Age Comparison Using UTK-Face Dataset

DEEP LEARNING FOR ENGINEERING APPLICATIONS

Electronic Engineering for Intelligent Vehicles

University of Bologna

A.A. 2025–2026

Kartik Purushottam Kanchan

December 27, 2025

Contents

Abstract

This project addresses the problem of relative age estimation from facial images, where the objective is to determine which of two given faces corresponds to the younger individual. A convolutional neural network (CNN) is first trained as an age regressor in order to learn age-aware facial representations. The learned latent embeddings are then used to construct a pairwise comparison model that predicts relative age relationships. The proposed approach follows a two-stage learning strategy and demonstrates that age supervision can be effectively leveraged to solve the relative age comparison task.

Introduction

Estimating human age from facial images is a well-studied problem in computer vision. While absolute age estimation remains challenging due to facial variability, illumination, and expression changes, relative age estimation—determining which of two individuals appears younger—can be a more robust and practical task.
The objective of this project is to design a deep-learning-based system that, given two face images, outputs whether the left or right image corresponds to the younger subject. Rather than directly predicting ages at inference time, the system is trained to learn discriminative age-related features and perform pairwise comparison.

1 Dataset and Preprocessing

1.1 Dataset

For this task, a custom dataset of face images with age annotations encoded in file names was used. Each image file name starts with an age label (e.g., 30. . . jpg), which identifies the age of the person in the image. The dataset was shuffled and split for training and validation during model training.

1.2 Prerequisite

- **Pandas** - An open-source Python library used for data handling and manipulation. In this project, Pandas is used to organize image file paths and corresponding age labels into structured dataframes, as well as to perform dataset shuffling and inspection.
- **NumPy** – A fundamental library for numerical computing in Python. NumPy is used for efficient array operations, normalization of image pixel values, reshaping image tensors, and handling multi-dimensional data required for neural network inputs.
- **Matplotlib**– An open-source visualization library used to display sample images, visualize age distributions, and plot training and validation loss curves during model training.
- **Seaborn**– A high-level data visualization library built on top of Matplotlib. Seaborn is used to generate statistical plots such as age distribution histograms, providing better insight into the dataset characteristics.

- **TensorFlow (Keras API)**– TensorFlow is an open-source deep learning framework developed by Google. In this project, the Keras high-level API within TensorFlow is used to design, train, and evaluate convolutional neural networks (CNNs) for age-aware feature learning and relative age comparison.
- **Keras**– Keras is a user-friendly neural network API integrated into TensorFlow. It provides modular building blocks such as convolutional layers, pooling layers, fully connected layers, and optimizers, enabling rapid development of deep learning models used in this project.
- **PIL (Python Imaging Library)**– Used for loading and processing image files. PIL enables reading facial images from disk before conversion into NumPy arrays for further preprocessing.
- **tqdm**– A utility library used to display progress bars during time-consuming operations such as image loading and feature extraction, improving usability and monitoring during preprocessing.

Importing main libraries

Listing 1: Imported libraries used for the age comparison model

```
## importing libraries
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from tqdm import tqdm
warnings.filterwarnings('ignore')

import tensorflow as tf
from keras.preprocessing.image import load_img
from keras.models import Sequential, Model
from keras.layers import Dense, Conv2D, Dropout, Flatten,
    MaxPooling2D, Input
```

The required software libraries for data handling, visualization, and deep learning are shown in Listing 1.

2 Data Preprocessing

In this project, the raw dataset consists of facial images stored as image files, with the corresponding age information embedded in the file names. Since deep learning models require numerical inputs of fixed dimensions, several preprocessing steps are applied to convert the raw images into a suitable format for training.

2.1 Dataset Loading and Label Extraction

The dataset is loaded by scanning a specified directory containing facial images. Only image files with a **(.jpg)** extension are considered. For each image, the full file path is stored, and the corresponding age label is extracted from the filename, where the age value precedes an underscore. Two lists are created: one to store image paths and another to store the associated age labels. This process establishes a structured mapping between each face image and its age annotation for subsequent training.

Listing 2: Loading Dataset

```
##Load the Dataset

Base_Dir = r"C:\Users\KartikLaptop\Downloads\Deeplearning\
    Age_Comparison\Directory"

# lables age
image_paths = []
age_labels = []

for filename in tqdm(os.listdir(Base_Dir)):
if not filename.lower().endswith(".jpg"):
continue

image_path = os.path.join(Base_Dir, filename)
age = int(filename.split("_")[0])
image_paths.append(image_path)
age_labels.append(age)
```

2.2 Image Loading and Label Extraction

Each image file name begins with the age of the individual (e.g., 25xxx.jpg). During dataset preparation, the age labels are extracted from the file names and stored along with the corresponding image paths in a Pandas DataFrame. The dataset is then randomly shuffled to eliminate any ordering bias during training.

Listing 3: Convert to Dataframe

```

# convert to dataframe
df = pd.DataFrame({
  "image": image_paths,
  "age": age_labels
})
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
```

2.3 Visualization of Preprocessed Images

To gain insight into the dataset, exploratory visualizations are performed. First, a histogram of age values is plotted using Seaborn to visualize the overall age distribution within the dataset. A kernel density estimate (KDE) is included to highlight the underlying

distribution trend. This visualization helps identify the range, spread, and potential imbalance of age labels.
Additionally, a grid of sample facial images is displayed to qualitatively inspect the dataset. The first 25 images are arranged in a 5×5 grid, where each image is shown alongside its corresponding age label. This allows visual verification of image quality, facial diversity, and the correctness of extracted age annotations.

Listing 4: Visualizing Dataset

```
# plot classifications
plt.figure(figsize=(6, 4))
sns.histplot(df['age'], kde=True)
plt.xlabel("Age")
plt.ylabel("Density")
plt.title("Age Distribution")
plt.show(block=False)

#plot grid of images
plt.figure(figsize=(10, 10))

files = df.iloc[:25].reset_index(drop=True)

for i, row in files.iterrows():
plt.subplot(5, 5, i + 1)

img = load_img(row["image"])
img = np.array(img)

plt.imshow(img)
plt.title(f"Age: {row['age']}")
plt.axis("off")

plt.tight_layout()
plt.show(block=False)
```

Sample visualizations confirm that the facial structures are preserved after preprocessing and that the labels align correctly with the images.
Output:

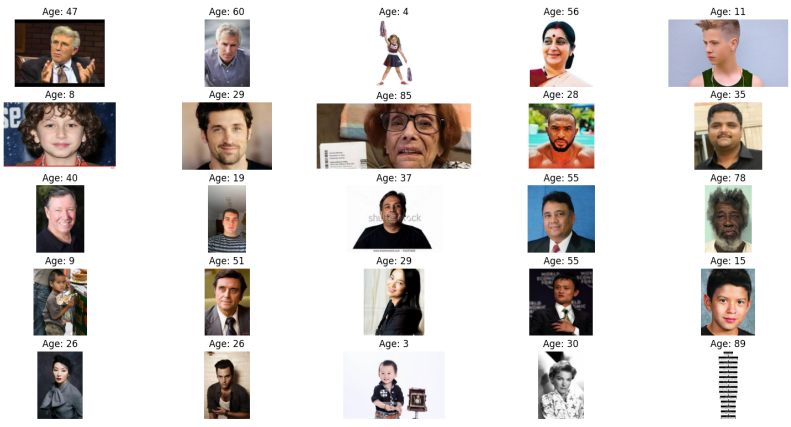

Figure 1: Dataset Visualization

2.4 Image Preprocessing and Feature Extraction

To standardize the input data, the images are processed using the following steps:

1. Each image is loaded in **grayscale** format to reduce computational complexity.
2. Images are resized to a fixed resolution of **128 × 128 pixels**.
3. The pixel values are converted into NumPy arrays.
4. All pixel intensities are normalized to the range **[0, 1]** to ensure stable neural network training.

A dedicated function is defined to perform this preprocessing for the entire dataset:

Listing 5: Extraction functions

```
# extraction functions
def extract_features(images):
features = []
for path in tqdm(images):
img = load_img(path, color_mode="grayscale", target_size=(128,
    128))
img = np.array(img)              # extraction into numpy
features.append(img)

features = np.array(features)        # (N, 128, 128)
features = features.reshape(len(features), 128, 128, 1)  # (N,
    128, 128, 1)
return features
```

The extracted features are normalized as follows:

Listing 6: Normalizing

```
print("Normalizing features...")
X = X / 255.0                        # normalization
```

2.5 Dataset Caching

To avoid repeated image loading and preprocessing during multiple runs, the processed image tensors and age labels are stored locally using NumPy files. If cached files are available, they are directly loaded, significantly reducing preprocessing time

Listing 7: Caching

```
    np.save(X_CACHE, X)
    np.save(Y_CACHE, y_age)
```

2.6 Pairwise Dataset Preparation

Since the objective of this project is relative age estimation, the preprocessed dataset is further transformed into a pairwise dataset. Random pairs of images are selected, and a binary label is assigned indicating whether the left image corresponds to a younger individual than the right image.

Listing 8: Creating Pairs

```
y_rel.append(1 if y[i] < y[j] else 0)
```

This step converts the absolute age estimation problem into a relative age comparison task, which is used for training the final classification model.

3 Creating the CNN Model Architecture

The convolutional neural network (CNN) architecture is implemented using the **Keras Functional API**, which provides flexibility in designing complex models and enables easy extraction of intermediate features. The primary objective of this CNN is to learn **age-aware facial representations** from input images. The network is trained using age regression and later employed as a feature extractor for relative age comparison.
The model architecture can be divided into three main parts:

3.1 Input Layer

The input to the model consists of grayscale facial images resized to a fixed resolution of **128 × 128 pixels**. Each image is represented as a single-channel tensor of shape:**(128×128×1)**

Listing 9: Input shape

```
input_shape = (128, 128, 1)
inputs = Input((input_shape))
```

This standardized input size ensures consistent feature extraction across all samples.

3.2 Hidden Layers (Feature Extraction Backbone)

The core of the model is composed of a sequence of convolutional and pooling layers designed to extract hierarchical facial features.

Convolutional Feature Extractor

- **Convolutional Block 1**
  - 32 filters with a kernel size of 3×3
  - ReLU activation function
  - Followed by 2×2 Max-Pooling
- **Convolutional Block 2**
  - 64 filters with a kernel size of 3×3

  - ReLU activation function
  - Followed by 2×2 Max-Pooling

- **Convolutional Block 3**
  - 128 filters with a kernel size of 3×3
  - ReLU activation function
  - Followed by 2×2 Max-Pooling

- **Convolutional Block 4**
  - 256 filters with a kernel size of 3×3
  - ReLU activation function
  - Followed by 2×2 Max-Pooling

Fully Connected Layers

After the convolutional stages, the extracted feature maps are flattened into a one-dimensional vector and passed through a series of fully connected layers:

- A dense layer with 256 neurons and linear activation
- A dropout layer with a dropout rate of 0.3 to mitigate overfitting
- A dense embedding layer with 128 neurons and ReLU activation

The embedding layer captures compact, high-level facial features that encode age-related information.

Listing 10: Convolution Layers

```
conv_1 = Conv2D(32, kernel_size=(3,3), activation='relu')(
    inputs)
maxp_1 = MaxPooling2D(pool_size=(2,2))(conv_1)

conv_2 = Conv2D(64, kernel_size=(3,3), activation='relu')(
    maxp_1)
maxp_2 = MaxPooling2D(pool_size=(2,2))(conv_2)

conv_3 = Conv2D(128, kernel_size=(3,3), activation='relu')(
    maxp_2)
maxp_3 = MaxPooling2D(pool_size=(2,2))(conv_3)

conv_4 = Conv2D(256, kernel_size=(3,3), activation='relu')(
    maxp_3)
maxp_4 = MaxPooling2D(pool_size=(2,2))(conv_4)
```

Flattening Layer

After convolution layers, feature maps are flattened into a one-dimensional vector.

Listing 11: Flatten Layers

```
flatten = Flatten()(maxp_4) # matrix to single dimention vectors
```

Fully Connected Layers

The flattened features are passed through fully connected layers for high-level representation learning.

Listing 12: Fully Connected Layers

```
  # fully connected layers
  dense_1 = Dense(256, activation='linear')(flatten)
  dropout_1 = Dropout(0.3)(dense_1)
```

The Dense layer aggregates global features then Linear activation preserves numeric scale and Dropout reduces overfitting during training

Embedding Layer (Core Feature Representation)

A dense embedding layer of 128 neurons captures compact age-related facial features.

Listing 13: Flatten Layers

```
embedding = Dense(128, activation='relu', name='embedding')(
   dropout_1)
```

This produces a 128-dimensional latent vector then this layer is later reused as a feature extractor that is central to relative age comparison

3.3 Output Layer (Age Regression Head)

The final output layer consists of a single dense neuron with ReLU activation, which performs age regression during training. This output is used only to guide the learning of age-aware features and is not used for final inference in the relative age comparison task.

Architecture Visualization

The model architecture is visualized to understand layer connectivity and parameters.

Listing 14: Flatten Layers

```
keras.utils.plot_model(
model,
to_file="model_architecture.png",
show_shapes=True,
show_layer_names=True
)
```

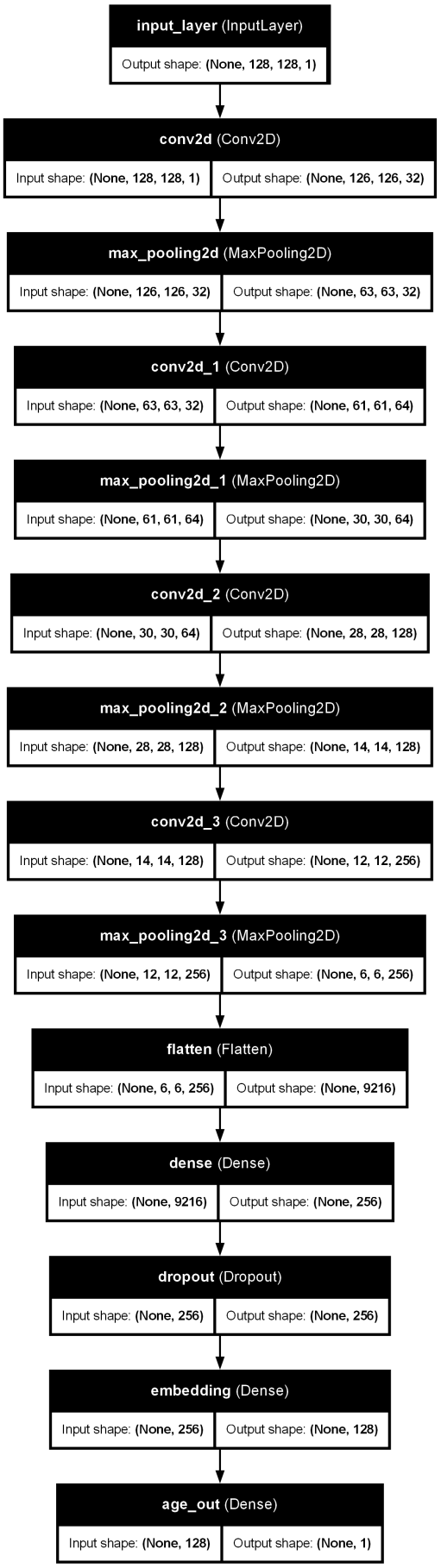

Figure 2: Architecture Visualization

4 Model Training

The training procedure is carried out in two stages. First, a convolutional neural network (CNN) is trained for age regression to learn age-aware facial representations. Subsequently, a comparison network is trained on pairs of learned embeddings to perform relative age estimation.

4.1 CNN Training for Age Regression

The CNN is compiled using the **Huber loss function** and optimized using the **Adam optimizer** . A validation split is used to monitor the generalization performance during training.

Listing 15: CNN compilation and training for age regression

```
model.compile(loss= tf.keras.losses.Huber(delta=5.0), optimizer='
    adam')

  history = model.fit(
  X,
  y_age,
  batch_size=32,
  epochs=15,
  validation_split=0.2
  )
```

The model is trained for a fixed number of epochs with an 80–20 training-validation split. Training and validation loss values are recorded to analyze convergence behavior.

4.2 Saving and Reloading the Model

After training, the CNN model and training history are saved to disk. This allows the model to be reloaded in subsequent runs without retraining, ensuring reproducibility and efficient experimentation.

Listing 16: Save Model History

```
model.save(MODEL_PATH)
np.save(HISTORY_PATH, history.history)
```

If a trained model already exists, it is loaded directly instead of retraining.

4.3 Feature Extractor Construction

Once training is complete, the CNN is converted into a feature extractor by removing the age regression head. The output of the intermediate embedding layer is used as the final representation of each facial image.

Listing 17: Feature extractor

```
feature_extractor = Model(
inputs=model.input,
outputs=model.get_layer("embedding").output)
```

This transformation allows the CNN to map each face image to a fixed-length latent feature vector encoding age-related information.

4.4 Training the Pairwise Comparison Network

To perform relative age estimation, embeddings from two images are concatenated and used as input to a multilayer perceptron (MLP). This comparison network is trained as a **binary classifier** using binary cross-entropy loss.

Listing 18: Comparison of input

```
  cmp_model.compile(
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.003),
  loss='binary_crossentropy',
  metrics=['accuracy']
  )

  cmp_model.fit(
  E_pairs,
  y_rel,
  epochs=10,
  batch_size=32,
  validation_split=0.2
  )
```

The output of this network indicates which of the two input images corresponds to the younger individual.

4.5 Training Monitoring

Training performance is visualized by plotting the training and validation loss curves, providing insight into convergence and potential overfitting

Listing 19: Plotting loss

```
if history_dict is not None:
plt.figure(figsize=(6, 4))
plt.plot(history_dict["loss"], label="Training Loss")
plt.plot(history_dict["val_loss"], label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.title("Training vs Validation Loss")
plt.legend()

plt.savefig("training_loss.png", dpi=300, bbox_inches="tight")
plt.show(block=False)
```

Key Training Parameters

- Optimizer: Adam
- Loss (CNN): Huber loss
- Loss (Comparison Network): Binary cross-entropy

- Batch size: 32
- Validation split: 20%

Training and validation loss curves for the age regression CNN, showing effective convergence and controlled overfitting.

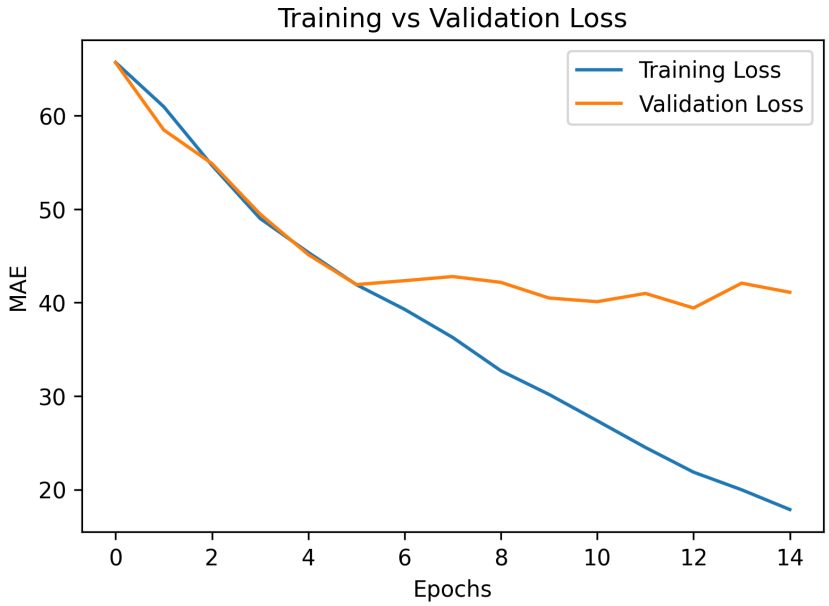

Figure 3: Plot Training Loss

5 Prediction and Relative Age Visualization

After training the age regression and comparison models, a qualitative evaluation is performed by randomly selecting two facial images from the dataset and comparing their ages. This step demonstrates the complete inference pipeline of the proposed system.

5.1 Random Sample Selection

Two distinct samples are randomly selected from the dataset to ensure unbiased evaluation. Each selected image has already undergone preprocessing, including grayscale conversion, resizing to 128×128 pixels, and normalization.

Listing 20: Image selection

```
image_index_1,image_index_2 = np.random.choice(len(X), size=2,
    replace=False)

img1 = X[image_index_1]
img2 = X[image_index_2]
```

5.2 Ground-Truth Age Retrieval

The original age labels corresponding to the selected images are retrieved from the dataset. These ground-truth values are used only for reference and visualization purposes and do

not influence the model's relative age decision.

Listing 21: Original Age

```
true_age_1 = y_age[image_index_1]
true_age_2 = y_age[image_index_2]
```

5.3 Absolute Age Prediction

The trained CNN regressor predicts the absolute age for each selected image. Since the model expects batched input, each image is expanded along a new axis before inference. The predicted ages are continuous values and are later displayed for comparison with the ground-truth ages.

Listing 22: Predict Age

```
pred_age_1 = model.predict(img1[np.newaxis, ...])[0][0]
pred_age_2 = model.predict(img2[np.newaxis, ...])[0][0]
```

5.4 Feature Embedding Extraction

Instead of directly relying on absolute age predictions for comparison, the model extracts age-aware feature embeddings from the intermediate embedding layer. Each image is mapped to a fixed-length latent representation that captures discriminative age-related facial characteristics.

Listing 23: Embedding

```
e1 = feature_extractor.predict(img1[np.newaxis, ...])
e2 = feature_extractor.predict(img2[np.newaxis, ...])
```

5.5 Relative Age Comparison

The embeddings of the two images are concatenated to form a single feature vector, which is then passed to a separate comparison network. This network outputs a probability score indicating whether the first image corresponds to a younger individual than the second. A threshold of 0.5 is used to convert this probability into a binary decision.

Listing 24: Age Comparison

```
pair_emb = np.concatenate([e1, e2], axis=1)
rel = cmp_model.predict(pair_emb)[0][0]
```

- Concatenates embeddings into one feature vector:
  - `e1` shape: $(1, 128)$
  - `e2` shape: $(1, 128)$
  - `pair_emb` shape: $(1, 256)$
- `cmp_model` outputs a probability `rel` $\in [0, 1]$.

Interpretation

- `rel` $> 0.5 \Rightarrow$ the model predicts the **left image is younger than the right**
- `rel` $\leq 0.5 \Rightarrow$ the model predicts the **right image is younger than the left**

Listing 25: Young or Old decision

```
younger = "Left image" if rel > 0.5 else "Right image"
older   = "Right image" if rel > 0.5 else "Left image"
```

5.6 Visualization of Results

The two images are displayed side-by-side for visual interpretation. Each image is annotated with:

- A top label indicating whether the subject is classified as *Younger* or *Older*
- The original age from the dataset
- The predicted age estimated by the regression model

This visualization provides an intuitive understanding of both the absolute age prediction accuracy and the relative age comparison performance.

Listing 26: Side-by-side visualization of image pairs with age annotations

```
plt.figure(figsize=(8, 4))

# Left image
ax1 = plt.subplot(1, 2, 1)
ax1.imshow(img1.squeeze(), cmap="gray")
ax1.axis("off")

ax1.text(
0.5, 0.95,
"Younger" if younger == "Left image" else "Older",
transform=ax1.transAxes,
ha="center",
va="top",
fontsize=12,
fontweight="bold",
color="yellow",
bbox=dict(facecolor="black", alpha=0.6, pad=4)
)

ax1.text(
0.5, -0.10,
f"Original Age: {true_age_1}",
transform=ax1.transAxes,
ha="center",
va="top",
fontsize=10,
```

```
    color="black"
    )

    ax1.text(
    0.5, -0.20,
    f"Predicted Age: {pred_age_1:.1f}",
    transform=ax1.transAxes,
    ha="center",
    va="top",
    fontsize=10,
    color="blue"
    )

    # Right image
    ax2 = plt.subplot(1, 2, 2)
    ax2.imshow(img2.squeeze(), cmap="gray")
    ax2.axis("off")

    ax2.text(
    0.5, 0.95,
    "Younger" if younger == "Right image" else "Older",
    transform=ax2.transAxes,
    ha="center",
    va="top",
    fontsize=12,
    fontweight="bold",
    color="yellow",
    bbox=dict(facecolor="black", alpha=0.6, pad=4)
    )

    ax2.text(
    0.5, -0.10,
    f"Original Age: {true_age_2}",
    transform=ax2.transAxes,
    ha="center",
    va="top",
    fontsize=10,
    color="black"
    )

    ax2.text(
    0.5, -0.20,
    f"Predicted Age: {pred_age_2:.1f}",
    transform=ax2.transAxes,
    ha="center",
    va="top",
    fontsize=10,
    color="blue"
    )

    plt.tight_layout()
```

```
78  plt.show()
```

Plot Results:

Figure 4: Architecture Visualization

Listing 26 illustrates the procedure used to visualize the age comparison results for an image pair.
A figure of size 8×4 is created to display both facial images side by side in a single row. The function `subplot(1, 2, i)` is used, where $i \in \{1, 2\}$ selects the left and right images, respectively. Each image is rendered in grayscale using `cmap="gray"`. The `squeeze()` operation removes the singleton channel dimension from the image tensor to enable correct visualization.
For each image, a top-centered annotation indicates whether the model predicts the image as *Younger* or *Older*. The annotation is drawn using normalized axis coordinates (`transAxes`) and enclosed in a semi-transparent black bounding box to ensure readability across different backgrounds.
Additional annotations are placed below each image to display both the ground-truth age and the predicted age. The predicted age is formatted to one decimal place to provide a concise numerical estimate. These values facilitate qualitative assessment of the regression accuracy.
Finally, `tight_layout()` is applied to prevent overlap between text elements, and `show()` renders the final visualization.
This evaluation step integrates age regression, feature extraction, and pairwise comparison into a single inference pipeline. By combining quantitative predictions with qualitative visualization, the effectiveness of the proposed age comparison framework can be clearly demonstrated.

References

[1] GeeksforGeeks, *Age and Gender Prediction using CNN*, Online article, 2023. https://www.geeksforgeeks.org/deep-learning/age-and-gender-prediction-using-cnn/

[2] YouTube, *Age and Gender Prediction using CNN (Tutorial Video)*, Video tutorial, 2021. https://www.youtube.com/watch?v=vEJzsGXrB7O