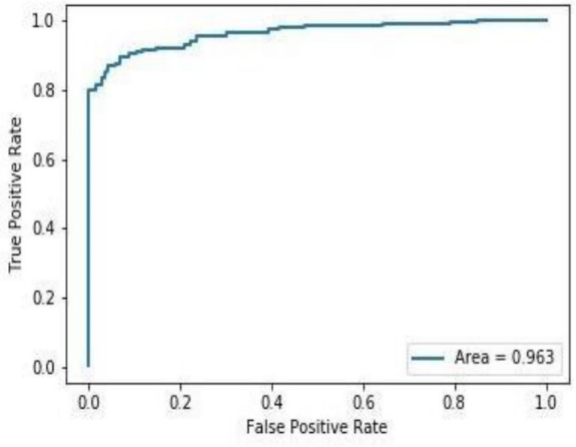
(b) *Struttura a blocchi.*



In *Fig. 2.13a* è possibile osservare la struttura, che prevede tre stadi di convoluzioni parallele affiancati da un layer di pooling. Questo layer messo in parallelo svolge un ruolo fondamentale, in quanto l'input ha dimensioni dimezzate come si vede da *Fig. 2.13b*. Questa struttura così sviluppata presenta un basso costo computazionale e serve ad evitare i colli di bottiglia e rallentare il flow dei dati.
Nel lavoro svolto da Gavrilov *et all.*[16] è stato usato Inception V3 con lo scopo di classificare le lesioni della pelle. Le immagini che sono state usate hanno dimensioni 700x700, quindi il primo loro step è stato quello di rimuovere i layer superiori della rete, in quanto non è possibile cambiare la forma dell'input dato alla rete. La motivazione della rimozione dei layer è che il modello è stato preso con il transfer learning già fatto, gli input con cui il modello era stato addestrato avevano dimensioni di 299x299.
Il secondo step svolto è stato quello di data augmentation; hanno usato il dataset ISIC e, grazie al processo di data augmentation, hanno addestrato i modelli su un milione di esempi. Per aumentare la percentuale di accuratezza sono stati addestrati cinque network con la stessa architettura, ma con dei pesi iniziali differenti. Dopo l'allenamento sono stati combinati in un unico modello assemblato che è stato testato per verificarne il corretto funzionamento.

A differenza di tanti altri lavori, sono state prese in analisi unicamente quattro classi di lesioni; nevi melanocitici, melanomi, cheratosi seborretiche e la classe "altre lesioni". L'AUC-ROC riportato nel paper è mostrato in *Fig. 2.14*

Figura 2.14: AUC

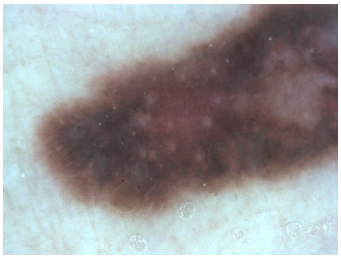


Viene riportato che il modello possiede un'accuracy del 91%, valore molto alto, ma senza avere i valori di precisione e r-call perde di significato in quanto il dataset usato è fortemente sbilanciato verso la classe nevi melanocitici. Inoltre, nel considerare unicamente quattro classi di lesioni anzichè tutte e sette, si è introdotto un bias all'interno del dataset; cheratosi attiniche, dermatofibromi e lesioni vascolari non dovrebbero essere accomunate in una sola classe. Come prima cosa hanno effetti e pesi patologici diversi, in secondo luogo non è stato implementato nessun metodo di apprendimento per esclusione (o è questo o allora deve essere quest'altro).

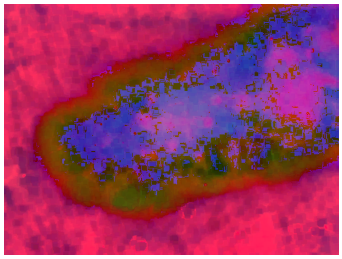
Capitolo 3

Materiali e metodi

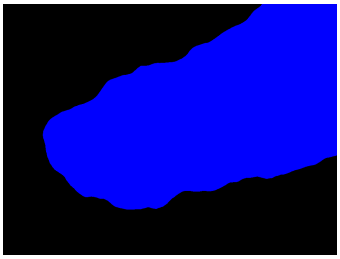
Il focus del presente lavoro è rappresentato dall'implementazione di un software capace di classificare le lesioni cutanee in sette classi di analisi, attraverso l'accesso remoto ad una macchina virtuale.
A seguito di un attento studio del problema e delle soluzione più significative fin qui adottate, l'approccio iniziale è stato di provare ad usare alcune tecniche manuali di estrazione di features, individuando tra quelle più significative: l'area della lesione, l'uniformità di colore e la presenza di simmetrie.
Dopo alcuni tentativi, insieme a **TESI srl** è stato valutato di abbandonare questa procedura di implementazione in quanto palesemente diseconomica in termini di potenza di calcolo e di tempo richiesti per l'elaborazione dei diversi risultati ai quali, tra l'altro, si sarebbero dovuti aggiungere i costi computazionali e temporali necessari per lo studio delle correlazioni tra tutte le features, al fine di scegliere quelle maggiormente scorrelate tra loro e con più elevato contenuto informativo per l'implementazione del futuro classificatore.
Per quanto interrotta, la parte di codice fin lì scritta, riusciva a trovare l'area della lesione in maniera soddisfacente, come mostrato in *Fig. 3.1c*



(a) *Immagine originale.*



(b) *Trasposizione in HSV.*



(c) *Maschera di segmetazione.*

Figura 3.1: Processo di ricerca e calcolo dell'area della lesione.

A livello concettuale, il processo seguito è stato relativamente semplice: inizialmente, sono state lanciate due operazioni di blurring sequenziali per filtrare l'eventuale rumore, la presenza di peli o altri elementi di disturbo. Essendo elementi piccoli, rispetto all'immagine si è scelto un kernel di 5x5, affinchè venissero eliminati gli elementi di disturbo e modificati il meno possibile gli altri elementi. Successivamente, è stato cambiato lo spazio colore di rappresentazione dell'immagine 3.1b, così da rendere i bordi di tutte le lesioni del dataset individuabili in maniera più efficiente. A valle di questa operazione, è stato possibile calcolare la prima delle features, cioè l'area della lesione e, successivamente, si è ripetutamente provato ad implementare il codice per il calcolo delle simmetrie. In assenza delle limitazioni computazionali e temporali sopra evidenziate, il processo usato per questo particolare calcolo, sarebbe stato:

- Cambiare lo spazio di colore in YUV.
- Proiettare l'immagine lungo l'asse X e lungo l'asse Y.
- Calcolo dell'istogramma delle intensità luminose.
- Calcolo della correlazione tra l'istogramma lungo X e lungo Y:
  - Se superiore a una certa percentuale scelta arbitrariamente all' 80% allora era presente un asse di simmetria.
  - Se inferiore, non sono presenti assi di simmetria.
- Ripetere gli step precedenti ruotando, di volta in volta, l'immagine di 2 gradi sino a 360 gradi.
- Evidenziare gli assi di simmetria principali.

La figura 3.2 mostra il processo appena descritto.
Per trovare il parametro "uniformità di colore" si sarebbero invece adoperati i gradienti al fine di evidenziare l'omogeneità o meno della colorazione e, nel caso, di quanto differissero due zone adiacenti. Questo approccio sarebbe stato applicato all'immagine originale più la maschera, così da non essere influenzata da eventuali fluttuazioni ai bordi.
Ciò detto, la seconda strada seguita è stata quella di creare una rete neurale capace di svolgere in automatico tutti i necessari compiti. In tal senso, fin da subito è apparso chiaro che, in questo caso, il problema sarebbe stato rappresentato dalla creazione di una complessa architettura capace di estrarre tutte le features necessarie al fine di ottenere elevate prestazioni.

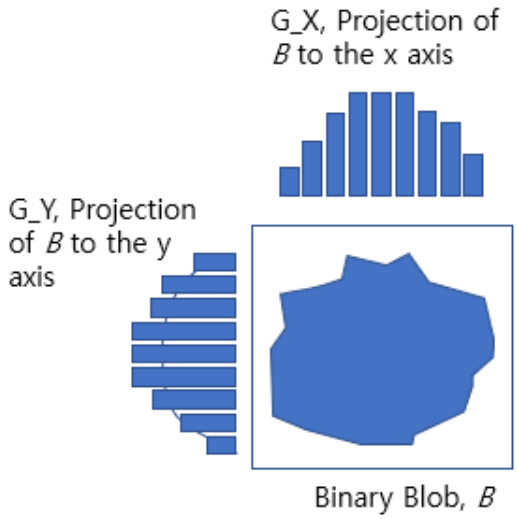


Figura 3.2: Processo per trovare le simmetrie.

Questo secondo approccio prevedeva l'uso di un'architettura a due branch di analsi, come mostrato in *Fig. 3.3*.

Gli input corrispondenti al ramo di destra sarebbero state le features categoriche, quali: Sede, Età e la correlazione tra le altre features significative. Per poter rendere possibile questo processo, tutte le features appenena elencate sono state trasformate, per l'appunto, in categoriche; features nominali, quindi stringhe, che possono essere mappate all'interno di un vettore (attraverso una seconda procedura) dove, ogni bit del corrispondente vettore, è associato a una delle possibili classi della feature scelta.

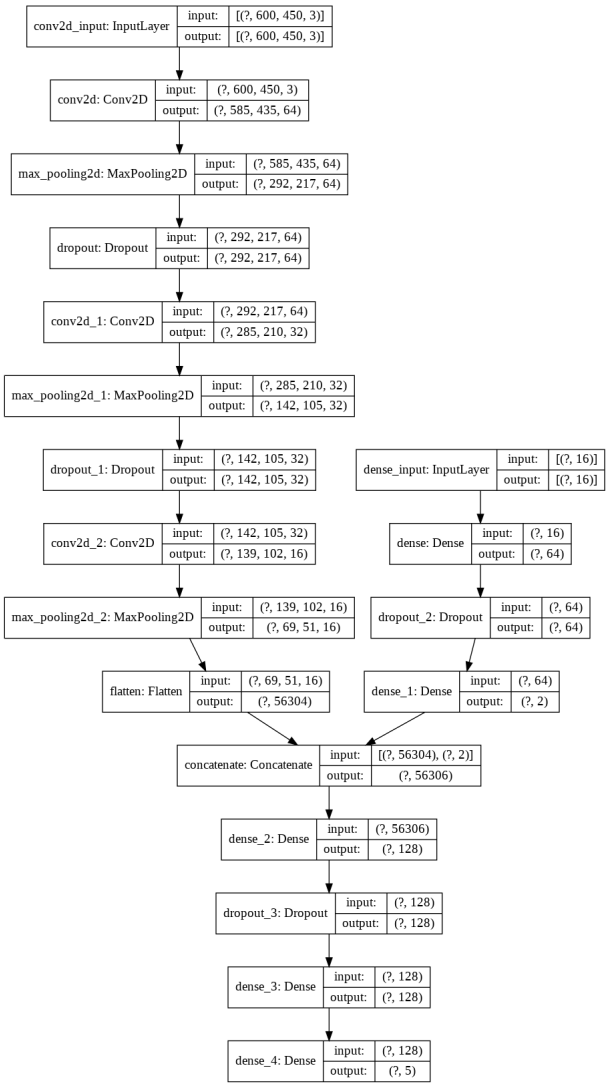
L'input del ramo di sinistra invece sarebbe stata la sola immagine.

La scelta di separare gli input, pertanto, di trattarli come fossero scorrelati, implica la creazione di due rami separati nella rete. Affinchè la rete riesca a trovare una relazione indipendente che leghi tutti gli input all'output, la classe desiderata, è necessario che i due rami si ricongiungano e che l'analisi delle features, dapprima totalmente separate, venga fatta lungo un unico ramo.

L'analisi delle features separate va poi a congiungersi con quella della sola immagine, che subisce concettualmente la stessa analisi. Dopo aver congiunto i due input, la rete cerca di trovare delle relazioni che riescano a 'legare' quelle trovate per la sola immagine, e quelle per le features categoriche. Questo processo dovrebbe massimizzare quindi le possibilità di apprendimento della rete, in quanto vengono analizzati anche degli input non deducibili dalle sole immagini deramtoscopiche.

È possibile vedere dalla *Fig. 3.3* che le dimensioni usate per le immagini in questa

Figura 3.3: Architettura a due branch.



architettura sono pari a 600x450.

Nonostante l'uso di due branch e di immagini di grandi dimensioni, l'architettura appena sviluppata si è rivelata estremamente 'povera' rispetto alle reali necessità operazionali necessarie per analizzare un'immagine in modo consono.

Per questo motivo, si è deciso di fare Transfert Learning e di adottare l'architettura di InceptionV3 preliminarmente addestrata sul dataset di ImageNet.

Come di seguito evidenziato, questo approccio in fase di implementazione si è rivelato performante in termini computazionali e temporali.

3.1 Analisi del dataset

Il primo problema che è stato affrontato e risolto, è la creazione del dataset. Inizialmente si pensava che potesse essere fornita dalla TESI srl ma, per problematiche non dipendenti da quest'ultima, ciò non è stato possibile. Pertanto, mi è stato inizialmente fornito il dataset ISIC dove, al suo interno, sono presenti sia le immagini dermatoscopiche di lesione della pelle, che i metadati classificati come in *Tab. 3.1*, per un totale di oltre quindici mila immagini.

Tabella 3.1: Esempio dei metadati.

ID	Sesso	Età	Dimensioni immagine	Classe	Sede	...
0000001	M	31	1224x768	Nevo	Torso	...
0000002	M	46	1220x770	Melanoma	Schiena	...
0000156	...	...	...	...	...	...

Dopo aver studiato sia la disposizione dei metadati e quali tra loro potessero rappresentare interesse a livello statistico, sia la classe della lesione stessa (ad esempio: riuscire a creare un dato che rappresentasse la correlazione tra età e sede della lesione), ho riscontrato numerosi problemi all'interno del dataset di cui i principali sono:

- Dimensione variabile delle immagini.
- Sovrabbondanza della classe Nevo rispetto a tutte le altre classi.
- Mancanza di dati.
- Presenza di classi con un numero di elementi inconsistenti.

Una parte dei problemi sopracitati sono stati risolti tramite il merge[32] con un nuovo dataset, l'HAM10000, un'altra parte invece durante la fase di preprocessing.

[32] Unione.

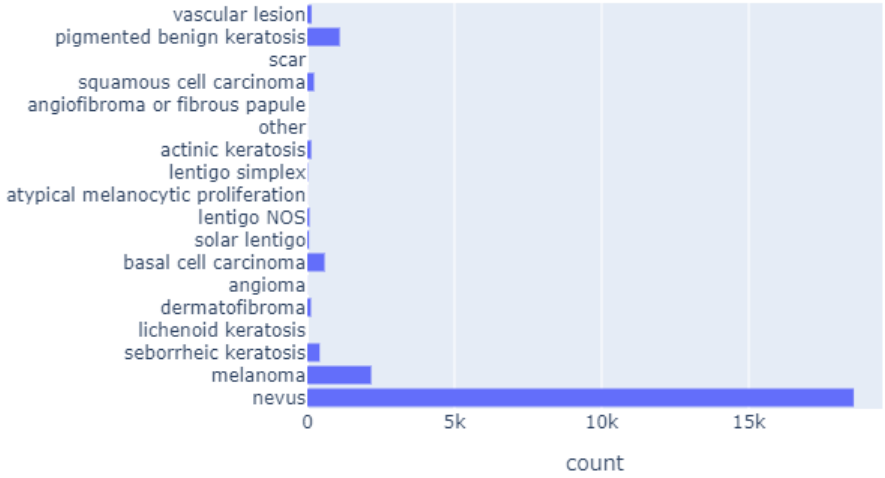
3.1.1 Dimensione variabile delle immagini

Avere una dimensione variabile delle immagini potrebbe apparire una tematica di poco conto o, addirittura, non esserlo per nulla; viceversa, rappresenta un problema di centrale importanza.
Infatti, se le dimensioni delle immagini variano, variano anche le dimensioni dell'input e, una rete neurale del tipo Inception non è stata implementa per poter processare dati di input con dimensioni variabili; ciò ha implicato la necessità di applicare un resizing a tutte le immagini.
Dopo aver analizzato i diversi lavori citati in precedenza si è notato che generalmente le dimensioni utilizzate non superavano i 450x300 pixel che, comunque, rappresentano limiti molto grandi; in un solo caso sono state usate dimensioni di 700x700.
Ovviamente, poter analizzare una immagine grande è un vantaggio in quanto si riesce ad estrarre più informazioni, ad avere maggior precisione nell'analisi e generalizzare meglio. Di contro, l'analisi di una immagine grande, comporta un elevato consumo di memoria: aumentando il numero di pixel analizzati aumenta lo spazio RAM necessario per poter contenere i risultati dell'analisi. Ma avere più memoria RAM si tradurrebbe in un aumento significativo dei costi economici rendendo, di fatto, tale processo non più conveniente.
Viceversa, l'analisi di immagini più piccole, non solo contiene lo spazio in memoria ma consente anche un'analisi dei dati più veloce, così risparmiando tempo e costi di ampliamento della memoria RAM. Considerato inoltre l'elevato numero di immagini da analizzare è stato valutato, quale compromesso ottimale tra le variabili in gioco, un resizing che mantenesse dimensioni intermedie di 350x250 pixel.
Per quanto riguarda il flow del processo eseguito, il resizing delle immagini è stato implementato durante la creazione del nuovo dataset, quindi durante il merge dei due dataset usati.

3.1.2 Sovrabbondanza della classe Nevo

All'interno del dataset preso in analisi, la classe che rappresentava i nevi melanociti benigni era preponderante. In *Fig. 3.4* si vede come questa sola classe sia composta da quasi ventimila immagini mentre, la seconda classe più popolosa ne propone meno di duemila.
Facendo un veloce ragionamento statistico e probabilistico prescindendo dall'immagine, chiunque classifichi sempre nevo, otterrebbe comunque risultati buoni considerato che, appunto, più del 90% delle immagini sono appartenenti alla classe nevi.
Al fine di ovviare a tale evidente sbilanciamento del dataset, inizialmente si è pensato di assumere un numero massimo di esempi della classe nevo, pari a cinquemila, condizione che, per quanto non scartata, è stata successivamente implementata in modo leggermente diverso, come più avanti descritto.

Figura 3.4: Distribuzione dataset set sbilanciata.

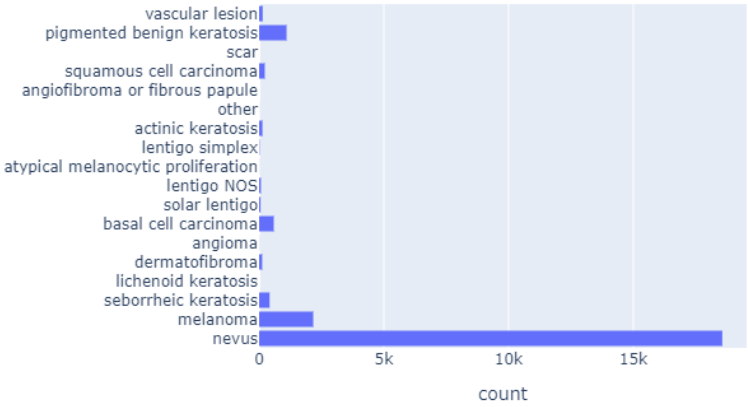


3.1.3 Mancanza di dati

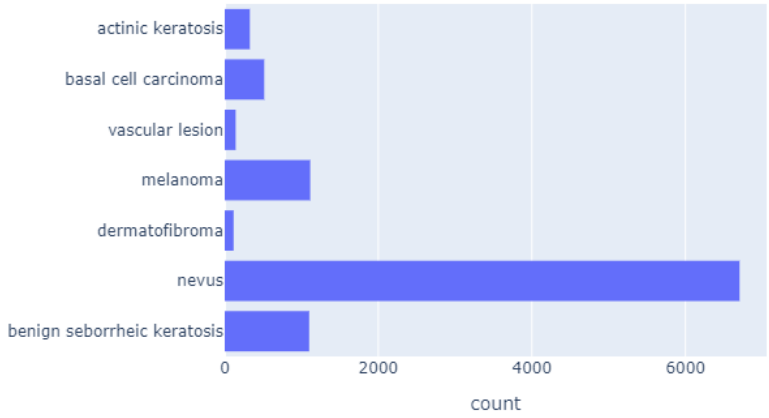
Per quanto a livello teorico il dataset sia molto grande e ben dettagliato, nella pratica è stata riscontrata l'assenza di tanti dati, condizione che ha generato un serio problema di analisi delle classi.
Ad esempio, spesso erano assenti i dati relativi alla sede della lesione cutanea, così come il sesso del soggetto. Ne conseguiva un problema di analisi rappresentato dalla necessità di dover eseguire un controllo incrociato sulle immagini presenti nel dataset e i dati delle classi presenti per poi, successivamente, eliminare tutte le immagini dove non erano presenti le relative informazioni, però, questa procedura riduceva sensibilmente il dataset di analisi, rendendolo non idoneo per addestrare la rete neurale.
Se, in alternativa, si fosse provato a fare un'analisi delle classi, ad esempio la correlazione tra la sede della lesione cutanea e l'età del paziente, il corrispondente parametro calcolato non si sarebbe potuto usare in quanto non sempre presente.

3.1.4 Presenza di classi inconsistenti

Il dataset ISIC, di dominio pubblico, per quanto vasto e con tante informazioni al suo interno, ha come limite principale quello di presentare troppe classi non sempre ben rappresentate attraverso un adeguato numero di esempi, come mostrato in *Fig. 3.5a*. Un esempio, è quello dei Dermatofibromi che presentano un numero di esempi inferiore a 250/300. In senso assoluto non sarebbero pochi ma, se paragonati con i 20.000 della classe Nevi, generano di fatto una classe inconsistente, ancor più poi, considerata la necessità di dover dividere i 300 esempi nei tre diversi sottogruppi



(a) *Distribuzione delle classi nel dataset ISIC.*

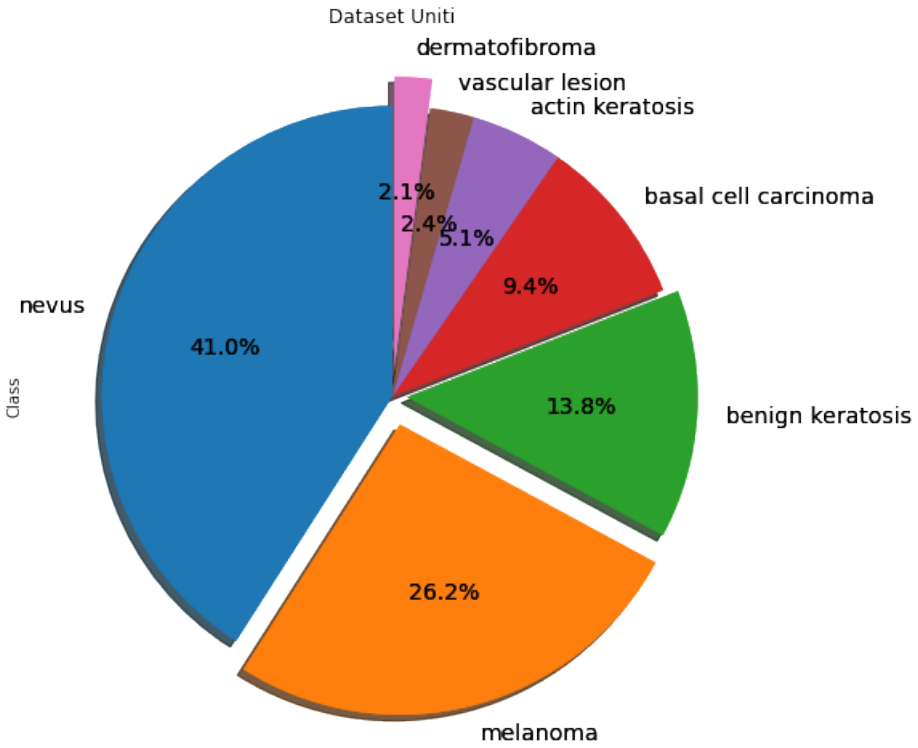


(b) *Distribuzione delle classi nel dataset HAM10000.*

Figura 3.5: Confronto tra le diverse distribuzioni.

del training, del validation e del test.
Cercare una soluzione a quest'ultimo problema ha rappresentato una delle principali motivazioni a supporto della decisione di unire questo dataset con un secondo: l'HAM10000.
Così operando, le classi meno consistenti quali dermatofibromi, lesioni vascolari, ecc., benché non rappresentative in senso assoluto, sono diventate più consistenti, ovviamente entro limiti ben precisi: non è possibile trovare dei dataset contenenti unicamente dermatofibromi, lesioni vascolari etc.
Il grafico a torta, mostrato in *Fig. 3.6*, mostra esattamente la distribuzione ottenuta delle sette classi con i due dataset uniti.
L'unione dei due dataset comporta delle problematiche di non poco conto: differenti dimensioni delle immagini e dei livelli di luminosità, angolazione delle fotografie,

Figura 3.6: Distribuzione delle sette classi nel dataset.



scelta più o meno dettagliata delle sedi delle lesioni, ecc.

Durante il processo di unione dei due dataset, il primo step eseguito è stato quello di uniformare le classi che non fossero quella indicante la dimensione delle immagini dermatoscopiche (la dimensione di input non ha importanza nell'operatore di resize, dove viene specificata unicamente la dimensione d'output desiderata. Quindi quello è un passaggio svolto in automatico una sola volta.): Sede, Classe, Età ecc. Per quanto riguarda il parametro Classe, si è scelto di uniformare a sette le classi di indagine; corrispondenti alle diverse diagnosi delle lesioni presenti nel dataset HAM10000. Lo stesso procedimento è stato applicato per quanto la Sede; è stata scelta l'indicizzazione più generica: non è possibile dedurre in che porzione specifica del torso sia una lesione, ma è possibile dedurre che se è locata sul pettorale sinistro allora è nel torso. Per quanto riguarda l'Età invece si è scelto di creare delle variabili discrete, dummies, per decadi: [0:10) anni, [10:20) e così via sino a [90:100]. Infine, tutte queste features sopraelencate sono state trasformate in categoriche per una eventuale analisi successiva (non sono state fornite come input alla rete).

Nel dataset ISIC erano presenti delle altre classi che, a livello medico, potevano essere raggruppate all'interno di alcune 'macro classi' come, ad esempio, cheratosi

benigne, etc. Le classi, che potevano essere incorporate all'interno delle 'macro classi', sono state raggruppate al fine di fornire maggiore consistenza alle rimanenti classi, e di evitare di creare un'analisi troppo complessa, che non avrebbe portato risultati positivi.
I problemi invece derivanti da forti differenze dell'esposizione luminosa, angolazioni, etc, sono stati risolti nella fase di preprocessing attraverso il data Augmentation.

3.1.5 Unione dei due dataset

L'ultimo step di analisi dei due dataset è stato quello di fondere i dataset tra loro. Ribadendo che la classe nevo è preponderante in entrambi i dataset, il processo di fusione adottato è stato quello di prendere tutte le sei classi oltre quella dei nevi e, per tale classe, impostare come limite massimo 5000 immagini dermatoscopiche. Le 5000 immagini dermatoscopiche scelte sono state selezionate, attraverso un processo casuale automatico, per metà da quelle fornite dal dataset ISIC e per metà da quelle fornite dal dataset HAM10000.
La creazione è stata facilitata grazie all'uso della libreria open source Pandas. Inoltre l'intero dataset è stato salvato ed esportato in formato pickle, per rendere più gestibile la sua apertura, in quanto, come mostrato nell'esempio, contiene le immagini già aperte. Ciò si traduce in una richiesta di memoria elevata. Per bilanciare tale richiesta, si sono eliminate le informazioni non rilevanti per questo lavoro, come ad esempio: sesso, età, sede della lesione etc.
Tale procedura è stata fatta per cercare di bilanciare il più possibile eventuali differenze, presenti nelle immagini dermatoscopiche all'intero della suddetta classe. Alla fine si è riusciti ad ottenere un dataset che è stato diviso in training, validation e test set come riportato in *Fig. 1.16*

3.2 Preprocessing

La fase di preprocessing è, di fatto, la fase che può influenzare maggiormente i risultati d'apprendimento della rete neurale che si va a creare. In tal senso, si è prestata particolare attenzione a questa fase.
Considerato che il numero totale di immagini ottenute nel nuovo dataset sarebbe comunque potuto risultare insufficiente e, che le immagini provenienti dai due dataset differivano per angolazioni, range di luminosità, etc, è stato deciso che per implementare questa fase nel migliore dei modi possibili, fosse necessario uno step di data augmentation. Il data augmentation è una strategia attraverso la quale si riesce ad accrescere in modo significativo la quantità dei dati disponibili e la loro 'diversità'.
Non sempre i dati che si riescono ad ottenere ed elaborare sono sufficienti a livello numerico e non sempre presentano abbastanza diversità all'interno delle stesse

classi, un esempio potrebbero essere rappresentato da un dataset composto da foto di mele: si hanno tante foto di mele verdi e gialle e qualche mela rossa. Così la nostra rete neurale impara che le caratteristiche importanti sono la forma e il colore, entro certi limiti. Dandogli come input una mela rossa è probabile che non la riconosca. Le possibili forme in cui una stessa classe di input si può presentare, rappresentano la diversità.
Nel nostro caso non sono stati modificati i colori, in quanto specifici marker di possibile presenza tumorale, ma sono state fatte altre modifiche, quali:

- Feature scaling delle immagini.
- Cambio dei range di esposizione.
- Rotazioni.
- Zoom in e zoom out.

Tutti i processi sopra elencati, vista la moltitudine di dati, non sarebbero potuti essere fatti a mano, senza che ciò comportasse una grande perdita a livello di tempo. La strategia adottata affinché il processo avesse dei tempi ragionevoli è stata quella di creare un generatore di immagini che applicasse queste modifiche alle immagini. Un generatore di immagini è un API[33]. La sua funzionalità è quella di permettere il data augmentation salvando spazio in memoria e renderlo possibile in tempi ragionevoli.

3.2.1 Generatore di immagini

Per quanto riguarda l'API è stato scelto di usarne una pubblica, dove poi bisogna fare il fitting di quest'ultima sui dati presenti nel train, validation e test set. Questo è il primo step, ed è quello fondamentale; durante il fitting del generatore, esso calcola tutti i dati matematici che gli servono per poter effettuare tutte le operazioni che si sono impostate nella riga dei comandi.
Nel lavoro svolto, il generatore di immagini usato è l'API fornita da Keras[34], che è stata usata con due diverse modalità: la prima per quanto riguarda il training set, e l'altra per il validation e il test set.

[33]Application Programming Interface. Un insieme di procedure raggruppate per funzioni, che servono per lo svolgimento di un determinato compito

[34]Libreria open-source per l'implentazione di neural network.

Generatore di immagini e training set Per quanto concerne il training set, la finalità ultima a cui si vuole arrivare attraverso il processo di data augmentation, è quella di generare più esempi di quelli disponibili. Questi esempi dovranno differire in qualcosa dai precedenti, altrimenti sarebbe una copia delle immagini, per cui non si creerebbe un aumento della conoscenza estratta.
Tra le operazioni eseguite per poter aumentare i dati in modo utile, due che assumono un ruolo centrale sono:

- Sottrazione della media.
- Divisione per il valore di deviazione standar.

La prima operazione, di fondamentale importanza, visto che il range degli input raw[35] è molto variabile è quella del *feature scaling*; operazione essenziale nel machine learning, considerato che in assenza di ciò, alcuni algoritmi potrebbero non funzionare o, funzionare in modo non soddisfacente. Il feature scaling serve per normalizzare il range di variabilità degli input.
Le due operazioni sopracitate fanno parte del processo di *Standardizzazione*. La standardizzazione delle features fa in modo che ogni feature, all'interno dell'intero set su cui è applicata, abbia un valore di media pari a zero e varianza unitaria. Considerando x l'immagine di input, quello che viene fatto è:

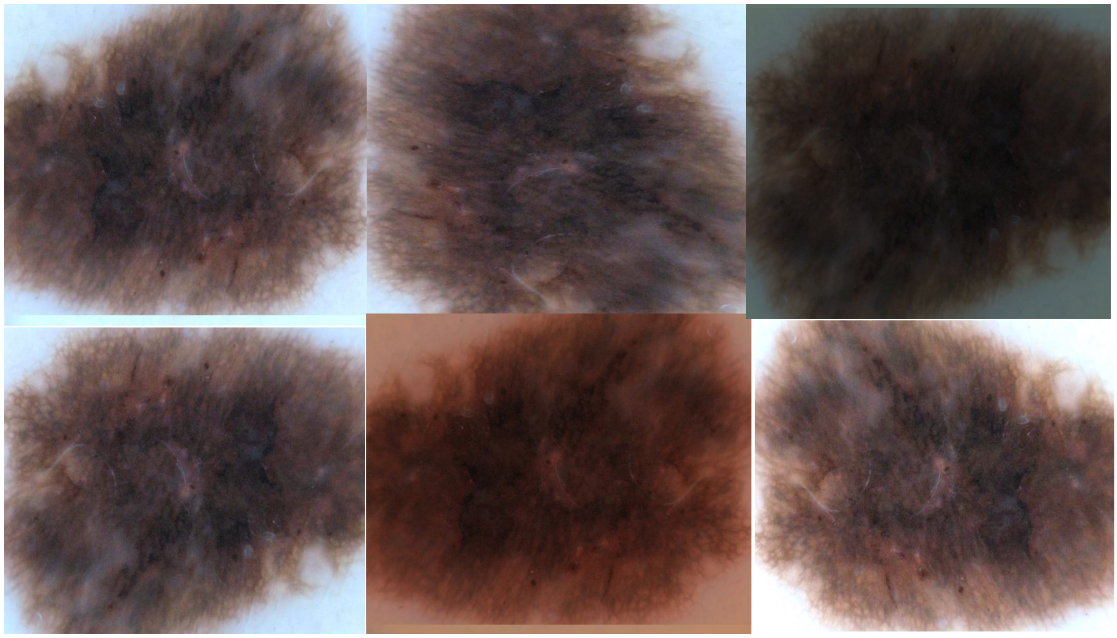
$$x_{normalizzata} = \frac{x - \bar{x}}{\sigma_x}$$

Nella pratica, sottrarre la media dei dati serve per centrare la curva, che descrive la loro distribuzione, sullo zero, mentre la divisione dei dati centrati per la standard deviation ha l'effetto di normalizzare i valori delle features. La ragione fondamentale per cui vengono svolte entrambe le operazioni è che, nel processo di training del network, gli input saranno moltiplicati per dei pesi e, successivamente, si va a sommare il bias; quando questo input viene analizzato tramite la tecnica dei gradienti, se le features che vengono analizzate presentano dei range variabili, nel processo di analisi delle features si creano degli scompensi.
Per fare un esempio pratico: le features da analizzare sono età e reddito. La prima ha un range compreso tra 0 e 100, la seconda ha un range compreso tra 0 e, supponiamo, 100000. Se le due features non fossero standardizzate, ciò che si otterrebbe sarebbe che la sola features del redditto avrebbe un peso 1000 volte maggiore rispetto all'età.

[35]Input che non hanno subito processi di trasformazione.

Pertanto, il processo di standardizzazione, non solo pesa in modo equilibrato tutte le features, ma rende anche nettamente meno sensibile il modello agli outliers[36]
È stato scelto di applicare il processo appena descritto alle singole immagini e non ai diversi batch al fine di evitare la presenza di batch centrati in punti differenti.
La seconda operazione svolta è stata quella del cambio dei range di esposizione. Le immagini provenienti dai due diversi dataset presentavano dei range di esposizione fondamentalmente diversi; le foto non sono state fatte nelle stesse condizioni. Per dare una maggiore uniformità ai dati, si è scelto di cambiare il range di esposizione delle immagini rendendole, in modo casuale, più o meno esposte. Tale operazione è stata implementata grazie a uno degli iperparametri resi disponibili dall'API, che ha permesso di scegliere dei livelli di esposizione compresi tra 0.2 e 1.0.
A livello procedurale, è stato fatto lo stesso per quanto riguarda le operazioni di rotazione e zoom; sono stati selezionati rispettivamente rotazioni verticali e orizzontali e uno zoom dal 70 al 100%.
L'immagine 3.7, mostrata nella pagina seguente, è esplicativa di tutto il processo appena descritto.

Figura 3.7: Risultati delle operazioni fatte dal generatore di immagini.



[36]Singoli valori, non del tutto significativi, che si distanziano fortemente dalla media dei valori della feature.

Generatore di immagini, validation e test set. Per quanto riguarda invece il validation set e il test set, l'unica operazione svolta è stata quella di feature scaling. Visto che questi due set non vengono presi in considerazione per quanto riguarda l'apprendimento, ma, unicamente come verifica dell'apprendimento stesso, non aveva senso che venissero svolte delle operazioni che andassero a modificare le immagini in questione. Invece, era importante svolgere l'operazione di feature scaling, poichè tutte le features apprese durante la fase di addestramento erano normalizzate e centrate. Se in una seconda fase di test, le features trovate presentassero dei valori al di fuori dei range delle features apprese, allora la rete non sarebbe in grado di classificare quelle immagini.

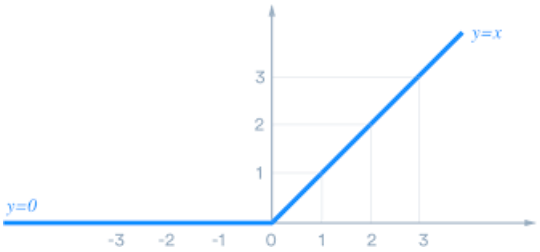
3.3 Architettura del modello finale

L'architettura del modello che, infine, è stata adoperata per gli scopi della tesi, è mostrata nell' **appendice A**. Salta subito all'occhio il riquadro tratteggiato che non ingloba tutta l'architettura: il box serve per dividere la porzione dell'architettura presa, Inception V3, dalle parti aggiunte durante questo lavoro.
Il primo riquadro fuori dal box, è un'aggiunta automatica, derivante dalla decisione di cambiare le dimensioni dell'input, che presenta dimensioni standard di 299x299x3 per l'architettura Inception V3. Affinchè input di dimensioni diverse possano essere elaborate da una rete che presenta una struttura dati già formata, serve il layer aggiuntivo in alto; come già scritto in precedenza, le dimensioni scelte sono di 350x250x3.

3.3.1 Classificatore finale

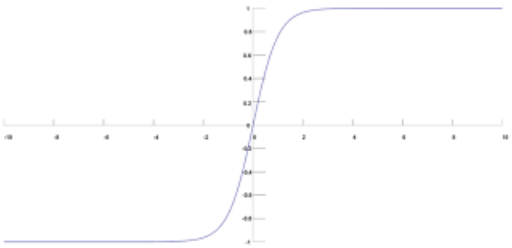
Invece, i rettangoli posti alla fine del modello compreso all'interno del riquadro tratteggiato, rappresentano il classificatore finale che è stato costruito.
Basa il suo funzionamento su tre strati a due passaggi ciascuno: uno stadio di Dropout e uno di Dense. Il primo layer, Dropout, è il più severo, e spegne un numero di neuroni pari al 50%. Questo layer è seguito da un secondo strato, Dense, composto da 128 neuroni con funzione di attivazione ReLu, mostrata in *Fig. 3.8*.
La funzione di ReLu è una delle tante e possibili funzioni di attivazione non lineari. Il vantaggio principale nell'utilizzo di questa funzione è che genera un'attivazione non simultanea dei neuroni. Conseguentemente: se la combinazione lineare degli input è minore di zero, causa la disattivazione del neurone. Il fatto che il gradiente di questa funzione assuma valore unitario per valori maggiori o uguali a zero, e valore nullo per valori della funzione di attivazione minori di zero garantisce un processo efficiente durante il processo di backpropagation; vengono propagati unicamenti i valori non nulli.

Figura 3.8: Funzione di attivazione ReLu (Rectified Linear Unit)



Le funzioni non lineari assumono un ruolo chiave all'interno del processo di apprendimento delle reti neurali, in quanto consentono di trovare trasformazioni non lineari dello spazio degli input. I dati non linearmente classificabili all'inizio, all'interno di questi nuovi spazi, potrebbero essere diventati linearmente classificabili. Se invece si usassero funzioni di attivazione lineari, si avrebbe certamente un guadagno a livello di costi computazionali, ma gli output rimarrebbero nello stesso spazio di rappresentazione di quelli degli input, cioè non linearmente classificabili.
Riprendendo con il classificatore costruito, lo step intermedio è costituito da un secondo Dropout pari al 40% e un secondo Dense, sempre con funzione di attivazione ReLu, ma con meno neuroni: pari a 32.
Il numero di neuroni presenti nello strato di Dense diminuisce poichè essi rappresentano le possibili classi di output. Pertanto, l'ultimo strato di Dense, che sarà quello che fornirà gli output divisi nelle loro classi, avrà solamente sette neuroni, ed è comunque preceduto da un layer di Dropout pari al 30%.
Nell'ultimo layer di Dense è stata usata una funzione di attivazione differente da quella usata nei due precedenti, ed è stata scelta la funzione di *softmax*, mostrata in *Fig. 3.9*

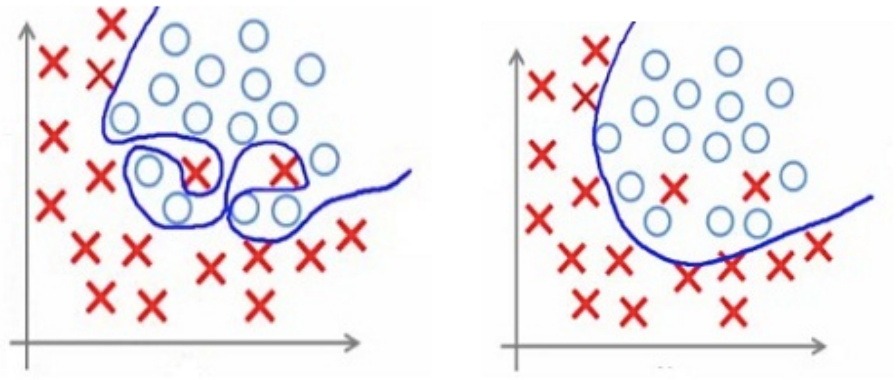
Figura 3.9: Funzione di Softmax.



La funzione di softmax non viene usata all'interno dei layer nascosti, o in layer diversi da quello di classificazione finale, in quanto fornisce le probabilità che l'input processato appartenga alle varie classi di indagine. L'output di tale funzione è la distribuzione delle probabilità divise nelle diverse classi, ciò implica che il risultato di questa funzione sia sempre un numero n tale per cui.

$$0 < n \leq 1$$

In tutti e tre i layer di Dense è stato implementato un regolarizzatore del kernel. Il concetto base dietro la regolarizzazione, è di cercare di evitare l'overfitting dei dati, come mostrato in figura 3.10b.



(a) *Correzione dei pesi senza regolarizzatore e overfitting.*

(b) *Correzione dei pesi con regolarizzatore.*

Figura 3.10: Risultato dell'uso del regolarizzatore.

Esistono tre diversi regolarizzatori; uno del kernel, uno dei bias e uno dell'attività. All'interno di questo lavoro, il regolarizzatore del kernel serve per ridurre la 'complessità' della matrice dei pesi W.
Si ricorda che la forma generale dell'equazione che ogni neurone calcola è del tipo:

$$y = f(\sum_i W_i x_i + b)$$

dove f è la funzione di attivazione, y è l'output, x è l'input, b è il termine di bias e W la matrice dei pesi. Ogni volta che il neurone fornisce un output y, esso viene confrontato con il reale risultato y', si calcola la differenza e attraverso il processo di backpropagation si modificano i pesi W e il bias b per riuscire a migliorare la classificazione.
Il regolarizzatore del kernel serve appunto per andare a cambiare i pesi della matrice W, ma con un criterio ben preciso. Nel caso specifico, il tipo di regolarizzatore usato si chiama L2.
Per poter modificare i pesi durante la backpropagation, il regolarizzatore aggiunge alla funzione di costo[37] un termine di regolarizzazione. Normalmente la funzione di costo si può esprimere come:

[37]Funzione che definisce le performance della rete.

Funzione di costo = Funzione di loss + Termine di regolarizzazione

Che in questo caso si traduce come:

$$Funzione\ di\ costo = Funzione\ di\ loss + \frac{\lambda}{2m} * \sum \parallel w \parallel^2$$

dove il termine λ è l'iperparametro di regolarizzaione, il cui valore è ottimizzato per dare i migliori risultati possibili. Il regolarizzatore L2 è anche noto come regolarizzatore del peso decadente; forza i pesi della matrice W a numeri sempre più piccoli e vicini allo zero. Un assunto del machine learning è che reti neurali con matrici dei pesi con valori più piccoli, portano a modelli più semplici, il che riduce notevolmente la probabilità di overfitting.

3.3.2 Generazione e struttura degli input

Per lo svuluppo della struttura degli input dati in ingresso all'architettura, è stato necessario creare dei batch, la cui creazione, però, non è sufficiente. Avendo come obbiettivo l'automatizione di tutto il processo.
A tale scopo, sono state sfruttate le potenzialità di una funzione offerta dalla prima API che, a livello di sequenziamento del processo, si colloca esattamente in mezzo tra l' API (Image Data Generator) e il modello in questione.
La funzione in questione si chiama Flow From DataFrame; sfrutta i risultati, ottenuti con la libreria Pandas per creare il dataframe, e crea in modo automatico delle sequenze di strutture dati che, nel caso specifico, costituiscono il vero input del modello. In questo modo si è creato un iteratore: una struttura che prende l'insieme dei dati, e divide il loro ingresso in tanti batch casuali, che si accodano uno dietro l'altro, creando il *flow*. Infatti l'input non è una singola immagine dermatoscopica o un batch di esse, l'input è il flow stesso, generato dalla funzione. Un secondo vantaggio è che ogni volta che il modello termina le epoche di addestramento e si riesegue il fitting della funzione, i batch cambiano, non di dimensioni, ma variano gli elementi presenti al loro interno: questo è un utile processo contro l'overfitting dei dati perchè il modello non vede mai la stessa struttura di batch in allenamenti diversi e, vedendo gruppi sempre diversi aiuta il processo di generalizzazione dei concetti e ostacola l'overfitting. I batch non cambiano tra le diverse epoche, cambiano unicamente quando, terminato l'addestramento di ad esempio, 50 epoche, si rifitta il Flow From DataFrame, che crea i nuovi batch, e si continua l'addestramento con altre epoche.
Ovviamente, affinchè tutta questa procedura abbia senso nel contesto in cui è stata applicata, si è dovuto implementare la stessa funzione in due modi diversi: una per il training set, e una per il validation e test set. Avere due flow diversi è essenziale in quanto, a livello di procedura, questa funzione richiama la precedente, dove, si

ricorda, sono implementate operazione diverse a seconda che prendano in ingresso il training set o uno degli altri due set.
Nel Flow From DataFrame i parametri che sono stati implementati sono:

- Il batch size[38].
- La colonna di dati da passare come input.
- La colonna di dati per verificare la correttezza dei risultati della rete.
- La classe dei dati che rappresentavano l'output.

Inoltre, nel caso del Flow per il validation e per il test set è stato valutato opportuno non usare il comando di shuffle[39] dei dati, in quanto, se fosse stata attivata la possibilità di shuffling, durante la fase di test, si sarebbe ottenuto un secondo mescolamento dei dati che non avrebbe permesso di controllare la veridicità dei risultati ottenuti in modo semplice.
È stato scelto il batch size dopo aver fatto alcune prove di addestramento: la grandezza del batch influisce sull'apprendimento della rete neurale. Logicamente, se si aumentano le dimensioni del batch e quindi la rete vede contemporaneamente un gruppo di immagini più grande, avrà meno difficoltà a generalizzare i concetti e trovare le relazioni 'generali' delle immagini analizzate. Viceversa, un batch size più piccolo induce la rete ad apprendere dettagli specifici delle singole immagini, riducendo la probabilità di generalizzare; di conseguenza aumenta la probabilità di overfitting.
Ciò detto, non è possibile ottenere entrambe le cose, indi per cui è stato necessario eseguire un *trade off*[40] tra le due strade per il quale, la dimensione che ho riscontrato portare risultati migliori, è di 32 immagini per batch.
Normalmente, si convertirebbero 'manualmente' le immagini da dare come input in oggetti, i quali sono il risultato di una classe. Tali oggetti presentano una struttura dati adatta ad essere processata come input dalla rete in questione.
Dato che il numero di immagini in questione è elevato, e che il processo di conversione dell'immagine richiede tanto spazio in memoria, tale procedura è stata implementata in modo automatico così da risparmiare lavoro manuale. La scelta di usare il Data Generator, e la funzione di flow per il data augmentation per la creazione sequenziale dei batch di input che presentassero la struttura necessaria si è rivelata ottimale.
Nel caso specifico di analisi, si è inizializzata la classe Image Data Generator con i parametri per la standardizzazione delle immagini; però, per poter consentire a

[38]Dimensione del batch.

[39]Mischiare i dati in modo casuale.

[40]Compromesso.

tale classe di procedere con la standardizzazione è stato necessario passare come parametri le immagini già aperte affinchè il Data Generator potesse calcolare i valori di media e di deviazione standard da sottrarre e per cui dividere le immagini. A monte di tutto ciò, lo step di apertura delle immagini è stato fatto durante la creazione del dataset unito.
Ogni volta che veniva preso un elemento da uno dei due dataset, si leggeva la struttura dell'immagine e si salvava all'interno di una nuova casella creata da me nel dataset. Così facendo è stato creato il dataset che presentava le strutture dati di tutte le immagini usate automatizzando il processo.
Trattandosi di un problema di classificazione, le classi sono state codificate in *one hot vectors*[41]. La rete stessa è stata pensata per lavorare con questa tipologia di input/output. La codifica in vettore delle possibili singole classi è stata eseguita dalla funzione di Flow stessa durante il feeding[42] dei dati alla rete.
Ho scelto inoltre di non passare all'interno di questa colonna alcun dato che indicasse sede della lesione, età del paziente o altre informazioni, in quanto l'architettura Inception non avrebbe potuto elaborare tale informazioni da sola. Sarebbe quindi stato necessario creare un secondo branch che prendesse in considerazione come input, le sole features categoriche e, che da queste tirasse fuori una correlazione con le diagnosi, per poi andare a ricongiungere i due branch e implementare un classificatore finale. L'implementazione di tale procedura è stata scartata in quanto:

- Il secondo branch avrebbe dovuto restituire degli output compatibili con gli output del primo branch.
- Non dispongo, al momento, di competenze adeguate per poter costruire un branch che si affianchi a una rete neurale stato dell'arte per migliorarla.
- Costi computazionali ancora più elevati.
- Sarebbe stato necessario un dispendioso lavoro manuale di encoding delle features categoriche.

Per cui l'input fornito alla rete è la sola immagine, mentre l'output desiderato è rappresentato dalla classe della lesione.

3.3.3 Funzione obbiettivo

La funzione obbiettivo, anche nota come funzione di loss, è quella che misura la qualità delle soluzioni trovate. Tale funzione prende in ingresso gli input, i pesi dei

[41] È il risultato del one hot encoding

[42] Dare i dati 'in pasto' alla rete.

neuroni e i bias relativi, l'output della rete e l'output desiderato, infine restituisce un valore numerico. L'obbiettivo finale della rete neurale è trovare i valori dei parametri che la minimizzino, se invece va massimizzata si parla di funzione di fitness, ma non è questo il caso.
Esistono diverse metriche che possono essere usate per calcolare la funzione di loss, quella che ho deciso di usare è la *sparse categorical crossentropy*. Prima di spiegare cosa è e come funziona tale metrica è importante capire la natura del problema: il task di classificazione delle lesioni cutanee è un problema multi classe, in quanto le classi analizzate sono più di due, però è di tipo mutuamente esclusivo. Dire che un problema multi classe è mutuamente esclusivo, implica dire che l'output y_i può appartenere esclusivamente una sola classe C. Significa, ad esempio, che non è possibile affermare che una lesione cutanea appartiene alla classe melanoma e, al contempo, alla classa di cheratosi benigna. O appartiene a una classe o a un'altra.
Compresa la natura del problema, è necessario definire la crossentropia categorica: la crossentropia categorica è una delle metriche capaci di misurare la 'bontà' delle predizioni, che fornisce l'ultimo layer con funzione di attivazione Softmax. In parole povere, misura quanto l'output della rete è simile al ground truth[43] espresso in one hot encoding[44]. Prendiamo come esempio pratico una classificazione fatta dalla mia rete neurale, ricordando che:

- x è il generico input.
- C è l'insieme delle possibili classi.
- GT è la classe corretta dell'input x.
- PC è il vettore delle probabilità per ogni classe.

La figura 3.11 mostra visivamente il processo per il calcolo della crossentropia categorica, trattando i dettagli matematici invece: la crossentropia categorica è l'opposto del risultato della sommatoria dei risultati della moltiplicazione vettoriale dei valori GT_i per il logaritmo di PC_i. L'equazione è mostrata di seguito:

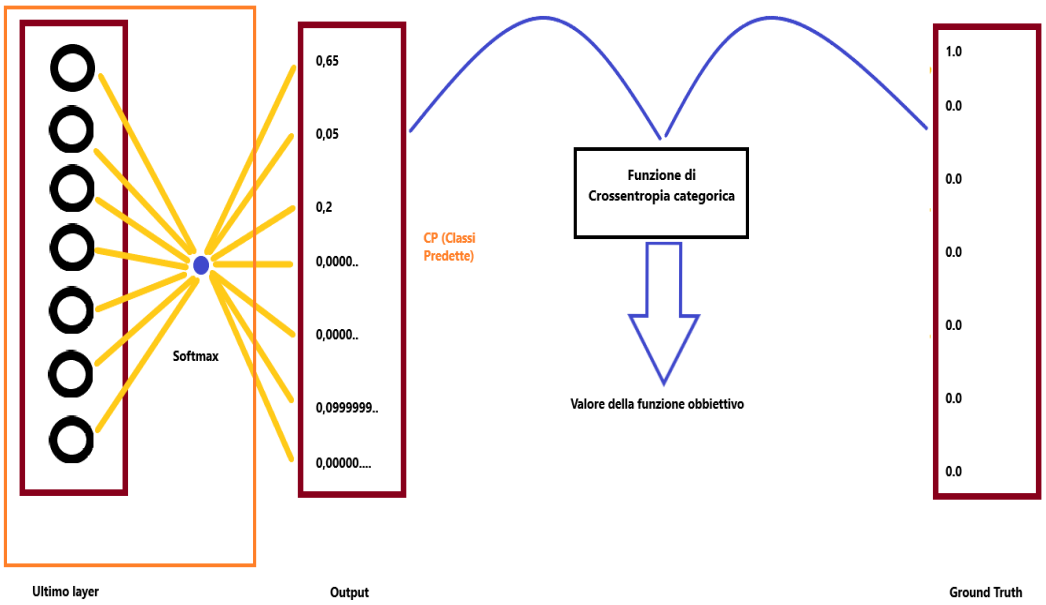
$$CC(PC, GT) = -\sum_{i \in C} GT_i x \log PC_i$$

Le quantità espresse da PC_i sono tutti numeri minori di uno, quindi il loro logaritmo fornisce un risultato negativo. Il segno meno fuori dalla sommatoria, serve per ottenere valori positivi, in quanto la crossentropia categorica è un valore sempre positivo.

[43] Vettore delle classi reali trasformato in notazione one hot encoding.

[44] Procedura che associa ad ogni variabile categorica un bit all'interno del vettore e a seconda della posizione dove il bit è settato a 1 si sa che è presente una feature piuttosto che un'altra.

Figura 3.11: Processo di calcolo della crossentropia categorica.



Si ottiene un unico valore per le regole di moltiplicazione vettoriale: si ha un vettore PC di dimensione 1x7 e un vettore GT di dimensione 7x1, quindi la loro moltiplicazione vettoriale risulta in un unico numero, che va poi a sommarsi con gli altri valori all'interno della sommatoria. Notando che la moltiplicazione vettoriale è tra i vettori PC e GT, dove GT presenta un solo valore unitario e tutti gli altri nulli, si ha che la crossentropia categorica è la sommatoria di un solo numero diverso da zero e tutti gli altri nulli.

L'unica differenza presente tra la crossentropia categorica e la crossentropia categorica diffusa è come viene scritto il ground truth nel one hot encoding.

Nel primo caso avremmo un one hot encoding del tipo:

$$
\begin{vmatrix} 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 \end{vmatrix} \\
\begin{vmatrix} 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0 \end{vmatrix} \\
\begin{vmatrix} 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0 \end{vmatrix}
$$

In questo caso si sono presi 3 ground truth di tre input. Nel caso invece di crossentropia categorica diffusa si ha che i ground truth verranno scritti nella seguente forma:

$$
\begin{vmatrix} 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 \\ 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0 \\ 0.0, 0.0, 0.0, 0.0, 5.0, 0.0, 0.0 \end{vmatrix}
$$

Tale rappresentazione è poi causa di un valore di loss superiore di uno.

3.3.4 Ottimizzatore

Ogni volta che una rete neurale termina il batch in analisi e finisce le predizioni degli input, la rete calcola un valore numerico che indica la differenza tra la classe reale e quella predetta, che viene usato per correggere i pesi della matrice dei pesi W e i bias b attraverso il processo di backpropagation. La correzione dei pesi indica la grandezza e la direzione dello step che verrà effettuato all'iterazione successiva. Lo step che si effettua è direttamente proporzionale al **learning rate**[45]. Questa correzione è essenziale per poter arrivare a convergenza, cioè riuscire ad avere una predizione massimamente simile alla classe reale; sostanzialmente si basa su una correzione iterativa dello step compiuto dalla rete.
Gli algoritmi di ottimizzazione servono per velocizzare le operazioni svolte dalla discesa del gradiente, il quale minimizza la funzione di costo.
Esistono diversi tipologie di ottimizzatori che compiono diversi procedimenti matematici per velocizzare la discesa del gradiente; nelle reti neurali, la tecnica base di discesa del gradiente è di tipo stocastico.
Tecnica del gradiente discendente stocastico (SGD):
L'SGD è l'algoritmo di ottimizzazione 'classico', e viene definito come un'approssimazione stocastica iterativa del gradiente discendente della funzione di loss. Un gradiente è un vettore di funzioni, ognuna delle quali è una derivata parziale della loss rispetto a ognuna delle variabili, e tali funzioni rappresentano i coefficienti angolari dell'iperpiano tangente alla funzione obbiettivo in quel punto. Ciò indica la direzione di massima variabilità di tale funzione.
Il gradiente, in sè, è una quantità che tende a massimizzare la funzione, mentre nel mio caso l'obbiettivo che si persegue è l'esatto opposto: la sua minimizzazione.
Il learning rate pesa quanto il gradiente influisce; per learning rate piccoli si ottiene che il gradiente ha un effetto piccolo, cioè si sposterà un punto in un intorno δ molto piccolo; per learning rate grandi l'intorno δ cresce di dimensioni e quindi lo step che si può fare lungo la funzione è maggiore, ma si rischia di oltrepassare il minimo della funzione.
Dopo aver fissato un input x, che è un batch di dimensioni molto minori rispetto al totale degli input disponibili, i pesi w e i bias b, si calcola il gradiente per quel

[45]Costante moltiplicativa della norma del gradiente

determinato input. All'iterazione successiva, però, l'input x cambia e conseguentemente, cambia anche la funzione che deriva da tale input. Il cambiamento che ne deriva è casuale, in quanto gli input sono stati raggruppati per batch in modo casuale, da ciò la definizione di stocastico.

Tra gli ottimizzatori della discesa del gradiente stocastico, ne vengono evidenziati due:

- **Adam:**

  Il nome Adam sta per *Adaptive Moment Estimation*[46]; è uno dei metodi per ottenere dei valori di learning rate che si adattano ai parametri.

  Tale algoritmo, oltre a mantenere in memoria la media del decadimento esponenziale del gradiente passato al quadrato m_t, mantiene anche la varianza del decadimento esponenziale del gradiente v_t, che è il momento statistico del secondo ordine. Il primo parametro è un momento m_t del primo ordine, il secondo, v_t è del secondo. Il problema è che tali valori tendono a zero, conseguentemente non vengono usati direttamente; vengono prima corretti attraverso degli stimatori, diventando $\hat{v}$ e $\hat{m}$, valori che vengono usati nella pratica.

  Mentre un momento può essere visto come una palla che rotola giù da una discesa, si può pensare ad Adam come a una pesante palla che genera attrito.

  I valori di m_t ev_t vengono calcolati come segue:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

  dove per brevità si è indicato il gradiente allo step t come g_t e $\beta_1 e \beta_2$ sono i parametri di decadimento, scelti empiricamente dagli autori a 0,9 e 0,9999. Servono per controllare i contributi dei gradienti passati e di quello presente. m_t e v_t sono rispettivamente stime del primo momento (la media) e il secondo

[46] *Stima del momento adattivo*

(la varianza) dei gradienti, la loro correzione è data dalla formula:

$$\hat{m} = \frac{m_t}{1 - \beta_1}$$
$$\hat{v} = \frac{v_t}{1 - \beta_2}$$

Infine, tali valori corretti vengono inseriti all'interno dei parametri del learning rate adattivo θ come segue:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}} + \epsilon}\hat{m}_t$$

- **Adadelta:**

  L'algoritmo Adadelta è l'evoluzione di un altro algoritmo che prende il nome di AdaGrad (Adaptive Gradient), il cui problema principale è avere un learning rate sempre decrescente: ciò si traduce in un arresto dell'addestramento prematuro, e alcune volte non corretto.

  L'AdaDelta cerca di risolvere il problema del learning rate costantemente decrescente. Non vengono presi tutti i precedenti gradienti al quadrato ma solo alcuni scelti all'interno di una finestr ω. Inoltre, a differenza di AdaGrad, la somma dei gradienti è ricorsivamente definita come media decadente dei gradienti passati. Ad ogni iterazione t viene calcolata la media mobile E del valore atteso del quadrato del gradiente, $E[g^2]_t$, che dipende unicamente dalla media passata e dal gradeinte corrente:

  $$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma){g^2}_t$$

  dove γ rappresenta una frazione del precedente vettore, solitamente viene fissata ad un valore attorno a 0,9. I parametri di update del learning rate $\Delta\theta_t$ vengono quindi aggiornati tramite la media decadente dei precedenti gradienti al quadrato $E[g^2]_t$:

  $$\Delta\theta_t = -\frac{\eta}{E[g^2]_t + \epsilon}g_t$$

  dove ϵ è un termine di regolarizzazione per evitare di dividere per 0. Inoltre si nota che il denominatore non è altro che il valore RMS del gradiente, quindi, l'espressione può essere riscritta come:

  $$\Delta\theta_t = -\frac{\eta}{RMS[g]_t}g_t$$

Con questa formulazione, gli autori del codice, hanno riscontrato che l'update dei parametri non veniva eseguito in maniera corretta. In particolare, nella formula appena scritta, è ancora presente la scelta aprioristica del learning rate η , inoltre si sono accorti che gli ordini di grandezza tra gradiente e denominatore non erano delle stesse magnitudo; da ciò derivavano problemi durante l'addestramento. Conseguentemente, viene definita un'altra media mobile esponenzialmente decadente, quella delle variazioni dei parametri al quadrato $\Delta\theta^2$:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

Il cui valore RMS è:

$$RMS[\Delta\theta]_t = \sqrt{E[E\Delta\theta^2]_t + \epsilon}$$

Dato che il valore RMS di $[\Delta\theta]_t$ non è noto esso viene approssimato con quello dello step precedente. E il valore del learning rate η viene sostituito con il valore RMS $[\Delta\theta]_{t-1}$. La regola di update è quindi:

$$\Delta\theta_t = \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Per cui tramite l'utilizzo di AdaDelta non si ha la necessità di settare un valore di default del learning rate. Tale algoritmo è stato valutato come ottimale per il mio lavoro, ed è stato quello adoperato.

3.3.5 Callbacks

All'interno delle tecniche di regolarizzazione sono presenti i callbacks; funzioni eseguibili che prendono in ingresso dei parametri che vengono passati da altre funzioni durante il training di una rete neurale, e possono essere di due tipi: sincorni o asincroni.
Tali tecniche hanno come scopo principale quello di evitare l'overfitting, quindi cercano di ridurre l'errore tra i risultati del training e del test set.
Tra quelli messi a disposizione da Keras, ho scelto di utilizzarne tre:

- Model Checkpoint.

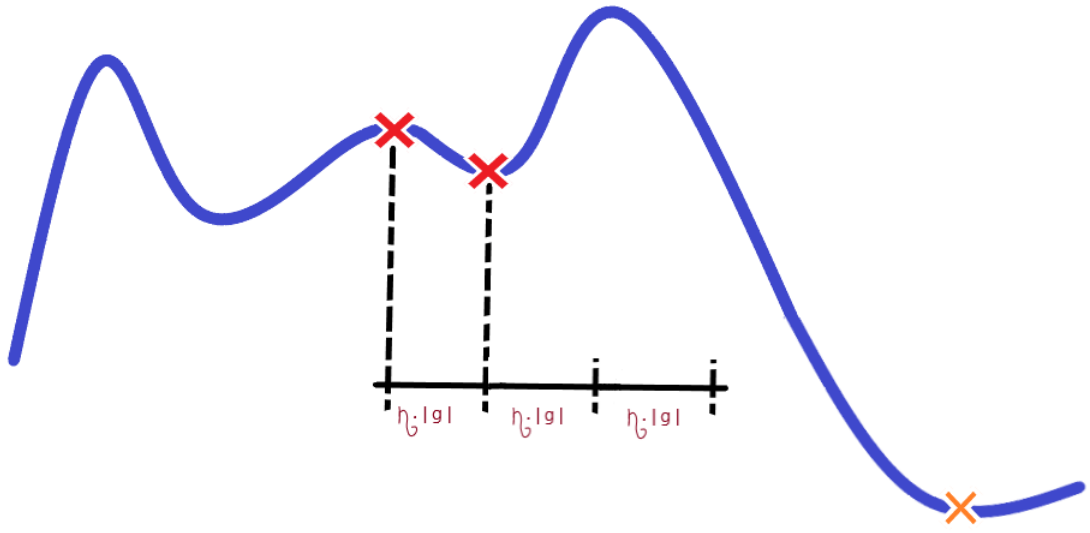
- Early Stopping.
- Reduce LR on Plateau.

Il Model Checkpoint viene utilizzato per creare un salvataggio, per l'appunto checkpoint, del modello che si sta addestrando. Il checkpoint può essere usato sia come modello finale, che come punto di partenza per continuare successivamente l'addestramento. Lo scopo è evitare di perdere i risultati ottenuti a causa di qualche problema casuale. Nel caso in analisi è stato necessario usare questo callback; a causa dell'eccessivo consumo di memoria durante l'addestramento, dovuto alla combinazione tra il numero di immagini e le loro dimensioni, non era infatti possibile addestrare il modello per più di 40 epoche alla volta.
Il Checkpoint è stato implementato in modo tale da monitorare il valore della **val loss**[47], e, per evitare un eccessivo consumo di memoria, si è scelto di salvare il checkpoint ogni cinque epoche. Inoltre, un secondo parametro aggiunto è stato quello per salvare, unicamente, il checkpoint migliore tra quelli visti nelle cinque epoche precedenti.
Durante il training della rete neurale, l'errore della funzione di loss è settato a un valore casuale iniziale, per poi iniziare a decrescere. Lo stesso andamento si dovrebbe ritrovare anche durante il test, ma non è sempre vero: succede che, a volte, l'errore, dopo un'iniziale decrescita, torni a crescere. La ricrescita dell'errore è sinonimo di overfitting, perchè la rete non generalizza più le caratteristiche degli input di training: impara a memoria le caratteristiche del training e non trovandole nello stesso modo, negli input di test, sbaglia la classificazione.
L'idea dell'Early Stopping consiste nell'adoperare un validation set su cui calcolare in modo periodico il valore della funzione di loss, anzichè testarla sul training, e nel momento in cui torna a risalire fermare l'addestramento del modello.
In realtà esistono dei parametri che permettono di inserire delle tolleranze, in quanto è plausibile che la funzione di loss si trovi all'interno di un minimo locale, e per poter raggiungere il minimo assoluto debba passare attraverso degli step peggiorativi. Tale concetto viene spiegato in figura 3.12, presente nella pagina seguente.
Come si può notare, se non si ammettessero step peggiorativi, la rete non arriverebbe al minimo globale in arancione, e resterebbe dentro il minimo locale in rosso.
In ogni caso, è importante non esagerare con gli step peggiorativi, quindi ho impostato il parametro *patience*[48] a cinque epoche.
Il learning rate, come già detto, moltiplica la norma del gradiente, quindi è un parametro di controllo dello step che si deve compiere. Alcune volte può succedere

[47] Valore della funzione di loss calolata nel validation set.

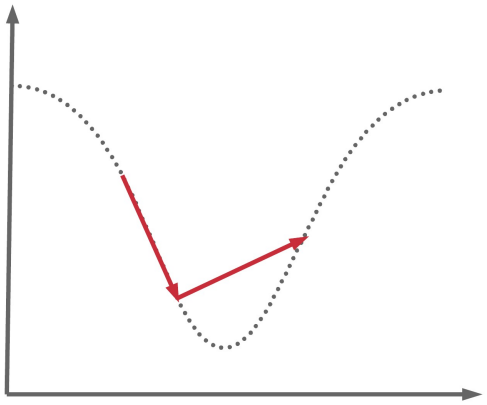
[48] Parametro che indica le epoche d'attesa prima di fermare l'addestramento.

Figura 3.12: Step peggiorativi per raggiungere il minimo globale.



che, pur avendo impostato bene gli ottimizzatori e le altre funzioni di controllo, la rete non riesca a convergere al minimo globale in tempi accettabili. Una delle tante cause può essere avere un learning rate costante. Alcune volte è necessario adattare lo step al punto della funzione in cui la rete si trova a navigare; se stiamo percorrendo la parte con una forte pendenza, il gradiente avrà un valore elevato, e se non si compie un passo adeguato si rischia di saltare il minimo, come mostrato in figura 3.13 nella pagina seguente.
Viceversa, con un learning rate troppo piccolo invece si rischia di perdere davvero tanto tempo, inoltre c'è il rischio che si blocchi in un minimo locale. Una possibile strategia per ovviare a questo problema è stata l'implementazione del Reduce Learning Rate on Plateau: tecnica di riduzione del learning rate se non vi sono miglioramenti nella metrica che si sta monitorando (in questo caso la loss calcolata sul validation set). Il momento in cui la metrica non migliora più è quando raggiunge un plateau nella funzione, allora bisogna ridurre il learning rate e dare alcune epoche all'ottimizzatore per esplorare nuovi spazi sulla superficie dell'errore affinchè trovi la strada giusta. Si è impostato un valore di patience pari a 3 epoche, con un

Figura 3.13: Step troppo grande.



fattore di riduzione del learning rate pari a 0.02. Inoltre, è stato impostato un parametro che definisce il learning rate minimo, sotto al quale non si può scendere, impostandolo ad un valore pari a 0.001.
Comunque sia non si è mai raggiunto un platau tale per cui è stato necessario che questa callback riducesse il learning rate: ciò si può tradurre in un buon setting di tutti gli altri parametri.

3.4 Analisi dei risultati

Il modello di rete neurale così implementato è stato allenato e testato sui tre set di dati che ho creato. Il train set presenta un numero di elementi pari a 5469, il validation set un numero pari a 1367 e il test pari a 1710. L'allenamento è terminato salvando un modello che presentava le caratteristiche di loss e accuracy riportate in *Tab. 3.2*

Tabella 3.2: Risultati del training.

Epoca	**Loss**	**Accuracy**	**Validation Loss**	**Validation Accuracy**
543	1.7782	0.8643	1.6572	0.8998
544	1.7779	0.8674	1.6477	0.9027
545	1.7706	0.8667	1.6516	0.9020

Conseguentemente il modello salvato è quello corrispondente all'epoca 544: presenta

il minor valore di validation loss e la validation accuracy maggiore. Il perchè la funzione di loss assuma dei valori sopra quello unitario è già stato discusso; il corrispettivo valore di 1,6477, rispetto al massimo di 7, è particolarmente decentrato verso il valore 1. Ciò indica che si ha un valore molto basso di loss; l'obbiettivo era minimizzare tale valore.
L'accuracy invece raggiunge un valore di 90,27%, che per il problema multiclasse affrontato è un risultato ottimo. Si ricorda che le classi non erano ben rappresentate e il numero totale di esempi non era sufficiente.
Come si poteva prevedere, nel test le statistiche decrescono:

Tabella 3.3: Risultati del test.

Test Loss	**Test Accuracy**
1.773985	0.852047

Ciò potrebbe indicare un inizio di overfitting, anche se la diminuzione percentuale è ancora troppo poco significativa perchè si parli di overfitting; inoltre, il test set è composto da 1710 elementi, visti un'unica volta e divisi in batch da 32 elementi. Conseguentemente, questi risultati vengono valutati sulla media di soli 54 batch visti una sola volta. Tali ragionamenti portano ad escludere la presenza di overfitting.
Inoltre, in un problema di classificazione con dati aventi tali distribuzioni, la sola metrica di accuracy non è minimamente significativa, in quanto il suo valore è sbilanciato a causa della predominanza della classe nevo. Per maggiore chiarezza si riporta di seguito come viene calcolata l'accuracy:

$$accuracy = \frac{1}{N_{classi}} \sum_{i \in N_{classi}} \frac{TP}{\sum x \in i}$$

Dove TP è il numero dei corretti classificati (True Positive), N_{classi} è il numero totale di classi, e $x \in i$ è il numero totale di elementi appartenenti a ciascuna classe. Considerando che è una media dei vari valori di accuracy delle singole classi, e che la classe nevo e melanoma costituiscono più del 65% del totale di immagini del test, si comprende perchè è una metrica non significativa.
Il modello presenta una percentuale globale di misclassificazione di circa il 13,5%.

Riferendoci alle prestazioni delle singole classi vengono riportate metriche più significative all'interno della tabella 3.4

Tabella 3.4: Risultati per classe.

Classe	**Precision**	**Rcall**	**F1-score**	**Support**
Cheratosi attinica	0,68	0,34	0,45	88
Basalioma	0,89	0,75	0,81	161
Cheratosi seborroica	0,63	0,91	0,74	237
Dermato-fibroma	1,00	0,06	0,11	36
Melanoma	0,94	0,85	0,89	447
Nevo	0,93	0,99	0,96	700
Lesioni vascolari	0,92	0,88	0,90	41

Tali metriche sono la precision, rcall e l'f1-score, vengono calcolate come:

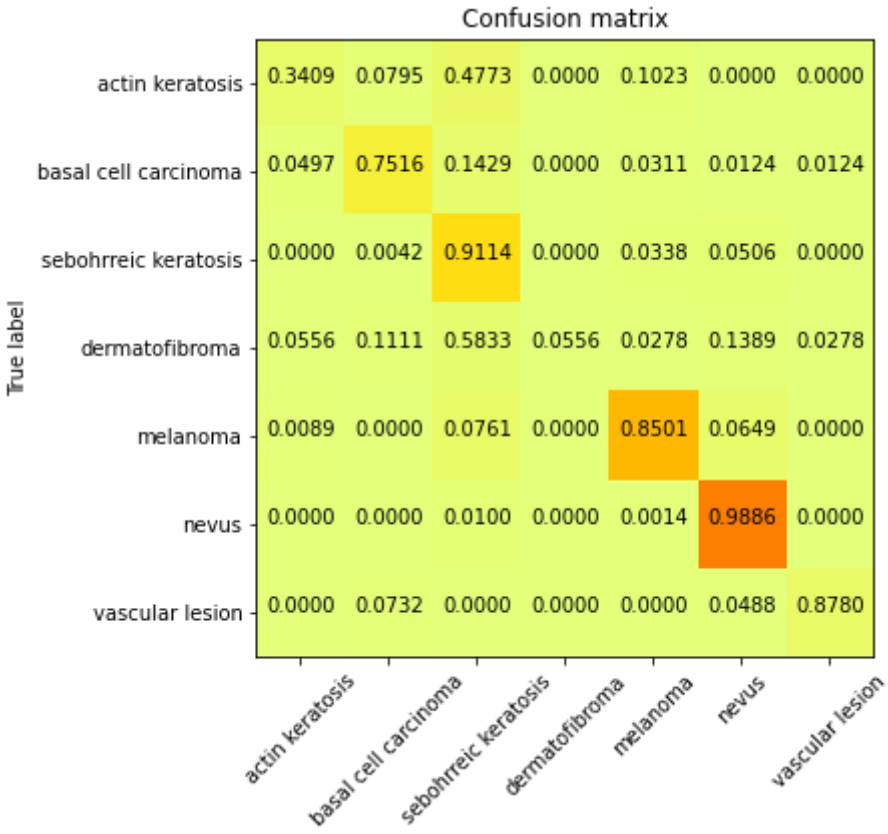
$$precision = \frac{TP}{TP + FP}$$

$$rcall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 * Rcall * Precision}{Rcall + Precision}$$

La precision indica l'abilità di un classificatore di non etichettare un'istanza negativa quando è positiva. L'rcall misura la capacità di un classificatore di trovare tutte le istanze positive, infine l'f1-score è una media armonica ponderata delle metriche di precisione e rcall. Invece il Support è solamente il numero di immagini per classe. Le prestazioni del modello addestrato sono riassunte all'interno della Confusion Matrix, riportata nella pagina successiva in figura 3.14

Figura 3.14: Confusion Matrix.



Dalla matrice è possibile vedere sia le percentuali di corretta classificazione, sia la popolosità delle predizioni in tale classe. La colorazione indica il numero di elementi presenti in tale classe: più è scuro il riquadro, più la classe è popolosa. Invece all'interno dei singoli quadrati sono presenti le percentuali.
In tabella 3.5 sono riportate altre due metriche usate per valutare il modello.

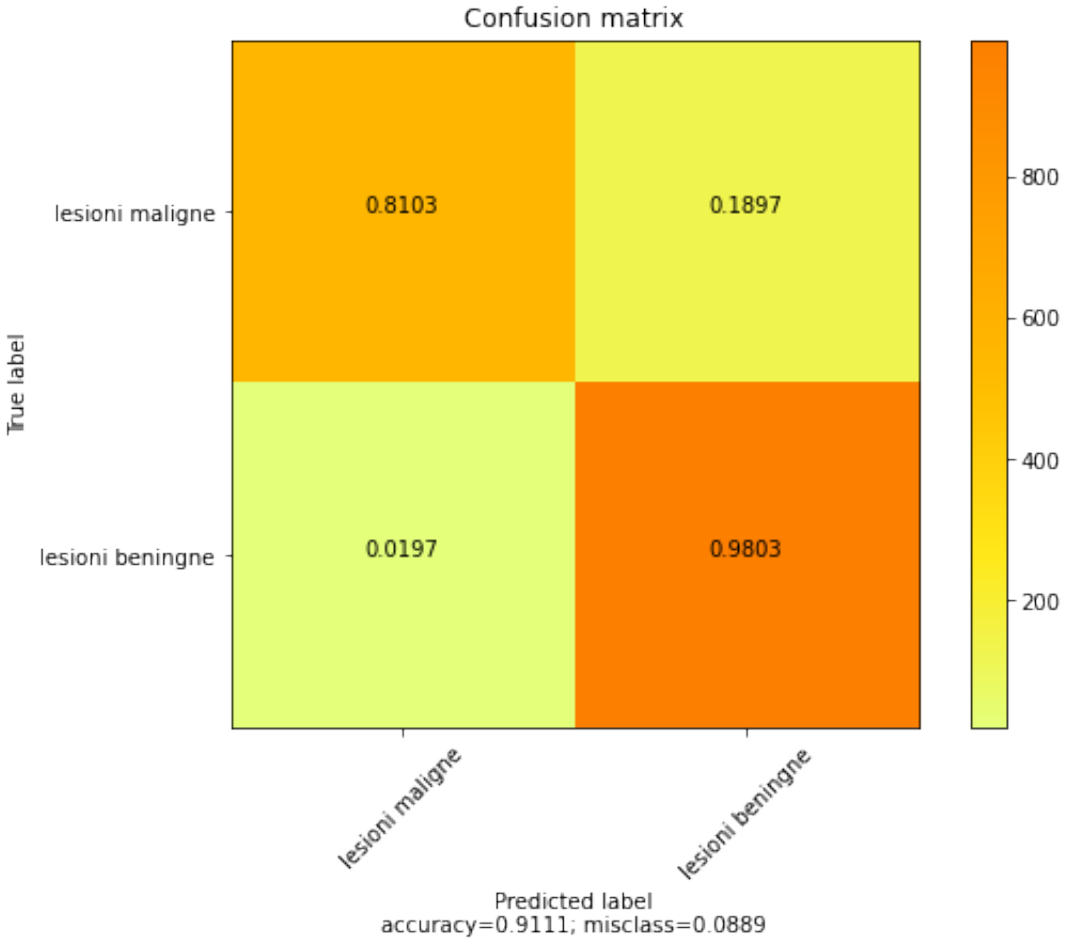
Tabella 3.5: Altre metriche.

	Precision	**Rcall**	**F1-score**	**Support**
Macro Media	0,85	0,66	0,67	1710
Media Pesata	0,86	0,85	0,84	1710

La macro media è una metrica che valuta tutte le classi allo stesso modo, quindi non viene influenzata dalla popolosità della classe e indica la 'bontà globale' del

classificatore, ed è un parametro particolarmente significativo quando le classi sono ben bilanciate. La media pesata è la macro media ottenuta pesando ogni classe per la propria popolosità. Indi per cui, tra le due è la metrica che ha un maggiore peso. Ho fatto un'ulteriore analisi delle prestazioni del classificatore, dove ho ricalcolato tutte queste metriche nel caso di classificazione binaria: classe benigna e classe maligna. In figura 3.15 si riporta la nuova matrice di confusione.

Figura 3.15: Classificazione binaria.



Le classi sono state suddivise in:

- Lesioni Maligne:
  - Melanoma
  - Basalioma
  - Cheratosi Attinica

- Lesioni Benigne:
  - Nevo
  - Dermatofibroma
  - Cheratosi Seborroica
  - Lesioni Vascolari

Le metriche sono state ricalcolate a mano, sulla base dei risultati della matrice di confusione multiclasse, altrimenti si sarebbe dovuto costruire un nuovo classificatore e avrebbe perso di senso.
Le percentuali di Precision, Rcall e F1-score per la nuova classificazione, vengono riportate in tabella 3.6.

Tabella 3.6: Altre metriche.

	Precision	**Rcall**	**F1-score**	**Support**
	0,9657	0,8103	0,8812	1710

Trasformando i risultati della classificazione multicalsse in binaria si è riusciti ad ottenere una classificazione ancora migliore della precedente.
Considerando invece gli aspetti negativi, è ben visibile che l'elemento più preoccupante in assoluto è il tasso dei falsi negativi: poco meno del 20%. È un valore significativamente alto, e indica che circa il 20% delle lesioni maligne sono state classificate come benigne.
Conseguentemente è stata fatta un'analisi sulla classificazione multiclasse da cui deriva tale classificazione binaria. La tabella 3.7 riporta i valori della sola classe di cheratosi attinca, quella con minor supporto tra le classi maligne.

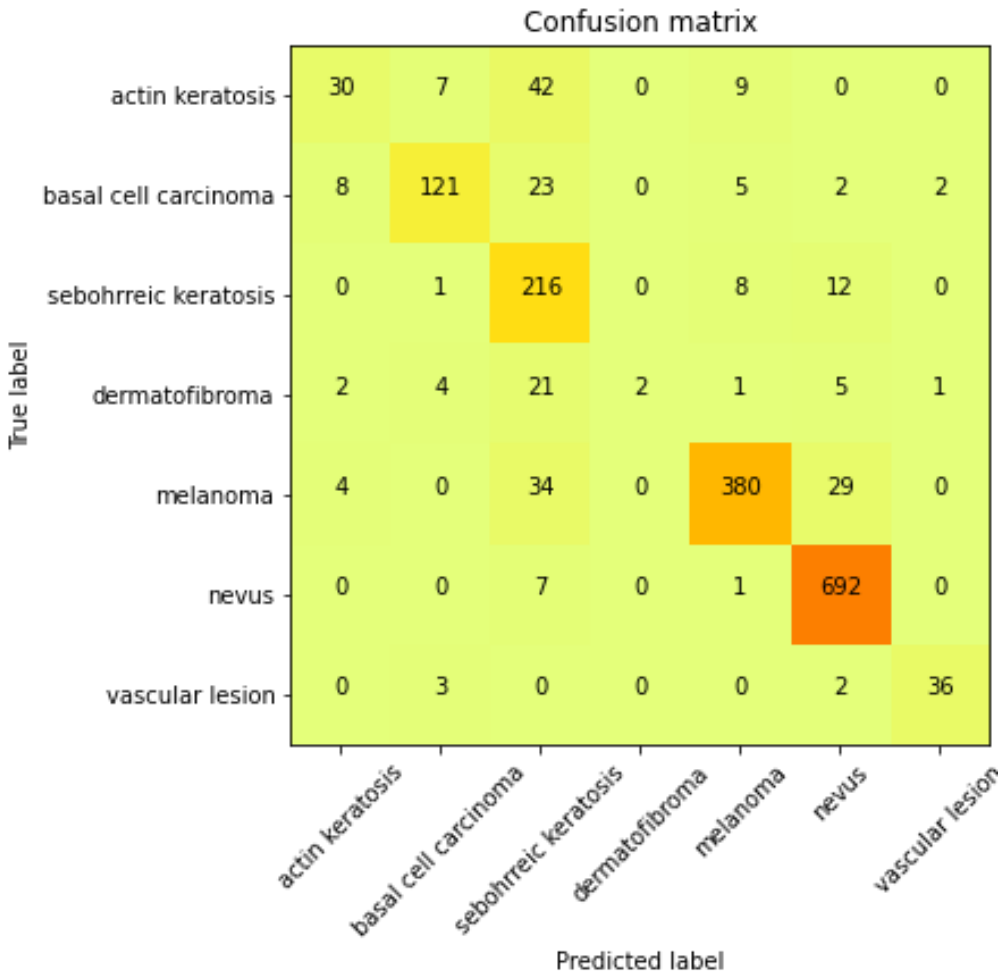
Tabella 3.7: Cheratosi attinica.

	Precision	**Rcall**	**F1-score**	**Support**
Cheratosi attinica	0,68	0,34	0,45	88

Sono le prestazioni più basse ottenute relativamente alle classi riferite come maligne. Tra le cause, una prima motivazione è data dal fatto che: la classificazione multiclasse è molto più complessa di una classificazione binaria e, conseguentemente, il

risultato riportato nella matrice di confusione, per quanto riguarda la classificazione binaria, è ampiamente influenzato dalle difficoltà intrinseche della classificazione multiclasse. Oltre ciò, si aggiunge un totale di istanze non rappresentativo.
Si riporta in figura 3.16 come le varie istanze vengano classificate dal modello.

Figura 3.16: Istanze della classe cheratosi attinica interpretate dal classificatore.



Ricordandosi delle statistiche delle varie classi, dalla *fig* 3.16 si capisce che più della metà delle istanze cheratosi attinica vengano interpretate come cheratosi seborretiche, ma nessuna cheratosi seborroica è identificata come attinica. Da ciò è chiaro che il modello abbia appreso erroneamente dei parametri relativi alle cheratosi attiniche e interpretati come fossero appartenenti alle cheratosi seborettiche. Tali 42 istanze, pesano più del 6% a livello di misclassificazione, e la percentuale di Rcall crescerebbe poco oltre l'87%. Sono meno gravi, in senso assoluto, le 9 istanze misclassificate come melanomi: entrambe le classi sono maligne, il problema

derivante è minore in quanto in ogni caso si manderebbe il paziente a fare l'istologico. Ciononostante, è indubbio che il modello abbia trovato una relazione tra melanoma e cheratosi attinica, infatti classifica 34 melanomi come cheratosi seborretiche (per la relazione che trova tra le prime due). Il secondo parametro preoccupante, primo per gravità, sono le 29 istanze di melanomi classificate come nevi. Non ho idea della dipendenze che il classificatore possa trovare, in quanto la classe dei nevi presenta un solo misclassificato nella classe dei melanomi.
Una mia ipotesi è che tale misclassificazione sia causa diretta dell'unione dei due dataset; aumentando la diversità intra-classe si sono ritrovati dei parametri tipici dei nevi all'interno delle immagini di melanomi. Una seconda causa può essere: le immagini presentavano la classificazione derivante da istologico, determinati melanomi potrebbero essere in uno stadio particolarmente precoce per cui, al momento della cattura dell'immagine dermatoscopica non presentavano ancora chiare caratteristiche melanocitiche.

Capitolo 4

Conclusioni

I risultati riportati in 3.4 rappresentano la conclusione del lavoro svolto. Il modello sviluppato è in grado di discriminare tra sette diverse classi di lesioni cutanee con alte prestazioni. Le classi dove è stato verificato il maggiore calo di prestazioni, sono quella dei dermatofibromii e quella della cheratosi attinica.

Come già detto, la causa di tale calo potrebbe essere attribuibile allo scarso numero di istanze che compongono la classe delle cheratosi attiniche. Al fine di avere una valutazione non solo quantitativa ma anche qualitativa riguardo il lavoro svolto, è stato ritenuto opportuno mettere a confronto il modello sviluppato con quello sviluppato in altri lavori analoghi. In particolare, si è preso in considerazione il lavoro svolto da [17] che si propone di creare un algoritmo di classificazione delle lesioni cutanee in sette classi differenti (le stesse classi usate nel lavoro appena descritto). Usando il dataset pubblico dell'ISIC e un approccio particolarmente brillante è stato preso il modello di rete neurale ResNet 101 combinando l'uscita di questo modello con quella di un secondo branch che prende come input delle caratteristiche topologiche. Infine i due branch, previa pesatura degli output singoli, vengono convogliati assieme attraverso un layer FC e, in ultimo, il tutto viene fatto classificare da un classificatore finale.

Il cuore di questo lavoro è la fusione di un'architettura convoluzionale con un branch per l'estrazione di parametri topologici, estratti tramite i valori statistici di PC (curva di persistenza) e PS (statistiche di persistenza). Sono state valutate le prestazioni del modello sia con tutte le features topologiche assieme, sia con un numero ridotto. La riduzione del numero di features si è rivelata la migliore.

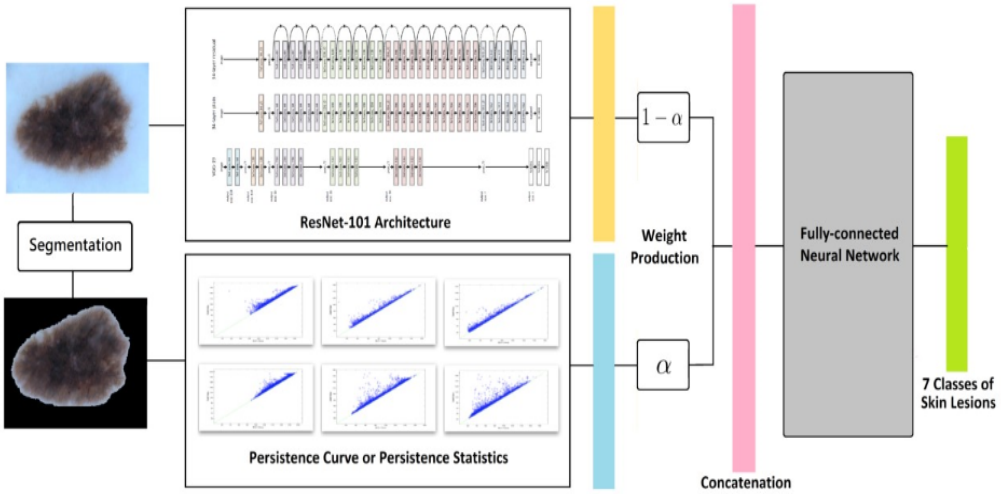
La fusione di questi due rami si è rivelata ottimale. In *Tab. 4.1* vengono riportati i risultati di questo lavoro.

Tabella 4.1: Esempio dei metadati.

Modello	**Accuracy**	α
ResNet 101	0.806	NaN
ResNet 101 PC RGB	0.837	0.22
ResNet 101 All Features (reduced)	0.851	0.23

Il parametro α rappresenta un coefficiente moltiplicativo all'uscita dei due rami. Un'idea più immediata la si può avere dalla figura 4.1, nella quale si mostra il processo di classificazione.

Figura 4.1: Modello topoResNet101.



L'approccio usato nel lavoro [17] è molto promettente, ciò nonostante anche in assenza dell'analisi delle features topologiche, il modello sviluppato in collaborazione con l'azienda TESI ottiene delle prestazioni di accuracy praticamente uguali. Sarebbe opportuno quindi poter valutare altre metriche con un maggior contenuto informativo, però non sono state riportate.
Tra i possibili obbiettivi utili per il proseguo di questo lavoro, evidenzio:

- La creazione di un dataset che presenti classi popolose e ben bilanciate. Per ottenere una condizione del genere è stato valutato che la popolazione minima

di immagini, per singola classe, debba essere di almeno 350 immagini dermatoscopiche. La creazione di tale dataset ha lo scopo di valutare la differenza di prestazioni che si otterrebbe, così confermando che una popolazione inconsistente all'interno di una classe, porta a risultati similari a quelli da me ottenuti per le cheratosi attiniche. Quindi migliorare il lavoro presentato in 3.1.

Per valutare l'aumento di prestazioni sarebbe utile che vengano sviluppati diversi modelli, addestrati singolarmente, con il dataset bilanciato a meno di due classi che, durante il corso dell'addestramento, vengono man mano ribilanciate. Le matrici di confusione che si dovrebbero ottenere dovrebbero presentare comunque un legame tra le due classi, come quello ritrovato nel mio lavoro in 3.4, ma con valori di misclassificazione minori.

- Attraverso la tecnica di transfer learning, modificare l'architettura di questo modello aggiungendo un branch di sole feature categoriche, come mostra la figura 3.3. Tale branch dovrà poi riconnettersi al branch dell'immagine prima del classificatore finale. Un lavoro similare servirebbe ad indagare a livello statistico la correlazione tra le sedi delle lesioni e la loro prognosi, come svolto da [17], però usando l'architettura di Inception e non quella di ResNet.

- Infine, la possibilità di tradurre e incorporare questo algoritmo all'interno di dispositivi dermatoscopici, di modo che diano come output sia l'immagine dermatoscopica che la diagnosi di tale lesione. Un tale sviluppo potrebbe portare a diverse migliorie:

  – Un considerevole risparmio a livello economico conseguente alla riduzione di esami istologici.

  – Una riduzione dei tempi medi di analisi: il dermatologo studierebbe unicamente la veridicità della prognosi fatta dal software, e non più l'immagine per poi, eventualmente, richiedere un esame istologico, con aumento dei possibili controlli, a parità di tempo impiegato.

  Sempre nell'ottica di un ulteriore sviluppo del lavoro svolto, sarebbe utile creare lo storico delle mappature eseguite per paziente. Valutando la correlazione sede/prognosi, e avendo inoltre lo storico dell'evoluzione della lesione nel tempo, si riuscirebbe ad ottenere delle prognosi con un'accuratezza sicuramente molto vicina al cento per cento, andando così a ridurre significativamente la necessità di esami istologici.

Quindi, ripercorrendo i passi seguiti per lo sviluppo del software, sono state messe in luce le criticità maggiori e per chiarezza riassuntiva viene riportata la tabella 4.2, dove si riportano tutti assieme i modelli e i loro risultati.

Tabella 4.2: Esempio dei metadati.

Modello	**Dataset**	**Classi**	**Transfer Learning**	**Accuracy**
ResNet 101	ISIC	7	No	0.806
ResNet 34	HAM10000	7	Si	0.905
VG 16	HAM10000	7	Si	0.780
Inception	ISIC	4	Si	0.91
ResNet 101 (Features Ridotte)	ISIC	7	No	0.851
Inception Modificato	ISIC HAM10000	7	Si	0.852
Inception Modificato	ISIC HAM10000	2	Si	0.911

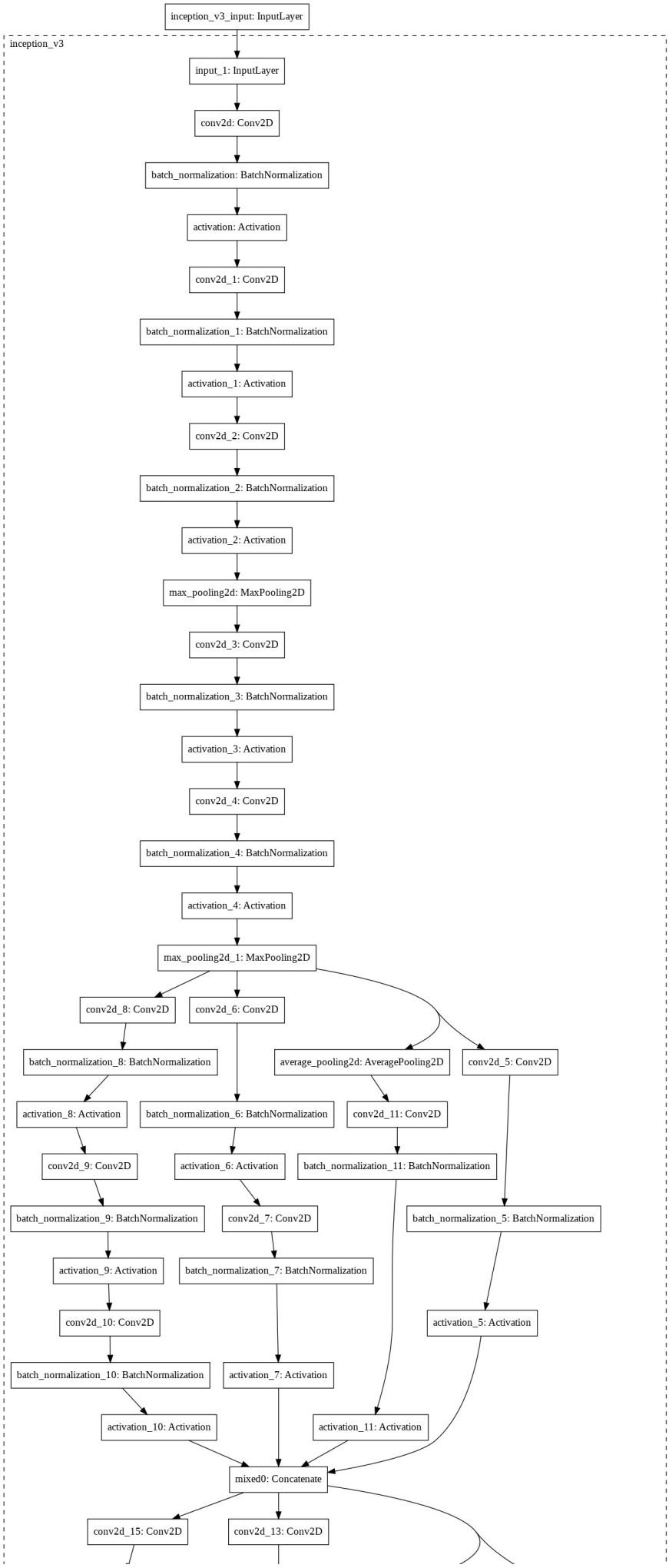
Purtroppo spesso mancavano metriche più significative della sola accuracy. Sarebbe stato interessante confrontare metriche come la precision o l'F1-score, che avrebbero potuto fornire informazioni più precise sulle reali prestazioni dei vari modelli. Comunque sia, il modello sviluppato con questo lavoro si attesta alla pari degli altri modelli.
I due modelli superiori, ResNet 34 e Inception raggiungono prestazioni più elevate per le seguenti motivazioni:

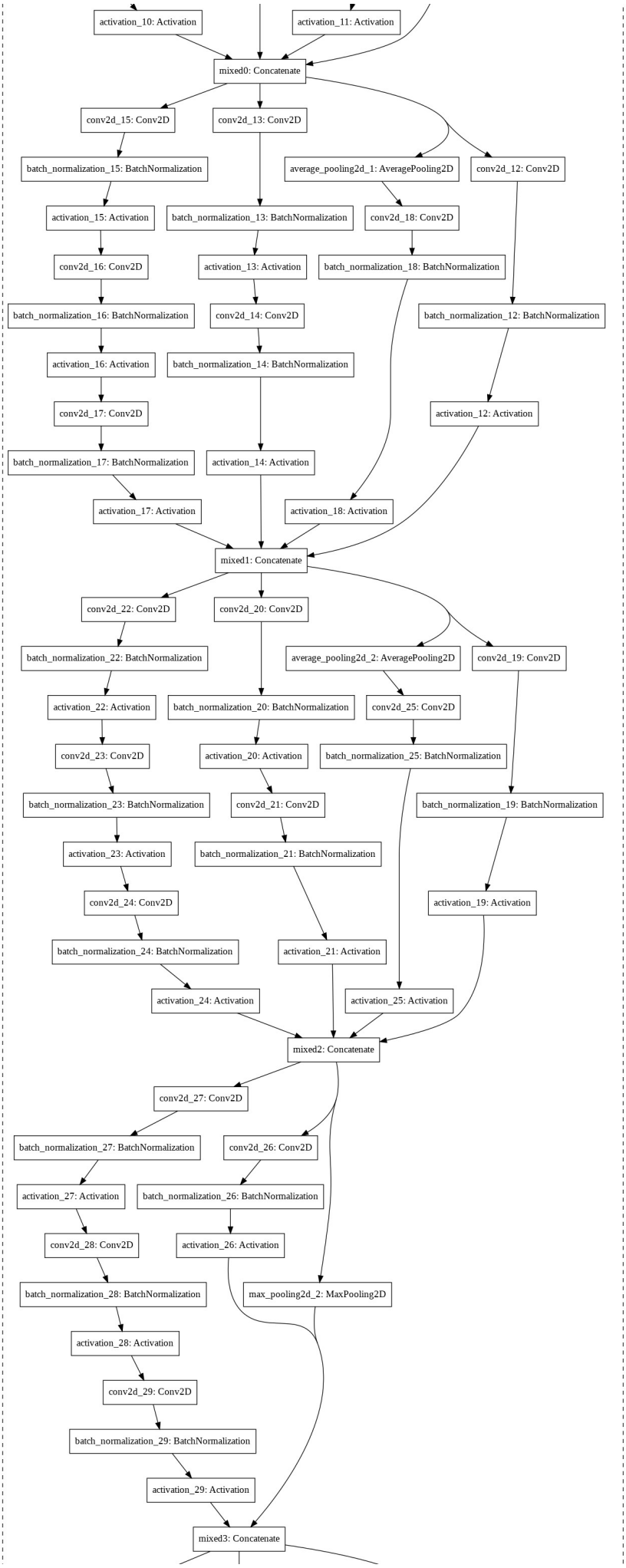
- è stato fatto transfer learning;
- è stato possibile adoperare delle immagini di dimensioni maggiori;
- nel caso di Inception, si è usato un numero di classi inferiori;
- nessuno dei modelli sviluppati negli altri lavori gestivano dati provenienti da diversi dataset: quindi analizzavano un dataset uniforme.

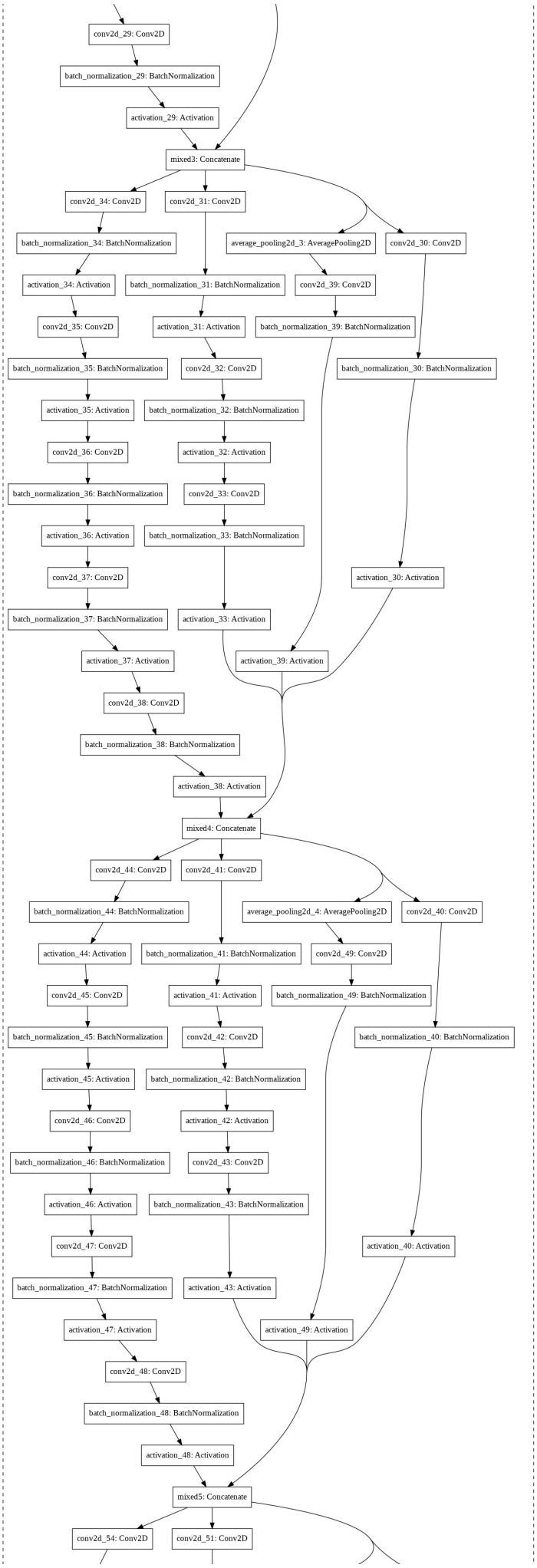
In conclusione si conferma che ancora oggi, quando si parla di DNN, una delle problematiche più significative riguarda l'analisi e la costruzione di un dataset coerente con le necessità del problema che il modello di rete neurale deve analizzare. Riuscire a limitare la dipendenza dell'apprendimento del modello dal dataset di partenza costituirebbe un notevole passo in avanti per la ricerca.

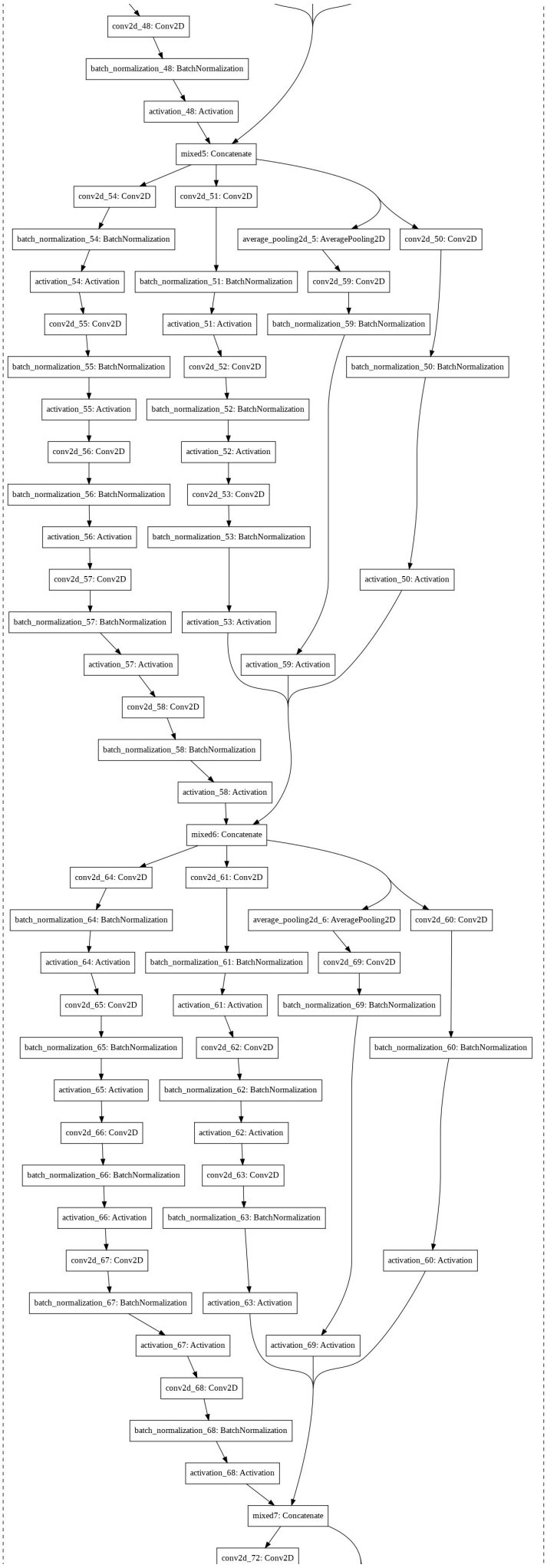
Appendice A

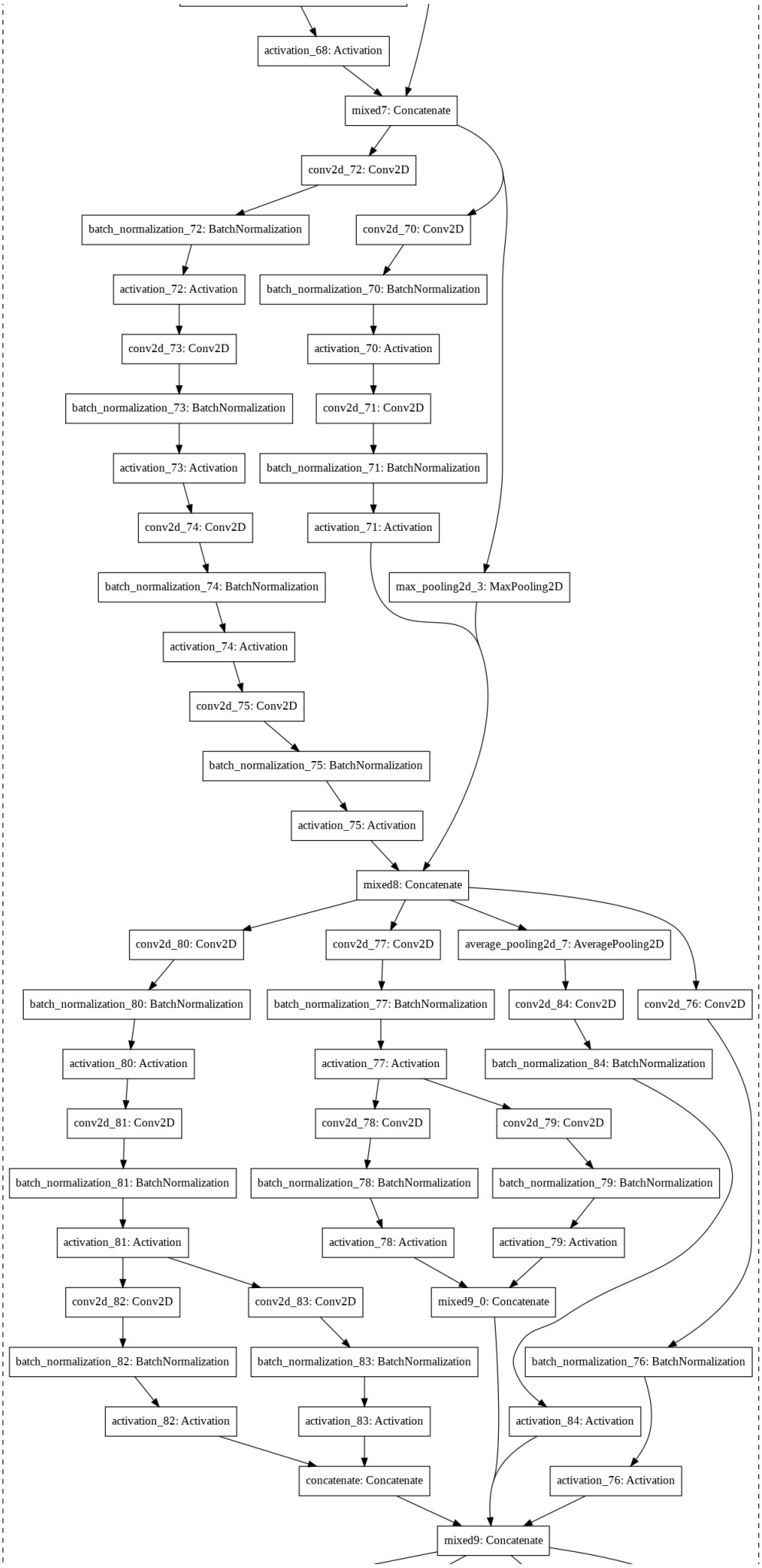
Architettura Inception V3

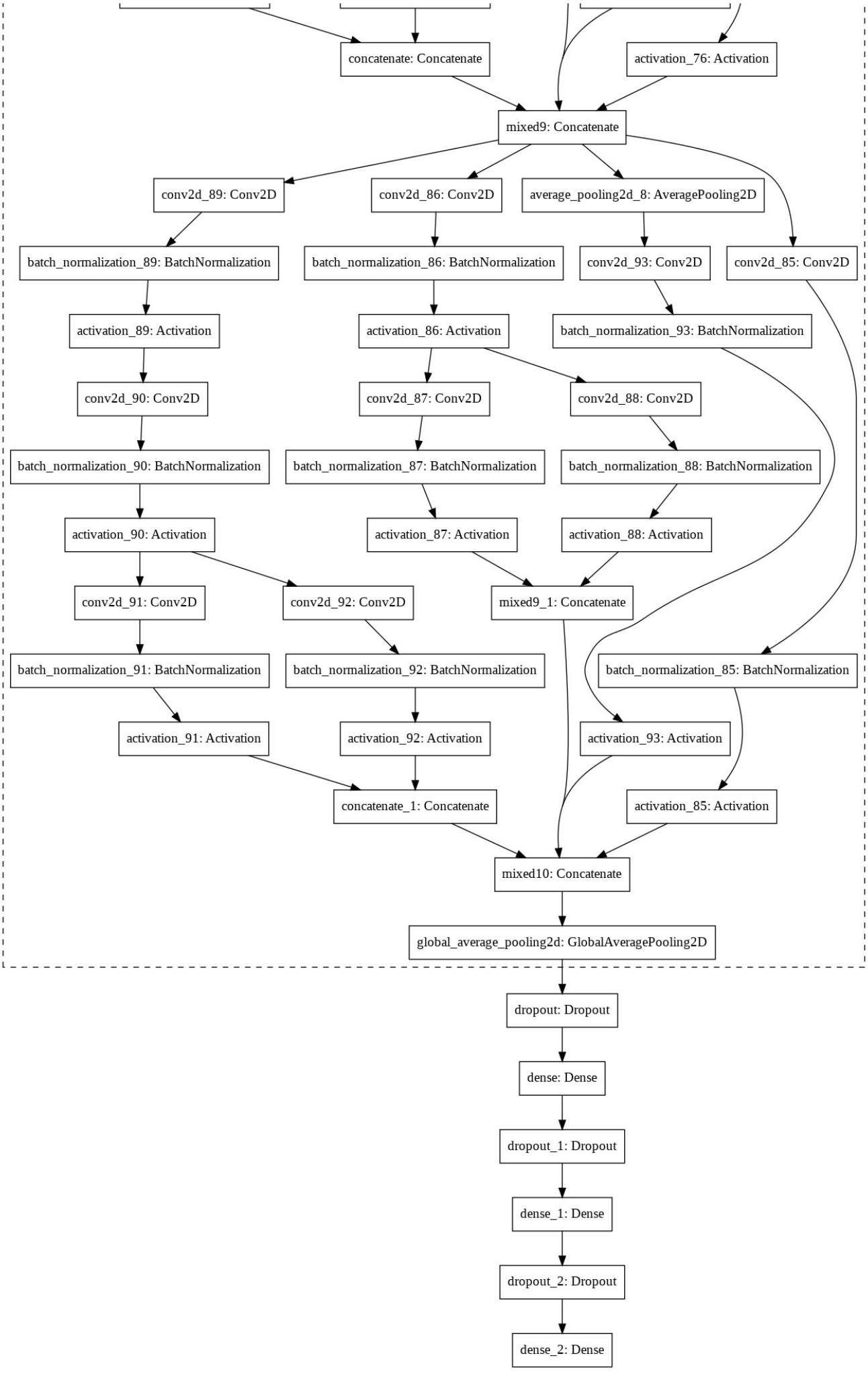

inception_v3_input: InputLayer
inception_v3
input_1: InputLayer
conv2d: Conv2D
batch_normalization: BatchNormalization
activation: Activation
conv2d_1: Conv2D
batch_normalization_1: BatchNormalization
activation_1: Activation
conv2d_2: Conv2D
batch_normalization_2: BatchNormalization
activation_2: Activation
max_pooling2d: MaxPooling2D
conv2d_3: Conv2D
batch_normalization_3: BatchNormalization
activation_3: Activation
conv2d_4: Conv2D
batch_normalization_4: BatchNormalization
activation_4: Activation
max_pooling2d_1: MaxPooling2D
conv2d_8: Conv2D
conv2d_6: Conv2D
average_pooling2d: AveragePooling2D
conv2d_5: Conv2D
batch_normalization_8: BatchNormalization
activation_8: Activation
batch_normalization_6: BatchNormalization
conv2d_11: Conv2D
conv2d_9: Conv2D
activation_6: Activation
batch_normalization_11: BatchNormalization
batch_normalization_9: BatchNormalization
conv2d_7: Conv2D
batch_normalization_5: BatchNormalization
activation_9: Activation
batch_normalization_7: BatchNormalization
conv2d_10: Conv2D
activation_5: Activation
batch_normalization_10: BatchNormalization
activation_7: Activation
activation_10: Activation
activation_11: Activation
mixed0: Concatenate
conv2d_15: Conv2D
conv2d_13: Conv2D


activation_10: Activation
activation_11: Activation
mixed0: Concatenate
conv2d_15: Conv2D
conv2d_13: Conv2D
batch_normalization_15: BatchNormalization
average_pooling2d_1: AveragePooling2D
conv2d_12: Conv2D
activation_15: Activation
batch_normalization_13: BatchNormalization
conv2d_18: Conv2D
conv2d_16: Conv2D
activation_13: Activation
batch_normalization_18: BatchNormalization
batch_normalization_16: BatchNormalization
conv2d_14: Conv2D
batch_normalization_12: BatchNormalization
activation_16: Activation
batch_normalization_14: BatchNormalization
conv2d_17: Conv2D
activation_12: Activation
batch_normalization_17: BatchNormalization
activation_14: Activation
activation_17: Activation
activation_18: Activation
mixed1: Concatenate
conv2d_22: Conv2D
conv2d_20: Conv2D
batch_normalization_22: BatchNormalization
average_pooling2d_2: AveragePooling2D
conv2d_19: Conv2D
activation_22: Activation
batch_normalization_20: BatchNormalization
conv2d_25: Conv2D
conv2d_23: Conv2D
activation_20: Activation
batch_normalization_25: BatchNormalization
batch_normalization_23: BatchNormalization
conv2d_21: Conv2D
batch_normalization_19: BatchNormalization
activation_23: Activation
batch_normalization_21: BatchNormalization
conv2d_24: Conv2D
activation_19: Activation
batch_normalization_24: BatchNormalization
activation_21: Activation
activation_24: Activation
activation_25: Activation
mixed2: Concatenate
conv2d_27: Conv2D
batch_normalization_27: BatchNormalization
conv2d_26: Conv2D
activation_27: Activation
batch_normalization_26: BatchNormalization
conv2d_28: Conv2D
activation_26: Activation
batch_normalization_28: BatchNormalization
max_pooling2d_2: MaxPooling2D
activation_28: Activation
conv2d_29: Conv2D
batch_normalization_29: BatchNormalization
activation_29: Activation
mixed3: Concatenate


conv2d_29: Conv2D
batch_normalization_29: BatchNormalization
activation_29: Activation
mixed3: Concatenate
conv2d_34: Conv2D
conv2d_31: Conv2D
batch_normalization_34: BatchNormalization
average_pooling2d_3: AveragePooling2D
conv2d_30: Conv2D
activation_34: Activation
batch_normalization_31: BatchNormalization
conv2d_39: Conv2D
conv2d_35: Conv2D
activation_31: Activation
batch_normalization_39: BatchNormalization
batch_normalization_35: BatchNormalization
conv2d_32: Conv2D
batch_normalization_30: BatchNormalization
activation_35: Activation
batch_normalization_32: BatchNormalization
conv2d_36: Conv2D
activation_32: Activation
batch_normalization_36: BatchNormalization
conv2d_33: Conv2D
activation_36: Activation
batch_normalization_33: BatchNormalization
conv2d_37: Conv2D
activation_30: Activation
batch_normalization_37: BatchNormalization
activation_33: Activation
activation_37: Activation
activation_39: Activation
conv2d_38: Conv2D
batch_normalization_38: BatchNormalization
activation_38: Activation
mixed4: Concatenate
conv2d_44: Conv2D
conv2d_41: Conv2D
batch_normalization_44: BatchNormalization
average_pooling2d_4: AveragePooling2D
conv2d_40: Conv2D
activation_44: Activation
batch_normalization_41: BatchNormalization
conv2d_49: Conv2D
conv2d_45: Conv2D
activation_41: Activation
batch_normalization_49: BatchNormalization
batch_normalization_45: BatchNormalization
conv2d_42: Conv2D
batch_normalization_40: BatchNormalization
activation_45: Activation
batch_normalization_42: BatchNormalization
conv2d_46: Conv2D
activation_42: Activation
batch_normalization_46: BatchNormalization
conv2d_43: Conv2D
activation_46: Activation
batch_normalization_43: BatchNormalization
conv2d_47: Conv2D
activation_40: Activation
batch_normalization_47: BatchNormalization
activation_43: Activation
activation_47: Activation
activation_49: Activation
conv2d_48: Conv2D
batch_normalization_48: BatchNormalization
activation_48: Activation
mixed5: Concatenate
conv2d_54: Conv2D
conv2d_51: Conv2D


conv2d_48: Conv2D
batch_normalization_48: BatchNormalization
activation_48: Activation
mixed5: Concatenate
conv2d_54: Conv2D
conv2d_51: Conv2D
batch_normalization_54: BatchNormalization
average_pooling2d_5: AveragePooling2D
conv2d_50: Conv2D
activation_54: Activation
batch_normalization_51: BatchNormalization
conv2d_59: Conv2D
conv2d_55: Conv2D
activation_51: Activation
batch_normalization_59: BatchNormalization
batch_normalization_55: BatchNormalization
conv2d_52: Conv2D
batch_normalization_50: BatchNormalization
activation_55: Activation
batch_normalization_52: BatchNormalization
conv2d_56: Conv2D
activation_52: Activation
batch_normalization_56: BatchNormalization
conv2d_53: Conv2D
activation_56: Activation
batch_normalization_53: BatchNormalization
conv2d_57: Conv2D
activation_50: Activation
batch_normalization_57: BatchNormalization
activation_53: Activation
activation_57: Activation
activation_59: Activation
conv2d_58: Conv2D
batch_normalization_58: BatchNormalization
activation_58: Activation
mixed6: Concatenate
conv2d_64: Conv2D
conv2d_61: Conv2D
batch_normalization_64: BatchNormalization
average_pooling2d_6: AveragePooling2D
conv2d_60: Conv2D
activation_64: Activation
batch_normalization_61: BatchNormalization
conv2d_69: Conv2D
conv2d_65: Conv2D
activation_61: Activation
batch_normalization_69: BatchNormalization
batch_normalization_65: BatchNormalization
conv2d_62: Conv2D
batch_normalization_60: BatchNormalization
activation_65: Activation
batch_normalization_62: BatchNormalization
conv2d_66: Conv2D
activation_62: Activation
batch_normalization_66: BatchNormalization
conv2d_63: Conv2D
activation_66: Activation
batch_normalization_63: BatchNormalization
conv2d_67: Conv2D
activation_60: Activation
batch_normalization_67: BatchNormalization
activation_63: Activation
activation_67: Activation
activation_69: Activation
conv2d_68: Conv2D
batch_normalization_68: BatchNormalization
activation_68: Activation
mixed7: Concatenate
conv2d_72: Conv2D


activation_68: Activation
mixed7: Concatenate
conv2d_72: Conv2D
batch_normalization_72: BatchNormalization
conv2d_70: Conv2D
activation_72: Activation
batch_normalization_70: BatchNormalization
conv2d_73: Conv2D
activation_70: Activation
batch_normalization_73: BatchNormalization
conv2d_71: Conv2D
activation_73: Activation
batch_normalization_71: BatchNormalization
conv2d_74: Conv2D
activation_71: Activation
batch_normalization_74: BatchNormalization
max_pooling2d_3: MaxPooling2D
activation_74: Activation
conv2d_75: Conv2D
batch_normalization_75: BatchNormalization
activation_75: Activation
mixed8: Concatenate
conv2d_80: Conv2D
conv2d_77: Conv2D
average_pooling2d_7: AveragePooling2D
batch_normalization_80: BatchNormalization
batch_normalization_77: BatchNormalization
conv2d_84: Conv2D
conv2d_76: Conv2D
activation_80: Activation
activation_77: Activation
batch_normalization_84: BatchNormalization
conv2d_81: Conv2D
conv2d_78: Conv2D
conv2d_79: Conv2D
batch_normalization_81: BatchNormalization
batch_normalization_78: BatchNormalization
batch_normalization_79: BatchNormalization
activation_81: Activation
activation_78: Activation
activation_79: Activation
conv2d_82: Conv2D
conv2d_83: Conv2D
mixed9_0: Concatenate
batch_normalization_82: BatchNormalization
batch_normalization_83: BatchNormalization
batch_normalization_76: BatchNormalization
activation_82: Activation
activation_83: Activation
activation_84: Activation
concatenate: Concatenate
activation_76: Activation
mixed9: Concatenate


concatenate: Concatenate
activation_76: Activation
mixed9: Concatenate
conv2d_89: Conv2D
conv2d_86: Conv2D
average_pooling2d_8: AveragePooling2D
batch_normalization_89: BatchNormalization
batch_normalization_86: BatchNormalization
conv2d_93: Conv2D
conv2d_85: Conv2D
activation_89: Activation
activation_86: Activation
batch_normalization_93: BatchNormalization
conv2d_90: Conv2D
conv2d_87: Conv2D
conv2d_88: Conv2D
batch_normalization_90: BatchNormalization
batch_normalization_87: BatchNormalization
batch_normalization_88: BatchNormalization
activation_90: Activation
activation_87: Activation
activation_88: Activation
conv2d_91: Conv2D
conv2d_92: Conv2D
mixed9_1: Concatenate
batch_normalization_91: BatchNormalization
batch_normalization_92: BatchNormalization
batch_normalization_85: BatchNormalization
activation_91: Activation
activation_92: Activation
activation_93: Activation
concatenate_1: Concatenate
activation_85: Activation
mixed10: Concatenate
global_average_pooling2d: GlobalAveragePooling2D
dropout: Dropout
dense: Dense
dropout_1: Dropout
dense_1: Dense
dropout_2: Dropout
dense_2: Dense

Appendice B

Codice sorgente

```
# -*- coding: utf-8 -*-
"""V3 7 Features.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1hJTB7xhoeDHUY_g-zL1VK-1siZOgqANc
"""

!pip install tensorflow==2.1 > /dev/null
import json
import os
import cv2
import tensorflow as tf
from tensorflow import keras
from os import path
import pandas as pd
import shutil
import pickle
import numpy as np
from PIL import Image as pil_image
from sklearn.metrics import confusion_matrix
import keras
from tensorflow.python.keras.callbacks import EarlyStopping, ModelCheckpoint, History
from tensorflow.keras.utils import Sequence
from tensorflow.python.keras.applications.inception_v3 import InceptionV3
from tensorflow.python.keras.models import Sequential, load_model
from tensorflow.python.keras.layers import Dense, Dropout
from keras import optimizers
from keras import regularizers
import matplotlib.pyplot as plt
from google.colab import drive
```

```
kaggle_dataset_name = "skin-cancer-mnist-ham10000"

kaggle_dataset_owner = "kmader"


base_path = "/content/drive/My Drive/Colab Notebooks"

model_checkpoint_path = f"{base_path}/Inception 7 Features/Saved Models/Checkpoint_7_Featrs_V3_Last10.h5"

saved_model_path = f"{base_path}/Inception 7 Features/Saved Models/Model_7_Featrs_V3_Last10.h5"


dataset=pd.read_pickle('/content/drive/My Drive/Colab Notebooks/Structured Code/Pickle')


dataset


def train_test_validation_split(df, class_column, test_frac=0.20, val_frac=0.20, subsample_class=None, subsample_to_keep=None):

    tmp_df = df

    if subsample_class is not None and subsample_to_keep is not None:

        tmp_df = subsample_df(tmp_df, class_column, subsample_class, subsample_to_keep)

    test_data = tmp_df.groupby(class_column, group_keys=False).apply(pd.DataFrame.sample, frac=test_frac)

    train_data = tmp_df.drop(test_data.index)

    validation_data = train_data.groupby(class_column, group_keys=False).apply(pd.DataFrame.sample, frac=val_frac)

    train_data = train_data.drop(validation_data.index)

    return train_data, validation_data, test_data


train_data, validation_data, test_data = train_test_validation_split(

    dataset,

    class_column="Class"

)
```

```
labels = 'Train Set', 'Validation Set', 'Test Set'
sizes = [train_data['image_id'].nunique(), validation_data['image_id'].nunique(),
test_data['image_id'].nunique()]
colors = ['yellowgreen', 'lightcoral', 'lightskyblue']
explode = (0.1, 0, 0)  # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()

import plotly.express as px
fig = px.histogram(train_data, x="Class", title='Istogramma popolosità delle classi')
fig.show()

train_datagen = keras.preprocessing.image.ImageDataGenerator(
    rescale= 1.0/255,
    shear_range= 0.5,
    brightness_range=(0.2, 1.0),
    zoom_range=(0.7, 1.0),
    fill_mode='reflect',
    horizontal_flip=True,
    vertical_flip=True,
    featurewise_std_normalization=True,
    featurewise_center= True,
)
```

```
validation_datagen = keras.preprocessing.image.ImageDataGenerator(
    featurewise_std_normalization=True,
    featurewise_center= True,
    rescale= 1.0/255
)

image_target_shape=(300,250)

train_datagen.fit(list(dataset['images']))
validation_datagen.fit(list(dataset['images']))

train_dataframe_iterator = train_datagen.flow_from_dataframe(
    train_data,
    batch_size=32,
    class_mode="sparse",
    x_col="image_id",
    y_col="Class",
    seed=1234,
    target_size=image_target_shape,
    interpolation="hamming"

)

validation_dataframe_iterator = validation_datagen.flow_from_dataframe(
    validation_data,
    batch_size=32,
    class_mode="sparse",
    x_col="image_id",
    y_col="Class",
    seed=1234,
    target_size=image_target_shape,
    interpolation='hamming')
```

```
test_dataframe_iterator = validation_datagen.flow_from_dataframe(
        test_data,
        batch_size=32,
        class_mode="sparse",
        x_col="image_id",
        y_col="Class",
        seed=1234,
        target_size=image_target_shape,
        shuffle= False
)

input_shape = (300, 250, 3)

num_labels = 7

model=tf.keras.Sequential([
  InceptionV3(include_top=False, input_shape=(250, 150, 3),pooling = 'avg', weights = 'imagenet'),
  Dropout(0.5),
  Dense(128, activation="relu",kernel_regularizer=regularizers.l2(0.02)),
  Dropout(0.4),
  Dense(32, activation="relu",kernel_regularizer=regularizers.l2(0.02)),
  Dropout(0.3),
  Dense(num_labels, activation = 'softmax',kernel_regularizer=regularizers.l2(0.02))
])

for layer in model.layers:
    layer.trainable = True
model.summary()

model.compile(optimizer = 'adadelta', loss = "sparse_categorical_crossentropy", metrics=["accuracy"])
```

```
# Fit the model
# Need to be fitted in 4/5 different trials due to the overload of RAM---> 80 *5= 400 epochs
#Prime 80 Andate.
#Seconde 80 andate, val acc 0.6964
#Terze 32 andate, val acc 07359(best)
#Quarte 80    77.91
#ultime 40    loss: 2.9065 - accuracy: 0.7872 - val_loss: 2.7888 - val_accuracy: 0.8200
#Altre 60     loss: 2.5599 - accuracy: 0.8161 - val_loss: 2.4857 - val_accuracy: 0.8376
#next 44      loss: 2.3798 - accuracy: 0.8100 - val_loss: 2.2773 - val_accuracy: 0.8405
#Last 40      loss: 2.1491 - accuracy: 0.8354 - val_loss: 2.0276 - val_accuracy: 0.8720
#Ultimate 47  loss: 1.9119 - accuracy: 0.8565 - val_loss: 1.8181 - val_accuracy: 0.8800
#Last 2
epochs = 10
history=model.fit(train_dataframe_iterator,
  epochs = epochs,
  validation_data = validation_dataframe_iterator,
  verbose = 1,
  callbacks=[
     tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0.001, patience=10,
restore_best_weights=True),
     tf.keras.callbacks.ModelCheckpoint(filepath=model_checkpoint_path, monitor = 'val_loss', patience = 5,
save_best_only=True),
     tf.keras.callbacks.ReduceLROnPlateau(
       monitor='val_loss',
       patience=3,
       verbose=1,
       factor=0.02,
       min_lr=0.001,
       save_best_only = True,
       mode = 'auto' ) ],
  workers=10)
```

```
model.save(saved_model_path)



from tensorflow.keras.callbacks import  History
#history.history

#1. Function to plot model's validation loss and validation accuracy
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(range(1,len(history.history['accuracy'])+1),history.history['accuracy'])
    axs[0].plot(range(1,len(history.history['val_accuracy'])+1),history.history['val_accuracy'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(history.history['accuracy'])+1),len(history.history['accuracy'])/10)
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(history.history['loss'])+1),history.history['loss'])
    axs[1].plot(range(1,len(history.history['val_loss'])+1),history.history['val_loss'])
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(history.history['loss'])+1),len(history.history['loss'])/10)
    axs[1].legend(['train', 'val'], loc='best')
    plt.show()
model=load_model(model_checkpoint_path)
```

```
#Previous model loss: 2.9323 - accuracy: 0.7696 test_accuracy = 0.769591  ;  test_loss = 2.932310
#Second model   loss: 2.6112 - accuracy: 0.7772 test_accuracy = 0.777193  ;  test_loss = 2.611180
          # loss: 2.0965 - accuracy: 0.8433 test_accuracy = 0.843275  ;  test_loss = 2.096517
test_loss, test_acc = model.evaluate(test_dataframe_iterator, verbose=1, workers =10)
print("test_accuracy = %f  ;  test_loss = %f" % (test_acc, test_loss))

#train_pred = model.predict(train_dataframe_iterator)
#train_pred_classes = np.argmax(train_pred,axis = 1)
test_pred = model.predict(test_dataframe_iterator)
# Convert predictions classes to one hot vectors
test_pred

test_pred_classes = np.argmax(test_pred,axis = 1)

test_pred_classes

np.unique(test_pred_classes)

test_dataframe_iterator.class_indices

cat_pred_classes=test_data['Class'].map(lambda x: test_dataframe_iterator.class_indices[x])

cat_pred_classes
```

```
def plot_confusion_matrix2(cm,
                           target_names,
                           title='Confusion matrix',
                           cmap=None,
                           normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot
    Arguments
        cm:           confusion matrix from sklearn.metrics.confusion_matrix
    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']
    title:        the text to display at the top of the matrix
    cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
                  see http://matplotlib.org/examples/color/colormaps_reference.html
                  plt.get_cmap('jet') or plt.cm.Blues
    normalize:    If False, plot the raw numbers
                  If True, plot the proportions
    Usage
    plot_confusion_matrix(cm           = cm,                  # confusion matrix created by
                                       # sklearn.metrics.confusion_matrix
                          normalize    = True,                # show proportions
                          target_names = y_labels_vals,       # list of names of the classes
                          title        = best_estimator_name) # title of graph
    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy
    if cmap is None:
        cmap = plt.get_cmap('Wistia')
```

```
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()

if target_names is not None:
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names, rotation=45)
    plt.yticks(tick_marks, target_names)

if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]



thresh = cm.max() / 1.5 if normalize else cm.max() / 2
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    if normalize:
        plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                 horizontalalignment="center",
                 color="black" if cm[i, j] > thresh else "black")
    else:
        plt.text(j, i, "{:,}".format(cm[i, j]),
                 horizontalalignment="center",
                 color="black" if cm[i, j] > thresh else "black")



plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))
plt.show()
```

```
# Previous array
   #([[  3,  12,  58,   0,  15,   0,   0],
#      [  0, 101,  45,   0,  10,   5,   0],
#      [  0,   3, 181,   0,  22,  31,   0],
#      [  0,   8,  20,   0,   0,   8,   0],
#      [  0,   4,  69,   0, 326,  48,   0],
#      [  0,   3,   3,   0,   6, 688,   0],
#      [  0,  10,   4,   0,   0,  10,  17]])

#Second array
    #([[ 10,  12,  57,   0,   7,   2,   0],
#      [  0, 111,  40,   0,   5,   5,   0],
#      [  0,   4, 182,   0,  22,  29,   0],
#      [  0,  10,  20,   1,   1,   2,   2],
#      [  3,   3,  82,   0, 314,  44,   1],
#      [  0,   0,   5,   0,   8, 687,   0],
#      [  0,  11,   0,   0,   2,   4,  24]]),

#Third array
     #[[ 16,  12,  46,   0,  13,   1,   0],
#      [  1, 129,  24,   0,   4,   0,   3],
#      [  1,   6, 200,   0,  19,  11,   0],
#      [  0,   6,  20,   0,   0,   6,   4],
#      [  2,   3,  46,   0, 354,  42,   0],
#      [  0,   0,   5,   0,   4, 691,   0],
#      [  0,   5,   0,   0,   0,   2,  34]
```

```
#array([[ 21,  10,  41,   0,  16,   0,   0],
#       [  3, 125,  23,   0,   6,   2,   2],
#       [  1,   5, 201,   0,  16,  14,   0],
#       [  1,   5,  20,   0,   2,   5,   3],
#       [  1,   5,  36,   0, 366,  39,   0],
#       [  0,   0,   3,   0,   0, 696,   1],
#       [  0,   4,   0,   0,   2,   2,  33]])



#array([[ 18,   7,  48,   0,  15,   0,   0],
#       [  4, 136,  12,   0,   6,   2,   1],
#       [  0,   7, 206,   0,  12,  11,   1],
#       [  2,   6,  23,   2,   1,   1,   1],
#       [  0,   2,  42,   0, 367,  36,   0],
#       [  0,   0,   4,   0,   3, 693,   0],
#       [  0,   2,   0,   0,   0,   4,  35]])



import seaborn as sns
from sklearn.metrics import classification_report, plot_confusion_matrix
#confusionmatrix =confusion_matrix(cat_pred_classes, test_pred_classes)
#confusionmatrix



a=(30+8+4)+(0+121+7)+(9+5+380)
b=42+23+34+0+0+0+0+2+29+0+2+0
c=0+2+0+0+1+4+0+3+8+1+1+0
d=216+0+12+0+21+2+5+1+7+0+692+0+0+0+2+36
a+b+c+d
```

```
plot_confusion_matrix2(cm=np.asarray([[ 30,   7,  42,   0,   9,   0,   0],
     [  8, 121,  23,   0,   5,   2,   2],
      [  0,   1, 216,   0,   8,  12,   0],
      [  2,   4,  21,   2,   1,   5,   1],
      [  4,   0,  34,   0, 380,  29,   0],
      [  0,   0,   7,   0,   1, 692,   0],
      [  0,   3,   0,   0,   0,   2,  36]],),
                  target_names=['actin keratosis', 'basal cell carcinoma', 'sebohrreic keratosis',\
                              'dermatofibroma','melanoma','nevus','vascular lesion'],
                 normalize= False)

#Previous statistics
#                  precision    recall  f1-score   support

#    actin keratosis       1.00      0.03      0.07        88
#basal cell carcinoma       0.72      0.63      0.67       161
#   benign keratosis       0.48      0.76      0.59       237
#    dermatofibroma       0.00      0.00      0.00        36
#          melanoma       0.86      0.73      0.79       447
#             nevus       0.87      0.98      0.92       700
#   vascular lesion       1.00      0.41      0.59        41

#          accuracy                           0.77      1710
#         macro avg       0.70      0.51      0.52      1710
#      weighted avg       0.79      0.77      0.75      1710

#                  precision    recall  f1-score   support

#    actin keratosis       0.77      0.11      0.20        88
#basal cell carcinoma       0.74      0.69      0.71       161
#   benign keratosis       0.47      0.77      0.58       237
#    dermatofibroma       1.00      0.03      0.05        36
```

```
#          melanoma      0.87      0.70      0.78       447
#             nevus      0.89      0.98      0.93       700
#    vascular lesion      0.89      0.59      0.71        41

#          accuracy                           0.78      1710
#         macro avg      0.80      0.55      0.57      1710
#      weighted avg      0.81      0.78      0.76      1710

#                precision    recall  f1-score   support

#    actin keratosis      0.80      0.18      0.30        88
#basal cell carcinoma      0.80      0.80      0.80       161
#   benign keratosis      0.59      0.84      0.69       237
#    dermatofibroma      0.00      0.00      0.00        36
#          melanoma      0.90      0.79      0.84       447
#             nevus      0.92      0.99      0.95       700
#    vascular lesion      0.83      0.83      0.83        41

#          accuracy                           0.83      1710
#         macro avg      0.69      0.63      0.63      1710
#      weighted avg      0.83      0.83      0.82      1710

#                precision    recall  f1-score   support
```

```
#     actin keratosis       0.78      0.24      0.37       88
#basal cell carcinoma       0.81      0.78      0.79      161
#    benign keratosis       0.62      0.85      0.72      237
#     dermatofibroma        0.00      0.00      0.00       36
#           melanoma        0.90      0.82      0.86      447
#              nevus        0.92      0.99      0.95      700
#     vascular lesion       0.85      0.80      0.83       41

#           accuracy                           0.84     1710
#          macro avg       0.70      0.64      0.64     1710
#       weighted avg       0.83      0.84      0.83     1710



#     actin keratosis       0.75      0.20      0.32       88
#basal cell carcinoma       0.85      0.84      0.85      161
#    benign keratosis       0.61      0.87      0.72      237
#     dermatofibroma        1.00      0.06      0.11       36
#           melanoma        0.91      0.82      0.86      447
#              nevus        0.93      0.99      0.96      700
#     vascular lesion       0.92      0.85      0.89       41

#           accuracy                           0.85     1710
#          macro avg       0.85      0.66      0.67     1710
#       weighted avg       0.86      0.85      0.84     1710

from sklearn.metrics import classification_report

label=test_data['Class'].nunique()
# Generate a classification report
#trainreport = classification_report(trainlabels, train_pred_classes, target_names=list(labels))
```

```
testreport = classification_report(cat_pred_classes, test_pred_classes, target_names=['actin keratosis', 'basal cell carcinoma', 'benign keratosis',\
                                'dermatofibroma','melanoma','nevus','vascular lesion'])
#print(trainreport)
print(testreport)

#[[555, 143],
#[32, 980]]

plot_confusion_matrix2(cm=np.asarray([[a, b],
                                [c, d]]),
                target_names=['lesioni maligne', 'lesioni beningne'])

precisione=a/(a+c)
precisione

Rcall=a/(a+b)
Rcall

F1=2*precisione*Rcall/(Rcall+precisione)
F1

num=a*d-b*c
den=(a+c)*(a+b)*(d+c)*(d+b)

MCC=num*num/(den)

MCC
```

Bibliography

[1] Renato Marchiori Bakos, Tatiana Pinto Blumetti, Rodrigo Roldán-Marín, and Gabriel Salerni. «Noninvasive Imaging Tools in the Diagnosis and Treatment of Skin Cancers». In: *Am. J. Clin. Dermatol.* 19.s1 (2018), pp. 3–14. ISSN: 11791888. DOI: 10.1007/s40257-018-0367-4. URL: https://doi.org/10.1007/s40257-018-0367-4 (cit. on p. 2).

[2] R Marchesini and M Carrara. «L'maging spettrofotometrico de lesioni cutanee pigmentate». In: *Alla ricerca non invasiva delle caratteristiche biofisiche di una lesione. hitech dermo* 3 (2007), pp. 43–50 (cit. on pp. 4, 8).

[3] Luigi Raimondo D'Ottavi. «XXXIII Convegno Nazionale di Aggiornamento: I tumori cutanei maligni di interesse ORL». In: (2009), pp. 1–351. URL: http://www.aooi.it/contents/attachment/c11/RU{_}aooi{_}2009.pdf (cit. on pp. 5, 7).

[4] *Manuale di dermatologia AIMS 2020* (cit. on p. 5).

[5] M. B. Lens and M. Dawes. «Global perspectives of contemporary epidemiological trends of cutaneous malignant melanoma». In: *Br. J. Dermatol.* 150.2 (2004), pp. 179–185. ISSN: 00070963. DOI: 10.1111/j.1365-2133.2004.05708.x (cit. on p. 6).

[6] Ralph Peter Braun, Harold S. Rabinovitz, Joachim Krischer, Jürgen Kreusch, Margaret Oliviero, Luigi Naldi, Alfred W. Kopf, and Jean H. Saurat. «Dermoscopy of pigmented seborrheic keratosis: A morphological study». In: *Arch. Dermatol.* 138.12 (2002), pp. 1556–1560. ISSN: 0003987X. DOI: 10.1001/archderm.138.12.1556 (cit. on p. 6).

[7] Davide Devescovi. «Impiego di classificatori nell'analisi di immagini». In: *Neural Networks* 14.2 (2010), pp. 1–15 (cit. on pp. 26, 28, 37).

[8] Giovanni Bianco. «Markov Random Field Teoria e applicabilità nell ' elaborazione delle immagini». In: (1998), pp. 1–90 (cit. on p. 31).

[9] Clifford Hammersley J.M. «Markov Field on Finite Graphs and Lattices.» In: () (cit. on p. 32).

[10] S Egmentation, Dzung L Pham, Chenyang Xu, and Jerry L Prince. «C m m i s». In: (2000) (cit. on p. 32).

[11] Segnali Multimediali. «La segmentazione». In: (2010), pp. 1–18 (cit. on p. 37).

[12] D I Scienze Matematiche and Fisiche E Naturali. «‘ degli Studi “‘ La Sapienza ” di Universit a Algoritmi bayesiani e adattivi per la segmentazione delle immagini». In: (2017) (cit. on p. 37).

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep residual learning for image recognition». In: *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* 2016-Decem (2016), pp. 770–778. ISSN: 10636919. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385 (cit. on p. 43).

[14] Rishu Garg, Saumil Maheshwari, and Anupam Shukla. «Decision Support System for Detection and Classification of Skin Cancer using CNN». In: (2019), pp. 1–9. arXiv: 1912.03798. URL: http://arxiv.org/abs/1912.03798 (cit. on p. 44).

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. «Rethinking the Inception Architecture for Computer Vision». In: *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* 2016-Decem (2016), pp. 2818–2826. ISSN: 10636919. DOI: 10.1109/CVPR.2016.308. arXiv: 1512.00567 (cit. on p. 46).

[16] D. A. Gavrilov, N. N. Shchelkunov, and A. V. Melerzanov. «Deep learning based skin lesions diagnosis». In: *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.* 42.2/W12 (2019), pp. 81–85. ISSN: 16821750. DOI: 10.5194/isprs-archives-XLII-2-W12-81-2019 (cit. on p. 49).

[17] Yu-Min Chung, Chuan-Shen Hu, Austin Lawson, and Clifford Smyth. «TopoResNet: A hybrid deep learning architecture and its application to skin lesion classification». In: May (2019). arXiv: 1905.08607. URL: http://arxiv.org/abs/1905.08607 (cit. on pp. 87–89).