



UNIVERSITÀ DEGLI STUDI DI TRIESTE
Dipartimento di Ingegneria e Architettura

Corso di Studi in Ingegneria Clinica

Progettazione e implementazione di un video-oculografo per la registrazione dei movimenti oculari dei bambini.

Tesi di Laurea Magistrale

Laureando:
Alessandro PELLEGRINO

Relatore:
Chiar.mo Prof. Agostino ACCARDO

...ai miei genitori...

Indice

Introduzione	6
Capitolo 1 – Misurazione della posizione oculare	8
1.1 Magneto-Oculografia (MOG).....	8
1.2 Elettro-Oculografia (EOG)	10
1.3 Misurazione tramite localizzazione dei riflessi ottici	13
1.4 Misurazione tramite segmentazione delle immagini	16
Capitolo 2 – Algoritmi di localizzazione della pupilla	18
2.2 Algoritmi per l’eye tracking	18
2.2.1 Starburst	18
2.2.2 Algoritmo di Kewato e Tetsutani	24
2.3 Algoritmi di riconoscimento della pupilla attraverso le reti neurali	25
Capitolo 3 – Preparazione del dataset	28
3.1 Descrizione del sistema usato per le acquisizioni.....	28
3.1.1 Mentoniera	29
3.1.2. Videocamera e schermo	30
3.1.3 Software: SpinView FLIR.....	31
3.1.4 Arduino	33
3.2 Analisi del dataset e labeling	35
3.3 Acquisizioni e dati	37
3.4 Data augmentation	40
3.4.1 Viewport dell’immagine	41
3.4.2 Shift dei punti	42
3.4.3 Density Function	44
3.4.4 Aumento del Dataset	47
3.4.5 Risultati del data augmentation	53
3.4.6 Creazione delle immagini artificiali e dei labels	57
Capitolo 4 – Localizzazione della pupilla tramite reti neurali	60
4.1 Convolutional Neural Networks	61
4.1.1 Principali tipi di layers in una Convolutional Neural Network	61
4.2 TensorFlow	63
4.3 Implementazione della rete ed architettura	63
4.4 Risultati degli esperimenti	67
Capitolo 5 – Risultati e Conclusioni	78

Bibliografia	83
Ringraziamenti.....	84

Introduzione

L'oculometria, nota anche come eye-tracking, è un processo mediante il quale si riesce a risalire al punto di fissazione oculare ed a misurare il moto della pupilla rispetto alla testa¹.

Grazie a questa tecnologia, sono stati implementati dispositivi e strumenti dai molteplici scopi e campi di applicazione, spaziando dall'assistenza (ad esempio i sistemi che consentono a persone paralizzate di comunicare) e riabilitazione, fino alla stima dell'attenzione e della concentrazione.

Le moderne tecniche di eye-tracking permettono di rilevare i movimenti in modo totalmente non invasivo, sfruttando videocamere che montano filtri per operare esclusivamente nel campo degli infrarossi. In questo modo si rende l'esecuzione dell'esame sicura e senza controindicazioni per il paziente.

Gli algoritmi di eye-tracking possono essere applicati durante la stessa acquisizione video (real-time) oppure essere applicati dopo l'acquisizione. La scelta di una tipologia rispetto all'altra, in genere, andrà ad incidere su parametri quali accuratezza della posizione della pupilla e tempo di elaborazione. Pertanto, sarà la natura del problema da risolvere a far preferire una tipologia di algoritmo rispetto ad un'altra.

Questo lavoro di tesi ruota attorno allo sviluppo di un video-oculografo ad alta risoluzione spaziale capace di diagnosticare patologie quali strabismo, e inoltre di essere un valido supporto alla prevenzione di condizioni come l'ambliopia.

Il sistema di eye-tracking usato per questo lavoro di tesi è costituito da una videocamera sensibile agli infrarossi, un filtro per la rimozione della luce visibile ed un pc. Parte del sistema è anche l'algoritmo che ha il compito di ricostruire il percorso della pupilla nella sequenza di fotogrammi presa in considerazione.

Inoltre, è stata utilizzata una mentoniera per il corretto e stabile posizionamento del soggetto di fronte alla videocamera. È stato infine necessario posizionare degli illuminatori ad infrarossi di fronte al soggetto attivati tramite impulsi TTL che non verranno ulteriormente approfonditi.

Una delle problematiche che sono emerse durante lo sviluppo di tale dispositivo è stata quella di accertarsi che durante l'acquisizione video lo sguardo del neonato non andasse

¹ <https://it.wikipedia.org/wiki/Oculometria>

fuori campo, in quanto renderebbe vano l'esame.

Non potendo fare assegnamento sulla collaborazione del soggetto, si è pensato ad una possibile applicazione di un algoritmo real-time. Il problema emerso è stato che, anche implementando gli algoritmi real-time più efficienti, si perde qualità in termini di risoluzione spaziale.

Infatti, il sistema preso in esame esegue l'acquisizione dei frame a 100 Hz. Questo vuol dire che l'algoritmo da implementare deve essere in grado di riconoscere il centro della pupilla in una finestra temporale di 10ms.

L'obiettivo di questo lavoro di tesi è dunque quello di sperimentare se sia possibile o meno implementare un algoritmo con le caratteristiche sopra descritte, non necessariamente con una risoluzione spaziale ottimale, ma che rientri nei tempi richiesti adottando soluzioni che sfruttano le reti neurali e il deep-learning.

Capitolo 1 – Misurazione della posizione oculare

I movimenti oculari sono fondamentali quando si vuole studiare una patologia, stimare l'attenzione e la concentrazione di un soggetto oppure capire il punto che si sta osservando.

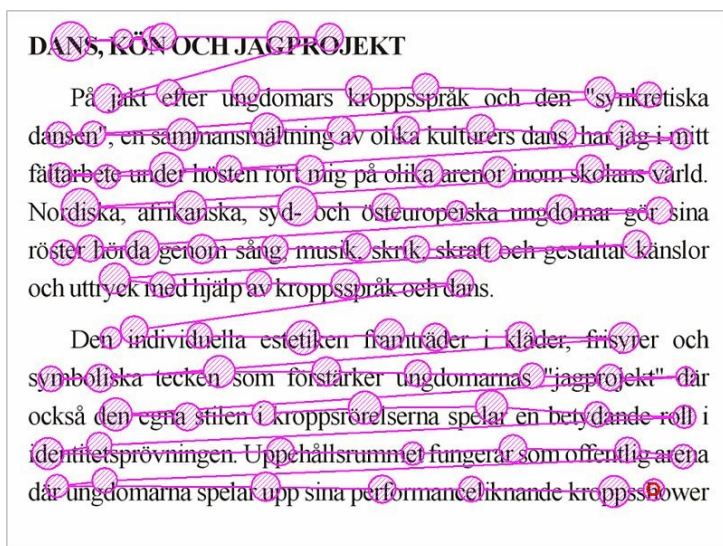


Figura 1 Questo è lo schema tipico del movimento degli occhi durante la lettura. Gli occhi non si muovono mai uniformemente sul testo fisso.

I primi tentativi di studio dei movimenti oculari avevano l'obiettivo di analizzare il processo di lettura, e vennero condotti dal francese Louis Emile Javal.

Successivamente, vennero messi appunto tecniche di eye-tracking sempre più sofisticate che si suddividono in invasive e non invasive.

Tra le tecniche invasive, vi sono quelle che fanno uso di lenti a contatto come la Magneto-Oculografia e quelle che fanno uso di elettrodi (Elettro-Oculografia).

Vi sono infine tecniche non invasive che fanno uso di videocamere che registrano i movimenti oculari che vengono successivamente analizzati tramite algoritmi specifici.

1.1 Magneto-Oculografia (MOG)

I primi esperimenti in materia furono condotti da Edmund Huey che aveva sviluppato un rudimentale eye-tracker, usando una specie di lente a contatto con un foro per la pupilla. La lente era collegata ad un puntatore in alluminio che si muoveva in risposta al movimento dell'occhio [1].

La Magneto Oculografia (MOG) consiste in un paio di lenti a contatto applicate direttamente agli occhi del paziente, che contengono una spira percorsa da corrente, che genera una perturbazione nel campo magnetico circostante, che dipende dall'orientamento del bulbo oculare.

Questa tecnica misura l'induzione elettromagnetica nella una bobina di rame incorporata nella lente a contatto e soggetta ad un campo magnetico.

Il campo magnetico viene generato da una coppia di bobine poste ortogonalmente ai lati del soggetto. Il dispositivo consiste in un anello in silicone modellato per aderire all'occhio. Le bobine sono costituite da un filo di rame e sono incorporate nell'anello di silicone.

La curvatura interna dell'anello è dimensionata per assicurare l'adesione al bulbo oculare mediante forze capillari e di aspirazione. Questo design permette di eliminare gli artefatti di misurazione dovuti a possibili slittamenti.

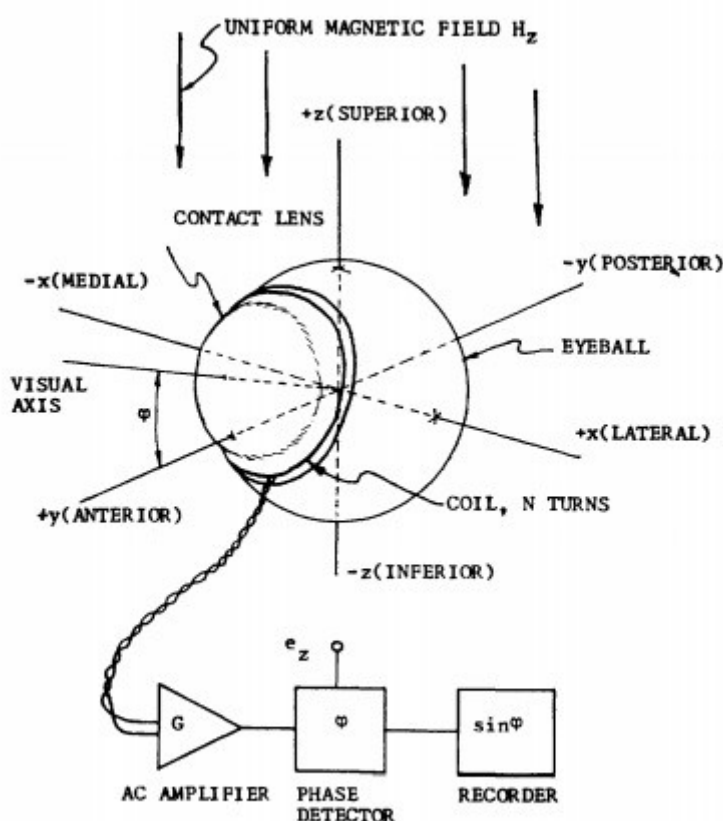


Figura 2 Schema semplificato di una bobina per ottenere la componente verticale del movimento oculare

rilevare il centesimo di grado [2].

Se, come mostrato in figura 2, una bobina con N spire viene avvolta vicino al bordo di una lente a contatto ed è soggetta ad un campo magnetico H_z , una tensione sarà indotta nella bobina in accordo con la legge di Faraday:

La piccola bobina collocata sulla lente può essere accuratamente calibrata prima dell'uso in modo che sia sufficiente un solo punto di calibrazione per determinare il suo orientamento rispetto all'asse visivo dell'occhio del partecipante.

Il metodo ha una risoluzione spaziale molto elevata, che consente di studiare anche i movimenti oculari più piccoli. La sua precisione è maggiore rispetto agli altri metodi e arriva al

$$e = -N \frac{d\Phi}{dt} \cdot 10^{-8}$$

In condizioni di riposo, l'occhio punta esattamente lungo l'asse x e quindi la bobina si trova sul piano x-z e pertanto non viene indotta nessuna tensione. Con una sola bobina sulla lente e l'applicazione di un solo campo magnetico consentirebbero la misurazione con un solo grado di libertà [2].

Il principale svantaggio di questo metodo è la sua natura invasiva: la cornea deve essere anestetizzata e il tempo di registrazione è in genere limitato a circa 30-45 minuti. Questi svantaggi la rendono una tecnica inadatta per molte applicazioni cliniche e per le registrazioni del movimento oculare nei bambini. Viene quindi usata principalmente in ambito di ricerca.

1.2 Elettro-Oculografia (EOG)

L'EOG è un esame molto utile per la diagnosi di diverse patologie, tra cui: distacco della retina, cisti della macula, maculopatie, retinopatie e malattie del sonno.

L'occhio è sede di un potenziale elettrico, e può essere descritto come un dipolo fisso avente polo positivo sulla cornea e polo negativo sulla retina. Per tale motivo, la relativa differenza di potenziale è definita come corneo-retinica. Essa è dell'ordine di 1 mV. L'origine di tale differenza di potenziale è attribuita a processi metabolici che avvengono nella retina.

Basandosi su questo fenomeno, la tecnica è pressoché indipendente da stimolazioni luminose (può essere rilevato con l'occhio in totale oscurità o chiuso).

Questa differenza di potenziale e la rotazione dell'occhio permettono la misura, mediante una coppia di elettrodi periorbitali superficiali, di un segnale noto come potenziale elettrooculografico (EOG).

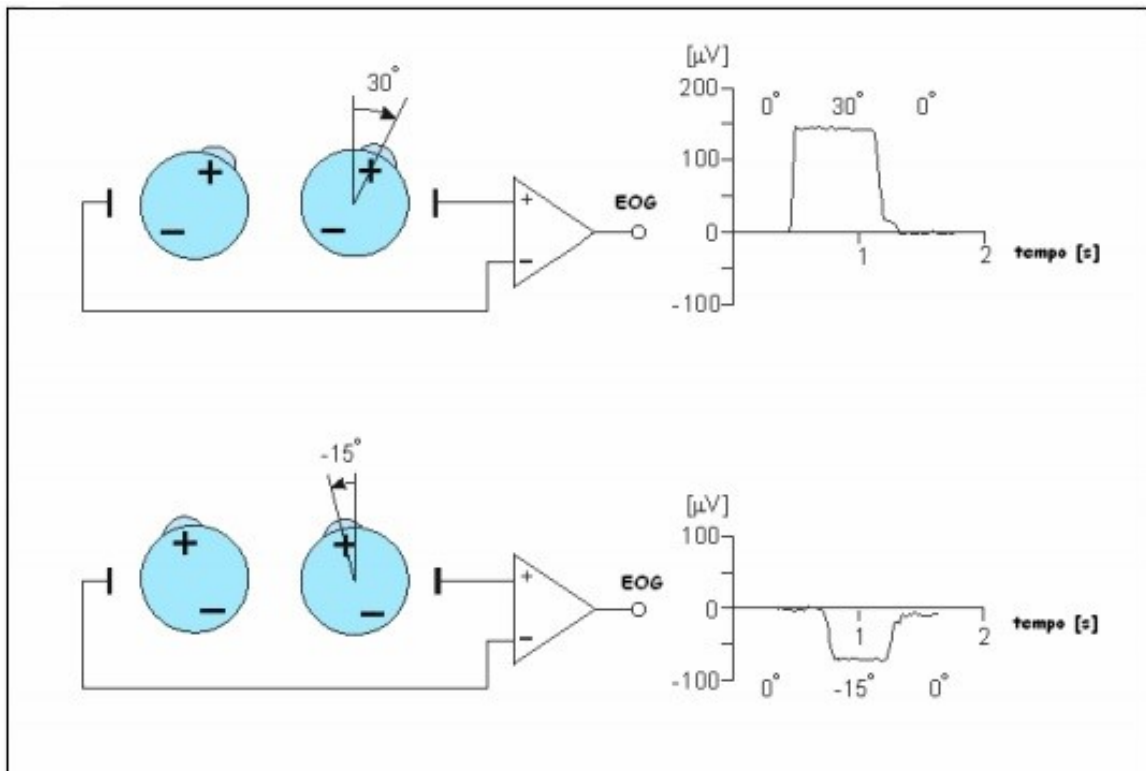


Figura 3 Schematizzazione della genesi del potenziale elettrooculografico indotto da rotazioni orizzontali oculari

Con l'occhio a riposo gli elettrodi sono allo stesso potenziale e non si registra nessun segnale (come nel caso della MOG).

Una rotazione dell'occhio causa invece una differenza di potenziale; in particolare, l'elettrodo che si affaccia sul lato verso il quale avviene la rotazione diventa positivo relativamente al secondo elettrodo.

Essenziale è la fase di calibratura del segnale, che può essere realizzata facendo guardare al paziente due punti che individuano un angolo noto e registrando l'EOG.

Sono tre le diverse fonti di rumore che si possono incontrare [1]:

- Rumore induttivo: questo è dovuto alle interferenze introdotte dai campi elettromagnetici dell'ambiente. Può essere ridotto correlando le tensioni misurate all'elettrodo di riferimento;
- Rumore termico: questo è generato dalla resistenza di ingresso dell'amplificatore e dalla resistenza di contatto degli elettrodi cutanei. Per ridurre la resistenza di contatto, la pelle deve essere opportunamente pulita, inoltre gli elettrodi dovrebbero essere

fatti di materiale relativamente non polarizzabile come argento-cloruro d'argento o oro e dovrebbero essere applicati con una pasta conduttiva;

- Rumore capacitivo: questo è dovuto all'attività elettrica di muscoli e neuroni. I soggetti devono cercare di evitare qualsiasi movimento eccetto quelli oculari. In particolare, viso e muscoli mandibolari dovrebbero rimanere rilassati.

L'elettro-oculografia presenta sia vantaggi che svantaggi rispetto ad altri metodi per la determinazione del movimento dell'occhio. Gli svantaggi più importanti sono relativi al fatto che il potenziale corneo-retinico può essere influenzato da svariati fattori, come ad esempio luce e affaticamento. Di conseguenza, è necessario calibrare frequentemente il sistema di registrazione. I vantaggi di questa tecnica sono la possibilità di eseguire l'indagine in oscurità totale e/o con gli occhi chiusi.

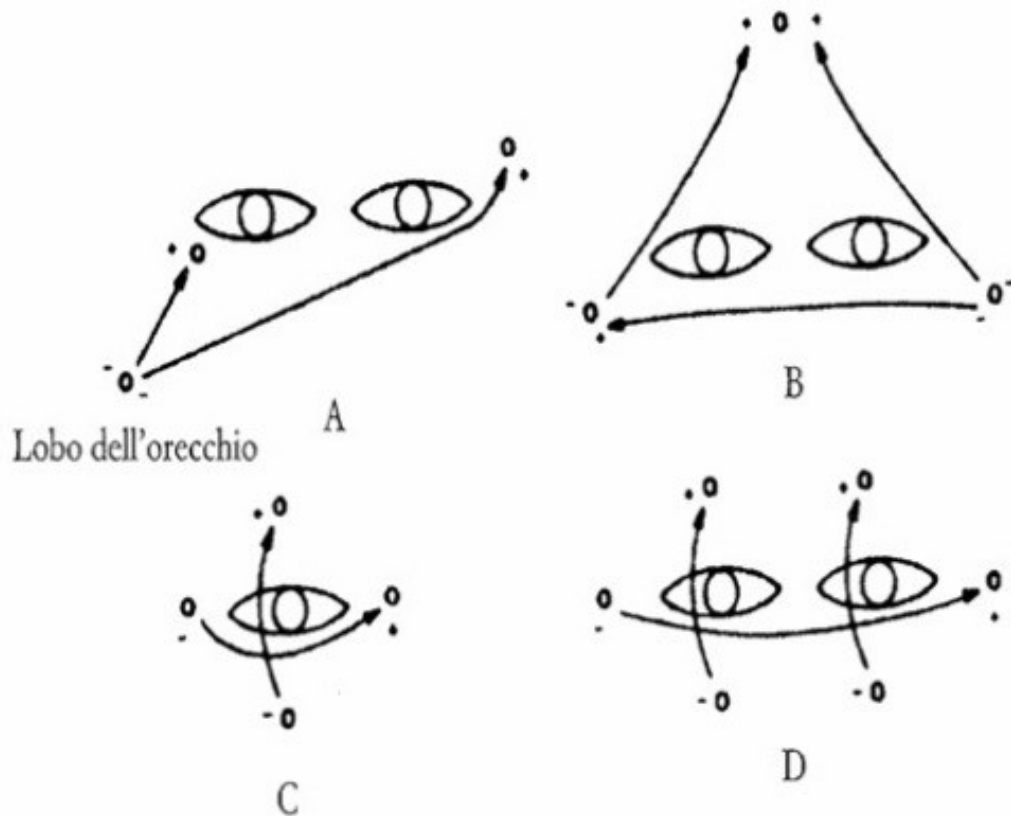


Figura 4 Montaggio degli elettrodi per la registrazione dei movimenti oculari. Sono riportate quattro diverse configurazioni.

1.3 Misurazione tramite localizzazione dei riflessi ottici

Nei precedenti paragrafi si sono descritte sommariamente le tecniche invasive per rilevare i movimenti oculari, evidenziandone i principali pregi e difetti, limiti e potenzialità.

Esistono anche tecniche innovative che non sono per niente invasive e non causano alcun disturbo al soggetto che si presta all'esame, tra cui vi è la video-oculografia.

Le tecniche di video-oculografia possono essere suddivise a seconda che si operi nel visibile, sfruttando la luce naturale o un led per illuminare l'occhio, o nello spettro infrarosso, ovvero utilizzando sorgenti di luce infrarosse.

Utilizzando una sorgente di luce naturale, è possibile andare a studiare il contorno fra iride e sclera e studiando il riflesso corneale si può andare a ricavare la direzione dello sguardo. Lo studio della pupilla, in questo caso, richiede un'attenta operazione di *pre-processing* dell'immagine che permetta di aumentare il contrasto tra la pupilla stessa e l'iride.

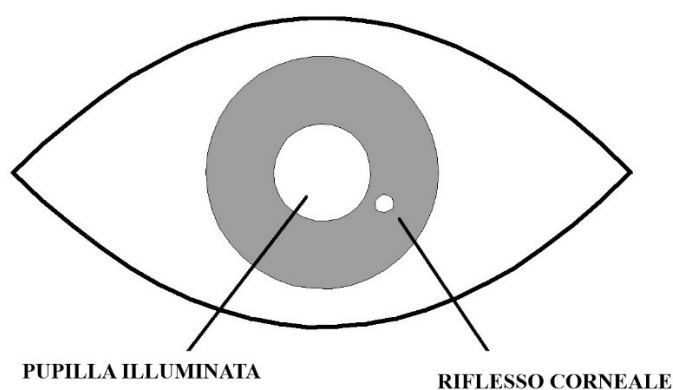


Figura 5 Riflesso corneale generato dall'illuminazione della pupilla

Le tecniche di eye tracking basate su oculografia all'infrarosso sfruttano la riflessione di una radiazione luminosa infrarossa (IR) di piccola potenza inviata sull'occhio. Mediante sensori ottici sensibili nello spettro infrarosso sono misurati i cambiamenti nella quantità di luce riflessa dall'occhio, da cui è possibile risalire ai movimenti oculari. La sorgente IR permette di utilizzare algoritmi di rilevamento della pupilla con risultati più soddisfacenti. Infatti, i raggi infrarossi permettono di avere un'immagine con l'iride che appare più chiara, indipendentemente dal colore degli occhi del soggetto e con maggiore contrasto rispetto alla pupilla.

Il vantaggio di utilizzare la luce all'infrarosso in alternativa alla luce naturale è la comodità del paziente, giacché essa è invisibile, e la qualità dell'immagine dell'occhio, che non è influenzata da interferenze nello spettro della luce visibile. Il principale svantaggio sono i danni che la luce infrarossa può fare all'occhio, ma la penetrazione della radiazione è minima e gli effetti della luce infrarossa dipendono da diversi fattori tra cui la distanza tra occhio e sorgente luminosa, che cambia da dispositivo a dispositivo. Un altro svantaggio da non sottovalutare nella video-oculografia agli infrarossi è il fatto che non si può svolgere quest'esame all'aperto, in quanto sono presenti diversi disturbi infrarossi nell'ambiente [6].

Ci sono diversi metodi utilizzati per determinare la posizione dello sguardo [3]:

- Video-Oculografia con riflesso corneale: Questa tecnica utilizza la posizione della pupilla e del riflesso corneale per determinare la direzione dello sguardo. Il sistema è costituito da un illuminatore e da una fotocamera sensibile all'infrarosso, che devono avere una posizione fissa per la durata del test. Il riflesso corneale è dato dall'illuminazione del bulbo oculare tramite una luce fissa, la cui posizione non cambia nonostante la rotazione dell'occhio.

Per questa ragione, viene utilizzato come un punto di riferimento per determinare la nuova posizione della pupilla e quindi la direzione dello sguardo. Quando lo sguardo è diretto verso la sorgente luminosa, il riflesso corneale sarà molto vicino al centro della pupilla; man mano che lo sguardo si sposta, la distanza tra queste due caratteristiche aumenta.

- Video-Oculografia utilizzando solo la pupilla: Questa tecnica utilizza la segmentazione della pupilla. In questo caso, è preferibile utilizzare camere sensibili all'infrarosso che permettono di avere immagini della pupilla a maggior contrasto. Per rendere il dispositivo indipendente dai movimenti della testa, si rende il sistema indossabile, montando la telecamera su occhiali oppure su un caschetto.

Gli eye tracker video si basano sulla posizione del centro della pupilla. Tuttavia, i sistemi che si basano sulle videocamere soffrono di un problema comune e non banale: il centro della pupilla può spostarsi quando cambia la dimensione della pupilla.

Questo artefatto, si verifica per motivi fisiologici anche quando l'occhio non ruota, il centro della pupilla può spostarsi in direzione orizzontale e/o verticale in funzione della dimensione

della pupilla [4].

Questo artefatto dipende dalle condizioni e dall'ambiente in cui è svolto l'esperimento. In alcune condizioni, piccole variazioni casuali della dimensione della pupilla e corrispondenti artefatti casuali possono raggiungere una media a zero nel tempo.

Uno dei motivi principali per cui vi sono dei cambiamenti nella dimensione della pupilla è il cambiamento di luminanza all'interno del campo visivo, e quindi ne conseguono elevate ampiezze di dilatazione oppure costrizione della pupilla, a latenze che sono molto rilevanti nella ricerca sul movimento oculare (circa 200-300ms) [5].



Figura 6 Sistema indossabile VOG

In genere, lo spostamento tipico del centro della pupilla variava notevolmente tra i soggetti (da 0,3 a 5,2 gradi di deviazione standard, media 2,6 gradi), ma anche tra gli occhi dei singoli soggetti (differenza da 0,1 a 3,0 gradi, differenza media 1,0 gradi). Tuttavia, l'artefatto della pupilla provocherà errori sistematici, se la registrazione include condizioni di diverse dimensioni della pupilla o se la dimensione della stessa differisce in modo critico tra la fase di calibrazione e quella di test.

Nonostante ciò, anche quando la luminanza della stanza è costantemente controllata grazie a sofisticate tecnologie, può comunque verificarsi la costrizione e dilatazione della pupilla in quanto dipende da diversi fattori. Questi fattori, in genere, sono legati alla concentrazione dell'individuo, al suo intento di rilevare il bersaglio, a fattori emotivi. Anche il contenuto del video-test mostrato al soggetto può far variare la dimensione della pupilla [5].

In definitiva, dal punto di vista del sistema di eye-tracking, questi fattori non sono rilevanti perché non possono essere in alcun modo controllati o evitati.

Ci sono tuttavia diversi studi che si avvalgono di metodi efficaci per risolvere questo tipo di artefatti, come la compensazione a 2-punti o a 3-punti.

1.4 Misurazione tramite segmentazione delle immagini

La segmentazione è il processo atto a isolare nell'immagine gli oggetti di interesse. Questo processo ha sia l'obiettivo di estrarre informazioni quantitative e qualitative, sia quello di riconoscimento, classificazione o interpretazione. Gli algoritmi che vengono utilizzati per la segmentazione delle immagini sono svariati e possono essere automatici o semi-automatici. I metodi più conosciuti sono:

- *Thresholding (sogliatura)*: viene scelto un certo valore, detto di soglia, del range di che può assumere un pixel. Se questo valore è inferiore al valore di soglia, questo viene portato a 0; altrimenti, il valore del pixel viene portato a saturazione. Il risultato sarà dunque un'immagine i cui pixel assumeranno solo due valori: 0 o 1, rendendo così massimo il contrasto tra le diverse regioni che sono andate in contro alla sogliatura. Questo è un metodo automatico.
- *Region Growing*: si inizia con un pixel (o gruppo di pixel) che appartiene alla struttura di interesse, chiamato "seme" e si passa poi ad esaminare uno alla volta i pixel nel suo intorno (gli 8 connessi) che sono inclusi o esclusi dalla regione in base ad un criterio di omogeneità. Se è un problema 3D, allora l'intorno è composto dai pixel 26-connessi, in questo modo la complessità del problema cresce esponenzialmente. La ricerca continua fino a che nessun ulteriore pixel può essere aggiunto alla regione. Questo metodo è semi-automatico.

Per ottenere un buon livello di segmentazione, è necessario avere delle immagini di ottima qualità prima che esse vengano date all'algoritmo di segmentazione. In particolare, è necessario che vi sia un ottimo contrasto tra le diverse regioni.

Per tali motivi, questa tecnica si presta ad essere efficace quando la qualità dell'attrezzatura è elevata e vengono usati gli illuminatori agli infrarossi. Inoltre, per rendere maggiormente autonomo ed efficace l'algoritmo, viene usata la trasformata circolare di Hough oppure reti neurali allenate.

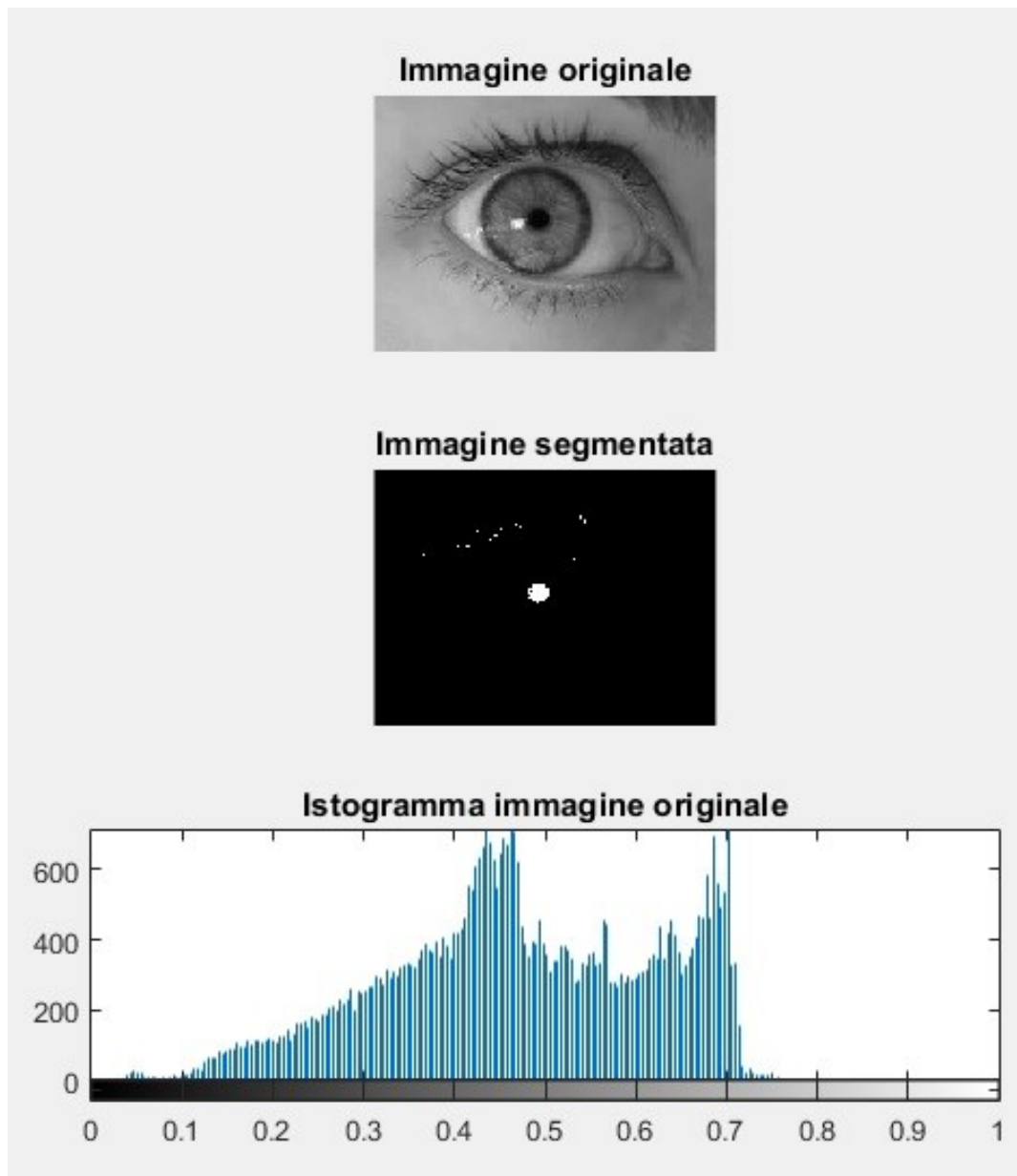


Figura 7 Questa figura rappresenta un'immagine prima e dopo il thresholding. Quando il thresholding risulta essere poco efficace, fare ricorso all'istogramma per capire la distribuzione dei diversi livelli di grigio risulta essere fondamentale.

Capitolo 2 – Algoritmi di localizzazione della pupilla

Vi sono numerosi algoritmi per rilevare la posizione oculare e l'eye-tracking. Si possono classificare in due principali categorie: quelli che funzionano in tempo reale e quelli che vengono fatti girare offline.

In genere, gli algoritmi che rilevano in tempo reale il centro della pupilla e ne tracciano i movimenti sacrificano precisione e accuratezza per avere un costo di elaborazione nettamente inferiore rispetto agli algoritmi che lavorano offline.

Pertanto, sarà proprio in funzione del problema che si vuole risolvere che si sceglierà un algoritmo più preciso ma molto più complesso piuttosto che un algoritmo più leggero ma meno accurato.

2.2 Algoritmi per l'eye tracking

Vi sono numerosi algoritmi per l'eye-tracking in tempo reale. Molti di questi, per ridurre il tempo di elaborazione, si occupano di ricercare la posizione dell'occhio piuttosto che la posizione della pupilla. Uno dei problemi fondamentali che deve essere risolto in questo tipo di algoritmi è la compensazione dei movimenti della testa, che, altrimenti, causerebbero errori grossolani.

2.2.1 Starburst

Descrizione generale e passi principali dell'algoritmo

Starburst è un algoritmo per usato tracciare i movimenti oculari, che combina gli approcci *feature-based* e *model-based*.

Starburst analizza un fotogramma per volta e ricerca la posizione della pupilla e il riflesso corneale calcolando il vettore differenza tra la posizione della pupilla e la posizione del riflesso per risalire al punto osservato. Successivamente, l'algoritmo inizia individuando e rimuovendo il riflesso corneale dall'immagine.

A questo punto, vengono localizzati i punti che formano il contorno della pupilla usando una tecnica iterativa basata su tecniche *feature-based*. In questa fase, viene interpolata un'ellisse

su un sottoinsieme del contorno della pupilla rilevato, usando il paradigma *Random Sample Consensus* (RANSAC) [6].

I parametri relativi al miglior *fitting* che si ottengono in questa fase, vengono utilizzati per la seconda fase che è quella *model-based*, dove si cerca un'ellisse che massimizzi il *fit* con il contorno della pupilla [6].

L'algoritmo non utilizza informazioni sui colori dell'immagine, perciò le immagini vengono catturate in scala di grigi, in questo modo l'elaborazione risulta più veloce, perché si ignorano completamente due delle tre componenti cromatiche dell'immagine.

Riduzione del rumore

Questa fase in realtà non è obbligatoria, ma è necessaria nel caso in cui si usano videocamere che producono immagini dalla qualità non ottimale. Le tipologie di rumore tipicamente incontrate sono due: *shot noise* (rumore granulare) e *line noise*.

Nel caso del rumore shot, la soluzione è applicare un filtro Gaussiano, ad esempio di dimensioni 5x5 con deviazione standard di 2 pixels.

Nel caso del *line noise*, la questione è più complessa poiché la sua distribuzione sull'immagine non è omogenea. La soluzione è quella di calcolare un fattore C e normalizzare ogni riga dell'immagine rispetto a questo fattore, in modo da spostare l'intensità media di tale riga verso la media ottenuta dai frame precedenti [6]. Questo fattore C, è così calcolato:

$$C(i, l) = \beta I(i, l) + (1 - \beta)C(i - 1, l)$$

Dove $I(i, l)$ è la media della riga dei frame analizzati e $\beta = 0.2$ [6].

Riconoscimento del riflesso corneale, localizzazione e rimozione

Il riflesso corneale corrisponde ad una delle regioni più luminose dell'immagine. Per questo motivo, è semplice individuarla utilizzando una sogliatura (*thresholding*). Tuttavia, applicare sempre lo stesso valore per la sogliatura è sicuramente una via non ottimale. Per questo

motivo, la soluzione migliore è quella di usare una sogliatura adattiva (*adaptive thresholding*).

Inizialmente, la soglia scelta è quella con il massimo valore possibile per produrre un'immagine binaria in cui vengono identificati come appartenenti al riflesso corneale solo i pixel con un'intensità maggiore alla soglia.

Successivamente, la soglia viene ridotta e viene calcolato il rapporto l'area della regione candidata più grande (che presumibilmente rappresenta il riflesso corneale) e l'area media delle altre regioni: inizialmente il rapporto crescerà perché l'area del riflesso corneale tenderà a crescere (questo succede perché vengono selezionati più pixel) più velocemente delle altre aree [6].

Una diminuzione del rapporto indica che l'area dei falsi candidati inizia a crescere, perciò verrà utilizzato il valore di soglia che genera il rapporto ottimale. Il centro del riflesso corneale viene infine calcolato come il centro geometrico dell'area di estensione maggiore.

Una volta localizzata la regione di pixels che racchiude il riflesso corneale, si procede alla sua rimozione. Per fare ciò viene effettuata una interpolazione radiale: ogni pixel appartenente alla regione, viene sostituito con la media dei pixels che formano il contorno della regione.

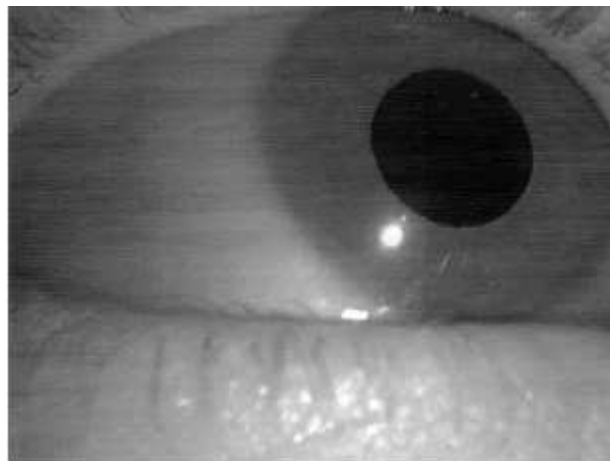


Figura 8 Immagine prima che venga rimosso il riflesso corneale

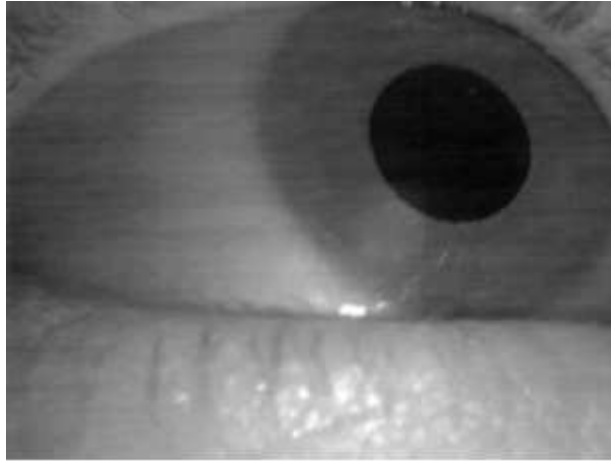


Figura 9 Immagine dopo aver rimosso il riflesso corneale

Localizzazione del contorno della pupilla

Per la localizzazione della pupilla è stata sviluppata una tecnica basata sulla localizzazione di particolari caratteristiche (dette *features*) che permette di trovare il contorno della pupilla. Questa tecnica è molto diversa dall'approccio classico per la ricerca della pupilla. Normalmente, viene utilizzata una tecnica chiamata *edge detection* che viene applicata sull'intera immagine o ad una zona centrata sulla pupilla: questo approccio porta quasi sicuramente ad un elevato utilizzo delle capacità computazionali, in quanto la pupilla occupa, di solito, un'area molto ridotta dell'immagine e non è detto che servano tutti i punti del contorno per poterlo stimare accuratamente [6].

Questo approccio è diviso in due fasi:

- nella prima si rilevano i bordi (ovvero le *feature* cercate) seguendo un limitato numero di raggi che si estendono da un punto centrale stimato come centro della pupilla (fig. 10 a).
- Nella seconda fase si aumenta l'accuratezza della stima cercando nuovamente *feature* seguendo i raggi che partono dai primi punti trovati e sono diretti verso il centro stimato (fig. 10 b e 10 c).

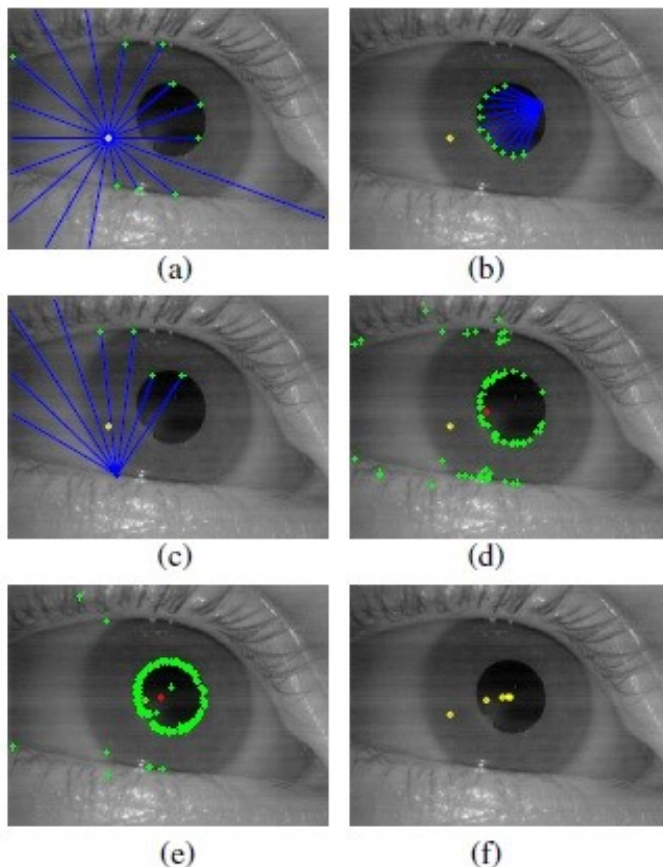
La stima iniziale del centro della pupilla è il centro dell'immagine per il primo fotogramma,

mentre è il centro precedentemente calcolato per i fotogrammi successivi al primo.

Successivamente vengono calcolate le derivate lungo gli N raggi che si estendono dal punto centrale, finché non viene superata una soglia: poiché viene utilizzata la tecnica *dark-pupil* vengono considerate solamente le derivate positive, ovvero la cui intensità cresce spostandosi lungo il raggio [6].

Per ogni *feature* candidata viene ripetuto il processo di *feature-detection* appena descritto: ma questa volta si utilizza il raggio di ritorno e la ricerca viene limitata su un angolo di 50 gradi per diminuire la complessità computazionale e per evitare di aggiungere *feature* che si trovino dalla parte esterna alla pupilla.

Nello step che segue, i punti trovati vengono interpolati per ricavare un'ellisse: i piccoli



errori compiuti nella rilevazione dei punti possono portare ad un grossolano errore nell'interpolazione, per questo motivo il processo a due stadi descritto precedentemente viene ripetuto. Per ogni iterazione viene utilizzato come punto di partenza il baricentro di tutti i punti rilevati nel passo precedente e il processo viene ripetuto finché la posizione del centro rimane pressoché costante [6].

Figura 10: steps

dell'algoritmo starburst

Il risultato della seconda iterazione dell'algoritmo è mostrato in figura 10 e; osservando la posizione dei punti di partenza ad ogni iterazione, si può notare come l'algoritmo converga velocemente alla posizione finale (10 f).

Fitting dell'ellisse

Durante la localizzazione della pupilla, come si è visto nel paragrafo precedente, i punti caratteristici vengono interpolati per ricavare un'ellisse. Per effettuare questa interpolazione si potrebbe utilizzare un metodo ai minimi quadrati, ma un errore presente nello stadio di localizzazione delle feature influenzerebbe pesantemente l'accuratezza dell'interpolazione [6].

Per ottenere un'interpolazione più precisa viene utilizzato il metodo denominato Random Sample Consensus (RANSAC): questo metodo viene spesso utilizzato in Computer Vision per adattare un modello ad una serie di dati ricavati sperimentalmente [6]; questi dati sono rappresentati come punti, e di questi punti alcuni sono corretti, mentre vi è una percentuale sconosciuta di punti errati.

Nel caso in esame i punti corretti sono quelli appartenenti al contorno della pupilla, quelli errati sono quelli esterni al bordo della pupilla.

Il metodo dei minimi quadrati utilizzerebbe, invece, tutti i punti disponibili, in quanto ipotizza che tutti i campioni siano corretti e siano soggetti solo ad un errore di misura.

RANSAC, d'altro canto, ammette la possibilità che esistano punti errati, e per questo motivo utilizza un sottoinsieme dei dati disponibili per adattare il modello.

In pratica, RANSAC è una procedura iterativa che seleziona a caso molti piccoli sottoinsiemi dei dati di partenza, utilizza questi sottoinsiemi per deformare il modello e, tra questi modelli, trova il modello che si adatta meglio all'intero insieme dei dati.

Mediamente, il 17% dei punti è errato [6]; perciò è stato introdotto un ulteriore passaggio dell'algoritmo che cerca migliorare la corrispondenza tra l'ellisse interpolata e l'immagine, in modo da ridurre l'errore.

Per migliorare la corrispondenza dell'ellisse interpolato con l'immagine analizzata viene utilizzata una ricerca basata su *local-model*: in questo caso non viene più utilizzata una ricerca basata su feature, ma viene utilizzata una ricerca non lineare chiamata Nelder-Mead [6].

In pratica si cercano i parametri dell'ellisse che minimizzano:

$$\frac{\int I(a + \delta, b + \delta, \alpha, x, y, \theta) d\theta}{\int I(a - \delta, b - \delta, \alpha, x, y, \theta) d\theta}$$

Dove $\delta = 1$, e $I(a, b, \alpha, x, y, \theta)$ è l'intensità del pixel all'angolo θ sul contorno dell'ellisse definito dai parametri a, b, x, y e α , dove a e b sono rispettivamente l'asse maggiore e l'asse minore dell'ellisse, il punto (x, y) è il centro e infine α è l'orientamento.

Infine, l'algoritmo starburst prevede un ulteriore passo che ha l'obiettivo di risalire al punto osservato dal soggetto. Ai fini della ricerca trattata in questo lavoro di tesi, questo passo non è stato implementato in quanto non utilizzato.

2.2.2 Algoritmo di Kawato e Tetsutani

Kawato e Tetsutani [7] propongono di estrarre la posizione degli occhi e tracciarla nei fotogrammi successivi utilizzando opportuni template.

Il primo passo dell'algoritmo consiste nella la ricerca degli occhi: per rilevare la posizione oculare vengono analizzate le differenze tra fotogrammi successivi, se la testa è immobile le uniche differenze saranno quelle causate battito delle palpebre; se invece la testa non è immobile, nasce il problema di distinguere le differenze causata dal movimento delle palpebre da quelle causate dal movimento della testa.

Per ovviare a questo problema, Kawato e Tetsutani hanno effettuato alcune considerazioni: ad esempio, nel caso di movimento rotatorio della testa le aree con le differenze maggiori saranno sicuramente vicine ai bordi del volto (se la rotazione è verso sinistra l'orecchio sinistro scomparirà), quindi le aree che presentano grandi differenze e sono vicine ai bordi del viso verranno eliminate.

Dopo aver individuato gli occhi, l'algoritmo si occupa di tracciarli: una semplice ricerca per aree corrispondenti non sarebbe efficace, visto che l'orientamento potrebbe essere diverso a causa dei movimenti della testa.

Per questo motivo è stata sviluppata una tecnica che utilizza un template fisso che viene inizializzato ad ogni occlusione, e un template variabile che viene aggiornato sulla base della

posizione degli occhi nel frame precedente: in questo modo si ottiene un metodo di tracciamento molto efficace.

2.3 Algoritmi di riconoscimento della pupilla attraverso le reti neurali

Ci sono inoltre algoritmi che sfruttano le reti neurali competitive per tracciare i movimenti oculari [8]. Le CNN sono quelle reti il cui apprendimento è di tipo “non supervisionato” e quindi i neuroni competono tra di loro, che vuol dire il neurone che riceve il maggior punteggio è quello che definisce l’output.

Con l'apprendimento competitivo i neuroni di uscita di una rete neurale competono tra di loro per divenire attivi. Ci sono 3 elementi base per un metodo di apprendimento competitivo:

- 1 - un insieme di neuroni uguali a meno di pesi sinaptici generati a caso che rispondono in maniera differente ad un dato insieme di inputs;
- 2 - un limite alla “forza” di ciascun neurone;
- 3 - un meccanismo che permette ai neuroni di competere per il diritto a rispondere ad un dato sottoinsieme di inputs cosicché un solo neurone o uno solo per gruppo è attivo in un certo momento. Il neurone che vince è chiamato winner-takes-all. In questo modo i neuroni tendono a specializzarsi su un insieme di input simili divenendo riconoscitori di caratteristiche per differenti classi di inputs.

Nel lavoro svolto da Marcone et al. [8] è stata realizzata una rete composta da 9 neuroni che competono tra loro.

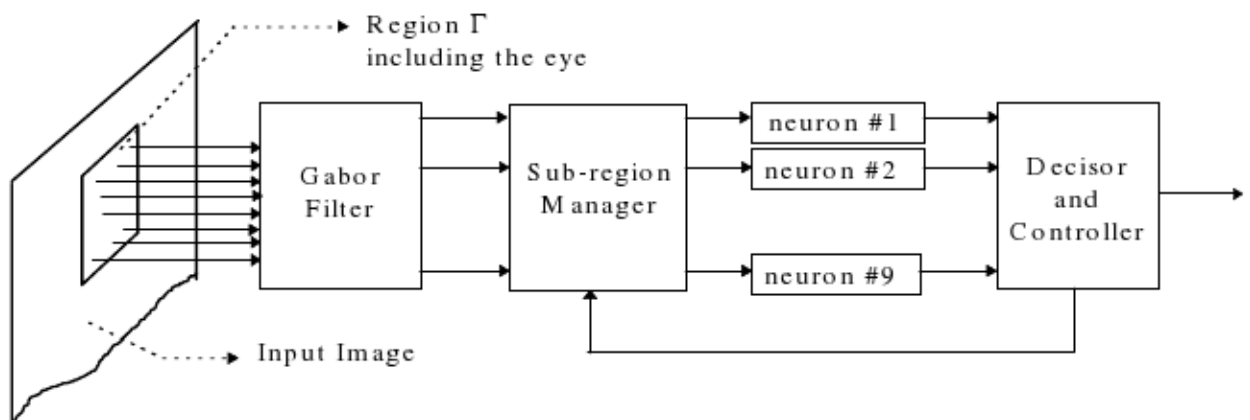


Figura 11 Schema della rete neurale competitiva proposta da Marcone et al.

I neuroni sono associati a 9 sotto regioni della regione Γ contenente l'occhio, decisa all'inizio del processo.

Nella prima fase la posizione dei centri delle 9 sotto-regioni coincide rispettivamente con il pixel centrale di Γ e gli 8 pixel vicini. Il neurone associato alla sotto-regione caratterizzata dal massimo della risposta media vince la competizione. Se il vincitore è proprio la sotto-regione centrale, questo passaggio si conclude.

Altrimenti, il controller della rete sposta la sotto-regione vincitrice nella posizione centrale. Il meccanismo competitivo è di nuovamente applicato e viene determinato il nuovo neurone vincente. Il processo continua fino a che non sono necessari ulteriori spostamenti. In questo caso il passaggio corrente è concluso.

I passaggi successivi vengono eseguiti in modo simile, ma con una dimensione ridotta delle sotto-regioni. Alla fine dell'ultimo passaggio, il pixel centrale della regione associato con il neurone vincente è considerato coincidente con la pupilla. Nella figura 3 è raffigurato il funzionamento della rete neurale nel localizzare la pupilla rispetto ad una singola immagine:

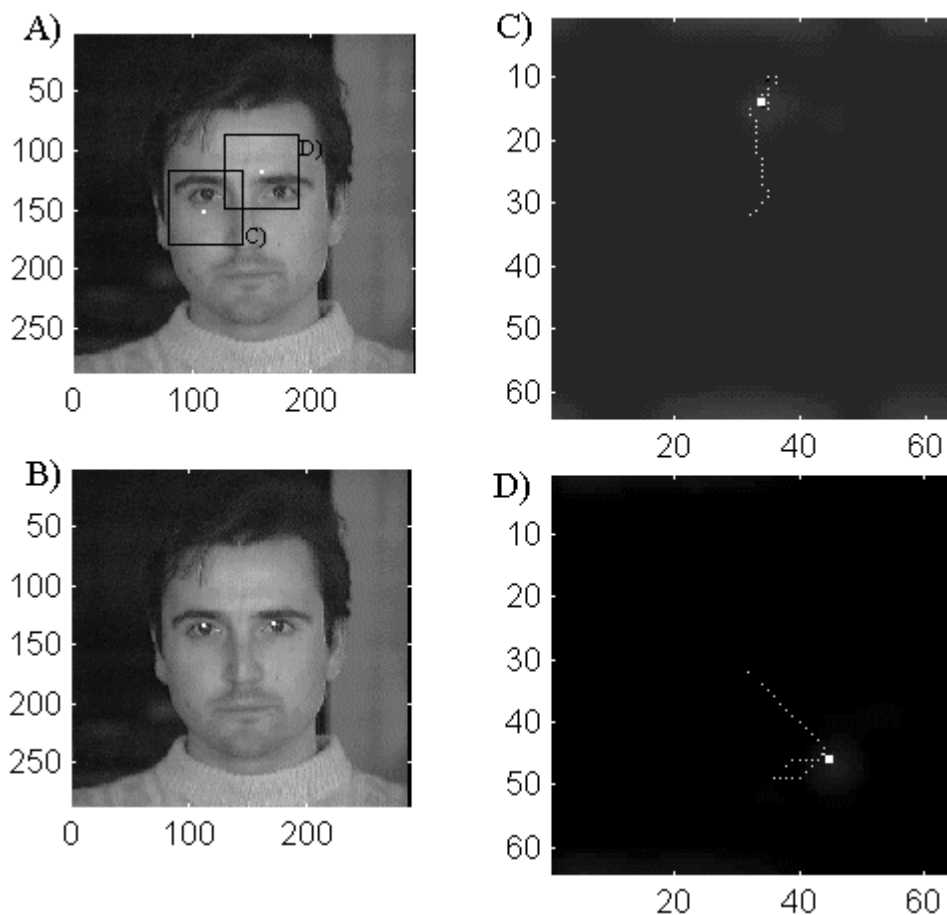


Figura 12 Immagine che schematizza il funzionamento della rete neurale

La fase di addestramento è, come vedremo anche più avanti, dal punto di vista computazionale molto complessa, ma secondo le analisi effettuate da Pedersen e Spivey [9] automatizzando la creazione del training set e avendo computer con una potenza di calcolo idonea questa tecnica potrebbe essere utilizzata anche in tempo reale.

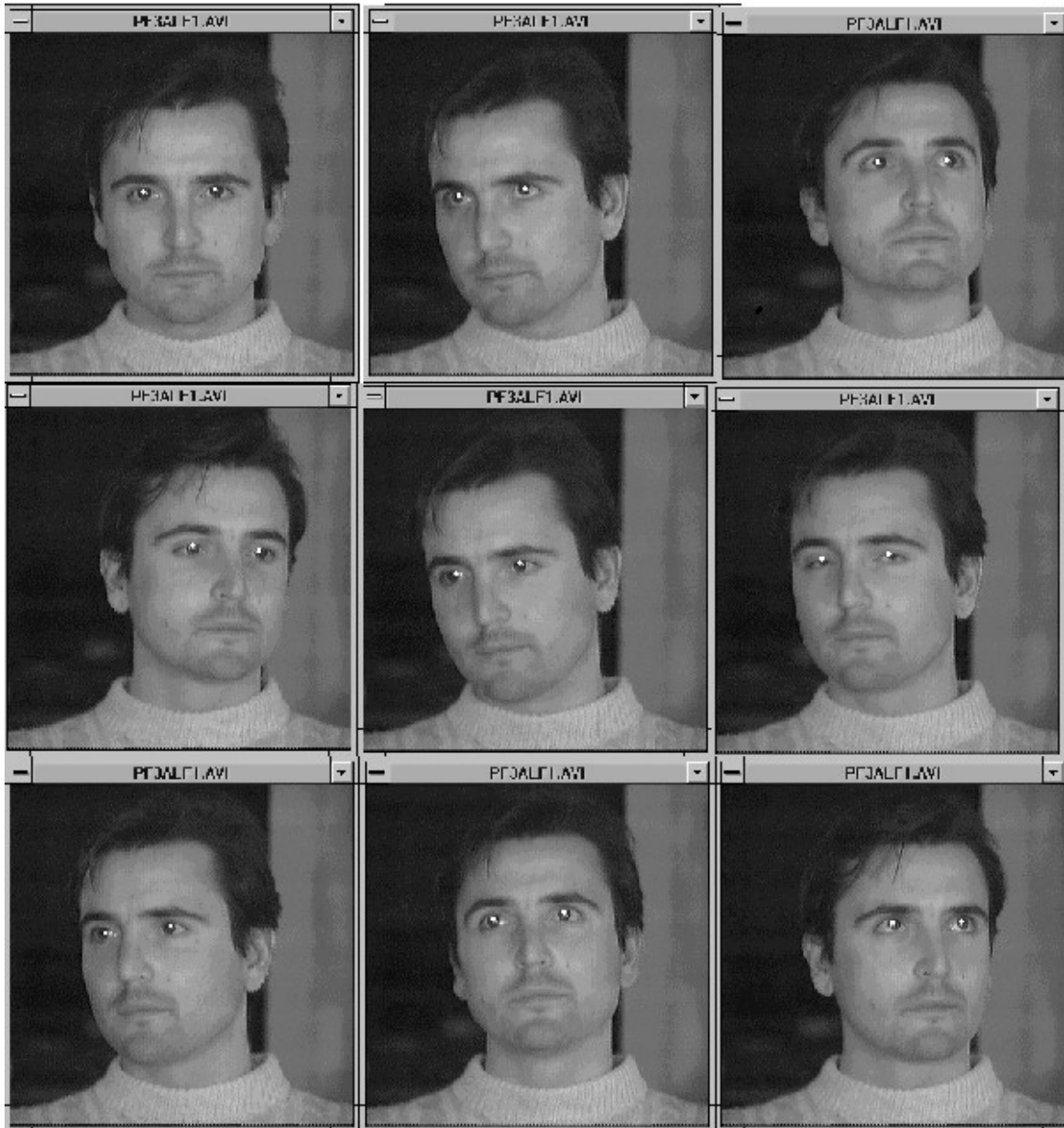


Figura 13 immagine che raffigura il tracciamento della pupilla attraverso una CNN

Capitolo 3 – Preparazione del dataset

3.1 Descrizione del sistema usato per le acquisizioni

Con questo progetto si vuole realizzare un sistema che consenta di registrare i movimenti oculari, in modo da poter applicare successivamente un algoritmo che trovi il centro della pupilla. L'obiettivo principale è quello di avere un consistente dataset di immagini correttamente etichettate con la posizione del centro della pupilla, che consenta di addestrare in modo efficiente una rete neurale.

È importante, all'interno di un dataset di immagini riguardante una singola sessione, che ci sia diversità tra le posizioni del centro della pupilla tra un'immagine e l'altra, in modo che la rete neurale venga addestrata con campioni diversi tra loro.

Per ovviare a questo problema, lo stimolo presentato è stato semplicemente un puntino che il soggetto doveva fissare, e ad una distanza temporale di circa un secondo riappariva in una posizione diversa (non troppo distante, il range di posizioni possibili è comunque ristretto).

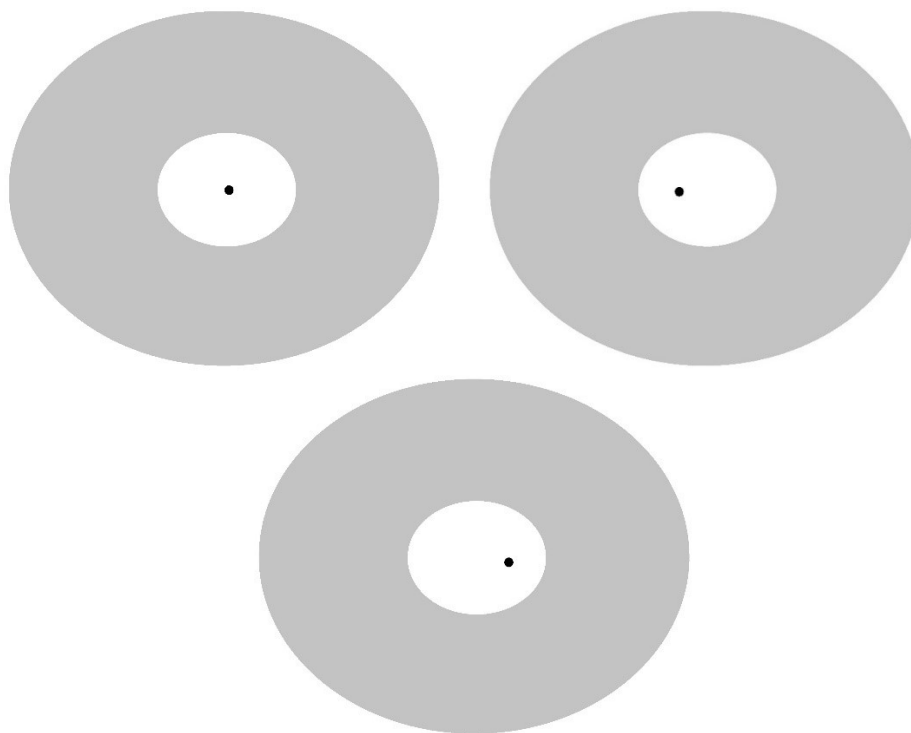


Figura 14: tre dei possibili stimoli somministrati ai soggetti durante le acquisizioni. È importante far variare di poco la posizione del puntino nero tra un frame ed un altro, in modo da indurre un lieve spostamento della pupilla

3.1.1 Mentoniera

Durante l'esecuzione del test i soggetti dovranno seguire attentamente con lo sguardo lo stimolo che comparirà sullo schermo. I soggetti dovranno evitare qualsiasi movimento della testa e, per assicurare una posizione fissa per tutta la durata della registrazione, si farà uso di un supporto.

Questo agevolerà il mantenimento della posizione atto ad evitare spostamenti troppo bruschi del soggetto. Il supporto è a sua volta fissato saldamente ad un tavolo su cui sono posizionati anche lo schermo e la telecamera.

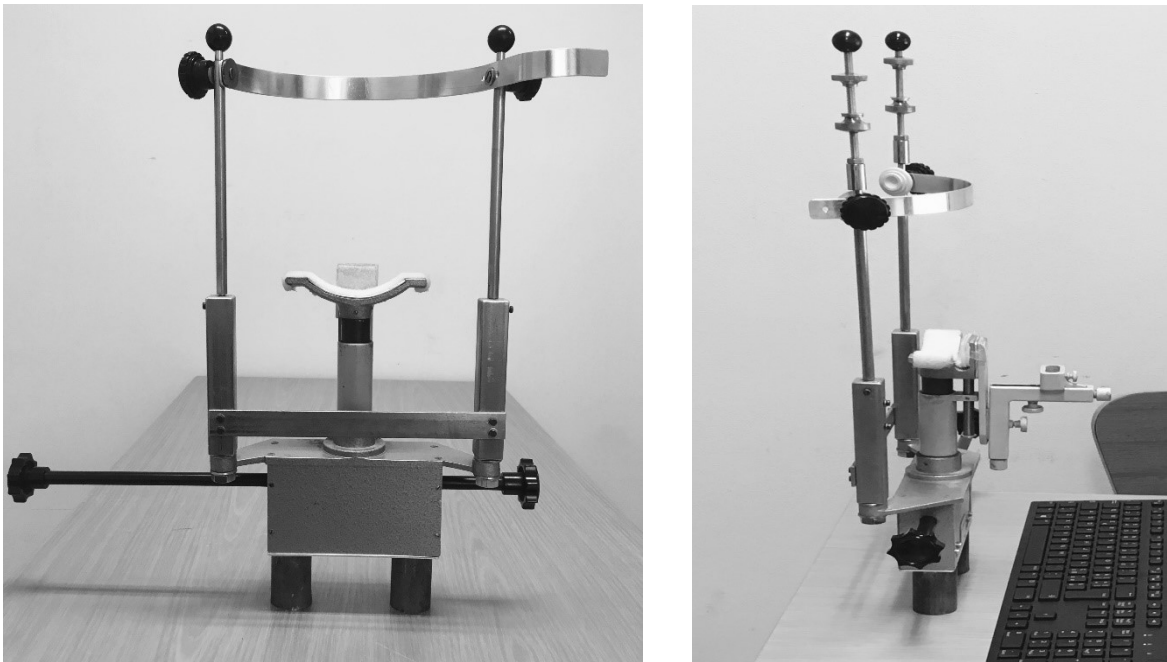


Figura 15: Mentoniera utilizzata durante le acquisizioni

Il supporto è costituito da due zone di appoggio: una mentoniera su cui appoggiare il mento e una fascia su cui appoggiare la fronte.

La mentoniera è regolabile in altezza attraverso una manopola girevole che consente di realizzare una variazione di altezza per un massimo di 5 cm. Con questa manopola, l'altezza del mento dal tavolo può essere regolata da un minimo di 25 cm a un massimo di 30 cm.

La regolazione è necessaria per far in modo che l'altezza degli occhi corrisponda al centro dello schermo, regolabile anch'esso in altezza e posizionato a una distanza di 50 cm dagli occhi. Anche la fascia su cui dovrà essere appoggiata la fronte è regolabile e ha tre possibili

posizioni. Tre fori di aggancio consentono di diminuire o aumentare il suo raggio per adattarla al meglio al soggetto esaminato.

Le dimensioni totali dell'intero supporto sono di 22 cm in larghezza e 41 cm in altezza.

3.1.2. Videocamera e schermo

Lo schermo utilizzato per la visualizzazione degli stimoli è un ASUS VG248QE che dispone di un range di refresh rate fino a 144Hz. La risoluzione è full HD 1920 x 1080, con rapporto di contrasto di 80.000.000:1 e luminosità di 350cd/m². Lo schermo misura 19,1 pollici e ha una dimensione interna di 53,2 * 31,2 cm. Lo schermo è regolabile in altezza e può quindi essere adattato ad ogni soggetto esaminato per consentirgli la visuale ottimale.

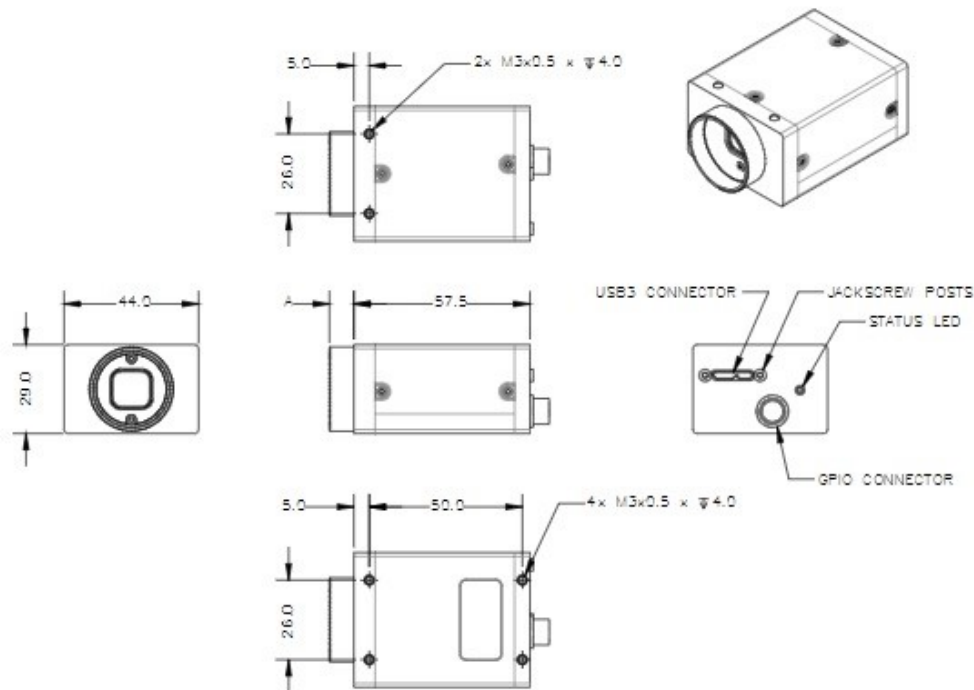


Figura 16: Estratto del datasheet della videocamera che riporta le varie dimensioni²

La videocamera utilizzata è una Flir Grasshopper 3 che permette di ottenere immagini ad alta qualità ed è adatta per molte applicazioni scientifiche tra cui l'oftalmologia. Il modello utilizzato è il GS3-U3-51S5M-C. Questo modello permette di fare acquisizioni fino a 75 FPS, l'uscita video è monocromatica e la risoluzione è di 5 Megapixels. Il sensore al suo

² <https://flir.app.boxcn.net/s/859w5hl6yv25yo4c5aqvdl2ask34n4i/file/418656969139>

interno è un Sony IMX250 di formato 2/3” e la dimensione dei suoi pixel è di 3,45µm.

Sulla telecamera è stato montato un obiettivo FUJINON 12XA-5M. Questo è un obiettivo ad alta risoluzione, compatibile con i sensori di 2/3” e di 3,45µm di pixel size, equivalenti a 5MP. La sua lunghezza focale è fissa ed è di 12mm.

Infine, sulla videocamera è stato montato un filtro per la luce visibile.

3.1.3 Software: SpinView FLIR

Il software SpinView fornisce una comoda interfaccia grafica per gestire la calibrazione dell'immagine ripresa dalla videocamera.

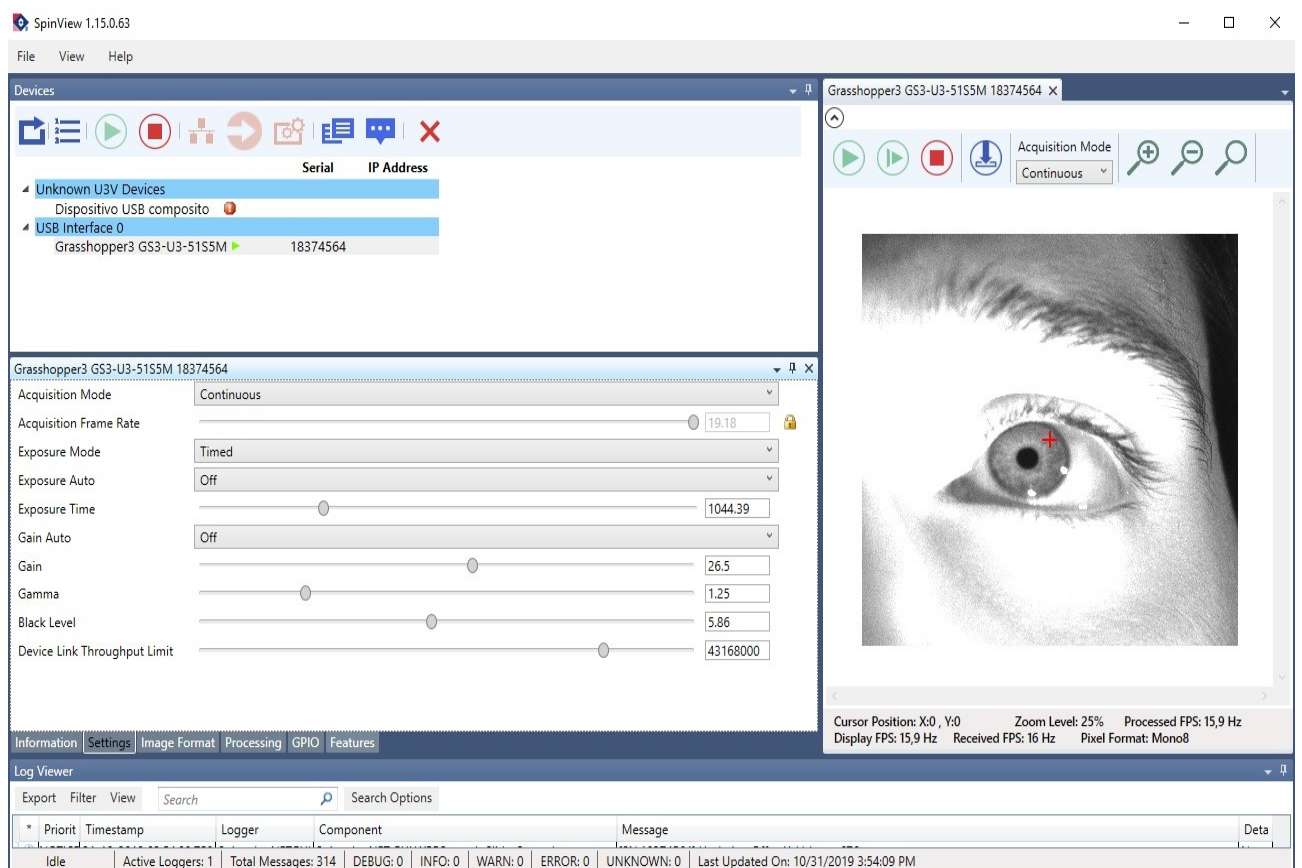


Figura 17 Interfaccia grafica SpinView

Il primo parametro da settare ai fini di una buona acquisizione è il **tempo di esposizione**, ovvero il tempo durante il quale l'otturatore rimane aperto per permettere alla luce di

raggiungere il sensore. In questo caso il valore è di 1044.39, ma può oscillare tra il valore 900 e 1050 in base alla luminanza della stanza.

Successivamente, si passa con il settaggio del **gain** (guadagno). Tale valore, sempre in base alla luminanza della stanza, oscilla tra 24 e 27. Non è possibile andare oltre, in quanto il rapporto segnale/rumore aumenta troppo.

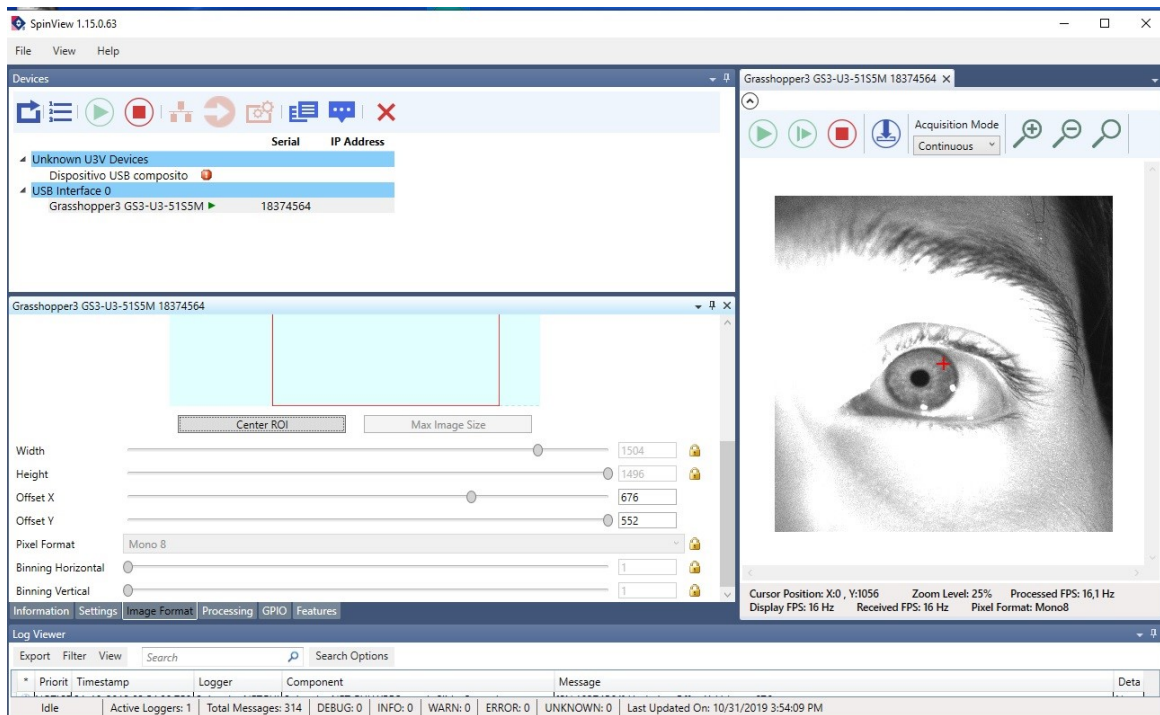


Figura 18 Interfaccia grafica SpinView

Dopo aver settato i parametri riguardanti la qualità dell'immagine, si passa al settaggio dei parametri che interessano le dimensioni dell'immagine e la selezione della regione d'interesse (ROI).

La dimensione scelta di default per ogni sessione di acquisizione è di 1504x1496 (il massimo settabile è 2048x2048).

Si noti che è possibile anche aggiungere degli offset sia lungo l'asse x che y, in modo da poter meglio centrare l'immagine senza dover spostare la videocamera o far muovere il soggetto.

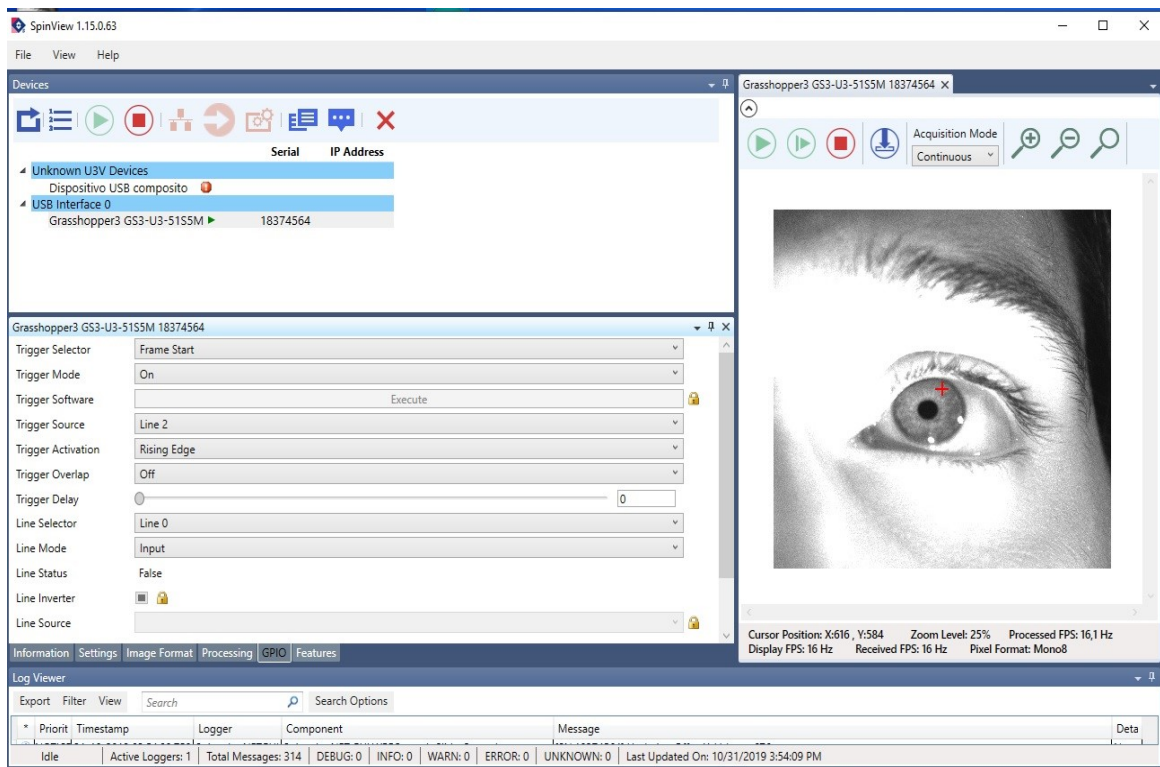


Figura 19 Interfaccia grafica SpinView

Infine, si settano i parametri che riguardano le impostazioni del trigger della fotocamera (il funzionamento sarà meglio discusso nel prossimo paragrafo):

- Trigger Selector → Frame Start
- Trigger Mode → On
- Trigger Source → Line 2
- Trigger Activation → Rising Edge
- Trigger Overlay → Off

3.1.4 Arduino

Infine, attraverso il microcontrollore Arduino Uno è stato controllata l'acquisizione delle immagini durante la sessione. Questo è possibile attraverso il connettore GPIO della videocamera, di cui di seguito è riportato il pinout:


Diagram	Color	Pin	Function	Description
	Black	1	IO	Opto-isolated input (default Trigger in)
	White	2	O1	Opto-isolated output
	Red	3	IO2	Input/Output/serial transmit (TX)
	Green	4	IO3	Input/Output/serial receive (RX)
	Brown	5	GND	Ground for bi-directional IO, V _{EXT.} +3.3 V pins
	Blue	6	OPTO_GND	Ground for opto-isolated IO pins
	Orange	7	V _{EXT}	Allows the camera to be powered externally
	Yellow	8	+3.3 V	Power external circuitry up to 150 mA
	To configure the GPIO pins, consult the General Purpose Input/Output section of your camera's Technical Reference Manual.			

Figura 15 Pinout GPIO connector della videocamera³

I pin utilizzati sono stati l'1 ed il 5:

- **Pin 1:** è il trigger della videocamera. Quando su questo pin viene posta una tensione (in questo caso di 5V tramite i pin digitali dell'Arduino) rispetto al ground (pin 5), la videocamera si attiva.
- **Pin 5:** pin collegato al GND di Arduino.

Ultimo elemento per il controllo dell'acquisizione è un fotodiodo che, posto nell'angolo in alto a sinistra del monitor, dice al microcontrollore quando mandare i 5V alla videocamera e salvare l'immagine d'interesse.

In pratica, si sceglie un valore di soglia e sempre con l'Arduino si legge costantemente il valore misurato dal fotodiodo (è un segnale analogico ad 8 bit, quindi compreso tra 0 e 1023). Quando il valore misurato dal fotodiodo supera il suddetto valore di soglia, l'Arduino setterà ad HIGH il pin digitale collegato alla videocamera, fornendo così i 5V.

³ <https://flir.app.boxcn.net/s/x5ga5jektmegz60i11zo56vg83ldno2x/file/418659633483>

3.2 Analisi del dataset e labeling

Una volta testato il corretto funzionamento del sistema, si è proceduto con le acquisizioni. Il dataset raccolto è stato di circa 5000 immagini.

Per ogni soggetto in una sessione si sono acquisite 500 immagini ad 1 immagine al secondo, il che portava ogni sessione a durare circa 8 minuti.

Per ogni soggetto, prima di cominciare l'acquisizione, si è provveduto a centrare la pupilla con il centro dell'immagine da acquisire, tramite l'ausilio di una croce disegnata da SpinView che identifica il centro immagine.

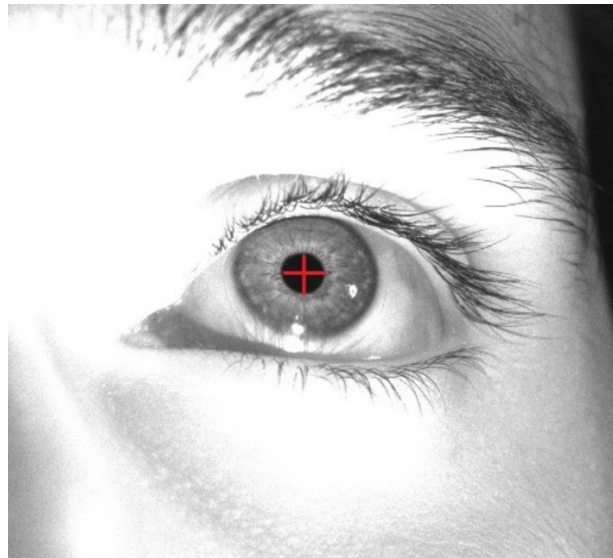


Figura 16 Calibrazione immagine prima dell'acquisizione

Una volta acquisito un numero soddisfacente di immagini, si è passati all'analisi delle stesse tramite un'implementazione in codice Python dell'algoritmo starburst, che è stato già ampiamente trattato nel capitolo precedente.

Nella figura successiva, è riportata la tabella di output prodotta dall'algoritmo di localizzazione del centro della pupilla.

1	0.0	590.1876238856403	345.0204258695899	43.85725332295615	40.94662600735037
2	1.0	577.9771531200242	387.0070300060487	46.95674273616069	43.79733748178519
3	2.0	589.8607097483214	344.5375359153028	45.35627883112532	42.01762925684129
4	3.0	586.8509308580342	354.870232632215	46.722739788164716	43.629800602678
5	4.0	504.734940109214	349.13623292166676	46.60037561371551	42.966039202022706
6	5.0	582.7567269101837	350.31950646598557	44.3487298940433	40.47446414379835
7	6.0	585.9507644097173	352.9989276782626	43.88970241290234	40.36007335909934
8	7.0	585.7008295299945	352.19931092809077	43.445768478088986	40.17143781169639
9	8.0	586.9150334198217	352.1567385194994	40.984697916590164	38.119099106317286
10	9.0	585.9977997679616	351.2736931006722	40.00335135813241	37.021653704158545
11	10.0	586.1289078308527	351.8395105721712	40.45151422868527	37.5796146776168
12	11.0	585.0841350520709	356.1142508216047	43.090367592576115	39.98004311275527
13	12.0	573.8059781561658	354.70696970572806	46.103558085793914	42.44698034778821
14	13.0	561.3196383890313	391.3724597707586	45.21404581575524	42.139134746730214
15	14.0	591.2746533787323	346.8151993821066	42.62210770360407	39.39682844132219
16	15.0	583.3491931744973	355.0228665679796	39.90387448715941	37.01601671792126
17	16.0	583.2098202847795	355.257249206504	41.35572998946406	38.251112850956176
18	17.0	579.334505392574	356.5279206435528	43.753821528977824	40.5255089352106
19	18.0	581.5710634552554	358.952875312939	42.9191700548258	39.76925604480166
20	19.0	583.2870779821211	360.3194561908391	42.55319774016296	39.48043253030744
21	20.0	582.3080727392875	362.7533013721396	42.168106850921575	38.74735134120144
22	21.0	583.1806670435003	363.5947892431987	43.453215058072125	40.389544806019124
23	22.0	579.6506549698236	359.1949797593594	44.1297948061256	41.19485167733769
24	23.0	581.7386438863969	362.35548453488303	42.48699190280762	39.338623741865646
25	24.0	587.7622478819535	362.70470337559584	40.56334592437618	37.704772184138314
26	25.0	585.2887492201094	362.7178151377759	41.589018536107545	38.788316553424416
27	26.0	584.6461168773783	364.834972948551	41.18706579897477	38.39488016517949
28	27.0	579.2148130475193	356.69126055756817	39.62980560274846	36.569321422605306
29	28.0	579.7325910531347	356.00299185335734	39.91529832796721	36.83701231258049
30	29.0	582.7636230273085	356.63776335414417	40.06758214054726	37.042433813022846
31	30.0	584.9815428281033	351.2087827209529	38.945228492915874	36.10178768954325
32	31.0	582.448424139995	350.73735919034743	39.303920078636835	36.26616019541836
33	32.0	581.0736388061948	350.97065931318366	39.33099548847377	36.39626589605343
34	33.0	584.184423188186	351.4845848628886	37.6828256044887	34.75285137866627

Figura 17: output dell' algoritmo di localizzazione del centro della pupilla

L'output dell' algoritmo di localizzazione del centro della pupilla è organizzato in tabelle dove ogni colonna riporta, riguardo ad ogni immagine:

- **Prima colonna:** id dell'immagine di riferimento;
- **Seconda colonna:** coordinata x del centro della pupilla;
- **Terza colonna:** coordinata y del centro della pupilla;
- **Quarta colonna:** lunghezza del semiasse maggiore dell'ellisse "fittata" con la pupilla;
- **Quinta colonna:** lunghezza del semiasse minore dell'ellisse "fittata" con la pupilla.

3.3 Acquisizioni e dati

Una volta validato il funzionamento del set-up, si è passati alle acquisizioni. In totale sono stati acquisiti oltre 25 soggetti, con una media di 250 immagini per soggetto, per un totale di oltre 6000 immagini acquisite.

Il primo set d'immagini raccolte ha riguardato 11 soggetti. Da ogni soggetto sono state acquisite 200 immagini, per un totale quindi di 2200 immagini.

Successivamente è stato riportato in un grafico (scatter plot o bubble-chart) la posizione dei centri della pupilla rispetto la dimensione di ogni immagine, relativamente ad ogni sessione:

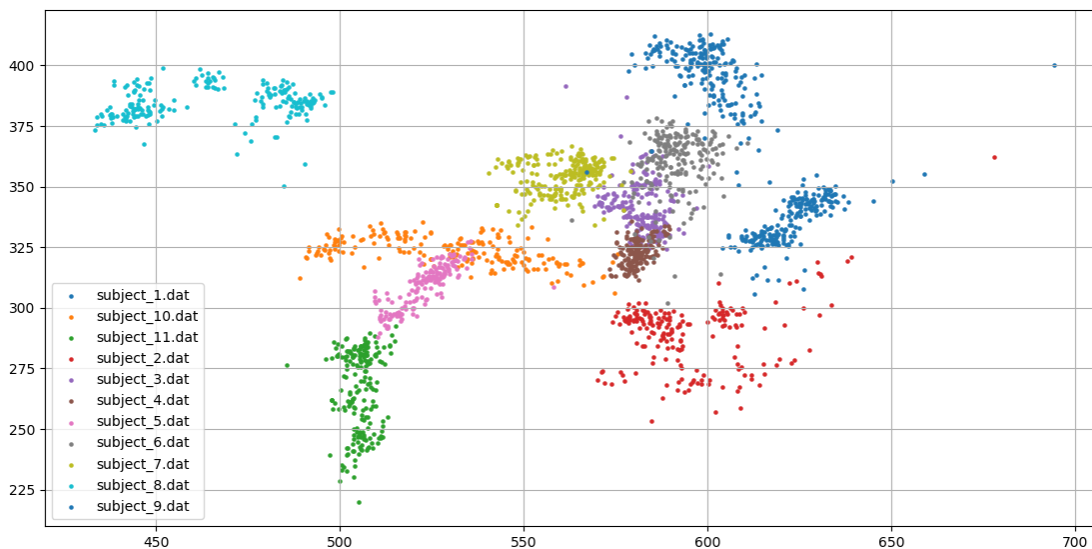


Figura 18 Bubble chart relativo al primo data set raccolto (11 soggetti, 2200 immagini)

C'è da evidenziare che prima dell'inizio di ogni acquisizione, la dimensione dell'immagine veniva modulata in funzione della posizione dell'occhio del soggetto. Questo permetteva di limitare l'immagine all'occhio stesso, ma non è stata una strategia ottimale in quanto questo ha portato, come sarà meglio descritto nel prossimo paragrafo, non pochi problemi durante la fase di data augmentation.

Per questo motivo, è stata ripetuta la fase di raccolta dati. Per prima cosa si è scelto un valore fisso come dimensione dell'immagine, ovvero **1504x1496**.

Questo sia per evitare di avere delle nuvole di centri troppo distanti tra un soggetto e l'altro,

sia per semplificare la successiva fase di data augmentation. Un altro problema che sarebbe potuto insorgere, era che questo set d'immagini avrebbe portato la rete neurale a riconoscere la pupilla solo se si fosse trovata in una finestra grande 80x80 pixels, che non rende certo ottimale né utile la stessa rete neurale.

Per questi motivi si è ritenuto necessario procedere con un'ulteriore raccolta dati. Questa volta sono stati esaminati oltre 20 soggetti. Prima di ogni acquisizione, veniva effettuata una calibrazione manuale. Questa fase consisteva nel far posizionare il centro della pupilla con il centro dell'immagine, esattamente nel modo descritto nel paragrafo 3.2 (tramite l'ausilio del software spinview). Alla fine, si produceva un *bubble chart* per ogni soggetto e se la nuvola di centri trovati fosse stata troppo distante dal centro immagine o dagli altri soggetti, l'acquisizione in questione sarebbe stata scartata. Tenuto conto di queste considerazioni, solo 14 soggetti sono stati ritenuti validi, per un totale di 4480 immagini (e relativi centri).

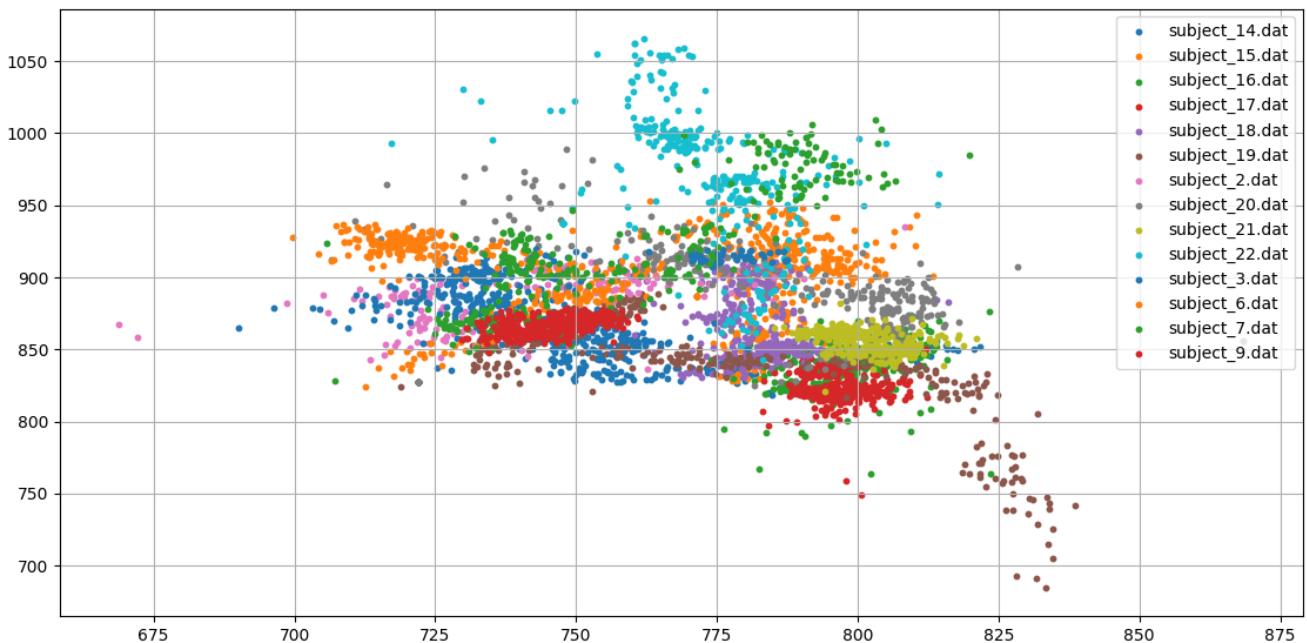


Figura 19 Bubble chart relativo al second data set raccolto (14 soggetti, 4480 immagini)

Lo *scatter-plot* è stato ottenuto tramite la libreria Python **matplotlib**. Di seguito è riportato il codice Python scritto per realizzarlo:

```
1  """
2  Created on 18/10/2019
3
4  @author: ap
5  """
6  import os
7  import os.path
8  import sys
9  sys.path.append(os.path.join(os.path.dirname(__file__), '..'))
10
11 from lsr import store
12 import numpy as ns
13 from matplotlib import pyplot as plt
14
15 data = []
16 x_y = []
17 n_file = os.listdir()
18
19 #Collecting data from files
20 for i in n_file[1:]:
21     data.append(store.loadAsciiArray(i))
22
23 #Crop
24 for i in data:
25     x_y.append(i[:, 1:3])
26
27 #Replacing 0,0 points with mean of all x and y
28 count = 0
29 for i in x_y:
30     mean_x = ns.mean(i[:, 0])
31     mean_y = ns.mean(i[:, 1])
32     for j in range(i.shape[0]):
33         if i[j, 0] == 0:
34             i[j, 0] = mean_x
35         if i[j, 1] == 0:
36             i[j, 1] = mean_y
37     count += 1
38
39
40 #Plot
41 figure, ax = plt.subplots()
42
43
44 for i in x_y:
45     ax.scatter(i[:,0], i[:,1], s=100)
46 print(count)
47 ax.legend(n_file[1:])
48 ax.grid(True)
49 plt.show()
50
```

Figura 20 Codice scritto per realizzare il bubble-chart

3.4 Data augmentation

Il data augmentation è una tecnica molto utile che viene utilizzata per generare un dataset sintetico a partire dai propri dati. Questo dataset “artificiale” viene ottenuto facendo delle modifiche alle immagini originali, come ad esempio ritaglio casuale, ingrandimenti, riduzione di dimensione, rotazione, ecc.

L’idea di base sarebbe quella di tagliare le immagini per ottenere uno shift della posizione della pupilla. Se, ad esempio, in un’immagine 1500x1500 viene trovato dall’algoritmo il centro della pupilla in posizione $x = 800$ e $y = 860$, e si effettua un *crop* che riguarda le prime cento righe e cento colonne, allora ne consegue che l’immagine avrà una risoluzione pari a 1400x1400 e quindi il centro della pupilla adesso si troverà in posizione $x = 700$ e $y = 760$.

In figura 21 è riportato proprio questo esempio. È riportata una figura contenente due immagini: quella di sinistra è quella originale, con dimensioni 1088x592 e centro nelle

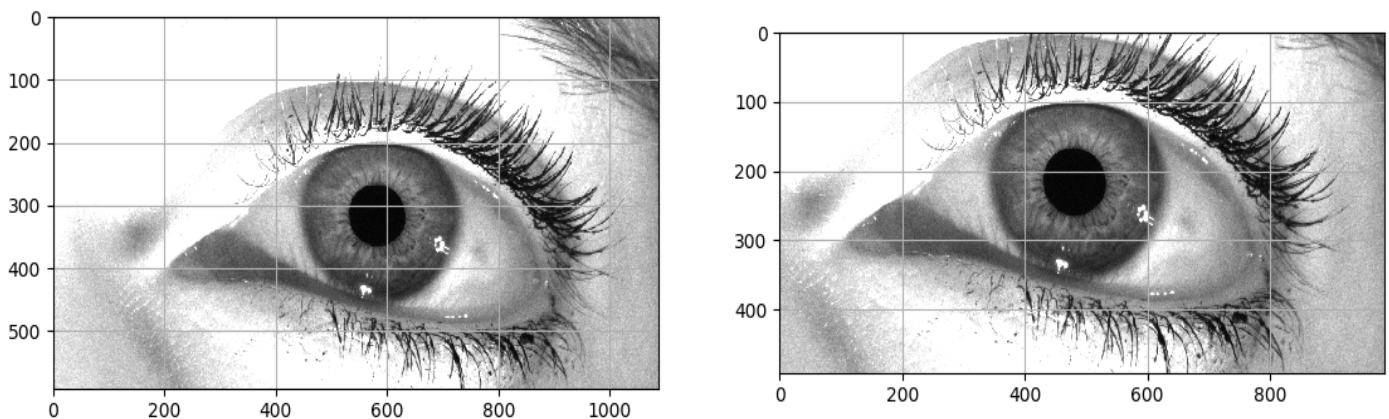


Figura 21 Esempio di shift della pupilla tramite un crop d’immagine

coordinate (580, 315); l’immagine di destra invece è stata ritagliata al centro e in alto (di 100 righe e 100 colonne), ne consegue che le dimensioni saranno 988x492 e il centro della pupilla sarà identificato dalle coordinate (480, 215).

Come emerge da questo esempio, è possibile quindi aumentare il proprio dataset applicando questo tipo di logica.

È evidente che il taglio deve rispettare certe regole affinché si produca un risultato efficace. Banalmente, affinché il ritaglio produca un effettivo shift deve essere effettuato a sinistra e/o in alto rispetto il centro dell'immagine, altrimenti il valore delle coordinate rimarranno invariate.

3.4.1 Viewport dell'immagine

Poiché gli inconvenienti dell'applicare un ritaglio su di un'immagine per effettuare un consistente data augmentation possono essere molteplici, è necessario creare una tabella contenente diverse informazioni riguardo l'immagine, ed anche per tener traccia delle operazioni effettuate su di esse.

1	0.0	0.0	739.0	734.0	44.0	42.0	0.0	140.0	1454.0	1405.0
2	0.0	2.0	740.0	709.0	46.0	44.0	0.0	140.0	1454.0	1405.0
3	0.0	3.0	723.0	731.0	46.0	44.0	0.0	140.0	1454.0	1405.0
4	0.0	4.0	729.0	729.0	45.0	43.0	0.0	140.0	1454.0	1405.0
5	0.0	5.0	733.0	736.0	43.0	41.0	0.0	140.0	1454.0	1405.0
6	0.0	6.0	720.0	726.0	43.0	40.0	0.0	140.0	1454.0	1405.0
7	0.0	7.0	723.0	728.0	43.0	40.0	0.0	140.0	1454.0	1405.0
8	0.0	8.0	726.0	739.0	45.0	43.0	0.0	140.0	1454.0	1405.0
9	0.0	9.0	716.0	732.0	44.0	42.0	0.0	140.0	1454.0	1405.0

Figura 22 Head della tabella contenente le informazioni riguardo ogni immagine del dataset

Ogni riga della tabella identifica una singola immagine, mentre le colonne sono così costituite:

- **Session_ID** → identificativo della sessione cui si fa riferimento. Questo dataset contiene dati provenienti da 14 sessioni, pertanto questo valore sarà compreso tra 0-13;
- **Image_ID** → identificativo dell'immagine all'interno della sessione di riferimento. Per ogni sessione sono state acquisite 320 immagini, pertanto questo valore sarà compreso tra 0-319
- **pup_x** → valore della coordinata x del centro della pupilla. Questo dato è di tipo intero.
- **pup_y** → valore della coordinata y del centro della pupilla. Questo dato è di tipo intero.
- **Pup_width** → valore del semiasse maggiore dell'ellisse che "fitta" la pupilla.
- **Pup_height** → valore del semiasse minore dell'ellisse che "fitta" la pupilla

- **shift_x** → valore che rappresenta lo spostamento lungo l'asse x del centro della pupilla
- **shift_y** → valore che rappresenta lo spostamento lungo l'asse y del centro della pupilla
- **num_rows** → numero delle righe che compongono l'immagine
- **num_col** → numero di colonne che compongono l'immagine

Le ultime 4 colonne, identificano il viewport di ogni immagine.

3.4.2 Shift dei punti

Prima di creare tale tabella, sono stati applicati dei criteri. Il primo criterio è stato quello di scartare i dati in cui l'algoritmo non ha identificato la pupilla, e pertanto la dimensione della pupilla sarà 0. Per fare ciò si è implementato il seguente metodo:

```
def onlyValid(dd):
    """
    Finds valid images. If pupil was not found, pupil size is zero, so we can just check last column in dd

    Returns a subarray of the original data array
    """
    jj = dd[:, -1] > 1.0
    return dd[jj, :]
```

Figura 23 Metodo python per la selezione dei soli punti validi dal dataset

Questo metodo prende in input l'array bidimensionale restituito dall'algoritmo di localizzazione della pupilla (colonne → coord_x, coord_y, pupil_width, pupil_height). Su questo array viene fatto un semplice controllo sulla dimensione della pupilla: se questa è maggiore di 1, allora all'interno dell'array "jj" scriverà TRUE, altrimenti FALSE. Il metodo restituisce un sub-array bidimensionale iniziale ma solo con gli indici validi (identificati appunto dall'array jj).

In secondo luogo, si è deciso di compattare la nuvola di punti provenienti da ogni dataset applicando i seguenti shift:

- **Shift_x** (per sessione): (0, -20, -10, -15, -30, -15, -15, -15, -15, -15, -15, -15, -15, -25)

- **Shift_y** (per sessione): (-140, -140, -140, -130, -150, -130, -140, -90, -40, -140, -130, -140, -140, -220)

Questo shift effettuato in fase pre-augmentation è stato fatto per i due motivi: 1) per compattare la nuvola dei centri della pupilla e 2) per avere le nuvole di punti relative ad ogni sessione centrata intorno al pixel (750,750) con i punti distribuiti tra le colonne/righe 700-800.

Cloud di punti dopo lo shift:

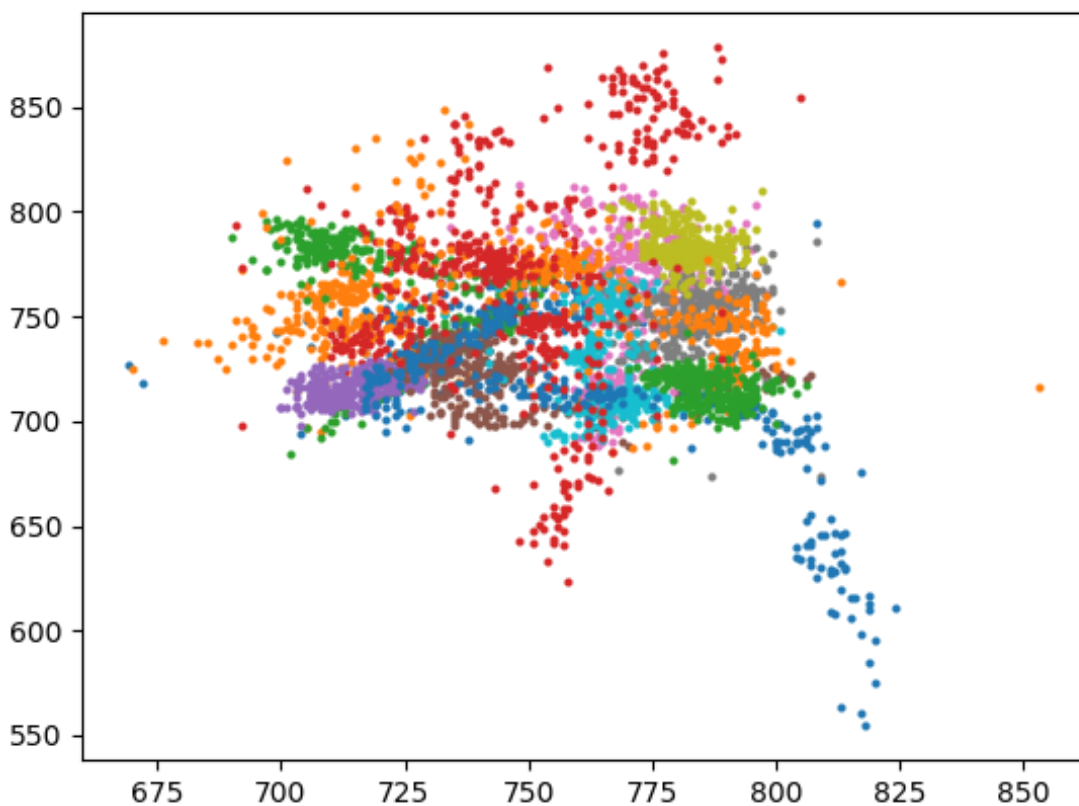


Figura 24 Bubble chart del dataset dopo aver effettuato lo shift

Successivamente, per avere una nuvola ancora più uniforme, si è selezionato solo una finestra di dimensioni 100x100 pixels di punti, ovvero quelli compresi tra le righe 700-800 e le colonne 700-800. Nella figura successiva, è riportato lo scatter-plot dei punti appena

citati.

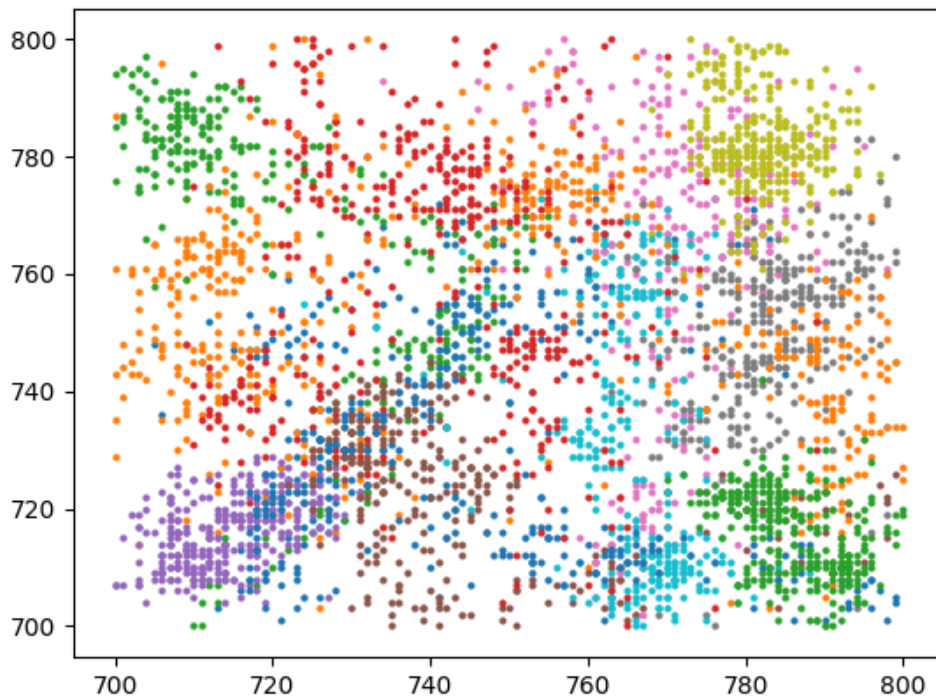


Figura 25 Bubble chart dei punti shiftati, compresi tra le righe/colonne 700:800 (un diverso colore indica un diverso soggetto)

3.4.3 Density Function

Prima di procedere iterazione che porterà all'aumento dei dati, è necessario definire il criterio secondo cui tale iterazione debba procedere. Quando si addestra una rete neurale l'ideale sarebbe fornire dataset uniforme, mentre in questo caso (come emerge dalla figura 25) ci sono molte zone vuote ed altre molto ricche di punti. È chiaro che addestrare una rete neurale con questo dataset di partenza porterebbe a dei risultati poco efficaci se si paragonano a quelli che si potrebbero ottenere se si somministrasse un dataset più uniforme.

Pertanto, è necessario operare secondo una density function che mostri quali sono le zone che presentano picchi di punti e quali invece hanno bisogno di data augmentation.

La density function applicata consiste in una funzione gaussiana applicata ad ogni singolo punto del dataset preso in considerazione.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad \text{con } x \in \mathbb{R}$$

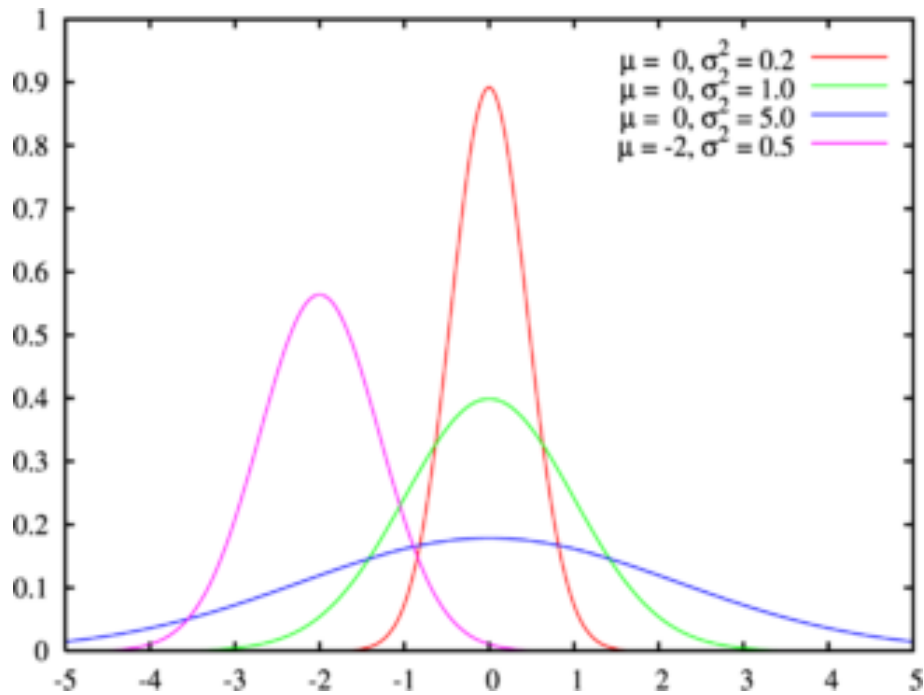


Figura 25 Funzione di densità con diversi valori di sigma

La precedente funzione prende il nome di distribuzione normale ed è la funzione applicata ai punti del dataset, attraverso il seguente metodo:

```
def computeDensity(dd, sigma, min_x, max_x, min_y, max_y):
    """
    Computes density function convolving data points with Gaussians, over the support range
    dd: Data set, one row per data point, two columns (x, y)
    sigma: Standard deviation of the 2-D Gaussian function
    """
    X, Y = np.meshgrid(np.arange(min_x, max_x+1), np.arange(min_y, max_y+1))
    Z = np.zeros_like(X, dtype=np.float)
    S = 2.0 * sigma**2
    N = dd.shape[0]
    for i in range(N):
        Z += np.exp(-(X - dd[i, 0])**2/S) * np.exp(-(Y - dd[i, 1])**2/S)
    return (X, Y, Z)
```

Figura 26 metodo Python per il calcolo della density function

Dopo aver calcolato le matrici X, Y e Z tramite l'ausilio della libreria **matplotlib** è possibile ottenere i seguenti grafici:

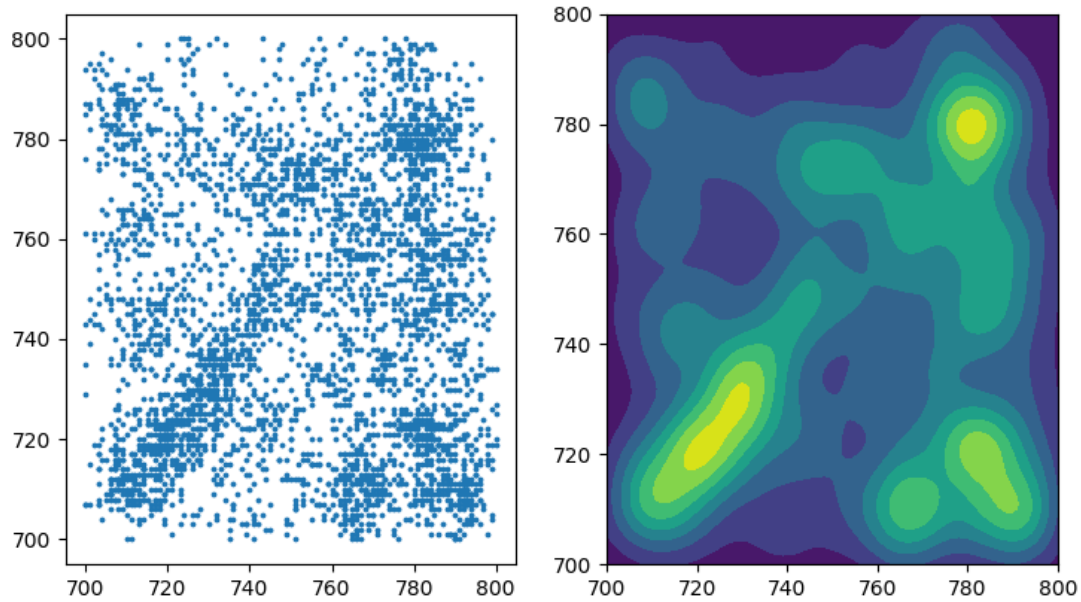


Figura 27 a) scatter plot dei punti b) grafico raffigurante la densità dei punti, applicando la gaussiana con una deviazione standard pari a 5

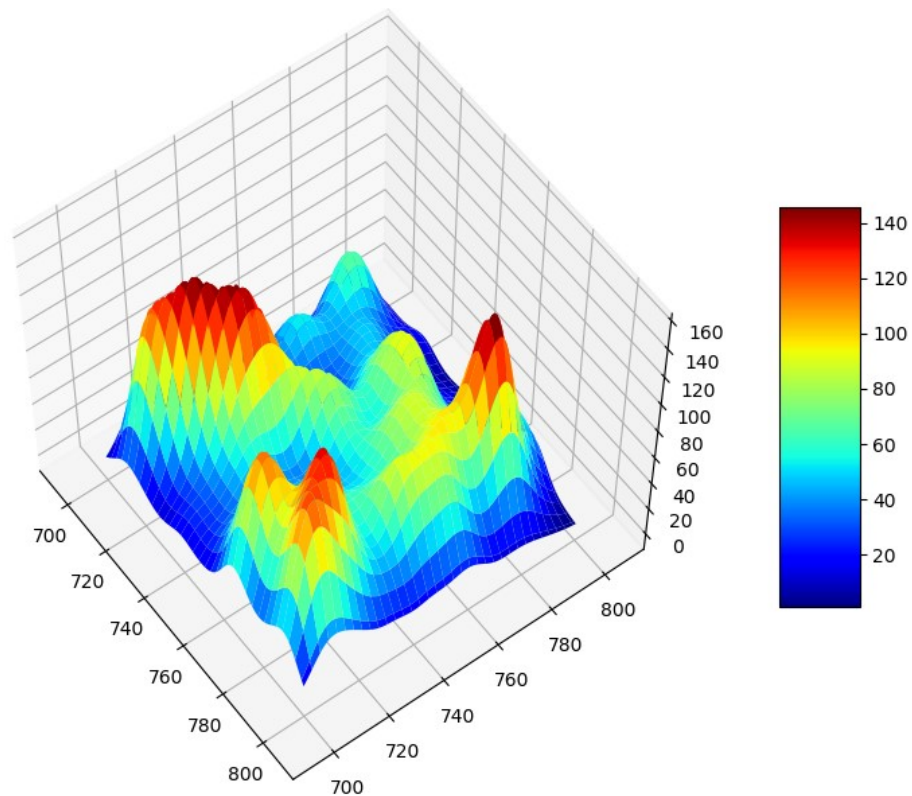


Figura 28 Grafico 3D del dataset dopo aver applicato la density function

Interessante è anche poter vedere come cambia il grafico della densità al variare di sigma:

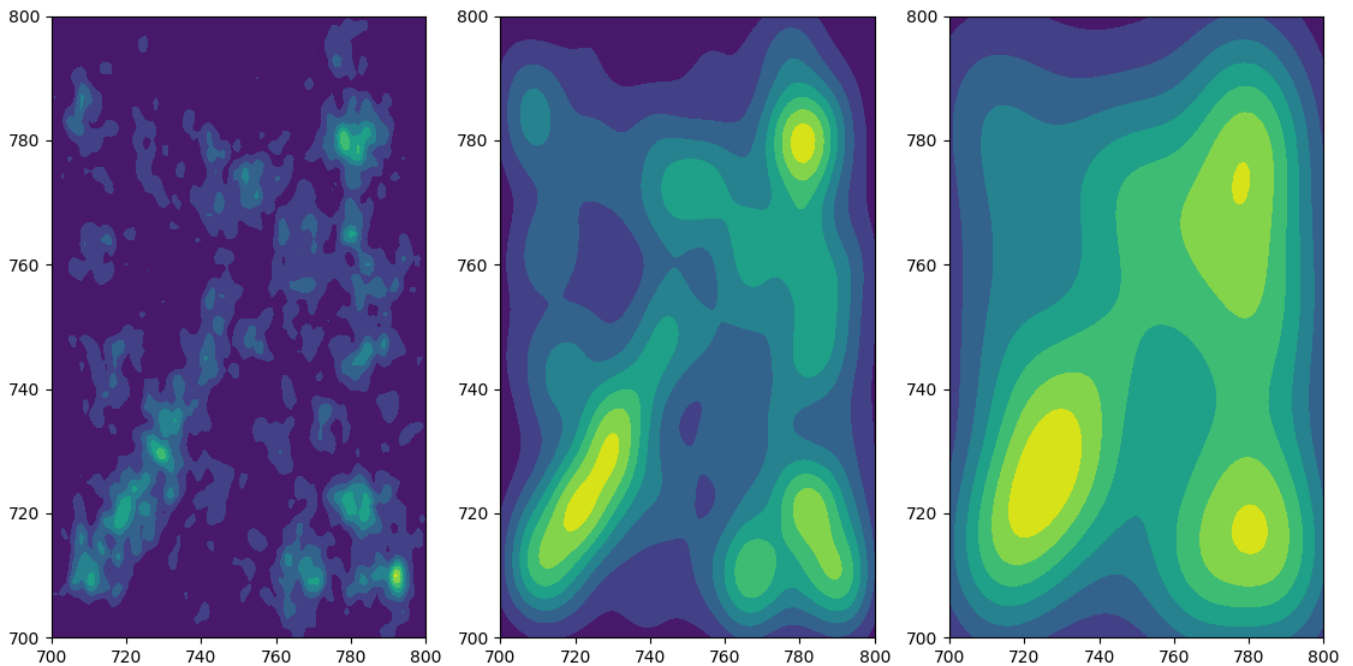


Figura 29 a) $\sigma = 1$; b) $\sigma = 5$; c) $\sigma = 10$

3.4.4 Aumento del Dataset

Arrivati a questo punto, si procede in questo modo:

- **Passo 1:** Si trova il minimo della density function;
- **Passo 2:** In questo punto appena trovato, si aggiunge un'immagine che viene presa random dal dataset originale, scartando quei punti che non sono validi per via dei limiti imposti dal viewport e anche quei punti che renderebbero il viewport dell'immagine troppo piccolo;
- **Passo 3:** Una volta completato il passo 2, si aggiunge questo nuovo punto alla density function e si ricomincia da capo (il numero di iterazioni viene scelto prima di applicare data augmentation).

Di seguito, è riportato il flowchart che descrive in modo sommario i passi dell'algorithm, per un numero di iterazioni prestabilito pari a 10000:

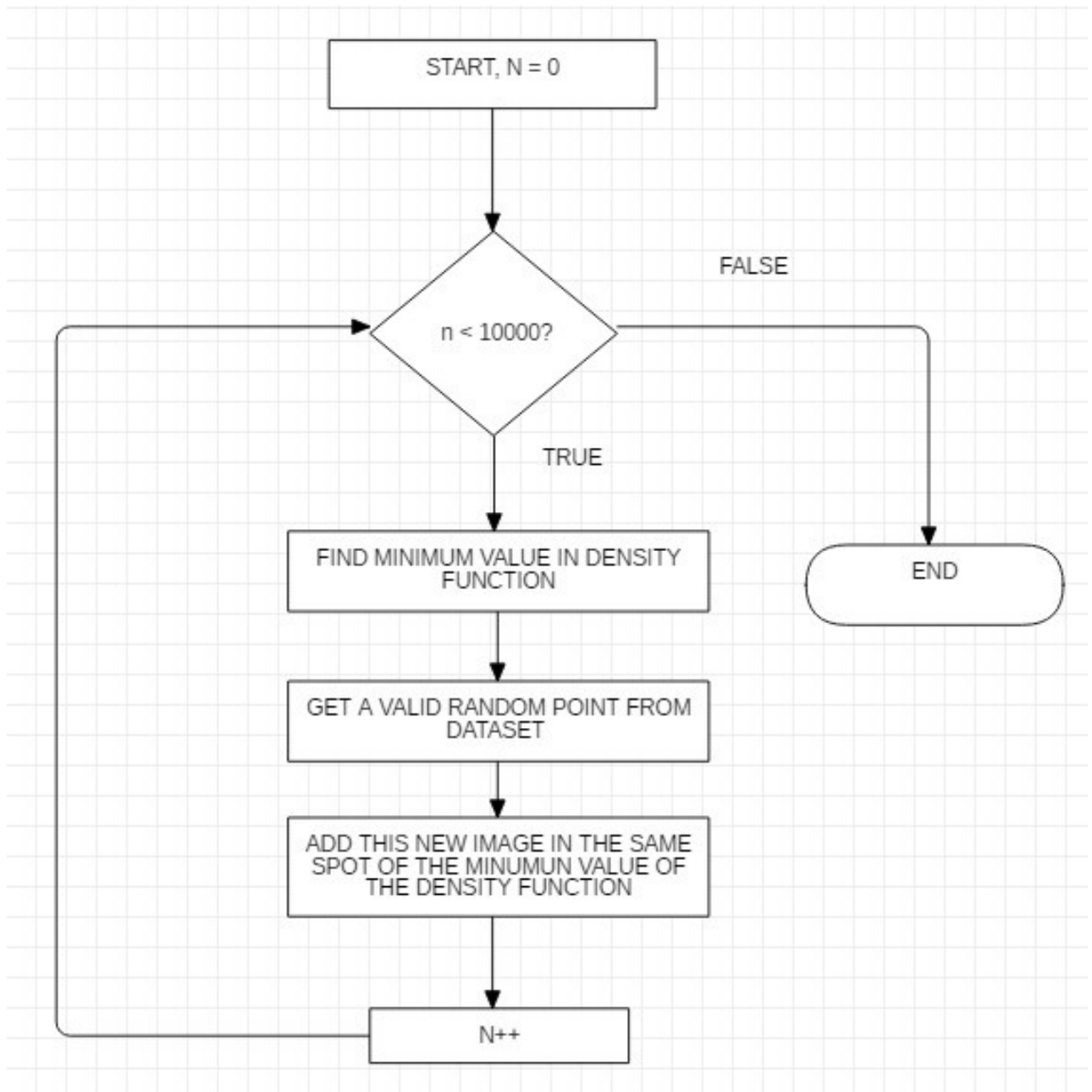


Figura 30 Flowchart dell'algorithm di data augmentation

Il metodo che trova il minimo della density function è stato implementato come segue:

```
def getNext(X, Y, Z, sigma):
    """
    Finds the minimum in the Gaussian density function, and returns it.
    The density function is updated to reflect this new point
    X, Y, Z:      Outputs of computeDensity
    sigma:        Standard deviation of the 2-D Gaussian function
    """
    S = 2.0 * sigma**2
    ind = np.unravel_index(np.argmin(Z, axis=None), Z.shape)
    Z += np.exp(-(X - X[ind])**2/S) * np.exp(-(Y - Y[ind])**2/S)
    return X[ind], Y[ind]
```

Figura 31 Metodo che restituisce il minimo della density function

Questo metodo restituisce le coordinate x,y del punto da creare artificialmente.

A questo punto, si deve estrarre in modo pseudo-casuale un'immagine dal dataset ed effettuare un *crop* in modo tale che il punto venga “*shiftato*” nella posizione desiderata (minimo della density function) e soprattutto si cambi il viewport in modo valido.

Il metodo in questione, *getFromDataset* prende in input:

- **dd** → tabella dataset, undici colonne, una riga per immagine;
- **isz** → dimensioni dell'immagine originale (limite del viewport);
- **pup_x** → coordinata x del punto da riempire;
- **pup_y** → coordinata y del punto da riempire;
- **p_size** → tabella che contiene per ogni punto il valore dei semiassi della pupilla

Il metodo in questione, deve selezionare in modo casuale un'immagine dal dataset che abbia il centro della pupilla non troppo distante dal punto che si vuole riempire, e soprattutto non renda il viewport non valido (questa condizione avviene quando i valori dello shift assumono un valore negativi).

Quindi il metodo continuerà a estrarre valori dal dataset finché non trova un punto che rispetta queste condizioni.

```

# New left edge of viewport cannot be smaller than zero
c1 = (dd[:, 6] - (pup_x - dd[:, 2])) >= 0

# New top edge of viewport cannot be smaller than zero
c2 = (dd[:, 7] - (pup_y - dd[:, 3])) >= 0

# New left edge of viewport must not infringe on pupil with a margin
c3 = (dd[:, 6] - (pup_x - dd[:, 2])) <= (pup_x - dd[:, 4] - 10)

# New top edge of viewport must not infringe on pupil with a margin
c4 = (dd[:, 7] - (pup_y - dd[:, 3])) <= (pup_y - dd[:, 5] - 10)

# New right edge of viewport must not infringe on pupil with a margin
c5 = (dd[:, 8] - (pup_x - dd[:, 2])) >= (pup_x + dd[:, 4] + 10)

# New bottom edge of viewport must not infringe on pupil with a margin
c6 = (dd[:, 9] - (pup_y - dd[:, 3])) >= (pup_y + dd[:, 5] + 10)

# Right edge of viewport must be within image (or almost)
c7 = (dd[:, 8] - (pup_x - dd[:, 2])) <= (isz[:, 0] + 20)

# Right edge of viewport must be within image (or almost)
c8 = (dd[:, 9] - (pup_y - dd[:, 3])) <= (isz[:, 1] + 20)

# Find acceptable points
index = np.nonzero(c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8)[0]

```

Figura 32 Parte del metodo getFromDataSet

In figura 32 è riportato l'inizio del metodo `getFromDataSet`. La prima cosa che è stata fatta è stato definire le condizioni di validità di estrazione dei punti:

- le condizioni **c1** e **c2** assicurano che i valori a seguito dello shift non assumano valori minori di zero. In altre parole, consentono di rimanere all'interno dell'immagine originale;
- le condizioni **c3**, **c4**, **c5**, **c6** garantiscono che la pupilla non sia compromessa a seguito del ritaglio dell'immagine. Infatti, assicurano che vengano scelti i punti che posizionerebbero la pupilla ad almeno 10 pixels da ogni bordo. In pratica con questa condizione si vanno ad evitare tutte quelle immagini artificiali in cui la pupilla viene parzialmente ritagliata;
- le condizioni **c7** e **c8** assicurano che l'immagine generata sia dimensioni minori uguali rispetto l'originale, in quanto non è possibile aggiungere immagini alla nuova immagine.

Successivamente, viene calcolato l'array *index* che conterrà gli indici dei punti del dataset che rispettano tutte e otto le condizioni.

A questo punto, si procede all'estrazione **pseudocasuale** del punto per effettuare data augmentation.

```
# If there are no points, raise an exception
if (len(index) == 0):
    raise ValueError('No suitable point found for data augmentation')

# Compute distance from desired point
d = (pup_x - dd[index, 2])**2 + (pup_y - dd[index, 3])**2

# and sort the indeces according to it
di = index[np.argsort(d)]

# Pick closest ~25% of points
i = di[:max(1, len(index) // 4)]

# Select from these the one that has been used least in the past
k = di[np.argmin(dd[di, -1])]

# Update its count
dd[k, -1] += 1

# and return it
return k
```

Figura 33 Parte del metodo getFromDataSet

Come controllo preliminare, se non ci sono punti che rispettano le condizioni viene sollevato un errore. A seguito di ciò si calcola la distanza tra il minimo della density function e i punti validi per il data augmentation, e si ordinano in base ad essa.

A questo punto si selezionerà il punto che è stato usato meno volte per il data augmentation, tra il 25% dei punti più vicini.

Infine, viene creata la nuova riga da aggiungere alla tabella *dataset* grazie al metodo `shiftViewportTo`:

```
def shiftViewportTo(dd, isz, pup_x, pup_y):
    """
    Shifts pupil location and viewport of an image
    dd:          A row with the data point and viewport
                (pupil_center_x, pupil_center_y, pupil_width, pupil_height, top_left_x, top_left_y, bottom_right_x, bottom_right_y)
    isz:         Original image size tuple (limits of the viewport)
    pup_x:       Desired x location for pupil
    pup_y:       Desired y location for pupil
    Note that the viewport might change size as a result of this operation
    Returns a new vector of the same size and format as the input
    It will raise an error if the image falls out of the viewport
    """
    dd = np.copy(dd)
    shift_x, shift_y = pup_x - dd[0], pup_y - dd[1]
    dd[0] = pup_x
    dd[1] = pup_y
    dd[4] -= shift_x
    dd[5] -= shift_y
    dd[6] -= shift_x
    dd[7] -= shift_y
    if (np.any(dd[2:4] < 0)):
        raise ValueError('Invalid shift')
    dd[6] = min(isz[0], dd[6])
    dd[7] = min(isz[1], dd[7])
    return dd
```

Figura 34 Metodo shiftViewportTo

3.4.5 Risultati del data augmentation

In questo paragrafo si riportano i risultati ottenuti a seguito del data augmentation, ottenuti con diversi valori di sigma (deviazione standard funzione gaussiana) e N (numero iterazioni).

a) risultati ottenuti usando una density function con sigma=10 e N=10000

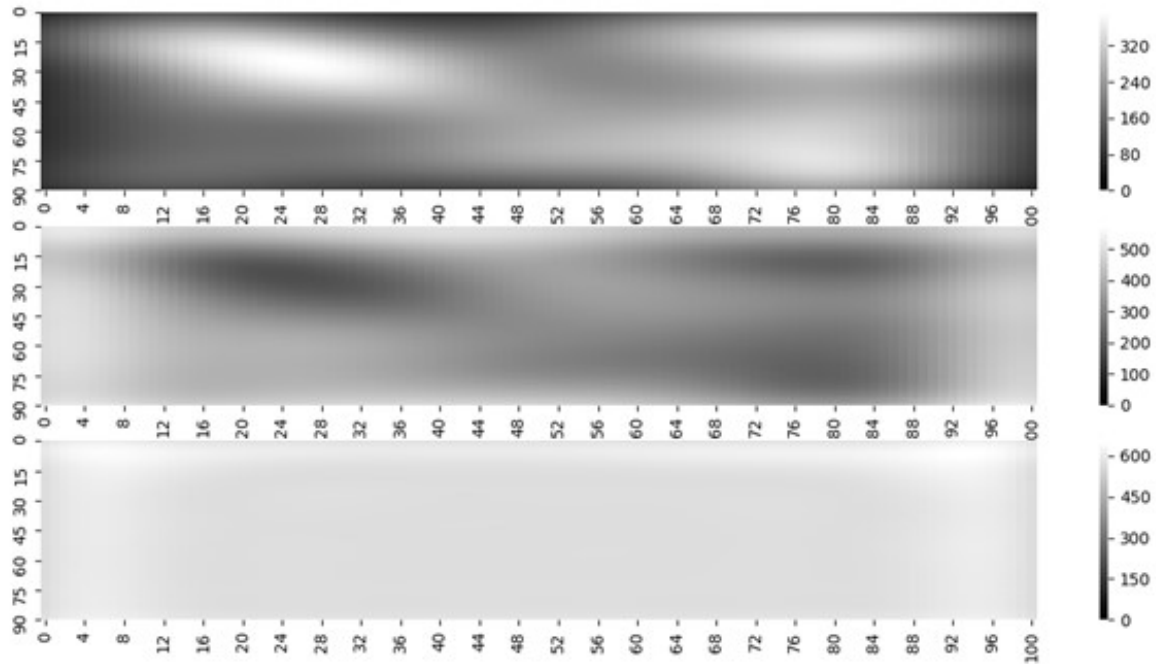
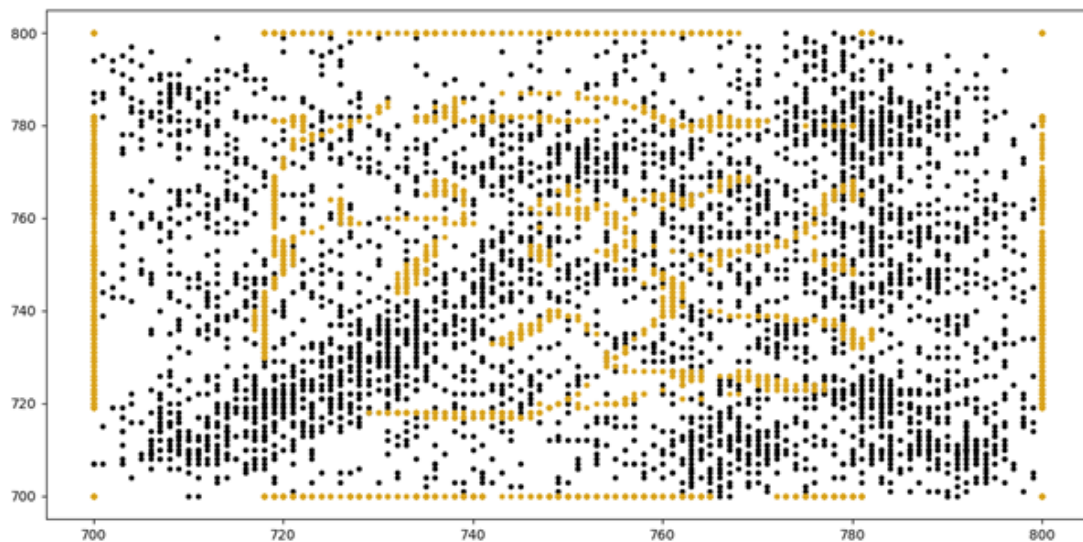


Figura 35 dall'alto verso il basso (sigma=10): density function del dataset originale; density function del dataset artificiale; density function del dataset aumentato

Figura 36 Scatter-plot del dataset aumentato: in nero i punti originali, in arancione i punti artificiali



Come si nota più evidentemente dalla figura 35, effettuando il data augmentation con un sigma pari a 10 non si ottiene il risultato sperato, in quanto creare una density function con una deviazione standard così alta fa aumentare la superficie delle zone più ricche di punti, e lasciando quindi zone vuote dopo il data augmentation.

Si è ripetuta quindi l'iterazione con un sigma pari a 5, in modo da ridurre appunto la superficie delle zone più concentrate di punti. Si ottengono sicuramente density function meno "smooth", ma l'uniformità dei punti nello scatter-plot (figura 37) inizia ad essere apprezzabile.

b) risultati ottenuti usando una density function con sigma=5 e N=10000

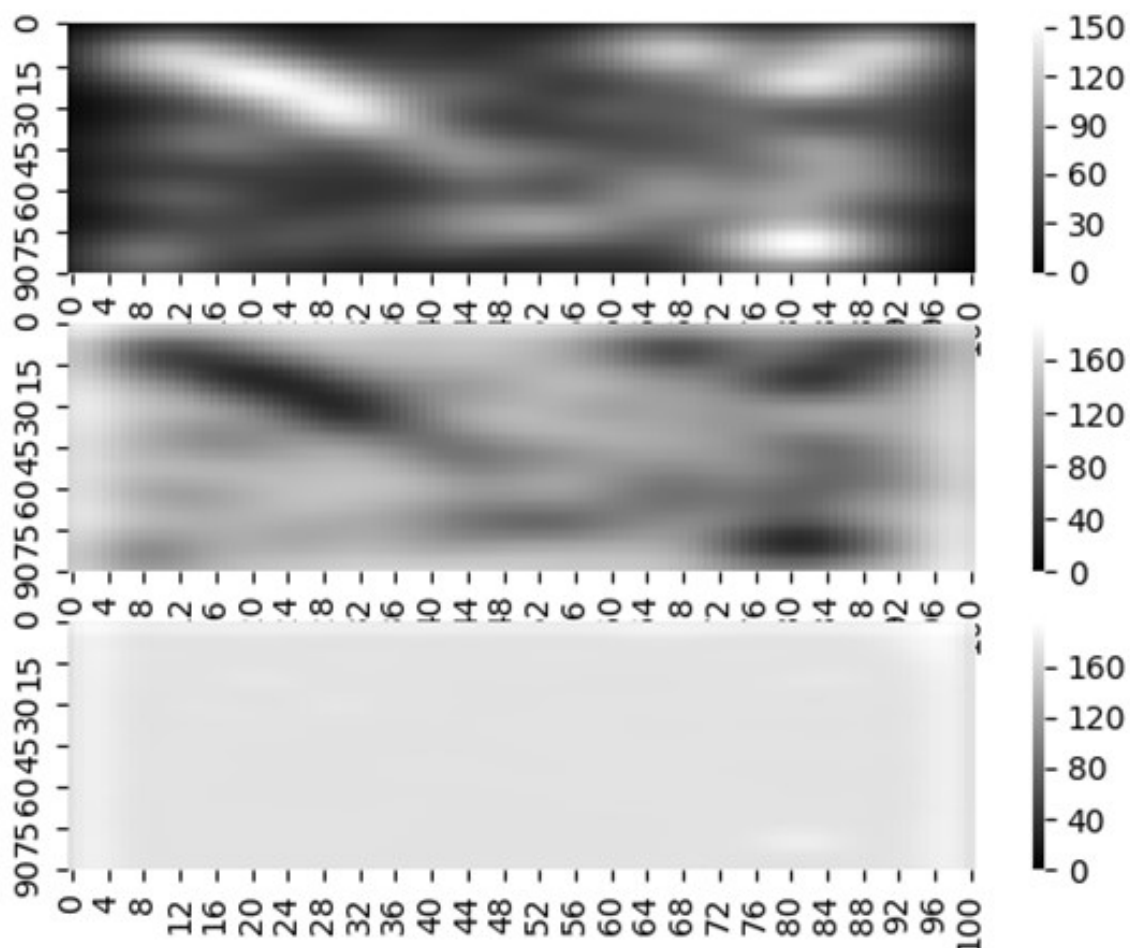


Figura 37 dall'alto verso il basso (sigma=5): density function del dataset originale; density function del dataset artificiale; density function del dataset aumentato

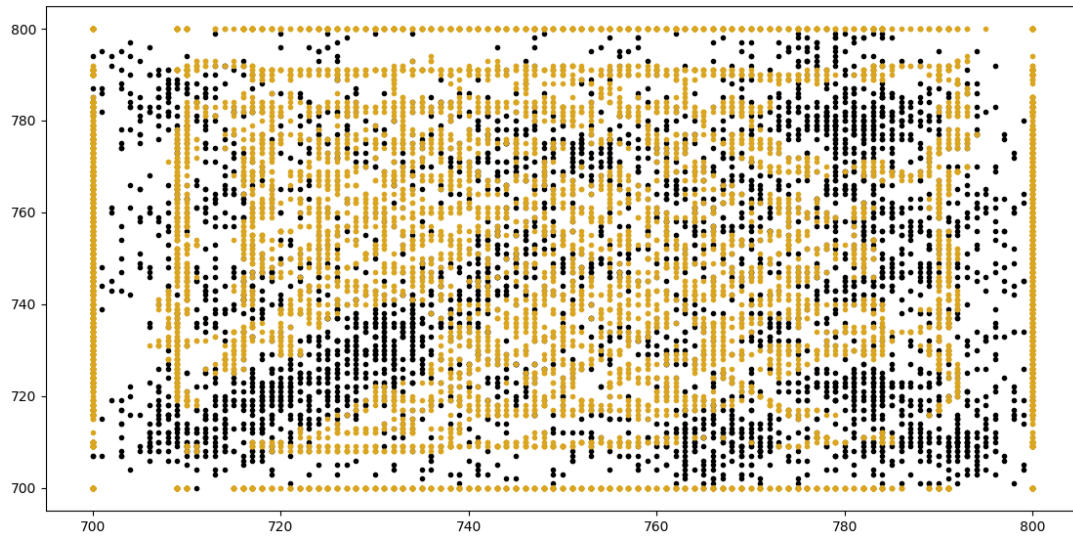


Figura 38 Scatter-plot del dataset aumentato: in nero i punti originali, in arancione i punti artificiali

c) risultati ottenuti usando una density function con $\sigma=1$ e $N=10000$

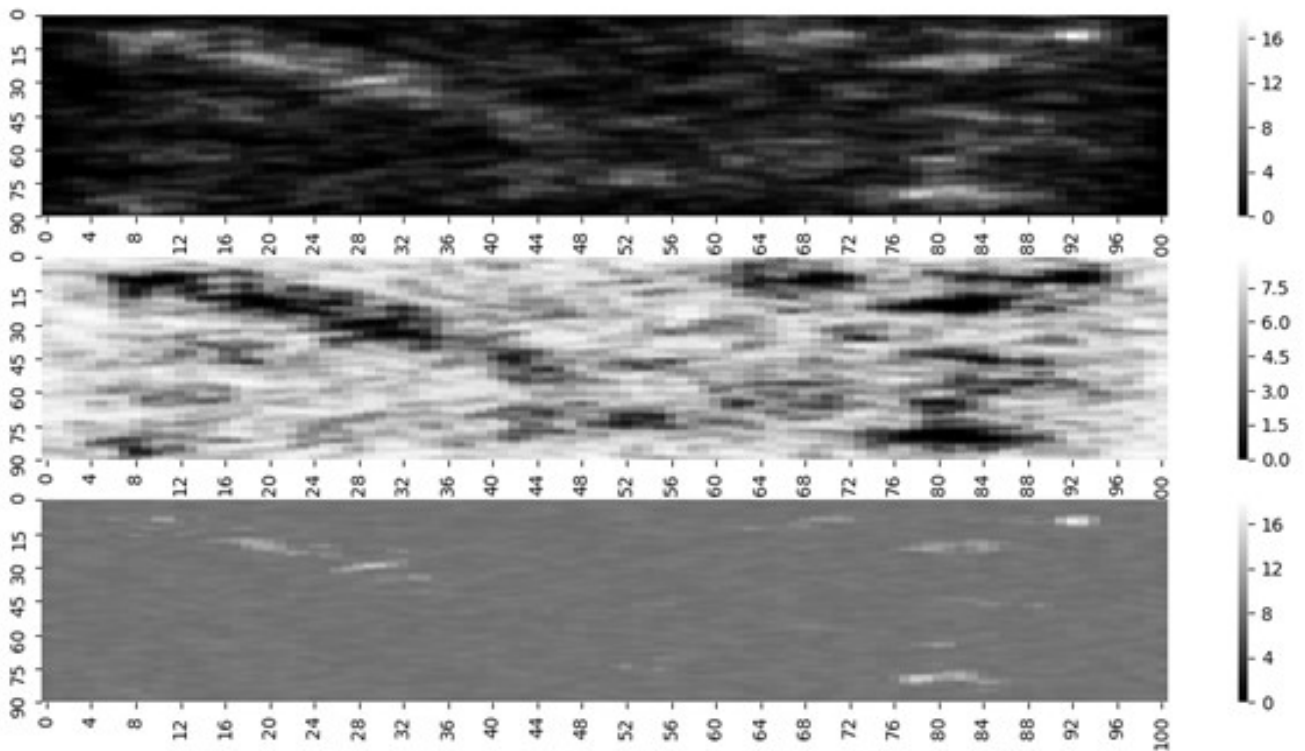


Figura 39 dall'alto verso il basso ($\sigma=5$): density function del dataset originale; density function del dataset artificiale; density function del dataset aumentato

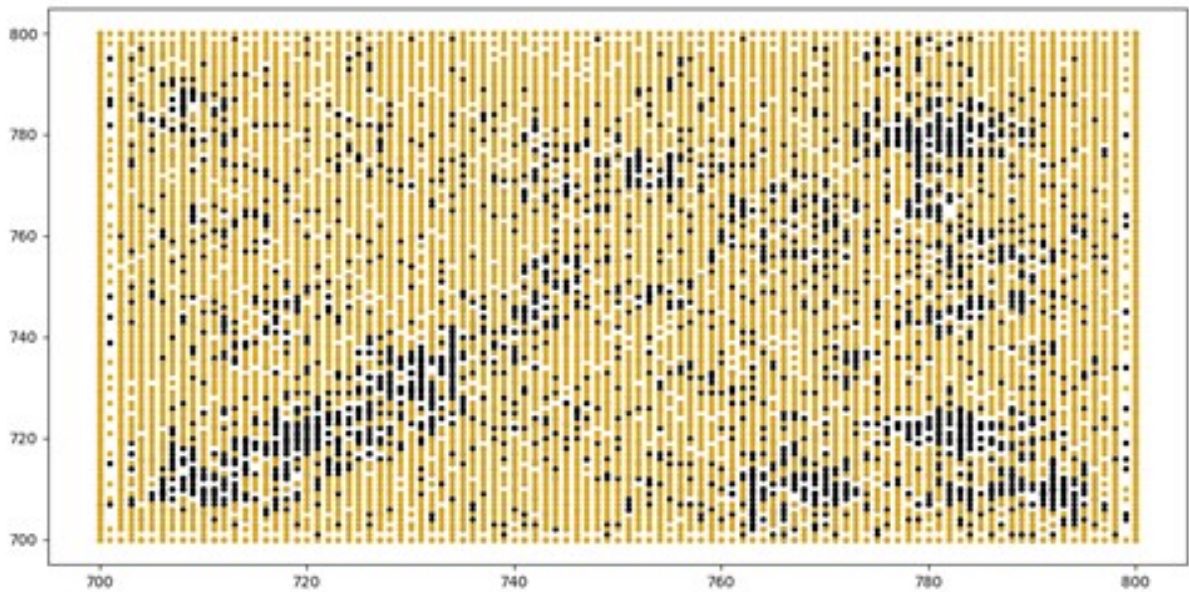
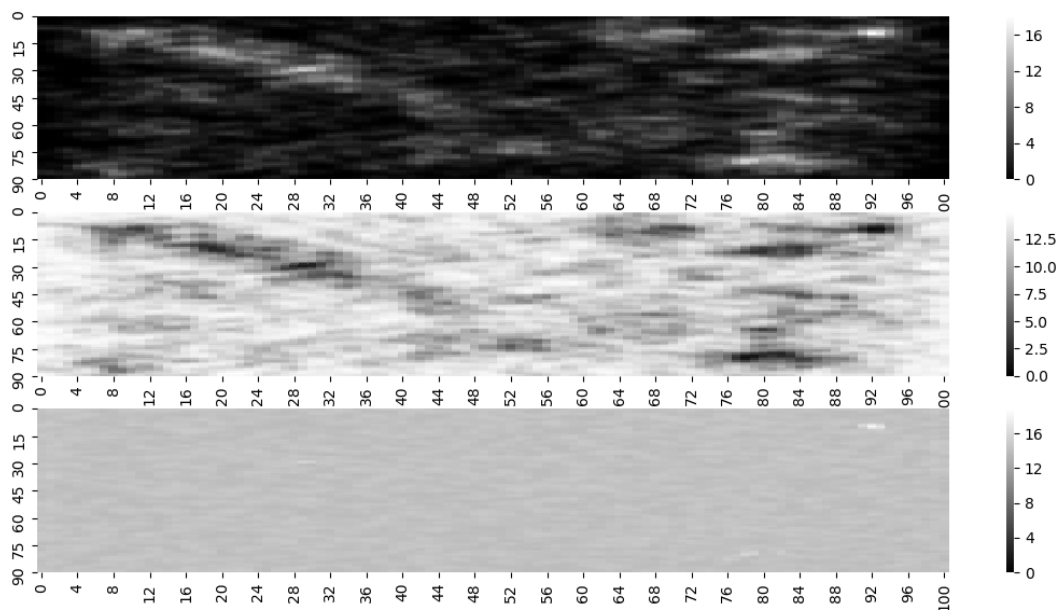


Figura 40 Scatter-plot del dataset aumentato: in nero i punti originali, in arancione i punti artificiali

Come emerge dalla figura 39, risultati soddisfacenti si ottengono con valori di sigma molto bassi. Inoltre, dalla figura 38.c emerge che ci sono ancora delle zone più intese rispetto che altre anche dopo aver effettuato data augmentation. Questo è dovuto ad un numero ridotto di iterazioni per il sigma scelto. Infatti, minore è il sigma scelto maggiore sarà il numero di iterazioni da applicare per ottenere una density function uniforme. Di seguito (figura 40) è riportato il grafico ottenuto con $N = 20000$.



La figura successiva è un istogramma che riporta quante volte ogni immagine è stata utilizzata per effettuare data augmentation.

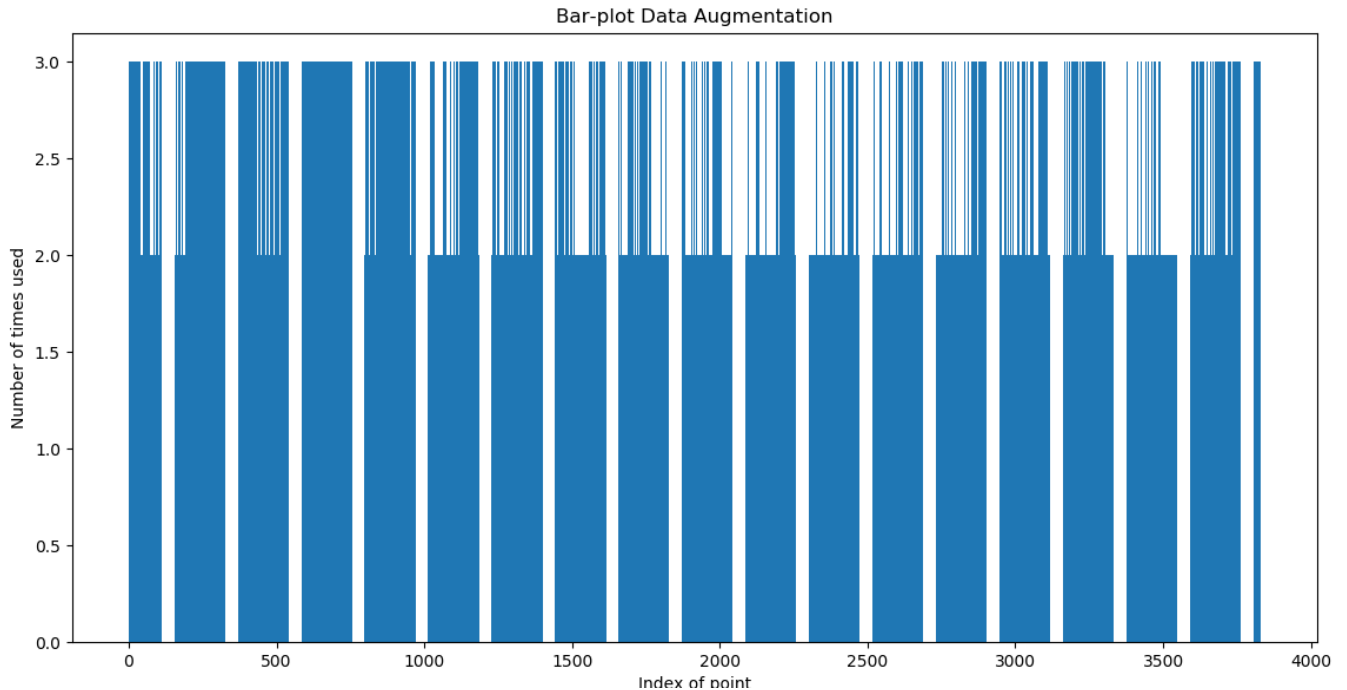


Figura 41 Istogramma che riporta quante volte ogni immagine è stata usata per il data augmentation

I dati raffigurati dall'istogramma sono molto positivi ed in linea con le aspettative, in quanto la selezione è stata molto uniforme lungo tutto il dataset, senza picchi.

3.4.6 Creazione delle immagini artificiali e dei labels

A questo punto, accertatosi che il viewport di tutto il dataset è valido e tutta la regione d'interesse è uniforme, si è pronti per la creazione delle immagini artificiali.

Per l'addestramento della rete, si è creato un dataset composto da immagini di 4 diverse dimensioni: 512x512, 256x256, 128x128, 64x64.

Inoltre, oltre ad aver anche adattato il valore delle coordinate alle rispettive dimensioni delle immagini, si è anche adottata la convenzione per i movimenti oculari, secondo la quale

l'origine degli assi è posto a centro immagine, sicché codesto si trovi in posizione (0,0); in tal modo gli assi risultano essere positivi se dal centro di va verso l'alto o verso destra.

```
def exportImages(bounds=(700, 800, 700, 800), size=512, downsamples=3):
    dd = lsr.store.loadAsciiArray(DATA_DIR + 'Training.dat')
    x0, y0 = (bounds[1] + bounds[0]) / 2, (bounds[3] + bounds[2]) / 2
    dx, dy = x0 - size // 2, y0 - size // 2
    dd[:, 6] += dx
    dd[:, 7] += dy
    dd[:, 8] = dd[:, 6] + size
    dd[:, 9] = dd[:, 7] + size
    # Change center of coordinates, and flip y, so that up is positive and down negative
    dd[:, 2] -= x0
    dd[:, 3] -= y0
    dd[:, 3] *= -1

    lsr.store.saveAsciiArray(dd[:, :6], DATA_DIR + 'Training_' + str(size) + '.dat')
    vv = np.zeros((dd.shape[0], size, size, 1), dtype=np.uint8)

    # Load images into vv array
    for subj, fn in enumerate(DATA_FILES):
        ifn = fnmatch.filter(os.listdir(IMAGES_DIR + fn), '*.png')
        for ff in ifn:
            j0 = ff.rfind('-')
            if (j0 >= 0):
                try:
                    img_num = int(ff[j0+1:-4])
                except:
                    continue
            img = lsr.util.loadImage(IMAGES_DIR + fn + os.path.sep + ff, ORIENT[subj]).astype(np.uint8)
            indeces = np.nonzero((dd[:, 0] == subj) & (dd[:, 1] == img_num))[0]
            for i in indeces:
                vv[i, :, :, 0] = img[int(dd[i, 7]):int(dd[i, 9]), int(dd[i, 6]): int(dd[i, 8])]
            indeces = np.nonzero((dd[:, 0] == (subj + 1000)) & (dd[:, 1] == img_num))[0]
            for i in indeces:
                vv[i, :, :, 0] = img[int(dd[i, 7]):int(dd[i, 9]), int(dd[i, 6]): int(dd[i, 8])]

    # Save images to disk
    np.save(DATA_DIR + 'Training_' + str(size) + '.npy', np.squeeze(vv))
```

Figura 42 Parte del metodo usato per la creazione delle immagini artificiali

La parte di codice riportata in fig. 42 esegue la creazione delle immagini artificiali in accordo su quanto riportato nel file Training.dat generato alla fine della fase di data augmentation e lo salva in un array.

```
for d in range(downsamples):
    size = size // 2
    dd[:, 2:6] /= 2
    lsr.store.saveAsciiArray(dd[:, :6], DATA_DIR + 'Training_' + str(size) + '.dat')
    vv = tf.nn.avg_pool2d(tf.convert_to_tensor(vv, dtype=tf.float32), ksize=(2, 2), strides=(2, 2), padding='VALID').numpy().astype(np.uint8)
    np.save(DATA_DIR + 'Training_' + str(size) + '.npy', np.squeeze(vv))
```

Figura 43 Parte del metodo usato per la creazione delle immagini artificiali

A questo punto si procede con il downsample delle immagini andando ad eseguire un average pooling delle immagini con passo (stride) pari a (2,2), come riportato in figura 43.

Capitolo 4 – Localizzazione della pupilla tramite reti neurali

La localizzazione delle coordinate del centro della pupilla tramite reti neurali è sicuramente un'area della ricerca in cui ci sono diversi studi in corso.

Un esempio è il lavoro svolto da alcuni ricercatori dell'Università di Tubinga: PupilNet [10].

PupilNet è costituito da una rete neurale di convoluzione costituita da due *pipeline*.

Nel primo stadio viene usata una CNN per effettuare una stima rapida della posizione della pupilla. Questa rapida stima serve per capire in quale sotto-regione dell'immagine la pupilla si trovi, in modo tale da poter selezionare solo quella e quindi ridurre rumore e il costo computazionale nel secondo stadio [10]. Quindi il primo passo è quello di ridimensionare l'immagine e dividerla in tante sotto-regioni. Per ogni sotto-regione, viene invocata la CNN ed la regione che avrà ottenuto il punteggio maggiore viene scelta per essere analizzata nel secondo stadio.

Nel secondo stadio, la stima fatta nel primo stadio viene perfezionata usando un'altra CNN. L'architettura di questa CNN prevede un layer di convoluzione 5x5, con stride unitario, senza padding [10]. Segue un layer di pooling di tipo "average" 4x4, con stride pari a 4 pixels. Infine, vi è un layer fully-connected con profondità unitaria [10]

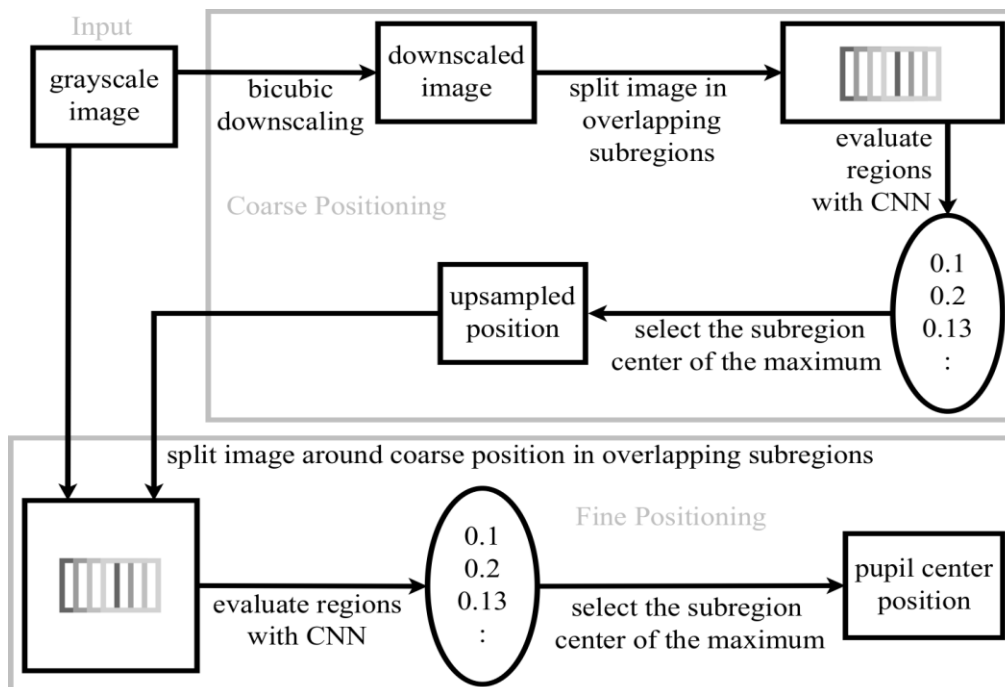


Figura 44 Schema rappresentate le due pipeline di PupilNet

4.1 Convolutional Neural Networks

Le reti neurali di convoluzione, chiamate anche CNN o ConvNet, rappresentano un particolare tipo di reti neurali in cui il pattern di connessione tra i neuroni si ispira alla struttura della corteccia visiva nel mondo animale⁴.

I singoli neuroni presenti in questa parte del cervello rispondono a determinati stimoli in una regione ristretta dell'osservazione, definita campo. La risposta di un singolo neurone a stimoli che hanno luogo nel suo campo recettivo può essere approssimata matematicamente da un'operazione di convoluzione.

Queste tipologie di reti neurali nascono principalmente per il riconoscimento di immagini, e rappresentano la struttura base della vera e propria visione artificiale. Il loro principale problema però è la scalabilità. Diventano più complesse tanto più la risoluzione è alta. Ad esempio, in un dataset come CIFAR-10 (dataset di numeri), le immagini hanno dimensioni pari a $32 \times 32 \times 3$. Ciò vuol dire che ogni neurone del livello di input avrebbe circa 3072 pesi da gestire. Mentre per un'immagine $250 \times 250 \times 3$ i pesi salirebbero a 196.608, comportando un aumento del costo computazionale.

4.1.1 Principali tipi di layers in una Convolutional Neural Network

Strato di convoluzione

È lo strato che caratterizza le reti neurali convoluzionali. I parametri consistono in un insieme di kernel (i cui valori variano durante l'apprendimento) di convoluzione.

È possibile modificare tre iper-parametri, tramite cui è possibile gestire/amplificare/ridurre l'effetto della convoluzione. Questi parametri sono: il numero di filtri da usare nella convoluzione, il passo del kernel (stride) e la presenza o assenza del padding degli input (che può essere di tipo 'valid' o 'same'). Il numero di filtri controlla il numero di neuroni nel livello che sono connessi alla stessa regione del volume di input. Ogni strato di neuroni imparerà a riconoscere un certo pattern in questa regione. Il passo controlla il numero di pixels lungo cui si sposterà il kernel sull'immagine durante la convoluzione (tanto maggiore sarà il pass, tanto più piccola sarà la matrice di output della convoluzione). Infine, il padding è un parametro che influenza le dimensioni spaziali del volume di output e agisce aggiungendo all'input un bordo di una certa dimensione contenente valori nulli. È

⁴ https://it.wikipedia.org/wiki/Rete_neurale_convolutionale

particolarmente utile quando si vuole fare in modo che le dimensioni spaziali dell'output e dell'input coincidano;

Strato di pooling

Lo strato di sotto-campionamento (pooling), è un altro tipico strato presente nelle CNN. Esistono diverse funzioni non lineari in grado di implementare il pooling e quelle maggiormente utilizzate sono il “*max pooling*” e “*average pooling*”. Questa tecnica divide l'immagine di input in tante piccole sotto-regioni, e per ciascuna delle regioni restituisce come output il valore massimo (nel caso del max pooling) oppure il valore medio (nel caso dell'average pooling). Ha due principali funzioni: il primo, è quello di ridurre le dimensioni delle immagini, in modo da far diminuire la complessità della rete e di conseguenza la sua complessità computazionale; il secondo scopo, è quello di prevenire situazioni come l'overfitting. L'overfitting è quella situazione in cui una rete viene eccessivamente addestrata, rendendo la rete troppo specifica e adatta solamente alle immagini che compongono il dataset di addestramento, facendo così ottenere score bassissimi alle immagini che non appartengono al dataset.

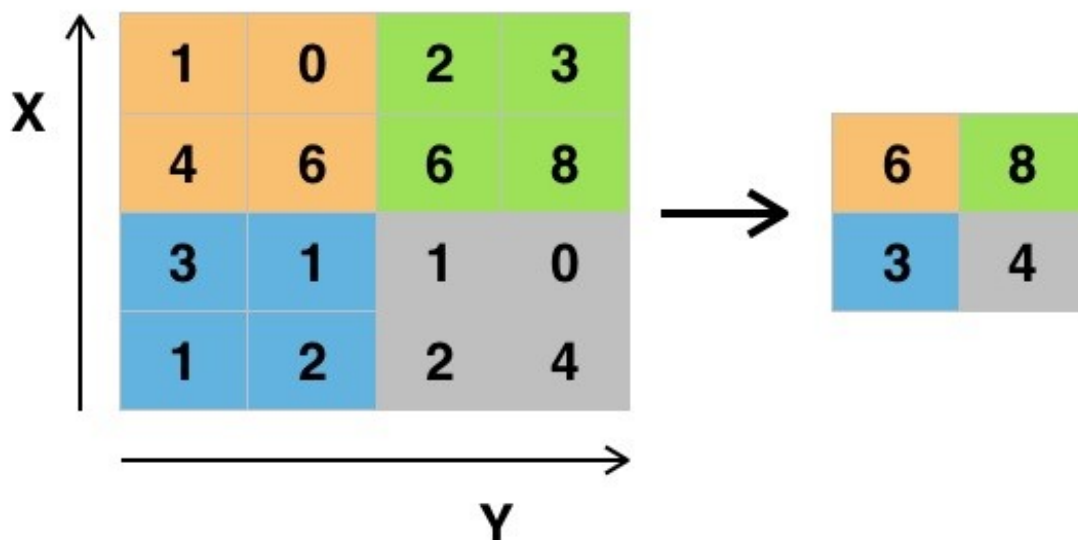


Figura 45 Esempio di pooling. Per ogni regione viene restituito il massimo valore

Nell'architettura tipica di una CNN vengono alternati ripetutamente livelli di convoluzione e livelli di pooling;

Strato completamente connesso (fully connected layer)

Lo strato completamente connesso rappresenta la parte finale di una CNN. I neuroni di un livello fully connected sono collegati a tutti i neuroni del livello precedente, come accade nelle reti neurali classiche, ed è in questo strato che viene calcolata e restituita la vera e propria predizione.

4.2 TensorFlow

TensorFlow è una libreria software open source per l'apprendimento automatico (*machine learning*), che fornisce moduli sperimentati e ottimizzati, utili nella realizzazione di algoritmi per diversi tipi di compiti percettivi e di comprensione del linguaggio⁵. Tensorflow è così flessibile da poter funzionare sia su più recenti computer che sugli smartphone dalla potenza di calcolo più ridotta.

TensorFlow permette di implementare una rete con poche righe di codice Python, il che lo rende molto versatile per questo tipo di applicazioni.

Ai fini di questo lavoro di tesi, per l'implementazione della rete neurale si è fatto uso del framework TensorFlow, versione 2.0.0.

4.3 Implementazione della rete ed architettura

Il primo passo è stato quello dello studio delle funzionalità base dei framework TensorFlow, Pandas e Keras attraverso dei tutorial offerti dal sito ufficiale.

Per gestire le immagini e i label, tensorflow richiede l'uso di un file .csv (Comma Separated Value) dove ogni riga della tabella contiene il percorso relativo al file dell'immagine e la rispettiva etichetta. In questo caso, la tabella avrà 5 colonne: la prima contiene il path dell'immagine, la seconda il valore della coordinata x, la terza il valore della coordinata y mentre quarta e quinta colonne la dimensione dell'immagine.

Per leggere e trarre i dati dal file .csv si è fatto uso della libreria pandas.

La grandezza del dataset dopo la fase di data augmentation è di 13.833 immagini.

Una volta creato questo file contenente tutte le informazioni necessarie riguardanti le immagini, si è passato alla creazione degli array che contenessero le immagini e i labels.

⁵ <https://it.wikipedia.org/wiki/TensorFlow>

Si è creato un array per le immagini e due per i labels (uno per la coordinata x ed uno per la coordinata y), successivamente salvato (e all'occorrenza caricato in memoria) per non dover ripetere inutilmente i calcoli ogni qual volta si proceda ad un addestramento.

Per quanto riguarda gli array delle immagini, ne sono stati creati diversi in base al downsize applicato alle stesse. Per le motivazioni elencate nel precedente paragrafo riguardante le CNN, è impensabile addestrare una rete con immagini che mediamente hanno una risoluzione pari a 1400x1400 in quanto il costo computazionale sarebbe troppo elevato.

Pertanto, dopo diverse prove, si è proceduto ad addestrare la rete con immagini a cui è stato applicato un downsize diverso: 64x64, 128x128 e 256x256.

Trattandosi di un problema di regressione, la scelta della funzione di perdita è ricaduta sulla **cosine similarity**. La similarità del coseno, o cosine similarity, è una tecnica euristica per la misurazione della similitudine tra due vettori effettuata calcolando il coseno tra di loro.

Come riportato nella documentazione ufficiale di tensorflow, questa funzione durante l'apprendimento assumerà valori compresi tra -1 e 0. Tanto più questo valore si avvicinerà a -1, tanto più grande sarà la similarità tra il valore vero e quello previsto.

```
tf.keras.losses.cosine_similarity(  
    y_true,  
    y_pred,  
    axis=-1  
)
```

Note that it is a negative quantity between -1 and 0, where 0 indicates orthogonality and values closer to -1 indicate greater similarity. This makes it usable as a loss function in a setting where you try to maximize the proximity between predictions and targets.

```
loss = -sum(y_true * y_pred)
```

Args:

- `y_true`: Tensor of true targets.
- `y_pred`: Tensor of predicted targets.
- `axis`: Axis along which to determine similarity.

Figura 46 Estratto della documentazione ufficiale di Tensorflow riguardo la cosine similarity.

Per adattare le labels delle immagini a questa funzione di perdita, è stato necessario fare operazioni. Deciso quale fosse il numero di neuroni in uscita della rete neurale, le labels da assegnare alle immagini durante l'addestramento sarà una matrice NxM (N numero di immagini, M il numero di neuroni) dove ogni riga conterrà la distribuzione gaussiana di ciascun punto su un array che conterrà appunto massimo M elementi.

```
def gaussWeights(train_labels, n_neurons, size, sigma):
    X = np.linspace(-size/10, size/10, n_neurons)
    labels = []

    for j in range(len(train_labels)):
        L = np.zeros(n_neurons)
        for i in range(n_neurons):
            L[i] = np.exp(-(X[i] - train_labels[j])**2/(2*sigma**2))
        labels.append(L)

    return np.asarray(labels)
```

Figura 47 Funzione usata per calcolare la distribuzione gaussiana delle coordinate lungo un array

Ottenendo un label per ogni immagine di questo tipo:

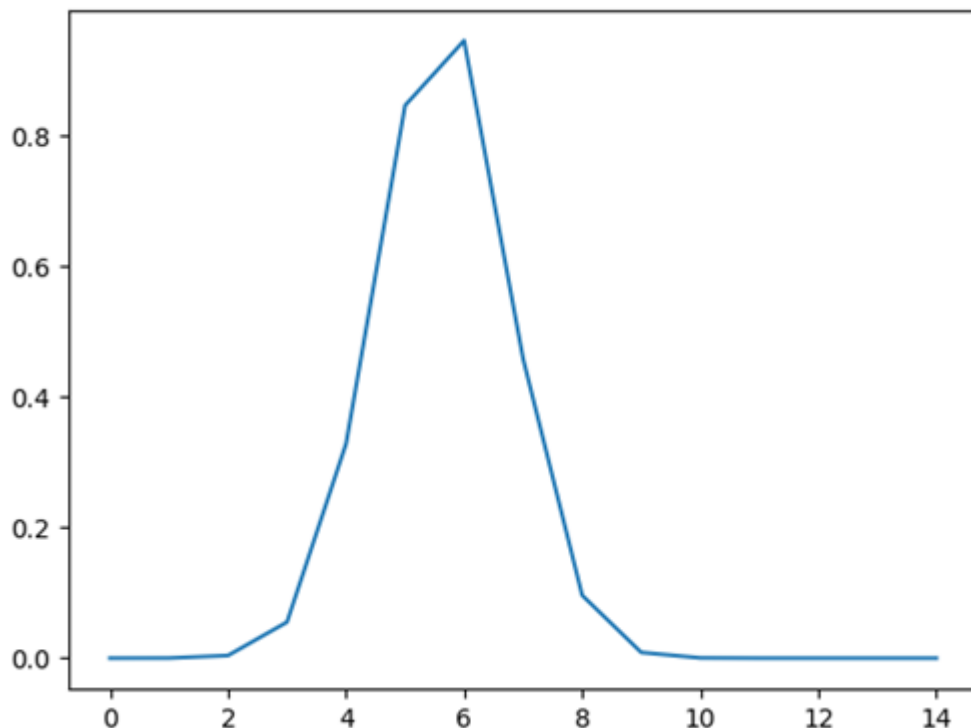


Figura 48 Plot di un label dopo aver applicato la distribuzione gaussiana

Parametro importante per la riuscita dell'addestramento è stato proprio il valore di sigma (deviazione standard) da usare per creare le labels. Dopo molte prove il valore usato per gli esperimenti è stato 2, che permetteva di ottenere i migliori risultati.

Una volta definita l'architettura di base, si è passati all'implementazione della rete tramite l'ausilio del framework tensorflow:

```
def createModel(image_shape = (size, size, 1), kernel_size = (3, 3)):
    """
    Create a model of Convolutional Neural Network, with 4 layers of convolution.

    image_shape (tuple) --> tuple containing the dimensions of the images. Tensorflow expect RGB images,
    so the tuple must be (n,m,3). ---Default value is (64, 64, 3)---
    kernel_size (tuple) --> kernel size for convolution. ---Default value is (3,3)---
    """

    model = models.Sequential()
    model.add(layers.Conv2D(n_filtri, kernel_size, activation = 'relu', padding='same', input_shape= image_shape))
    model.add(layers.MaxPool2D(2,2))
    model.add(layers.Flatten())
    model.add(layers.Dense(N, activation='softmax'))

    model.compile(optimizer='adam',
                  loss='cosine_similarity',
                  metrics=['cosine_similarity'])

    print(model.summary())
    return model
```

Figura 49 Metodo per la creazione del modello della CNN

Come possiamo vedere dalla figura 49, è molto semplice creare un modello di CNN con TensorFlow e Keras, dove basta una riga di codice per creare uno strato della rete.

- **Strato di convoluzione:** implementato tramite l'API keras, richiamando il metodo **layers.Conv2D**. Questo metodo prende diversi iper-parametri come argomenti: il primo è il numero di filtri di convoluzione da usare, che come è stato detto è ciò che determinerà lo spessore del tensore. Successivamente si decide la dimensione del kernel (es. 3x3), il tipo di padding e la dimensione dell'immagine che il modello si aspetta in entrata. Questo è un parametro da conoscere a priori, e tutte le immagini dovranno rispettare la dimensione definita nel modello altrimenti il metodo solleverà delle eccezioni. Altri parametri opzionali che si possono impostare sono, ad esempio, lo stride, che assume valore unitario se non viene definito dall'utente.
- **Strato di pooling:** implementato tramite l'API keras, richiamando il metodo

layers.MaxPooling2D. Serve per effettuare il sotto-campionamento dell'immagine tramite il processo descritto nel precedente paragrafo. Prende come argomenti la dimensione delle sotto regioni da considerare e il valore dello stride.

- **Strato fully connected:** impletato tramite l'API keras, richiamando il metodo **layers.Dense.** Con questo metodo viene definito il numero di neuroni di output e la funzione di attivazione. In questo particolare caso, è stata preferita la funzione **softmax** piuttosto che la canonica sigmoide, in quanto poco adatta alle convolutional neural network.

Per la predizione sono state implementate due reti, una per la coordinata x ed una per la coordinata y. Sono stati effettuati numerosi addestramenti, per le immagini di tutte le dimensioni (64x64, 128x128, 256x256). Nel successivo paragrafo si riportano solo i risultati ottenuti con le architetture che si sono dimostrate più adatte a questo tipo di problema.

4.4 Risultati degli esperimenti

Nei paragrafi che seguono verranno mostrati gli esperimenti condotti sulle immagini 64x64, 128x128 e 256x256.

Esperimento 1 (64x64)

Sono state implementate due reti costituite da:

- Blocco di convoluzione – numero filtri (64), kernel_size (3,3), padding (same)
- Blocco di pooling – average pooling (2x2)
- Blocco fully connected – 10 neuroni (uno per ogni punto della x e della y), funzione di attivazione softmax.

Il fitting delle reti si è articolato in **10 epoch.**

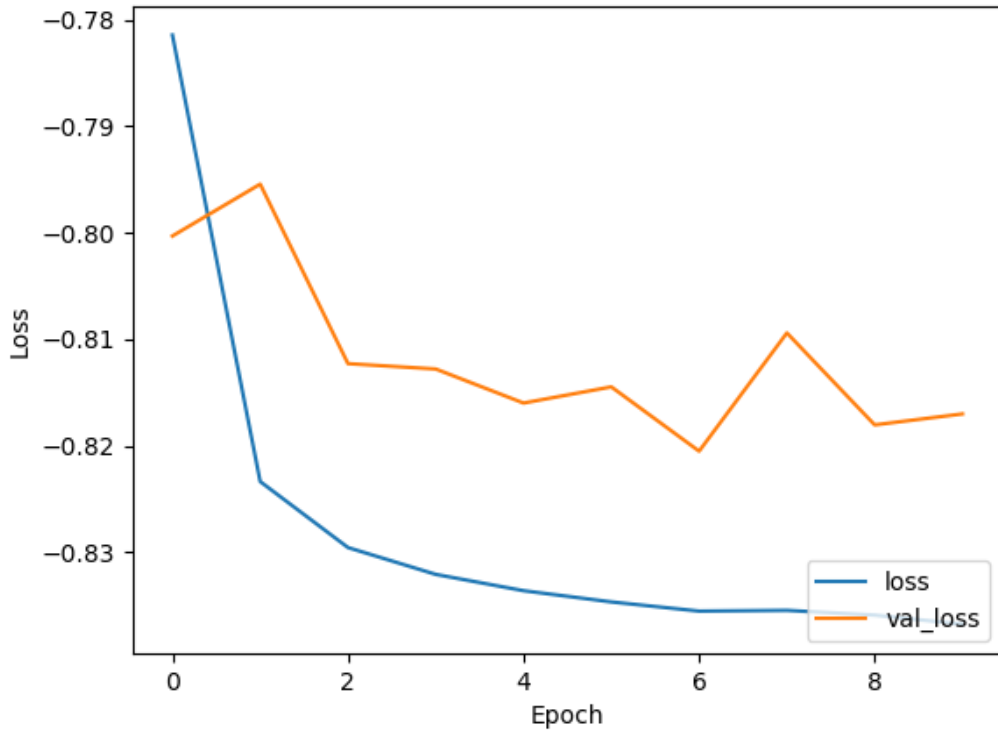


Figura 50 Grafico dell'andamento cosine/loss durante l'addestramento (rete x)

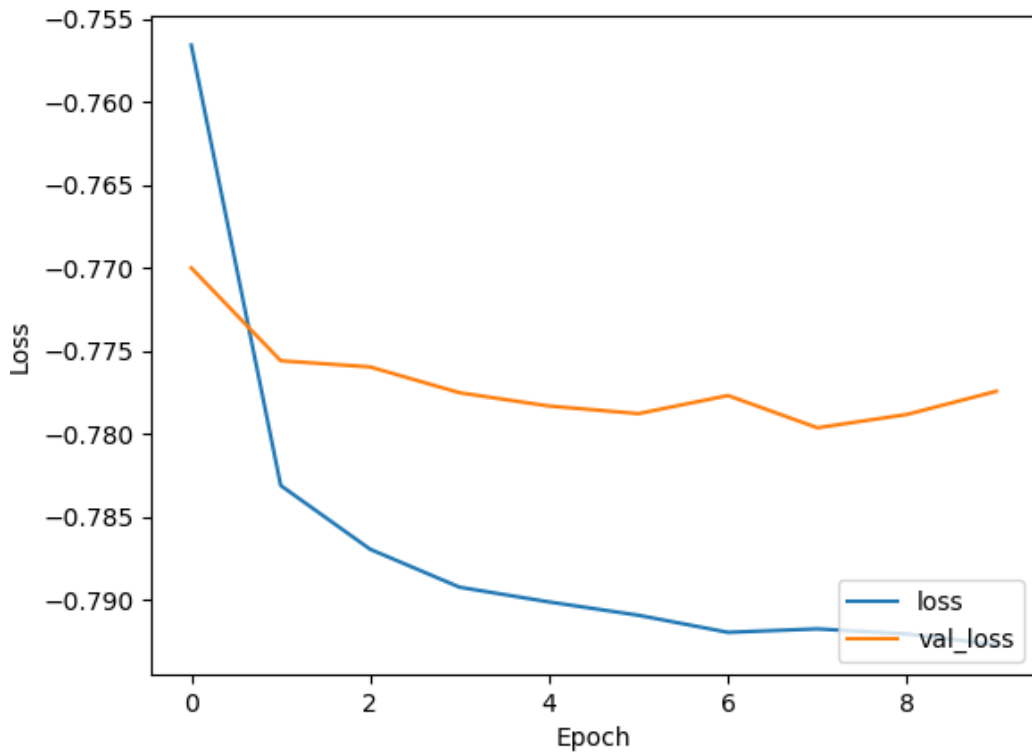


Figura 51 Grafico dell'andamento cosine similarity durante l'addestramento (rete y)

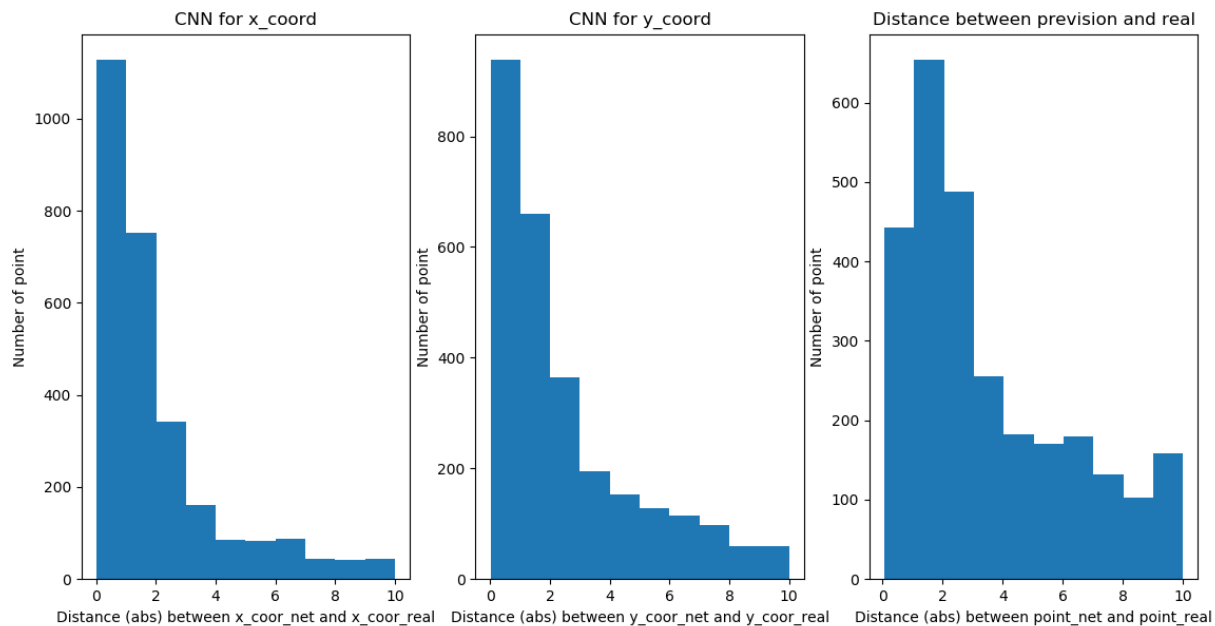


Figura 52 Istogramma che riporta gli errori (misurati in pixels) tra il punto previsto ed il punto reale

Il test della rete è stato effettuato con un test-set di 2767 immagini. In figura 51 è riportato l'istogrammi degli errori misurati in pixels della rete. Si considerano accettabili tutte le previsioni che rientrano in un errore massimo di 2 pixels. In questo caso, l'esperimento non è andato a buon fine in quanto si sono registrate molte previsioni con errore > 2 pixels.

Si sono fatte diverse prove con le immagini ridotte a 64x64, ma nessun esperimento si è ritenuto soddisfacente. Di seguito, è riportato un grafico dei punti che il modello non è riuscito a *fit*tare, facendo registrare un errore maggiore di 2 pixels.

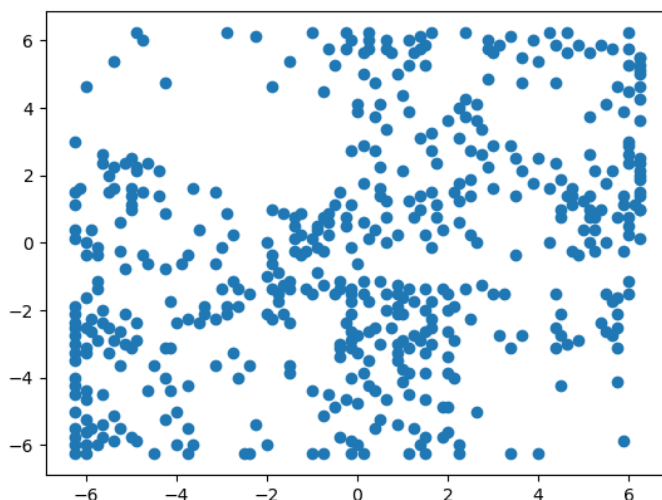


Figura 53 Scatter plot dei punti con errore > 2 pixels

Esperimento 2 (128x128)

Architettura delle reti implementate:

- Blocco di convoluzione – numero filtri (8), kernel_size (5,5), padding (same)
- Blocco di pooling – max pooling (2x2)
- Blocco fully connected – 15 neuroni (uno per ogni punto della x e della y),
funzione di attivazione softmax;

Il fitting delle reti si è articolato in **10 epoch**.

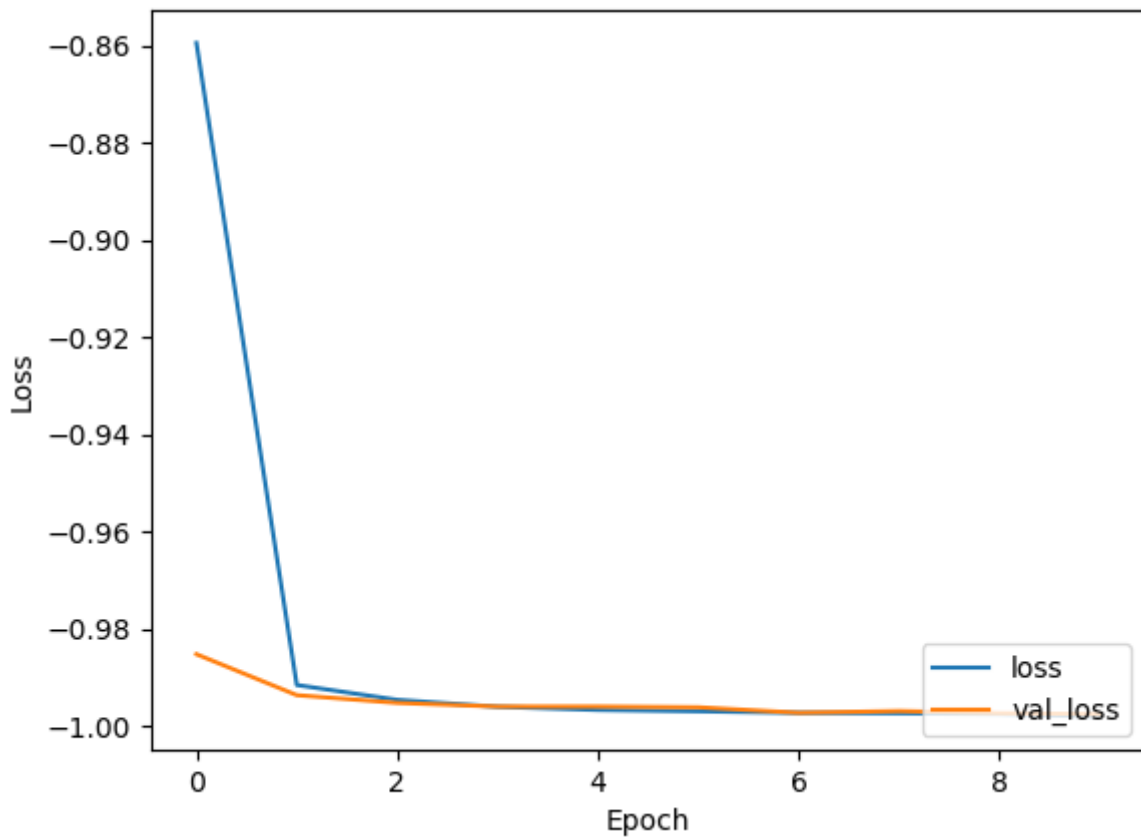


Figura 54 Grafico dell'andamento cosine similarity durante l'addestramento (rete x)

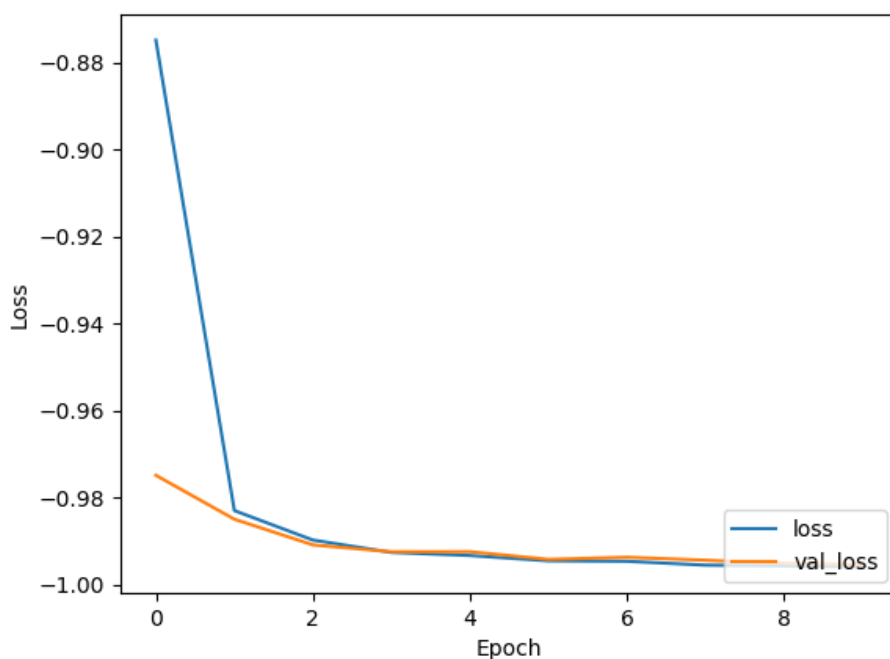


Figura 55 Grafico dell'andamento cosine similarity durante l'addestramento (rete y)

Di seguito è riportato l'istogramma degli errori, anche in questo caso il 98% delle previsioni si è dimostrato accettabile.

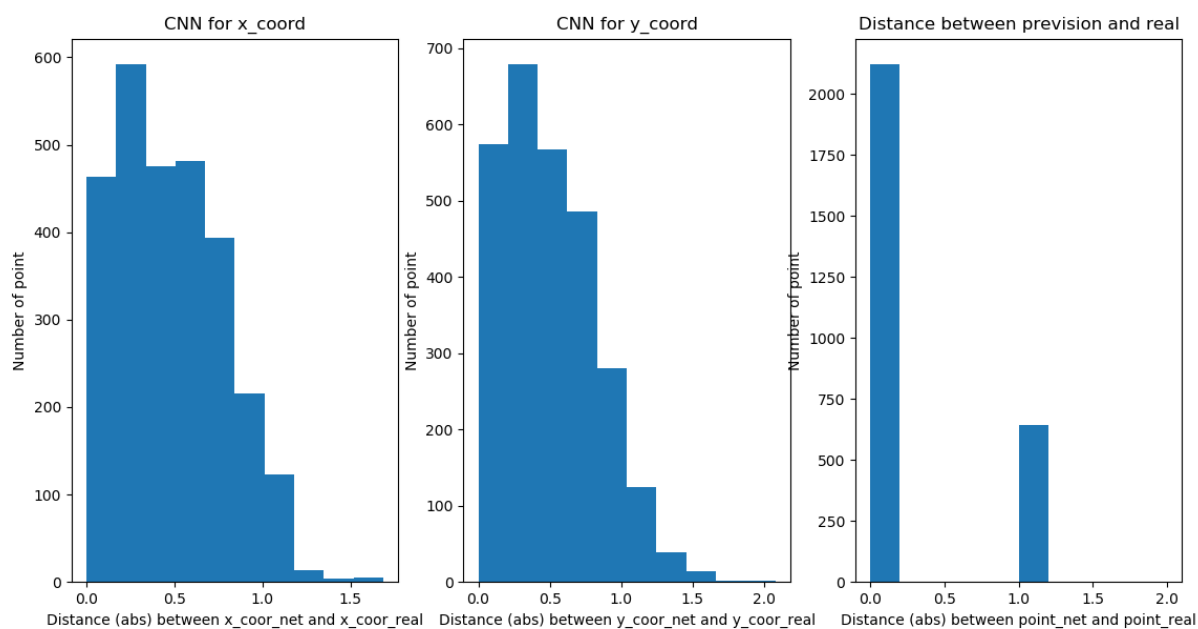


Figura 56 Iistogramma che riporta gli errori (misurati in pixels) tra il punto previsto ed il punto reale (esperimento 2). Read-out:top

In quest'esperimento i risultati sono molto positivi, in quanto non si sono registrate previsioni con errori > 2 pixels. Il read-out è stato fatto in due modi:

- prendendo il punto che statisticamente ha ricevuto un punteggio maggiore degli altri (figura 56);
- facendo una media ponderata tra quello che è l'array delle previsioni figura (57)

In entrambi i casi i modelli possono considerarsi accettabili.

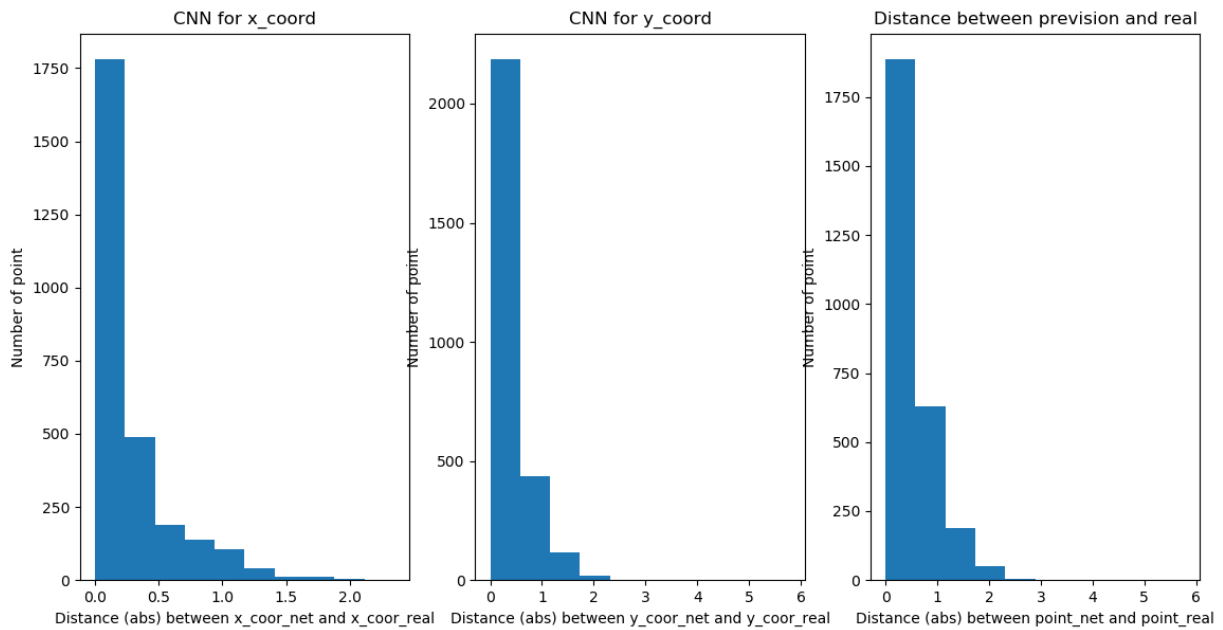


Figura 57 Istogramma che riporta gli errori (misurati in pixels) tra il punto previsto ed il punto reale (esperimento 2). Read-out: media ponderata

Esperimento 3 (128x128)

In questo esperimento, l'unica modifica che è stata fatta è stato aumentare il numero di neuroni che è passato da 15 a 20.

Architettura delle reti implementate:

- Blocco di convoluzione – numero filtri (8), kernel_size (5,5), padding (same)
- Blocco fully connected – 20 neuroni (uno per ogni punto della x e della y), funzione di attivazione softmax;

Il fitting delle reti si è articolato in **10 epoch**.

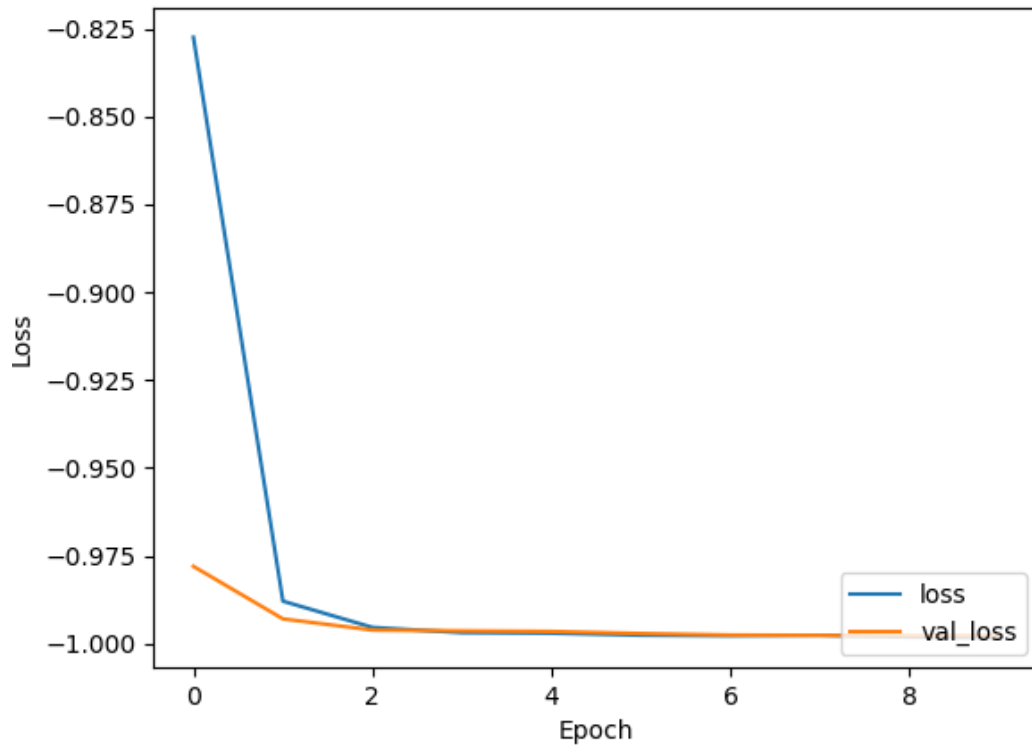


Figura 58 Grafico dell'andamento cosine similarity durante l'addestramento (rete x)

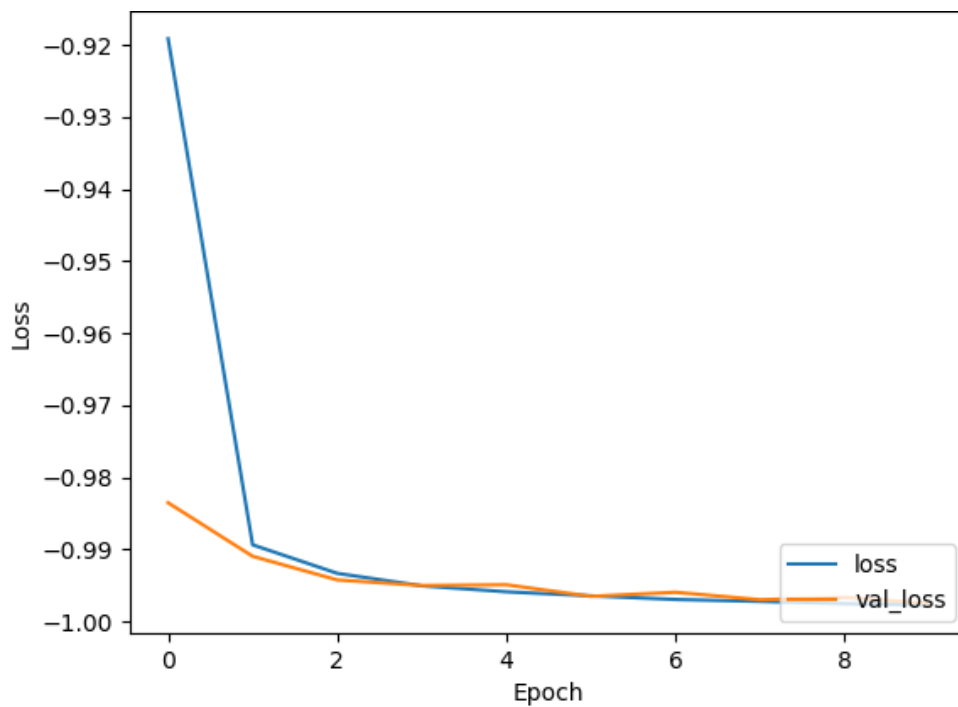


Figura 59 Grafico dell'andamento cosine similarity durante l'addestramento (rete y)

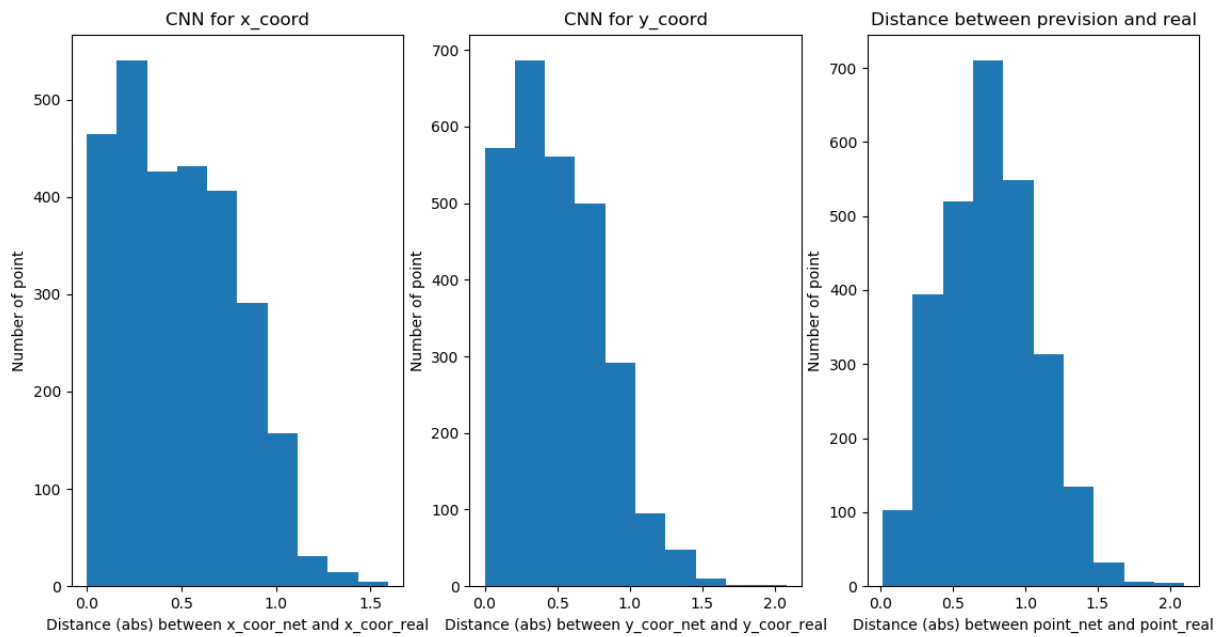


Figura 60 Istogramma che riporta gli errori (misurati in pixels) tra il punto previsto ed il punto reale (esperimento 3) Read-out:top

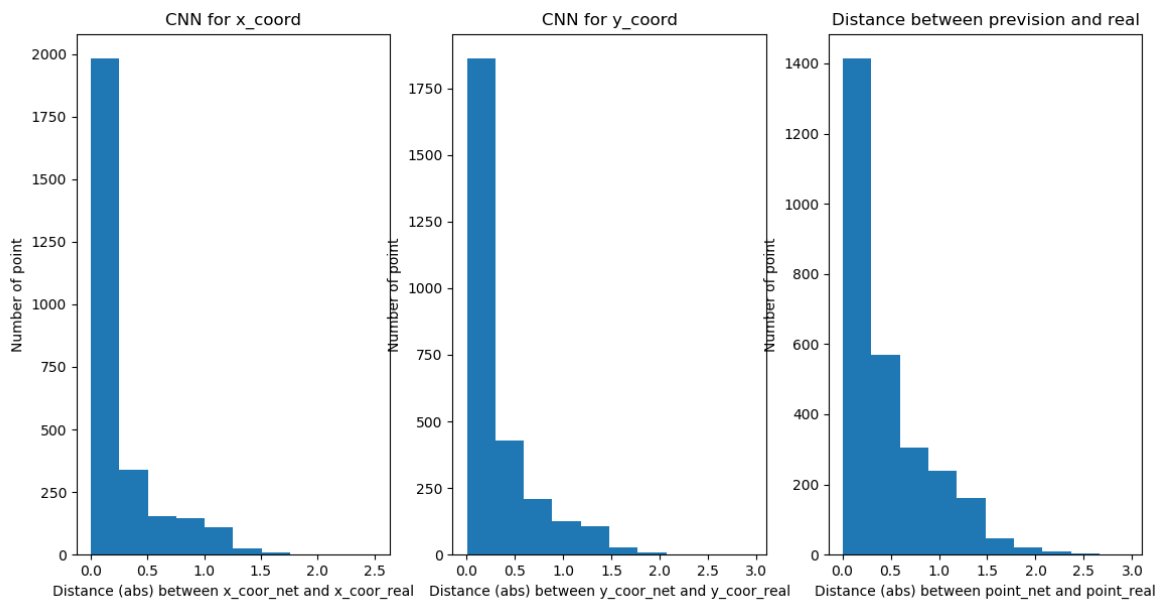


Figura 61 Istogramma che riporta gli errori (misurati in pixels) tra il punto previsto ed il punto reale (esperimento 3) Read-out: media ponderata

Esperimento 4 (256x256)

Architettura delle reti implementate:

- Blocco di convoluzione – numero filtri (16), kernel_size (7,7), padding (same)
- Blocco di pooling – max pooling (4x4)
- Blocco fully connected – 40 neuroni (uno per ogni punto della x e della y),
funzione di attivazione softmax;

Il fitting delle reti si è articolato in **10 epoch**.

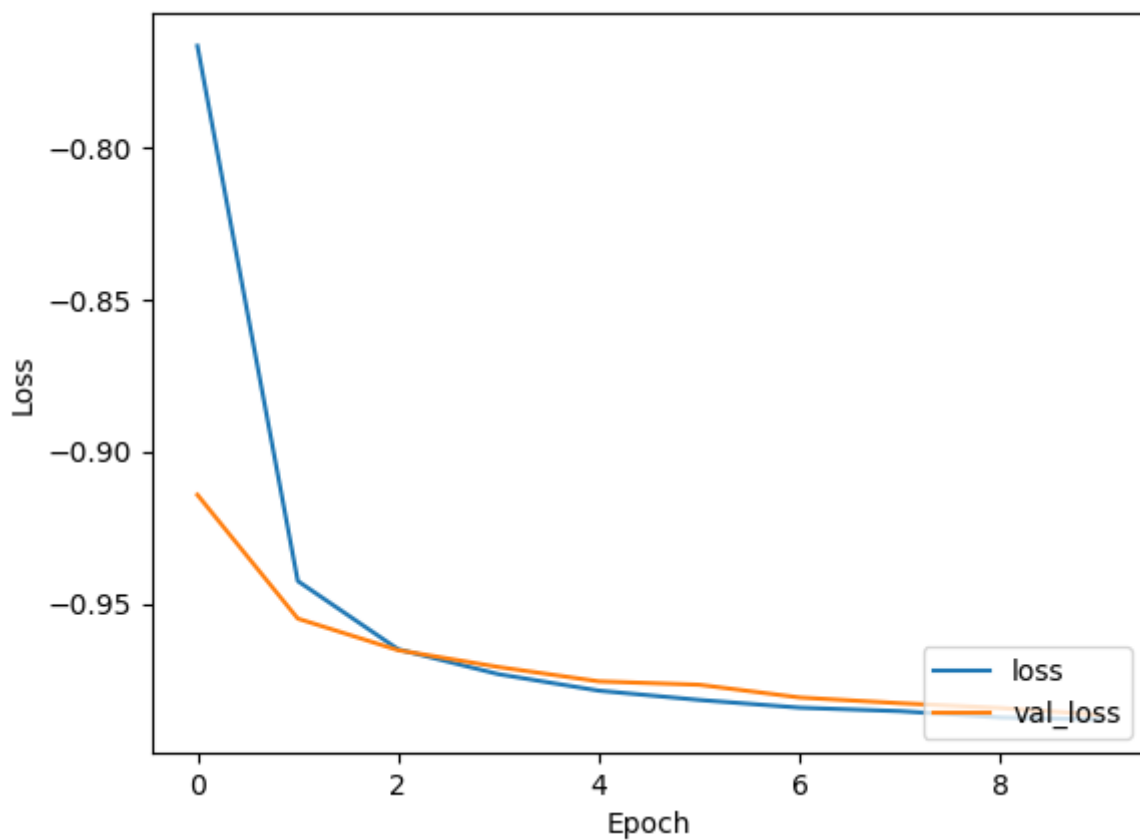


Figura 62 Grafico dell'andamento cosine similarity durante l'addestramento (rete x)

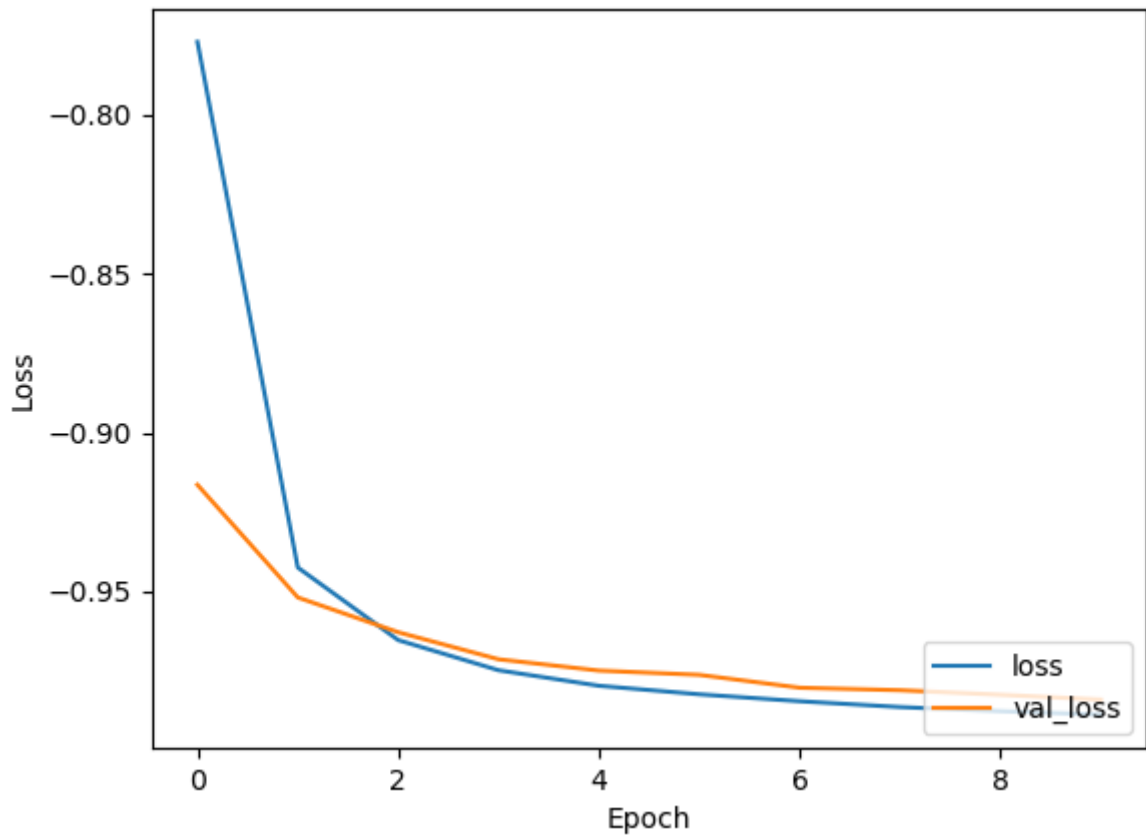


Figura 63 Grafico dell'andamento accuracy/loss durante l'addestramento (rete y)

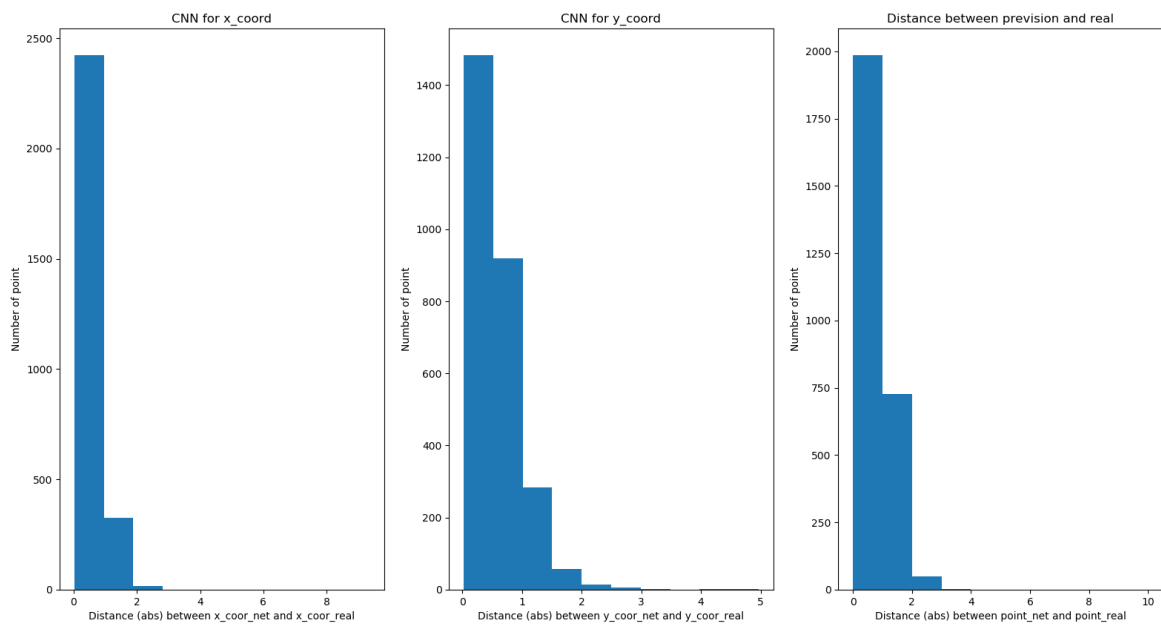


Figura 64 Istogramma che riporta gli errori (misurati in pixels) tra il punto previsto ed il punto reale (esperimento 4) Read-out: top

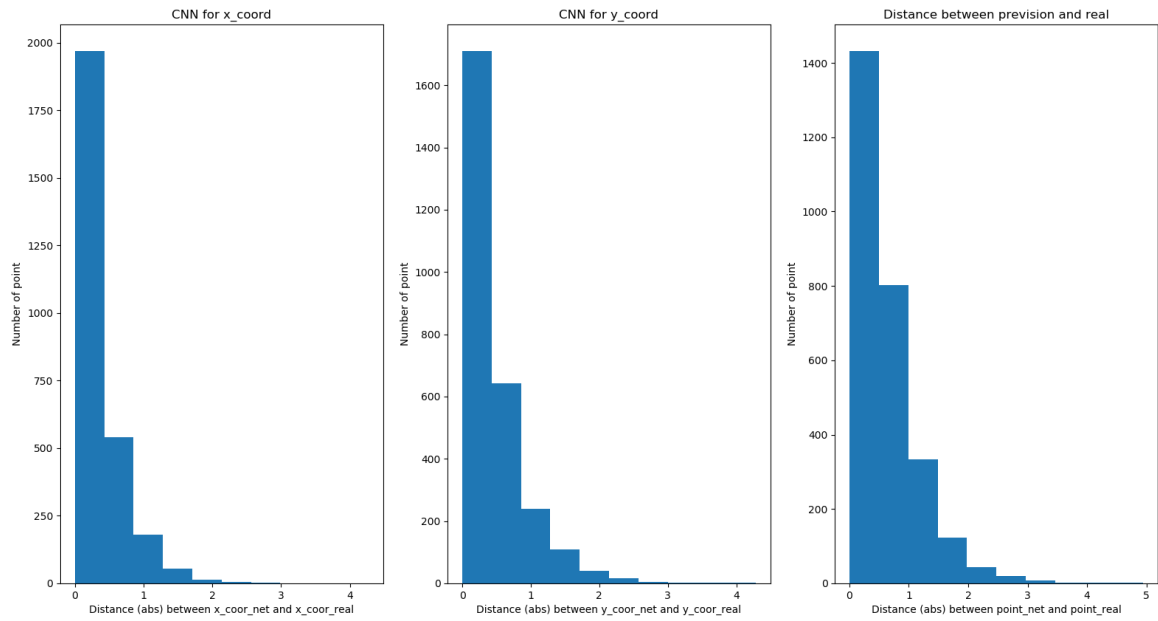


Figura 65 Istogramma che riporta gli errori (misurati in pixels) tra il punto previsto ed il punto reale (esperimento 4) Read-out: media ponderata

Anche i risultati ottenuti con le immagini 256x256 sono stati molto soddisfacenti, con oltre il 98% delle previsioni che non supera un errore di 1 pixel.

Capitolo 5 – Risultati e Conclusioni

Come emerge dai risultati degli esperimenti sui vari modelli di reti neurali, i migliori risultati sono stati raggiunti con le immagini di dimensioni 128x128 e 256x256. Inoltre, aumentando il numero di neuroni si stima che le previsioni diventino più accurate.

In particolare, il modello proposto nell'esperimento 3 prevede il 99% di punti entro il margine d'errore accettabile.

Di seguito sono riportati i risultati ottenuti con il modello proposto nell'esperimento 3, per immagini 128x128:

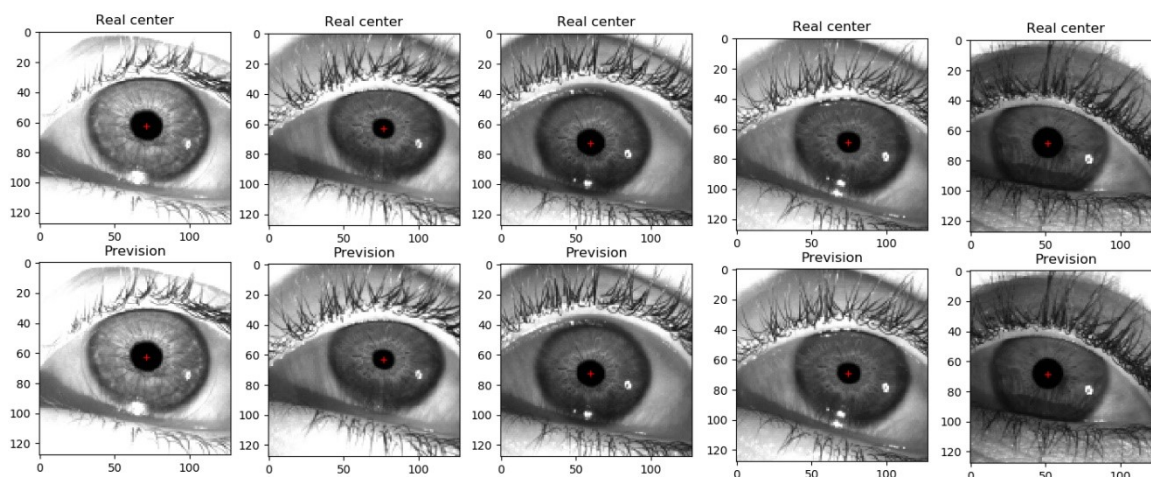


Figura 65 Plot di 5 coppie d'immagini 128x128. La prima riga contiene gli occhi con una croce rossa al centro che rappresenta il centro reale; la seconda riga contiene gli stessi occhi ma con la croce ottenuta dalla previsione delle reti neurali.

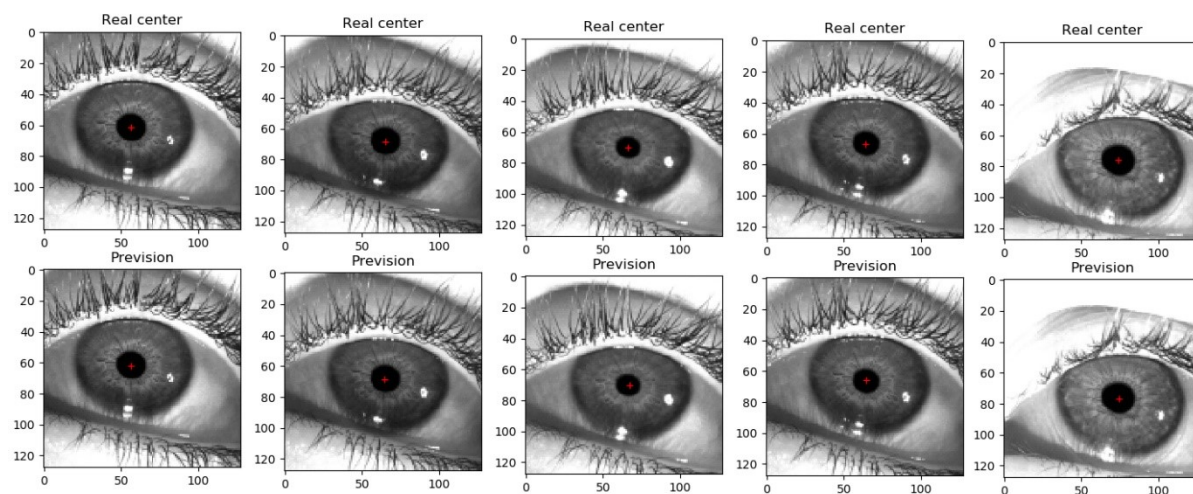


Figura 66 Plot di 5 coppie d'immagini 128x128. La prima riga contiene gli occhi con una croce rossa al centro che rappresenta il centro reale; la seconda riga contiene gli stessi occhi ma con la croce ottenuta dalla previsione delle reti neurali.

Di seguito sono riportati i risultati ottenuti con il modello proposto nell'esperimento 4, per immagini 256x256:

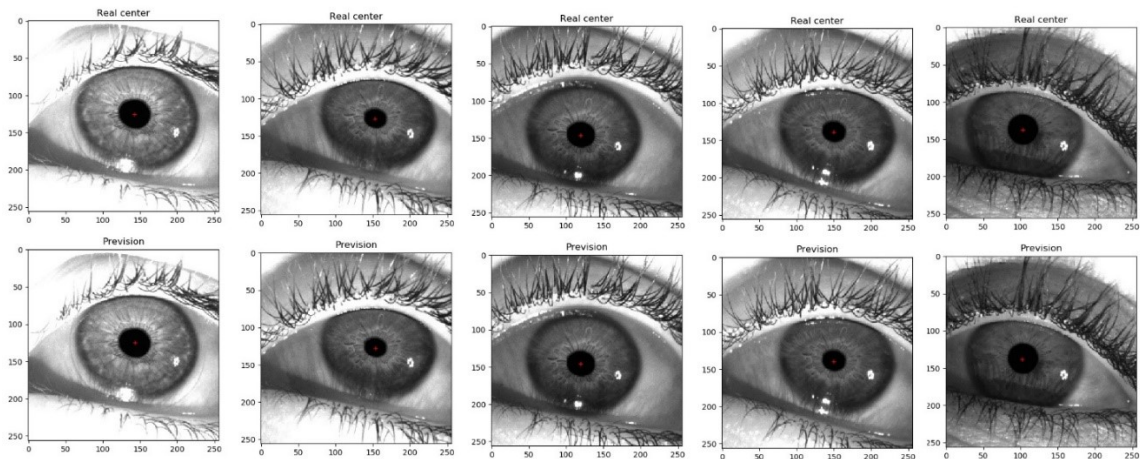


Figura 67 Plot di 5 coppie d'immagini 256x256. La prima riga contiene gli occhi con una croce rossa al centro che rappresenta il centro reale; la seconda riga contiene gli stessi occhi ma con la croce ottenuta dalla previsione delle reti neurali.

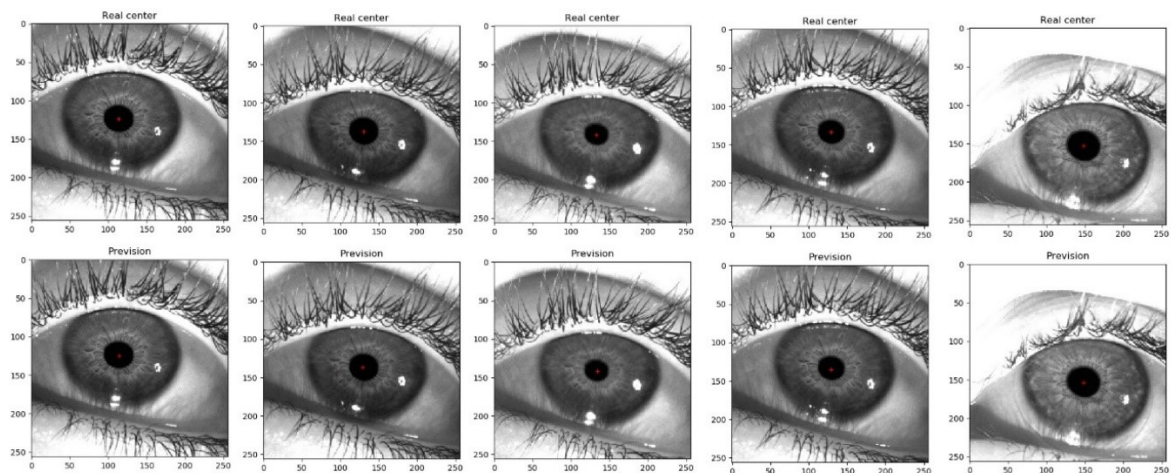
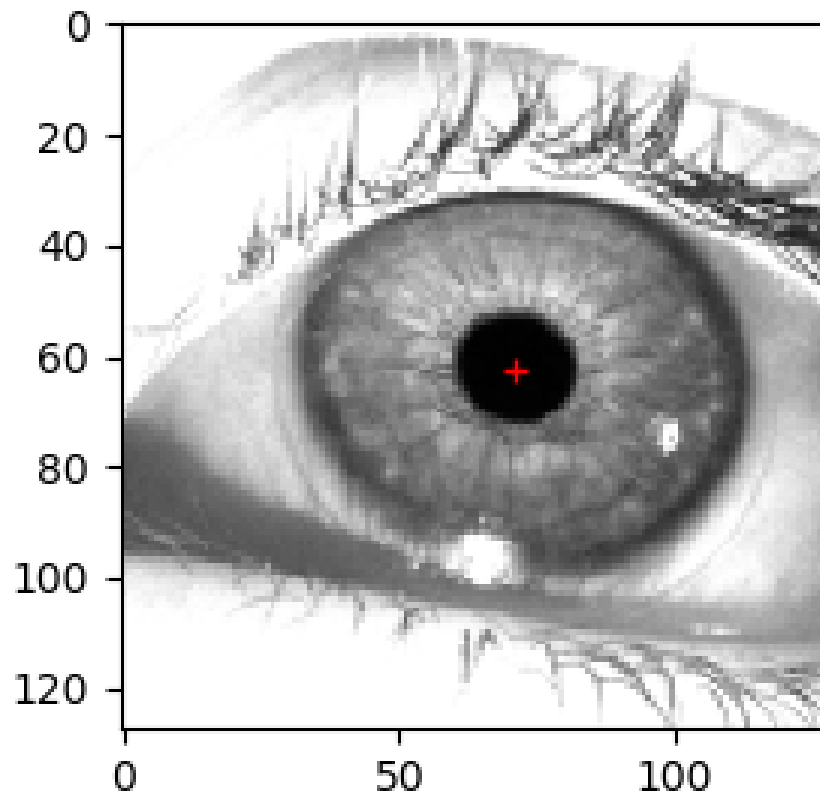
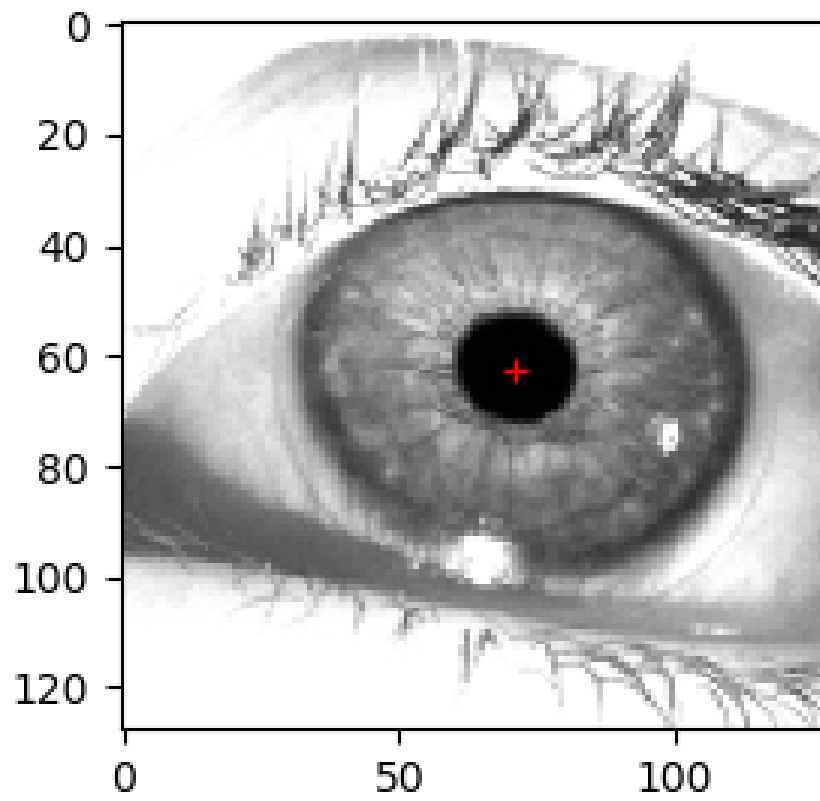


Figura 68 Plot di 5 coppie d'immagini 256x256. La prima riga contiene gli occhi con una croce rossa al centro che rappresenta il centro reale; la seconda riga contiene gli stessi occhi ma con la croce ottenuta dalla previsione delle reti neurali.

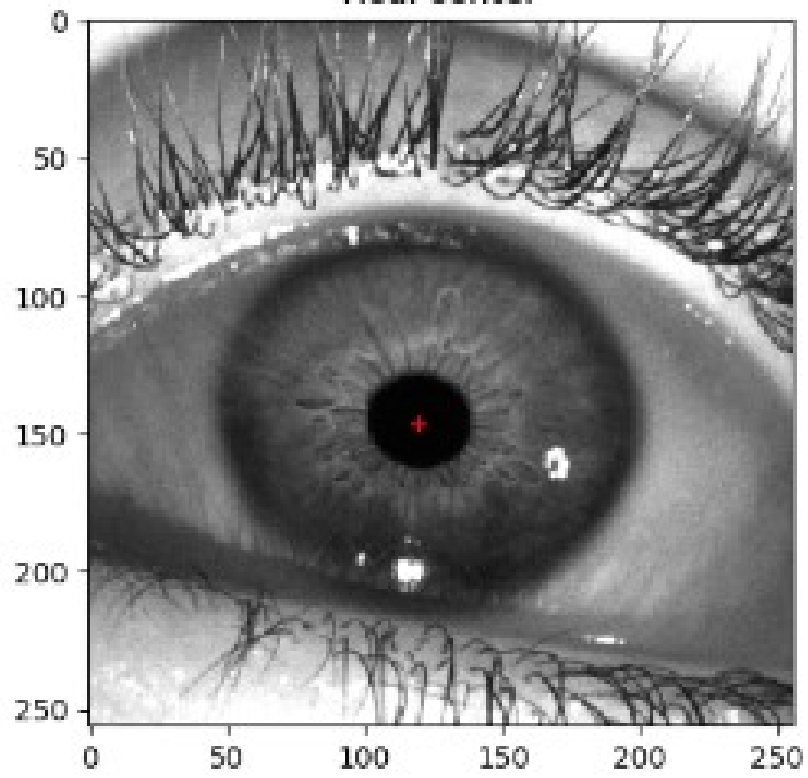
Real center



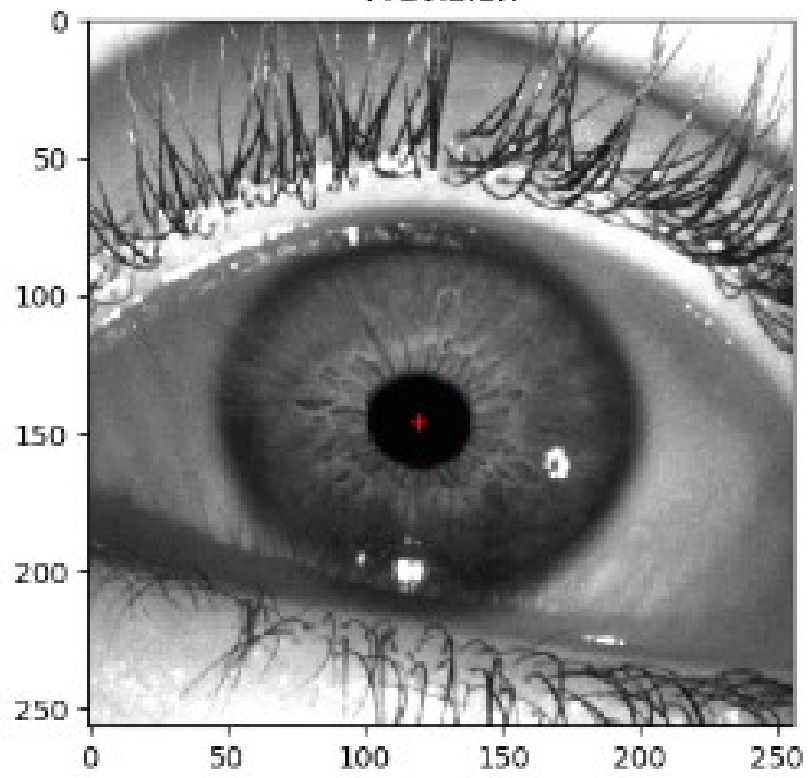
Prevision



Real center



Prevision



Le due precedenti figure sono state inserite a pagina intera per permettere di apprezzare meglio la croce che segna il centro della pupilla.

Mediamente, con tensorflow la previsione (calcolata attraverso la libreria tic-toc di Python) viene effettuata con i seguenti tempi:

- 2.62 ms per le immagini 128x128
- 6.37 ms per le immagini 256x256

Questi valori fanno sì che le specifiche di progetto poste all'inizio del progetto di tesi siano rispettate.

Bibliografia

- [1] Thomas Eggert. *Eye movement recordings: Methods*. Dev Ophthalmol. Basel, Karger, vol 40, pp 15-34, 2007.
- [2] David A. Robinson. *A Method of Measuring Eye Movement Using a Scleral Search Coil in a Magnetic Field*. The Johns Hopkins University, 1963.
- [3] Matthew J. Hollomon, Daniela Kratchounova, *Current Status of Gaze Control Research and Technology Literature Review*, Aerospace Medicine Technical Reports, 2017.
- [4] Wolfgang Jaschinski, *Pupil size affects measures of eye position in video eye tracking: implications for recording vergence accuracy*, Leibniz Research Centre for Working Environment and Human Factors, Technical University of Dortmund, Germany, 2016.
- [5] Jan Drewes, Weina Zhu, Yingzhou Hu, Xintian Hu, *Smaller Is Better: Drift in Gaze Measurements due to Pupil Dynamics*, Centre for Vision Research, York University, Toronto, Canada, 2014.
- [6] Dongheng Li, David Winfield, Derrick J. Parkhurst, *Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches*, Human Computer Interaction Program Iowa State University, Ames, Iowa, p. 2
- [7] Shinhiro Kawato, Nobuji Testutani, *Detection and Tracking of Eyes for Gaze-camera Control*, ATR Media Information Science Laboratories, 2002
- [8] Giuseppe Marcone, Giuseppe Martinelli, Lamberto Lancetti, *Eye Tracking in Image Sequences by Competitive Neural Networks*, University of Rome "La Sapienza", V. Roma, Italy, 1998
- [9] Bo Pedersen, Michael Spivey: *Offline tracking of eyes and more with a simple webcam*, (2006)
- [10] Wolfgang Fuhl, Gjergji Kasneci, Thiago Santini, Enkelejda Kasneci: *PupilNet: Convolutional Neural Networks for Robust Pupil Detection*, University of Tübingen, Germany, (2016)

Ringraziamenti

Ringrazio innanzitutto il Prof. Accardo per la disponibilità e per avermi dato la possibilità di lavorare nell'ambito dei movimenti oculari. È stata davvero un'opportunità unica, che mi ha permesso di imparare moltissime cose.

Ringrazio l'Ing. Christian Quaia per avermi seguito assiduamente durante questo lavoro di tesi, per avermi spronato a dare il meglio e per avermi sopportato durante questo percorso. Lavorare insieme a lui è stata un'opportunità di crescita unica, sia dal punto di vista professionale che dal punto di vista umano.

Ringrazio tutte le persone conosciute qui a Trieste che hanno reso questi anni speciali ed indimenticabili: Rosi, Bea, Stivella, Wallace, Richi e Sara.

Ringrazio tutti i soggetti che si sono prestati a fare le acquisizioni necessarie a portare avanti questo lavoro di tesi. In particolare, vorrei ringraziare i miei fedelissimi soggetti Rosita, Willis e Roberto per essersi sottoposti innumerevoli volte alle acquisizioni! Grazie per la pazienza!

Non posso non ringraziare i miei coinquilini Marco, Piergiorgio, Peppone, ma soprattutto il grandissimo Roberto.

Un pensiero speciale va ai miei amici di sempre: Gabriele, Peppe, Valerio, Ciccio, Bullé e Gionni. Nei momenti di sconforto pensare che qualsiasi cosa succedesse voi ci sarete sempre stati mi ha tirato sempre su.

Un ringraziamento particolare va fatto alla mia compagna di viaggio, Francesca. Grazie per tutte le risate, per la pazienza, per gli insegnamenti e per essere una delle poche persone che non ha mai smesso di credere fermamente in me. Alle persone come te sono enormemente ed infinitamente grato.

Infine, ringrazio i miei genitori. Vi sarò sempre debitore della vita che mi avete dato e dei sacrifici che fate costantemente. A voi va il grazie più grande.