

# Analisi del Codice Sorgente Progetto di Riconoscimento Facciale

Ghidini Matteo

12 gennaio 2026

## Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Architettura Generale</b>	<b>2</b>
<b>3</b>	<b>Analisi dei File Sorgente</b>	<b>3</b>
3.1	config.py . . . . .	3
3.2	detector.py . . . . .	3
3.3	recognizer.py . . . . .	3
3.4	utils.py . . . . .	3
3.5	augment.py . . . . .	4
3.6	image_to_embedding.py . . . . .	4
3.7	extract_embeddings.py . . . . .	4
3.8	initializer.py . . . . .	4
3.9	main_camera.py . . . . .	4
3.10	main_image.py . . . . .	5
3.11	main_look_alike_offline.py . . . . .	5
3.12	main_look_alike_online.py . . . . .	5
3.13	camera.py e old_augment.py . . . . .	5
<b>4</b>	<b>Flow di utilizzo applicazione</b>	<b>5</b>
4.1	Utilizzo immediato . . . . .	5
4.2	Aggiunta di persona al dataset di riconoscimento . . . . .	5
4.3	Aggiunta di persona al dataset per operazione Look-alike . . . . .	6

## 1 Introduzione

Questo documento analizza la struttura del codice sorgente del progetto di riconoscimento facciale. L'obiettivo è descrivere lo scopo di ogni file Python nella directory `src/` e mappare le relazioni e le dipendenze tra di essi per fornire una visione chiara dell'architettura complessiva.

## 2 Architettura Generale

Il sistema è suddiviso in tre aree principali:

1. **Componenti Core:** Moduli riutilizzabili che forniscono funzionalità atomiche come il rilevamento di volti (`detector.py`) e la generazione di embedding (`recognizer.py`).
2. **Script di Preparazione Dati:** Programmi utilizzati per processare i dati offline. Questo include l'aumento dei dati (`augment.py`) e l'estrazione degli embedding da salvare su disco (`image_to_embedding.py`, `extract_embeddings.py`).
3. **Script Eseguibili (Entrypoint):** I programmi principali che l'utente avvia per eseguire le diverse funzionalità del progetto, come il riconoscimento da webcam (`main_camera.py`) o la ricerca di sosia (`main_look_alike_*.py`).

Il file `config.py` agisce come un punto di configurazione centrale, fornendo percorsi e costanti a tutti gli altri moduli.

## 3 Analisi dei File Sorgente

### 3.1 config.py

**Scopo:** Centralizza tutte le costanti e i percorsi utilizzati nel progetto. Definisce dove trovare i modelli, i dataset, le immagini da classificare e dove salvare gli output come gli embedding.

**Relazioni:** Viene importato da quasi tutti gli altri file del progetto che necessitano di accedere a risorse del filesystem. Questo approccio disaccoppia la logica del codice dai percorsi specifici.

### 3.2 detector.py

**Scopo:** Contiene la classe `FaceDetector`, che utilizza un modello YOLO (caricato tramite la libreria `ultralytics`) per individuare i volti all'interno di un'immagine.

**Funzionalità:** Fornisce metodi per rilevare i volti restituendo i loro riquadri di delimitazione (bounding box) e, opzionalmente, i punti di riferimento facciali (keypoints).

**Relazioni:** È una dipendenza fondamentale per tutti gli script che devono processare un volto, come `image_to_embedding.py`, `extract_embeddings.py` e tutti i file `main_*.py`.

### 3.3 recognizer.py

**Scopo:** Contiene la classe `FaceRecognizer`. Il suo compito è prendere un'immagine di un volto (già ritagliata) e trasformarla in un vettore numerico ad alta dimensionalità, chiamato "embedding".

**Funzionalità:** Utilizza un modello di deep learning pre-addestrato (basato su architetture come FaceNet/VGGFace2) per generare un embedding normalizzato (L2). Questi embedding possono essere poi confrontati tramite similarità cosenica per determinare se due volti appartengono alla stessa persona.

**Relazioni:** È una dipendenza chiave per `image_to_embedding.py`, `extract_embeddings.py` e per tutti gli script `main_*.py` che eseguono il riconoscimento o la ricerca di similarità.

### 3.4 utils.py

**Scopo:** Libreria di funzioni di utilità generiche, utilizzate in più parti del progetto per evitare la duplicazione del codice.

**Funzionalità:**

- Funzioni di preprocessing delle immagini per adattarle agli input dei modelli.

- Funzioni per caricare gli embedding salvati su file `.npz`.
- La funzione `recognize_faces`, che orchestra il rilevamento e il riconoscimento su un'immagine.
- Funzioni per disegnare etichette e riquadri sulle immagini di output.
- Funzioni per la ricerca dei top-k volti più simili.

**Relazioni:** Importato da tutti i file `main_*.py` e da altri script che necessitano di queste funzionalità ausiliarie.

### 3.5 `augment.py`

**Scopo:** Script per l'aumento dei dati (data augmentation). Aumenta il numero di immagini nel dataset applicando trasformazioni casuali (flip, luminosità, crop) alle immagini originali.

**Funzionalità:** Per ogni persona nel dataset, controlla se il numero di immagini aumentate è inferiore a una soglia e, in caso affermativo, ne genera di nuove.

**Relazioni:** È uno script di preparazione dati. Non è una dipendenza di altri moduli, ma il suo output (le immagini aumentate) viene processato da `image_to_embedding.py`.

### 3.6 `image_to_embedding.py`

**Scopo:** Script cruciale per la preparazione del database di riconoscimento. Scansiona il dataset (immagini originali e aumentate), rileva i volti, calcola i loro embedding e li salva in un file compresso `embeddings.npz` per ogni persona.

**Flusso:**

1. Itera su ogni persona nel dataset.
2. Per ogni immagine, usa `detector.py` per trovare il volto.
3. Passa il volto ritagliato a `recognizer.py` per ottenere l'embedding.
4. Salva tutti gli embedding di una persona in un unico file `.npz`.

**Relazioni:** Utilizza `detector.py` e `recognizer.py`. Produce i file `.npz` che sono l'input fondamentale per `main_camera.py` e `main_image.py`.

### 3.7 `extract_embeddings.py`

**Scopo:** Simile a `image_to_embedding.py`, ma specifico per la funzionalità "look-alike". Processa due cartelle separate (`people` e `known_people`) e crea due database di embedding distinti (`people.npz` e `known.npz`).

**Relazioni:** Utilizza `detector.py` e `recognizer.py`. I suoi file di output sono usati da `main_look_alike_offline.py` e `main_look_alike_online.py`.

### 3.8 `initializer.py`

**Scopo:** Contiene un solo metodo, per inizializzare `detector` e `recognizer`. Prende due path che portano ai rispettivi pesi e crea i modelli.

**Relazioni:** Dipende da `FaceDetector` e `FaceRecognizer` per creare i modelli e dalla classe `InceptionResnetV1` per la struttura del modello.

### 3.9 `main_camera.py`

**Scopo:** Entrypoint principale per il riconoscimento facciale in tempo reale tramite webcam.

**Flusso:**

1. Carica in memoria tutti gli embedding del dataset usando `utils.load_dataset_embeddings`.
2. In un ciclo infinito, cattura un frame dalla webcam.
3. Usa la funzione `utils.recognize_faces` (che a sua volta usa `detector` e `recognizer`) per trovare e identificare i volti nel frame.
4. Disegna i risultati sul frame e lo mostra a schermo.

**Relazioni:** Dipende da `detector.py`, `recognizer.py` e `utils.py`.

### 3.10 `main_image.py`

**Scopo:** Entrypoint per il riconoscimento facciale su un set di immagini statiche.

**Flusso:** Simile a `main_camera.py`, ma invece di un ciclo sulla webcam, itera sui file di immagine presenti in una cartella specificata in `config.py`.

**Relazioni:** Dipende da `detector.py`, `recognizer.py` e `utils.py`.

### 3.11 `main_look_alike_offline.py`

**Scopo:** Esegue una ricerca "look-alike" (sosia) offline. Confronta ogni persona nel database "known" con tutte le persone nel database "people" per trovare le corrispondenze più simili.

**Flusso:** Carica i due database di embedding pre-calcolati da `extract_embeddings.py` e calcola la similarità cosenica per trovare i top-K match per ogni persona "known".

**Relazioni:** Dipende da `utils.py` per caricare gli embedding e trovare i match. Non necessita di `detector` o `recognizer` perché lavora su embedding già esistenti.

### 3.12 `main_look_alike_online.py`

**Scopo:** Esegue una ricerca "look-alike" in tempo reale. Rileva un volto dalla webcam e cerca la persona più simile all'interno del database "people".

**Flusso:**

1. Carica il database di embedding "people".
2. In un ciclo, cattura un frame, rileva un volto con `detector.py` e calcola il suo embedding al volo con `recognizer.py`.
3. Confronta l'embedding appena calcolato con quelli del database per trovare il miglior match.

**Relazioni:** Dipende da `detector.py`, `recognizer.py` e `utils.py`.

### 3.13 `camera.py` e `old_augment.py`

**Scopo:**

- `camera.py`: Fornisce una semplice classe wrapper per `cv2.VideoCapture`. Attualmente non sembra essere utilizzata da nessuno degli script principali, che chiamano direttamente `cv2`. Potrebbe essere un residuo o un componente per usi futuri.
- `old_augment.py`: Sembra una versione precedente e più semplice di `augment.py`. Probabilmente è stato deprecato in favore della versione più robusta.

## 4 Flow di utilizzo applicazione

### 4.1 Utilizzo immediato

Se si vuole far semplicemente partire l'applicativo basta scrivere su bash shell:  
`python -m src.main_*` Questo farà partire una dei main presenti in `src`.

### 4.2 Aggiunta di persona al dataset di riconoscimento

Se invece si vuole aggiungere persone alla knowledge base serve fare alcuni passaggi aggiuntivi:

1. Aggiungere alla dir `data/dataset` la dir con nome quello della persona che si vuole aggiungere alla KB

2. Eseguire su bash shell:  
`python -m augment`  
Questo agumenterà l'immagini presenti nel dataset
3. Per creare gli embeddings dalle immagini: `python -m src.image_to_embedding`
4. Si può far partire i `main_camera` o `main_image` con  
`python -m src.main_*`

### 4.3 Aggiunta di persona al dataset per operazione Look-alike

Se invece si vuole aggiungere persone nel dataset dei Look-alike serve fare alcuni passaggi aggiuntivi e il primo punto cambia per modalità online (via camera) e offline (img nel dataset)

1. Aggiungere alla dir `data/similarity_images/known_people` l'immagine con nome quello della persona che si vuole aggiungere alla lista di persone che si vogliono confrontare con i Look-alike
2. Aggiungere alla dir `data/similarity_images/people` le immagini di persone "sconosciute" con cui si vuole fare un confronto (nota che questo passaggio è abbastanza computationally intensive data la mole delle immagini. Consiglio di NON fare aggiungere e quindi ricreare gli embeddings della dir `similarity_images/people`)
3. Per creare gli embeddings dalle immagini in `similarity_images`:  
`python -m src.extract_embeddings`  
Nota che di base è commentata la parte di codice che fa l'embeddings delle `similarity_images/people` per sicurezza (ultime righe del file `.py`)
4. Si può far partire i main interessati al look-alike comparison con  
`python -m src.main_look_alike_offline` per confrontare immagini in `similarity_images/known_people`;  
`python -m src.main_look_alike_online` per confrontare direttamente attraverso la camera.