TRINETRA DETAILS

---

1. `siameese.py` (Training a Siamese Neural Network for Face Recognition)

Overview

This script trains a **Siamese Neural Network** to recognize faces by learning a similarity function. It uses **contrastive loss** to distinguish between similar and dissimilar faces.

Code Breakdown

```
import utils
import numpy as np
from keras.layers import Input, Lambda
from keras.models import Model
```

- `utils`: A helper module that contains utility functions like loading data, defining the network, and computing the loss.
- `numpy`: Used for numerical operations.
- `Input, Lambda`: Keras layers to define input tensors and apply custom functions (like Euclidean distance).
- `Model`: Keras model class used to create the Siamese network.

```
faces_dir = 'dataset/'
```

- **Defines the dataset directory** where face images are stored.

```
(X_train, Y_train), (X_test, Y_test) = utils.get_data(faces_dir)
num_classes = len(np.unique(Y_train))
```

- Calls `utils.get_data(faces_dir)` to **load the training and testing data**.
- **`X_train`**: Training images.
- **`Y_train`**: Training labels (person identity).
- Y_train = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, ...]#ylabel possibly look like this
- **`X_test, Y_test`**: Testing data.
- **`num_classes`**: Number of unique people in the dataset.

---

input_shape = X_train.shape[1:]
shared_network = utils.create_shared_network(input_shape)

- **Defines the input shape** based on training data.
- Calls `utils.create_shared_network(input_shape)`, which builds the **Siamese network's shared feature extractor**.

---

input_top = Input(shape=input_shape)
input_bottom = Input(shape=input_shape)

- Creates **two input layers** for the two face images being compared.

---

output_top = shared_network(input_top)
output_bottom = shared_network(input_bottom)

- Both inputs are **processed by the shared network**, generating embeddings.

---

distance = Lambda(utils.euclidean_distance, output_shape=(1,))([output_top, output_bottom])

- **Computes the Euclidean distance** between the embeddings of the two face images.

---

model = Model(inputs=[input_top, input_bottom], outputs=distance)

- Defines the **Siamese model** with two inputs and a single output (distance score).

---

training_pairs, training_labels = utils.create_pairs(X_train, Y_train, num_classes=num_classes)

- Calls `utils.create_pairs` to **generate pairs of similar and dissimilar images** for training.

---

model.compile(loss=utils.contrastive_loss, optimizer='adam', metrics=[utils.accuracy])

- Uses **contrastive loss** (from `utils.contrastive_loss`) to **minimize distance for similar images** and **maximize it for different images**.
- Uses **Adam optimizer**.

---

model.fit([training_pairs[:, 0], training_pairs[:, 1]], training_labels, batch_size=128, epochs=10)

- Trains the model on the **paired face images** for **10 epochs**.

---

model.save('siamese_nn.h5')

- **Saves the trained model**.

---

2. `recoggggg1.py` (Face Recognition with Multiple Cameras & Email Alerts)

Overview

This script:

- Captures **live video from 4 cameras**.
- Uses the **Siamese network to recognize a specific person**.
- Sends an **email alert** if the person is recognized.

Code Breakdown

import os
import cv2
import utils
import numpy as np
import face_detection
import time

import threading
import subprocess

- `os, cv2, numpy`: Standard libraries for image processing.
- `face_detection`: Detects faces in camera frames.
- `threading`: Handles real-time processing.
- `subprocess`: Calls `face_verify.py` for secondary verification.

---

from keras.models import load_model

- Loads the **trained Siamese network model**.

---

model = load_model('siamese_nn.h5', custom_objects={'contrastive_loss': utils.contrastive_loss, 'euclidean_distance': utils.euclidean_distance})

- Loads the **pretrained model** with custom loss functions.

THRESHOLD = 0.6
RECOGNITION_INTERVAL = 2 # Run every 2 sec
last_recognition_times = [0] * 4

- **Threshold** determines whether a face matches.
- Ensures recognition **runs every 2 seconds per camera**.

---

cameras = [cv2.VideoCapture(i) for i in range(4)]

- Opens **4 camera feeds**.

```
def update_camera_feed():
    for i, cam in enumerate(cameras):
        ret, frame = cam.read()
        if ret:
            process_frame(frame, i)
```

- **Continuously processes frames from each camera**.

---

def process_frame(frame, cam_number):
    small_frame, faces, face_coords_list = face_detection.detect_faces(frame)

- **Detects faces** in the camera feed.

---

similarity = 1 - model.predict([true_img, face_gray])[0][0]

- **Computes similarity** between the detected face and the reference image.

---

if highest_similarity >= THRESHOLD:
    recognized_image_path = f"recognized_faces/face_{int(time.time())}.jpg"
    cv2.imwrite(recognized_image_path, face_crop)
    face_queue.put((recognized_image_path, cam_number))

- **If the similarity is high**, saves the recognized face and sends it for verification.

---

def secondary_face_verification(image_path):
    result = subprocess.run(["python", "face_verify.py", image_path, "true_img.png"],
capture_output=True, text=True)
    return int(result.stdout.strip()) if result.stdout.strip().isdigit() else -1

- Calls `face_verify.py` to **confirm face match**.

---

def send_email(recognized_image_path, cam_number, recipient=r):

- Sends an **email notification** with the recognized face image.

---

3. `face_verify.py` (Face Matching using HOG & OpenCV)

Overview

This script:

- Loads two images.
- Detects faces using **HOG** and **OpenCV Haarcascade**.
- Encodes faces using `face_recognition`.
- **Computes similarity**.

Code Breakdown

```
import face_recognition as fr
import cv2
import numpy as np
import sys
```

- Uses **`face_recognition`** for encoding and comparison.

```
def enhance_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    equalized = cv2.equalizeHist(gray)
    return cv2.cvtColor(equalized, cv2.COLOR_GRAY2RGB)
```

- **Enhances contrast** for better face detection.

```
def detect_faces_with_opencv(image_path):
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
```

- Uses **OpenCV Haarcascade as a backup** if HOG fails.

```
faceLocOne = fr.face_locations(RgbFaceOne, model="hog")
faceLocTwo = fr.face_locations(RgbFaceTwo, model="hog")
```

- Detects faces using **HOG**.

```
encodingsOne = fr.face_encodings(RgbFaceOne, [faceLocOne])
encodingsTwo = fr.face_encodings(RgbFaceTwo, [faceLocTwo])
```

- Encodes detected faces.

```
face_distance = fr.face_distance([faceOneEnco], faceTwoEnco)[0]
threshold = 0.5
MatchResult = int(face_distance < threshold)
print(MatchResult)
```

- Computes **distance between face encodings**.
- **Threshold = 0.5** (lower means a closer match).

Final Thoughts

- `siameese.py` → **Trains the model**.
- `recogggg1.py` → **Runs live recognition with multiple cameras & emails**.
- `face_verify.py` → **Performs secondary verification using facial embeddings**.

This `util.py` script contains **helper functions** for training and using a **Siamese neural network** for face recognition. It includes functions for computing **Euclidean distance**, **contrastive loss**, **accuracy calculation**, **dataset processing**, and **model architecture** creation.

4.UTILS.py

Importing Required Libraries

```
import numpy as np
import random
import os
import cv2
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing.image import img_to_array
```

from tensorflow.keras.backend import sum as K_sum, square as K_square, sqrt as K_sqrt, maximum as K_maximum, epsilon as K_epsilon, mean as K_mean, equal as K_equal, cast as K_cast

Explanation

- `numpy` → Used for numerical operations (arrays, matrix computations).
- `random` → Helps in generating random negative samples when creating pairs.
- `os` → Used for file operations like accessing dataset directories.
- `cv2` **(OpenCV)** → Handles image processing (resizing, grayscale conversion).
- `tensorflow.keras` → Used for defining the neural network.
- **Keras backend (**`tensorflow.keras.backend`**)** → Used for tensor computations required for loss and accuracy functions.

Computing Euclidean Distance

```
def euclidean_distance(vectors):
    vector1, vector2 = vectors
    sum_square = K_sum(K_square(vector1 - vector2), axis=1, keepdims=True)
    return K_sqrt(K_maximum(sum_square, K_epsilon()))
```

Explanation

- Computes **Euclidean distance** between two feature vectors:
  d=∑(A−B)2d = \sqrt{\sum (A - B)^2} where:
  - AA and BB are feature vectors of two images.
- `K_square(vector1 - vector2)`: Computes squared difference.
- `K_sum(..., axis=1, keepdims=True)`: Sums over feature dimensions.
- `K_sqrt(...)`: Computes square root.
- `K_maximum(sum_square, K_epsilon())`: Ensures numerical stability.

Contrastive Loss Function

```
def contrastive_loss(Y_true, D):
    margin = 1
```

```
    return K_mean(Y_true * K_square(D) + (1 - Y_true) * K_maximum((margin -
D), 0))
```

Explanation

- **Purpose**: Helps the Siamese network differentiate between similar and dissimilar pairs.
- **Equation**: L=(1−Y)12D2+(Y)12max(0,m−D)2L = (1-Y) \frac{1}{2} D^2 + (Y) \frac{1}{2} \max(0, m-D)^2
  - If images **are the same** (`Y_true=1`), it **minimizes distance**.
  - If images **are different** (`Y_true=0`), it **increases distance up to a margin (m=1)**.

Accuracy Metric

```
def accuracy(y_true, y_pred):
    return K_mean(K_equal(y_true, K_cast(y_pred < 0.5, y_true.dtype)))
```

Explanation

- Compares `y_pred` (predicted distance) with **threshold = 0.5**.
- If distance **< 0.5**, the model classifies as the **same person**.
- Uses `K_equal` and `K_cast` to compare predictions with true labels.
- Computes mean accuracy across the dataset.

Creating Pairs for Training

```
def create_pairs(X, Y, num_classes):
    X = np.array(X)  # Ensure X is a NumPy array
    Y = np.array(Y)  # Ensure Y is a NumPy array
    pairs, labels = [], []
    class_idx = [np.where(Y == i)[0] for i in range(num_classes)]
    min_images = min(len(class_idx[i]) for i in range(num_classes)) - 1

    for c in range(num_classes):
        for n in range(min_images):
            # Positive pair
            pairs.append((X[class_idx[c][n]], X[class_idx[c][n + 1]]))
```

```
        labels.append(1)

        # Negative pair
        neg_list = list(range(num_classes))
        neg_list.remove(c)
        neg_c = random.choice(neg_list)
        pairs.append((X[class_idx[c][n]], X[class_idx[neg_c][n]]))
        labels.append(0)

    return np.array(pairs), np.array(labels)
```

Explanation

- **Purpose**: Creates pairs of similar (`label=1`) and dissimilar (`label=0`) face images.
- **Process**:
  1. Groups all images by **class/identity**.
  2. Creates **positive pairs** (same person).
  3. Selects **negative pairs** by randomly picking a different class.
  4. Returns pairs as **NumPy arrays** for training.

Creating the Shared Convolutional Network

```
def create_shared_network(input_shape):
    model = Sequential(name='Shared_Conv_Network')
    model.add(Conv2D(64, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D())
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Flatten())
    model.add(Dense(128, activation='sigmoid'))
    return model
```

Explanation

- **Purpose**: Creates a **feature extractor** (CNN) for face embeddings.
- **Architecture**:
  1. **64 filters, 3×3 Conv, ReLU** → Extracts patterns from images.
  2. **MaxPooling** → Reduces size to retain important features.
  3. **Another Conv layer** → Refines feature extraction.

4. **Flatten + Dense (128, sigmoid)** → Outputs a **128-dimensional embedding**.

Loading and Processing Dataset

```
def get_data(dir, img_size=(100, 100)):
    X_train, Y_train = [], []
    X_test, Y_test = [], []

    subfolders = sorted([file.path for file in os.scandir(dir) if file.is_dir()])

    for idx, folder in enumerate(subfolders):
        for file in sorted(os.listdir(folder)):
            img_path = os.path.join(folder, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # Read as
grayscale
            if img is None:
                print(f"Warning: Unable to load {img_path}")
                continue
            img = cv2.resize(img, img_size)  # Resize to fixed size
            img = img.astype('float32') / 255.0  # Normalize
            img = np.expand_dims(img, axis=-1)  # Add channel dimension

            if idx < 35:
                X_train.append(img)
                Y_train.append(idx)
            else:
                X_test.append(img)
                Y_test.append(idx - 35)

    X_train = np.array(X_train, dtype=np.float32)
    X_test = np.array(X_test, dtype=np.float32)
    Y_train = np.array(Y_train, dtype=np.int32)
    Y_test = np.array(Y_test, dtype=np.int32)

    return (X_train, Y_train), (X_test, Y_test)
```

Explanation

- **Loads face images from the dataset folder**.
- **Converts images to grayscale** for simplicity.

- **Normalizes pixel values** (0-1 range).
- **Splits into training (`<35 identities`) and testing (`>35 identities`) sets**.

Writing Text on Video Frame

def write_on_frame(frame, text, text_x, text_y):
    (text_width, text_height), _ = cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, thickness=2)
    box_coords = ((text_x, text_y), (text_x + text_width + 20, text_y - text_height - 20))
    cv2.rectangle(frame, box_coords[0], box_coords[1], (255, 255, 255), cv2.FILLED)
    cv2.putText(frame, text, (text_x, text_y - 10), cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(0, 0, 0), thickness=2)
    return frame

Explanation

- **Draws a rectangle & text overlay** on a video frame.
- Useful for **displaying face recognition results** in real-time.

Final Summary

- `util.py` provides essential functions for:
    - **Computing distances** (`euclidean_distance`).
    - **Loss function** (`contrastive_loss`).
    - **Generating face pairs** (`create_pairs`).
    - **Defining the CNN feature extractor** (`create_shared_network`).
    - **Loading and preprocessing images** (`get_data`).

Would you like a more detailed explanation on any specific function? 🚀

ECULEDIAN DISTANCE AND CONTRASTIVE LOSS

Let's break it down with a **real-world example** to make it more intuitive.

Example Scenario: Identifying Faces Using Siamese Network

Imagine you are building a face recognition system using a **Siamese Neural Network**. The goal is to compare two face images and determine if they belong to the same person or not.

Step 1: Extracting Feature Vectors

You have two images:

- **Image A**: A picture of "John"
- **Image B**: A second picture that may be of "John" or a different person.

Both images go through the **same** neural network (shared weights), which extracts a feature vector for each image.

Step 2: Compute Euclidean Distance

The **Euclidean distance** between the feature vectors of Image A & Image B is calculated.

- If both images are **of the same person (John & John)** → Feature vectors should be **very similar**, meaning the Euclidean distance **should be small** (close to 0).
- If the images are **of different people (John & David)** → Feature vectors should be **very different**, meaning the Euclidean distance **should be large**.

Step 3: Apply Contrastive Loss

The **contrastive loss function** ensures the network learns the correct distances.

Scenario	Expected Dist	Contrastive Loss Action

	ance	
Same person (John & John)	**Small** (close to 0)	If distance is **large**, penalize it! (reduce the distance)
Different persons (John & David)	**Large** (high value)	If distance is **small**, penalize it! (increase the distance)

How Contrastive Loss Works in Training

Now, let's consider actual numbers:

Case 1: Same Person (John & John)

- Suppose the network computes a Euclidean distance of **1.5** (it should be closer to 0).
- **Contrastive loss will penalize this!**
  → It tells the model: "Make the distance smaller when faces are the same."
  → On the next training iteration, the network adjusts its weights to **reduce the distance** for matching images.

Case 2: Different People (John & David)

- Suppose the network computes a Euclidean distance of **0.8** (it should be higher).
- **Contrastive loss will penalize this!**
  → It tells the model: "Increase the distance for different faces."

→ The network updates its weights so that next time, the distance between non-matching faces is **larger**.

Intuition Behind Contrastive Loss

Contrastive loss **teaches the network what "similar" and "different" look like**:

- If two images belong to the **same person**, the model should **pull** them closer in feature space (reduce distance).
- If two images belong to **different people**, the model should **push** them further apart (increase distance).

Over time, the network **learns a proper embedding space** where: ✅ **Same people have small distances**
❌ **Different people have large distances**

Key Takeaway

Contrastive loss does **not directly measure similarity**. Instead, it **teaches** the model to **adjust the similarity** correctly by penalizing wrong distances.