**Università degli studi di Padova**

SCUOLA DI INGEGNERIA

Corso di Laurea Magistrale in Bioingegneria

TESI DI LAUREA MAGISTRALE

# Conta cellulare su immagini di endotelio corneale mediante utilizzo di reti neurali

Candidato:
**Giovanni Nicoletti**
**Matricola 1179549**

Relatore:
**Enrico Grisan**

**Anno Accademico 2018/2019**

Alla mia famiglia.

# Sommario

Il presente elaborato tratterà algoritmi in grado di stimare, in modo automatico e per mezzo di regressione la quantità di cellule presenti nell'endotelio corneale a partire da immagini acquisite in vivo grazie alle tecniche di microscopia confocale o speculare. Conteggiare il numero di oggetti presenti in un'immagine è relativamente semplice per gli esseri umani, ma può essere molto complesso per algoritmi automatici, specialmente quando diverse istanze di uno stesso oggetto possono variare significativamente in termini di forma, colore o dimensione. In casi come questi, gli algoritmi di apprendimento automatico sono molto utili ed i più efficaci. Lo sviluppo di algoritmi automatici è utile anche al fine di evitare risultati soggettivi che dipendono dall'operatore che effettua l'esame [1].
Lo stesso obbiettivo può essere perseguito attraverso due approcci concettualmente differenti. Il primo approccio consiste nell'individuare prima tutti gli oggetti di interesse presenti nell'immagine e poi contarli. Il secondo approccio, invece, si avvale dell'utilizzo di oggetti puntiformi per individuare la posizione di un oggetto. Questo secondo approccio sarà quello utilizzato nel corso di questa tesi, in quanto fornisce risultati promettenti pur essendo meno oneroso dal punto di vista computazionale [2].
L' algoritmo di conta cellulare verrà implementato mediante l'utilizzo di reti neurali convoluzionali, mentre il training verrà effettuato con immagini che utilizzano annotazioni puntiformi per segnalare la presenza di un oggetto da conteggiare.
Nel corso dell'elaborato verrà fornita un'introduzione relativa all'endotelio corneale e alle modalità con cui vengono acquisite questo tipo di immagini. In seguito, ci sarà un'introduzione dettagliata alla teoria di apprendimento automatico con particolare riferimento alle reti neurali e reti neurali convoluzionali. Verranno poi illustrate alcune tra le tecniche attualmente più utilizzate per effettuare il conteggio cellulare che saranno in seguito applicate al problema esaminato in questo elaborato.
Dopo una attenta descrizione del setup sperimentale utilizzato, verranno infine confrontati i diversi approcci in termini di accuratezza, misurata in termini di errore quadratico medio.

## Indice

# Elenco delle figure

# Capitolo 1

# L’endotelio corneale

## 1.1 L’occhio

L’occhio è un complesso organo di senso (*figura 1.1*). Al suo interno e quindi ben protetti dal suo involucro, si trovano uno strato di recettori, una lente e un sistema di neuroni per la conduzione degli impulsi dai recettori al cervello. La sclera, strato protettivo esterno del globo oculare, si modifica anteriormente a formare la cornea trasparente, attraverso la quale entrano i raggi luminosi. All’interno della sclera si trova la coroide, strato pigmentato ricco di vasi sanguigni per la nutrizione delle strutture oculari. I due terzi posteriori della coroide sono rivestiti dalla retina, tessuto nervoso contenente le cellule recettrici dalla quale ha origine il nervo ottico. La faccia posteriore della retina, la più estesa, è disposta regolarmente sulla faccia profonda della coroide, ed essendo responsabile della funzione visiva prende il nome di parte ottica della retina. La faccia anteriore, però, non contiene elementi di natura nervosa e prende il nome di parte cieca della retina visto che non contribuisce alla formazione del nervo ottico. I principali componenti della retina sono i bastoncelli, i coni, le cellule bipolari e le cellule gangliari. I coni e i bastoncelli sono i fotorecettori. I loro prolungamenti centripeti terminano con sinapsi sui processi delle cellule bipolari. Queste, a loro volta si connettono, mediante sinapsi, con le cellule gangliari. Gli assoni delle cellule gangliari riunendosi formano il nervo ottico. Lo strato ricettivo della retina riposa sulla coroide, e i raggi luminosi devono attraversare le cellule gangliari per poter raggiungere i coni e i bastoncelli. Lo strato della coroide adiacente alla retina è pigmentato e assorbe la luce, impedendo che questa venga riflessa attraverso la retina. Il punto di uscita del nervo ottico e di entrata dei vasi sanguigni retinici è situato a 3 *mm* all’interno e un po' superiormente al polo posteriore del globo oculare. Questa regione si chiama papilla, o disco, del nervo ottico; è priva di fotorecettori, ed è pertanto cieca (punto cieco). Al polo posteriore dell’occhio si trova la macula lutea, e al centro di questa la fovea centrale, meno spessa, priva di bastoncelli, nella quale i coni sono più addensati, le cellule sono scarse e mancano i vasi sanguigni. La fovea è altamente sviluppata nell’uomo; in essa l’acuità visiva è massima. Quando, infatti, un oggetto attrae la nostra attenzione, gli occhi si dirigono in modo che i raggi luminosi provenienti dall’oggetto vadano a cadere sulla fovea. La lente o cristallino è una struttura molto trasparente e si attacca al corpo ciliare, che è il bordo anteriore, ispessito, della coroide. Davanti alla lente sta l’iride, che contiene fibre muscolari circolari e longitudinale che restringono e, rispettivamente, dilatano la pupilla, potendo variare in tal modo sino a 5 volte la quantità di luce che va alla retina. La camera anteriore dell’occhio, situata fra la cornea e la lente, è piena di umor acqueo. Lo spazio fra la lente e la retina è ripieno di una sostanza incolore, gelatinosa, chiamata umor vitreo.

Cornea, umor acqueo, cristallino e umor vitreo costituiscono l'apparato diottrico dell'occhio, che può essere considerato come una lente convergente dotata di ottimo potere refrattivo [3].

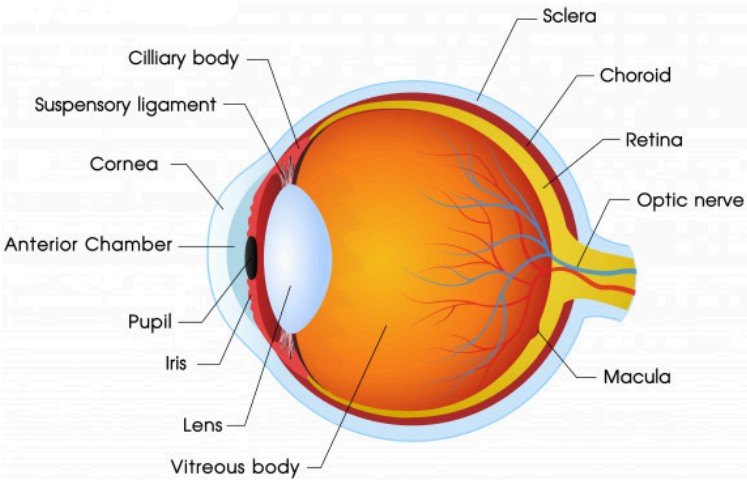


*Figura 1.1: Principali componenti dell'occhio [4]*

## 1.2 La cornea

La cornea è la più potente lente dell'intero apparato visivo, con un potere totale di 48 diottrie nella sua superficie anteriore convessa e -5 diottrie in quella concava. La cornea ha due funzioni principali, che la rendono un elemento fondamentale per la nostra vista: essa permette il passaggio della luce dall'esterno focalizzando i raggi verso la fovea e protegge il bulbo oculare da agenti esterni o da corpi estranei.
La cornea ha in tutto e per tutto la forma di una lente: è infatti convessa anteriormente e concava posteriormente, con raggio di curvatura anteriore maggiore di quello posteriore. La superficie anteriore ha la forma di un'ellissi, l'area centrale è approssimativamente sferica e la sua curvatura non è regolare. È inoltre trasparente, incolore, speculare e priva di vasi sanguigni.

La cornea, schematizzata in *figura 1.2*, è un tessuto composto da cinque strati con diverse composizioni. Questi sono, dall'esterno verso l'interno:

- Epitelio corneale
- Membrana di Bowman
- Stroma corneale
- Membrana di Descemet
- Endotelio

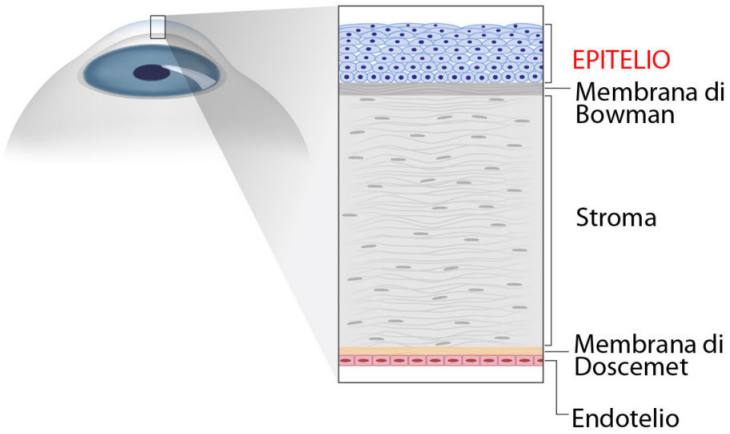



*Figura 1.2: Composizione della cornea [5]*

Ciascuno di questi ha una sua composizione e funzione:

**Epitelio corneale:** si tratta dello strato più esterno della cornea, a sua volta composto da 5-6 strati microscopici formati da cellule basali, le cellule intermedie e le cellule alari, e hanno la funzione di proteggere l'occhio da abrasione e aggressione di agenti o corpi esterni. Ha uno spessore di circa 50 $\mu m$.

**Lamina di Bowman:** detta anche membrana elastica anteriore, ha uno spessore di 12 $\mu m$ ed è uno strato privo di cellule, agglomerato di fibrille collagene che hanno funzione di strato elastico tra epitelio e stroma.

**Stroma corneale**: spesso 500 $\mu m$, è lo strato più esteso della cornea, composto per lo più da fibre collagene, il cui spessore aumenta dalla zona centrale verso quella esterna, oltre che naturalmente con l'età. Lo stroma compone la cornea per una percentuale dal 75% al 90% a seconda dei casi.

**Membrana di Descemet**: questa membrana, quarto tra gli strati che compongono la cornea, è molto simile alla Lamina di Bowman come composizione, composta anch'essa da fibre collagene che donano elasticità. Ha uno spessore variabile di 4-12 $\mu m$.

**Endotelio**: si tratta del quinto e più profondo strato della cornea, composto da un singolo strato di cellule piatte e dalla forma esagonale. Le sue cellule sono strettamente adese tra loro e la cui funzione è quella di filtro posteriore e di idratazione per tutti gli strati superiori della cornea. Nel paragrafo seguente verrà descritto in dettaglio.

A questi se ne aggiunge uno ulteriore, chiamato **Strato di Dua,** di recente scoperta. Si pone nella parte più interna della cornea ed essendo molto sottile, per essere analizzato è necessario il microscopio elettronico.

La cornea può essere affetta da malattie di tipo sistemico, che alterano morfologia e funzionalità di alcune delle sue strutture, o da patologie e distrofie, che vengono classificate in base alla loro localizzazione anatomica. Caratteristica principale di una cornea danneggiata è la comparsa di opacità che causa un deficit visivo che varia a seconda dell'entità del danno. La gestione delle distrofie corneali varia da caso a caso. In alcune circostanze si interviene farmacologicamente o chirurgicamente tramite la rimozione della parte danneggiata del tessuto corneale [6].

## 1.3 L'endotelio corneale

L'endotelio corneale è composto da uno strato cellulare di cellule piatte e poligonali, la cui densità media, in individui adulti, è 2700 $cellule/mm^2$. Ha uno spessore di circa 6 $\mu m$ e un'area di circa 250$\mu m^2$. Le cellule che compongono questo tessuto hanno una forma poligonale con un numero di lati che va da quattro a otto. Una cornea perfetta dovrebbe avere il 100% delle cellule con forma esagonale mentre, una cornea sana dovrebbe avere almeno il 60% di esse con forma esagonale. Lo schema di posa a nido d'ape produce la massima efficienza, in termini di perimetro totale ricoperto, in quanto l'esagono è il poligono con la più grande superficie in rapporto al suo perimetro. Durante l'infanzia le cellule endoteliali hanno tutte forma esagonale e dimensione omogenea; l'invecchiamento o l'eventuale presenza di patologie, come il diabete ad esempio, fanno perdere questa caratteristica di regolarità: le cellule che compongono l'endotelio corneale non sono in grado di riprodursi perciò quando una di questa muore, il suo posto viene occupato dalle cellule adiacenti che si spostano ed allargano. Quantificare i cambiamenti morfologici delle cellule endoteliali corneali fornisce perciò uno strumento utile al fine di valutare lo stato di salute della cornea. I parametri clinici valutati sono: densità cellulare, polimegatismo (distribuzione della dimensione cellulare) e il pleomorfismo (distribuzione dei lati delle cellule).
Se le cellule endoteliali sono in salute, esse forniscono nutrimento alla cornea. Queste cellule infatti permettono all'humor acqueo di entrare nella cornea. Dopo che le cellule della cornea sono state nutrite, le cellule pompano il liquido fuori dalla cornea. Se la pompa endoteliale è compromessa, la cornea verrà idratata troppo e diventerà opaca. Se la densità di cellule endoteliali scende tra 500 e 100 $cellule/mm$, il meccanismo di pompa non è più in grado di nutrire la cornea in modo tale da garantire la trasparenza ottica e ciò causa quello che viene chiamato edema corneale. Questa patologia causa un'alterazione della visione e può anche portare alla perdita della vista [7].

## 1.4 Importanza dello sviluppo di algoritmi automatici di conta endoteliale

La conta endoteliale è uno strumento che ci permette di studiare le cellule che compongono l'endotelio corneale verificando la vitalità, densità, forma, dimensioni e variabilità. I dati raccolti da questo tipo di esame permettono di verificare lo stato di

salute ed efficienza di queste cellule, sia in vista di interventi chirurgici, sia dopo l'esecuzione di interventi chirurgici con lo scopo di verificare la presenza di eventuali ricadute.
Effettuare un esame di conta endoteliale consiste nel fotografare, attraverso un'apposita fotocamera, la superficie più interna della cornea. Il paziente deve fissare per qualche istante una sorgente luminosa mentre la fotocamera esegue una serie di fotografie ed un computer elabora le informazioni ricevute.
La conta endoteliale fornisce un valido supporto anche in casi in cui sia necessario un intervento chirurgico per curare le malattie che affliggono la cornea ed è di fatto necessaria nel caso si dovesse effettuare un trapianto tramite la recente tecnica DMEK (Descemet Membrane Endothelial Keratoplasty) [8] in quanto permette valutare lo stato di salute del tessuto endoteliale del paziente e di quello proveniente dal donatore. Questa innovativa tecnica di trapianto permette di trapiantare solo le cellule danneggiate dell'endotelio corneale, lasciando l'occhio praticamente intatto. Si parla di un tessuto grande da 5 a 10 micron e ciò che ne consegue sono tempi di degenza ridotti di molto rispetto alle tecniche chirurgiche finora utilizzate, che prevedevano in trapianto dell'intera cornea.
Nei primi studi, la conta cellulare veniva eseguita a mano dal medico. Di recente però, la quantità di dati ottenuti tramite tecniche di imaging avanzate è diventata enorme. Diventa perciò impossibile eseguire manualmente la conta vista la grande quantità di oggetti di interesse presenti in ogni singola immagine. La necessità di sviluppare algoritmi automatici di conta cellulare diventa quindi ovvia. Altri problemi che un algoritmo automatico può risolvere sono la variabilità intra-operatore e la possibilità di comparare differenti insiemi di dati [9].

## 1.5 Acquisizione delle immagini:

L'acquisizione di immagini dell'endotelio corneale è possibile grazie alla microscopia confocale oppure grazie alla microscopia speculare, entrambe basate sul fenomeno di riflessione della luce ma che sfruttano diverse tecniche per l'ottenimento dell'immagine. In seguito, entrambe le tecniche verranno descritte.

### 1.5.1 Microscopia Speculare:

La microscopia speculare viene usate per vedere in modo non invasivo lo strato di cellule dell'endotelio corneale. Tutti i microscopi speculari utilizzati in ambito clinico sono basati sul design sviluppato da David Maurice nel 1968 e forniscono una vista notevolmente ingrandita della luce speculare riflessa dall'endotelio corneale. Il sistema di illuminazione e di osservazione è osservabile schematicamente in *figura 1.3*.

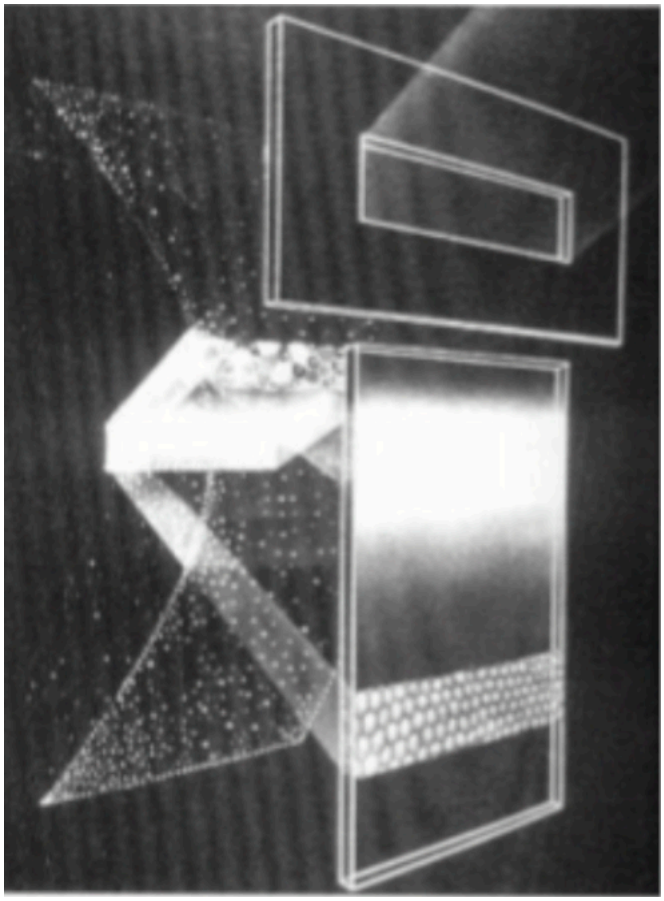


*Figura 1.3: Schema del sistema di illuminazione e visualizzazione nel microscopio speculare [10].*

Il riflesso speculare avviene all'interfaccia tra due superfici lisce, regolari e con indici di riflessione diversi. Le cellule endoteliali possono essere fotografate in quanto il loro indice di rifrazione è molto più elevato rispetto a quello dell'umor acqueo.
L'area superficiale del riflesso speculare dipende dal raggio di curvatura della superficie riflettente, perciò anche l'area speculare riflessa sarà influenzata da questo. Un ulteriore restrizione dell'area di luce riflessa è dovuta dalla vicinanza tra epitelio ed endotelio. La superficie epiteliale riflette molto a cause della grande differenza che esiste tra gli indici di rifrazione di aria e dell'epitelio. Quando il fascio luminoso attraversa la cornea, esso verrà riflesso all'interfaccia tra epitelio ed endotelio, perciò l'area visualizzabile sarà frutto di un compromesso tra la larghezza del fascio luminoso e lo spessore della cornea. A causa di questo fenomeno, l'area visualizzabile attraverso questo strumento sarà rettangolare la cui altezza sarà influenzata dal raggio di curvatura della cornea.
Gli strumenti da laboratorio che eseguono la microscopia speculare possono essere suddivisi in due categorie: quelli che richiedono il contatto tra lo strumento con lo strato epiteliale corneale e quelli che non richiedono il contatto. Gli strumenti a contatto utilizzano una lente che servirà all'applanazione corneale, perciò richiedono l'utilizzo di anestesia topica. Durante l'applanazione della cornea, il raggio di curvatura della stessa diminuisce permettendoci di catturare un'area speculare riflessa maggiore. Gli strumenti che non richiedono un contatto con la cornea utilizzano tecniche di focus automatiche. L'area speculare riflessa per questo tipo di

strumentazione è minore rispetto alla strumentazione a contatto a causa della superficie corneale riflettente che è curva. In entrambi i tipi di strumentazione però, l'area di cellule endoteliali visibili dipendono dalla superficie di riflessione epiteliale e dallo spessore della cornea. Ne consegue perciò che la larghezza delle immagini non cambia a seconda della tecnica utilizzata.

### 1.5.2 Microscopia confocale:

La microscopia confocale prese piede alla fine degli anni 80' con lo scopo di migliorare la qualità delle immagini dell'occhio a livello cellulare.
Uno dei limiti imposti dalla microscopia convenzionale è che la luce riflessa dalle strutture adiacenti al punto di osservazione fanno si che l'immagini presenti zone scure. Da ciò deriva un peggioramento del contrasto nell'immagine. Come risultato, gli ingrandimenti massimi ottenibili da questo tipo di strumentazione raggiungono i 40X, con una risoluzione di circa 20μm. Un ulteriore aumento dell'ingrandimento causerebbe una sfocatura eccessiva.
Il principio di microscopia confocale fu introdotto da Marvin Minsky nel 1957. Egli infatti propose un sistema per la quale il punto di fuoco è in comune sia per l'illuminazione sia per il punto di osservazione, da qui il nome di microscopia "confocale". Nella *figura 1.4* viene presentata una rappresentazione schematica del principio ottico di funzionamento della microscopia confocale.

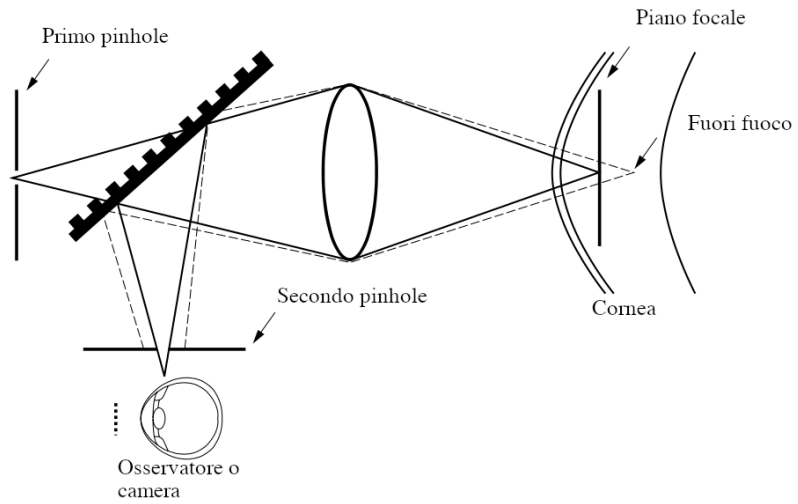



*Figura 1.4: Schema di funzionamento della microscopia confocale [11]*

Questa tecnica permette di aumentare la risoluzione su tutti gli assi, eliminando regioni fuori fuoco. La risoluzione laterale può raggiungere a 1-2 μm, mentre quella assiale 5-10μm. Questi fattori ci permettono di raggiungere ingrandimenti fino a 600X, in base all'apertura della lente utilizzata dall'obbiettivo.
La microscopia confocale fornisce un ridotto campo visivo, perciò è necessario spostare rapidamente il punto focale attraverso il campione e ricostruire l'immagine

affinché sia consentita una visualizzazione in tempo reale. Nel corso degli anni sono state presentate alcune tecniche che permettono di superare parzialmente questa criticità. La prima è denominata TSCM (tandem scanning confocal microscopy), utilizza un disco di Nipkov, cioè un disco metallico con una serie di fori microscopici posti in posizione tale da formare una spirale archimedea. La foratura consente l'illuminazione di molteplici punti singoli e la rotazione del disco permette la scansione dell'intero campione. La seconda, denominata SSCM (slit scanning confocal microscopy), utilizza fessure ottiche molto fini. Questa tecnica permette un'illuminazione maggiore rispetto alla prima, mentre entrambe garantiscono tempi di scansione molto rapidi.
Come abbiamo visto, nella microscopia confocale le sorgenti luminose possono essere puntiformi o a fessura. A causa del rischio di foto-tossicità causato da fasci laser coerenti, nella strumentazione clinica si preferisce usare una luce bianca incoerente assieme a luce ultravioletta e filtri infrarosso.
L'uso in vivo della microscopia confocale è limitato dal fatto che è una tecnica sensibile ai movimenti involontari del paziente come la respirazione. Da ciò deriva la necessità di avere sistemi di cattura e acquisizione dell'immagine molto veloci, con tempi di esposizione inferiori a 1/30 per ciascun frame, in modo da evitare sfocature nell'immagine catturata.
Un altro vantaggio fornito dalla microscopia confocale è la possibilità di controllare manualmente il piano focale dell'obbiettivo. Controllando la posizione dell'obbiettivo a mano a mano che si catturano immagini permette di valutare la profondità alla quale ciascuna immagine si riferisce.
Per eliminare fenomeni di riflessione si usano obbiettivi immersi in acqua con aperture numeriche elevate facendo attenzione al fatto che l'apertura elevata limita le distanze di lavoro e la profondità a cui possiamo penetrare.
Al giorno d'oggi la microscopia confocale ci consente di avere risoluzioni sufficienti per la visualizzazione delle cellule corneali, tuttavia non ci permette di vedere la struttura interna delle cellule [11].

# Capitolo 2

# Reti neurali

## 2.1 Cos'è il Machine Learning

L'apprendimento automatico, spesso chiamato Machine Learning, consiste nel programmare un calcolatore in modo tale che esso possa apprendere dai dati a sua disposizione. In poche parole, l'apprendimento è quel processo che converte l'esperienza in competenza. L'input di un algoritmo di apprendimento consiste nei dati di training, che rappresentano l'esperienza, mentre l'output è la competenza, rappresentata da un algoritmo in grado di eseguire un determinato compito.
Un tipico esempio di apprendimento automatico consiste nel mettere a punto un filtro che etichetti le e-mail di spam dalla casella di posta elettronica. Una soluzione semplice potrebbe essere quella di memorizzare tutte le precedenti e-mail ricevute che sono state etichettate come spam da un utente umano. Alla ricezione di una nuova e-mail, il calcolatore cercherà precedenti e-mail che sono considerate spam e se questa coinciderà con una di loro verrà eliminata, sennò verrà inserita nella cartella di posta ricevuta. Sebbene questo approccio possa sembrare efficace, manca di un aspetto fondamentale del sistema di apprendimento, cioè la capacità di etichettare e-mail mai viste, che possono essere differenti da ognuna di quelle già ricevute. L'algoritmo di apprendimento deve essere in grado di arrivare ad una generalizzazione più ampia a partire dai dati a sua disposizione. Nel caso del filtro antispam, ciò consiste nel fare in modo che l'algoritmo di apprendimento analizzi le e-mail contrassegnate come spam ed estragga un set di parole chiavi la cui presenza in un messaggio di posta sia indicativa della presenza di spam. Perciò, quando arriva una nuova e-mail, l'algoritmo può controllare tutte le e-mail già viste e valutare la presenza di parole sospette. Dopo questo procedimento l'e-mail verrà etichettata come genuina o spam. Un sistema di questo tipo è in grado di predire l'etichetta di e-mail mai viste.
Un altro aspetto fondamentale affinché un processo di apprendimento possa avere successo è la presenza di conoscenza a priori. Questa, infatti, influenzerà il processo di apprendimento ed è comunque inevitabile nel decretare il successo di questo processo. Intuitivamente, maggiore sarà la conoscenza a priori più facile sarà il processo di apprendimento. Tuttavia, più forte sarà la conoscenza a priori minore sarà la flessibilità dell'algoritmo, in quanto è vincolato, a priori, a soddisfare determinate assunzioni.
L'apprendimento è un campo di ricerca molto vasto ed è composto da diversi paradigmi.

I principali tre sono:

- **apprendimento supervisionato:** può essere visto come un processo in cui si usa l'esperienza per arrivare alla competenza. In questo caso l'algoritmo viene guidato da un "insegnante" che gli fornirà, per ogni input, gli output desiderati

- **apprendimento non supervisionato:** l'input fornito all'algoritmo non è già stato classificato, per cui lo scopo dell'algoritmo sarà quello di dedurre una funzione in grado di descrivere la struttura o i pattern dell'input

- **apprendimento rinforzato:** la sequenza di input sono un insieme di segnali di feedback provenienti dall'ambiente dinamico che l'algoritmo sta analizzando. L'obbiettivo è di eseguire una determinata azione come, ad esempio, guidare una macchina.

Il problema affrontato in questo elaborato richiede un algoritmo di apprendimento supervisionato, perciò in seguito verrà discusso soltanto questo paradigma di apprendimento.

## 2.2 Apprendimento supervisionato

Come discusso in precedenza, l'apprendimento supervisionato richiede l'utilizzo di un insieme di dati noti. Questo viene fornito in modo tale che contenga sia gli input sia gli output corretti. L'algoritmo dovrà poi essere in grado di costruire, da questo insieme di dati, un modello in grado di predire l'output corretto qualsiasi sia l'input. A questo punto, il modello di predizione dovrà essere validato con un altro insieme di dati noti, che dovrà essere indipendente da quello fornito nella fase di training. Solo in seguito ad una buona performance nella fase di validazione l'algoritmo potrà essere considerato affidabile.

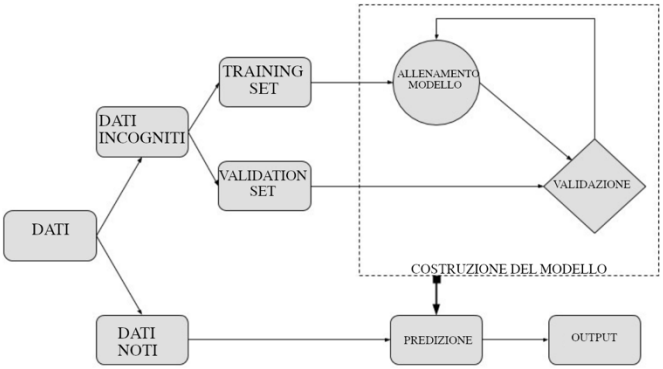



*Figura 2.1: Diagramma di flusso dell'apprendimento supervisionato [12]*

Dato perciò un problema di apprendimento supervisionato, i passi necessari ai fini dell'apprendimento sono:

**Scelta dell'algoritmo:** consiste nello scegliere l'algoritmo da utilizzare. Ci sono diversi tipi di algoritmi che si possono utilizzare, ma non esistono algoritmi che funzionano in modo ottimale per qualsiasi tipo di problema. Si dovrà quindi scegliere, volta per volta, e in base al compito da risolvere quale sarà il miglior algoritmo. Tra gli algoritmi più utilizzati ci sono: macchine a vettori di supporto (SVM), alberi decisionali, reti neurali artificiali.

**Allenamento:** in questa fase è necessario l'utilizzo di un insieme di dati che sia il più rappresentativo possibile del problema in esame. Se questa specifica non viene soddisfatta l'apprendimento sarà destinata a fallire. Il training set (insieme di dati di allenamento) dovrà essere composto da coppie di input e output desiderato.
L'algoritmo viene quindi allenato con il training set. L'obbiettivo di questa fase è costruire un modello il grado di descrivere i dati del training set nel miglior modo possibile.

**Validazione:** la fase di validazione ci permette di valutare le prestazioni del modello che abbiamo costruito nella fase di allenamento. In questa fase useremo un altro insieme di dati, chiamato insieme di validazione (insieme di dati per la validazione). Questo insieme di dati deve avere le stesse caratteristiche del training set ma allo stesso tempo deve essere indipendente da questo. Nella fase di validazione perciò si usa il modello precedentemente ottenuto per classificare nuovi input. Gli output ottenuti dal modello verranno poi confrontati con gli output desiderati contenuti nell'insieme di validazione al fine di valutare le prestazioni del modello. Se i risultati ottenuti sono soddisfacenti si può passare alla prossima e ultima fase, altrimenti bisognerà eseguire nuovamente le fasi di allenamento e validazione.
La validazione, e le tecniche di validazioni disponibili, permettono di avere una stima dell'errore di generalizzazione, cioè dell'errore medio che il modello commetterà quando debba elaborare dati che non hanno contribuito al suo apprendimento.

**Utilizzo del modello:** una volta che l'algoritmo è stato allenato e validato esso potrà essere utilizzato come sistema automatico per risolvere il problema originale su nuovi dati.

## 2.3 Reti neurali

Una rete neurale artificiale è un modello ispirato dalla struttura di reti neurali nel cervello. Essa è composta da un grande numero di elementi computazionali semplici (neuroni) che sono collegati tra di loro in modo da formare una rete di comunicazione complessa in modo tale da portare a termine calcoli molto complessi.
L'apprendimento attraverso l'utilizzo di reti neurali fu proposto durante la metà del ventesimo secolo. Solo recentemente però è stato dimostrato come questo tipo di modelli sia in grado di raggiungere prestazioni molto elevate in molti compiti di apprendimento.
Una rete neurale può essere descritta come un grafo i cui nodi corrispondono ai neuroni, mentre gli archi corrispondono ad un collegamento tra essi. Ciascun neurone riceve come input una somma pesata degli output dei neuroni connessi ad esso. Ci

concentreremo in seguito sulle reti feedforward visto che il loro grafo non contiene cicli.

### 2.3.1 Reti neurali feedforward

Una rete neurale feedforward è descritta da un grafo aciclico, G = (V, E), e una funzione peso per gli archi, $w: E \rightarrow \mathbb{R}$. I nodi del grafo corrispondono ai neuroni. Ogni neurone viene modellato come una semplice funzione scalare, $\sigma: \mathbb{R} \rightarrow \mathbb{R}$. La funzione $\sigma$ prende il nome di funzione di attivazione e saranno discusse in seguito. Ogni arco del grafo collega l'output di un neurone all'input di un altro neurone. L'input di un neurone consiste nella somma pesata degli output dei neuroni connessi ad esso, dove i pesi sono dati in accordo alla funzione $w$. Assumeremo ora che la rete sia organizzata in strati, cioè che l'insieme di nodi possa essere decomposto in una unione di sottoinsiemi disgiunti non vuoti, $V = \cup_{t=0}^{T} V_t$, in modo tale che ogni arco in E connetta qualche noto in $V_{t-1}$ a qualche nodo in $V_t$ per qualche $t \in [T]$. Lo strato di fondo, $V_0$, è chiamato strato di input. Questo contiene $n+1$ neuroni, dove $n$ è la dimensione dello spazio degli input. Per ogni $i \in [n]$, l'output del neurone $i$ in $V_0$ è semplicemente $x_i$. L'ultimo neurone in $V_0$ è un neurone il cui output è costantemente pari a 1. Chiameremo ora $v_{t,i}$, l'$i$-esimo neurone del $t$-esimo strato mentre $o_{t,i}(x)$, l'output di $v_{t,i}$ quando viene dato in ingresso alla rete il vettore $x$. Avremo perciò, per $i \in [n]$, $o_{0,i}(x) = x_i$ e per $i = n+1$ abbiamo $o_{0,i}(x) = 1$. Si può procedere ora con i calcoli strato per strato. Supponendo di aver calcolato gli output dei neuroni nello strato $t$-esimo si possono calcolare gli output dello strato $t+1$ come segue. Si fissa $v_{t+1,j} \in V_{t+1}$. Sia $a_{t+1,j}(x)$ l'input di $v_{t+1,j}$ quando alla rete viene fornito il vettore di input $x$. Allora:

$$a_{t+1,j}(x) = \sum_{r:(v_{t,r}, v_{t+1,j}) \in E} w\left((v_{t,r}, v_{t+1,j})\right) o_{t,r}(x)$$

e

$$o_{t+1,j}(x) = \sigma\left(a_{t+1,j}(x)\right)$$

$\sigma$ rappresenta la funzione di attivazione che verrà analizzata nel prossimo paragrafo.

Da ciò segue che l'input del neurone $v_{t+1,j}$ sarà una somma pesata degli output dei neuroni presenti nello strato $V_t$ che sono connessi a $v_{t+1,j}$. I pesi saranno assegnati in accordo alla funzione $w$ e l'output di $v_{t+1,j}$ è semplicemente dato dall'applicazione della funzione di attivazione $\sigma$ ai suoi input.
Gli strati $V_1, \ldots, V_{T-1}$ sono spesso chiamati strati nascosti. Lo strato superiore $V_T$ viene chiamato strato di output. In semplici problemi di predizione lo strato di output conterrà un singolo neurone per l'output che coinciderà con l'output della rete. Useremo il termine $T$ per indicare il numero di strati della rete (escluso $V_0$) o, equivalentemente, la "profondità" della rete. La dimensione della rete sarà invece $|V|$. La larghezza sarà $max_t|V_t|$.

In *figura 2.2* viene presentata un'illustrazione di una rete neurale feedforward di profondità 2, dimensione 20 e larghezza 5. Si noti la presenza di un neurone nello strato nascosto che non ha archi in ingresso. Questo neurone produrrà come output una costante [13].

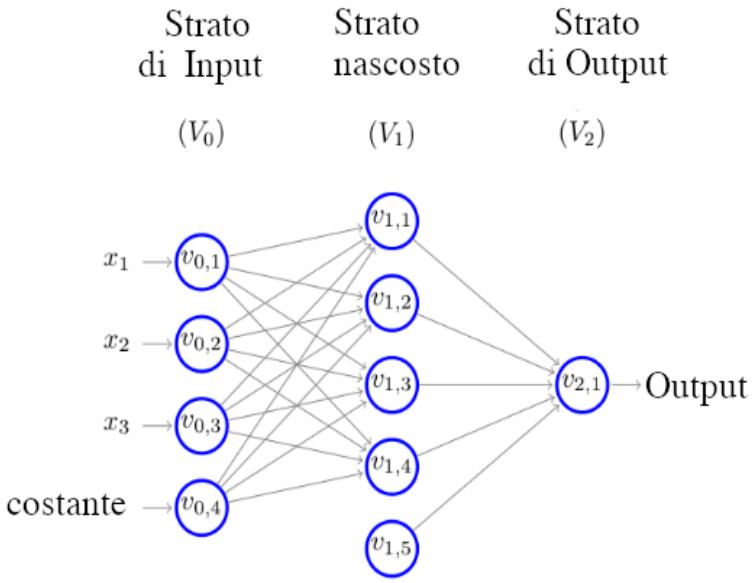



*Figura 2.2: Esempio di rete neurale feedforward [13].*

### 2.3.2 Funzioni di attivazione

La funzione di attivazione è di fondamentale importanza in una rete neurale poiché lega in maniera non lineare i valori di ingresso ad un neurone e la sua uscita. Ci sono molte funzioni che possono essere usate come funzioni di attivazione, in questo elaborato però verranno mostrate soltanto quelle più utilizzate.

#### Funzione sigmoidale

Questa è la funzione più usata nell'ambito delle reti neurali ed è una funzione strettamente crescente che mostra un bilanciamento tra comportamento lineare e non lineare. Questa funzione ha anche la caratteristica di approssimare la funzione threshold se c → $+\infty$.

$$\sigma(z) = \frac{1}{1 + \exp(-c * z)}$$

$$\frac{d\,\sigma(z)}{dz} = \frac{e^z}{(e^z + 1)^2}$$

In *figura 2.3* e *figura 2.4* viene presentato l'andamento di $\sigma(z)$ e $\frac{d\,\sigma(z)}{dz}$.

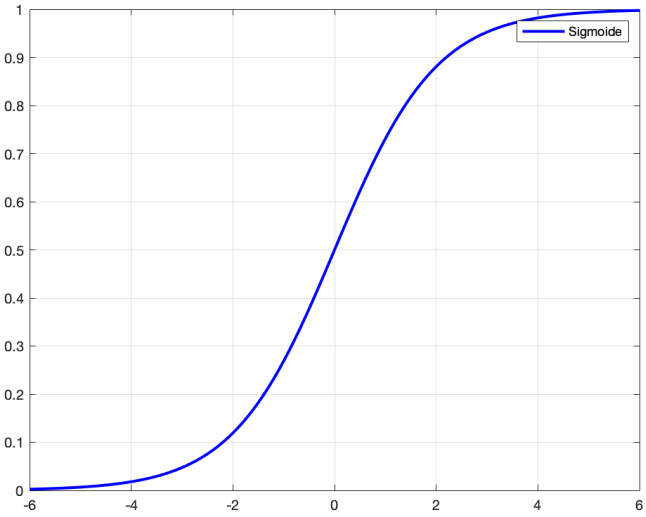



*Figura 2.3: Andamento della funzione sigmoidale*

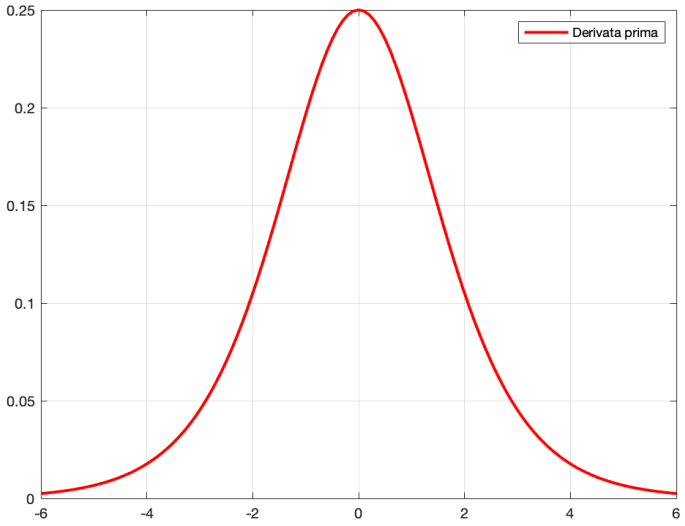



*Figura 2.4: Andamento della derivata della funzione sigmoidale*

## Rectified Linear Unit

Il rettificatore è una funzione di attivazione definita come la parte positiva del suo argomento.

$$f(x) = \max(0, x)$$

$$\frac{df(x)}{dx} = u(t)$$

dove u(t) denota il gradino unitario.

In *figura 2.5* e *figura* 2.6 viene presentato, rispettivamente, l'andamento della funzione di attivazione e della sua derivata.

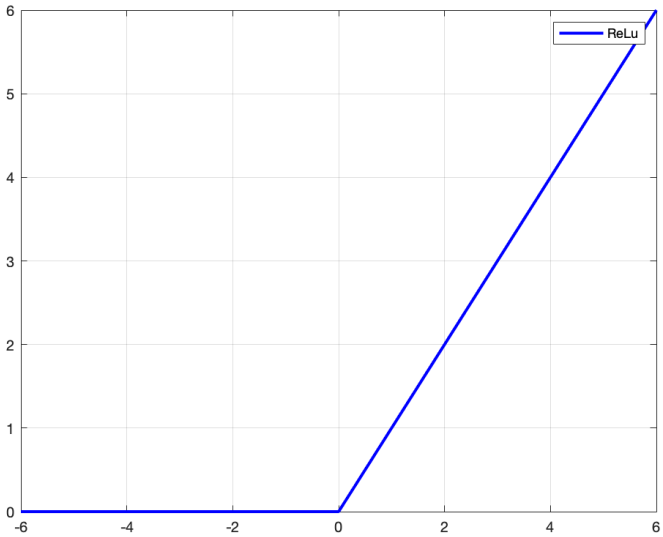



*Figura 2.5: Andamento della funzione di attivazione ReLu*

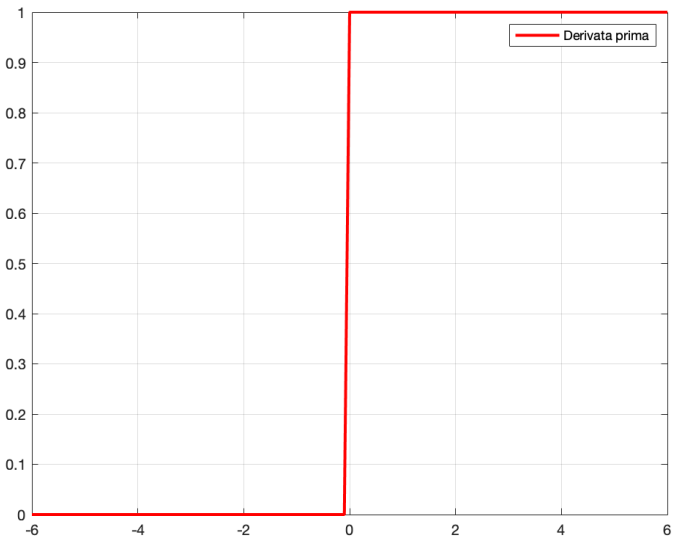



*Figura 2.6: Andamento della derivata della funzione ReLu*

## 2.4 Apprendimento mediante l'uso di reti neurali

Finora abbiamo solamente analizzato la struttura delle reti neurali. Una volta che abbiamo fissato il grafo *(V,E)* e scelto la funzione di attivazione $\sigma$, com'è possibile far apprendere qualcosa alla rete neurale? Ricordando quanto detto a riguardo dell'apprendimento supervisionato, avremo bisogno di un processo di calcolo che permetta di individuare i parametri opportuni della rete a partire dal training set che verrà sottoposto più volte alla rete. La misura della qualità della rete ottenuta si misura

tramite una funzione di errore che definiremo in seguito. Nel corso della fase di allenamento l'obbiettivo sarà quindi la minimizzazione della funzione di errore.

### 2.4.1 Funzione di errore

Al fine di valutare le prestazioni di una rete neurale è necessario definire la funzione di errore. Questa può essere definita come segue:

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m} E^{(i)}$$

dove *m* denota il numero di campioni presenti nel training set mentre $E^{(i)}$ è l'errore tra l'output calcolato della rete e l'output ottimo fornito dal training set per il medesimo campione *i*-esimo.

$$E^{(i)} = \frac{1}{2}|y^{(i)} - h_\theta(x^{(i)})|^2$$

Si può notare che $E^{(i)}$ varia anche in funzione dei parametri utilizzati $\theta$.

### 2.4.2 Algoritmi per la minimizzazione della funzione di errore

Al fine di minimizzare la funzione di errore si possono usare diversi algoritmi di natura iterativa. In questo tipo di algoritmi è sempre possibile distinguere due fasi distinte:

- Fase 1: calcolo delle derivate della funzione di errore rispetto ai parametri $\theta$. Uno dei metodi più efficaci per il calcolo di tali derivate è chiamato backpropagation.
- Fase2: si calcolano i nuovi parametri della rete usando le derivate appena calcolate.

Si noti che le due fasi sono distinte. Infatti, ad ogni iterazione, prima si calcola la derivata della funzione di errore ottenuto sul training set, poi si aggiornano i parametri della rete con una formula di aggiornamento. La formula di aggiornamento più utilizzata si chiama discesa del gradiente. Essa realizza una variazione al generico parametro $\vartheta_{j,i}$ secondo la relazione:

$$\vartheta_{j,i} = \vartheta_{j,i} - \eta\frac{\partial E}{\partial \vartheta_{j,i}}$$

con $\eta$ parametro costante compreso nell'intervallo (0,1) chiamato *tasso di apprendimento*.

## Backpropagation

L'algoritmo di backpropagation ci consente di calcolare la derivata della funzione di errore utilizzando i valori di output dei nodi della rete. È fondamentale tenere presente che un nodo di una rete feedforward calcola una somma pesata dei suoi input:

$$a_j = \sum_i w_{i,j} * o_i$$

Dove $o_i$ rappresenta l'output di un nodo connesso al nodo j.
Il valore di output del nodo j è invece dato da:

$$o_j = \sigma(a_j)$$

con $\sigma$ funzione di attivazione.
Per il calcolo di $\partial E / \partial \vartheta_{j,i}$ dovremo sfruttare la seguente relazione:

$$E = \sum_{i=1}^{n} E^{(i)}$$

la quale specifica che l'errore totale commesso dalla rete su un training set di dimensione *n* è la somma degli errori che la rete commette su ogni singolo elemento del training set. Questa relazione ci consentirà di focalizzarci sul calcolo di $\partial E^{(i)} / \partial \vartheta_{j,i}$.
La prima ipotesi necessaria che deve essere fatta per il calcolo delle derivate parziali della funzione di errore è che $E^{(i)}$ possa essere espresso come funzione delle variabili di output.
Estraggo una coppia $(x_i, y_i)$ dal training set e suppongo di darla in input alla rete che sto costruendo, calcolando così le attivazioni di tutti i nodi interni e di output attraverso la forward propagation. Si osserva che $E^{(i)}$ dipende da $\vartheta_{j,i}$ attraverso $a_j$.
Quindi si può dimostrare che:

$$\frac{\partial E^{(i)}}{\partial \vartheta_{j,i}} = \delta_j * o_i \ , \qquad \text{con } \delta_j = \frac{\partial E^{(i)}}{\partial a_j}$$

Questa relazione ci consente di dire che la derivata si ottiene moltiplicando il valore di output del nodo iniziale della connessione avente parametro $\vartheta_{j,i}$ per un certo valore $\delta_j$ associato al nodo alla fine della connessione avente parametro $\vartheta_{j,i}$. Si può quindi affermare che la derivata della funzione di errore rispetto al parametro della connessione che va dal nodo i al nodo j è calcolata attraverso l'uso di una formula locale.
Per il calcolo di $\delta_j$ si useranno le seguenti relazioni:

- Per le unità di output:

$$\delta_k = \frac{\partial E^{(i)}}{\partial a_k} = \sigma'(a_k) * \frac{\partial E^{(i)}}{\partial o_k}$$

- Per le restanti unità:

$$\delta_j = \frac{\partial E^{(i)}}{\partial a_j} = \sigma'(a_j) * \sum_k \vartheta_{k,j} * \delta_k$$

Nella prima equazione, il termine $\sigma'(a_k)$ è una misura di quanto velocemente la funzione di attivazione sta cambiando, mentre $\partial E^{(i)}/\partial y_k$ misura quanto velocemente la funzione di errore cambia in funzione dell'attivazione dell'output *k*-esimo.
La seconda equazione invece ci permette di utilizzare l'errore $\delta_k$ dello strato di input per calcolare l'errore di output dello strato immediatamente precedente. Si ha quindi una formula di tipo ricorsivo che ci permette di risalire dai nodi più esterni, in direzione inversa rispetto alla forward propagation, al valore $\delta_j$ di tutti i nodi interni; da qui il nome di backpropagation.
Ora possiamo riassumere la backpropagation per il calcolo delle derivate della funzione di errore $E^{(i)}$ in quattro fasi:

1. Applico un input alla rete provocando una forward propagation
2. Calcolo i valori di $\delta_k$ per i nodi di output, dove $a_k$ è l'input del *k*-esimo nodo di output
3. Calcolo i valori di $\delta_j$ per i restanti nodi interni della rete propagando all'indietro il valore dei $\delta$
4. Calcolo le derivate richieste.

**Backpropagation: dimostrazione**

Dimostreremo ora le precedenti equazioni relative alla backpropagation.
La funzione di errore $E^{(i)}$ dipende da un peso $\vartheta_{j,i}$ solo attraverso $a_j$ che è l'input al nodo *j*.
Possiamo quindi scrivere:

$$\frac{\partial E^{(i)}}{\partial \vartheta_{j,i}} = \frac{\partial E^{(i)}}{\partial a_j} * \frac{\partial a_j}{\partial \vartheta_{j,i}}$$

Dato che:

$$a_j = \sum_h \vartheta_{j,i} * o_h$$

Risulta che:

$$\frac{\delta a_j}{\delta \vartheta_{j,i}} = o_i$$

Quindi:

$$\frac{\partial E^{(i)}}{\partial \vartheta_{j,i}} = \frac{\partial E^{(i)}}{\partial a_j} * o_i$$

Ponendo allora:

$$\delta_j = \frac{\partial E^{(i)}}{\partial a_j}$$

Si ottiene:

$$\frac{\partial E^{(i)}}{\partial \vartheta_{j,i}} = \delta_j * o_i$$

Ora rimangono da dimostrare solo le formule per ottenere i $\delta_j$. Per un generico nodo della rete, $E^{(i)}$ dipende da $a_j$ tramite gli $a_k$ dei nodi che ricevono connessioni dal nodo *j*-esimo, cioè:

$$\frac{\partial E^{(i)}}{\partial a_j} = \sum_k \frac{\partial E^{(i)}}{\partial a_k} * \frac{\partial a_k}{\partial a_j}$$

Dato che:

$$a_k = \sum_h \vartheta_{k,h} * o_h = \sum_h \vartheta_{k,h} * \sigma_h(a_h)$$

Si ottiene:

$$\frac{\partial a_k}{\partial a_j} = \vartheta_{k,j} * \sigma'_j(a_j)$$

Si possono ora dimostrare le equazioni precedenti, quindi:

$$\frac{\partial E^{(i)}}{\partial a_j} = \sum_k \frac{\partial E^{(i)}}{\partial a_k} * \vartheta_{k,j} * \sigma'_j(a_j)$$

$$\frac{\partial E^{(i)}}{\partial a_j} = \sigma'_j(a_j) * \sum_k \vartheta_{k,j} * \frac{\partial E^{(i)}}{\partial a_k}$$

E dato che per definizione $\delta_j = \partial E^{(i)} / \partial a_j$ si ottiene:

$$\delta_j = \sigma'_j(a_j) * \sum_k \vartheta_{k,j} * \delta_k$$

Con *k* indice dei nodi che hanno un arco proveniente dal nodo *j*. Si ricorda inoltre che questa formula è valida soltanto per i nodi interni di una rete neurale feedforward. Consideriamo ora un generico nodo di output *k* visto che $E^{(i)} = E(o_1, o_2, \ldots, o_n)$ e $o_k = \sigma_k(a_k)$ si ha:

$$\delta_k = \frac{\partial E^{(i)}}{\partial a_k} = \frac{\partial E^{(i)}}{\partial o_k} * \sigma'_k(a_k)$$

valida soltanto per i nodi di output di una rete neurale feedforward [14].

## Backpropagation: pseudocodice

Utilizzando quanto ottenuto in precedenza si presenta ora il procedimento passo per passo che ci permetterà di implementare l'algoritmo di backpropagation.
Data una rete neurale e un training set e scelti una funzione di attivazione $\sigma$, il tasso di apprendimento $\eta$, la funzione di errore E, e una condizione di arresto l'implementazione dell'algoritmo è la seguente:

**Algoritmo di backpropagation**

1. Inizializzo tutti i parametri della rete con valori casuali di piccola dimensione
2. **while** la condizione di arresto non è verificata **do**
3. **for** ogni coppia (x, y) presente nel training set
4. propago l'input attraverso la rete
5. propago l'errore all'indietro attraverso la rete
6. calcolo l'errore commesso in output dalla rete
7. **for** ogni nodo interno della rete
8. calcolo il suo errore
9. **end for**
10. **for** ogni nodo interno della rete
11. calcolo la derivata dell'errore rispetto ai parametri
12. aggiorno i parametri della rete con la regola di discesa del gradiente
13. **end for**
14. **end for**
15. **end while**

## 2.5 Reti neurali convoluzionali

Le reti neurali convoluzionali (CNN) sono uno dei diversi tipi di reti neurali artificiali attualmente presenti in ambito scientifico. Infatti, come ogni rete neurale artificiale, esse sono composte da neuroni che sono collegati tra di loro attraverso degli archi; di conseguenza tutto quanto è stato precedentemente esposto sull'allenamento di una rete neurale rimane valido anche in questo contesto.
Le reti neurali convoluzionali, a differenza di altre, assumono che l'input abbia una precisa struttura di dati e ciò che ne consegue è la possibilità di ridurre la quantità di parametri utilizzata dalla rete visto che potremo fare a meno di utilizzare architetture completamente connesse. Verranno usate architetture completamente connesse solo localmente. Questo vuol dire che i neuroni di uno strato sono connessi solo a pochi neuroni dello strato precedente, invece che a tutti i neuroni come in un'architettura completamente connessa. Questa è una delle principali caratteristiche che differenziano una CNN da una normale rete completamente connessa.
Le reti neurali convoluzionali hanno particolare utilità nel rilevamento di pattern nelle immagini apprendendo direttamente da esse. Queste consentono infatti di apprendere direttamente dalle immagini utilizzando i pattern rilevati ai fini della classificazione, evitando di dover eseguire l'estrazione manuale delle caratteristiche.

### 2.5.1 Principali tipi di strati di una rete neurale convoluzionale

Nelle reti neurali convoluzionali esistono diversi tipi di strati, ciascuno adibito ad una specifica funzioni. Tra questi, certi tipi di strati contengono parametri che possono essere allenati mentre altri implementano una funzione fissata.
Presenteremo da qui in seguito le tipologie di strati più utilizzati nelle reti neurali convoluzionali.

#### Strato convoluzionale:

È lo strato principale per questa tipologia di reti neurali da cui, infatti, ne prendono il nome. Lo scopo di questo strato è realizzare un banco di filtri numerici che verranno poi usati per elaborare le immagini fornite in input alla rete. Ciascun filtro ha dimensioni spaziali ridotte e verrà infatti applicato a piccole porzioni di input. Perciò, durante la fase di forward propagation, si convolve ciascun filtro lungo tutta l'area dell'input. Verrà effettuata, quindi, una moltiplicazione scalare tra i valori del filtro e la porzione di input alla quale esso viene applicato e ciò verrà ripetuto per tutta l'immagine fornita come input. L'obbiettivo della rete sarà quello di mettere a punto dei filtri che riescano a riconoscere una specifica caratteristica nella regione dell'immagine considerata.
Le reti neurali convoluzionali solitamente possiedono molteplici strati convoluzionali. Ci consentono infatti di ottenere caratteristiche di basso livello come linee, angoli e contorni se siamo vicini alla zona di input, mentre più ci avviciniamo all' output più ci consentono di ottenere caratteristiche di alto livello come oggetti complessi.

## Strato di pooling

Lo strato di pooling è un'altra componente indispensabile nelle reti neurali convoluzionali. La loro presenza in una rete è periodica e il loro scopo è di ridurre la dimensione dei volumi in uno specifico stadio della rete. Ciò implica che possiamo ridurre i parametri della rete e quindi la complessità computazionale della rete. Altra caratteristica importante dello strato di pooling è che permette di controllare l'overfit. A differenza dello strato convoluzionale lo strato di pooling implementa una funzione ben definita e non ha quindi bisogno di essere allenato. Il loro contributo perciò verrà apprezzato soltanto durante la fase di forward propagation, mentre nella fase di backward propagation non farà altro che propagare l'errore. Le funzioni più usate nell'operazione di pooling sono:

- Average Pooling: si calcola il valore medio della porzione di input considerata
- Max Pooling: calcolo il valore massimo della porzione di input considerata

Dopo l'operazione di pooling, otterremo quindi un riassunto delle caratteristiche individuate nell'immagine di input. Per capire meglio questa operazione in *figura 2.7* viene riportato un esempio di max pooling [10].

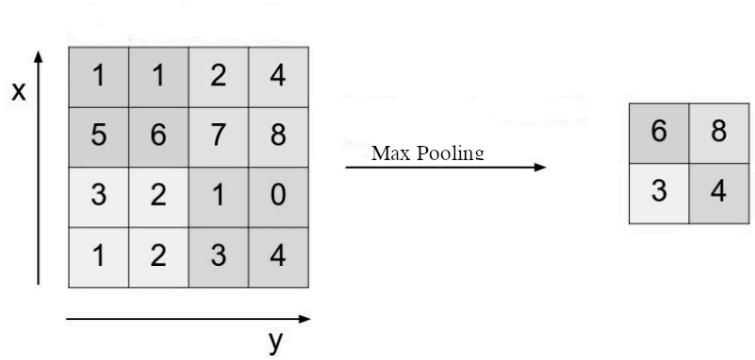



*Figura 2.7: esempio di Max Pooling [12].*

# Capitolo 3

# Stato dell'arte della conta cellulare

## 3.1 Introduzione

Il conteggio cellulare è un importante problema e consiste nel contare o quantificare il numero di cellule per fini diagnostici e terapeutici. Ad esempio, il conteggio del numero di cellule ematiche può aiutare un medico a determinare le cause del malessere di un paziente. Il problema del conteggio cellulare è un problema affrontato in molti ambiti di ricerca, come in medicina o in ambito biologico. Nei recenti anni sono stati sviluppati molti algoritmi di conta cellulare, che si basano su approcci diversi tra di loro. In questo capitolo verranno presentati alcuni dei più recenti algoritmi di conta cellulare utilizzati in ambito oftalmico, ponendo particolare attenzione al metodo con cui viene risolto il problema di conta e i risultati che si ottengono.

## 3.2 Metodo 1

**Titolo:** Corneal endothelial cell segmentation by classifier-driven margin of oversegmented images.

**Autori:** J.P. Vigueras-Guillén, E.R. Andrinopoulou, A. Engel, H. G. Lemij, J. van Rooij, K. A. Varmeer, L. J. van Vliet.

**Pubblicato in:** IEEE Transactions on Medical Imaging (Volume: 37, Numero: 10, Ottobre 2018).

**Tipo di dati:** Immagini dell'endotelio corneale ottenute con il microscopio speculare Topcon SP-1P

**Metodo:**

1) Pre-elaborazione: le immagini da fornire al classificatore vengono prima segmentate molto finemente.
2) Classificatore: un classificatore SVM riceve le immagini pre-elaborate in ingresso e restituisce le immagini correttamente segmentate.
3) Il calcolo della densità cellulare avviene sulle immagini segmentate.

**Dati:** 30 immagini di endotelio corneale di occhi affetti da glaucoma. Circa 250 cellule per immagine.

**Risultati ottenuti:** Densità cellulare dell'endotelio corneale stimata in immagini microscopiche speculari: $22 \pm 30\ cellule/mm^2$, coefficiente di variazione $\mathrm{CV}(\%) = 1.7 \pm 1.8$.

## 3.3 Metodo 2

**Titolo:** Fully automatic evaluation of the corneal endothelium from in vivo confocal microscopy

**Autori:** B. Selig, K. A Vermeer, B. Rieger, T. Hillenaar, C. L Luengo Hendriks.

**Pubblicato in:** BMC Medical Imaging 15, Articolo numero 13, 2015

**Tipo di dati:** Immagini dell'endotelio corneale ottenute con il microscopio confocale Nidek Technologies Confoscan 4

**Metodo:** La immagini vengono analizzate nel dominio della frequenza, la quale permette di inferire la dimensione media di una cellula. L'idea consiste nel proiettare lo spettro di frequenza bidimensionale in una funzione mono dimensionali. Da questa si estrare la frequenza alla quale si verifica il picco maggiore. Questa frequenza, chiamata frequenza caratteristica, è correlata alla dimensione media della cellula e quindi alla densità.

**Dati:** 52 immagini di endotelio corneale provenienti da 23 pazienti con distrofia endoteliale di Fuchs.

**Risultati ottenuti:** Gli errori relativi stimati sono riportati in *figura 3.1*. Per ogni metodo di analisi in frequenza si riporta il boxplot dell'errore relativo commesso dall'algoritmo rispetto ai valori stimati a mano. Il rettangolo indica l'intervallo interquartile, la line al suo interno indica la mediana, il baffo indica gli estremi e i punti indicano i valori anomali.

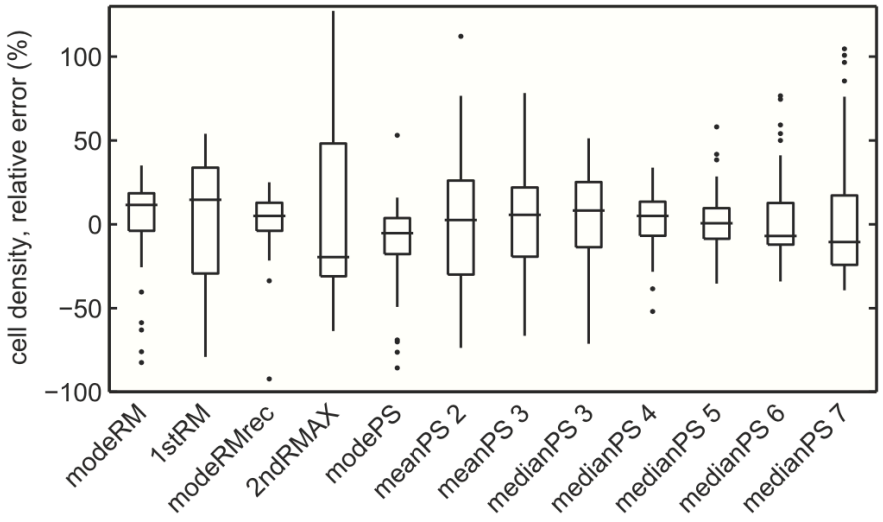


*Figura 3.1: Risultati ottenuti da B. Selig, K. A Vermeer, B. Rieger, T. Hillenaar, C. L Luengo Hendriks*

## 3.4 Metodo 3

**Titolo:** Automated segmentation of the corneal endothelium in a large set of 'real-world' specular microscopy images using the U-Net architecture

**Autori:** M. C. Daniel, L. Atzrodt, F. Bucher, K. Wacker, S. Bohringer, T. Reinhard, D. Bohringer.

**Pubblicato in:** Nature, Scientific Reports 9, Articolo numero 4752, 2019

**Tipo di dati:** Immagini dell'endotelio corneale ottenute con il microscopio speculare Topcon SP-3000

**Metodo:**

1) Pre-elaborazione: Le immagini sono state filtrate con un filtro Gaussiano (raggio di 1px, SD = 2px). In seguito, viene applicato l'operatore morfologico di 'Top-Hat'. Infine, si applica una equalizzazione dell'istogramma.
2) Le immagini vengono date in ingresso ad una rete neurale con architettura U-Net che ne segmenta le cellule. La densità viene calcolata come inverso del valore mediano dell'area occupata da una cellula.

**Dati:** Sono state considerate 385 immagini, dalle quali sono state estratte patch di dimensioni 78 x 78 sulle quali è stato eseguito data augmentation. Si sono così ottenute 11,376 patch per il training della rete.

**Risultati ottenuti:** La precisione media nel calcolo della densità in fase di validazione, calcolata come numero di cellule individuate dall'algoritmo diviso il numero di cellule contate dall'operatore umano è pari a 0.84.

## 3.5 Metodo 4

**Titolo:** Fully convolutional architecture vs sliding-windows CNN for corneal endothelium cell segmentation

**Autori:** J.P. Vigueras-Guillèn, B. Sari, S. F. Goes, H. G: Lemij, J. Van Rooij, K. A. Vermeer, L. J. van Vliet

**Pubblicato in:** BMC Biomedical Engineering 1, articolo numero 4, 2019

**Tipo di dati:** immagini ottenute con microscopio speculare Topcon SP-1P

**Metodo:**

1) Pre-elaborazione dei dati: viene applicata un'equalizzazione adattativa dell'istogramma alle immagini di training
2) Data augmentation: le immagini pre-elaborate vengono ruotate orizzontalmente e verticalmente in modo da quadruplicare i dati a disposizione.
3) Le immagini vengono segmentate sia con la rete neurale SW-net sia con la rete neurale U-net per poterne confrontare le prestazioni
4) Ottenute le immagini segmentate, la densità cellulare endoteliale viene calcolata come l'area totale occupata da tutte le cellule in un'immagine diviso il numero di cellule presenti.

**Dati:** 50 immagini ottenute con microscopio speculare. Dopo data augmentation il training set è composto da 200 immagini.

**Risultati ottenuti:** I risultati vengono riportati in termini di errore di stima, calcolato come la differenza del valore di densità stimato dall'algoritmo e il valore stimato manualmente.

| **Errore di stima** | Densità ($cellule/mm^2$) | CV (%) |
|---|---|---|
| SW- net | $9.9 \pm 23.1$ | $0.5 \pm 1.6$ |
| U - net | $3.2 \pm 10.2$ | $0.4 \pm 0.7$ |

## 3.6 Tabella riassuntiva

| | Tipo di immagini | Quantità di immagini utilizzate | Performance |
|---|---|---|---|
| **Metodo 1** | Immagini speculari ottenute con Topcon SP-1P | 30 | Stima della densità: $22 \pm 30 \frac{cell}{mm^2}$, $CV(\%) = 1{,}7 \pm 1.8$ |
| **Metodo 2** | Immagini confocali ottenute con Nidek Technologies Confoscan4 | 52 | Errori relativi per i modelli utilizzati sotto forma di boxplot (vedi *figura 3.1)* |
| **Metodo 3** | Immagini speculari ottenute con Topcon SP-3000 | 385 | Rapporto tra il numero di cellule rilevate e il numero di cellule contate manualmente. Risultato: l'algoritmo rileva l'84% delle cellule presenti nel dataset |
| **Metodo 4** | Immagini speculari ottenute con Topcon SP-1P | 50 | Stima della densità per i due modelli utilizzati: SW-Net: $9.9 \pm 23.1 \frac{cell}{mm^2}$, $CV(\%) = 0.5 \pm 1.6$ U-Net: $3.2 \pm 10.2 \frac{cell}{mm^2}$, $CV(\%) = 0.4 \pm 0.7$ |

# Capitolo 4

# Metodologia

## 4.1 Introduzione

Il codice realizzato in questa tesi è stato implementato in PyTorch [15], una libreria open source utilizzata specialmente per applicazioni di computer vision. PyTorch è implementato sia in python sia in c++ [16] e uno dei grandi vantaggi che offre è la compatibilità con la piattaforma Nvidia CUDA [17], la quale permette di ridurre di molto il tempo necessario ad effettuare la fase di training per le reti neurali. In questo capitolo verranno specificati l'hardware e il software utilizzati in questo elaborato, inoltre si parlerà del dataset e di come è stato elaborato. Si discuterà inoltre delle due architetture di reti neurali scelte per effettuare la conta cellulare e di come sono state implementate.

## 4.2 Hardware utilizzato

La elevata complessità computazionale richiesta durante la fase di training di una rete neurale fa si che i computer comunemente disponibili in commercio siano spesso non adatti a svolgere questo compito. Con il diffondersi dell'utilizzo di algoritmi di machine learning, sono nati servizi come Azure [18] ed AWS [19] che permettono di noleggiare potenti unità di calcolo remote le quali consentono di eseguire la fase di training di reti neurali molto complesse e con una grande quantità di dati.
Per eseguire la fase di training dei modelli presentati in questa tesi ho utilizzato il cluster di calcolo 'Blade' [20] messo a disposizione dal dipartimento di Ingegneria dell'informazione dell'Università degli studi di Padova. In *Appendice 1* viene proposta una guida al suo funzionamento.
Il server è composto da 14 unità di calcolo, divise in 13 unità CPU e una GPU. Le unità di calcolo più utilizzate per lo sviluppo di questa tesi sono:

1) runner-13 e runner 14 equipaggiati con:
    - 4 Processori 16-Core AMD Opteron 6378 @2.4GHz
    - 256 GB di memoria RAM

2) gpu1 equipaggiato con
    - 4 Processori 12-Core Intel Xeon Gold 5118 @2.3Ghz
    - 1 TB di memoria RAM
    - 7 GPU NVIDIA GTX 1080 Ti

Durante tutto lo sviluppo dei modelli si è preferito l'uso del server gpu1 in quanto l'unico dotato di schede grafiche Nvidia e quindi l'unico ad essere equipaggiato con le librerie Nvidia CUDA. Quando il server gpu1 era occupato ho utilizzato i server runner equipaggiati di sole CPU.

## 4.3 Software utilizzato

Per lo sviluppo e il training dei modelli presentati in questa tesi è stato utilizzato Anaconda [21], una distribuzione open source di Python. La versione di Python utilizzata è Python 3.7, mentre per PyTorch è stata utilizzata la versione PyTorch 1.3. Per scrivere il codice è stato utilizzato Spyder [22] che è un ottimo ambiente di sviluppo integrato per Python.


*Figura 4.1 Software utilizzato per lo sviluppo del codice*

## 4.4 Il dataset

Il dataset su cui questo elaborato vuole effettuare la conta cellulare è costituito da immagini di endotelio corneale ottenute attraverso tecniche di microscopia speculare e confocale. Le immagini sono state messe a disposizione da Nidek Technologies, uno dei più importanti costruttori di apparecchiature per oftalmologia diagnostica.
Il dataset è composto da 64 immagini di dimensione *768 x 576 pixel* in uno spazio colore *RGB*. Uno dei primi problemi che bisognerà affrontare è che la quantità di dati a disposizione potrebbe essere troppo contenuta affinché una rete neurale possa apprendere come effettuare la conta cellulare. Un altro problema proviene dal fatto che la qualità di alcune delle 64 immagini è talmente bassa che nemmeno un operatore umano esperto riuscirebbe ad effettuare la conta cellulare, quindi non possono essere utilizzate. Dopo aver eliminato dal dataset tutte le immagini che non ci permettevano di effettuare alcun tipo di elaborazione, le totalità delle immagini restanti hanno presentato comunque delle zone in cui effettuare la conta cellulare è rimasto impossibile, come mostrato in *figura 4.2, figura 4.3 e figura 4.4*.

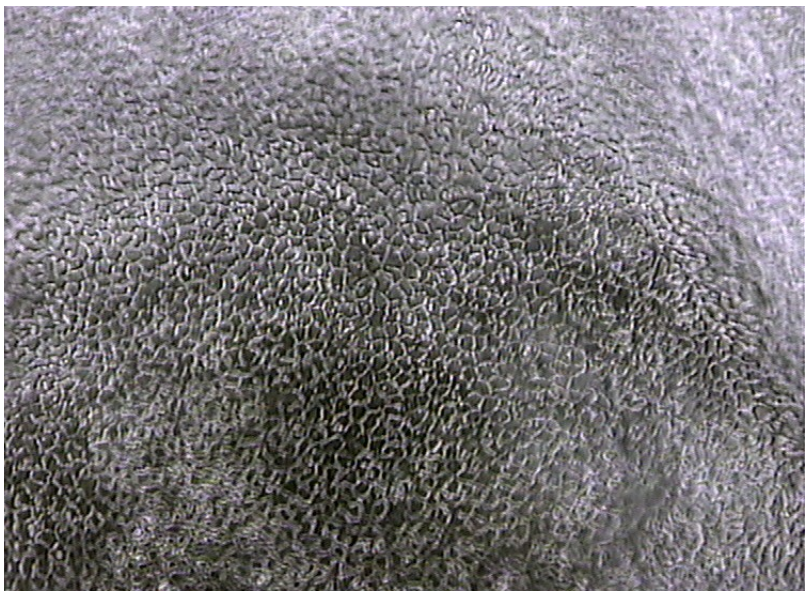


*Figura 4.2: Esempio 1 di immagine proveniente dal dataset.*

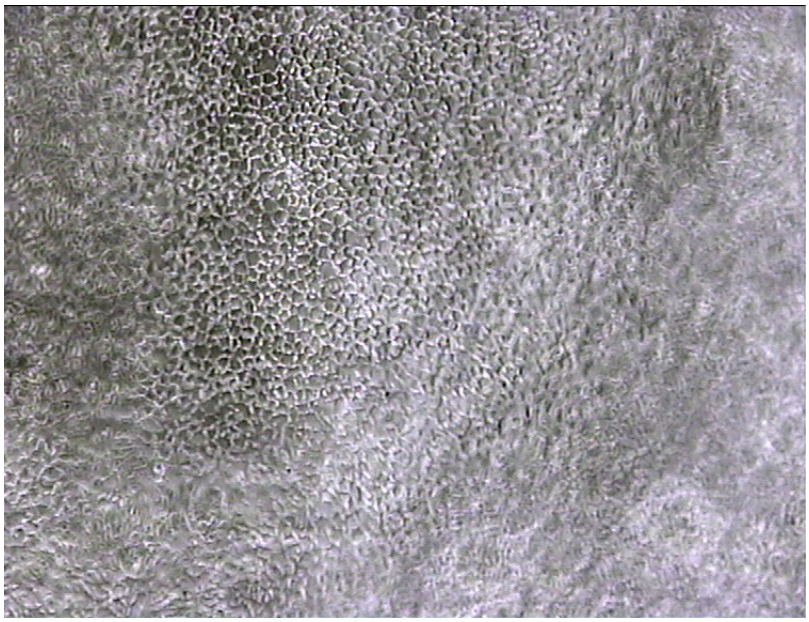


*Figura 4.3: Esempio 2 di immagine proveniente dal dataset*

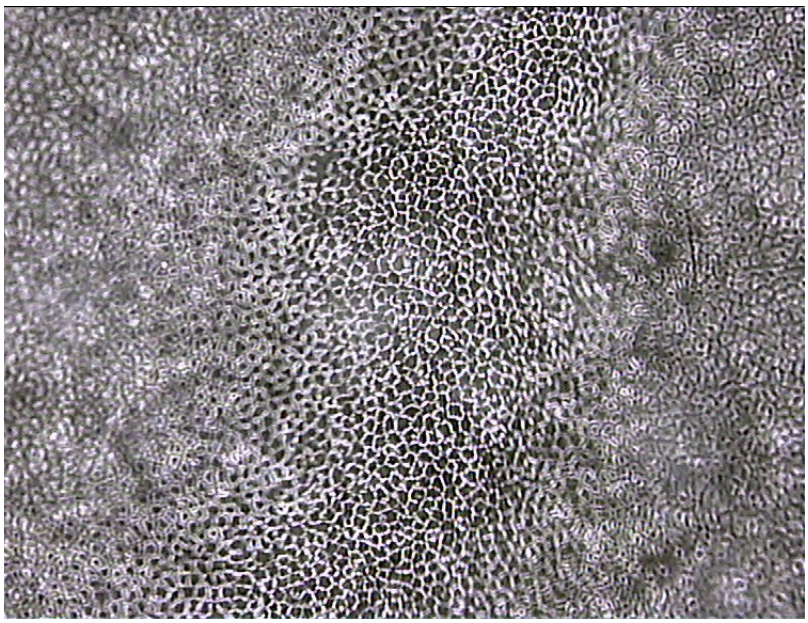


*Figura 4.4: Esempio 3 di immagine proveniente dal dataset*

Si può notare infatti che la qualità di alcune delle regioni di tutte le immagini è talmente bassa da non poter effettuare alcuna elaborazione.
Nei prossimi paragrafi verrà illustrata la strategia utilizzata per cercare di superare tutte queste criticità.

### 4.4.1 Preparazione del dataset

Come prima operazione si sono prese tutte le immagini che presentavano delle regioni con una definizione buona e da queste si sono estratte delle regioni della dimensione di *144x144 pixel*. Dopo questa operazione si sono ottenute 58 immagini con definizione soddisfacente, come mostrato in *figura 4.5*.

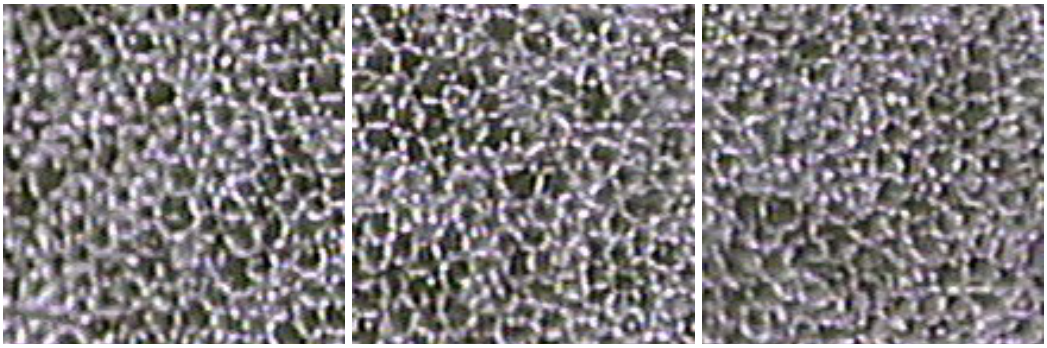


*Figura 4.5: Tre esempi di immagini estratte dal dataset.*

### 4.4.2 Creazione del ground truth

Come verrà spiegato nel *paragrafo 4.5*, in questo lavoro la ground truth realizzata è composta da una immagine avente sfondo nero e un punto rosso di dimensione *1 x 1 pixel* in corrispondenza della posizione del centro di una cellula. Questa operazione è

stata eseguita manualmente per tutte e 58 le immagini che sono state estratte dal dataset fornito grazie al software gratuito GIMP [23]. In *figura 4.6*, *figura 4.7* e *figura 4.8* vengono riportati tre esempi dove lo sfondo nero è stato sostituito con uno sfondo giallo per poterle apprezzare al meglio.

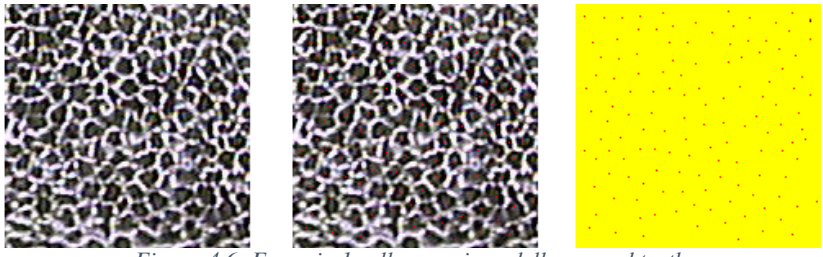


*Figura 4.6: Esempio 1 sulla creazione della ground truth*

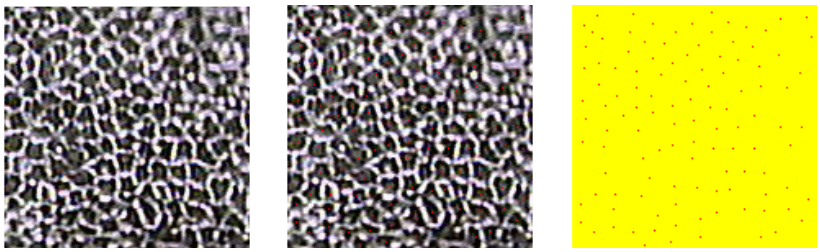


*Figura 4.7: Esempio 2 sulla creazione della ground truth*

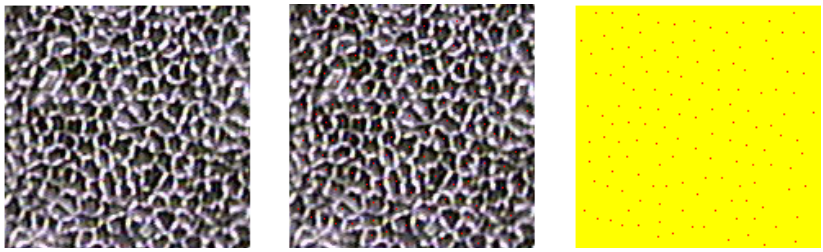


*Figura 4.8: Esempio 3 sulla creazione della ground truth*

### 4.4.3 Data augmentation

Sotto il nome di data augmentation vanno tutte quelle tecniche che permettono di espandere il dataset grazie ad opportune modifiche alle immagini che si hanno a propria disposizione. Nell'ambito della bioingegneria queste tecniche sono quasi sempre necessarie quando si usano algoritmi di machine learning in quanto, solitamente, i dataset a disposizione contano poche decine di elementi. Nel nostro caso, si ha a disposizione solamente 58 immagini. A titolo di esempio si pone all'attenzione del lettore la famosa architettura VGGNet [24] che ha lo scopo di riconoscere oggetti comuni nelle immagini; la fase di allenamento di questa rete è stata effettuata con un dataset composto da 1.3 milioni di immagini.
Considerato che le immagini a disposizione hanno una qualità mediamente bassa si è deciso di non applicare trasformazioni che modifichino il contenuto delle immagini, ma ci si è limitati ad applicare delle semplici rotazioni in modo da non alterare il rapporto segnale-rumore *(SNR)* del dataset. Ogni rotazione eseguita sul dataset originario ci permetterà di aumentarne le dimensioni di un fattore moltiplicativo senza dover modificare il contenuto delle immagini.

Visto che il dataset a disposizione è di piccole dimensioni si è deciso di eseguire l'operazione di data augmentation "offline". Questa strategia ha permesso di aver maggior controllo sulla qualità finale delle immagini dopo che sono state applicate le varie trasformazioni, oltre ad essere implementabile più facilmente. Le operazioni di data augmentation devono essere applicate contemporaneamente a ciascuna immagine del dataset e alla sua relativa ground truth per far si che a ciascuna immagine corrisponda la sua corretta ground truth.
Per eseguire le operazioni di rotazione sul dataset si è utilizzato XnConvert [25], un software gratuito che permette di modificare grandi quantità di immagini quasi istantaneamente. Dopo aver caricato nel software le 58 immagini provenienti dal dataset e le 58 immagini di ground truth le operazioni eseguite sono state:

- Rotazione di 90°
- Rotazione di 180°
- Rotazione di 270°

Queste tre operazioni di rotazione ci hanno permesso di quadruplicare le dimensioni del dataset, passando 58 elementi a 232. Dopo diverse prove sperimentali è risultato che anche questo nuovo dataset è di dimensioni troppo piccole
Per espandere ulteriormente le immagini a disposizione si è quindi deciso di estrarre da ciascuna immagine del dataset e dalla sua relativa ground truth 9 patch di dimensioni *72 x 72 pixel*. In *figura 4.9* viene mostrato uno schema volto a mostrare la strategia con cui queste patch sono state ottenute.

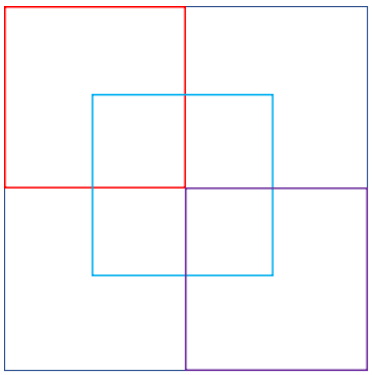


*Figura 4.9: Schema dell'estrazione delle patch*

Il quadrato esterno blu rappresenta l'immagine di partenza avente dimensione *144 x 144 pixel*. Il quadrato rosso rappresenta la prima patch in posizione *(x = 0, y = 0),* quello blu in posizione *(x = 36, y = 36)* e quello rosso in posizione *(x = 72, y = 72)*. Per *x = 0* verranno poi estratte due ulteriori patch, una in corrispondenza di *y = 36* e un'altra in corrispondenza di *y = 72*. Dopo di che si passa alla posizione *x = 36,* dalla quale verranno estratte tre ulteriori patch in corrispondenza di *y = 0, y = 36* (quadrato azzurro) e *y = 72*. Si estraggono infine le ultime 3 patch a partire dalla posizione *x = 72* come fatto per i due casi precedenti.
Dopo aver eseguito questa operazione le patch ottenute sono *2088* e saranno quelle utilizzate in seguito per la fase di training, validazione e test della rete neurale.

## 4.5 Architettura delle reti neurali utilizzate

Come già affermato in precedenza, lo sviluppo di algoritmi di conta cellulare in ambito biologico o medico è molto importante; nel nostro caso riuscire a stabilire la densità delle cellule nell'endotelio corneale può aiutare l'oftalmologo a stabilire lo stato di salute dell'occhio del paziente. Il problema di conteggio cellulare automatico può essere approcciato in due modi: conteggiare le cellule dopo averle identificate, approccio che ne richiede la segmentazione [26], oppure stimare la densità cellulare senza la necessità di identificare le cellule [27]. In questo elaborato seguiremo il secondo approccio.
Seguendo i risultati ottenuti in [27], poniamo il problema di conteggio cellulare come un problema di apprendimento supervisionato che cerca di imparare una mappa che lega un'immagine *I(x)* alla sua mappa di densità *D(x)*, cioè $F: I(x) \rightarrow D(x)\ con\ I \in R^{m \times n}, D \in R^{m \times n}$ con $n \times m$ dimensione delle immagini. La mappa di densità *D(x)* è una funzione che ha come dominio i pixel dell'immagine e integrando questa sull'area di una regione dell'immagine otteniamo la stima del numero di cellule in quella regione. La mappa di densità *D(x)* si ottiene convolvendo le immagini di ground truth con un filtro Gaussiano. Lo scopo dei modelli utilizzati sarà quindi ottenere, attraverso regressione, la mappa di densità *D(x)* dalla corrispondente immagine *I(x)*.
In questo elaborato useremo due architetture di reti neurali per effettuare la regressione delle mappe di densità corrispondenti alle immagini del dataset. La prima è denominata Fully convolutional regression network [28] mentre la seconda è denominata U-Net [29].

### 4.5.1 Fully convolutional regression network

L'architettura di questa rete neurale presenta normali blocchi di convoluzione-ReLu-Max Pooling negli strati iniziali della rete come avviene nelle architetture di reti neurali convoluzionali più comuni. Quando l'immagine passa attraverso questi strati subirà una perdita di risoluzione causata dall'operazione di max pooling eseguita per tutti gli strati iniziali della rete. Per recuperare le informazioni spaziali perse verranno utilizzati dei blocchi upsample-ReLu-convoluzione negli strati successivi a quelli iniziali. Durante la fase di upsampling verrà utilizzata inizialmente un'operazione di interpolazione bilineare, seguita da un'operazione di convoluzione eseguita con filtri che verranno appresi durante la fase di allenamento della rete. La rete utilizza filtri di dimensioni *3x3 pixel* per tutta la rete e tre operazioni di max pooling per aggregare le informazioni spaziali. Gli stessi autori di questa rete hanno proposto una variante di questa, che consiste nell'utilizzo di filtri *5x5* per tutta la rete e due operazioni di max pooling per l'aggregazione delle informazioni spaziali. La prima variante verrà denominata in seguito come FCRN_A, mentre la seconda FCRN_B. In questo elaborato verrà utilizzata solamente FCRN_A, di cui è possibile vedere l'architettura in *figura 4.10*.

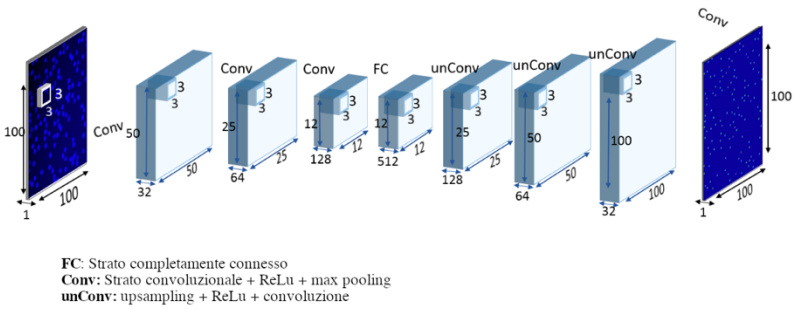



*Figura 4.10: Architettura FCRN_A [28].*

L'implementazione di FCRN_A in Python si può trovare nell'*Appendice 2*.

### 4.5.2 U-Net

L'architettura di U-Net (*figura 4.11)* consiste in un percorso di contrazione dove le immagini subiscono down-sampling seguito da un percorso di espansione dove le immagini subiscono up-sampling in modo tale che le dimensioni dell'output siano le stesse dell'input. Il percorso di contrazione consiste nella applicazione ripetuta di blocchi convoluzione-ReLu-max pooling. Nella fase di contrazione la dimensione dei filtri utilizzati nella fase di convoluzione è *3x3 pixel*, mentre l'operazione di max pooling ha dimensione *2x2*. Il percorso di espansione consiste di blocchi up-sampling-convoluzione che vengono concatenati al percorso di contrazione per poi eseguire altre due convoluzioni ciascuna delle quali seguita da ReLu.
Questa architettura è nata con lo scopo di segmentare immagini dove sono presenti cellule quindi andrà modificata affinché essa sia in grado di predire la mappa di densità cellulare di cui si è già discusso. Le modifiche effettuate consistono nella sostituzione delle funzioni di loss utilizzate originariamente dagli autori; si sono quindi rimosse le funzioni di entropia incrociata e funzione esponenziale normalizzata e al loro posto è stata utilizzata la funzione di errore quadratico medio.
L'implementazione di U-Net in Python si può trovare nell'*Appendice 2*.

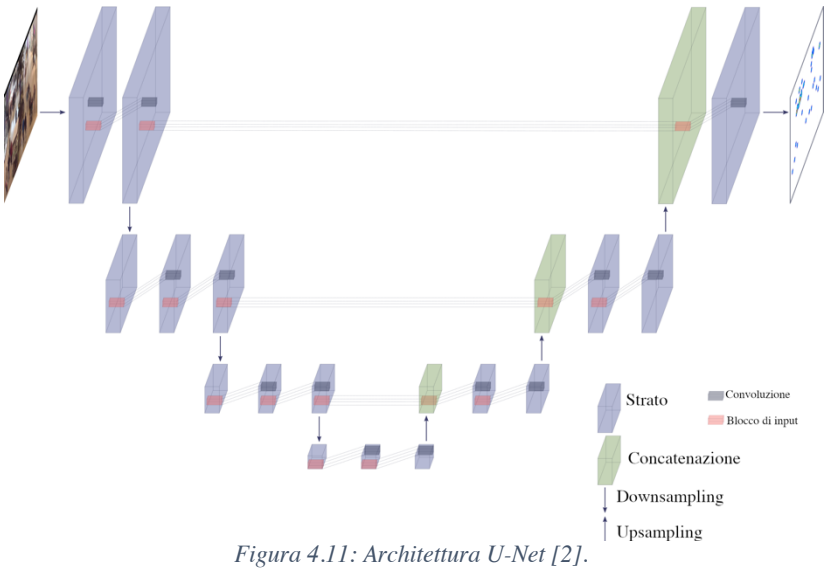



*Figura 4.11: Architettura U-Net [2].*

# Capitolo 5

# Prove sperimentali

In questo capitolo verrà presentata la modalità con cui è stata eseguita la fase di training, validazione e test della rete U-Net e della rete FCRN_A. Verranno quindi presentati i risultati ottenuti in fase di training e in seguito, verranno presentati i risultati ottenuti nella fase di test. Il codice che implementa la fase di training/validazione dei due modelli viene presentato nell'*Appendice 2,* mentre il codice che implementa la fase di test viene presentato nell'*Appendice 3*.

## 5.1 Organizzazione del dataset

Il dataset ottenuto dopo le operazioni di data augmentation è stato così suddiviso:

- 1546 patch per la fase di training
- 522 patch per la fase di validazione
- 20 patch per la fase di test

Per evitare situazioni di overfit possibilmente causate dall'elaborazione sequenziale eseguita sul dataset ogni volta che si effettua la fase di training si esegue un mescolamento casuale del dataset.

## 5.2 Funzione di loss e ottimizzatore utilizzati

La funzione di loss utilizzata durante la fase di training e validazione di entrambi i modelli è la funzione di errore quadratico medio, una delle più utilizzate nei problemi di regressione. La formula che la implementa è la seguente:

$$MSE = \frac{\sum_{i=1}^{n}(t_i - t_i^p)^2}{n}$$

Dove $t_i$ rappresenta il numero vero di cellule presenti nella patch, $t_i^p$ il numero di cellule che la rete ha rilevato, mentre $n$ è il numero di immagini utilizzate. L'algoritmo di ottimizzazione utilizzato per entrambi i modelli è Adamax, una variante dell'algoritmo di ottimizzazione stocastica Adam [30]. La scelta di questo algoritmo di ottimizzazione è dovuta al fatto che esso necessita di soli due parametri da impostare, tasso di apprendimento e decadimento dei pesi, a differenza di altri algoritmi dove i parametri da impostare sono più numerosi. I due modelli utilizzano inoltre un tasso di apprendimento dinamico, che consiste nella riduzione progressiva del tasso di apprendimento della rete man mano che si procede nella fase di training. Questa tecnica, infatti, consente di tenere un tasso di apprendimento più elevato nella fase iniziale di training in modo tale avvicinarci da convergere più facilmente al minimo della funzione di loss. La fase di training è stata inoltre eseguita con la tecnica di *mini-batch*, che consiste nell'allenare la rete con un piccono insieme di dati. Questa

tecnica è stata utilizzata in quanto permette di evitare situazioni di overfit [31] a discapito del tempo necessario a completare la fase di training.

## 5.3 Risultati ottenuti della fase di training del modello U-Net

Per la fase di training della rete U-Net sono stati utilizzati i seguenti parametri:

- Tasso di apprendimento: *1e-5*
- Diminuzione del tasso di apprendimento: 10% ogni 23 epoche.
- Decadimento dei pesi: *1e-4*
- Dimensione mini-batch: *8*
- Epoche: 200

Il tempo impiegato a svolgere la fase di training sul server gpu1 è di poco superiore a due ore.
In figura 5.1 e figura 5.2 viene riportato, rispettivamente l'andamento della funzione di loss durante la fase di training e quella di validazione.

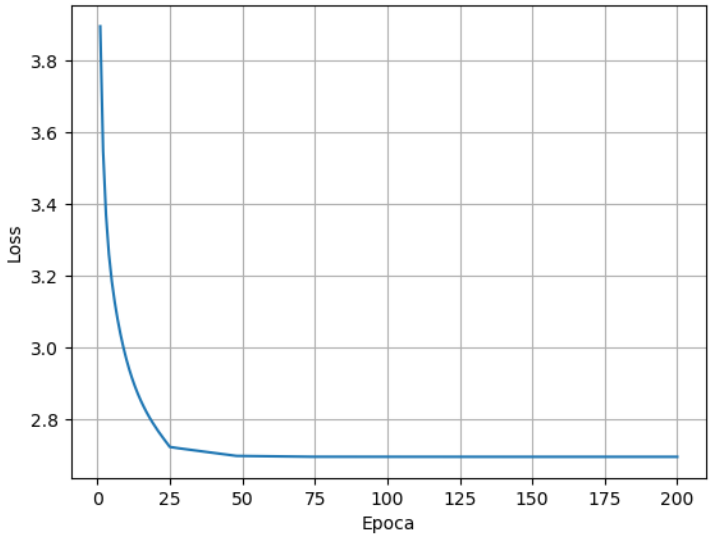



*Figura 5.1: Andamento della funzione loss durante la fase di training per il modello U-Net*

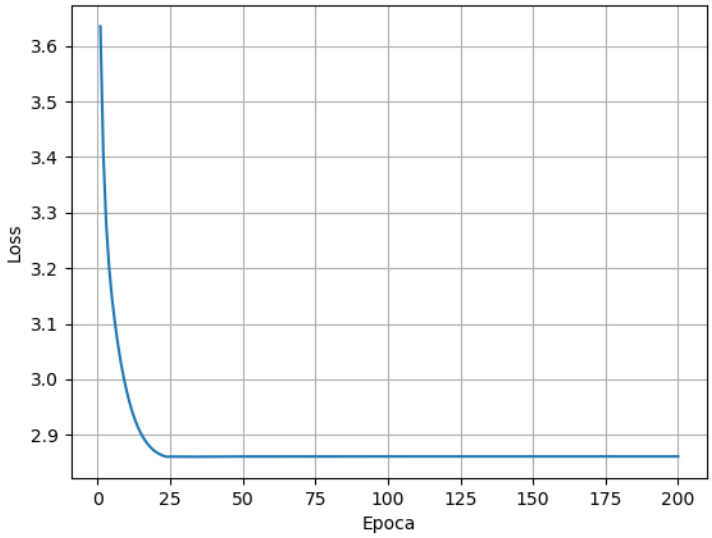



*Figura 5.2: Andamento della funzione di loss nella fase di validazione per il modello U-Net*

Il miglior modello ottenuto dalla fase di training ha ottenuto i seguenti risultati nella fase di validazione:

- Errore medio: 1.097
- Deviazione standard dell'errore: 3.46
- Quantità di cellule individuate: 95.8%
- Indice di correlazione di Pearson: 0.62.

Nella *figura 5.2* e *figura 5.3* è possibile invece il grafico che mostra la regressione lineare tra il numero di cellule realmente presente in una immagine e il valore stimato dal modello per gli insiemi di training e validazione.

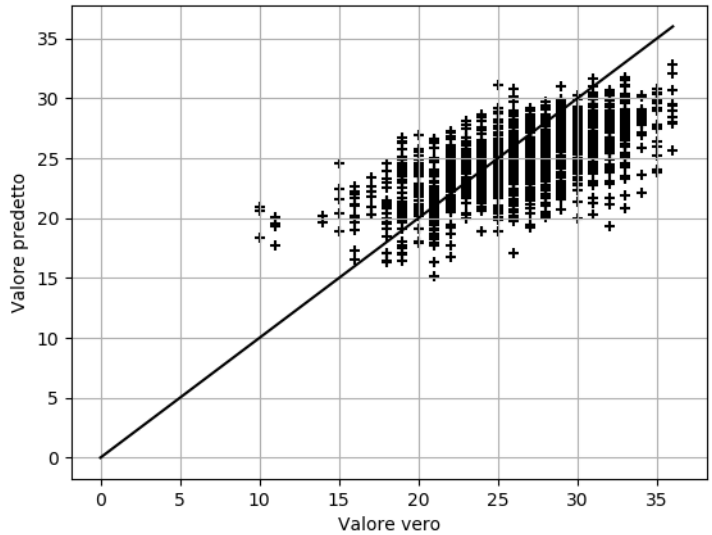



*Figura 5.3: Grafico della regressione tra valore vero e valore predetto per il modello U-Net nella fase di training.*

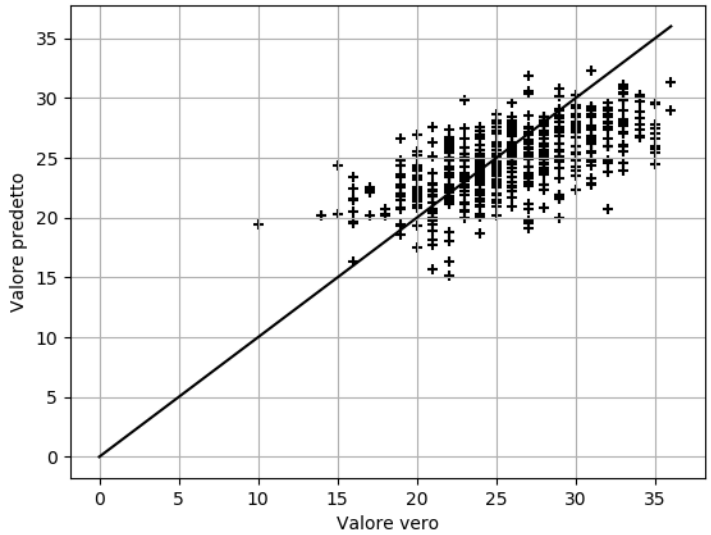



*Figura 5.4: Grafico della regressione tra valore vero e valore predetto per il modello U-Net nella fase di validazione.*

## 5.4 Risultati ottenuti della fase di training del modello FCRN_A

Per la fase di training della rete FCRN_A sono stati utilizzati i seguenti parametri:

- Tasso di apprendimento: *1e-2*
- Diminuzione del tasso di apprendimento: 10% ogni 2 epoche.
- Decadimento dei pesi: *1e-4*
- Dimensione mini-batch: *8*
- Epoche: *100*

Il tempo impiegato a svolgere la fase di training sul server gpu1 è di poco superiore a 4 ore.
In figura 5.5 e figura 5.6 viene riportato, rispettivamente l'andamento della funzione di loss durante la fase di training e quella di validazione.

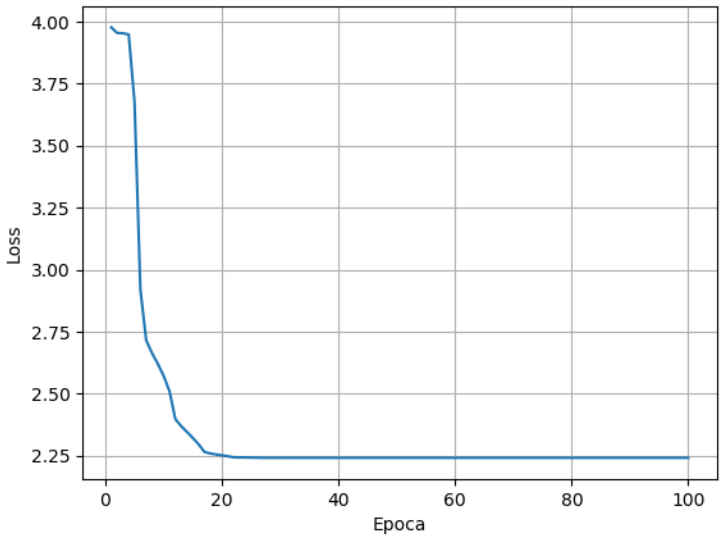



*Figura 5.5: Andamento della funzione loss durante la fase di training per il modello FCRN_A*

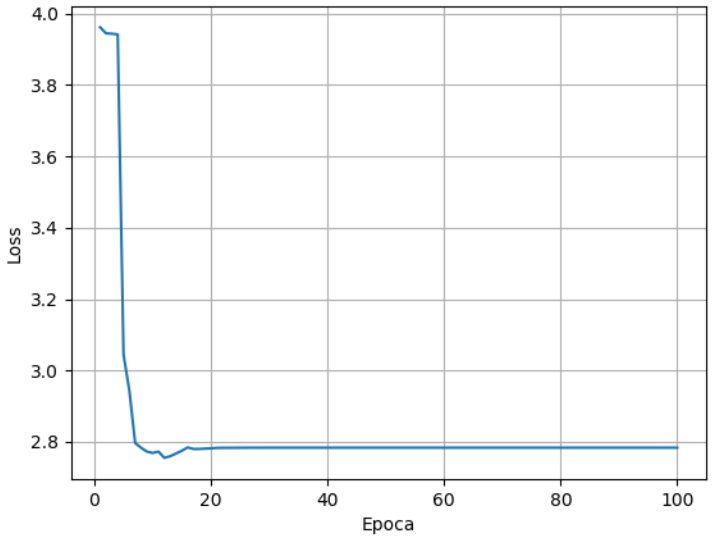



*Figura 5.6: Andamento della funzione loss durante la fase di validazione per il modello FCRN_A*

Il miglior modello ottenuto dalla fase di training ha ottenuto i seguenti risultati nella fase di validazione:

- Errore medio: 0.559
- Deviazione standard dell'errore: 3.13
- Quantità di cellule individuate: 97.9%
- Indice di correlazione di Pearson: 0.71.

Nella *figura 5.7* e *figura 5.8* è possibile invece il grafico che mostra la regressione lineare tra il numero di cellule realmente presente in una immagine e il valore stimato dal modello per gli insiemi di training e validazione.

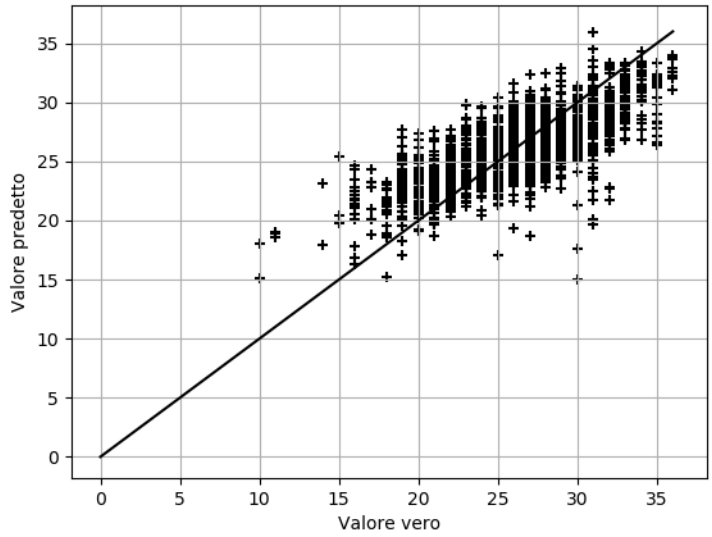



*Figura 5.7: Grafico della regressione tra valore vero e valore predetto per il modello FCRN_A nella fase di training.*

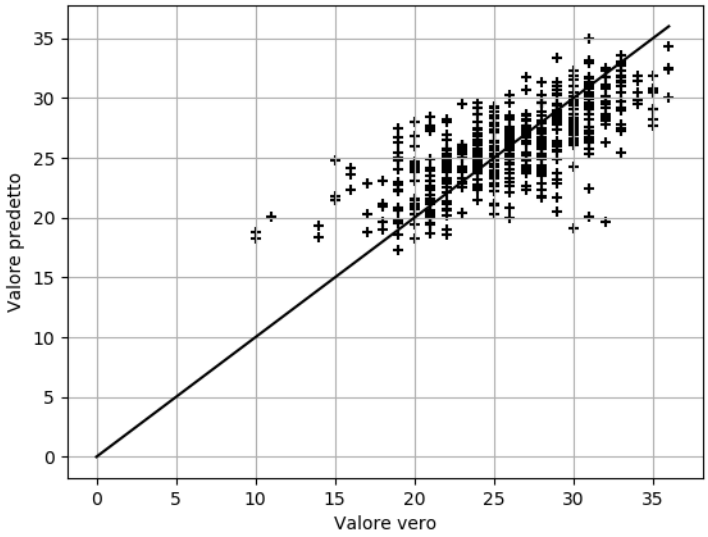



*Figura 5.8: Grafico della regressione tra valore vero e valore predetto per il modello FCRN_A nella fase di validazione*

## 5.5 Test dei modelli allenati

In questa fase l'insieme di immagini di test verranno sottoposte ai modelli U-Net ed FCRN_A ottenuti dopo la fase di training/validazione. Ottenuta la stima della quantità di cellule presenti in ogni immagine fornita in ingresso, si confronteranno i risultati ottenuti con la stima effettuata da un operatore umano.

### 5.5.1 Risultati ottenuti dal test del modello U-Net

Nella tabella sottostante è possibile osservare il confronto tra il numero di cellule rilevate dal modello U-Net e dall'operatore umano.

| Immagine # | conta automatica | conta manuale |
|---|---|---|
| 1 | 25 | 24 |
| 2 | 27 | 24 |
| 3 | 30 | 30 |
| 4 | 22 | 28 |
| 5 | 24 | 27 |
| 6 | 24 | 27 |
| 7 | 28 | 32 |
| 8 | 22 | 29 |
| 9 | 28 | 31 |
| 10 | 28 | 31 |
| 11 | 25 | 26 |
| 12 | 20 | 15 |
| 13 | 28 | 25 |
| 14 | 25 | 27 |

| | | |
|---|---|---|
| 15 | 25 | 26 |
| 16 | 29 | 32 |
| 17 | 26 | 31 |
| 18 | 22 | 23 |
| 19 | 20 | 19 |
| 20 | 24 | 24 |

In *figura 5.9* viene riportata la retta di regressione tra la quantità di cellule stimate automaticamente e dall'operatore umano.

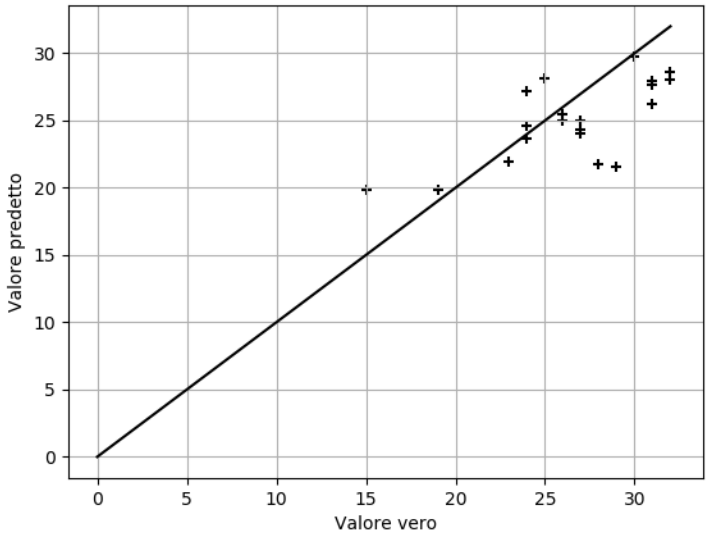



*Figura 5.9: Retta di regressione ottenuta dal modello U-Net sul sull'insieme di test*

In *figura 5.10*, *figura 5.11* e *figura 5.12* vengono riportati 3 immagini di test alle quali è stata sovrapposta la mappa di densità stimata manualmente e la mappa di densità stimata dal modello U-Net.

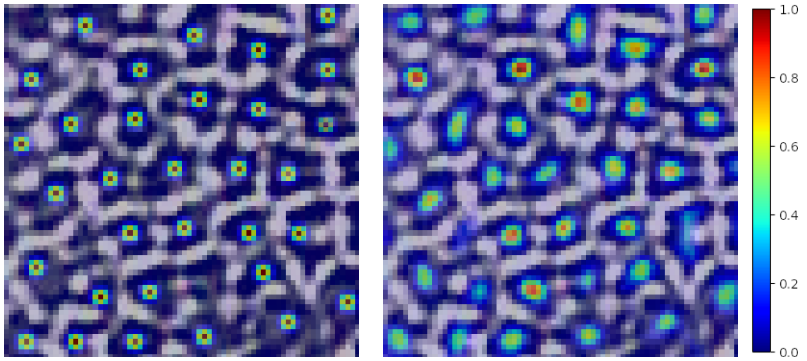



*Figura 5.10: Immagine originale #2 dell'insieme di test a cui è stata sovrapposta la mappa di densità stimata manualmente(sinistra) e la mappa ottenuta dal modello U-Net (destra).*

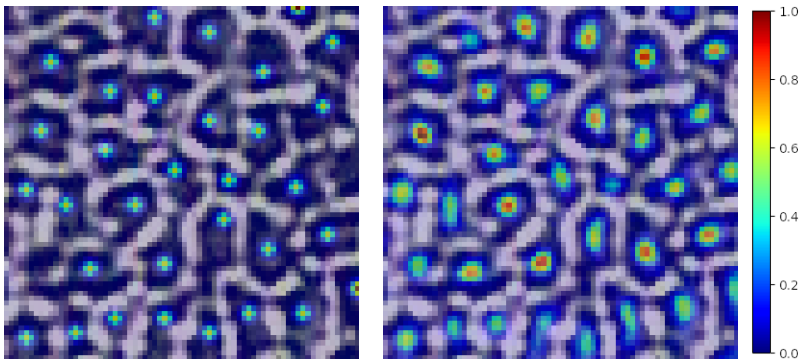



*Figura 5.11: Immagine originale #9 dell'insieme di test a cui è stata sovrapposta la mappa di densità stimata manualmente(sinistra) e la mappa ottenuta dal modello U-Net (destra).*

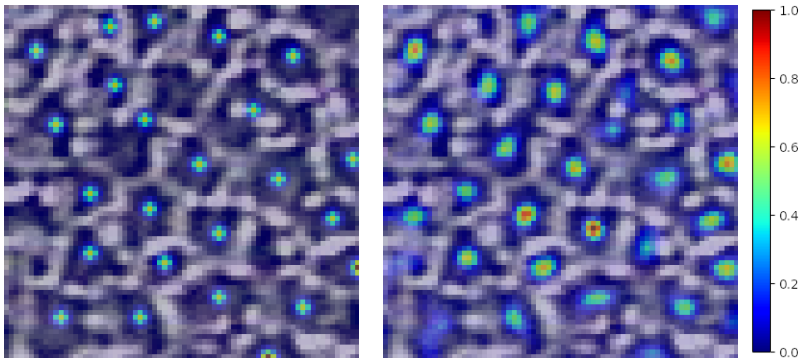



*Figura 5.12: Immagine originale #19 dell'insieme di test a cui è stata sovrapposta la mappa di densità stimata manualmente(sinistra) e la mappa ottenuta dal modello U-Net (destra).*

Il modello ha rilevato il 94.3% delle cellule presenti nelle immagini di test. L'errore quadratico medio commesso dal modello è pari a 2,75 mentre l'errore medio è 1.53 $\pm$ 3.05. L'indice di correlazione di Pearson tra i due insiemi di valori è invece 0.702.

### 5.5.2 Risultati ottenuti dal test del modello FCRN_A

Nella tabella sottostante è possibile osservare il confronto tra il numero di cellule rilevate dal modello FCRN_A e dall'operatore umano.

| Immagine # | conta automatica | conta manuale |
|---|---|---|
| 1 | 23 | 25 |
| 2 | 25 | 23 |
| 3 | 32 | 35 |
| 4 | 28 | 23 |
| 5 | 28 | 29 |
| 6 | 26 | 26 |
| 7 | 22 | 21 |
| 8 | 26 | 22 |
| 9 | 28 | 30 |
| 10 | 31 | 29 |

| | | |
|---|---|---|
| 11 | 26 | 26 |
| 12 | 20 | 21 |
| 13 | 31 | 28 |
| 14 | 27 | 25 |
| 15 | 27 | 29 |
| 16 | 28 | 27 |
| 17 | 22 | 21 |
| 18 | 20 | 24 |
| 19 | 28 | 26 |
| 20 | 28 | 29 |

In *figura 5.13* viene riportata la retta di regressione tra la quantità di cellule stimate automaticamente e dall'operatore umano.

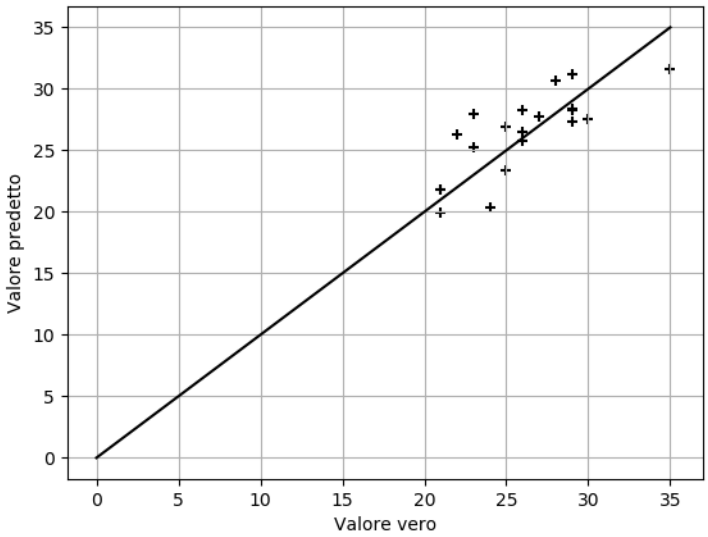



*Figura 5.13: Retta di regressione ottenuta dal modello FCRN_A sul sull'insieme di test*

In *figura 5.14*, *figura 5.15* e *figura 5.16* vengono riportati 3 immagini di test affiancate dalla mappa di densità stimata dal modello U-Net.

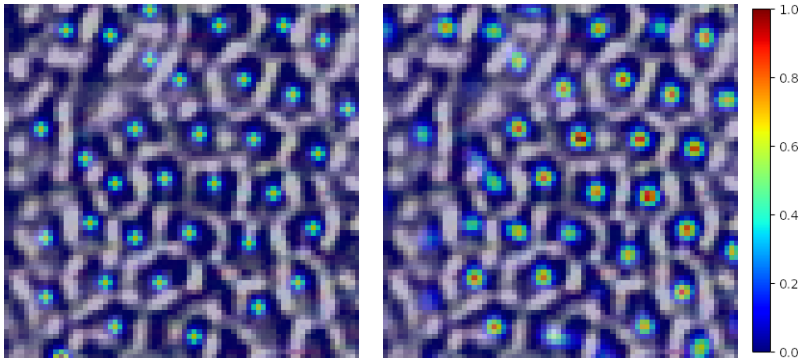



*Figura 5.14: Immagine originale #2 dell'insieme di test a cui è stata sovrapposta la mappa di densità stimata manualmente(sinistra) e la mappa ottenuta dal modello FCRN_A (destra).*

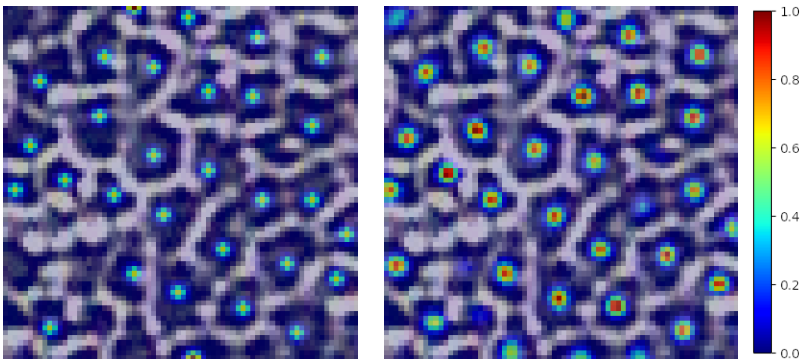


*Figura 5.15: Immagine originale #4 dell'insieme di test a cui è stata sovrapposta la mappa di densità stimata manualmente(sinistra) e la mappa ottenuta dal modello FCRN_A (destra).*

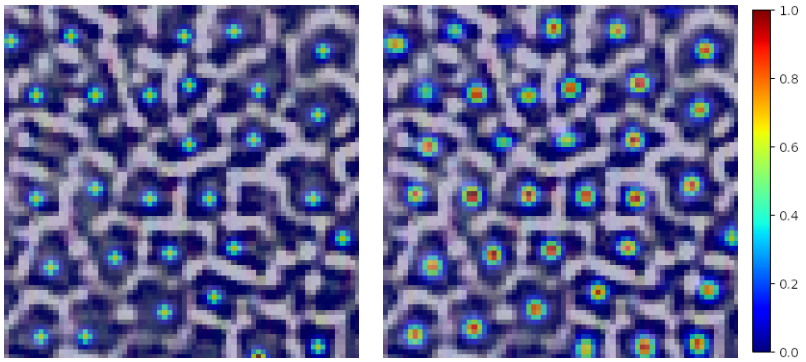


*Figura 5.16: Immagine originale #9 dell'insieme di test a cui è stata sovrapposta la mappa di densità stimata manualmente(sinistra) e la mappa ottenuta dal modello FCRN_A (destra).*

Il modello ha sovrastimato il numero di cellule presenti nelle immagini di test dell'1.5%. L'errore quadratico medio commesso dal modello è pari a 2,67 mentre l'errore medio è -0.39 $\pm$ 2.32. L'indice di correlazione di Pearson tra i due insiemi di valori è invece 0.777.

## 5.6 Test qualitativo dei modelli allenati sulle immagini originali

In questo paragrafo si vogliono valutare qualitativamente le capacità dei due modelli a quantificare la quantità di cellule presenti nelle immagini originali. Considerato che queste presentano sia zone nitide sia zone fuori fuoco, ci si aspetta che i due modelli riescano a conteggiare le cellule solo quando queste sono chiaramente distinguibili. Nelle zone di qualità peggiore ci si aspetta infatti che i modelli utilizzati non riescano ad eseguire la conta cellulare come del resto avverrebbe ad un operatore umano che si trovasse a svolgere questo compito.
Questa fase di test è stata eseguita utilizzando il codice sviluppato nelle precedenti fasi di test, modificato opportunamente affinché il modello possa ricevere in ingresso le immagini con la loro dimensione originale di *768 x 576 pixel*. I modelli utilizzati sono gli stessi che abbiamo ottenuto dopo le fasi di allenamento riportate nel *paragrafo 5.3* e nel *paragrafo 5.4*.
Sono state in ingresso alle due reti neurali 10 immagini originali di endotelio corneale e si sono ottenute le mappe di densità relative a queste immagini. In seguito, le mappe di densità sono state sovrapposte alle immagini originali per valutare qualitativamente

le prestazioni dei due modelli prestando particolare attenzione alle discontinuità tra zone a fuoco e zone fuori fuoco. Per ogni modello verranno presentate una di queste immagini operando degli ingrandimenti opportuni al fine di valutare qualitativamente i risultati ottenuti.

### 5.6.1 U-Net

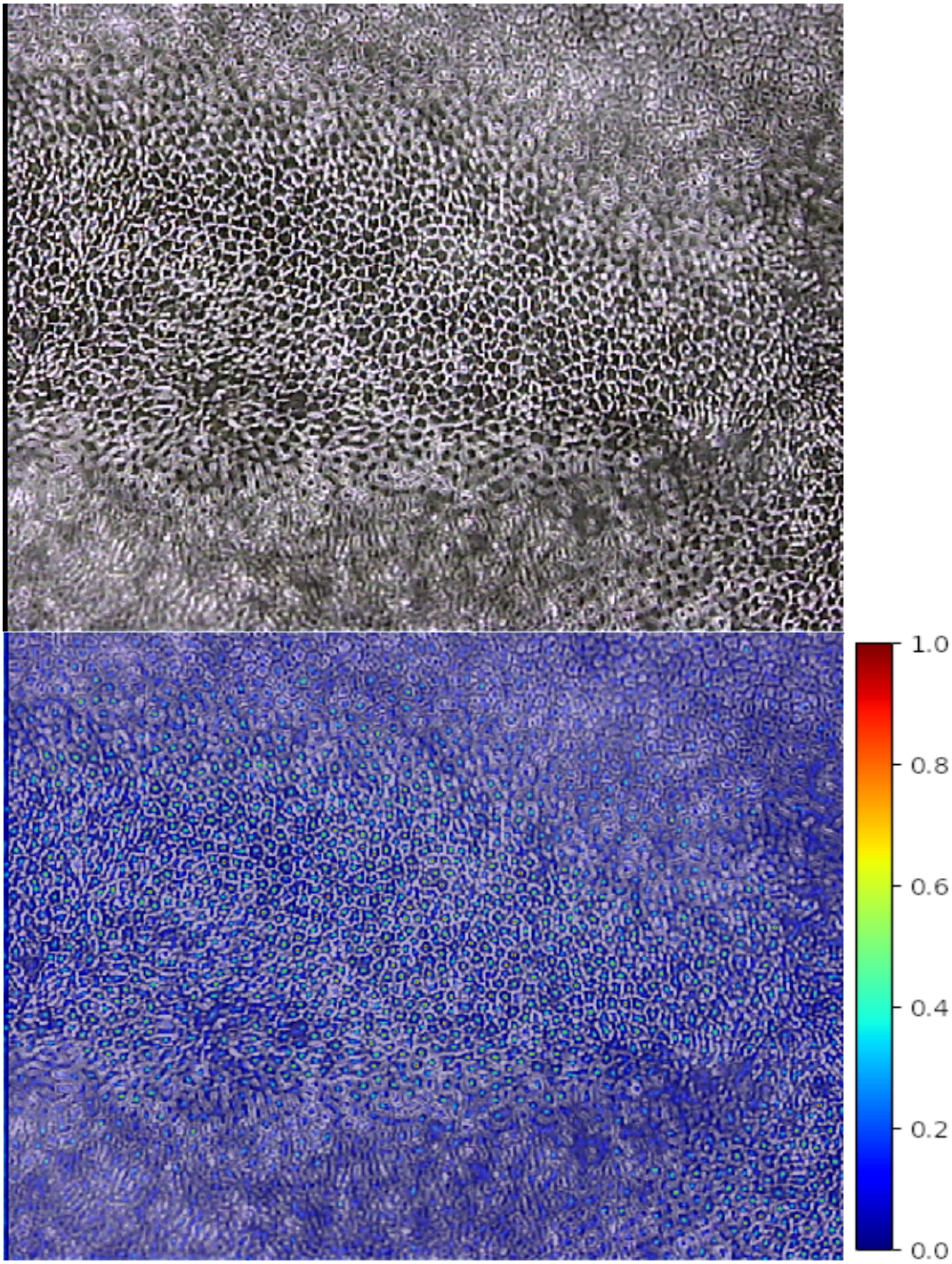



*Figura 5.17: Immagine originale (in alto) e immagine originale a cui è stata sovrapposta la mappa di densità stimata da U-Net (in basso).*

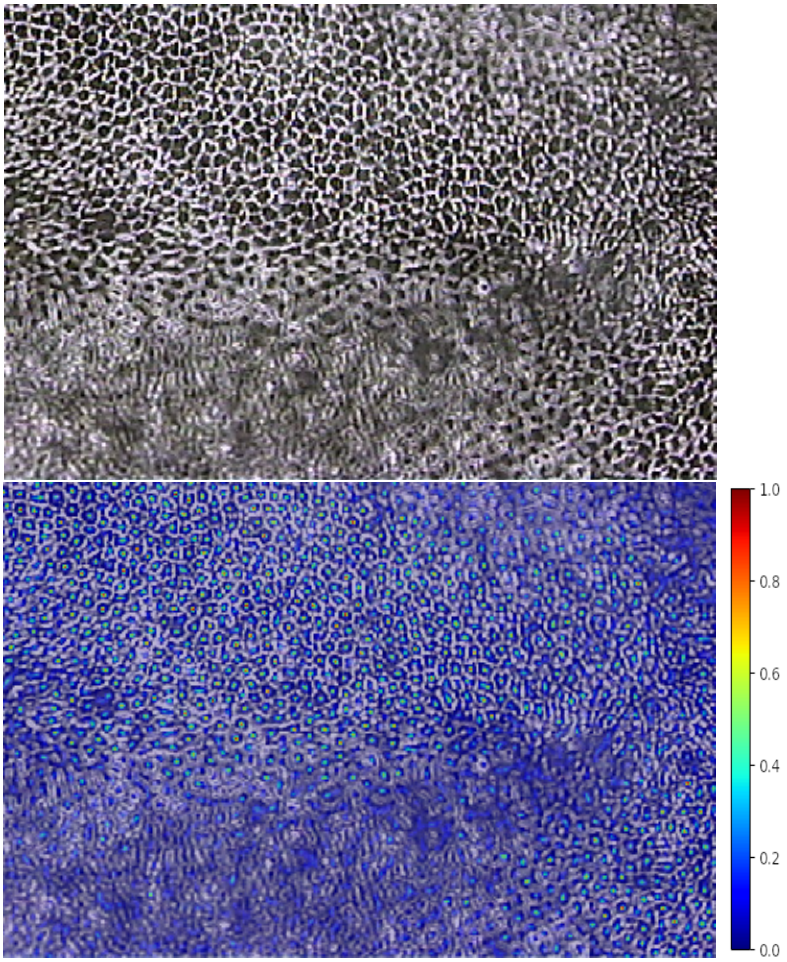
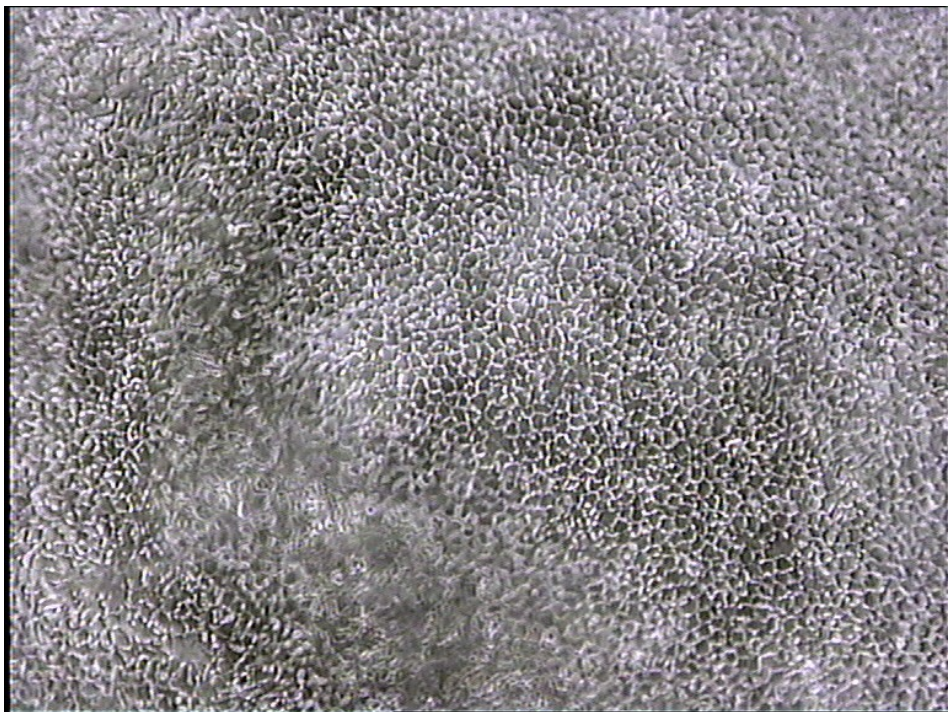



*Figura 5.18: Ingrandimento dell'angolo inferiore sinistro delle due immagini presenti in figura 5.17*

### 5.6.2 FCRN_A



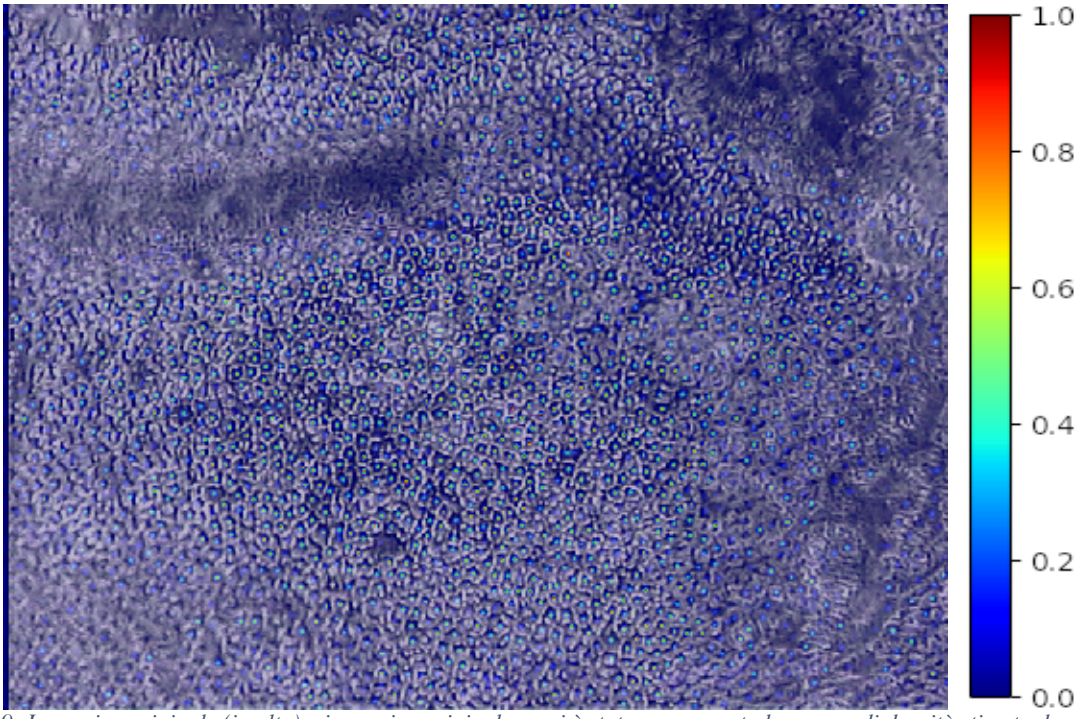



*Figura 5.19: Immagine originale (in alto) e immagine originale a cui è stata sovrapposta la mappa di densità stimata da FCRN_A (in basso).*

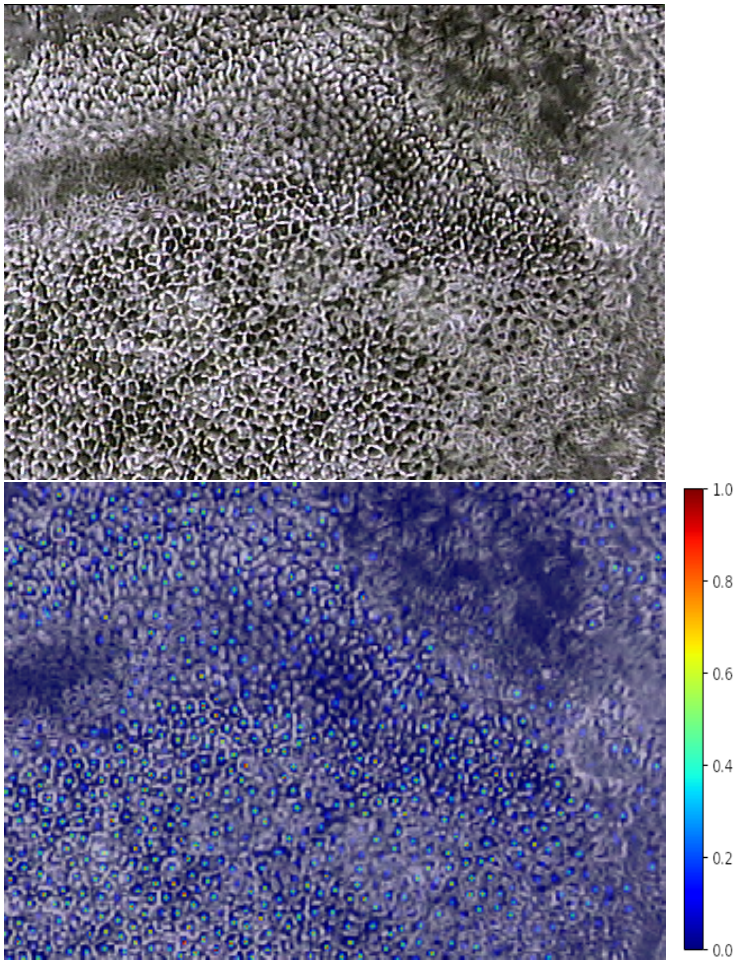



*Figura 5.20: Ingrandimento dell'angolo superiore destro delle due immagini presenti in figura 5.19.*

### 5.6.3 Commenti sui risultati ottenuti

I risultati ottenuti da entrambi i modelli sulle immagini originali di endotelio corneale sono stati in linea con le aspettative. Entrambi i modelli riconoscono e conteggiano le cellule presenti nelle zone delle immagini che presentano buona definizione. Quando si ha una transizione tra una zona nitida e una zona sfuocata entrambi i modelli riescono comunque a conteggiare le cellule, qualora queste siano ancora riconoscibili. Nelle zone completamente fuori fuoco o con presenza di artefatti i modelli non sono più in grado di distinguere la presenza di cellule, comportamento in linea con le aspettative.

# Capitolo 6

# Conclusioni e suggerimenti per studi futuri

In questo elaborato sono state utilizzate due architetture di reti neurali con lo scopo di quantificare il numero di cellule presenti in immagini dell'endotelio corneale acquisite in vivo. Per effettuare la conta cellulare è stato utilizzato un approccio che mira a stimare direttamente la quantità di cellule presenti nelle immagini senza la necessità di effettuare la segmentazione delle stesse. Questo ha infatti consentito un notevole risparmio di tempo nella preparazione del dataset utilizzato per la fase di training delle due reti neurali, in quanto le annotazioni fornite ai due modelli consistono in semplici immagini a sfondo nero con presenza di punti rossi in corrispondenza alla presenza di una cellula e non in immagini segmentate manualmente. Un altro punto di forza del paradigma utilizzato in questo elaborato è la mancanza della necessità di apprendere come segmentare le immagini sulla quale bisogna poi eseguire la conta cellulare, operazione che risulta molto più complessa sia dal punto di vista teorico sia computazionale.
Le architetture di reti neurali qui implementate hanno dimostrato buone prestazioni se confrontate con i risultati ottenuti dall'operatore umano; l'architettura U-Net è riuscita ad individuare il 94.3% delle cellule presenti nelle immagini di test, mentre il modello FCRN_A ha rilevato il 101.5% delle cellule presenti. Sebbene quest'ultima architettura offra prestazioni migliori in termini di precisione non vanno sottovalutate le prestazioni ottenute con il modello U-Net. Questo modello, infatti, sebbene abbia prestazioni sensibilmente inferiori, è allenabile molto più in fretta in quanto è composto da circa 600.000 parametri in confronto al modello FCRN_A che ne conta circa 4.000.000. Da ciò ne consegue che U-Net necessita di tempi notevolmente inferiori per la fase di training rispetto al secondo modello.
Va sottolineato, inoltre, che le prestazioni dei due modelli sono influenzate anche dalla qualità delle immagini e dalle immagini di ground truth preparate dall'operatore umano; come dimostrato in precedenti lavori [32], la conta manuale delle cellule presenti in un'immagine può avere risultati diversi in base all'operatore che la esegue. Risulta quindi difficile fornire agli algoritmi delle immagini di ground truth perfettamente accurate e ciò potrebbe influire sulla qualità dei risultati che si possono ottenere mediante l'utilizzo di un qualsiasi algoritmo di machine learning. Un altro fattore che influenza le prestazioni delle due reti neurali è la quantità di immagini utilizzata per effettuare la fase di training/validazione. In questo elaborato sono state utilizzate solamente 58 immagini, che sono state poi ritagliate e modificate in modo di espandere la quantità di dati a disposizione.
Per futuri studi che utilizzeranno l'approccio qui riportato si suggerisce per prima cosa l'utilizzo di dataset di dimensioni maggiori e di maggior qualità visiva. Utilizzare immagini di migliore qualità perme infatti l'utilizzo di tecniche di data augmentation più sofisticate come le tecniche di deformazioni elastiche delle immagini che permettono di espandere le dimensioni del dataset di diversi ordini di grandezza. Un dataset di maggiori dimensioni può consentire inoltre di ottenere risultati migliori in termini di errori commessi nel conteggio delle cellule. Si suggerisce inoltre l'utilizzo

della tecnica dropout nelle reti neurali implementate, in quanto può aiutare a migliorare le prestazioni che si possono ottenere nella fase di training. Si consiglia infine l'impiego di più operatori che eseguano la conta delle cellule presenti nelle immagini che verranno utilizzate nella fase di training, in modo da ottenere delle immagini di ground truth di qualità più elevata e con una ridotta presenza di errori.

# Appendice 1

# Utilizzare il cluster di calcolo 'Blade':

Il cluster di calcolo 'Blade' adotta il sistema di gestione Sun Grid Engine. Le istruzioni che il cluster di calcolo deve eseguire vanno specificate in un file di testo con estensione '*.job'*, denominati da qui in seguito file job.
Una volta creato il file job, che serve a specificare al server quali script eseguirli e come eseguirli; esso andrà caricato nel sistema e poi sottoposto al sistema di gestione delle code del server.

### Il file 'job'

Il file job può essere creato con un semplice editor di testo non formattato come Microsoft Notepad oppure Apple TextEdit. Nel file job potremo specificare diverse opzioni come:

- Eseguire il codice su server gpu1
- Posizione della cartella di lavoro
- Eventuale presenza di file di output e/o di errore
- Attivare notifiche quando il programma inizia/finisce/si interrompe a causa di errori.

Va sottolineato che il cluster di calcolo 'Blade' di base utilizza Python in versione 2.7 nella quale sono assenti molte delle librerie usate nel codice elaborato. Bisognerà quindi specificare esplicitamente al cluster di calcolo la necessità di utilizzare Python in versione 3.7.
Si riporta in *figura A1.1* un esempio di file job che richiede al cluster di calcolo di eseguire un codice con Python 3.7 su server gpu1, che è l'unico a supportare le librerie Nvidia CUDA; specifichiamo inoltre che l'output del codice venga salvato nel file '*output.txt'* mentre l'eventuale presenza di errori nel file *'errore.txt'* e che ci venga notificato quando lo script inizia ad essere eseguito e quando finisce.

cuda.job

```
#!/bin/bash
#
# Imposta il valore 'name' di qstat
#$ -N test
#
# Richiedo utilizzo del server gpu1
#$ -q gpu
#
# Reindirizzo l'output in un file di testo
#$ -o output.txt
#
# Reindirizzo gli errori in un file di testo
#$ -e error.txt
#
#
# Richiedo una notifica a inizio job, a fine job e in caso di errori
#$ -m eab
#
# Attivo Python 3.7
source /nfsd/opt/anaconda3/anaconda3.sh
conda activate /nfsd/opt/anaconda3/tensorflow

python3 train.py -n UNet -lr 1e-4 -e 150 --batch_size 8 --plot
```

*Figura A1.1 Esempio su come creare un file job*

Una volta che il file job è pronto bisognerà caricarlo nel server prima di sottoporlo al sistema di gestione Sun Grid Engine.

## Caricare i file nel cluster di calcolo

Affinché il codice possa essere eseguito, questo deve essere caricato all'interno del server insieme al dataset.
Per caricare tutti i file necessari all'interno del cluster di calcolo bisogna utilizzare un programma che supporti il protocollo di trasferimento dati SFTP. Si possono utilizzare diversi programmi per eseguire il trasferimento di file in modo sicuro, quello utilizzato per questo elaborato è *Filezilla [33]*; questo programma oltre che essere di facile utilizzo è anche gratuito.
Installato il programma bisognerà connettersi in remoto al cluster di calcolo all'indirizzo *login.dei.unipd.it* utilizzando il proprio account personale. In *figura A1.2* è possibile vedere la schermata con i vari parametri da configurare.

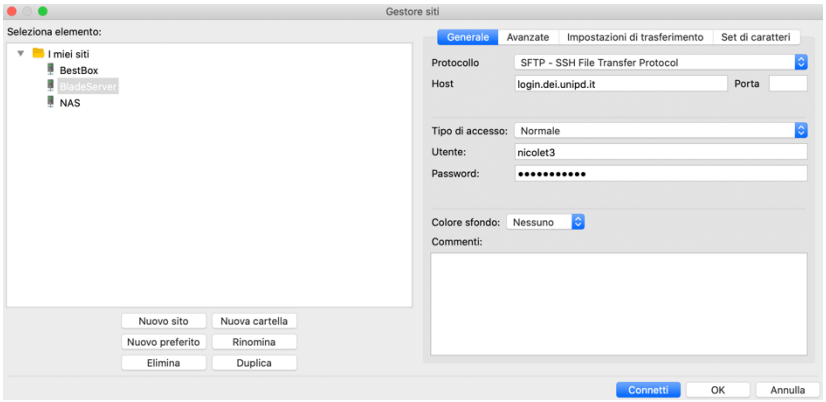



*Figura A1.2 Stabilire una connessione remota con il cluster di calcolo 'Blade'*

Stabilita la connessione sarà necessario creare una cartella di lavoro e caricare all'interno il dataset, il codice da eseguire e il file job che specifica le istruzioni che il server deve eseguire. In *figura A1.3* è presente la schermata dove è possibile vedere che tutti i file sono stati caricati correttamente all'interno del server remoto.

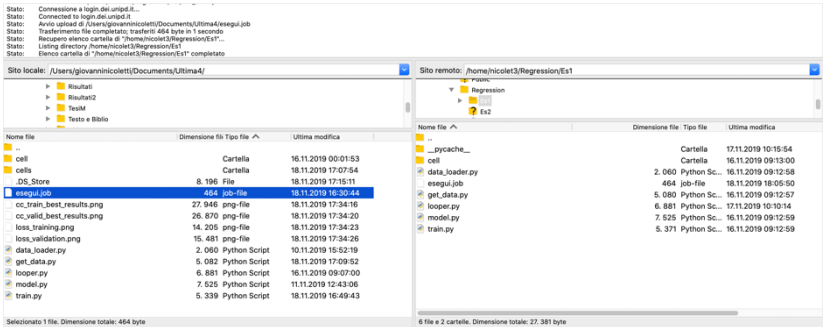


*Figura A1.3: Nella finestra a destra è possibile vedere i file caricati nella cartella di lavoro*

## Sottoporre il file job al Sun Grid Engine

Dopo aver caricato correttamente tutti i file necessari all'esecuzione del codice e il file job, questo va sottoposto al sistema di gestione della coda Sun Grid Engine tramite il frontend del cluster di calcolo. Per accedere al frontend dovremo stabilire una connessione *ssh* con il server grazie all'uso di un terminale di comandi. L'indirizzo attraverso la quale si accede al server è *nome@dei.unipd.it*. Al posto di 'nome' bisognerà inserire il nome utente del proprio account DEI e quando verrà richiesta bisognerà inserire la relativa password. In *figura A1.4* viene presentato un esempio di tentativo di accesso al 'frontend' del cluster di calcolo 'Blade'.

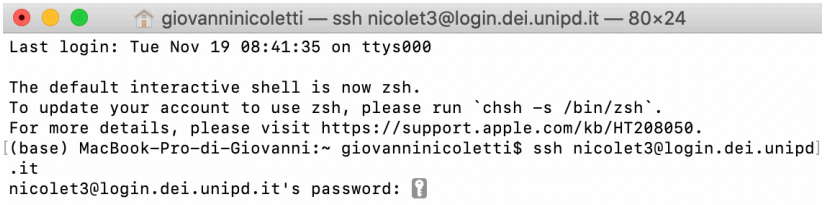



*Figura A1.4: Login al frontend del cluster di calcolo 'Blade'*

Stabilita la connessione con il server ci apparirà un messaggio di benvenuto come mostrato in *figura A1.5* dopo la quale saremo pronti a sottomettere i nostri file job.

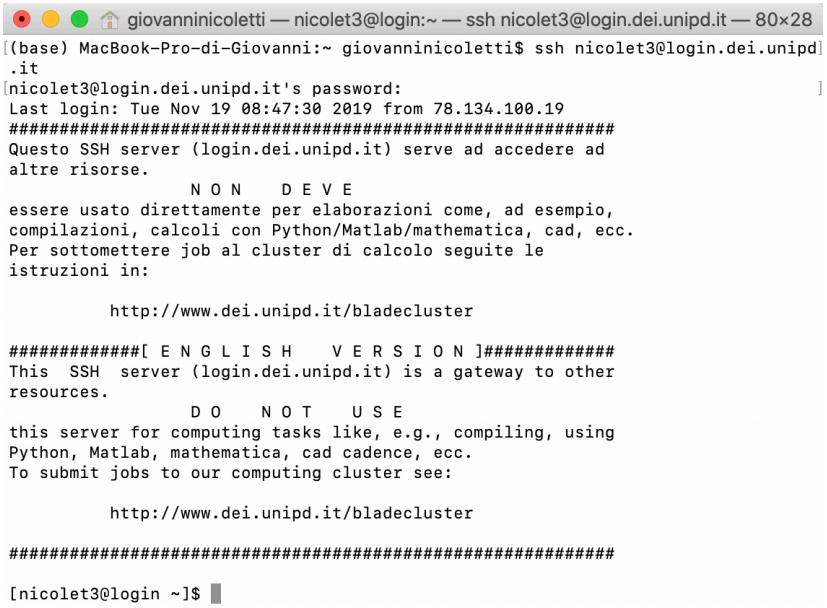



*Figura A1.5: Messaggio di benvenuto dopo che la connessione con il cluster di calcolo è stata stabilita con successo*

Ora bisognerà spostarci nella cartella di lavoro dove sono presenti i nostri file e il file job mediante il comando *'cd'* e poi possiamo sottomettere il file job attraverso il comando *'qsub'* come mostrato nella *figura A1.6*.

```
[nicolet3@login ~]$ cd Regression
[nicolet3@login Regression]$ cd Es1
[nicolet3@login Es1]$ qsub esegui.job
Your job 2668211 ("test") has been submitted
```

*Figura A1.6: Sottomissione del file job*

A questo punto la richiesta di esecuzione del codice verrà messa in coda e non appena il cluster di calcolo avrà le sufficienti risorse il codice passerà in esecuzione. Per vedere se questo è in attesa di essere eseguito oppure se è in esecuzione è necessario usare il comando *'qstat'* come mostrato in *figura A1.7*. Un codice in coda presenterà la dicitura *'qw'* nella colonna *'state'*, mentre un codice in esecuzione presenterà la dicitura *'r'*.

```
[nicolet3@login Es1]$ qstat
job-ID  prior   name       user         state submit/start at     queue
                 slots ja-task-ID
-----------------------------------------------------------------------------
-----------------------------------
2668195 0.40032 test       nicolet3     r     11/19/2019 08:48:19 Q@runner-19.de
i.unipd.it           1
2668211 0.00000 test       nicolet3     qw    11/19/2019 10:34:11
                 1
```

*Figura A1.7: Stato della coda di esecuzione del cluster di calcolo 'Blade'*

## Recuperare i risultati dopo l'esecuzione del codice

Una volta che il codice finisce di essere eseguito il sistema manderà una notifica via e-mail che ci darà anche alcune informazioni utili, come il tempo impiegato per l'esecuzione come si può vedere in *figura A1.8*.

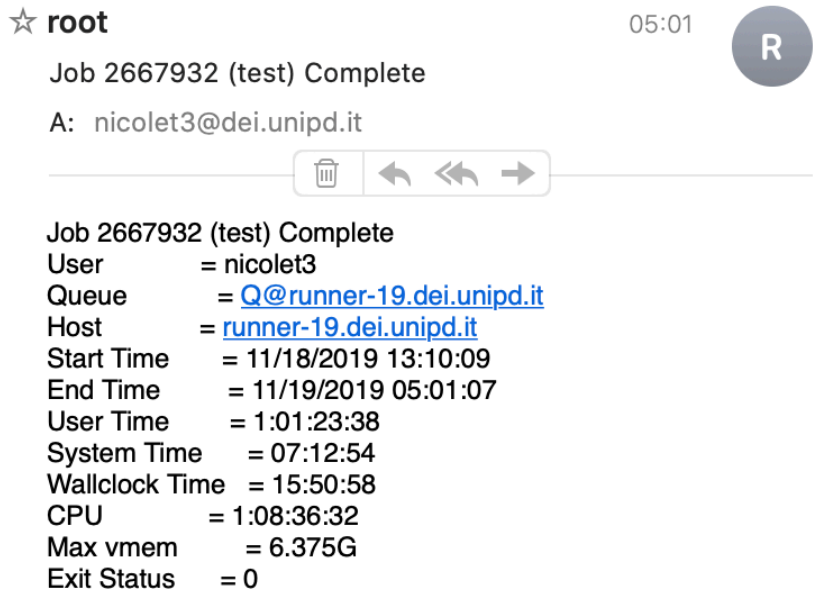



*Figura A1.8: Notifica di fine esecuzione.*

Per recuperare i risultati bisognerà connettersi nuovamente al cluster di calcolo via SFTP utilizzando *Filezilla*, entrare nella cartella di lavoro e scaricare tutti i file risultato che il codice ha prodotto durante la sua esecuzione. In questo caso, gli errori verranno salvati nel file *'errore.txt'*, mentre tutti gli output prodotti sulla console di Python verranno salvati nel file *'output.txt'* come abbiamo specificato nel file job.

# Appendice 2

# Implementazione dei modelli FCRN_A e U-Net in Python

```
from typing import Tuple
import torch
from torch import nn


def conv_block(channels: Tuple[int, int],
               size: Tuple[int, int],
               stride: Tuple[int, int]=(1, 1),
               N: int=1):
    """
    Creo un blocco con N strati convoluzionali e funzione di attivazione ReLu
    Il primo strato ha dimensioni IN x OUT, i rimanenti OUT x OUT.

    Argomenti:
        channels: (IN, OUT) - no. di canali di input e output
        size: dimensione del kernel (fissato per tutte le convoluzioni in un blocco)
        stride: stride (fissato per tutte le convoluzioni in un blocco)
        N: no. di strati convoluzionale.

    Restituisce:
        Un container sequenziale di N strati convoluzionali.
    """
    # Singola convoluzione + normalizzazione del batch + blocco ReLU
    block = lambda in_channels: nn.Sequential(
        nn.Conv2d(in_channels=in_channels,
                  out_channels=channels[1],
                  kernel_size=size,
                  stride=stride,
                  bias=False,
                  padding=(size[0] // 2, size[1] // 2)),
        nn.BatchNorm2d(num_features=channels[1]),
        nn.ReLU()
    )
    # Crea e restituisce un container sequenziale di strati convoluzionali
    # Dimensioni dell'input = channels[0] per il primo blocco e channels[1] per i restanti
    return nn.Sequential(*[block(channels[bool(i)]) for i in range(N)])


class ConvCat(nn.Module):
    """Convoluzione con upsampling + concatenzazione del blocco."""

    def __init__(self,
                 channels: Tuple[int, int],
                 size: Tuple[int, int],
                 stride: Tuple[int, int]=(1, 1),
                 N: int=1):
        """
        Creo un container sequenziale con bloccho convoluzionale
        con N strati convoluzionale e fattore di upsampling pari a 2
        """
        super(ConvCat, self).__init__()
        self.conv = nn.Sequential(
            conv_block(channels, size, stride, N),
            nn.Upsample(scale_factor=2)
        )

    def forward(self, to_conv: torch.Tensor, to_cat: torch.Tensor):
        """Passaggion in avanti (Forward).
```

```
59.
60.         Aromenti in ingresso:
61.             to_conv: input passato al blocco convoluzionale e di upsampling
62.             to_cat: input concatenato con l'output di un blocco convoluzionale
63.         """
64.         return torch.cat([self.conv(to_conv), to_cat], dim=1)
65.
66.
67. class FCRN_A(nn.Module):
68.     """
69.     Fully Convolutional Regression Network A
70.
71.     Ref. W. Xie et al. 'Microscopy Cell Counting with Fully Convolutional
72.     Regression Networks'
73.     """
74.
75.     def __init__(self, N: int=1, input_filters: int=3, **kwargs):
76.         """
77.         Creo il modello FCRN-A con:
78.
79.             * dimensione del kernel fissa = (3, 3)
80.             * dimensione del kernel di max pooling fissa = (2, 2) e fattore di upsampling = 2
81.             * no. di filtri come definito nel modello originale:
82.               dimensione input -> 32 -> 64 -> 128 -> 512 -> 128 -> 64 -> 1
83.
84.         Argomenti in ingresso:
85.             N: no. di strati convoluzionali per blocco
86.             input_filters: no. di canali dell'input
87.         """
88.         super(FCRN_A, self).__init__()
89.         self.model = nn.Sequential(
90.
91.             # downsampling
92.             conv_block(channels=(input_filters, 32), size=(3, 3), N=N),
93.             nn.MaxPool2d(2),
94.
95.             conv_block(channels=(32, 64), size=(3, 3), N=N),
96.             nn.MaxPool2d(2),
97.
98.             conv_block(channels=(64, 128), size=(3, 3), N=N),
99.             nn.MaxPool2d(2),
100.
101.             # "Convoluzionale completamente connesso"
102.             conv_block(channels=(128, 512), size=(3, 3), N=N),
103.
104.             # upsampling
105.             nn.Upsample(scale_factor=2),
106.             conv_block(channels=(512, 128), size=(3, 3), N=N),
107.
108.             nn.Upsample(scale_factor=2),
109.             conv_block(channels=(128, 64), size=(3, 3), N=N),
110.
111.             nn.Upsample(scale_factor=2),
112.             conv_block(channels=(64, 1), size=(3, 3), N=N),
113.         )
114.
115.     def forward(self, input: torch.Tensor):
116.         """Passaggio in avanti (forward)."""
117.         return self.model(input)
118.
119.
120. class UNet(nn.Module):
121.     """
122.     Implementazione di U-Net.
123.
124.     Ref. O. Ronneberger et al. "U-net: Convolutional networks for biomedical
```

```
125.    image segmentation."
126.    """
127.
128.    def __init__(self, filters: int=64, input_filters: int=3, **kwargs):
129.        """
130.        Creo un modello U-Net con:
131.
132.            * dimensione del kernel fissa = (3, 3)
133.            * dimensione del kernel di max pooling fissa = (2, 2) e fattore di upsampling = 2
134.            * no. costante di strati convoluzionali per blocco = 2 (see conv_block)
135.            * no. costante di filtri per strato convoluzionale
136.
137.        Args:
138.            filters: no. di filtri per strato convoluzionale
139.            input_filters: no. di canali di input
140.        """
141.        super(UNet, self).__init__()
142.
143.        # Primo blocco dimensione del canale
144.        initial_filters = (input_filters, filters)
145.
146.        # Dimensione del canale per downsampling
147.        down_filters = (filters, filters)
148.
149.        # Dimensione del canale per upsampling (input raddoppiato a causa della concatenzazione)
150.        up_filters = (2 * filters, filters)
151.
152.        # Downsampling
153.        self.block1 = conv_block(channels=initial_filters, size=(3, 3), N=2)
154.        self.block2 = conv_block(channels=down_filters, size=(3, 3), N=2)
155.        self.block3 = conv_block(channels=down_filters, size=(3, 3), N=2)
156.
157.        # Upsampling
158.        self.block4 = ConvCat(channels=down_filters, size=(3, 3), N=2)
159.        self.block5 = ConvCat(channels=up_filters, size=(3, 3), N=2)
160.        self.block6 = ConvCat(channels=up_filters, size=(3, 3), N=2)
161.
162.        # Predizione della densità
163.        self.block7 = conv_block(channels=up_filters, size=(3, 3), N=2)
164.        self.density_pred = nn.Conv2d(in_channels=filters, out_channels=1,
165.                                      kernel_size=(1, 1), bias=False)
166.
167.    def forward(self, input: torch.Tensor):
168.        """Passaggio in avanti (forward)."""
169.        # Uso per tutta la rete la stessa dimensione del kernel di max pooling
170.        pool = nn.MaxPool2d(2)
171.
172.        # Downsampling
173.        block1 = self.block1(input)
174.        pool1 = pool(block1)
175.        block2 = self.block2(pool1)
176.        pool2 = pool(block2)
177.        block3 = self.block3(pool2)
178.        pool3 = pool(block3)
179.
180.        # Upsampling
181.        block4 = self.block4(pool3, block3)
182.        block5 = self.block5(block4, block2)
183.        block6 = self.block6(block5, block1)
184.
185.        # Predizione della densità
186.        block7 = self.block7(block6)
187.        return self.density_pred(block7)
```

# Appendice 3

# Script per il training delle due reti neurali:

```
import os
import click
import torch
import numpy as np
import matplotlib
matplotlib.use('agg')
from matplotlib import pyplot

from data_loader import H5Dataset
from looper import Looper
from model import UNet, FCRN_A

def train(network_architecture: str,
          learning_rate: float,
          epochs: int,
          batch_size: int,
          unet_filters: int,
          convolutions: int,
          plot: bool):

    """Alleno il modello scelto sul dataset."""
    dataset_name = 'cell'

    # Utilizzo GPU se disponibile
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

    dataset = {}     # dataset di training e validazione in formato HDF5
    dataloader = {}  # dataloaders di training e validazione

    for mode in ['train', 'valid']:
        # I file HDF5 sono in dataset_name/(train | valid).h5
        data_path = os.path.join(dataset_name, f"{mode}.h5")

        dataset[mode] = H5Dataset(data_path)

        dataloader[mode] = torch.utils.data.DataLoader(dataset[mode],
                                                       batch_size=batch_size)

    # Numero di canali utilizzato dal dataset
    input_channels = 3

    # Inizializzo il modello specificato in network_architecture
    network = {
        'UNet': UNet,
        'FCRN_A': FCRN_A
    }[network_architecture](input_filters=input_channels,
                            filters=unet_filters,
                            N=convolutions).to(device)
    network = torch.nn.DataParallel(network)

    # Inizializzo loss, ottimizzatore e learning rate scheduler
    loss = torch.nn.MSELoss()
    optimizer = torch.optim.Adamax(network.parameters(),
                                   lr=learning_rate,
                                   weight_decay= 1e-4)

    lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                   step_size=23,
```

```
                                    gamma=0.1)

    # Se il flag del plot è attivo creo un live plot aggiornato da Looper
    if plot:
        pyplot.ion()
        fig1, plots1 = pyplot.subplots(nrows=1, ncols=1)
        fig2, plots2 = pyplot.subplots(nrows=1, ncols=1)
        fig3, plots3 = pyplot.subplots(nrows=1, ncols=1)
        fig4, plots4 = pyplot.subplots(nrows=1, ncols=1)
        fig5, plots5 = pyplot.subplots(nrows=1, ncols=1)
    else:
        plots1 = [None]
        plots2 = [None]
        plots3 = [None]
        plots4 = [None]

    # creo Loopers di allenamento e validazione per la singola epoca
    train_looper = Looper(network, device, loss, optimizer,
                          dataloader['train'], len(dataset['train']), plots1, plots2,
                          validation = False)
    valid_looper = Looper(network, device, loss, optimizer,
                          dataloader['valid'], len(dataset['valid']), plots3, plots4, plots5,
                          validation=True)

    # Miglior risultato corrente (calcolando il minor errore assoluto medio sulla validazione)
    current_best = np.inf

    for epoch in range(epochs):
        print(f"Epoch {epoch + 1}\n")

        # Eseguo un epoca di allenamento e aggiorno il learning rate
        train_looper.run()
        lr_scheduler.step()

        # Eseguo epoca di validazione
        with torch.no_grad():
            result = valid_looper.run()

        # Salvo un checkpoint se il risultato è il migliore di quelli ottenuti
        if result < current_best:
            current_best = result
            torch.save(network.state_dict(),
                       f'{dataset_name}_{network_architecture}.pth')

            print(f"\nNuovo miglior risultato: {result}")

            cc_t_strFile = "cc_train_best_results.png"
            cc_v_strFile = "cc_valid_best_results.png"
            l_t_strFile = "loss_training.png"
            l_v_strFile = "loss_validation.png"
            P_corr_strFile = "Pearson_correlation.png"

            if os.path.isfile(cc_t_strFile):
                os.remove(cc_t_strFile)
                os.remove(cc_v_strFile)

            fig1.savefig(cc_t_strFile)
            fig3.savefig(cc_v_strFile)


        print("\n", "-"*80, "\n", sep='')
        fig2.savefig(l_t_strFile)
        fig4.savefig(l_v_strFile)
        fig5.savefig(P_corr_strFile)

    print(f"[Allenamento eseguito] Miglior risultato: {current_best}")
```

```
125.
126. if __name__ == '__main__':
127.     train()
```

# Codice della funzione Looper:

```
1.  from typing import Optional
2.  import torch
3.  import numpy as np
4.  from scipy.stats import pearsonr
5.  import matplotlib
6.
7.
8.  class Looper():
9.      """Looper gestisce una singola epoca con risultati e plot."""
10.
11.     def __init__(self,
12.                  network: torch.nn.Module,
13.                  device: torch.device,
14.                  loss: torch.nn.Module,
15.                  optimizer: torch.optim.Optimizer,
16.                  data_loader: torch.utils.data.DataLoader,
17.                  dataset_size: int,
18.                  cc_plots: Optional[matplotlib.axes.Axes]=None,
19.                  l_plots: Optional[matplotlib.axes.Axes]=None,
20.                  corr_plots: Optional[matplotlib.axes.Axes]=None,
21.                  validation: bool=False):
22.         """
23.         Inizializzo Looper.
24.
25.         Args:
26.             network: modello già inizializzato
27.             device: dispositivo sulla quale il modello lavora
28.             loss: funzione costo
29.             optimizer: ottimizzatore già inizializzato legato ai parametri della rete
30.             data_loader: data loader già inizializzato
31.             dataset_size: no. di campioni nel dataset
32.             plot: assi matplotlib
33.             validation: flag per impostare le modalità di allenamento e validazione
34.
35.         """
36.         self.network = network
37.         self.device = device
38.         self.loss = loss
39.         self.optimizer = optimizer
40.         self.loader = data_loader
41.         self.size = dataset_size
42.         self.validation = validation
43.         self.cc_plots = cc_plots
44.         self.l_plots = l_plots
45.         self.corr_plots = corr_plots
46.         self.running_loss = []
47.         self.running_pearson = []
48.
49.     def run(self):
50.         """Eseguo una singola epoca.
51.
52.         Restituisce:
53.             Errore medio assoluto.
54.         """
55.
```

```
56.         # Elimino i risultati correnti e aggiungo spazio per la running loss
57.         self.true_values = []
58.         self.predicted_values = []
59.         self.running_loss.append(0)
60.         self.running_pearson.append(0)
61.
62.         # Imposto la modalità: allenamento o validazione
63.         self.network.train(not self.validation)
64.
65.         for image, label in self.loader:
66.             # Sposto immagini ed etichette nel dispositivo
67.             image = image.to(self.device)
68.             label = label.to(self.device)
69.
70.             # Se in modalità di training pulisco il gradiente accumulato
71.
72.             if not self.validation:
73.                 self.optimizer.zero_grad()
74.
75.             # Ottengo la predizione del modello (una mappa di densità)
76.             result = self.network(image)
77.
78.             # Calcolo la loss e aggiorno la running loss
79.             loss = self.loss(result, label)
80.             self.running_loss[-1] += image.shape[0] * loss.item() / self.size
81.
82.             # Aggiorno i parametri della rete se in modalità di allenamento
83.             if not self.validation:
84.                 loss.backward()
85.                 self.optimizer.step()
86.
87.             # Scorro i batch di campioni
88.             for true, predicted in zip(label, result):
89.
90.                 # Integro una mappa di densità per ottenere il no. di oggetti
91.                 # Nota: la mappa di densità era stata normalizzata a 100 * no. di oggetti
92.                 true_counts = torch.sum(true).item() / 100
93.                 predicted_counts = torch.sum(predicted).item() / 100
94.
95.                 # Aggiorno i risultati dell'epoca corrente
96.                 self.true_values.append(true_counts)
97.                 self.predicted_values.append(predicted_counts)
98.
99.         # Calcolo gli errori e le deviazioni standard
100.        self.update_errors()
101.
102.        # Aggiorno il grafico
103.        if self.cc_plots and self.l_plots is not None:
104.            self.plot()
105.
106.        # Stampo il resoconto della corrente epoca
107.        self.log()
108.
109.        return self.mean_abs_err
110.
111.    def update_errors(self):
112.        """
113.        Calcolo errori e deviazioni standard basate sul valore vero e
114.        valore predetto correnti.
115.        """
116.        self.err = [true - predicted for true, predicted in
117.                    zip(self.true_values, self.predicted_values)]
118.
119.        self.err_perc =  sum(self.predicted_values)/sum(self.true_values)
120.
121.        self.abs_err = [abs(error) for error in self.err]
```

```
122.         self.mean_err = sum(self.err) / self.size
123.         self.mean_abs_err = sum(self.abs_err) / self.size
124.         self.std = np.array(self.err).std()
125.
126.         # Calcolo l'indice di correlazione
127.         self.running_pearson[-1], _ = pearsonr(self.predicted_values, self.true_values)
128.
129.
130.     def plot(self):
131.         """ Plot del conteggio veri vs predetti e loss"""
132.
133.         # Conteggio veri vs predetti
134.         true_line = [[0, max(self.true_values)]] * 2  # y = x
135.         self.cc_plots.cla()
136.         self.cc_plots.set_title('Train' if not self.validation else 'Valid')
137.         self.cc_plots.set_xlabel('Valore vero')
138.         self.cc_plots.set_ylabel('Valore predetto')
139.         self.cc_plots.plot(*true_line, 'k-')
140.         self.cc_plots.scatter(self.true_values,self.predicted_values,marker='+',color = k')
141.         self.cc_plots.grid()
142.
143.         # loss
144.         epochs = np.arange(1, len(self.running_loss) + 1)
145.         self.l_plots.cla()
146.         self.l_plots.set_title('Train' if not self.validation else 'Valid')
147.         self.l_plots.set_xlabel('Epoca')
148.         self.l_plots.set_ylabel('Loss')
149.         self.l_plots.plot(epochs, self.running_loss)
150.         self.l_plots.grid()
151.
152.         if self.validation:
153.             self.corr_plots.cla()
154.             self.corr_plots.set_title('Coefficiente di correlazione di Pearson')
155.             self.corr_plots.set_xlabel('Immagini di validazione')
156.             self.corr_plots.set_ylabel('Correlazione')
157.             self.corr_plots.scatter(epochs, self.running_pearson, marker = 'x',color = 'r')
158.             self.corr_plots.grid()
159.
160.         matplotlib.pyplot.pause(0.01)
161.         matplotlib.pyplot.tight_layout()
162.
163.     def log(self):
164.
165.         """Print current epoch results."""
166.         print(f"{'Train' if not self.validation else 'Valid'}:\n")
167.         print(f"\tLoss media: {self.running_loss[-1]:3.4f}\n")
168.         print(f"\tErrore medio: {self.mean_err:3.3f}\n")
169.         print(f"\tErrore medio assoluto: {self.mean_abs_err:3.3f}\n")
170.         print(f"\tSTD errore: {self.std:3.3f}\n")
171.
172.         if self.validation:
173.             print(f"\tErrore percentuale medio: {self.err_perc:3.3f}%\n")
174.             print(f"\tCellule contate vs cellule realmente presenti\n")
175.             a = np.array([[i+1, self.predicted_values[i], self.true_values[i]] for i in    range(self.size)])
176.             args = ["# Immagine", "V. Predetto", "V. Vero"]
177.             frmt = ("{:>12.0f}"+"{:>12.0f}"*2)
178.             print(("{:^15}"*3).format(*args))
179.             for i in a:
180.                 print(frmt.format(*i))
```

# Appendice 4

# Script per il test delle due reti neurali:

```
import os
import torch
import matplotlib
matplotlib.use('agg')
from matplotlib import pyplot

from data_loader import H5Dataset
from TLooper import TLooper
from model import UNet

def test(network_architecture: str,
         plot: bool):

    dataset_name = 'cell'
    network_architecture = 'UNet'

    # Utilizzo GPU se disponibile
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

    dataset = {}     # dataset di training e validazione in formato HDF5
    dataloader = {}  # dataloaders di training e validazione

    for mode in ['test']:
        # I file HDF5 sono in cell/test.h5
        data_path = os.path.join(dataset_name, f"{mode}.h5")

        dataset[mode] = H5Dataset(data_path)

        dataloader[mode] = torch.utils.data.DataLoader(dataset[mode],
                                                       batch_size= 20)

    # Numero di canali utilizzato dal dataset
    input_channels = 3

    # Inizializzo la rete U-Net
    network = {'UNet': UNet }[network_architecture](input_filters=input_channels,
                            filters=64,
                            N=2).to(device)

    network = torch.nn.DataParallel(network)

    # Assegno alla rete i parametri ottenuti dalla fase di training
    network.load_state_dict(torch.load('cell_UNet.pth',map_location=torch.device('cpu')))
    network.eval()

    fig, plots = pyplot.subplots(nrows=1, ncols=1)
    mfig, mplots = pyplot.subplots(nrows=1, ncols=1)

    # Inizializzo loss
    loss = torch.nn.MSELoss()

    # Eseguo un ciclo di test
    test_looper = TLooper(network, loss, device, dataloader['test'], len(dataset['test']), plots, mplots)
    test_looper.run_test()

    # Plot dei risultati
    fig.savefig('test.png')

```

```
59. if __name__ == '__main__':
60.     test(network_architecture = 'UNet', plot = 'true')
```

# Codice della funzione TLooper:

```
1.  from typing import Optional
2.  import torch
3.  import numpy as np
4.  from scipy.stats import pearsonr
5.  import matplotlib
6.  import matplotlib.pyplot as plt
7.
8.
9.  class TLooper():
10.     """Lancio un ciclo di test"""
11.
12.     def __init__(self,
13.                  network: torch.nn.Module,
14.                  loss: torch.nn.Module,
15.                  device: torch.device,
16.                  data_loader: torch.utils.data.DataLoader,
17.                  dataset_size: int,
18.                  test_plots: Optional[matplotlib.axes.Axes]=None,
19.                  map_plots: Optional[matplotlib.axes.Axes]=None):
20.         """
21.         Inizializzo Looper.
22.
23.         Args:
24.             network: modello già inizializzato con i risultati ottenuti nella fase di training
25.             device: dispositivo sulla quale il modello lavora
26.             loss: funzione costo per la fase di test
27.             data_loader: data loader già inizializzato con le immagini di test
28.             dataset_size: no. di campioni utilizzati per la fase di test
29.             plot: assi matplotlib dove verranno visualizzati i risultati
30.         """
31.         self.network = network
32.         self.loss = loss
33.         self.loader = data_loader
34.         self.device = device
35.         self.size = dataset_size
36.         self.test_plots = test_plots
37.         self.map_plots = map_plots
38.         self.running_loss = []
39.         self.running_pearson = []
40.
41.     def run_test(self):
42.
43.         """Eseguo la fase di test
44.
45.         Restituisce:
46.             Errore Medio Assoluto durante la fase di test
47.             Plot dei risulati ottenuti nella fase di test
48.             Log dei risultati ottenuti nella fase di test
49.         """
50.
51.         # Inizializzo i vettori
52.         self.true_values = []
53.         self.predicted_values = []
54.         self.test_loss = []
55.         self.pearson_corr = []
56.         self.orig = []
57.         self.maps = []
58.         self.labels = []
```

```
59.
60.         for image, label in self.loader:
61.             # Sposto immagini ed etichette nel dispositivo scelto
62.             image = image.to(self.device)
63.             label = label.to(self.device)
64.
65.         self.orig = image
66.         self.labels = label
67.         #img = image[10,:,:,:] in self.loader
68.
69.         # Ottengo la predizione del modello (una mappa di densità)
70.         result = self.network(image)
71.         self.maps = result
72.
73.         # Calcolo la loss e aggiorno la running loss
74.         loss = self.loss(result, label)
75.         self.test_loss = image.shape[0] * loss.item() / self.size
76.
77.         # Scorro le immagini di test
78.         for true, predicted in zip(label, result):
79.
80.             # Integro una mappa di densità per ottenere il no. di oggetti
81.             # Nota: la mappa di densità era stata normalizzata a 100 * no. di oggetti
82.             true_counts = torch.sum(true).item() / 100
83.             predicted_counts = torch.sum(predicted).item() / 100
84.
85.             # Aggiorno i risultati
86.             self.true_values.append(true_counts)
87.             self.predicted_values.append(predicted_counts)
88.
89.         # Calcolo gli errori e le deviazioni standard sul test set
90.         self.update_test_errors()
91.
92.         # Faccio il plot dei risultati sul test set
93.         self.plot_test()
94.
95.         self.plot_maps()
96.
97.         # Salvo il log dei risultati sul test set
98.         self.log_test()
99.
100.        return self.mean_abs_err
101.
102.
103.    def update_test_errors(self):
104.
105.        """
106.        Calcolo errori e deviazioni standard basate sul valore vero e
107.        valore predetto dalla rete allenata.
108.        """
109.        self.err = [true - predicted for true, predicted in
110.                    zip(self.true_values, self.predicted_values)]
111.
112.        self.err_perc =  sum(self.predicted_values)/sum(self.true_values)
113.
114.        self.abs_err = [abs(error) for error in self.err]
115.        self.mean_err = sum(self.err) / self.size
116.        self.mean_abs_err = sum(self.abs_err) / self.size
117.        self.std = np.array(self.err).std()
118.
119.        # Calcolo l'indice di correlazione
120.        self.pearson_corr, _  = pearsonr(self.predicted_values, self.true_values)
121.
122.
123.    def plot_test(self):
124.        """ Plot del conteggio veri vs predetti e loss"""
```

```
125.
126.         # Conteggio veri vs predetti
127.         true_line = [[0, max(self.true_values)]] * 2  # y = x
128.         self.test_plots.cla()
129.         self.test_plots.set_title('Test set')
130.         self.test_plots.set_xlabel('Valore vero')
131.         self.test_plots.set_ylabel('Valore predetto')
132.         self.test_plots.plot(*true_line, 'k-')
133.         self.test_plots.scatter(self.true_values, self.predicted_values, marker = '+', color = 'k')
134.         self.test_plots.grid()
135.
136.     def plot_maps(self):
137.         """ Plot del conteggio veri vs predetti e loss"""
138.         fstring = 'IMG/'
139.         mapstring = 'predicted.png'
140.         imgstring = 'orig.png'
141.         labelstring = 'true.png'
142.
143.         for idx in range(1, 20):
144.
145.             # Ottengo le mappe di densità vere dai campioni di test
146.             dlabel = self.labels[idx, :, :, :].detach().permute(1, 2, 0)
147.             oklb = fstring + str(idx) + labelstring
148.             plt.axis('off')
149.             plt.imshow(np.squeeze(dlabel), cmap = 'jet')
150.             plt.savefig(oklb, bbox_inches='tight', transparent=True, pad_inches=0)
151.
152.
153.             # Ottengo le mappe di densità predette dal modello
154.             dmap = self.maps[idx, :, :, :].detach().permute(1, 2, 0)
155.             okms = fstring + str(idx) + mapstring
156.             plt.axis('off')
157.             plt.imshow(np.squeeze(dmap), cmap = 'jet')
158.             plt.savefig(okms, bbox_inches='tight', transparent=True, pad_inches=0)
159.
160.             # Ottengo le immagini corrispondenti alle mappe
161.             dorig = self.orig[idx, :, :, :].detach().permute(1, 2, 0)
162.             okos = fstring + str(idx) + imgstring
163.             plt.axis('off')
164.             plt.imshow(np.squeeze(dorig))
165.             plt.savefig(okos, bbox_inches='tight', transparent=True, pad_inches=0)
166.
167.         # torchvision.utils.make_grid(result[:, :, :, :], 'mappa.png', nrow = 5, padding = 5)
168.
169.         # Immagine originale
170.         #torchvision.utils.save_image(img)
171.
172.         # Mappa di densità
173.
174.         #torchvision.utils.save_image(result[10, :, :, :])
175.
176.
177.     def log_test(self):
178.
179.         """Print current epoch results."""
180.         print(f"Risultati sul test set:\n")
181.         print(f"\tLoss media: {self.test_loss:3.4f}\n")
182.         print(f"\tErrore medio: {self.mean_err:3.3f}\n")
183.         print(f"\tErrore medio assoluto: {self.mean_abs_err:3.3f}\n")
184.         print(f"\tSTD errore: {self.std:3.3f}\n")
185.         print(f"\tIndice di correlazione di Pearson: {self.pearson_corr:3.3f}\n")
186.
187.         print(f"\tErrore percentuale medio: {self.err_perc:3.3f}%\n")
188.         print(f"\tCellule contate vs cellule realmente presenti\n")
189.         a = np.array([[i+1, self.predicted_values[i], self.true_values[i]] for i in range(self.size)])
190.         args = ["# Immagine", "V. Predetto", "V. Vero"]
```

```
191.        frmt = ("{:>12.0f}"+"{:>12.0f}"*2)
192.        print(("{:^15}"*3).format(*args))
193.        for i in a:
194.            print(frmt.format(*i))
```

# Bibliografia

[1] B. E. McCarey, H. F. Edelhauser, M. J. Lynn. “Review of corneal endothelial specular microscopy for FDA clinical trials of refractive procedures, surgical devices and new intraocular drugs and solutions”. In US National Library of Medicine, National institute of health.

[2] T. Bonus, T. Golan. “Object counting by estimating a density map with convolutional neural networks”. Nel sito web ufficiale di Neurosys. URL: https://neurosys.com/article/objects-counting-by-estimating-a-density-map-with-convolutional-neural-networks/

[3] W.F. Ganong. “Fisiologia Medica” 4° Edizione. 2° Edizione italiana a cura di G. Stella. Piccin Editore, Padova (1971).

[4] L. Thomas. “How does the eye work?”. In News medical lifesciences (2019). URL: https://www.news-medical.net/health/How-Does-the-Eye-Work.aspx.

[5] “L’anatomia della cornea”. In Vision Future (2017). URL: https://visionfuture.it/anatomia-cornea/.

[6] Dott. Amedeo Lucente. “Cenni di anatomia e fisiologia Oculare”. Nel sito web del Dottor Amedeo Lucente. URL: https://www.amedeolucente.it/articoli-scientifici-di-oftalmologia.asp

[7] Cornea Donor Study Investigator Group. “Donor age and corneal endothelial cell loss 5 years after successful corneal transplantation: specular microscopy ancillary study results”. In US National Library of Medicine, National institute of health.

[8] L. Ham, C. van Luijk, I. Dapena, TH. Wong, R. Birbal, J. van der Wees, GR. Melles. “Endothelial cell density after descmet membrane endothelial keratoplasty: 1- to 2-year follow-up”. In Americal Joutnal of Ophthalmology, October 2009, Vol 148, Issue 4, Pages 521-527.

[9] M. Satue, M. Idoipe, A. Gavin, M. Romero-Sanz, V. S. Liarakos, A. Mateo, E. Garcia-Martin, A. Blasco-Martinez, A. Sanchez-Perez. “Early changes in visual quality and corneal structure after DMEK: does DMEK approach optical quality of a healthy cornea?”. In US National Library of Medicine, National institute of health.

[10] F. Scarpa, C. Dalla Gassa, A. Ruggeri. Automated Morphometric Analysis of in-vivo Human Corneal Endothelium, *Proc. MICCAI-OMIA 2016,* pp. 89-96, Iowa Research Online, Iowa City, 2016.

[11] I. Jalbert, F. Stapleton, E. Papas, D. F. Sweeny, M. Coroneo. “In vivo confocal microscopy of the human cornea”. In US National Library of Medicine, National institute of health (2003).

[12] E. Vincenzi. “A deep learning approach to the analysis of retinal images”. In Padova Digital University Archive (2017).

[13] S. Shalev-Shwartz, S. Ben-David. “Understanding Machine Learning: from theory to algorithms”. Published by Cambridge University Press in 2014.

[14] R. Prevete. “Dispense per l’insegnamento di Reti Neurali e Machine Learning”. Presso Università degli Studi di Napoli Federico II nel 2010.

[15] https://pytorch.org/, visitato il giorno 18/9/2019.

[16] https://pytorch.org/cppdocs/frontend.html, documentazione di PyTorch, visitato il giorno 18/9/2019.

[17] https://developer.nvidia.com/cuda-zone, visitato il giorno 18/9/2019.

[18] https://docs.microsoft.com/en-us/azure/python/, introduzione all’utilizzo di Python su server Azure, visitato il giorno 18/9/2019.

[19] https://aws.amazon.com/it/pytorch/, introduzione a PyTorch su Amzon web services (AWS), visitato il giorno 18/9/2019.

[20] https://www.dei.unipd.it/bladecluster, documentazione sul Cluster di calcolo ‘Blade’, visitato il giorno 18/9/2019.

[21] https://www.anaconda.com/distribution/, indirizzo dove è stata scaricata l’ultima distribuzione di Anaconda, visitato il giorno 20/9/2019.

[22] https://www.spyder-ide.org/, indirizzo dove è stata scaricata l’ultima versione di Spyder, visitato il giorno 20/9/2019.

[23] https://www.gimp.org/downloads/, indirizzo dove è stata scaricata l’ultima versione di GIMP, visitato il giorno 20/9/2019.

[24] K. Simonyan, A. Zisserman. “Very deep convolutional networks for large scale image recognition”. In arXiv:1409.1556.

[25] https://www.xnview.com/en/xnconvert/#downloads, indirizzo dove è stata scaricata l’ultima versione di XnConvert, visitato il giorno 20/11/2019.

[26] C. Arteta, V. Lempitsky, J. A. Noble, A. Zisserman. “Learning to detect cells using non-overlapping extremal regions. Pubblicato in MICCAI, 2012.

[27] L. Lempitsky, A. Zisserman. “Learning to count objects on images”. In NIPS, 2010.

[28] W. Xie, J. A. Noble, A. Zisserman. “Microscopy cell counting with fully convolutional regression network”. In MICCAI 1st workshop on deep learning in medical image analysis, 2015.

[29] Ronneberger O., Fischer P., Brox T. (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab N., Hornegger J., Wells W., Frangi A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science, vol 9351. Springer, Cham.

[30] D. P. Kingma, J. L. Ba. “Adam: a method for stochastic optimization”. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[31] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P. T. P. Tang. “On large-batch training for deep learning: generalization gap and sharp minima” In ICLR 2017, Toulon (France), 2017.

[32] A. Ruggeri, E. Grisan, J. Jaroszewski. “A new system for the automatic estimation of endothelial cell density in donor corneas”. In Br J Ophthalmol, 2005.

[33] https://filezilla-project.org/download.php?type=client, indirizzo dove è stata scaricata l’ultima versione di Filezilla, visitato il giorno 20/11/2019.