

EYE-PUPIL TRACKER

Presentazione progetto e attività progettuale di Sistemi Digitali M


Introduzione

- Realizzazione di un **app in Android** in grado di riconoscere gli occhi e le pupille di uno o più utenti.
- Rete neurale convoluzionale (CNN) ottimizzata per sistemi embedded.
- Detection real-time nelle immagini tramite fotocamera interna e esterna.
- Mini gioco Quiz per sfruttare la rete neurale.

Dataset occhio

- Dataset pubblico fornito da Google Open Image V6:
 1. 30 mila immagini contenenti una o più persone;
 2. File CSV contenenti posizioni geografiche degli occhi per ogni immagine.

- Creato file label_map.pbtxt.



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

- Software CSV Converter per associare a ogni immagine un file .xml contenente le informazioni utili alla generazione dei **TFRecord**:
 1. Nome immagine;
 2. Posizione x e y degli occhi.

Dataset pupilla

- Dataset pubblico fornito da Kaggle:
 - 5 mila immagini contenenti pupille umane;
- Tool grafico **LabelImage** per costruire attorno alle pupille i boxes e per avere in output i corrispettivi file .xml

- Creato file label_map.pbtxt.



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

- I files .xml contengono le informazioni utili alla generazione dei **TFRecord**:
 1. Nome dell'oggetto da riconoscere;
 2. Posizione x e y della pupilla.

Training

- Utilizzo di Modello SSD MobileNet V2 FPNLite 640x640: leggero e veloce da essere eseguito su smartphone, senza consumo di risorse eccessivo, mantenendo comunque una precisione adeguata.
- Configurato adeguatamente file *pipeline.config*.

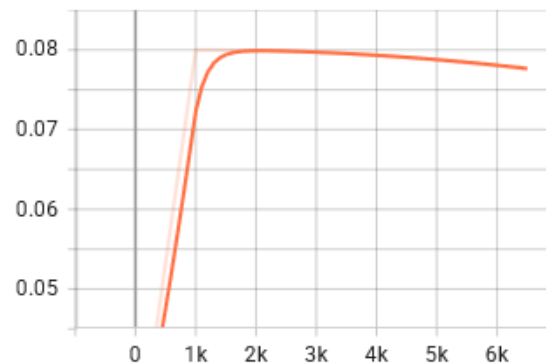
```
model {
  ssd {
    num_classes: 1 #TO CHANGE <-----
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
```

```
train_input_reader: {
  label_map_path: "annotations/label_map.pbtxt" #TO CHANGE <-----
  tf_record_input_reader {
    input_path: "annotations/train.record" #TO CHANGE <-----
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader: {
  label_map_path: "annotations/label_map.pbtxt" #TO CHANGE <-----
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.record" #TO CHANGE <-----
  }
}
```

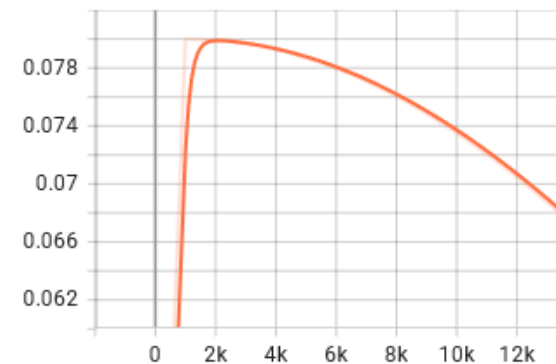
Training TensorBoard

- TensorBoard fornisce la visualizzazione e gli strumenti necessari per la sperimentazione del machine learning:
 1. Monitoraggio e visualizzazione di metriche come perdita e precisione;
 2. Visualizzazione del grafico del modello (operazioni e livelli);
 3. Visualizzazione degli istogrammi di pesi, distorsioni o altri tensori, man mano che cambiano nel tempo.

Learning Rate:



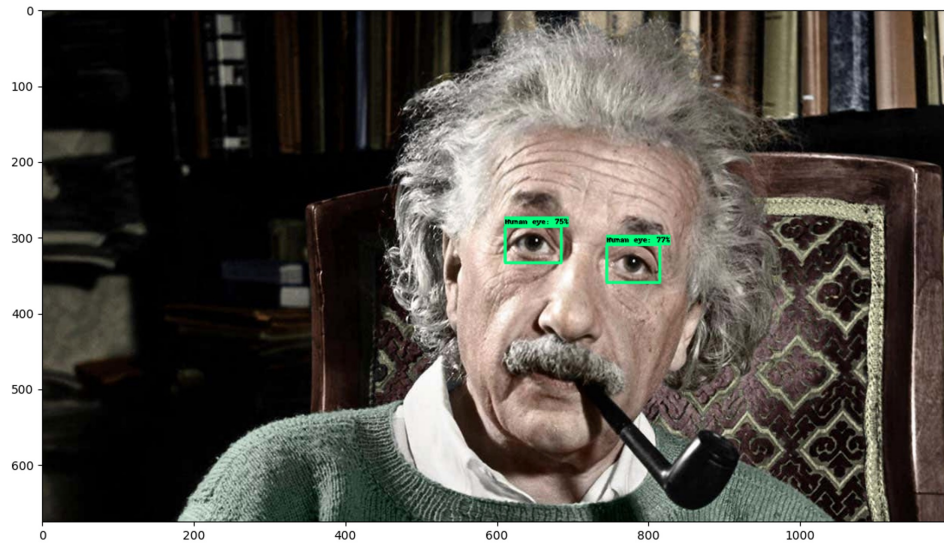
EYE



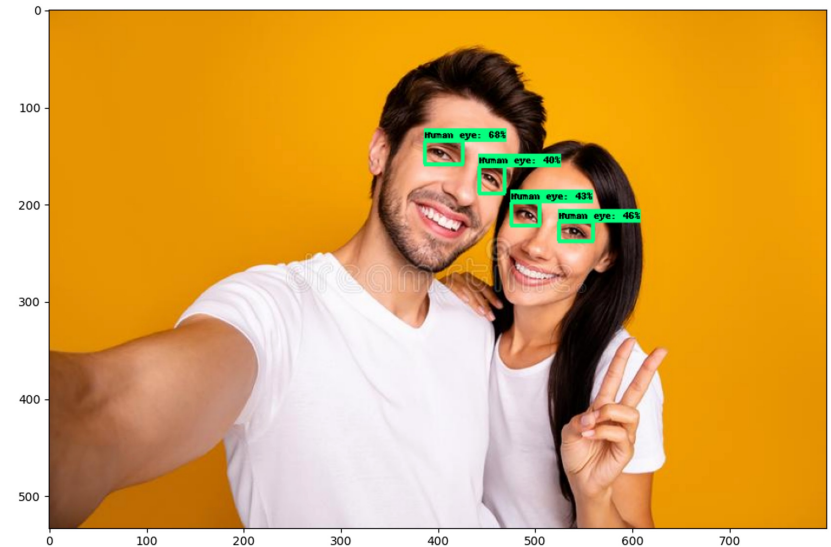
PUPIL

Testing

- GUI da shell usando TkAgg della libreria Matplotlib.



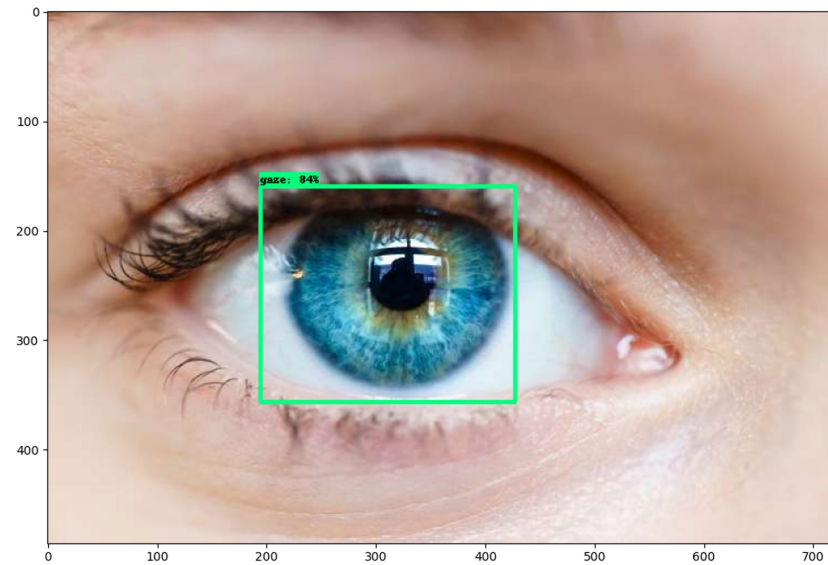
Test con persona famosa



Test multiplo

Testing

- GUI da shell usando TkAgg della libreria Matplotlib.



Test pupilla

Tensorflow Lite

- 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.

```
saved_model_dir = 'pathtosaved_model/saved_model'

# Convert the model
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite_model = converter.convert()
```



Otteniamo in output un file .tflite

Tensorflow Lite

- Tflite richiede un file metadata nel caso di tensori di input di tipo `kTfLiteFloat32` per pre-processare le immagini:

```
ObjectDetectorWriter = object_detector.MetadataWriter
_MODEL_PATH = "eyemodel.tflite"
_LABEL_FILE = "labelmap.txt"
_SAVE_TO_PATH = "eyemodel_metadata.tflite"

writer = ObjectDetectorWriter.create_for_inference(
    writer_utils.load_file(_MODEL_PATH), [127.5], [127.5], [_LABEL_FILE])
writer_utils.save_file(writer.populate(), _SAVE_TO_PATH)
```



Ora si importa il *metadata.tflite* come file Machine Learning *ml* all'interno dell'applicazione Android

Android

Interfaccia Home

- **Play Game**, al centro per giocare al mini gioco;
- **Calibration**, in alto a sinistra per calibrare la fotocamera prima del gioco;
- **Nerd mode**, in alto a destra per filtraggio real time della fotocamera.



Android

Interfaccia Nerd Mode

Individua real time gli occhi e pupille della persona, permettendo di vedere i boxes attorno.

Interfaccia che presenta *3 bottoni*:

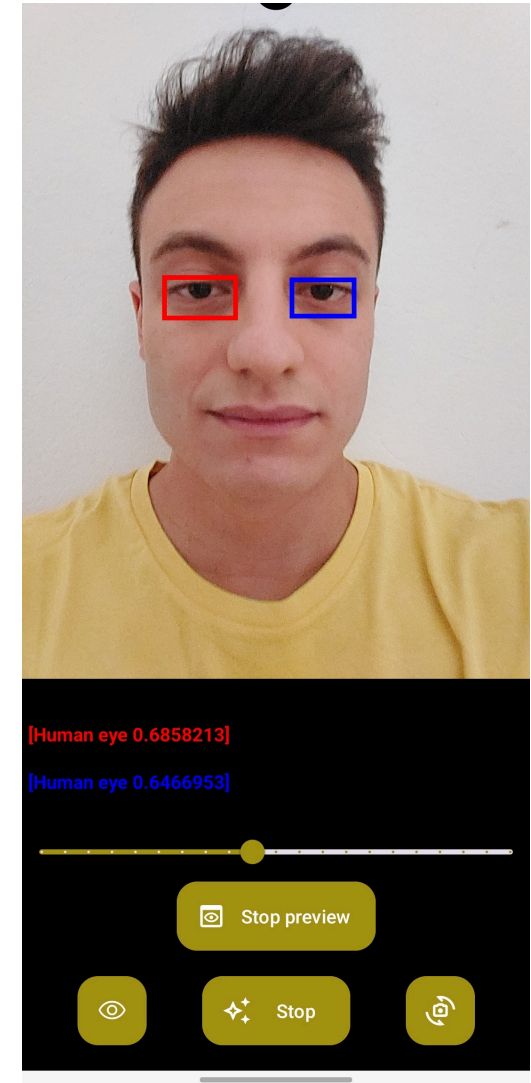
- **Preview** che permette di avviare la propria fotocamera frontale o esterna;
- **Analyze** che permette la visualizzazione dei boxes creati dalla rete neurale in real time;
- **Fotocamera frontale/esterna** per cambiare da una fotocamera all'altra.



Android

Interfaccia Nerd Mode Analyze

Inserita anche una barra di scorrimento in modo tale da impostare e mostrare a schermo gli occhi individuati solo sopra un certo livello di precisione della rete. (0% minimo – 100% massimo)



Android

Interfaccia Nerd Mode Analyze con 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.

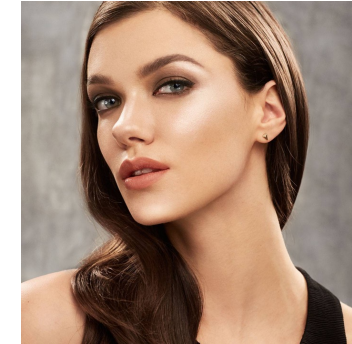
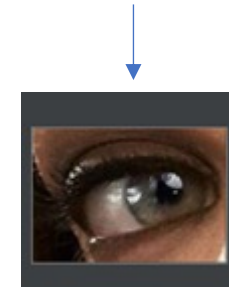
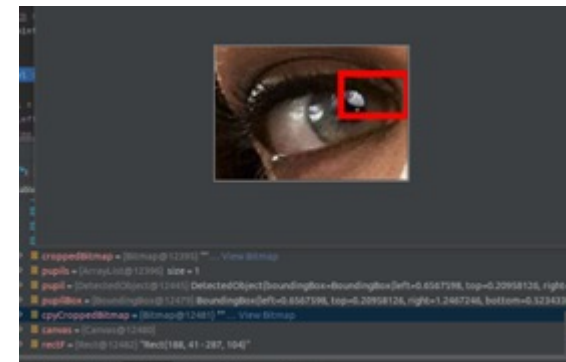


Foto originale



Riconoscimento occhio



Riconoscimento pupilla

Android

Interfaccia 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.

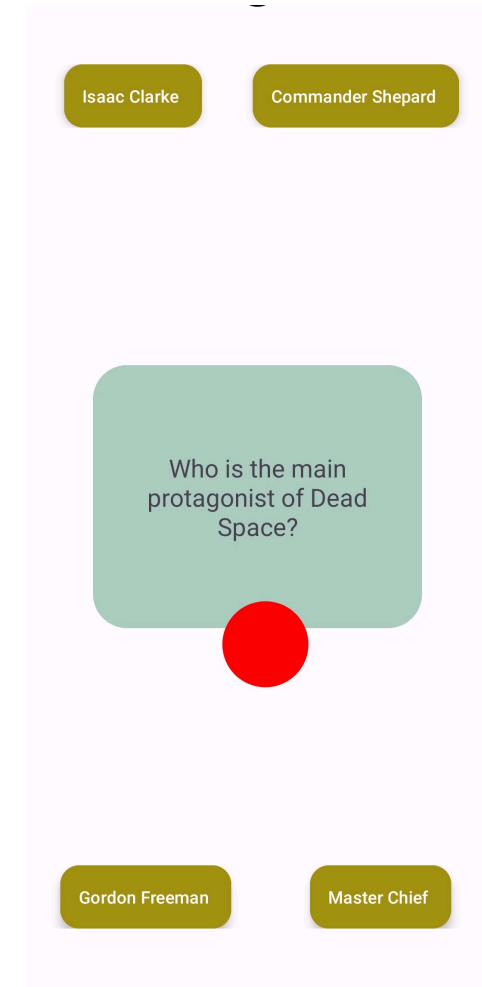


Android

Interfaccia Game

Sfrutta le potenzialità della rete neurale sottostante. Tramite un Quiz, l'utente risponde alle domande casuali semplicemente spostando il cellulare: infatti viene riconosciuto l'occhio e viene visualizzato un puntatore che lo identifica sullo schermo in modo tale da portarlo sulla risposta corretta posizionata su uno dei 4 angoli dello schermo.

I dati delle domande e delle risposte recuperate tramite API pubbliche: viene inviata una richiesta Url e vengono ricevuti i dati casuali in formato JSON.



Conclusioni

- Obiettivi del progetto sono stati raggiunti a pieno e con risultati soddisfacenti;
- 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 pupil tracking superiore al 95%.

Link utili

- https://github.com/DaniDF/eye_pupil_tracker,
Repository Github del progetto.
- https://github.com/DaniDF/eye_pupil_tracker/blob/master/Report/train_custom_object_detection.ipynb,
Jupyter Notebook del progetto.