IdentiMood:
a face and mood-based authentication system

Daniele Solombrino, solombrino.1743111@studenti.uniroma1.it
Davide Quaranta, quaranta.1715742@studenti.uniroma1.it
Emanuele Volanti, volanti.1761743@studenti.uniroma1.it

Introduction

IdentiMood is an authentication software, composed of **face** and **mood verification**.

Using their webcams, users can **enroll**, by providing a **neutral** and **mood-influenced** face, and can then submit probes for the subsequent authentication stages.
The submitted neutral template is saved in the gallery, whilst the mood extracted from the mood-influenced template is saved as a user's **metadata**.

Similarly, verification is composed of two distinct phases: first, the user must provide a neutral face, then it is asked to provide a face which mimics the mood decided during enrollment.

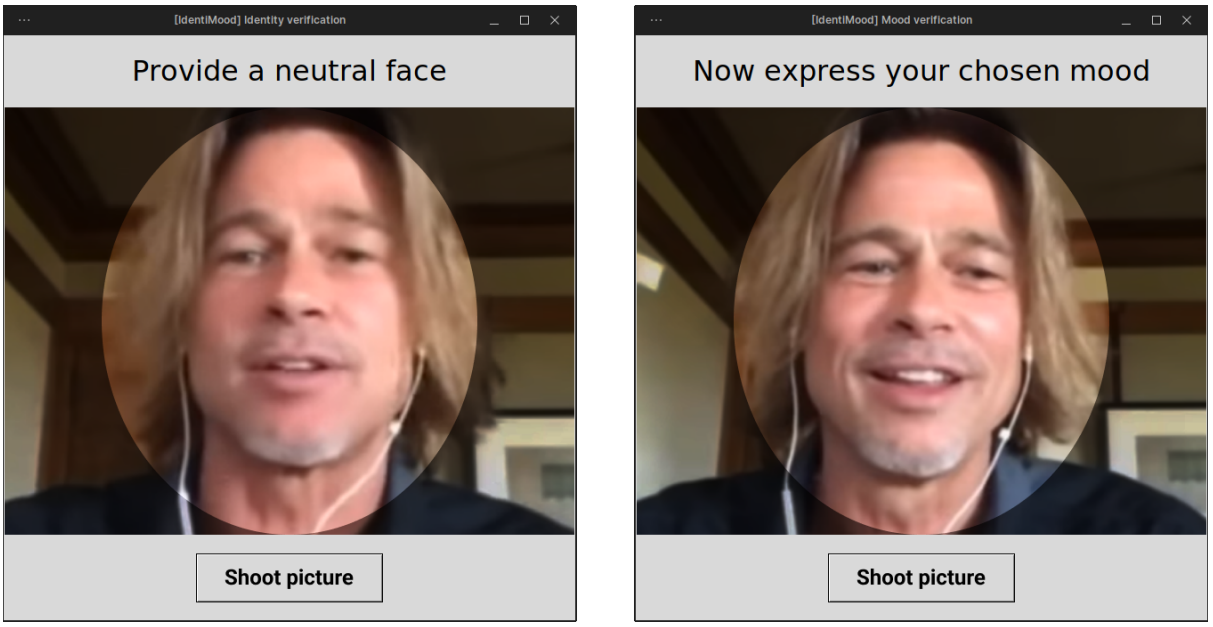


Fig. 1: Screenshots of the two phases of the IdentiMood application

If the **neutral** probe **matches** a gallery template for the **claimed identity** and the mood extracted from the **mood**-influenced probe matches the mood in the user's metadata, then the user is **authenticated**.
Otherwise, the authentication is denied.

Use cases

Some use-case examples can be:

- Guarantee access to a specific directory inside the user's profile.
- Second authentication factor upon an SSH login.
- Unlock a keychain.
- Allow a git push operation.
- Allow opening an email client.

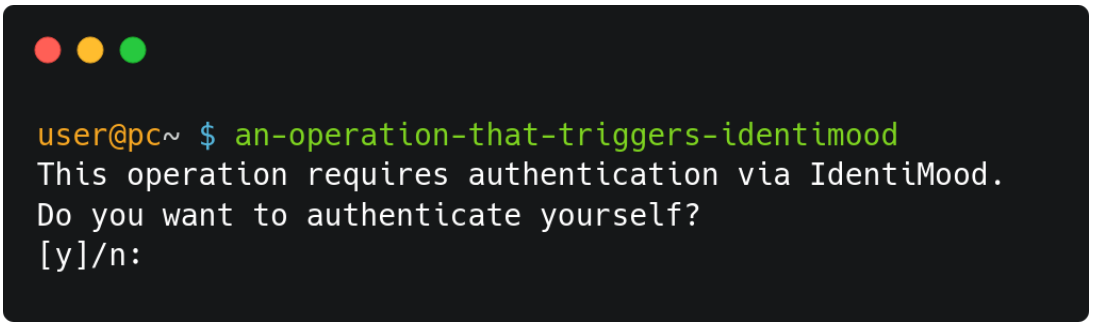


Fig. 2: Example of a shell command that can trigger IdentiMood

System design

The system is a **Python 3 object-oriented** application, that performs both enrollment and recognition. Specifically:

1. Users can call the **enrollment** flow, which requires them to provide via webcam:
    1. A neutral sample, which will be saved as a template.
    2. An emotion-distorted sample, that will be used to associate the extracted mood information to the user.
2. Users provide an **identity claim** and can be **recognized** providing:
    2.1. A neutral sample
    2.2. A sample in which an emotion is mimicked

The system returns a successful response if the user is recognized and it has mimicked the emotion configured during the enrollment process.

As a **security feature** to avoid a **side-channel attack**, 2.1 and 2.2 are independently processed, with the authentication decision computed in the end.

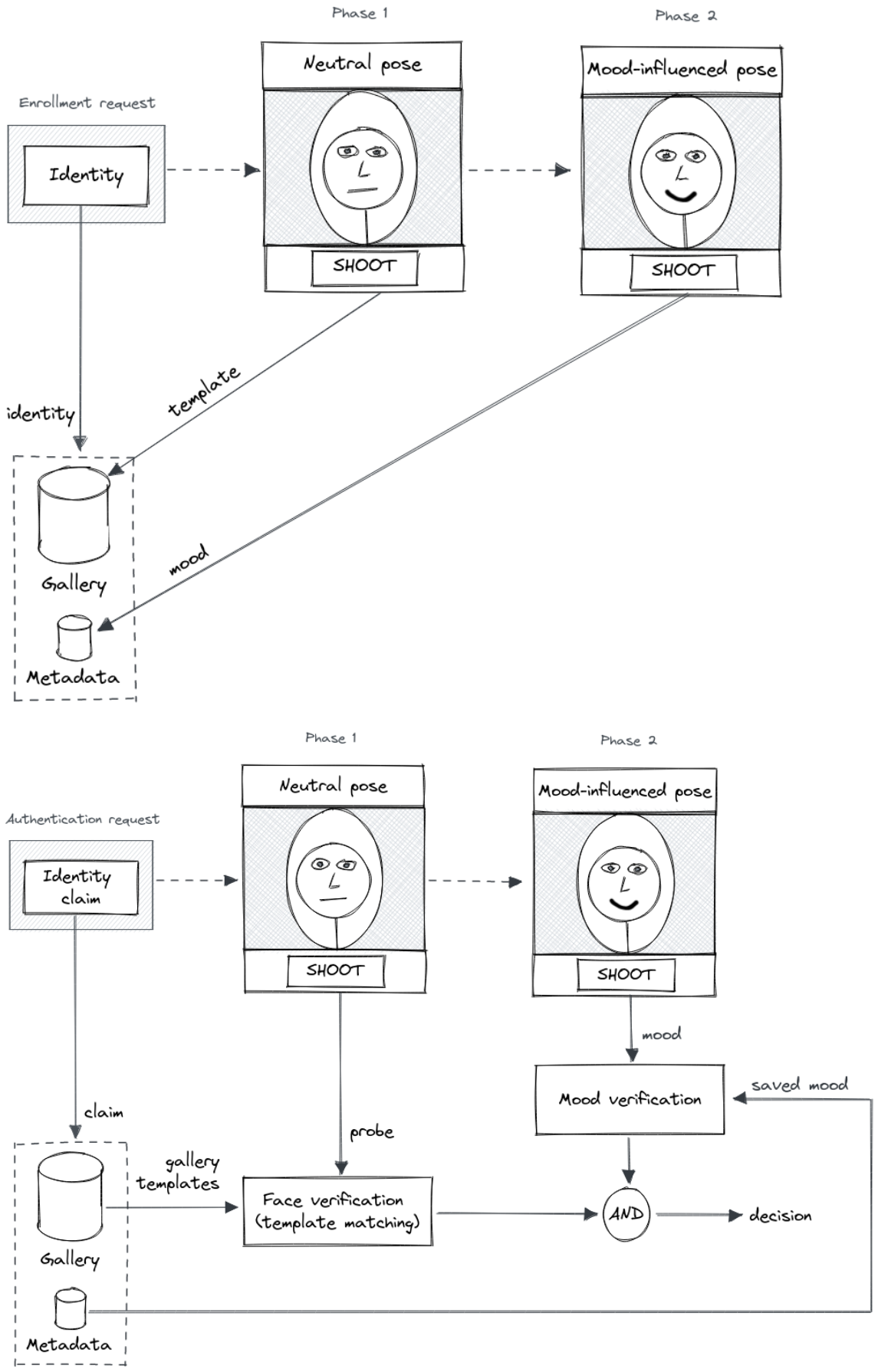


Fig. 3: High level IdentiMood enrollment (top) and verification (bottom) logical flows

Project structure

The system is composed of the following main components (classes):

- **App**: coordinator and entrypoint.
- **Enroller**: handler for the enrollment procedure.
- **Operations**: handler for the recognition operations.
- **Window**: handler for GUI objects, using the Tkinter[1] library.

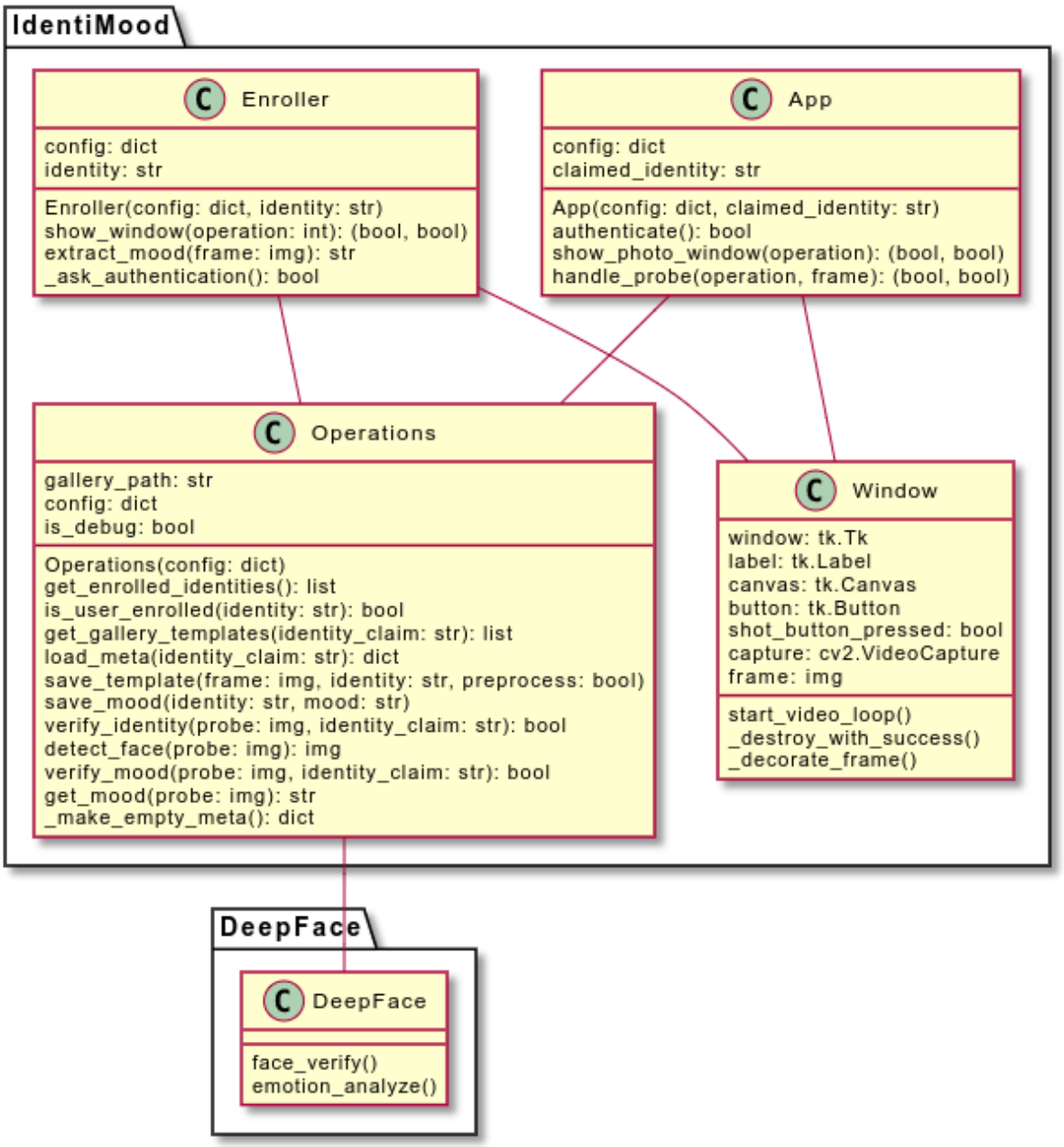


Fig. 4: IdentiMood class diagram

[1] https://docs.python.org/3/library/tkinter.html

The software can be **configured** and tweaked per user needs, by changing options inside the `config.json` file, located in the project root.

```
{
    "gallery_path": "./gallery",
    "verify": {
        "threshold": 0.12,
        "model_name": "Dlib",
        "detector_backend": "opencv",
        "distance_metric": "cosine",
        "normalization": "base",
        "save_normalized_templates": true
    },
    "mood": {
        "use_delta_percent": true,
        "delta_percent_threshold": 70.0,
        "detector_backend": "opencv"
    },
    "debug": true
}
```

The **gallery** is implemented as a directory containing a sub-directory for each identity and storing the templates as PNG files, named with an **UUID**.

As an example:

```
.
├ mary
│    ├── 21eb2e26-a166-4e7e-92a8-065502e335bb.png
│    └── meta.json
└ john
    ├── 80a30386-cd18-4337-851a-9d2c965e07e3.png
    ├── 82662f64-1188-46b8-b3c8-e895647c5dac.png
    └── meta.json
```

The `meta.json` file contains **metadata** about the user, such as:

```
{
      "name": "Mary Smith",
      "favorite_mood": "happy"
}
```

DeepFace library: why and how

DeepFace[2] offers some of the core functionalities needed by IdentiMood, namely **face verification** and **mood detection**.

Additionally, DeepFace transparently handles **preprocessing** steps, when needed, such as:

- Face detection
- Alignment
- Normalization
- Cropping
- Grayscale conversion

Every DeepFace call is highly parameterized: the library supports multiple **models**, **distance measures** and **detector backends**, all of which can be selected by the caller, whenever a function is invoked.

The available APIs respect standards of: usability, out of the box accuracy, fair execution times and deployability.
All of these characteristics gave us the opportunity to build a system based on solid grounds, with a high level of portability and **flexibility**.

The face verification and mood recognition functions return both a **distance score**. The first one is the distance between the given images (Fig. 5), the latter the probability that the input face is making an expression/mood (Fig. 6).
The **authentication decision** can then be taken according to such returned scores.

Fig. 5: Face verification

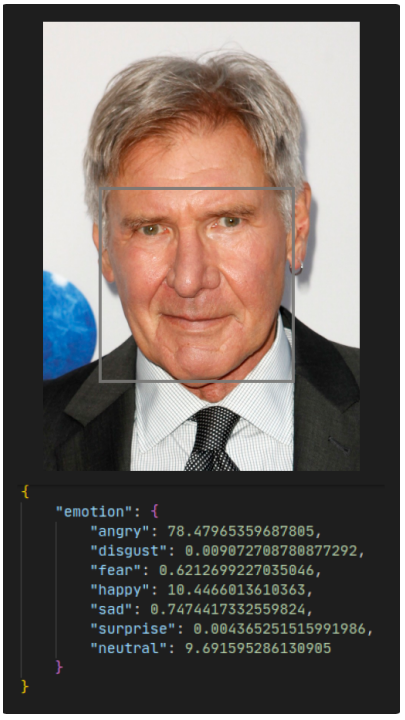


Fig. 6: Mood recognition

[2] https://github.com/serengil/deepface

Models

DeepFace offers **pre-trained**, state of the art Deep Learning models, namely:

- VGG-Face
- OpenFace
- Facenet
- Facenet512
- DeepFace
- DeepID
- Dlib
- ArcFace

Most of them use vanilla **Convolutional Neural Networks**, whilst others use **Inception**, **Deformable** and **Residual Neural Networks** (FaceNet, FaceNet512, DeepID and Dlib).

Distance metrics

The following **distance metrics** can be used to compute the distance between two feature vectors:

- Cosine
- Euclidean
- Euclidean with L2 normalization

Detector backends

DeepFace can automatically perform **face detection**, so that the user does not need to crop and/or align input images.

A variety of face detectors (pre-trained, state-of-the-art **Deep Convolutional Neural Networks**) are offered:

- OpenCV
- SSD
- Dlib
- MTCNN
- RetinaFace

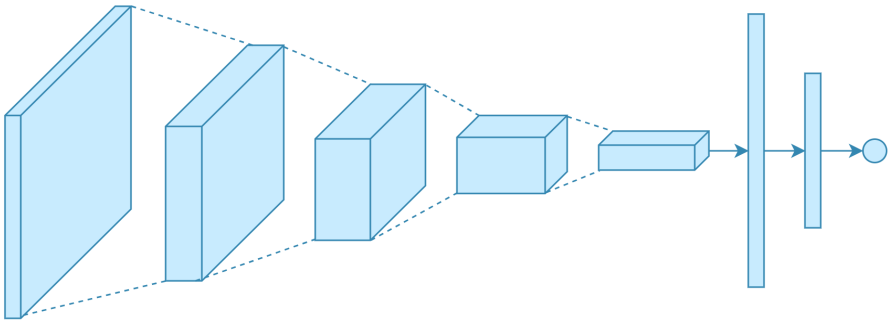


Fig. 7: High level Convolution Neural Network schematics

Evaluation goal

IdentiMood uses **two authentication steps**, thus two different evaluations have been performed.
Appropriate evaluation goals were set for each portion.

For the **verification** part, an **all-against-all distance matrix** has been used to produce benchmarks.
By processing all the possible probe and gallery pairs, each image plays in turn both the roles of genuine and impostor, in probe and gallery both.

Specifically, Detection Error Tradeoff (**DET**), Receiver Operating Characteristic (**ROC**) and "**Thresholds** vs. **FAR** and **FRR**" charts have been used to discover the best combination of model, similarity/distance measure and threshold, in order to achieve as low recognition errors as possible.

As far as the **emotion recognition verification** is concerned, a custom-conceived evaluation has been used.
The goal was to see how the adoption of different thresholds, combined with different models, impacted the emotion recognition decision.

Evaluation setup

In order to achieve the proposed goals, an entire **evaluation pipeline and environment** has been crafted, involving datasets, statistics, metrics, plots manipulations and computations.

Datasets

The first step was to search for some datasets compatible with our requirements, i.e. **gender**, **ethnicity** and **PIE** variations.

Name	**# subjects**	**# photos per subject**	**Age range** (years)	**Ethnicity variations**	**Emotion variations**	**Resolution** (px)	**PIE variations** or **other distortions**
TUTFS	39 F 74 M	5	[10, 60]	Asian African Latino Caucasian	Neutral Happy Surprised	4608 × 3072	Eyes closed Smile Open mouth Sunglasses
KDEF	35 F 35 M	7	[20, 30]	Caucasian	Scared Angry Disgusted Happy Neutral Surprised Sad	562 × 762	Illumination
VGG-Face2	~3651 F ~5480 M	[87, 843]	NA	Asian African Latino Caucasian	Neutral Happy	avg 137×180	Illumination Pose
YaleFaces	1 F 11 M	576	[20, 40]	Asian African Latino Caucasian	Neutral	640 × 480	Illumination Pose

Table 1: Selected datasets, according to aforementioned desired characteristics

All datasets have a directory-based structure. Every directory contains all the samples of a single identity, storing in the file an encoded description of the sample (ground truths), such as:

- Subject ID.
- Emotion.
- Illumination.
- Whether it has glasses on or not.
- Gender.

For example, the structure of the TUTFS dataset is the following:

```
./TUTFS/
├── AF01
│   ├── AF01AFS.JPG
│   ├── AF01ANS.JPG
│   ├── AF01DIS.JPG
│   ├── AF01HAS.JPG
│   ├── AF01NES.JPG
│   ├── AF01SAS.JPG
│   └── AF01SUS.JPG
└── AF02
    ├── AF02AFS.JPG
    ├── AF02ANS.JPG
    ├── AF02DIS.JPG
    ├── AF02HAS.JPG
    ├── AF02NES.JPG
    ├── AF02SAS.JPG
    └── AF02SUS.JPG
```

Subsampled datasets: what, why and how

Due to the abundance of data samples, models and distance metrics, the total number of possible matches for some datasets (VGG-Face2 and YaleFaces) would've reached the **order of billions**, resulting in an impractical estimated execution time that would've exceeded one decade!

To mitigate this issue, **subsampled** versions of the aforementioned dataset have been generated.

Due to its size and lack of descriptive labels (such as gender, PIE variations and ethnicity), VGG-Face2 has been stochasticly sampled.

On the other hand, YaleFaces offered meaningful file descriptors and sported less total number of images, making the partitioning and sampling easily feasible by hand. "**Urns**" have been manually created, according to gender, PIE **variations** and ethnicity. Samples have then been extracted from said "urns", trying to create as **balanced** as possible sets (Fig. 8).

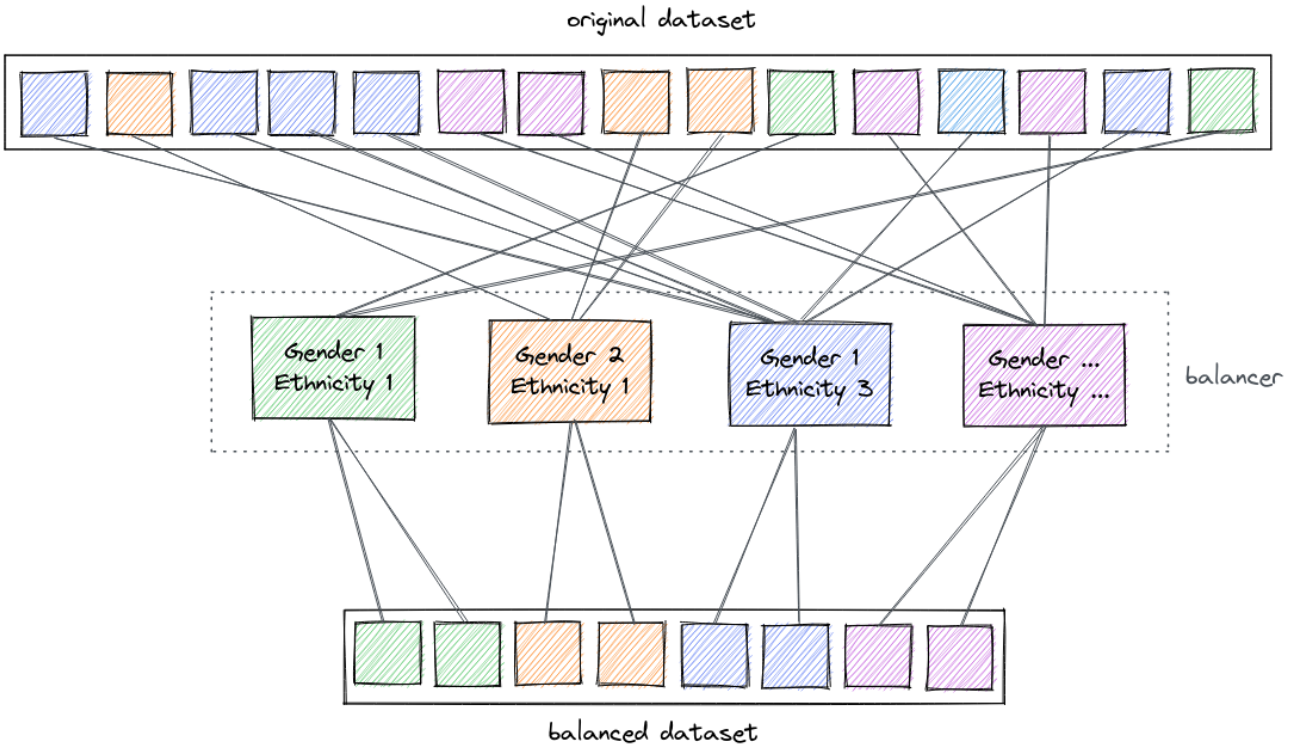


Fig. 8: High level representation of the random balanced subsampling process

For every dataset, a plain text file containing the list of all image paths has been generated and then used to feed inputs to the DeepFace methods.

Face detectors

The **best** face detector (the one with lowest rate of undetected faces) has been selected, in order to stick to it throughout all the runs and reduce the total number of performed matchings.

All the available detectors (listed in Detector backends) were tested on all the images of all the datasets (for a total number of ~18k unique files), in order to gather their number of **successful face detections**.
A Python script has been devised to load all the possible input images, perform the aforementioned tests and dump the **statistics** in an external JSON file, eventually fed to another commodity Python script, to transform the number of failures into **rates** and to **plot** statistics.

Face verifier

The **all-against-all** method has been selected as evaluation technique.

Images in the various datasets have been used to create the **probe** and **gallery** collections, in order to ensure that they all play **both roles**, as per all-against-all methodology design.

After calling the evaluation script with the proper configuration parameters (file list, similarity measures and models), the high-level algorithm structure is the following:

- For each pair $(p,\ g)\ s.t.\ p \neq g$ of images, where p and g are respectively probe and gallery templates:
  - Invoke DeepFace's face **verification** function on p and g (Fig. 9).
  - Store the returned distance score.
  - For every $threshold$[3]:
    - Update statistics according to distance score and threshold[3]:
      - $distanceScore \leq threshold \ \wedge id(p) = id(g) \Rightarrow GA\ ++$
      - $distanceScore \leq threshold \ \wedge id(p) \neq id(g) \Rightarrow FA\ ++$
      - $distanceScore > threshold \wedge id(p) = id(g) \Rightarrow FR\ ++$
      - $distanceScore > threshold \wedge id(p) \neq id(g) \Rightarrow GR\ ++$
    - Update number of genuine and impostor attempts:
      - $id(p) = id(g) \Rightarrow GenuineAttempts\ ++$
      - $id(p) \neq id(g) \Rightarrow ImpostorAttempts\ ++$

Where $id(x)$ is a function that returns the true identity of template x.

All of the above steps are repeated for every possible combination of model and distance metric.

```
DeepFace.verify("probe.png", "gallery.png", model, distance_metric)
```

Fig. 9: `DeepFace.verify()` function call

[3] Since the distance score can be reused multiple times for every **threshold**, this step has been moved here to optimize run times.
In fact, `DeepFace.verify()` calls are very time consuming (two images loading from disk, two face localizations and the Deep Learning model query must be performed).

Fig. 10: DeepFace.verify() output

All the above metrics (**Genuine Acceptances**, **Genuine Rejections**, **False Acceptances** and **False Rejections**) are used to compute the **DET**, **ROC** and "**Thresholds vs. FAR and FRR**" plots, which are then useful to elect the best performing IdentiMood settings (more on them in the Face Verifier paragraph of Evaluation Results section).

For each kind of plot, a specific parameter can be used to select the best curve:

- In **DET** the best curve is the one for which the area under the curve (**AUC**), is minimum.
- In **ROC** the best curve is the one for which the AUC is maximum.
- In "**Thresholds vs. FAR and FRR**" the best pair of FAR and FRR is the one for which the **Equal Error Rate** (**EER**) has the lowest y-coordinate.

The following tools have been used:

- The scikit-learn[4] for AUC value.
- Matplotlib[5] for DET, ROC and "Thresholds vs. FAR and FRR" plots.
- Shapely[6] for the EER, since it's required to find the intersection point between the FAR and FRR lines.

[4] https://scikit-learn.org/stable/
[5] matplotlib.org
[6] https://github.com/shapely/shapely

Emotion verifier

Since the project also relies on emotion as the second authentication element, it was also necessary to evaluate its **performances**.

For every sample in the aforementioned dataset, `DeepFace.analyze()` returns the emotion (sad, angry, surprise, fear, happy, disgust, neutral) **probability** distribution.

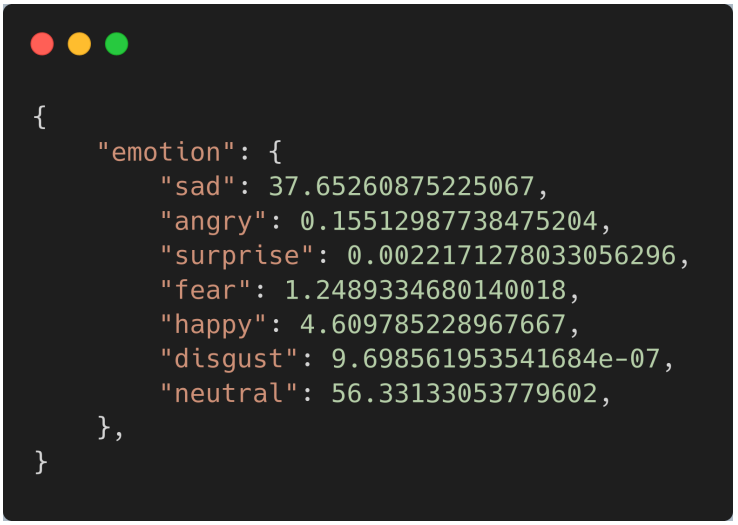


Fig. 11: Emotion probability distribution returned as Python dictionary by DeepFace

After getting the **ground truth** (actual emotion) of the analyzed sample similarly to the all-against-all algorithm, the emotion verifier checks whether DeepFace **correctly classified** the emotion.

Specifically, the procedure verifies whether:

1. The most probable emotion returned by DeepFace is correct (from now on "**naive emotion verification**").
2. The "confusion" among the two most probable emotions is low enough (from now on "**delta-aware emotion verification**").

Naive emotion verification simply considers the emotion verification correct if DeepFace's result is the same as the ground truth.

For the **delta-aware emotion verification**, let:

- $actualEmotion$ be the ground truth emotion, for a given sample
- e_1 the most probable emotion
- e_2 the second most probable emotion
- se_1 and se_2 the probability scores (as per Fig. 11) respectively for e_1 and e_2
- $\Delta = abs(se_1 - se_2)$
- thr a threshold value

An emotion is then considered correctly classified if all of the following conditions hold true at the same time:

- $e_1 = actualEmotion$
- $\Delta \geq thr$

The following table shows some possible examples:

Ground truth emotion	e_1	e_2	se_1	se_2	Δ	thr	Emotion verified according to delta-aware method
Happy	Happy	Sad	0.63	0.21	0.42	0.35	Yes
Happy	Happy	Sad	0.40	0.35	0.05	0.35	No
Neutral	Fear	Neutral	0.74	0.12	0.64	0.35	No
Neutral	Fear	Angry	0.83	0.05	0.78	0.35	No

Table 2: Different scenario examples of the Delta-aware emotion verification

The key goal is to assess how the delta-aware emotion verification result changes, upon threshold variations.
A similar intuition to the one described in all-against-all is used to deal with multiple thresholds.

Δ = 1st emotion ranking - 2nd emotion ranking

Fig. 12: High level representation of delta in Delta-aware emotion verification

Evaluation execution

The evaluation procedures have been **simultaneously** executed on **multiple systems**, in order to distribute the overall CPU usage, since the number of combinations to evaluate is very large and DeepFace calls are onerous.

Thanks to the configurability of the evaluation scripts, it has been possible to **parallelize** by configuration (what models, distance measure and datasets to use). Multiple **virtual machines** have been started in parallel, to tackle a specific **model-measure-dataset configuration**.

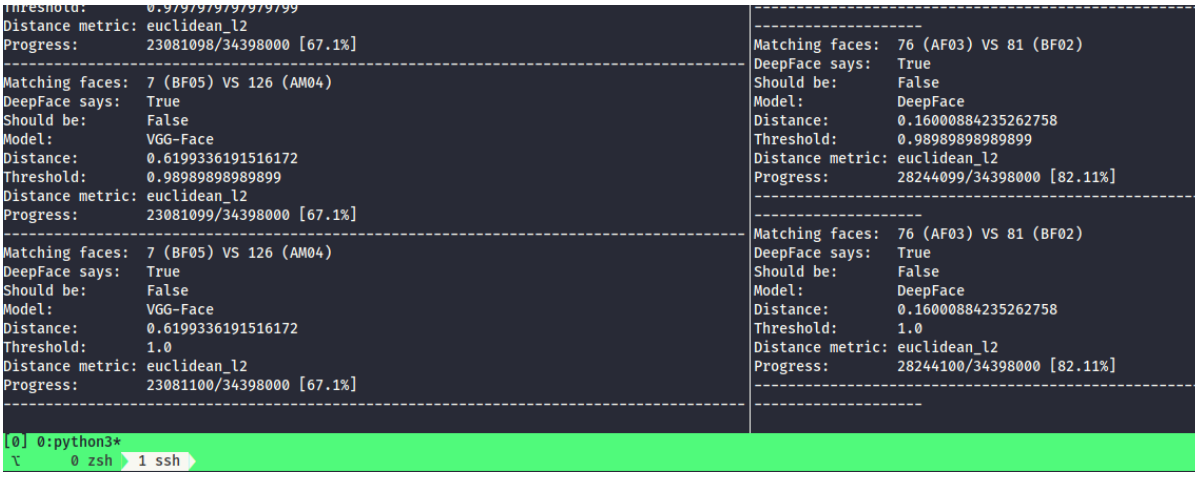


Fig. 13: Screenshot of an execution of two simultaneous all-against-all scripts via SSH

The various executions have been performed on local and remote machines (Fig. 13), the latter rented on AWS, Vultr and DigitalOcean **cloud providers**.

High **memory** and **compute-oriented** instances with a high number of cores and threads have been used, in order to speed up image load times and the query to the Deep Learning models.
Storing the datasets into a **ramfs** (directly in RAM) and using a high number of cores and threads greatly reduced latencies.
As a result, the images were instantly available for the massively parallel computations.

To facilitate the **deployment** of the multiple instances - i.e. the set of packages to install - a **custom-made shell script** has been automatically executed upon instance creation.
This also resulted in the **standardization** of the evaluation environment, useful for easy and accurate experiments reproducibility.

Operations on the VMs have been performed via SSH.

As far as the invocation is concerned, no matter if via CLI or as library, the following **configuration parameters** can be passed to it:

- Path of file storing **list of input images** composing probe and gallery collections.
- Whether to **limit** the probe and gallery collections to a specific number of files.
- Parameters to generate the **linear space** to be used as **threshold list**.
- What **metrics** to use (see Distance Metrics).
- What **models** to use (see Models).
- Whether to show **verbose** output during the computations

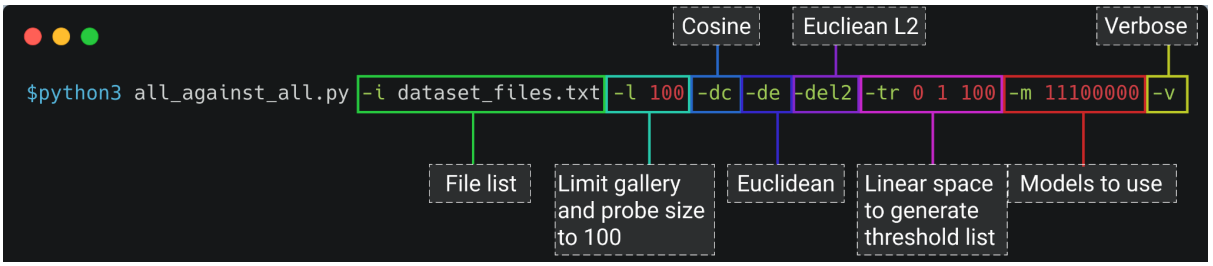


Fig 14: Usage example of the all_against_all script invocation from CLI

Evaluation results

This section reports the results gathered from the evaluation execution described above.
The results are divided by the evaluated system module:

- Face detector (explanation on the tested metrics).
- Face verifier (explanation on the tested metrics).
- Emotion verifier (explanation on the tested metrics).

Face detector

RetinaFace is the best face detector, since it has the **lowest** rate (~13%) of undetected faces. Given this result, all the upcoming evaluations have been performed using RetinaFace as detector backend.

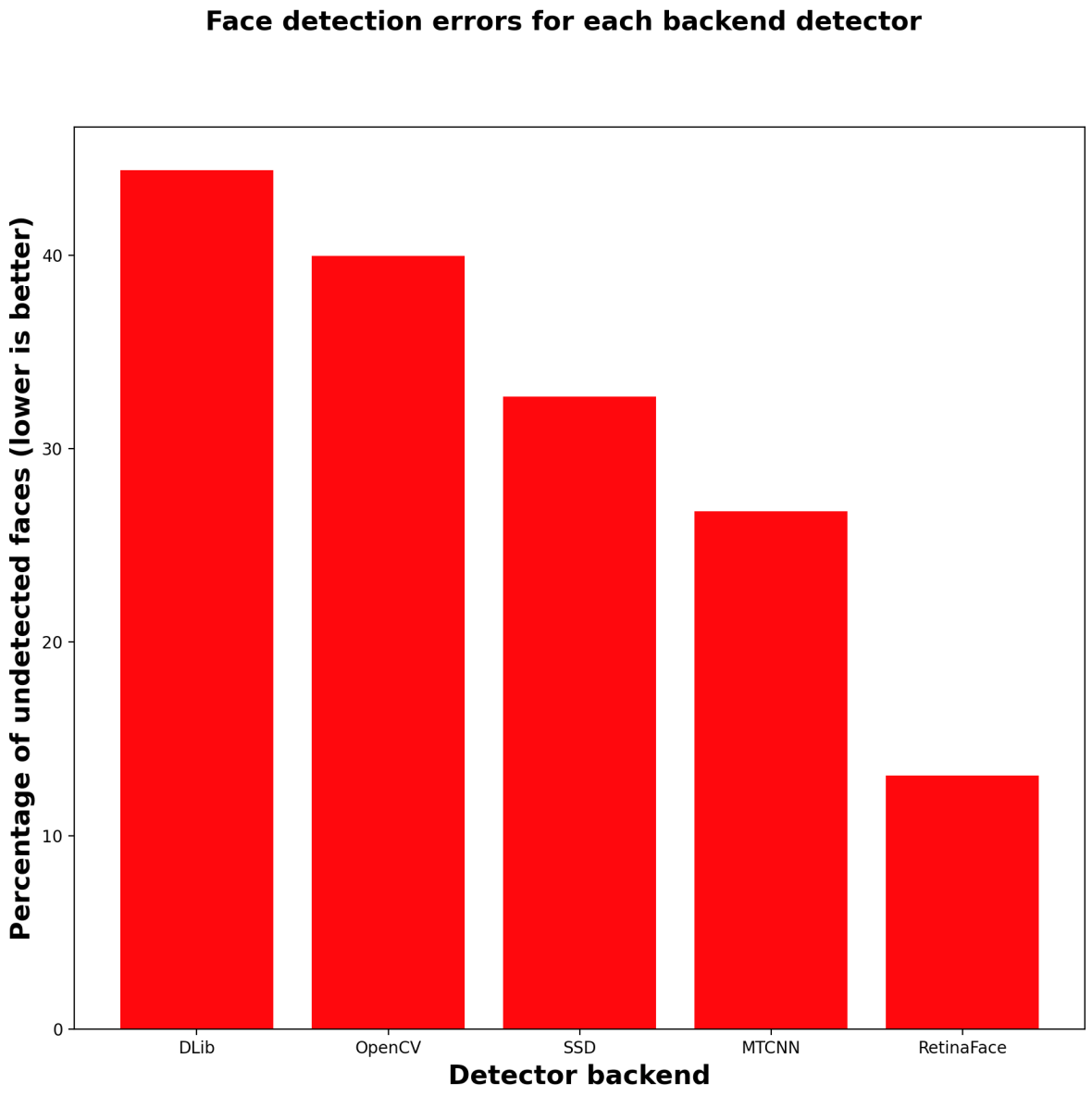


Chart 1: percentage of undetected faces for each face detector backend

Face verifier

In order to offer the most comprehensive and flexible evaluation, the **best** and **worst** performers have been discovered, according to their described metrics.

For every datasets, **all possible** model **and** distance **combinations** have been tested. As a consequence, fixing any component of the combo (i.e. the model or the distance measure), future users/reviewers can find the best performing combination which includes the desired component.
For example, suppose to be a future user which would like to stick to the Dlib model for some reason.
Looking at the Table 3 below, it's easy to observe what distance measure performs better, when paired to Dlib, across the different performance metrics.

Detailed comparison results

For each `dataset` (YaleFaces, VGG-Face, TUTFS, KDEF) and distance measure, the table shows the **best** and **worst model**, according to the DET, ROC and "Threshold vs. FAR and FRR" (abbreviated as THR*).

YaleFaces	**Best**			**Worst**		
	DET	**ROC**	**THR***	**DET**	**ROC**	**THR***
Cosine	Facenet512	VGG-face	Facenet512	DeepID	DeepID	DeepID
Euclidean	Facenet512	Facenet512	Facenet512	DeepID	DeepID	DeepID
Euclidean L2	Facenet512	Facenet512	Facenet512	DeepID	DeepID	DeepID
VGG-Face2	**Best**			**Worst**		
	DET	**ROC**	**THR***	**DET**	**ROC**	**THR***
Cosine	Dlib	Dlib	Dlib	DeepID	DeepID	DeepFace
Euclidean	Facenet512	Facenet512	Facenet512	DeepID	DeepID	DeepID
Euclidean L2	Facenet	Dlib	Dlib	DeepID	DeepID	DeepID
TUTFS balanced	**Best**			**Worst**		
	DET	**ROC**	**THR***	**DET**	**ROC**	**THR***
Cosine	Facenet512	Dlib	Facenet512	DeepID	DeepID	DeepID
Euclidean	Facenet512	ArcFace	Facenet512	DeepID	DeepID	DeepID
Euclidean L2	Facenet512	ArcFace	Facenet512	DeepID	DeepID	DeepID
KDEF balanced	**Best**			**Worst**		
	DET	**ROC**	**THR***	**DET**	**ROC**	**THR***
Cosine	ArcFace	Facenet512	Dlib	DeepID	DeepID	ArcFace
Euclidean	Facenet	Facenet512	Facenet512	DeepFace	OpenFace	DeepFace
Euclidean L2	Facenet512	Facenet512	Facenet512	OpenFace	OpenFace	OpenFace

Table 3: Best and worst performing models per metric, for each distance measure.

From the table, it can be observed that:

- **Facenet512** and **Dlib** tend to perform better than the others.
- **Facenet** and **ArcFace** can sometimes outperform the previous ones, but fail to consistently deliver good performances
- **DeepID** is the overall worst performer, independently of the configuration.

See the dedicated paragraph for the definition of best and worst **DET**, **ROC** and "**Thresholds vs. FAR and FRR**".
In biometric systems, the "Thresholds vs. FAR and FRR" plot helps to determine the **best threshold** to use in production.
The common point p between the **FAR** and **FRR** lines is found.
The y-coordinate of p represents the **EER**, whilst the x-coordinate represents the threshold yielding to the aforementioned EER.
Once p is found, the threshold of the system can be set to its x-coordinate.

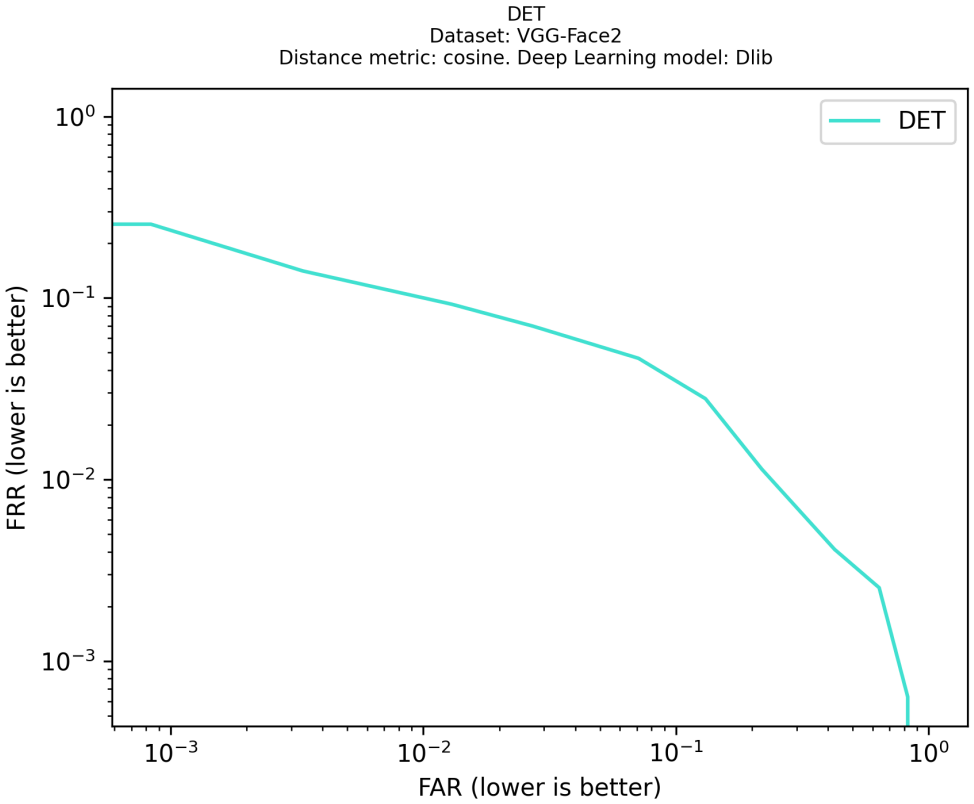
Since the threshold is very important in the "Thresholds vs. FAR and FRR" plot, its section in the previous table has been extracted and expanded, to accommodate information regarding the best threshold, for every model and distance measure.

YaleFaces	**Best THR metric**	
	Model	**Threshold**
Cosine	Facenet512	0.48
Euclidean	Facenet512	22.63
Euclidean L2	Facenet512	0.98
VGG-Face2	**Best THR metric**	
	Model	**Threshold**
Cosine	Dlib	0.12
Euclidean	Facenet512	27.08
Euclidean L2	Dlib	0.48
TUTFS balanced	**Best THR metric**	
	Model	**Threshold**
Cosine	Facenet512	0.53
Euclidean	Facenet512	23.36
Euclidean L2	Facenet512	1.04
KDEF balanced	**Best THR metric**	
	Model	**Threshold**
Cosine	Dlib	0.07
Euclidean	Facenet512	24.12
Euclidean L2	Facenet512	1.05

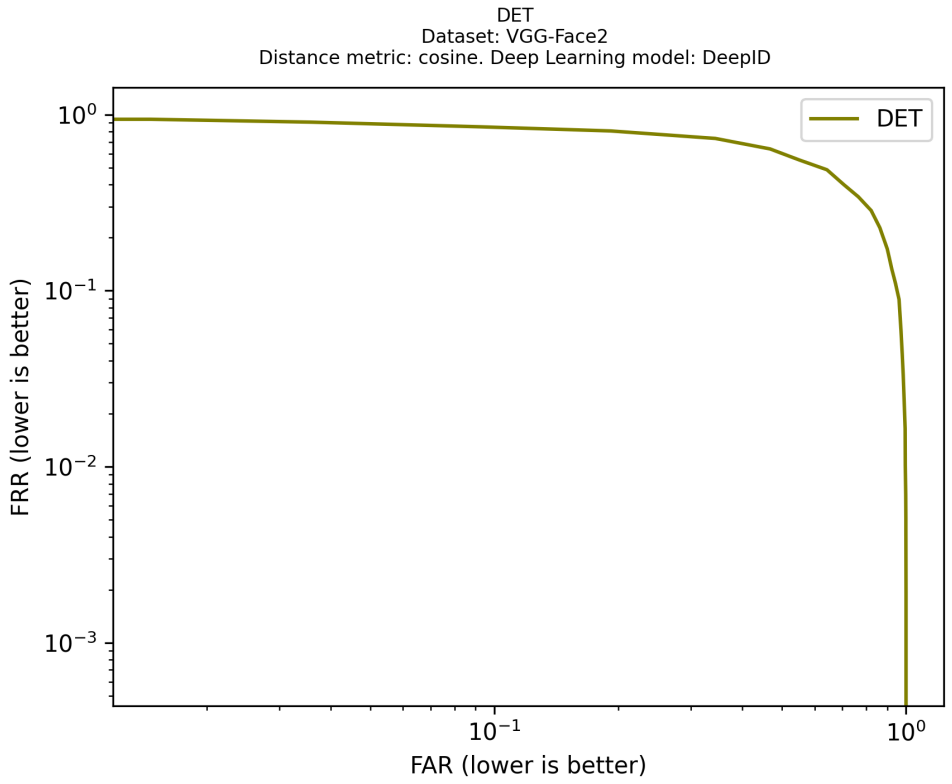
Table 4: Best threshold for the best model for each distance measure and tested dataset

The **total number of plots** surpasses **300** figures, so only a small portion of them is presented in this report (the complete collection of plots is in the attached material). The reported portion has been sampled from the highlighted row in Tables 3 and 4.

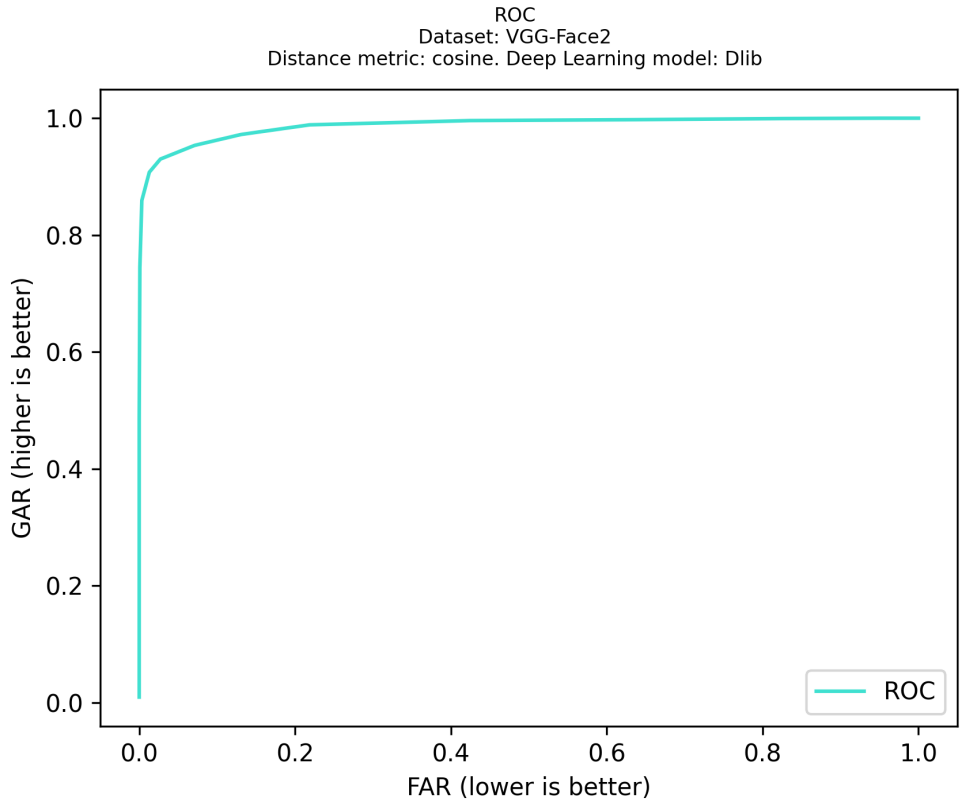
```
$ tree
.
├── KDEF
│   ├── ArcFace
│   │   ├── cosine
│   │   │   ├── DET_22_01_28_19-24-20.png
│   │   │   ├── ROC_22_01_28_19-24-20.png
│   │   │   └── thresholds_vs_FAR_FRR_22_01_28_19-24-19.png
│   │   ├── euclidean
│   │   │   ├── DET_22_01_28_19-24-54.png
│   │   │   ├── ROC_22_01_28_19-24-53.png
│   │   │   └── thresholds_vs_FAR_FRR_22_01_28_19-24-52.png
│   │   └── euclidean_l2
│   │       ├── DET_22_01_28_19-24-42.png
│   │       ├── ROC_22_01_28_19-24-42.png
│   │       └── thresholds_vs_FAR_FRR_22_01_28_19-24-41.png
(498 more lines)
```



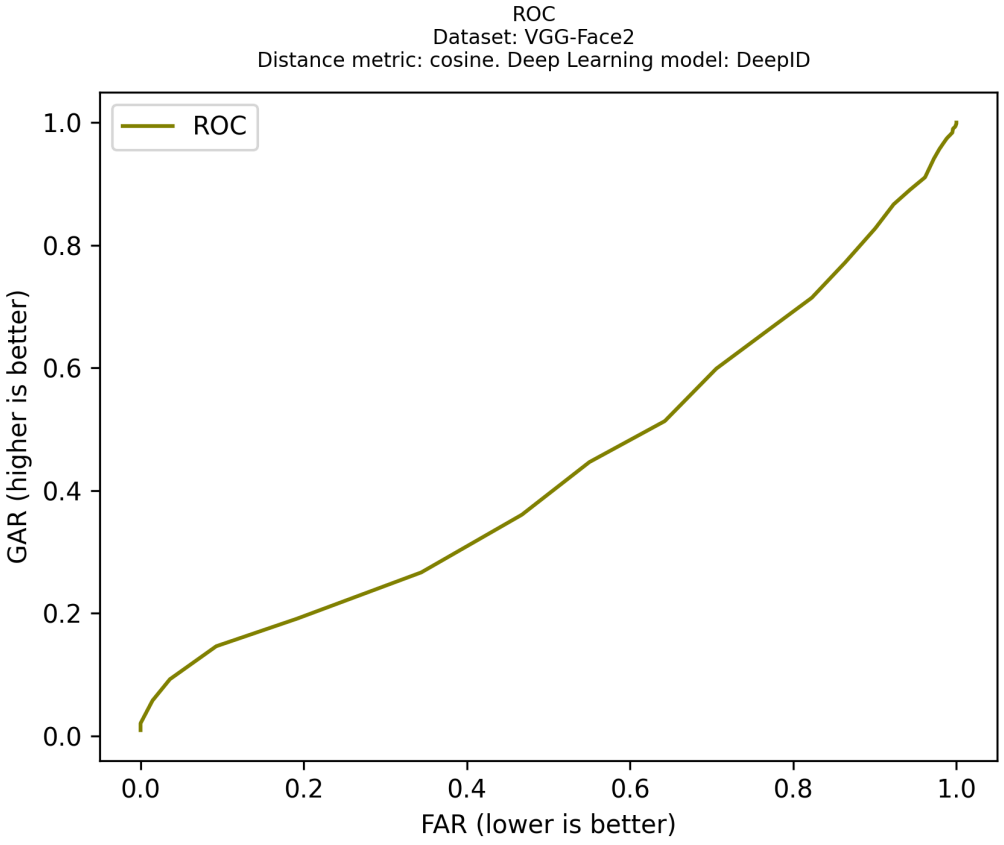
*Plot 1: **Best** DET for VGG-Face2 dataset, with cosine distance and Dlib model*



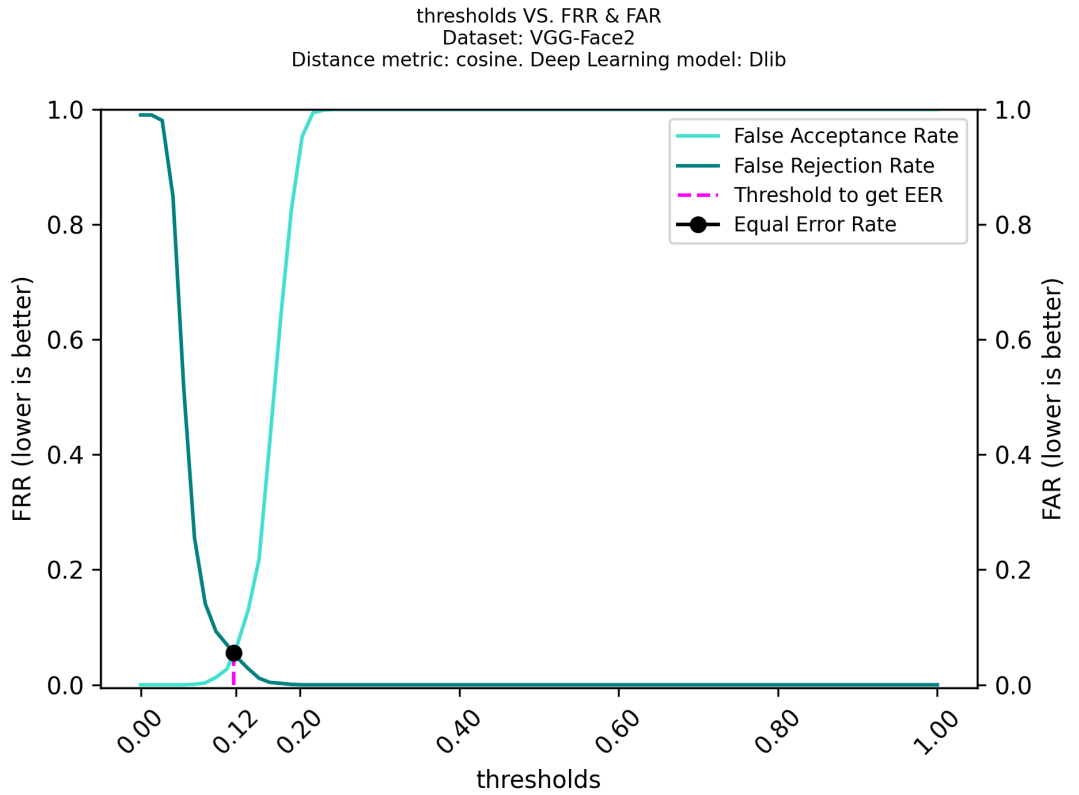
*Plot 2: **Worst** DET for VGG-Face2 dataset, with cosine distance and DeepID model*



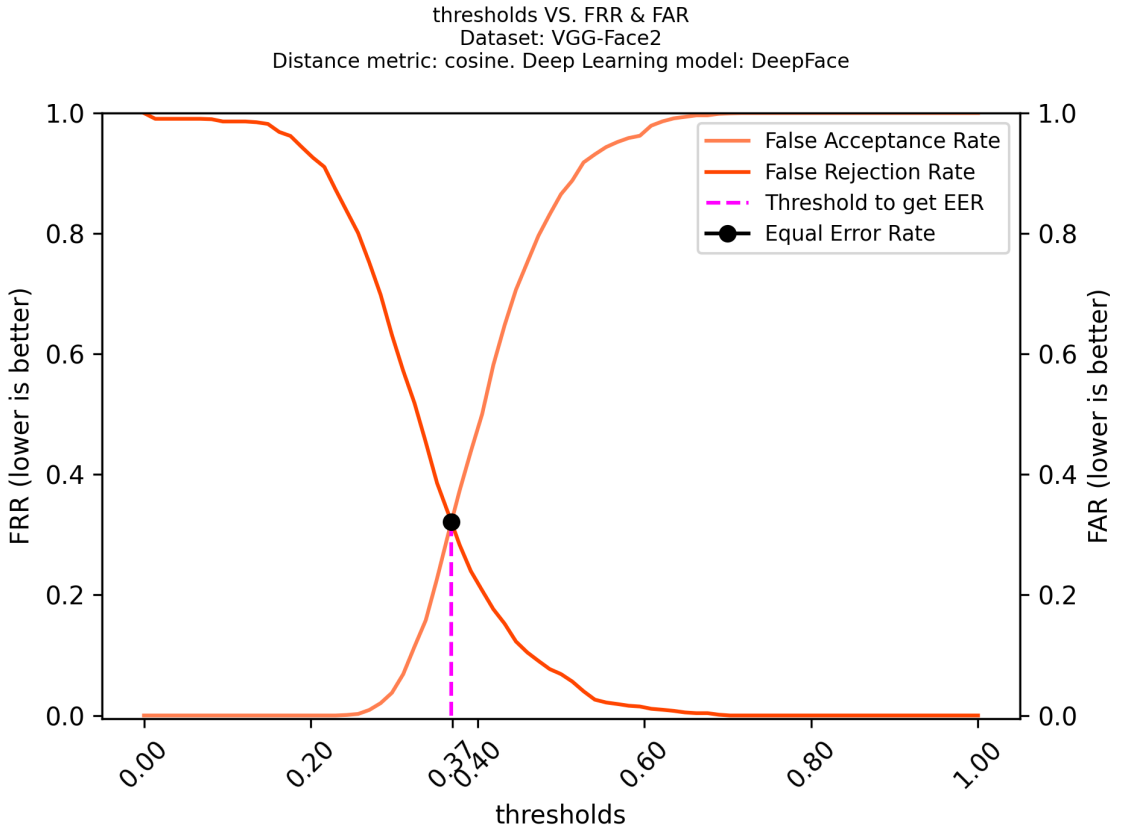
*Plot 3: **Best** ROC for VGG-Face2 dataset, with cosine distance and Dlib model*



*Plot 4: **Worst** ROC for VGG-Face2 dataset, with cosine distance and DeepID model*



*Plot 5: **Best** Threshold vs FAR-FRR for VGG-Face2 dataset, with cosine distance and Dlib model*



*Plot 6: **Worst** Threshold vs FAR-FRR for VGG-Face2 dataset, with cosine distance and DeepFace model*

Emotion verifier

The following chart summarizes the result of the **naive emotion verification** process.

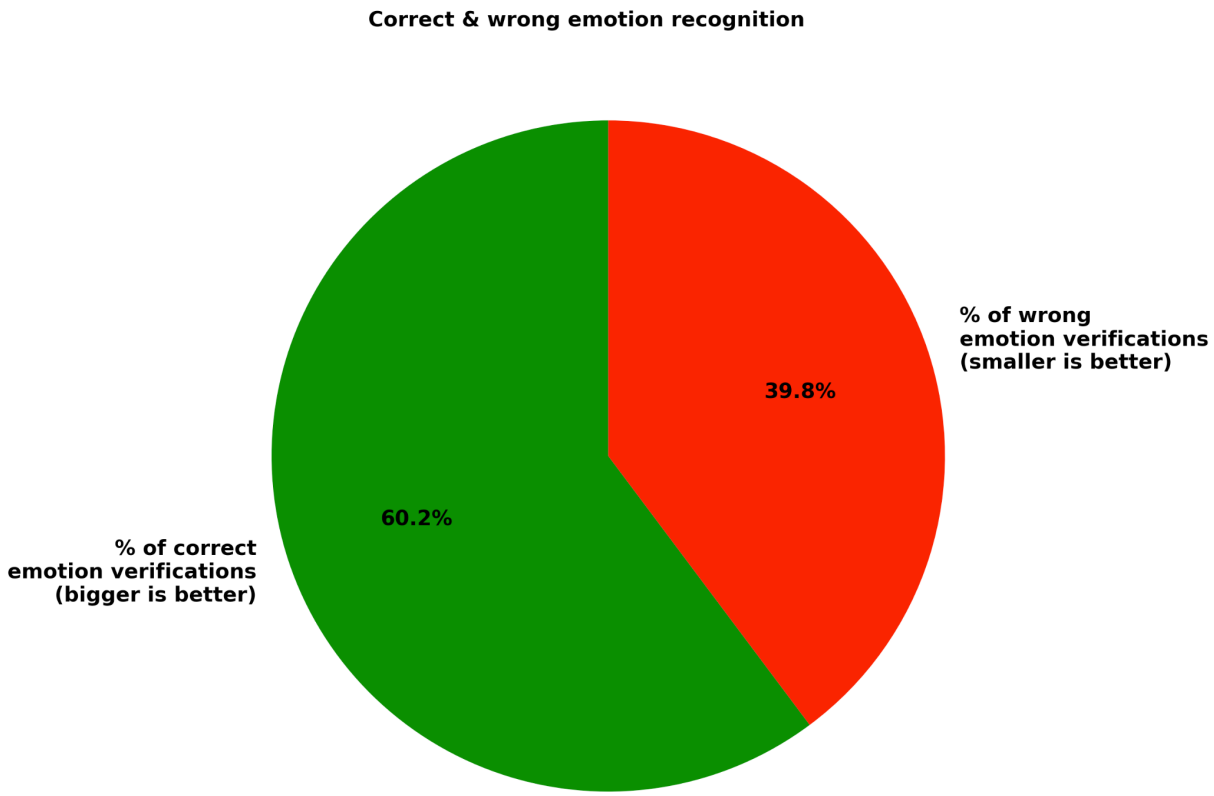
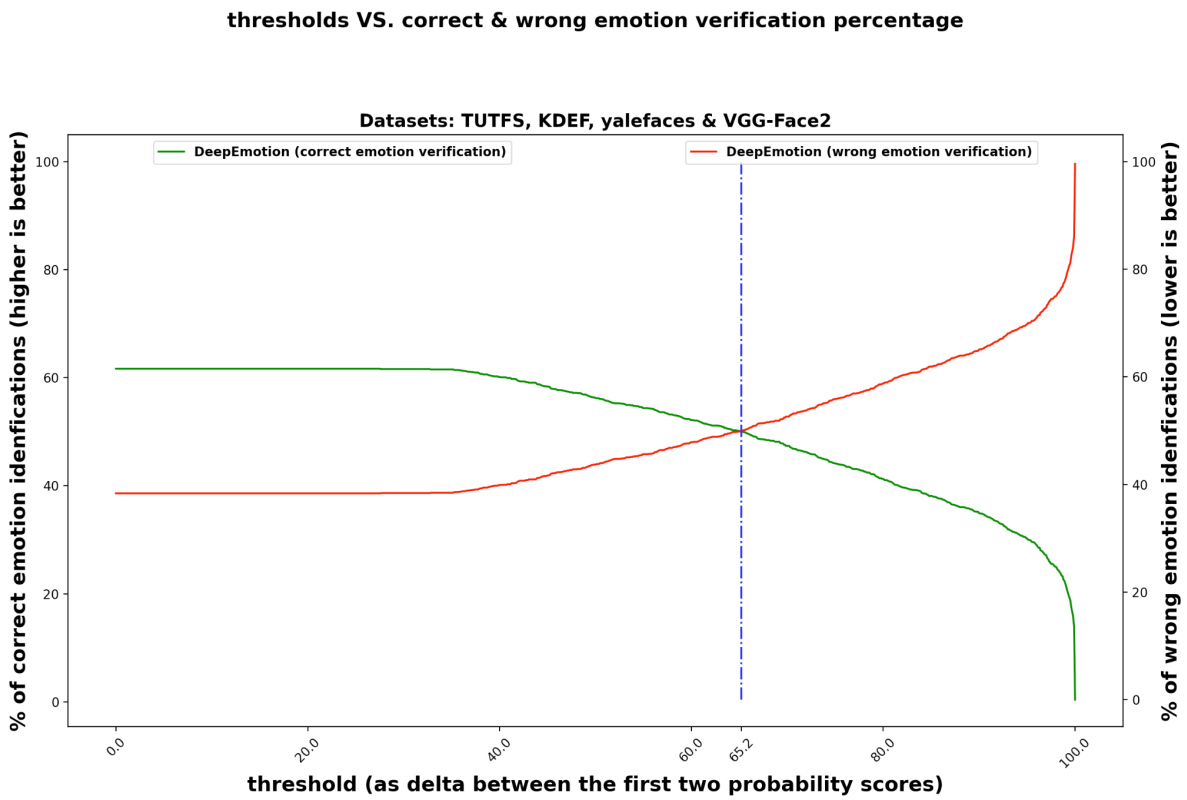


Chart 2: Pie chart comparing correct and wrong emotions detection



Plot 8: Delta-aware emotion recognition performance with increasing thresholds

Plot 8 clearly shows the information we are interested in, when evaluating the **delta-aware emotion recognition**.

The threshold corresponding to the intersection between the two lines (65.2) makes the component perform exactly as a **random classifier**.
Consequently:

- Thresholds **before** the pivoting point result in better performances than a random classifier (because the green line is higher and the red line is lower, which is the ideal case).
- Thresholds **after** the pivoting point determine performances even **worse** than the random classifier.

Once again, similarly to the face verification evaluation, supposing that a hypothetical future reviewer/user wants to stick to a specific system configuration, they can easily discover the best delta-threshold for their selected system configuration.

Future work

Extended DeepFace use

DeepFace offers additional, not yet cited functionalities, which could come in handy for some possible future **IdentiMood expansions**:

- Embracing DeepFace's support for **data streams**, real-time and **video**-based use-cases could be supported by IdentiMood.
- Providing additional Deep Learning models, IdentiMood could support an even wider range of face verifiers.
  For example, it would be interesting to test **ensemble**-based models.

Additional verification factors

Independently from DeepFace, due to IdentiMood's scalable, portable and **flexible** nature, any of its components can be easily expanded to add new features.
The `meta.json` file (described in System Design) is a possible example of such a statement: by adding further discriminating data (for example **gender** and **age**), supplemental **verification methods** may be offered.

Resolution against performances evaluation

A "resolution–aware" evaluation may be carried out, in order to assess whether there is a correlation between the results of any possible **figure of merit** (ROC, DET, "thresholds vs. FAR and FRR", time and/or memory) and the **input image resolution**.

For example, in extreme time or memory-bound use cases, it may be very useful to discover that a certain image downsizing does not determine any notable performance change, as far as the biometric figures of merit are concerned.

Anti-spoofing

Anti-spoofing is a very broad topic which would require a completely **dedicated project**.

Assuming an already developed **black-box** anti-spoofing module, its **integration** would be effortless, thanks to IdentiMood's flexible and modular design.
Evaluation could be based on the currently developed framework as well.

Identification system

IdentiMood's **architecture** could be a good starting point even for the creation of a new project, focused on the **face and emotion identification**, rather than verification.