

**ALMA MATER STUDIORUM - UNIVERSITÀ  
DI BOLOGNA**

---

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA

**PROGETTO E ATTIVITÀ PROGETTUALE IN SISTEMI  
DIGITALI M**

**EYE-GAZE TRACKER**

**Luigi di Nuzzo - Daniele Foschi - Filippo Veronesi**

---

Anno Accademico 2021/2022

Maggio 2022



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Rete Neurale</b>	<b>2</b>
2.1	Eye Detection . . . . .	3
2.1.1	Dataset . . . . .	3
2.1.2	Training . . . . .	3
2.1.3	TfLite . . . . .	5
2.1.4	Metadata . . . . .	5
2.2	Gaze Detection . . . . .	6
2.2.1	Dataset . . . . .	6
2.2.2	Training . . . . .	6
2.2.3	TfLite . . . . .	7
2.2.4	Metadata . . . . .	7
<b>3</b>	<b>Progetto Android</b>	<b>8</b>
3.1	Nerd Mode . . . . .	8
3.1.1	Bottone Analyze . . . . .	10
3.1.2	Riconoscimento pupilla . . . . .	11
3.2	Calibration . . . . .	12
3.3	Game . . . . .	13
<b>4</b>	<b>Conclusioni</b>	<b>14</b>

## Elenco delle figure

1	Eye Detection labelmap.pbtxt . . . . .	3
2	Tensorboard Learning_rate . . . . .	4
3	Eye Testing GUI TkAgg . . . . .	5
4	Gaze Detection labelmap.pbtxt . . . . .	6
5	Gaze Testing GUI TkAgg . . . . .	7
6	Homepage App . . . . .	8
7	Schermata Nerd mode . . . . .	9
8	Schermata Nerd mode con analyze attiva . . . . .	10
9	Schermata Nerd mode con pupilla . . . . .	11
10	Schermata Calibration . . . . .	12
11	Schermata Game . . . . .	13

## 1 Introduzione

Questo progetto prevede la realizzazione di un applicativo Android che, sfruttando due reti neurali, sia in grado di riconoscere gli occhi e le pupille dell'utente. Il sistema sfrutterà due modelli di rete neurale addestrati in modo tale da essere in grado di riconoscere gli occhi e le pupille di uno o più utenti grazie ad un dataset di 10 mila immagini di visi umani ed ad un altro contenente 5 mila immagini di pupille. L'applicativo permetterà di svolgere un filtraggio "live", cioè apprendendo la camera frontale ed esterna dello smartphone e individuando real-time gli occhi e le pupille dell'utente.

Inoltre, è stato implementato un gioco composto da un Quiz di 4 domande aiutato da un tool di calibrazione. L'utente tramite il riconoscitore di occhi è in grado di rispondere ad una certa domanda spostando, tramite gli occhi e il cellulare, un puntatore verso la risposta giusta posta ai 4 angoli del dispositivo. In caso di risposta corretta, viene posta un'ulteriore domanda, altrimenti si viene avvertiti della domanda sbagliata tramite un alert.

Tale caso di studio è un classico esempio di applicazione di Machine Learning e il software farà ricorso a una rete neurale convoluzionale (CNN). Tale scelta è dovuta al fatto che una rete neurale rappresenta il modo più comodo e pratico per problemi di object detection, come quello di questa attività in cui vengono individuati gli occhi e le pupille.

L'applicativo, inoltre, è pensato per la piattaforma Android e quindi tale progetto pone attenzione anche all'uso di risorse in quanto dovrà funzionare su smartphone, ovvero dispositivi embedded.

## 2 Rete Neurale

L'obiettivo primario di questo task è produrre un modello di rete neurale addestrata in grado di riconoscere gli occhi di una o più persone. Successivamente, si è prodotto un modello di rete neurale addestrata in grado di riconoscere le pupille degli esseri umani. A tali scopi si è utilizzato un modello messo a disposizione dalla libreria *TensorFlow 2 Detection Model Zoo* ottimizzato per il training di modelli specifici per object detection di immagini. Visto l'ambito dei sistemi embedded nel quale il progetto complessivo si colloca si è optato per un modello SSD MobileNet<sup>1</sup>, una CNN pensata per dispositivi mobile. MobileNet è il primo modello di computer vision pensato per dispositivi embedded basato su TensorFlow. MobileNet è sufficientemente leggera e veloce da essere eseguita su smartphone senza consumo di risorse eccessivo mantenendo comunque una precisione adeguata.

---

<sup>1</sup>SSD MobileNet V2 FPNLite 640x640

## 2.1 Eye Detection

### 2.1.1 Dataset

Per il training del modello si è utilizzato un dataset pubblico rilasciato da Google<sup>2</sup> contenente 10 mila immagini di visi umani, dove ogni foto conteneva da uno a più visi umani.

Oltre alle immagini il dataset conteneva un utile file *Excel* che indicava le coordinate della posizione degli occhi delle persone all'interno dell'immagine.

Prima dell'addestramento della rete era però necessario convertire questi dati di input in file *\*.xml* in modo tale da poterli poi usare per generare i **TFRecord**, utili per TensorFlow. Per questo motivo è stato implementato uno script *Java* in grado di creare un file *.xml* per ogni foto del dataset estraendo le informazioni dal file *Excel* le informazioni geografiche della posizione degli occhi e il nome dell'oggetto da riconoscere. Inoltre, per la corretta generazione dei **TFRecord**, si è reso necessario compilare il file *labelmap.pbtxt* con al suo interno i nomi e gli id di nostri oggetti da riconoscere: nel nostro caso un solo item di id pari a 1 e con nome "Human eye".

Figura 1: Eye Detection labelmap.pbtxt

```
item {  
    id: 1  
    name: 'Human eye'  
}
```

### 2.1.2 Training

Si è quindi proceduto all'addestramento della rete tramite Python usando TensorFlow e le API di Keras. Si è reso necessario modificare il file *pipeline.config* adattandolo alle nostre esigenze:

- *numclasses* che corrisponde al numero di item da identificare;
- *path* vari da adattare al nostro workspace, tra cui il path per i checkpoint, per i record di input e per il labelmap.

---

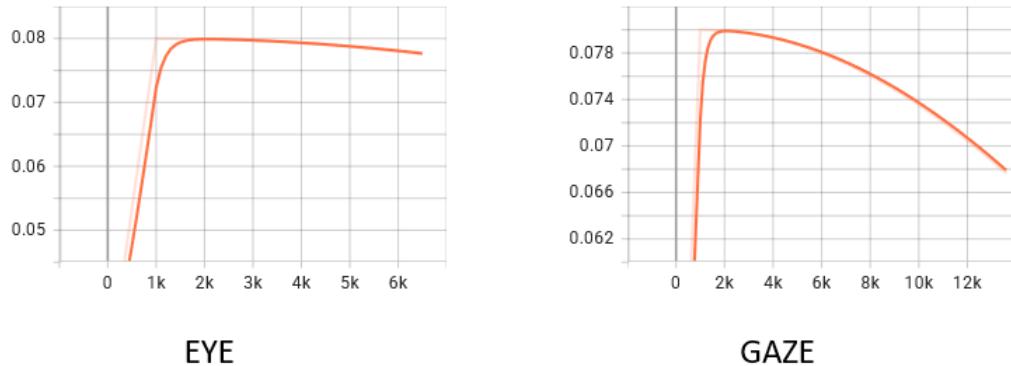
<sup>2</sup>Open Image V6, <https://storage.googleapis.com/openimages/web/index.html?v6>

Durante il procedimento di training si sono tenuti controllati i valori della nostra rete. In particolare, abbiamo usato **Tensorboard**: un framework grafico utile a capire l'andamento della nostra rete, infatti abbiamo avuto modo di controllare ad ogni training le performance della nostra rete di machine learning.

TensorBoard fornisce la visualizzazione e gli strumenti necessari per la sperimentazione del machine learning:

- Monitoraggio e visualizzazione di metriche come perdita e precisione;
- Visualizzazione del grafico del modello (operazioni e livelli);
- Visualizzazione degli istogrammi di pesi, distorsioni o altri tensori man mano che cambiano nel tempo.

Figura 2: Tensorboard Learning\_rate



Si è ottenuto in output un modello addestrato e pronto all'uso.

Prima di testare direttamente su Android, abbiamo deciso prima di testare e analizzare la nostra nuova rete utilizzando la libreria Matplotlib, utile per visualizzare graficamente via shell i nostri risultati. In particolare abbiamo usato **Tkinter**, l'unico framework GUI incluso nella libreria standard di Python.

A fini di testing si è preso ad esempio il volto di un personaggio pubblico, quello di Albert Einstein, e anche un'immagine contenente più persone in modo tale da poter verificare la correttezza e la precisione della rete anche in condizioni più difficili.

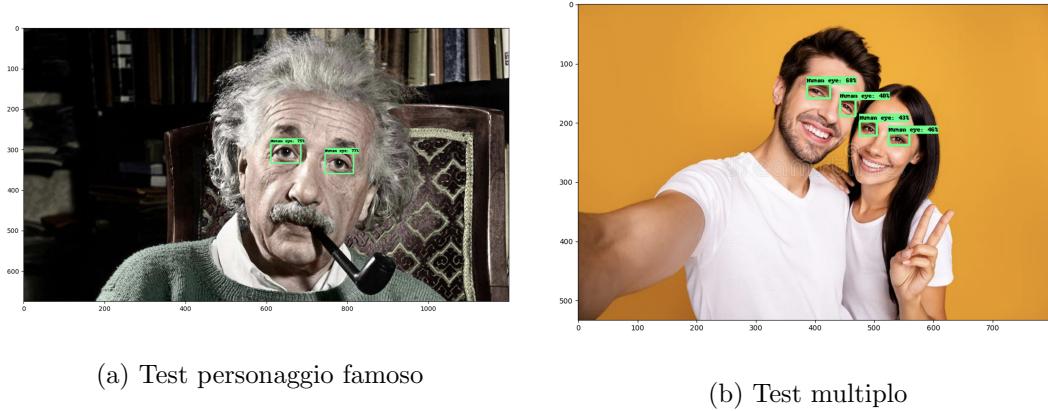


Figura 3: Eye Testing GUI TkAgg

### 2.1.3 TfLite

Dopo esserci accertati che la rete funzionasse correttamente e avesse dei livelli di precisione sopra una certa soglia, si è ottenuto in output un modello addestrato e pronto all’uso, che poi è stato convertito e quantizzato in un formato adatto ai sistemi embedded, quello di TensorFlow Lite.

### 2.1.4 Metadata

Per un corretto funzionamento della rete neurale all’interno di Android, TfLite richiede che venga generato anche un file Metadata che contiene le informazioni necessarie per pre-processare le immagini. Questo file si rende necessario in quanto i nostri tensori di input sono di tipo kTfLiteFloat32. È necessario creare un nuovo *label\_map.txt* con al suo interno nella prima riga il nostro item per la detection: nel nostro caso ”*Human eye*”.

Una volta generato questo nuovo file metadata.tflite siamo pronti ad importarlo all’interno della nostra applicazione Android come file di Machine Learning ”ml”.

## 2.2 Gaze Detection

### 2.2.1 Dataset

Per il training del modello si è utilizzato un dataset pubblico rilasciato da Kaggle<sup>3</sup> contenente 5 mila immagini di sole pupille umane, comodo per il raggiungimento del nostro obiettivo di gaze detection.

Questa volta, per ottenere i file *\*.xml* di ogni immagine, abbiamo usufruito di un tool grafico chiamato **LabelImage**, scaricabile gratuitamente su Github<sup>4</sup>. Questo tool ci permetteva di costruire i boxes attorno alla pupilla ottenendo in output per ogni immagine un file *\*.xml* contenente le informazioni necessarie: come il nome dell'oggetto su cui si è costruito attorno il box ("gaze"), e le relative coordinate geografiche dei 4 punti del box (xmin, ymin, xmax, ymax)

Prima dell'addestramento della rete era però necessario, usando questi nuovi dati *\*.xml*, generare i **TFRecord**, utili per TensorFlow. Inoltre, per la corretta generazione dei **TFRecord**, si è reso necessario compilare il file *labelmap.pbtxt* con al suo interno i nomi e gli id di nostri oggetti da riconoscere: nel nostro caso un solo item di id pari a 1 e con nome "gaze".

Figura 4: Gaze Detection labelmap.pbtxt

```
item {  
    id: 1  
    name: 'gaze'  
}
```

### 2.2.2 Training

Per quanto riguarda il Training, abbiamo effettuato le stesse scelte effettuate per l'eye detection.

Prima di testare direttamente su Android, anche in questo caso abbiamo deciso prima di testare e analizzare la nostra nuova rete utilizzando la libreria Matplotlib e usando **Tkinter**.

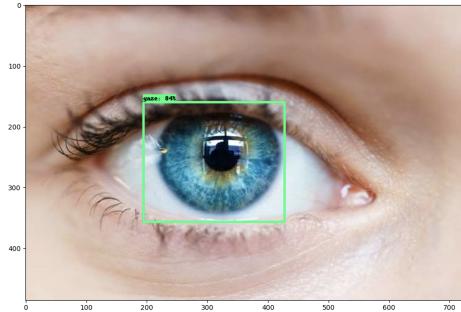
---

<sup>3</sup><https://www.kaggle.com/datasets/pavelbiz/eyes-rtte>

<sup>4</sup><https://github.com/tzutalin/labelImg>

A fini di testing si è preso ad esempio l'occhio qui sotto:

Figura 5: Gaze Testing GUI TkAgg



### 2.2.3 TfLite

Seguendo gli stessi identici procedimenti eseguiti durante la conversione dell'eye detection, anche qui dopo esserci accertati che la rete funzionasse correttamente e avesse dei livelli di precisione sopra una certa soglia, si è ottenuto in output un modello addestrato e pronto all'uso, che poi è stato convertito e quantizzato in un formato adatto ai sistemi embedded, quello di TensorFlow Lite.

### 2.2.4 Metadata

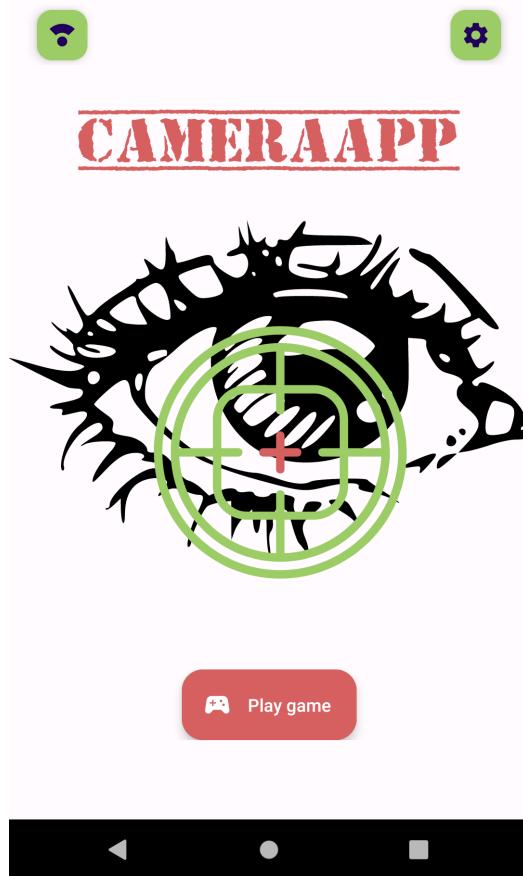
Per un corretto funzionamento della rete neurale all'interno di Android, TfLite richiede che venga generato anche un file Metadata che contiene le informazioni necessarie per pre-processare le immagini. Questo file si rende necessario in quanto i nostri tensori di input sono di tipo kTfLiteFloat32. È necessario creare un nuovo *label\_map.txt* con al suo interno nella prima riga il nostro item per la detection: nel nostro caso "gaze".

Una volta generato questo nuovo file metadata.tflite siamo pronti ad importarlo all'interno della nostra applicazione Android come file di Machine Learning "ml".

### 3 Progetto Android

L'applicazione Android una volta avviata presenta la seguente interfaccia:

Figura 6: Homepage App



Sono da segnalare i 3 principali bottoni dell'applicazione:

- **Nerd Mode** in alto a destra;
- **Calibration** in alto a sinistra;
- **Play Game** al centro dello schermo.

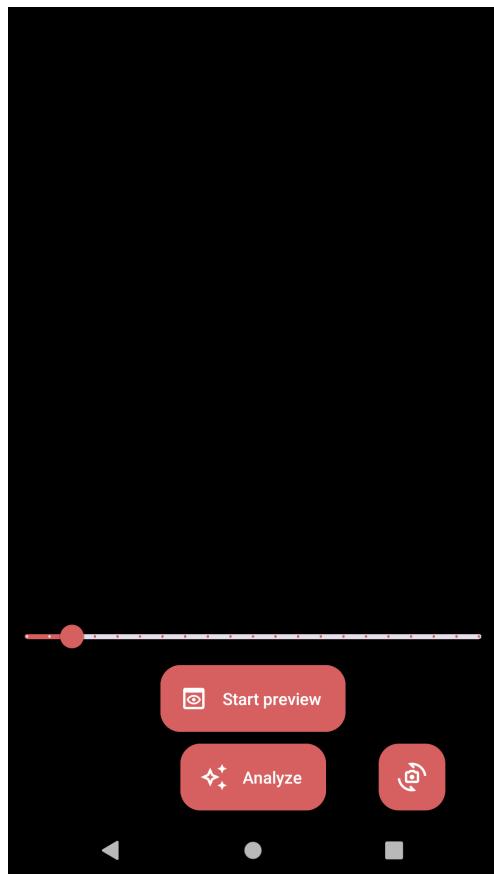
#### 3.1 Nerd Mode

Modalità la cui utilità è quella di individuare real time gli occhi della persona, in particolare permette di vedere i boxes creati dalla rete neurale attorno agli occhi dell'utente.

Interfaccia che presenta 3 bottoni:

- **Preview** che permette di avviare la propria fotocamera frontale o esterna;
- **Analyze** che permette la visualizzazione dei boxes creati dalla rete neurale in real time;
- **Fotocamera** frontale/esterna per cambiare da una fotocamera all'altra.

Figura 7: Schermata Nerd mode

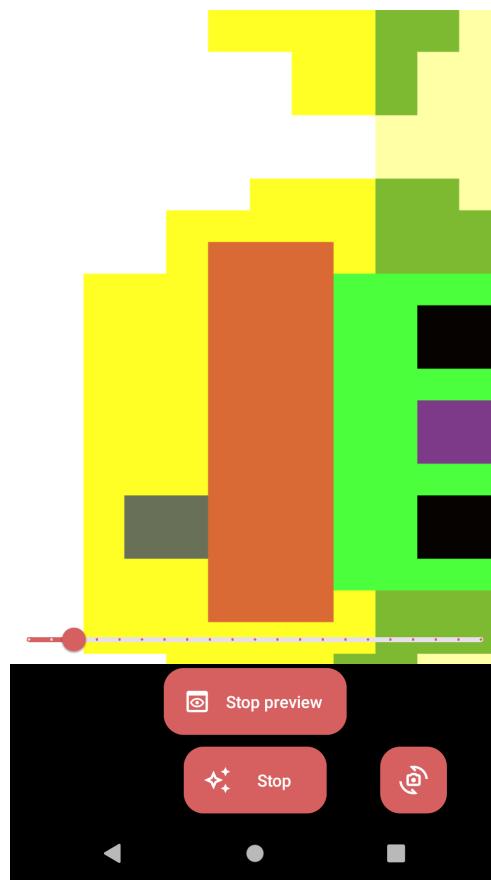


### 3.1.1 Bottone Analyze

Bottone premuto una volta avviata la *Preview*, ci permette di vedere per la prima volta la nostra rete addestrata al lavoro su Android. È stata inserita anche una barra di scorrimento in modo tale da impostare e mostrare real time a schermo gli occhi individuati solo sopra una certo livello di precisione della rete. (0% minimo - 100% massimo). Su Android se si inquadra bene gli occhi, in media si ha una precisione del 80%.

Chiaramente più viene allontanato il telefono dalla faccia, più cala la precisione, ma comunque la rete individua sempre e comunque gli occhi a meno che non si provino a nascondere gli occhi o ci si metta di profilo.

Figura 8: Schermata Nerd mode con analyze attiva

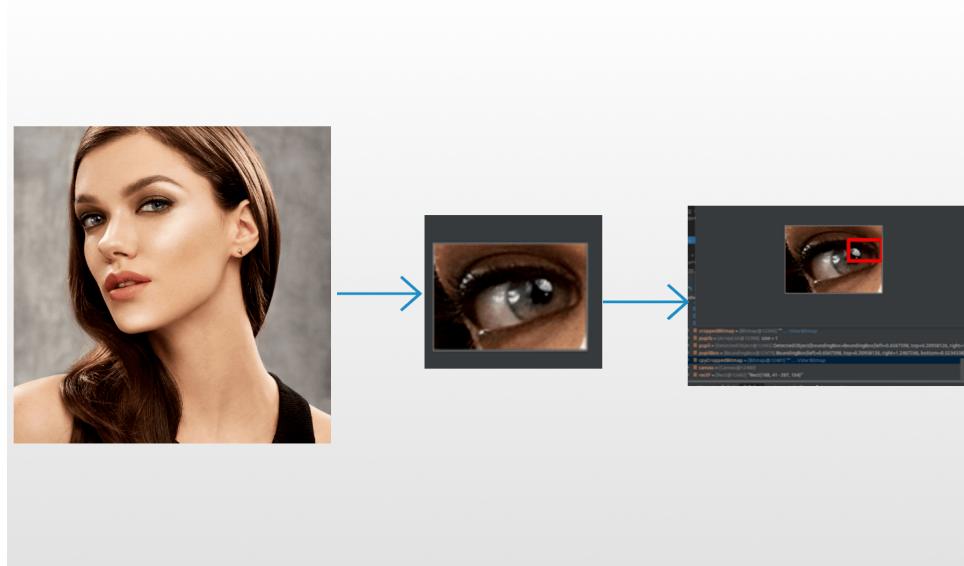


### 3.1.2 Riconoscimento pupilla

Il riconoscimento della pupilla sfrutta in un primo momento la rete neurale del riconoscimento degli occhi, la quale restituisce al nostro sistema solamente l'immagine contenente l'occhio.

Quindi, la rete neurale della pupilla usa questa immagine ritagliata per riconoscere e calcolare i bordi attorno alla pupilla, restituendo quindi i quadrati attorno a queste ultime.

Figura 9: Schermata Nerd mode con pupilla



### 3.2 Calibration

Modalità usata per calibrare al meglio la fotocamera e ottenere risultati i più veritieri possibili. Utile per svolgere al meglio il mini gioco descritto in seguito. Qui di sotto la sua schermata.

Figura 10: Schermata Calibration



### 3.3 Game

Modalità usata per completare un mini-gioco sfruttando le potenzialità della rete neurale sottostante. Tramite un mini gioco Quiz infatti, l'utente può rispondere alle domande del Quiz semplicemente spostando il cellulare verso uno dei 4 angoli dove è posizionata la risposta corretta: infatti viene riconosciuto l'occhio e viene visualizzato un puntatore che lo identifica sullo schermo. La domanda viene posta al centro dello schermo per motivi pratici.

I dati delle domande e delle risposte sono state recuperate da uno dei tanti servizi di Api pubblici che si trovano in rete: tramite appunto una richiesta Url, vengono ricevuti i dati casuali in formato JSON per una singola domanda, e una volta ricevuta la risposta, corretta o sbagliata che sia, viene generata un'altra richiesta in modo tale da poter continuare il mini gioco.

Figura 11: Schermata Game



## 4 Conclusioni

Gli obiettivi del progetto sono stati raggiunti a pieno e con risultati soddisfacenti.

L'applicativo è conforme alle aspettative e svolge i compiti adeguatamente. Le parti realizzate in Android risultano con prestazione adeguate, nonostante i cali di precisione dovuti alla conversione in modello .tflite.

Il modello addestrato restituisce un output corretto la maggior parte delle volte con una precisione di eye tracking superiore al 85% e di gaze tracking superiore al 95%, precisione che una volta importato su Android cala, ma mantenendo sempre risultati soddisfacenti.