EmotiVision: Emotion detection and gaze analysis of retrieved faces

Michele Giarletta, Vincenzo Lapadula, and Giacomo Salici
Course of Computer Vision and Cognitive Systems
Computer Engineering Master Degree
University of Modena and Reggio Emilia
{273839,267759,270385}@studenti.unimore.it

Abstract

The widespread adoption of intelligent assistants that has occurred in the last months will become more and more relevant and only possible by capturing environmental information. In our opinion, to be effective in human interactions, they need to be aware of people's feelings. Our work focuses on this topic, and it is structured in a three-step pipeline. Firstly, we have analyzed eyes and gazes by comparing and testing several approaches for both face detection and pupil localization. Having the pose estimation, we then analyzed the faces of the people facing the camera. Using a pre-trained retrieval network based on Inception-Resnet placed just after an MTCNN, we have been able to identify the faces of enrolled people which we know to be not interested in. Lastly, we created and trained using distillation a neural network based on a reduced VGG19 architecture that is able to detect up to seven emotions. Our results got an F1 score of 97%. Finally, we built a demo robot using Arduino to see our pipeline in action within a realistic scenario. The robot is able to track a person's face until he shifts his gaze away from it.

1. Introduction

In this computer vision project, we focused on building a pipeline made of three steps: gaze analysis, retrieval and emotion classification eventually placed in action on a demo robot built by us, as described in §3.
The first part of the pipeline (§4) is done using classical computer vision processing. We call it a gaze analysis, rather than detection or tracking, because it is actually composed of more steps, and for each of them, we compared different methods to get a more wide idea of what classical processing approaches can offer. We were indeed aware of deep methods more functional but for this part, we wanted to explore a not-deep solution (however, a comparison just for the face detection step has been made during the preprocessing of the second part). To get the gaze direction we first detect the faces, comparing a Haar cascade method with one based on HOG. Then we get a precise pupil localization, using both a method based on means of gradient and one made by us based on filters, which is very naive but surprisingly effective. Finally, we computed the result using PnP perspective, camera calibration, and some geometric manipulations.
In the second part (§5), we designed a highly customizable retrieval system that also includes a testing framework based on the LFW dataset, in this way, we managed to find the perfect combination of metrics, threshold values along with other hyperparameters. We managed to make comparisons between methods of classical image processing (seen in the first part) and deep approaches. For deep face detection and recognition, we used a very popular approach based on MTCNN and FaceNet which union effectively satisfies the tasks related to facial biometrics.
Finally, in the third part (§6), we trained our own CNN (FERNet). Since the network should have been used in real-time by a robot, it was necessary to consider the limited computational power. However, we decided to train a complex network (VGG19) that could guarantee excellent performance and, subsequently, we transferred its knowledge to a smaller one (FERNet) using distillation. By doing so, we managed to obtain a smaller network that has similar performance compared to the larger one. The codebase of the project has been released open-source at github.com/SLG-Vision/EmotiVision.

2. Related Works

Gaze Analysis In the literature, several face detection approaches are proposed. According to the literature, they can be differentiated into three different types of models. 1) Feature-based ones have the goal to detect distinguished features - like eyes and nose - and compute from their geometrical position if a face is present. 2) In template-based methods, the input is compared with an already present pattern using for instance an SVM. They require a good initialization near a real face and thus may be not the best for fast

face detection. Finally, 3) appearance-based methods deal with small rectangular-shaped patches that are overlapped on several windows of the input image, looking for the target. Many authors have proposed different approaches, and generally, a cascade is applied to refine the results, and a pyramid sub-octave is used to get a better scale invariance. We utilized one of the best-known methods appearance-based, from Viola and Jones [Viola and Jones, 2005] and we compared it with the more general HOG method [Dalal and Triggs, 2005]. A similar categorization can be seen also seen for the gaze estimation task: as highlighted by [Sugano et al., 2014], 1) model-based methods use a 3d eyeball model to estimate the gaze direction using geometric features and again, 2) appearance-based estimation makes instead use of non-geometric features and learn mapping functions. Moreover, a lot of methods, like [Cheng et al., 2021], rely on a deep network, but, as we have said in the introduction, we preferred a classical solution. We found very interesting the approach of [Khan and Lee, 2019] that applies gaze tracking to advanced driving assistance systems (ADAS). The authors analyzed the correlation between head pose and eye pose, and how much the gaze classification improves if both of those inputs are provided.

Retrieval Network During the solutions design phase we evaluated well-known approaches in the literature: from SVMs to deep models, we also evaluated a modern retrieval approach that has been shown to be effective for text retrieval but still applicable to images [Borgeaud et al., 2021]. This last approach would have allowed a higher quality of the classification, we had to discard this idea since in our case the classification task (mood detection) was in conflict with the retrieval task (identification). It would have been possible to insert a further step within the pipeline between the retrieval task and the mood detection one which would have retrieved the top-scoring N faces with the detected mood and then finally classify them all thus having a classification based on multiple samples hence an increased precision. The need to be able to run the pipeline in real time led us to reach a tradeoff in which we could not further burden the pipeline.

Emotion detection Facial expression recognition (FER) is a field that has significantly improved in recent years thanks to deep learning. Today it is applied in various areas such as marketing (customers' reaction to the products offered by the company to understand their preferences), safety (analysis of behavior or monitoring of driver fatigue while driving), mental health, robotics (to improve the interaction between robots and humans) and education (to monitor students' attention during lessons). The expressions recognized in the literature typically are anger, disgust, fear, happiness, sadness, and amazement.

Figure 1. Our naive robot. Note the stepper mode that is on the tripod. Both the stand for the motor and the boards' enclosure are 3d printed.

According to [Li and Deng, 2018] in literature, there are two approaches: static image FER and dynamic sequence FER. In the first one, the feature representation is strictly encoded with only spatial information from the sampled image while, the second one, considers the temporal relation among contiguous frames in the input sequence. We used a CNN, widely used in the literature, for the classification of the expression. We were inspired by [Hui Ding, 2016] that used a similar approach. During stage (a) they removed the Fully-Connected layer from EmotionNet in order to realize a sort of distillation, in particular, they used a frozen network, FaceNet, to improve the training of EmotionNet's convolutional layers.

3. Image acquisition and naive tracking

As we have described in the abstract, we strongly believe in the practical application of this project. For this reason, we considered applying this pipeline to the vision system of a robot that can move in a crowded environment. Detecting the gazes and emotions of the people in the surroundings is a non-trivial but essential task that a robot must perform in order to interact with them. Our initial idea was to test our work on one of the robots available at the university laboratory[1]. However, since it would have presented additional challenges beyond the scope of this project, we decided to implement a demo robot using Arduino, a webcam, and a 3D-printed enclosure designed by ourselves (see Figures 1 and 2). Although it is a very basic and experimental result that cannot physically move, its stepper motor allows it to track the face of the person looking at it. This is achieved by receiving the required rotation angle from the serial port to

[1] https://aimagelab.ing.unimore.it/imagelab/

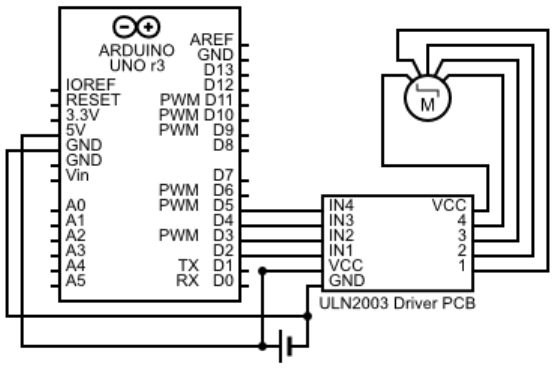


Figure 2. The schematics of our Arduino robot. The board receives the signal from the serial port, currently via the USB cable.

keep the face (specifically, the widest face among those detected) as centered as possible within the camera's field of view. If the camera does not detect any face, it remains idle, without moving. As soon as a face is detected, the motor starts rotating (±5° each frame) until the distance between the center of the face and the center of the camera view are less than a fixed threshold (150px). Further developments could implement more advanced tracking techniques, since as of now, we simply detect the faces (it will be discussed below) frame per frame and compute the alignment of the widest.

4. Gaze analysis using classical processing

We studied a process composed of four different steps: face detection, facial landmark detection, facial pose estimation, and finally precise pupil localization.

The facial pose estimation is needed to know if the person is facing the camera and, in that case, the pupil localization is computed. We applied a heuristic threshold to approximate whether the gaze is following the face direction. In the next paragraphs, we will discuss the choice of this implementation and why we don't compute the pose estimation directly from the eye.

4.1. Face detection

During the development of the first step of this part, we aimed to detect the faces present in the image stream. Initially, we used the Haar Cascade classifier that is based on the above-mentioned Viola-Jones [Viola and Jones, 2005]. The AdaBoost algorithm is applied over a set of Haar features, the differences between the sum of pixels within adjacent rectangular windows of the image scaled with several factors. We applied the *OpenCV* library and we tested it detecting faces in *Fer2013*. We also made the same test

Face detected	0	1	2	Time
Hog *dlib*	10932	24955	0	34.18 s
Haar Cascade *OpenCV*	15272	20608	7	38.38 s

Table 1. Comparison between two different classical face detection algorithms. They were tested on 35887 images containing a single image. They reach an accuracy of 0,70 and 0,57 respectively. Test made on a 2020 M1 MacBook Air.

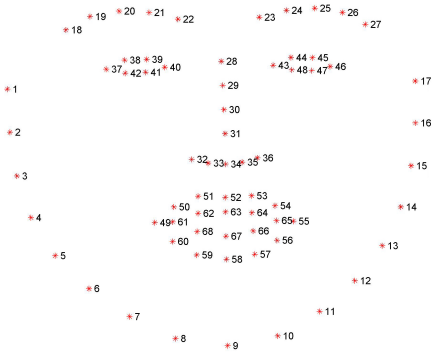


Figure 3. The 68 landmarks detected with *dlib*.

using the implementation of the histogram of oriented gradients by [Dalal and Triggs, 2005] present in *dlib*, another C++ library that offers several interesting machine learning and computer vision algorithms. The results of the second method were slightly better as shown in the table. Although it may not be the best testing scenario since the faces are often occluded and the test we made not measures important metrics like precision and recall, it has been a very quick way to verify how both methods perform under real-life scenarios. We had seen that *dlib* method handles better occlusions and it is rotation invariant, contrary to Viola-Jones' one.

4.2. Facial landmark detection

For what concern the second step of the process, we get the facial landmarks detection simply another *dlib* method. The algorithm is based on a paper from CVPR [Kazemi and Sullivan, 2014] and trained on *IBug300-W* [Sagonas et al., 2013] and it makes use of a cascade ensemble of regression trees to do shape invariant feature selection based on thresholding the difference of intensity values at the pixels level.

The *dlib* method finds out, with a low error, 68 landmarks present along the face, as shown in Figure 3. From there, we only select a few that we have used in the next steps.

4.3. Pose Estimation

Since the goal of this first part of the pipeline is to detect whether or not a person is facing the camera, we have to compute the facial pose estimation of the faces detected. We had to solve the so-called Perspective-n-Point (PnP) pose computation problem, computing the rotation and translation $[R|T]$ matrix given the homogeneous coordinates of the face projected on the $[u, v, 1]$ 2d plane, the 3d world face points p_w expressed in world coordinate system and the camera matrix K.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \ \Pi \ [R|T] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

If expanded, the right-hand side becomes the following, note that Π is the perspective projection model.

$$\begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

While the face points are given from the landmark detection at the previous points, the 3d world point is set using a standard face 3d model. For the best results, the 3d points could be measured directly on the target face, but since there is more than a single target, the model is perfectly fitted for a standard human face.

Camera Calibration For the camera calibration, we applied the chessboard method implemented in *OpenCV*, computing the intrinsic camera matrix K starting from several photos of a chessboard.

Rotation Matrix To obtain the actual rotation matrix R we used an iterative method based on a Levenberg-Marquardt optimization present in *OpenCV*. Then we applied a refinement method also present in the library to get a better prediction.

Having R, we finally found the pitch, roll, and yaw (see Figure 4) of the faces computing the Euler angles from the rotation matrix (get using the Rodrigues formula).

$$\text{Pitch} = \text{atan2}(r_{32}, r_{33})$$
$$\text{Yaw} = \text{atan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2})$$
$$\text{Roll} = \text{atan2}(r_{21}, r_{11})$$

Ideally, the face is facing the camera if all three Euler angles are 0. In the real-life scenario, because of the image noise and the not-always-perfect accuracy of the angles found, we had to put a threshold under which the face is considered to be facing. During the testing face, we noticed that a threshold of 20° is fine.

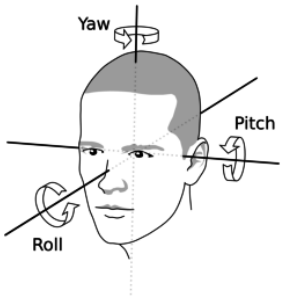


Figure 4. Pitch, roll, and yaw

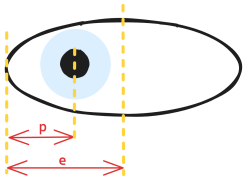


Figure 5. The horizontal pupil ratio hr can be found having the half length of the eye e and the distance of the pupil from the eye corner p. Then, $hr = (p/e) - 0.5$.

4.4. Precise eye center localization

If the face is not facing the camera but the person is still looking at it due to the eye movements, the system should still detect the gaze direction toward the camera. At first tries, we studied how to get a head-pose invariant gaze detection, trying to get the eye direction using directly the PnP estimation.

The main problem with this approach is that the majority of gaze estimators cited in the literature make use of a calibration procedure that measures pupil movement while the eyes are looking at dots on a screen. Other methods need more cameras or more lights (normals or IR). This is not feasible with our pipeline, in which the detection happens "in the wild" captured from a moving single camera.

To solve the problem we simply considered the pupil position (found using two different methods presented below) with respect to the eye corners.

The horizontal pupil ratio expresses how the pupil position within the eye, from -0.5 to 0.5, where 0.0 represent the position when the pupil is centered in the eye and 0.5 when the pupil is completely shifted towards the left corner (Figure 5).

Having the ratio we can compute:

- if the eye is gazing toward the same direction as the

face or,

- if the person is gazing at the camera when the face is facing somewhere else.

As you probably have imagined this method is not suitable for an accurate estimation of the second point since the direction of the eye is really biased from the distance between the person and the camera. The first point, on the other hand, can be solved quickly just by setting a threshold as we have done with the face facing.

Regarding the second point, we made a proof of concept setting some parameters that are corrected with a person standing around 0.5m far from the camera. See Figure 6 to look at the script in action.

4.4.1 Means of gradient

As the third step, we have read in detail the method proposed by [Timm and Barth, 2011]. This method lets us get the exact center of the pupil also in images with low resolution and bad lighting. The searched point can be found by comparing the gradient vector g_i at position x_i with the displacement vector of a possible center d_i.

$$c^* = \arg\min_c \left\{ \frac{1}{N} \sum_{i=1}^{N} w_c (d_i^\top g_i)^2 \right\},$$

$$d_i = \frac{x_i - c}{||x_i - c||_2}, \forall i : ||g_i||_2 = 1$$

This method requires an input of a cropped image of an eye, so we used the information from the second step to get the needed processing. The authors of the method suggest also a preprocessing and post-processing phase that have been done to obtain optimal results. Preprocessing is made with the weight w_c helps find the dark pupils since it is the grey value at (c_x, c_y) of the smoothed and inverted input image. A Gaussian filter also is needed to remove bright outliers like reflections of the glasses. Then as post-processing, a threshold has been applied to remove possible results connected to borders, like eyebrows, glasses, or hair. We then developed a script of the above-presented method, partially following an already existing work [Trishume, nd] and its Python implementation [abhisuri97, nd].

4.4.2 Filtering

The above-proposed method is very interesting and works very well in difficult conditions, but it is also very computationally intense. In real-time systems may not be feasible. So, we tried and implemented a simple but effective filtering approach that works very well and requires less effort. Having the cropped eye, we apply a series of filtering and preprocessing aimed to make stand out the pupil, that it is always the darker and more circular object in the eye. In particular, we increased the contrast of the image and we equalized it. Then we applied a soft Gaussian blur to remove the noise and an erosion filter that removes the smaller parts like the eyelashes. We then use the *OpenCV* adaptive thresholding method to binarize the image and lastly, we get the counters of all of the blob. Looking for the largest one we find the pupil. See the figure 7 for the application of the approach.

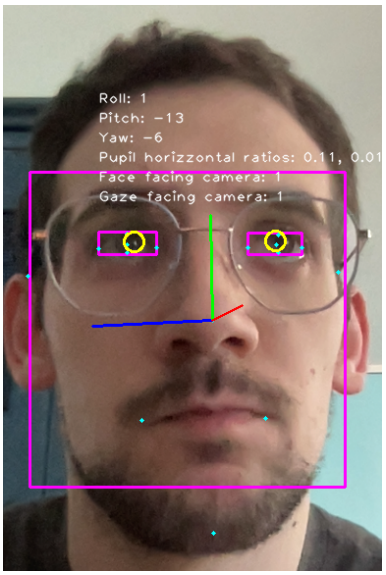


Figure 6. Results of the first part of the pipeline.

(a) Higher contrast and equalization. (b) Gaussian blur and erosion. (c) Adaptive thresholding.

Figure 7. Eye processing during filtering method. The figures show the processing during each partial step, please note that each of them takes as input the previous one's output image.

4.4.3 Testing

To compare the two methods, we run both of them over a dataset[2] containing around 1500 images of faces, all hand labeled with the exact position of the pupils. We measure the just-found pupils' positions by comparing them with the labels, computing the Hausdorff distance (described in [Jesorsky et al., 2001]) which is the maximum all the distances between each point of a set and the closest point in the other set. For each image of the dataset, we found the distances and then computed the mean.

$$H(A, B) = \max(h(A, B), h(B, A))$$

[2] https://www.bioid.com/facedb/

	Mean	Std. Dev.	Outliers	Avg. time
MoG	3.004	2.220	3	0.112
Filtering	3.198	1.156	3	0.028

Table 2. Test results of the pupil localization method. The time has been measured on a 2020 M1 MacBook Air.

$$h(A,B) = \max_{a \in A} \min_{b \in B} ||a - b||$$

The results are shown in Table 2. Please note that the mean and the standard deviation are done after removing outliers that were completely wrong ($\frac{x-\mu}{\sigma} > 5$). It is pretty straightforward to notice how the average time to compute the position is about 4 times longer with the means of gradients than with filtering, the means are quite the same and despite the fact that the first method is slightly more precise, it has also a higher deviation.

It has not been said that the first method is actually much more precise in low-light environments (contrary to the "easy" one of the dataset), but for situations like ours real-time detection filtering method works fine and faster so we have chosen to use that for the next steps.

5. Face retrieval

The core scope of this section is to build the face retrieval feature, in order to achieve valid and accurate face embeddings we first need to further elaborate our input image and only after building our embeddings database. After defining a distance metric we compare the given input face, achieved from real-time inference, against our blacklisted embeddings; thanks to a defined threshold we are eventually able to classify if a face is blacklisted or not.

5.1. Preprocessing

Some functionalities of this pre-processing part have also been developed in the classical image processing way; this allowed us to be able to isolate and understand how the deep networks we used, as the *InceptionResnet* [Szegedy et al., 2016] and the *MTCNN* actually worked. In this way, we have also been able to compare the *ConvNet* metrics against the classical one.

5.1.1 Face alignment

In this first step we wanted to achieve a standardized, deepnet-ready crop, aligned and landmarked image; we managed to achieve this through the [Zhang et al., 2016] Multi-Task Cascaded Convolutional Neural Networks method. This paper proposes a MTCNN made up of three main components each with its specific task, we can so define:

- Proposal Network (P-Net): The first neural network is designed to detect regions of the image that might contain faces. It is actually a fully connected network and thus does not have any fully connected at the end. Here we compute bounding box regressions, face classifications and landmark localizations with a high recall hence with many false positives.
- Refine Network (R-Net): The second neural network verifies whether the face-containing region proposals detected by the P-Net are indeed faces and also calibrates the bounding boxes. Eventually the network uses a non-maximum suppression algorithm to eliminate overlaps and retain only the best proposals of regions that contain faces.
- Output Network (O-Net): The third neural network aligns the face cropped by the R-Net to a standard coordinate system so that facial features such as the eyes, nose, and mouth are positioned coherently and comparably to each other. The O-Net produces precisely the five landmarks: left eye, right eye, nose, left mouth corner and right mouth corner.

5.1.2 Pre-whitening

Prewhitening transforms data such that the covariance between variables is reduced to zero, and the variance of each variable is normalized to one. We use this to normalize the pixel values of the image before it is processed by the final deepnet. This helps to reduce variations in lighting and contrast between images hence improving the ability of the model to distinguish between differences in image features.

At first, the mean is subtracted from the data, so that the data has a zero mean; second the standard deviation (and the variance as a consequence) is scaled to a unit.
This normalization is achieved by dividing each value by the adjusted standard deviation of the data. We also notice that before the division the data standard deviation is clamped with a lower bound of $\frac{1}{\sqrt{tensor.numel}}$ in this way we are also ensuring that we won't have a too small value by taking into account the tensor size in the clamp operation.

```
def prewhiten(x):
    mean = x.mean()
    std = x.std()
    std_adj = std.clamp(
        min=1.0/(float(x.numel()))**0.5))
    y = (x - mean) / std_adj
    return y
```

As example the tensor `[1, 2, 3]` would become `[-1, 0, +1]` or the tensor `[1, 2, 3, 4, 5]` would be `[-1.2649, -0.6325, 0.0000, 0.6325, 1.2649]` prewhitened.

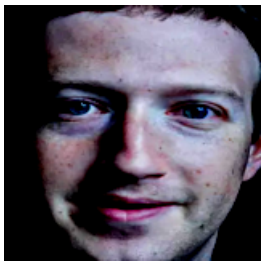


Figure 8. Prewhitened image

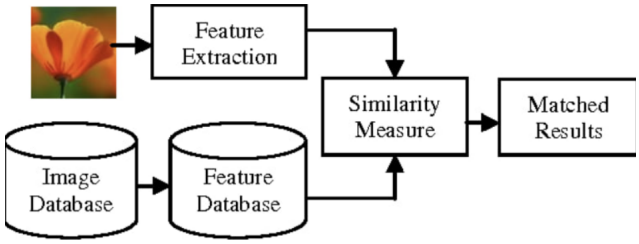


Figure 9. The retrieval core pipeline

5.2. Building a database of embeddings

A common retrieval system is used as shown in figure 9 along with 174 non-augmented pictures of one single person that compose our actual image database. We built a module that fetches all these images from a specific folder, and evaluates them with *InceptionResnet* [Szegedy et al., 2016] pre-trained on the *VGGFaces2* dataset (this same model and weights will be used during the real-time inference) and eventually builds the feature database. The *InceptionResnet* architecture was born as a combination of *Inception CNN* and *ResNet*. The idea behind Inception is to perform parallel convolutions of the input feature maps with filters of different sizes, and then concatenate the resulting feature maps into a single output. This allows the network to capture both local and global features of the input image thus increasing performances. Eventually combined with *ResNet*, proved useful to avoid gradient vanishing thanks to residual connections which allow us to have a deeper and more accurate network. This network outputs a `[1,512]` shape tensor, which will be eventually compared with a `[batch_size, 1, 512]` shape tensor.

The pre-trained ported weights are originally from the FaceNet authors [Schroff et al., 2015] which used an innovative loss function called triplet loss (see [1]), which has proved again the effectiveness of contrastive loss training. It makes use of an anchor, a positive sample, and a negative sample (figure 10).

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p))\|_2^2 - \|f(x_i^a) - f(x_i^n))\|_2^2 + \alpha \right]_+ \tag{1}$$

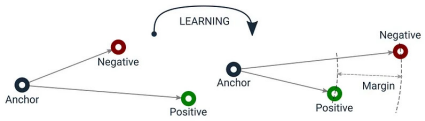


Figure 10. The triplet loss training schema

5.2.1 Data Augmentation

During the embedding database construction phase, we made it possible to increase the comparison dataset by simply activating a specific flag. It is, therefore, possible to pass the list of possible transformations or to use a list of recommended transformations, for each transformation we have defined non-extreme boundaries for random generation so as not to excessively change the range and domain compared to the original image. The final size of the comparison dataset, therefore, corresponds to:

$$\begin{aligned} Dataset\ Size = \\ augmented\ dataset\ size = samples\ number + \\ samples\ number * transformation\ number \\ * augmentation\ iter \end{aligned}$$

This happens because all cropped-only original frames are kept, then are transformed for all the transformations we chose but on top of this, this last step of transformations happens *augmentation_iter* times.

The 11 ready-to-use transformations are here reported:

- resize
- blur
- motion blur
- rotate
- flip
- brightness
- contrast
- saturation
- zoom
- tilt
- translate

During testing, we defined a suggested list of transformations that just removes the zoom and flip among these eleven since we found that these were too distorting in our case, eventually lowering the performances since in real-time applications we have a (1, N) comparison useless transformations must be avoided.

5.3. Real-time retrieval inference

We developed this step in a second module that evaluates the input image (same model seen in §5.2), achieving the face embedding, and also computes a distance metric. So far we have tried three different measurements: Euclidean distance, Cosine distance, and Manhattan distance. The last important step of this section is the threshold hyperparameter. The value has been chosen against the distance output, an optimal value is needed for correct face classification.

5.4. Results

To achieve a standard measure we built a 400 pictures dataset of true positives shuffled with LFW (Labeled Faces in the Wild) [Huang et al., 2007] dataset made of 13,233 images, making up our whole test set of which the original LFW images will be considered all true negatives now on.

We were therefore able to calculate the retrieval metrics and performance, managing to achieve a more than satisfactory hyperparameter configuration thanks to the important flexibility with which we designed this part.

The testing described here against this test dataset happened in an offline mode. In order to evaluate online real-time performances instead we used OpenCV timing functions thus obtaining the exact frame rate. This last approach however would have not allowed us to compute more fine metrics such as: precision, F1, recall, MTCNN errors etc..

Some of the tested approaches have been reported in a convenient flowchart 11, let's now analyze the results obtained and the possible choices that we have implemented in each block.

5.4.1 Building phase solutions

With regard to the embedding database building process, the implemented variations are the pre-trained weights with which the cropped faces are evaluated and whether or not to do data augmentation and if so, with what number of iterations.
The default parameter is zero augmentation (no augmentation) since we did not notice an improvement in results even if this may change if we want to blacklist more than one person along with the weights from VggFaces2 since changing to CASIA WF did not show any significant difference.

5.4.2 Inference Results

Here we will focus on the choices regarding the inference phase. The first choice we had to make, even before specifying what was to be done in each block, we had to decide the order of the blocks. By this we mean that we evaluated various cases: the first saw the use of classical processing only applying the same regularization transformations that we used during the construction phase of the embeddings database with the MTCNN in order to preserve the image domain, the second consisted of the full deep and eventually a hybrid approach with all possible permutations. Initially, the performance of the totally deep approach was the winner among the others, however, we subsequently managed to bring the only classic approach to the same performance as the first (check §7), this allowed an overall increase in performance given by a reduction in the computational load thus making the retrieval computation possible on the edge.

Regarding the *similarity function* block as we said we defined the L1 (Manhattan cityblock distance), L2 (Euclidean) and Cosine similarity functions:

$$Ł1(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{2}$$

$$Ł2(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3}$$

$$cosine(x, y) = \frac{x \cdot y}{\|x\|\|y\|} \tag{4}$$

Regarding the *element to group function* we defined: the **max** which defines the (1, N) distance as the distance between our input element and the most similar element in the group, this has proved to have a very high recall but a not great precision; the **median**, which defines the distance as the input element and the median distance value in the group; the **average** that has been our final choice which defines the distance as the one between the input and the average of the similarities in the group.

Most of the tests we have done have seen a single blacklisted face, in this situation in all the tests carried out (initially taking a dataset aimed at a single well-known and famous person and then creating a dataset depicting a member of the group) we have found the most performing function to perform an average. However, it is necessary to point out that in the tests in which you want to blacklist and therefore identify multiple faces and not just one, averaging brought poor results, the max or even better a median followed by an average was much more effective. We reported some of the results computed in the table 3, the final configuration we went for is the first one. Note that pre-whitening is not included in the table, but as described before, it has been applied in every scenario.

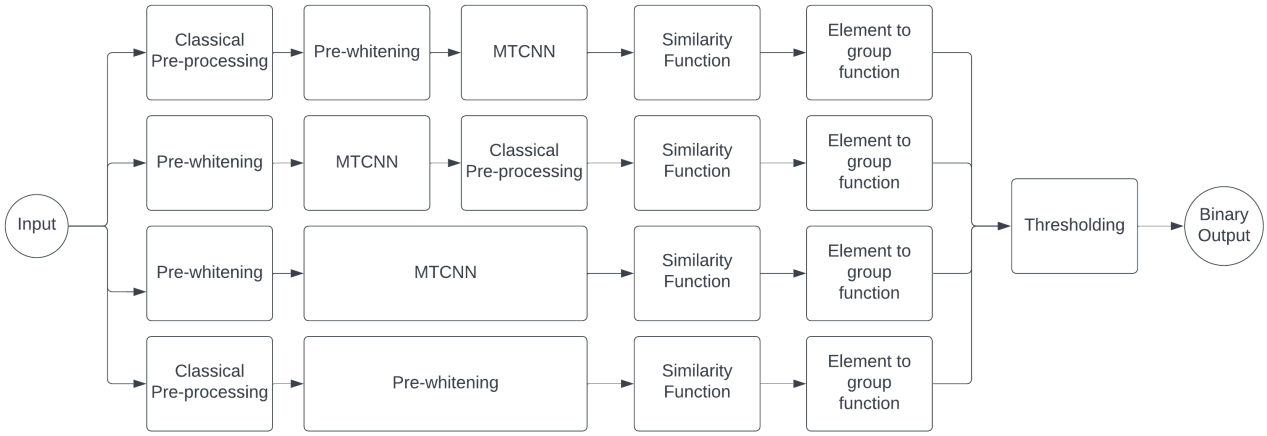


Figure 11. Tested inference approaches diagram

	Augm.	Sugg. Transf.	Aug. Iter	Precision	Recall	F1-score	Weights	Group Fun.	Metric	Thr.
1	False	False	0	100.0	100.0	100.0	VGGFace2	Average	Cosine	0.18
2	False	False	0	100.0	100.0	100.0	CASIA-WF	Average	Cosine	0.18
3	True	True	5	100.0	100.0	100.0	CASIA-WF	Average	Cosine	0.18
4	False	False	0	82.43	100.0	90.4	VGGFace2	Max	Cosine	-
5	False	False	0	93.0	87.0	90.0	VGGFace2	Average	L2	0.82
6	False	False	0	96.4	98.9	97.6	VGGFace2	Median	L2	0.82

Table 3. Retrieval results and metrics with used configuration

5.5. Results schema

The results are printed on a complete JSON structure generated by our testing framework which allowed a unique and standard metric so as to obtain the best combination of hyperparameters.


```
{
"metrics": {
    "precision": 100.0,
    "recall": 100.0,
    "f1_score": 100.0
},
"test_session_info": {
    "using_image_cap": false,
    "image_cap_value": 0,
    "threshold_used": 0.18,
    "distance_metric_used": "cosine",
    "pretrained_face_weights": "vggface2"
},
"true_positives:": {
    "details": { ... },
    "outcome_summary": {
        "total_tp_dataset_size": 391,
        "detected_positives": 389,
        "false_negatives": 0,
        "errors": 2,
        "accuracy": 100.0
    }
},
"true_negatives:": {
    "details": { ... },
    "outcome_summary": {
        "total_tn_dataset_size": 13233,
        "detected_negatives": 13232,
        "false_positives": 0,
        "errors": 1,
        "accuracy": 100.0
    }
}
}
```


6. Emotion Detection

6.1. Network Architecture

The teacher network is the *VGG19* shown in Figure 12 while the architecture created by us is the FERNet and is structured as shown in Figure 13. In addition, at the end of the paper, Tables 6 and 7 describes the structure of the two networks. Each layer consists of a Convolutional layer, a BatchNormalization layer, a Relu, and a MaxPooling layer.

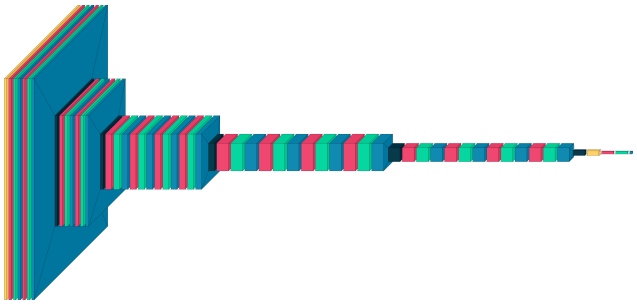


Figure 12. VGG19 architecture with 1 input channel

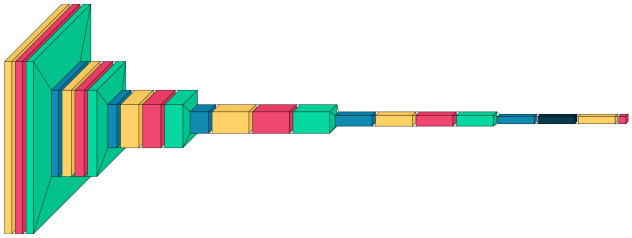


Figure 13. FERNet architecture

The size chosen for the convolutive kernels is fixed and is 3x3 with a stride of 1. The output channels are respectively 64, 128, 256, and 512. We used *SGD* as optimizer for both networks using a learning rate equal to 0.01, momentum equal to 0.9, and weight decay set to $5e{-}4$ (in FERNet we only set the learning rate to 0.001). We choose batch sizes equal to 40 and 10 respectively for the training set and testing set.

6.2. Approach

The architectural choices envisaged a small network that could be performant in a robot with limited power. However, having small dimensions, the network may not achieve adequate performance for the task. For this reason, we decided to use distillation [Geoffrey Hinton, 2015]. We used VGG19 trained on *CK+* as a teacher and FERNet as a student.

6.2.1 DataAugmentation

As data augmentation, we applied the following transformations in the training and testing phases:

- Grayscale
- RandomHorizontalFlip
- RandomAdjustSharpness (set to 2 to improve input sharpness)

Initially, we also tried to use RandomCrop transformation in order to improve the generalization capabilities of the model but we noticed the occurrence of a particular phenomenon: the model was able to obtain excellent metrics during the testing phase but at the time it had to predict the emotion of a single sample (from the same test set or camera) failed by always returning the same emotion. In general, the proposed approach works quite well however there are many misclassifications during use within the project due to the subjectivity of the expressions and the difference between those present in the dataset and those acquired from the wild. For this reason, we have enriched the *CK+* dataset with our own images, in order to amalgamate these misclassification.

6.3. Gradient Clipping

We thought it appropriate to try using gradient clipping to avoid the phenomenon of exploding gradients but this did not lead to significant improvements. Gradient clipping ensures that the gradient vector g has norm at most c.

$$g = c * \frac{g}{||g||}$$

This helps gradient descent to behave reasonably even if the loss landscape of the model is irregular. Figure 14 shows an example of an extremely steep cliff in the loss landscape. Without clipping, the parameters take a huge step downhill and leave the "good" region. With gradient clipping, the size of the drop step is limited and the parameters remain in the "good" region.

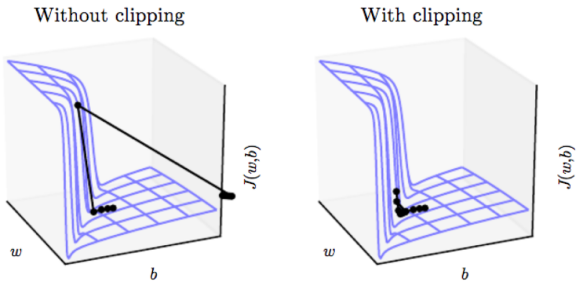


Figure 14. Example of the effect of gradient clipping in a recurrent network with two parameters w and b

6.4. Distillation

Knowledge distillation is one of many techniques that can help with overcoming the increasing requests of memory and computational power by very big deep learning models, by using and 'distilling' the knowledge of a complex model, into a smaller model, much easier to deploy, without much loss in terms of metrics and performances on the validation data. Knowledge distillation is a compression technique that makes it possible to train a small model (called "student") by transferring knowledge from a bigger, more complex model (called "teacher"). The student

learns how to mimic the teacher by leveraging its knowledge, to achieve similar accuracy. There are three major forms of distillation: Response-based knowledge, Feature-based knowledge, and Relation-based knowledge.

6.4.1 Response-based knowledge

Figure 15 has shown the pipeline. It focuses on the output of the final layer of the teacher model so that the student will learn to mimic its predictions. This can be achieved by using the distillation loss, which captures the difference between the logits of the student and the teacher model respectively. As this loss is minimized over training, the student model will become better at making the same predictions as the teacher. We implemented this form of distillation.

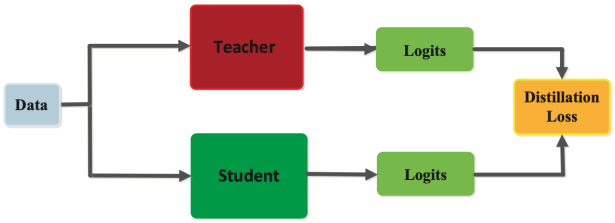


Figure 15. Tipical pipeline of Response-based knowledge

6.5. Dataset

The dataset used is the *CK+*, a large dataset of facial images, that is often used for training and evaluation of FER algorithms. *CK+* contains 593 video sequences of 123 subjects representing 7 universal facial expressions (anger, disgust, fear, happiness, sadness, surprise, and neutrality). We have also added our own images to better balance the dataset. We have used for further purposes another dataset of faces, which we have decided to call *BigFER*, containing 35.9k (the training set consists of 28,709 examples and the public test set consists of 3,589 examples) and the *FER2013* contains approximately 30,000 RGB images of different facial expressions of dimensions 48x48, also divided into 7 types of expressions (angry, disgust, fear, happy, sad, surprise, neutral). The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image.

6.6. Loss Function

We used cross-entropy during the VGG19 and FERNet training.

$$L_{CE} = -\sum_{i=1}^{n} t_i \log p_i$$

where n is the number of classes, t_i is the truth label and p_i is the Softmax probability for the i^{th} class. During the distillation, we used the Kullback-Leibler divergence loss to measure the discrepancy between the probability distributions predicted by the teacher network and the student network.

$$L_{KL} = L(y_{pred}, y_{true}) = y_{true} * log \frac{y_{true}}{y_{pred}}$$

where

$$y_{pred} = LogSoftmax(\exp \frac{x_i}{T}) = \log \frac{\exp \frac{x_i}{T}}{\sum_j \exp \frac{x_j}{T}}$$

$$y_{true} = Softmax(\exp \frac{x_i}{T}) = \frac{\exp \frac{x_i}{T}}{\sum_j \exp \frac{x_j}{T}}$$

KL divergence loss measures how much the student model deviates from the target distribution and encourages the student model to generate probabilities similar to those of the teacher model. The final loss will be

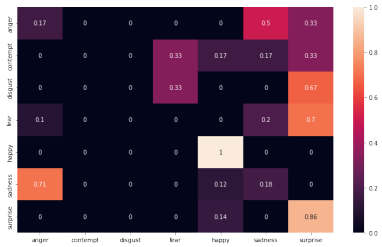
$$L = \alpha L_{KL} + (1 - \alpha) L_{CE}^{Student}$$

The α parameter is used to weigh the distillation loss and the loss student. An α of 0 means that only the distillation loss is considered, and vice versa. The temperature parameter scales the uncertainty of the teacher's predictions, this is used to smooth the probability distribution generated by the softmax. Higher values of temperature make the distribution softer, distributing the probability among the different classes in a more uniform way while lower values of temperature make the distribution harder, increasing the probability of the class with the highest probability and reducing the probabilities of the other classes. We have chosen α equal to 0.25 and temperatures equal to 1.

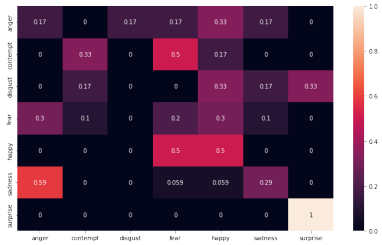
6.7. Results

The performances obtained during the FERNet testing using distillation are shown in Figure 18. As it's possible notice we reached really good performances as represented in Table 4 FERNet has been trained on 35 epochs because of its simplified architecture and distillation process during the training phase. We also tried several experiments obtaining various performances as shown in Figure 17. VGG19 has been trained over 90 epochs. We achieved very good results as shown in Table 5 we didn't report precision, recall, and F1-Score because they are almost equal to accuracy. In addition, it's possible to notice that during training using FER2013 and BigFER, the VGG19 has been overfitted whereas this didn't occur during training with CK+.

We have also created a small dataset of 65 photos to conduct a validation step for both networks. Despite the excellent results obtained during the testing phase, our networks failed when attempting to predict facial expressions from photos captured by the camera, as shown in Figure 16. The consistent difference in performance could be attributed to the subjective nature of facial expressions. For instance,



(a) FERNet confusion matrix



(b) VGG19 confusion matrix

Figure 16. Confusion matrixes based on the validation step with our handcrafted dataset.

Accuracy	Loss	Precision	Recall	F1-Score
99.89	0.014	99.24	99.66	99.42

Table 4. Accuracy and loss obtained during FERNet testing using distillation and VGG19 trained on CK+ as teacher

	CK+	BigFER	FER2013
Accuracy	98.47	66.40	65.71
Loss	0.0153	1.87	1.85

Table 5. Accuracy and loss obtained during VGG19 testing with different datasets

both networks excel at predicting emotions such as surprise and happiness, as these are overt and easily generalizable expressions. However, sadness is correctly predicted only when accompanied by closed eyes or a particular facial inclination.

7. Conclusions and future developments

In our work, we have already seen several interesting results. We learned with a practical approach the whole classic pipeline in the field of computer vision, starting from the construction of different datasets in the right manner and also getting to know using those in the literature. Further development could open an API to offer the learned information to external services, like large language models that could set their response accordingly to the target emotion.

A significant improvement could be achieved by switching from MTCNN to FastMTCNN[3] in which a stride is implemented in the MTCNN algorithm. This version proved to achieve up to 55 FPS at full resolution but it would need the whole stack of the latest N frames available. This is needed because the stride, in this context, is just an approximation in which we assume that the next **S** frames would have the same detections and landmarks as the frame before. This implementation would, as of today, go against our project architecture in which every frame is independent of its neighbors since our naive approach allowed us to study each block of the pipeline and analyze deeply the single block performance and compute interesting comparisons. Future development could be to improve the quality of the training data by representing expressions in a more meaningful and compact way, thus avoiding classification dependencies on facial inclination/rotation. We have managed to obtain satisfactory results in every part of the project even if we have also encountered the great difficulties that this discipline brings, managing to resolve most of them.
We are therefore left with a great deal of experience from an educational project held for the Computer Vision course at the University of Modena and Reggio Emilia.

References

[abhisuri97, nd] abhisuri97 (n.d.). Abhisuri97/pyelike: Basic pupil tracking with gradients in python (based on fabian timm's algorithm). 5

[Borgeaud et al., 2021] Borgeaud, S., Mensch, A., Hoffmann, J., and Sifre, L. (2021). Improving language models by retrieving from trillions of tokens. 2

[Cheng et al., 2021] Cheng, Y., Wang, H., Bao, Y., and Lu, F. (2021). Appearance-based gaze estimation with deep learning: A review and benchmark. *CoRR*, abs/2104.12668. 2

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1. 2, 3

[Geoffrey Hinton, 2015] Geoffrey Hinton, Oriol Vinyals, J. D. (2015). Distilling the knowledge in a neural network. abs/1503.02531. 10

[Huang et al., 2007] Huang, G. B., Ramesh, M., Berg, T., and Learned-Miller, E. (2007). Labeled faces in the

[3]https://www.kaggle.com/code/timesler/fast-mtcnn-detector-55-fps-at-full-resolution?scriptVersionId=29201326

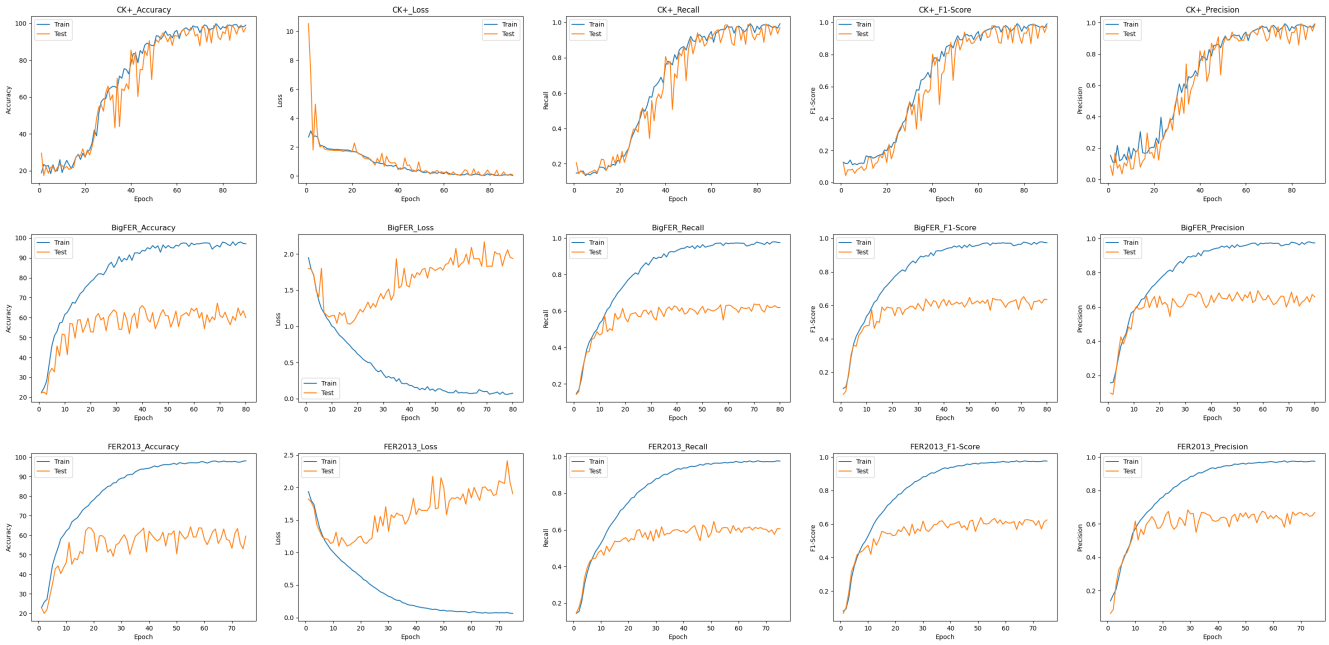


Figure 17. VGG's metrics based on its testing using CK+, BigFER, FER2013

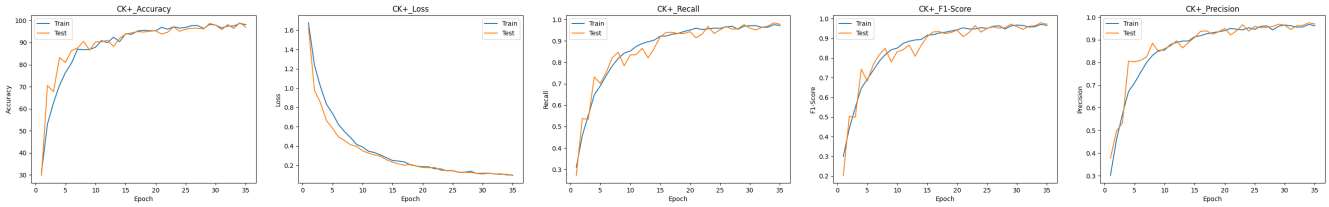


Figure 18. FERNet's metrics based on distillation using VGG19 testing on CK+

wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst. 8

[Hui Ding, 2016] Hui Ding, Shaohua Kevin Zhou, R. C. (2016). Facenet2expnet: Regularizing a deep face recognition net for expression recognition. *IEEE Transactions on Affective Computing 2020*, abs/1609.06591. 2

[Jesorsky et al., 2001] Jesorsky, O., Kirchberg, K. J., and Frischholz, R. W. (2001). Robust face detection using the hausdorff distance. In *Audio-and Video-Based Biometric Person Authentication: Third International Conference, AVBPA 2001 Halmstad, Sweden, June 6–8, 2001 Proceedings 3*, pages 90–95. Springer. 5

[Kazemi and Sullivan, 2014] Kazemi, V. and Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3

[Khan and Lee, 2019] Khan, M. Q. and Lee, S. (2019). Gaze and eye tracking: Techniques and applications in adas. *Sensors*, 19(24):5540. 2

[Li and Deng, 2018] Li, S. and Deng, W. (2018). Deep facial expression recognition: A survey. *IEEE Transactions on Affective Computing 2020*, abs/1804.08348. 2

[Sagonas et al., 2013] Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., and Pantic, M. (2013). 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 397–403. 3

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 7

Layer (type)	Output Shape	Param #
Conv2d-1	[1, 64, 48, 48]	640
BatchNorm2d-2	[1, 64, 48, 48]	128
ReLU-3	[1, 64, 48, 48]	0
MaxPool2d-4	[1, 64, 24, 24]	0
Conv2d-5	[1, 128, 24, 24]	73,856
BatchNorm2d-6	[1, 128, 24, 24]	256
ReLU-7	[1, 128, 24, 24]	0
MaxPool2d-8	[1, 128, 12, 12]	0
Conv2d-9	[1, 256, 12, 12]	295,168
BatchNorm2d-10	[1, 256, 12, 12]	512
ReLU-11	[1, 256, 12, 12]	0
MaxPool2d-12	[1, 256, 6, 6]	0
Conv2d-13	[1, 512, 6, 6]	1,180,160
BatchNorm2d-14	[1, 512, 6, 6]	1,024
ReLU-15	[1, 512, 6, 6]	0
MaxPool2d-16	[1, 512, 3, 3]	0
Conv2d-17	[1, 512, 3, 3]	2,359,808
BatchNorm2d-18	[1, 512, 3, 3]	1,024
ReLU-19	[1, 512, 3, 3]	0
MaxPool2d-20	[1, 512, 1, 1]	0
AvgPool2d-21	[1, 512, 1, 1]	0
Linear-22	[1, 7]	3,591
Total params:	3,916,167	
Trainable params:	3,916,167	
Non-trainable params:	0	

Table 6. FERNet architecture

[Sugano et al., 2014] Sugano, Y., Matsushita, Y., and Sato, Y. (2014). Learning-by-synthesis for appearance-based 3d gaze estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1821–1828. 2

[Szegedy et al., 2016] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. 6, 7

[Timm and Barth, 2011] Timm, F. and Barth, E. (2011). Accurate eye centre localisation by means of gradients. *Visapp*, 11:125–130. 5

[Trishume, nd] Trishume (n.d.). Trishume/eyelike: A webcam based pupil tracking implementation. 5

[Viola and Jones, 2005] Viola, P. and Jones, M. (2005). Robust real-time face detection. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. IEEE Comput. Soc. 2, 3

[Zhang et al., 2016] Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503. 6

Layer (type)	Output Shape	Param #
Conv2d-1	[1, 64, 48, 48]	640
BatchNorm2d-2	[1, 64, 48, 48]	128
ReLU-3	[1, 64, 48, 48]	0
Conv2d-4	[1, 64, 48, 48]	36,928
BatchNorm2d-5	[1, 64, 48, 48]	128
ReLU-6	[1, 64, 48, 48]	0
MaxPool2d-7	[1, 64, 24, 24]	0
Conv2d-8	[1, 128, 24, 24]	73,856
BatchNorm2d-9	[1, 128, 24, 24]	256
ReLU-10	[1, 128, 24, 24]	0
Conv2d-11	[1, 128, 24, 24]	147,584
BatchNorm2d-12	[1, 128, 24, 24]	256
ReLU-13	[1, 128, 24, 24]	0
MaxPool2d-14	[1, 128, 12, 12]	0
Conv2d-15	[1, 256, 12, 12]	295,168
BatchNorm2d-16	[1, 256, 12, 12]	512
ReLU-17	[1, 256, 12, 12]	0
Conv2d-18	[1, 256, 12, 12]	590,080
BatchNorm2d-19	[1, 256, 12, 12]	512
ReLU-20	[1, 256, 12, 12]	0
Conv2d-21	[1, 256, 12, 12]	590,080
BatchNorm2d-22	[1, 256, 12, 12]	512
ReLU-23	[1, 256, 12, 12]	0
Conv2d-24	[1, 256, 12, 12]	590,080
BatchNorm2d-25	[1, 256, 12, 12]	512
ReLU-26	[1, 256, 12, 12]	0
MaxPool2d-27	[1, 256, 6, 6]	0
Conv2d-28	[1, 512, 6, 6]	1,180,160
BatchNorm2d-29	[1, 512, 6, 6]	1,024
ReLU-30	[1, 512, 6, 6]	0
Conv2d-31	[1, 512, 6, 6]	2,359,808
BatchNorm2d-32	[1, 512, 6, 6]	1,024
ReLU-33	[1, 512, 6, 6]	0
Conv2d-34	[1, 512, 6, 6]	2,359,808
BatchNorm2d-35	[1, 512, 6, 6]	1,024
ReLU-36	[1, 512, 6, 6]	0
Conv2d-37	[1, 512, 6, 6]	2,359,808
BatchNorm2d-38	[1, 512, 6, 6]	1,024
ReLU-39	[1, 512, 6, 6]	0
MaxPool2d-40	[1, 512, 3, 3]	0
Conv2d-41	[1, 512, 3, 3]	2,359,808
BatchNorm2d-42	[1, 512, 3, 3]	1,024
ReLU-43	[1, 512, 3, 3]	0
Conv2d-44	[1, 512, 3, 3]	2,359,808
BatchNorm2d-45	[1, 512, 3, 3]	1,024
ReLU-46	[1, 512, 3, 3]	0
Conv2d-47	[1, 512, 3, 3]	2,359,808
BatchNorm2d-48	[1, 512, 3, 3]	1,024
ReLU-49	[1, 512, 3, 3]	0
Conv2d-50	[1, 512, 3, 3]	2,359,808
BatchNorm2d-51	[1, 512, 3, 3]	1,024
ReLU-52	[1, 512, 3, 3]	0
MaxPool2d-53	[1, 512, 1, 1]	0
AvgPool2d-54	[1, 512, 1, 1]	0
Linear-55	[1, 7]	3,591
Total params:	20,037,831	
Trainable params:	20,037,831	
Non-trainable params:	0	

Table 7. VGG19 architecture