Computer Vision Project
Summer Term 2023
Florian Kordon, Vincent Christlein, Mathias Seuret,
Thomas Köhler, Tobias Koch, Mathias Zinnen

Sheet 4, starting from June 12, 2023, due July 3, 2023

The goal of this project is to develop a basic face recognition system. The system can be trained from facial data and is able to (re-)identify faces in video data. We cover techniques from *supervised* and *unsupervised* learning. The supplementary material contains a skeleton of that system along with training and test datasets. The system comprises two parts:

- **Training:** The training module (`training.py`) allows us to capture videos and to extract facial data (Exercise 4.1). Our training supports an identification mode to collect *labeled* training data (faces of known identity) for face identification (Exercise 4.2) and a clustering mode to acquire *unlabeled* data (faces without identity label) for re-identification (Exercise 4.3). After completing the training, the respective model (identification or clustering, depending on the mode) is stored to use it during testing. Figure 1 depicts training on an example face.
- **Testing:** The testing module (`test.py`) uses the trained models either for face identification (identification mode) or re-identification (clustering mode) and visualizes our results. Figure 2 depicts both tasks for an example video.

This exercise will guide you through the implementation of our face recognition system. Please refer to the documentation of the skeleton for additional hints and details. You may use the provided datasets extracted from the popular *YouTube Faces* database[1] to train and test the system. Notice that the test datasets contains faces of human subjects not included in the training set. In addition, you can capture a live video with a webcam and the OpenCV `VideoCapture` instance to evaluate the system.

Exercise 4.1: Face Detection, Tracking, and Alignment

We implement some preprocessing algorithms for video-based face recognition. This comprises detection, tracking, and alignment methods. The `face_detector` module contains a `FaceDetector` class for these tasks. The `detect_face` method determines the largest face bounding box in an image using the Multi-task Convolutional Neural Network (MTCNN) of Kaipeng Zhang[2]. The `align_face` method performs alignment by cropping and normalizing faces to a fixed size (224×224 pixel).

a) Tracking Faces in Videos

Face detection in videos could be performed frame-by-frame using MTCNN which is, however, computationally demanding. We can track face movements over time to accelerate the detection.

Implement the `track_face` method that takes an input frame and tracks the face position relative to an initial detection in a reference frame using *template matching*. In template matching, we locate the position of a template (i. e., face bounding box) in a new frame by searching for the position with maximum similarity to the reference for a given similarity measure.

1. Use the first frame of a video to detect a face using `detect_face`. Capture the detection as a reference for template matching.
2. Use template matching to track the face in new frames. Limit the search window to small

[1] https://www.cs.tau.ac.il/ wolf/ytfaces/
[2] MTCNN Python package available at `https://pypi.org/project/mtcnn/`

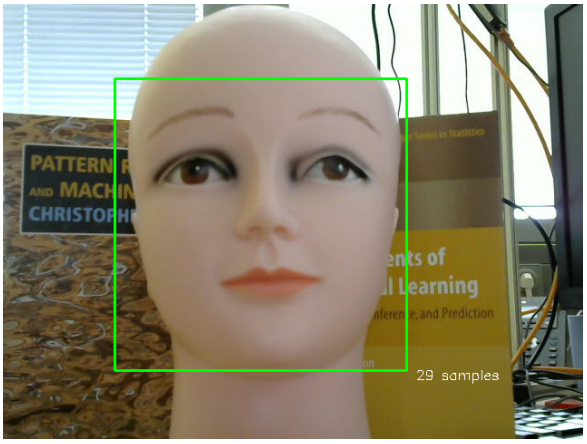

Figure 1: Training of our face recognition system (supporting face identification and clustering).

rectangular regions (e. g., reference bounding box ± 20 pixels at each side) centered at the face position in the reference to speed-up template matching under the assumption of small motions.

Hint: Use `matchTemplate` with `TM_CCOEFF_NORMED` similarity and `minMaxLoc` from OpenCV.

3. Template matching is sensitive to extreme head pose variations and requires a re-initialization if the track is lost. Identify some example limitations. Re-initialize your tracker using MTCNN and update the reference detection (replacing initial face detection by the new detection) if the template matching response falls below a certain threshold.

4. Test your tracker and adjust the parameters (window size and similarity threshold).

b) Alignment for Pose Normalization

We use an alignment step to normalize the sizes of detected face bounding boxes before extracting features from a face. Integrate the alignment to your tracking algorithm.

Exercise 4.2: Face Identification and Verification

Next, we develop *face identification* and *verification* using our detection, tracking, and alignment implemented in the first exercise. The `face_recognition` module contains `FaceNet` to extract deep features aka. *embeddings* from aligned faces using a convolutional neural network (CNN). The `FaceRecognizer` class provides functionality for predicting the identity of a subject in an image using these embeddings. The gallery is assembled by collecting and labeling faces in video frames.

We approach identification with a distance-based method in the space of face embeddings. Given two faces represented by the embeddings[3] $\mathbf{x}_1$ and $\mathbf{x}_2$, we define their distance in the Euclidean space by:

$$d(\mathbf{x}_1, \mathbf{x}_2) = ||\mathbf{x}_1 - \mathbf{x}_2||_2 \quad , \tag{1}$$

where we assume that $||\mathbf{x}_1||_2 = ||\mathbf{x}_2||_2 = 1$.

a) Extracting Deep Features

The `FaceNet` class extracts 128-dimensional embeddings from aligned faces using a ResNet-50 neural network architecture trained on the MS-Celeb-1M dataset followed by fine-tuning on the VGGFace2 dataset. It is parameterized by a ONNX model file describing the ResNet-50 along with its pre-trained

[3]Since all functions for face recognition are implemented in the space of embeddings, we use the terms *face* and *embedding* interchangeable in the remainder of the exercise.

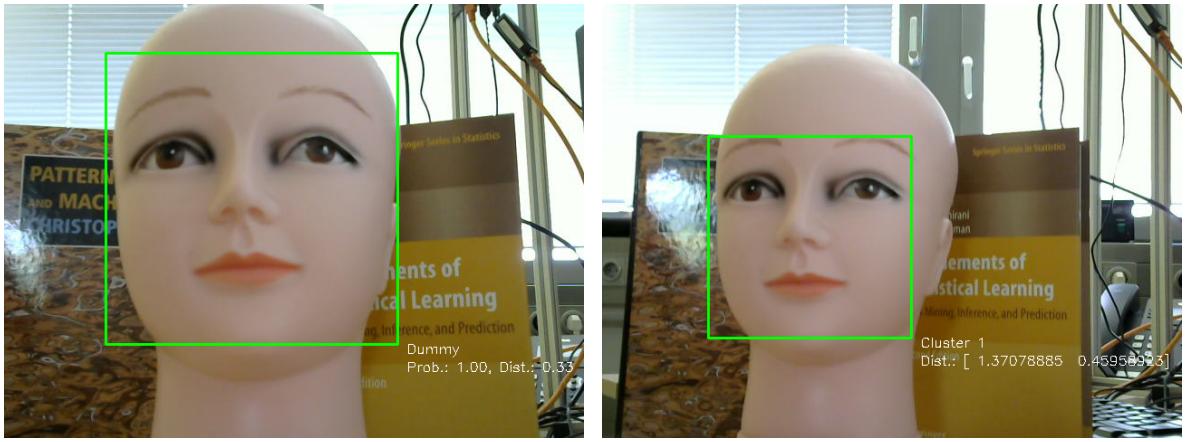

Figure 2: Face identification (left) and re-identification (right) for $k = 2$ classes/clusters. Identification aims at predicting the identity of a face using labeled training data (gallery). The posterior probability and the distance to a best matching gallery item are computed along with the predicted label. Re-identification retrieves the closest cluster using clustering of unlabeled faces. For a given face, the distribution of the distances to different clusters can be computed.

parameters[4]. The ONNX file is included using the OpenCV Deep Neural Network Module. Integrate the `FaceNet` class for feature extraction in your recognizer.

b) Closed-Set Protocol

First, we implement face identification under a closed-set protocol. In this situation, all identities (labels) that are used for testing are also contained in the gallery, i. e., they are known during training. We use k-nearest neighbor (k-NN) for classification.

1. Implement the `update` method that takes a new aligned face of known class from a video, extracts its embedding, and stores it as a training sample in the gallery.

2. Implement the `predict` method that assigns a class label to an aligned face using k-NN. That is, the label is determined by taking the majority of the k nearest neighbors in the embedding space. You can employ pair-wise distance calculations and brute-force search to implement k-NN.

3. Extend the `predict` method by returning the posterior probability $p(\mathcal{C}_i|\mathbf{x})$ that a face $\mathbf{x}$ falls into class $\mathcal{C}_i$, $i = 1, \ldots, C$ given C classes in the gallery. If $k_i \leq k$ neighbors among the k nearest neighbors belong to class $\mathcal{C}_i$, the posterior probability can be approximated by:

$$p(\mathcal{C}_i|\mathbf{x}) = \frac{k_i}{k} \quad . \tag{2}$$

4. Extend the `predict` method by returning the distance of the face $\mathbf{x}$ to the predicted class $\mathcal{C}_i$. Specifically, compute the distance $d(\mathcal{C}_i|\mathbf{x}) = \min(d(\mathbf{x}, \mathbf{x}_1), \ldots, d(\mathbf{x}, \mathbf{x}_{k_i}))$ of the $k_i \leq k$ neighbors $\mathbf{x}_1, \ldots, \mathbf{x}_{k_i}$ that belong to the predicted class $\mathcal{C}_i$.

5. Train the face identification with different persons. Test the identification and adjust the parameters of the k-NN classifier (number of neighbors k).

6. In addition to simple baseline tests (e. g., identification from frontal views), try to challenge your algorithm and investigate its robustness. Some examples are:

   - Profile views (yaw angle $\geq 30°$)
   - Variations in facial expression (mouth opening or occlusion of face regions)

[4]Trained models are publicly available at `https://github.com/ox-vgg/vgg_face2`

- Variations in illumination

c) Open-Set Protocol

In the open-set protocol, face identification can be confronted with faces of persons not contained in the gallery. We need to assign an "unknown" label to such faces instead of simply using the k-NN rule. To this end, the threshold nearest neighbor approach is used.

1. Implement the decision rule for open-set identification of a given face $\mathbf{x}$: If the distance $d(\mathcal{C}_i|\mathbf{x})$ is above the threshold τ_d *or* the posterior probability $p(\mathcal{C}_i|\mathbf{x})$ is below the threshold τ_p, predict "unknown". Otherwise predict the class label according to k-NN.

2. Test the face identification with several unknown persons not contained in the gallery. Try to find good thresholds for open-set identification.

3. Investigate the influence of the decision rule to detect unknowns. Is it sufficient to use the posterior probability or the distance threshold as a single criterion?

Exercise 4.3: Face Clustering

In this exercise, we investigate unsupervised techniques for face recognition. The goal is to enable a person re-identification instead of simply predicting an identity from a labeled gallery. We approach this task by clustering faces in videos according to their identity without manual labeling. This functionality should be provided by the `FaceClustering` class.

a) k-Means Clustering

The k-means clustering algorithm is one of the simplest methods to cluster feature vectors into $k \geq 2$ clusters. See the lecture slides for details on this algorithm. The clustering is performed in the space of embeddings predicted by `FaceNet`.

1. Implement the `update` method that extracts and stores an embedding for a new face. The update is similar to managing the gallery for face identification but does not include class labels.

2. Implement the k-means algorithm and integrate it to the `fit` method. For calculating the initial cluster centers, choose k embeddings of the input data at random. Store the estimated cluster centers and the labels assigned to the faces.

3. Depict the value of the k-means objective function over the iterations (use a diagram or table) and analyze the convergence behavior of your implementation on several test sets. How sensitive is the clustering with respect to the initialization?

b) Person Re-Identification

Once the clustering is done, we can re-identify a face by finding its best matching cluster. We develop the re-identification with a closed-set protocol.

1. Implement the `predict` method. To this end, compute the distribution of distances $\mathbf{d}_\mathbf{x} = (d(\mathbf{c}_1, \mathbf{x}), \ldots, d(\mathbf{c}_k, \mathbf{x}))^\top$, where $\mathbf{c}_i$ is the i-th cluster center and $\mathbf{x}$ is a face in embedding space.

2. The best matching cluster j has the smallest distance between its center and a given face, i. e. $j = \text{argmin}_i d(\mathbf{c}_i, \mathbf{x})$. Return the best matching cluster j and the distribution of distances $\mathbf{d}_\mathbf{x}$.

3. Collect facial data from $k \geq 2$ persons and test your re-identification algorithm.

Exercise 4.4: Evaluation of Face Recognition

This exercise aims at evaluating open-set face identification using detection and identification rate (DIR) curves. The DIR curve describes the performance of an identification system as a tradeoff between erroneously recognized *unknown* (false alarms) and the identification of *knowns*. The supplementary material contains a skeleton and data for this evaluation.

The files `evaluation_training_data.pkl` and `evaluation_test_data.pkl` form a training and a test dataset, respectively. Each dataset comprises embeddings and corresponding identity labels. The `dir_curve.py` script imports both datasets, feeds them into an `OpenSetEvaluation` instance, and plots a DIR curve from evaluation results.

This evaluation is conducted for a nearest neighbor classifier (`classifier.py`), which provides a `fit` method to learn a model from training data and a `predict_labels_and_similarities` method to classify test data. A similarity is obtained by negating the distance of a probe embedding to its nearest neighbor in the gallery (see lecture slides).

Your task is to complete the evaluation by implementing the `run` method.

1. Implement the `calc_identification_rate` method that takes a set of predicted class labels as input, compares them to the target labels, and computes the identification rate at rank 1.

2. Implement the `select_similarity_threshold` to determine a threshold from predicted similarities such that open-set identification on the test data yields a given false alarm rate.

   Hint: You can find the desired threshold by considering the p-percentile (`np.percentile`) of the similarities, where p is related to the false alarm rate.

3. Implement the `run` method according to the following algorithm:

   - Fit the classifier on the training data.
   - Predict similarities on the test data.
   - For each false alarm rate, find a similarity threshold that yields this false alarm rate on the test data and compute the corresponding identification rate.
   - Return all false alarm rates, identification rates, and similarity thresholds.

4. The DIR curve allows us to select parameters for a face identifications system. Find suitable similarity thresholds from the DIR curve under the following requirements:

   - The false alarm rate should not exceed 1 % and the identification rate should be maximized.
   - The identification rate should not fall below 90 % and false alarms should be minimized.

Exercise 4.5: Additional Exercise – Face Recogntion Challenge

This exercise poses a challenge on developing an own open-set face recognition method using learning with known classes (KCs) and known unknown classes (KUCs). We employ two of the open-set training strategies presented in the lecture: single pseudo label (SPL) and multi pseudo label (MPL). Both approaches can use a closed-set classifier as a backbone to form open-set models. For additional background please refer to the respective scientific publication[5].

The file `challenge_validation_data.csv` includes validation data with 128-dimensional feature vectors and class labels for 1680 subjects (610 KCs and 1070 KUCs). Class labels -1 denote KUCs and labels ≥ 0 are KCs. Notice that the underlying feature extraction method is hidden and no input images are provided.

We will develop open-set recognition based on `challenge_validation_data.csv`. You are free to adopt any estimator form sklearn as a backbone for SPL and MPL. Alternatively you can implement an own estimator. Feel free to use any additional pattern recognition tricks if desired, e.g., for preprocessing, feature transforms, etc. It is allowed to use the following Python dependencies: sklearn, numpy, scipy. Use the templates of `spl_training` and `mlp_trainingin` in `osr_learning.py`. Submit your `osr_learning.py` with the solutions of this exercise.

We will benchmark your submissions on a hidden test set based on the same feature extraction. The test set comprises images of the same 610 KCs and 1070 KUCs but captured under different intrinsic (pose, facial hair etc.) and extrinsic conditions (illumination, image quality, etc.). In addition, there are images of 4096 subjects not contained in the validation data referred to as unknown unknown classes (UUCs). Benchmarking will be done according to:

- DIR@FAR=1%
- DIR@FAR=10%
- Rank-1 identification rate
- Average computation time per sample to obtain a prediction
- Computation time to fit your models

The best, second-best, and third-best method per category will gain 3, 2, and 1 point, respectively. The overall score of your method will be done by computing the sum over all categories, i.e., you obtain a maximum of 15 points. SPL and MPL will be tested separately and there will be one winner per track (SPL and MPL).

1. Implement SPL and design a backbone classifier (function `spl_training`).

   Hint: If you decide to use a sklearn classifier as backbone, consider the optimization of its hyperparameters under the performance measures of this challenge. Avoid overfitting on your validation data since benchmarking will be done one a hidden test set.

2. Implement MPL and design a backbone classifier (function `mpl_training`). Notice that you can either re-use the same backbone as for SPL or use a different backbone.

   Hint: Not all classifiers can handle the increasing number of classes for MPL. Use a backbone that is computationally efficient for multi-class problems with large number of KCs und KUCs.

[5]Koch, T., Riess, C., Köhler, T. (2023). LORD: Leveraging Open-Set Recognition with Unknown Data. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 4386-4396).