

# Progetto di Riconoscimento Facciale

## Deep Learning e Architetture di Reti Neurali

Matteo Ghidini

3 febbraio 2026

# Sommario

- 1 Introduzione
- 2 Architettura del Sistema
- 3 Preparazione Dati
- 4 Architetture di Reti Neurali
  - DNN
  - CNN
  - Transfer Learning
  - ResNet Pre-trained
- 5 Training e Ottimizzazione
- 6 Risultati e Confronto
- 7 Conclusioni

# Obiettivi del Progetto

- **Face Recognition:** identificazione automatica di persone
- **Look-alike Search:** ricerca di volti simili
- **Architettura modulare:** detector + recognizer
- **Confronto architetture:** DNN, CNN, Transfer Learning, ResNet

## Principio di funzionamento:

- Rimozione logit layer → embedding vettoriale
- Distanza nello spazio n-dimensionale = similarità
- Cosine similarity per confronto embeddings

# Componenti Principali

## Detector (YOLO)

- Localizzazione volti
- Bounding boxes
- Confidence scores

## Recognizer

- Generazione embeddings
- Normalizzazione L2
- Confronto similarità

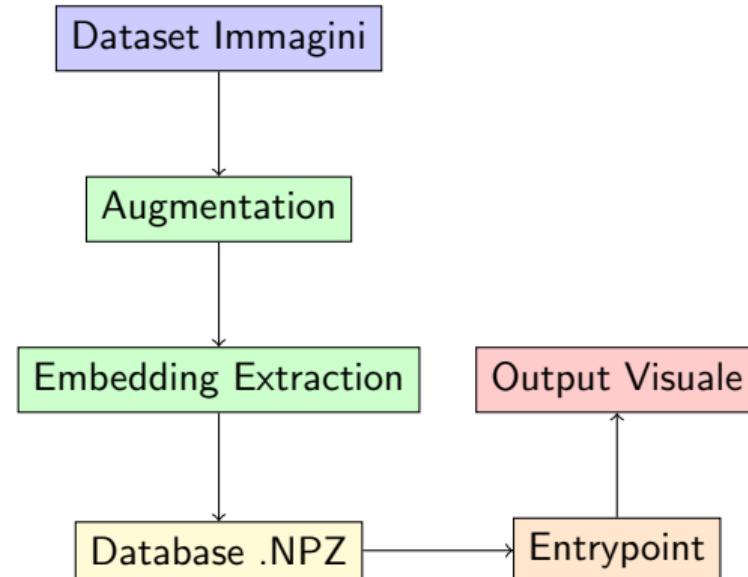
## Moduli di supporto

- utils.py: orchestrazione
- initializer.py: factory pattern
- config.py: configurazione

## Entry point

- main\_camera
- main\_image
- main\_look\_alike

# Flusso Operativo



# Detector: Rilevamento Volti

```
1 def detect(self, frame):
2     results = self.model(frame, conf=self.conf, verbose=False)
3     faces = []
4
5     for result in results:
6         boxes = result.boxes.xyxy.cpu().numpy()
7         scores = result.boxes.conf.cpu().numpy()
8
9         for i in range(len(boxes)):
10             x1, y1, x2, y2 = map(int, boxes[i])
11             faces.append({
12                 'bbox': (x1, y1, x2, y2),
13                 'score': float(scores[i])
14             })
15
16     return faces
```

# Recognizer: Generazione Embeddings

```
1 def get_embedding(self, face_bgr: np.ndarray) -> np.ndarray:
2     face_tensor = self._preprocess(face_bgr)
3
4     with torch.no_grad():
5         if self.use_get_embedding:
6             embedding = self.model.get_embedding(face_tensor)
7         else:
8             embedding = self.model(face_tensor)
9
10    embedding = embedding.cpu().numpy()[0]
11
12    # Normalizzazione L2
13    norm = np.linalg.norm(embedding)
14    embedding = embedding / norm if norm > 0 else embedding
15
16    return embedding
```

# Riconoscimento: Cosine Similarity

```
def recognize_faces(frame, detector, recognizer,
                    embeddings_array, labels_list, threshold=0.60):
    faces = detector.detect(frame)
    results = []

    for face in faces:
        (x1, y1, x2, y2) = face['bbox']
        face_crop = frame[y1:y2, x1:x2]

        emb = recognizer.get_embedding(face_crop)

        if embeddings_array.size > 0:
            # cosine similarity con embeddings L2-normalizzati
            sims = embeddings_array @ emb
            best_idx = np.argmax(sims)
            best_sim = sims[best_idx]

            if best_sim > threshold:
                name = labels_list[best_idx]
                confidence = (best_sim - threshold) / (1.0 - threshold)

            results.append({'bbox': (x1, y1, x2, y2), 'name': name})

    return results
```

## Organizzazione directory:

- data/dataset/: KB riconoscimento
- data/classify\_images/: immagini da classificare
- data/similarity\_images/: look-alike
- data/test\_images/: testing modelli

## Augmentation:

- Random flip
- Brightness adjustment
- Random crop
- Rotazioni

## Versioning:

- Hash-based: evita ricomputazione
- File .npz per modello

## GUI Features:

- ① **Model Selection:** scelta modello recognizer
- ② **Import & Process:** aggiunta persone + embedding
- ③ **Load Images:** caricamento immagini da classificare
- ④ **Run Scripts:** esecuzione main scripts
- ⑤ **Utilities:** manutenzione database embeddings
- ⑥ **Model Testing:** test sistematici + metriche
- ⑦ **Console Output:** log operazioni real-time

**Comando:** `python -m src.gui_app`

# Interfaccia Grafica



Immagini/GUI/Relazione\_GUI\_App.png

# Panoramica Modelli

Architettura	Parametri	Embedding	Approccio
ResNet (pre-trained)	-	512	State-of-the-art
DNN	51-101M	128-256	Fully connected
CNN	5M-37M	128-256	Convoluzionale
Transfer Learning	variabile	128-256	Fine-tuning

## Dataset utilizzati:

- **LFW**: 13.000 immagini, 5749 identità ( $\geq 10$  img/persona)
- **CelebA**: 200.000 immagini, attributi booleani (male, young, smiling, eyeglasses)

# Deep Neural Network

## Architettura:

Input<sub>128×128×3</sub> → Flatten → Dense<sub>n</sub> → ⋯ → Embedding<sub>d</sub> → Classifier

## Configurazioni testate:

Config	Hidden Sizes	Embedding	Dropout	Val Acc (%)
Baseline	[1024, 512]	128	0.5	34.45
Deep	[2048, 1024, 512, 256]	128	0.4	26.94
Optimized	[1536, 768, 384]	256	0.3	44.16

## Limitazioni:

- Alto numero parametri per dati spaziali
- Perdita informazioni strutturali immagine
- Prone a overfitting

# Convolutional Neural Network

## Architettura base:

Conv2D → BatchNorm → ReLU → MaxPool → ⋯ → FC → Embedding

## Variante con Residual Blocks:

- Skip connections:  $y = F(x) + x$
- Mitiga vanishing gradient
- Ispirata a ResNet

## LFW Dataset:

- Best: CNN Optimized
- Accuracy: 78.84%
- Parametri: 36.8M

## CelebA Dataset:

- Best: CNN Deep Residual
- Accuracy: 82.65%
- Parametri: 5.5M

# CNN: Risultati

Configurazione	Parametri	Val Acc LFW (%)	Val Acc CelebA (%)
Baseline	17.7M	57.80	80.88
Shallow	34.4M	64.16	81.34
Deep	18.9M	71.56	81.73
Large Kernels	17.8M	60.81	-
GAP	1.3M	13.06	79.03
Optimized	36.9M	<b>78.84</b>	82.40
Deep Residual	5.5M	52.25	<b>82.65</b>

## Osservazioni:

- Residual connections efficaci su CelebA
- Trade-off parametri vs accuracy
- Global Average Pooling riduce parametri ma degrada performance

# Transfer Learning e Fine-Tuning

## Approccio:

- ① Backbone pre-trained su ImageNet (ResNet18/34/50)
- ② Custom residual blocks
- ③ Fine-tuning opzionale

## Strategie:

- **Frozen backbone:** feature extraction
- **Fine-tuning:** unfreeze + LR basso ( $10^{-5}$ )

Config	Backbone	Val Acc LFW (%)	Val Acc CelebA (%)
ResNet18 Frozen	ResNet18	75.32	83.12
ResNet18 Finetuned	ResNet18	<b>81.45</b>	<b>85.67</b>
ResNet34 Frozen	ResNet34	77.89	84.21
ResNet50 Frozen	ResNet50	79.12	84.89

# InceptionResnetV1 (Baseline)

## Modello state-of-the-art:

- Pre-trained: VGGFace2, CASIA-WebFace
- Embedding: 512 dimensioni
- Architettura: Inception + Residual connections

## Motivazione:

- Baseline di riferimento
- Prestazioni elevate out-of-the-box
- Robustezza: addestrato su milioni di immagini
- Efficienza: no training da zero

**Utilizzo:** Cosine similarity su embeddings normalizzati L2

# Pipeline di Training

## Configurazione:

- **Loss:** CrossEntropyLoss + Label Smoothing
- **Optimizer:** Adam / RMSprop / SGD
- **Scheduler:** ReduceLROnPlateau
- **Regularization:** Dropout, Weight Decay, BatchNorm
- **Early Stopping:** patience=5-10 epoch

## Hyperparameters:

Parametro	DNN	CNN	Transfer Learning
Learning Rate	$10^{-3}$ - $5 \times 10^{-4}$	$10^{-3}$ - $3 \times 10^{-4}$	$10^{-3}$ (frozen), $10^{-5}$ (ft)
Batch Size	64-128	32-64	32-64
Epochs	10-20	35-45	35 (+5 ft)
Weight Decay	$0 - 10^{-4}$	$10^{-5} - 10^{-4}$	$10^{-4}$

## Dropout:

- Rate: 0.3-0.5
- Dopo FC e Conv layers
- Riduce co-adaptation

## Batch Normalization:

- Stabilizza training
- Accelera convergenza
- Dopo Conv/FC, prima ReLU

## Weight Decay:

- L2 regularization
- $\lambda = 10^{-5} - 10^{-4}$
- Penalizza grandi pesi

## Label Smoothing:

- $\epsilon = 0.05 - 0.1$
- Previene overconfidence
- Migliora generalizzazione

# Test su Dataset Personale

## Metodologia:

- Dataset personale: immagini reali
- Dataset AI-generated: Nano-Banana, ChatGPT-5
- Metriche: Accuracy, Precision, Recall

Architettura	Acc Reali (%)	Acc AI-Gen (%)
ResNet Pre-trained	88.1	88.9
Transfer Learning (ResNet18 FT)	50.0	18.5
CNN Optimized	16.7	11.1
CNN Deep Residual	40.5	0.0
DNN Optimized	42.9	77.8

## Punti di forza:

- **ResNet**: best accuracy, robusto
- **Transfer Learning**: ottimo trade-off
- **CNN**: customizzabile, efficiente

## Limitazioni:

- **ResNet**: black-box
- **Transfer Learning**: dipende da backbone
- **CNN**: richiede più training
- **DNN**: molto pesante

# Risultati Principali

## Contributi:

- ① Sistema completo face recognition + look-alike
- ② Confronto sistematico 4 architetture
- ③ GUI user-friendly

## Risultati:

- Transfer Learning > training da zero
- Residual connections essenziali per reti deep
- Regularization critica per generalizzazione
- Dataset quantity > quality

# Riferimenti e Risorse

## Dataset:

- LFW: <https://www.kaggle.com/datasets/jessicali9530/lfw-dataset>
- CelebA: <https://www.kaggle.com/datasets/jessicali9530/celeba-dataset>

## Modelli:

- ResNet: *He et al., CVPR 2016*
- InceptionResnetV1: *Szegedy et al., AAAI 2017*

## Repository:

- Codice:  
[https://github.com/Moryuss/Progetto\\_DeepLearning\\_e\\_architetture\\_di\\_Reti\\_Neurali](https://github.com/Moryuss/Progetto_DeepLearning_e_architetture_di_Reti_Neurali)
- Notebooks: training su Google Colab
- Pesi: <https://github.com/timesler/facenet-pytorch/tree/master?tab=readme-overfile#pretrained-models>

Grazie per l'attenzione

Domande?

## Tecnologie:

- **Framework:** PyTorch 2.0
- **Detection:** YOLO v8
- **GUI:** Tkinter
- **Training:** MuHack GPU NVIDIA Quadro RTX 8000 — 48 GB GDDR6

## Struttura codice:

- `src/`: detector, recognizer, utils
- `src/models/`: architetture custom
- `src/`: augmentation, embedding extraction
- `src/main_*.py`: entrypoint applicazione

## Face Recognition:

$$\text{Accuracy} = \frac{\text{Correct Matches}}{\text{Total Faces}}$$

## Look-alike Search:

$$\text{Similarity} = \cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$$

## Classification (training):

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$