Biometric Systems 2023/2024
"Exploring Facial Recognition Techniques: LBP versus Deep Learning on LFW Dataset"

Eleonora Fornaro, 1836820
Michela Giampaolo, 1838257
Damiano Gualandri, 1892871

January 22, 2024

1 Introduction

The landscape of facial recognition technology has undergone remarkable advancements, offering several methodologies to address the intricate challenges associated with detecting and identifying faces in varying and demanding scenarios. This report undertakes a comprehensive comparison between two distinct facial feature extraction methodologies: Local Binary Pattern (LBP) and Deep Learning. The assessment will be carried out using the extensively employed Labeled Faces in the Wild (LFW) dataset [5], known for its diversity and real-world complexity.

Local Binary Pattern (LBP) represents a traditional yet robust method for characterizing local facial textures. By capturing pixel-wise patterns and transitions within small regions, LBP offers a computationally efficient solution for facial recognition tasks. The simplicity of LBP makes it particularly useful in scenarios where computational resources are constrained. In this report, we explore the performance of LBP on the LFW dataset, examining its strengths and limitations in this context.

On the other end, we have **deep learning**. The introduction of deep learning has further revolutionized the field, enabling the training of complex models that can automatically learn the most discriminating features of facial images. In this context, DeepFace, developed by Facebook AI Research (FAIR), has proven to be a remarkably effective tool. This framework integrates deep neural networks for the extraction and representation of facial features, thus helping to raise the level of accuracy in face recognition. This report investigates also how DeepFace performs on the LFW dataset,

providing insights into its capacity to address challenges that may be less effective for traditional methods like LBP.

Through an analysis of both LBP and DeepFace on the LFW dataset, this report aims to shed light on their respective strengths, weaknesses, and the trade-offs associated with their application in facial recognition tasks.

Python modules such as OpenCV and Deepface were used to implement the project, respectively for the implementation of recognition using LBP and Deep Learning.

1.1 Identification Open Set

Before discussing the dataset and our development, we will briefly present the context in which we chose to work. Our purpose is to highlight the difference in the evaluation phase when using a face recognizer with LBP versus deep learning techniques. To represent the real difficulties of LBP in identity recognition under less than ideal conditions, we studied an identification case with an open set.
In the task of Open-set Identification, the biometric system aims to determine whether an individual's biometric signature matches that of someone in the gallery. Possible errors in this form of recognition depend on both the matching algorithm used and the acceptance recognition threshold. The threshold setting is crucial and can be customized to achieve low or high false alarm rates and/or low or high probabilities of detection and identification, depending on the specific application requirements.
Open-set face recognition is ideal for real-world applications where the number of identities is not limited and may change over time.

2 Dataset

Labeled Faces in the Wild was selected from all available face recognition datasets due to its representation of real-world scenarios, including images captured under uncontrolled conditions with partial occlusion, such as sunglasses and scarves.

The dataset is widely used in the research community and is provided in jpg format. We used a dataset of 40 individuals by selecting those with the highest number of photos from the LFW dataset. This was necessary as many individuals had only one or two photos available for identification. The faces in the dataset exhibit variations in terms of PIE (Pose, Illumination and Expression), as shown in the Figure 1 .

Then, from the moment we have implemented the Testing phase, we have divided our dataset into Gallery and Probes and we made a further division inside the probe folder to realize an open set environment: Genuines (P_G) and Impostors (P_N).

Figure 1: Some sample of Labeled Faces in the Wild dataset

It's important to underline that neither genuines nor impostors probes are in the gallery, so there are no overlap between Probes and Gallery.
In Identification Open Set, this division may influence the results according to the number of subjects in the gallery and the number of total probes. In our case, we have these numbers fixed, because we have chosen a fixed number of samples for probes and gallery. The number of impostors is the numbers of subdirectories that are in probe but not in the gallery, while the number of genuines is the number of subdirectories that are both in gallery and in probe (with different images).
Since in general we need best quality templates for the enrollment phase, we also tried to swap some photos between gallery and probe and obtained different results as we will see later.

3 Project Implementation

Regarding the implementation of the project, we divide the discussion in two sections: how we implemented the LBP part and how we implemented the Deep Learning part. For convenience of use, we thought of creating a small interface where we decide whether to run LBP or Deep learning first and we also added the possibility to view pictures of the results obtained.(Figure 2).

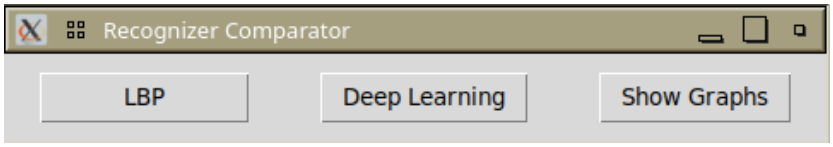

Figure 2: User Interface

3.1 Local Binary Pattern

Starting from an existing project [2], this first implementation aims at observing how the LBP, that works really well under certain constraints, works with a dataset of images in not ideal conditions.
Before being able to perform facial recognition, it is essential to locate the face within an image.
Face Localization and Recognition were developed from the existing project, but all management of the dataset, outcomes categorization and evaluation process have been entirely implemented from scratch.

3.1.1 Face Localization

Face localization involves detecting the presence of faces in an image and determine their locations. A computer program that categorizes an image as either positive (containing a face) or negative (not containing a face) is called a *classifier*. These classifiers undergo training using a large dataset of face and non-face images, enabling them to accurately classify new images. OpenCV offers two pre-trained face detection classifiers: the Haar Classifier and the LBP Classifier [4].

The Haar Classifier, created by Paul Viola and Michael Jones, utilizes a machine learning technique called AdaBoost for face detection. In this method, various features are extracted from training examples, and the most discriminative ones are selected. The detection process involves sliding a search window of different sizes across the image. For each window, the classifier evaluates whether it contains a face or not based on the extracted features. This approach allows for effective localization of faces in images by leveraging a learned model that focuses on the most relevant features for accurate detection.

On the other hand the LBP (Local Binary Patterns) Classifier serves as a visual and texture descriptor employed in computer vision for image classification. This classifier assigns labels to pixels by comparing the intensity of a pixel with its neighboring pixels and expresses the outcome as a binary number. Subsequently, LBP features are extracted to form a feature vector, which is utilized in the classification process to determine whether the image contains a face or not.

Each OpenCV face detection classifier has its strengths and weaknesses. So if more

accurate detections is needed, then the Haar Classifier is recommended. However, the LBP Classifier is faster. In our project the LBP Classifier was chosen and used in the function *face_detect()* in Figure 3.
Since LBP works on gray scales, while LFW provides color images, it was chosen to convert the color of the images to gray.

```python
def detect_face(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    face_cascade = cv2.CascadeClassifier('opencv-files/lbpcascade_frontalface.xml')

    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5)

    if len(faces) == 0:
        return None, None

    (x, y, w, h) = faces[0]
    return gray[y:y+w, x:x+h], faces[0]
```

Figure 3: detect_face() function

3.1.2 Face Recognition

OpenCV supports the Face Recognition by providing a built-in face recognizers: LBPH. Then after enrolling the images in the gallery, that is, associating the valid labels corresponding to each person (ground truth) and after detecting their faces , the function **LBPHFaceRecognizer_create()** was used to create the object of a face recognizer. The method **create** automatically sets neighbors value as 8 and radius value as 1 [1] where:

- Radius is used for building the Circular Local Binary Pattern. The greater the radius, the smoother the image but more spatial information you can get.
- Neighbors is the number of sample points to build a Circular Local Binary Pattern from. An appropriate value is to use 8 sample points. Tthe more sample points you include, the higher the computational cost.

In this case we made an attempt by changing the value of neighbors with 12 but the result was pretty much the same that we had with 8, with the difference that the time taken to execute it increased greatly.
Once the gallery has been prepared to be parsed from the face recognizer, the face recognition step is taken, through the function **predict**, to identify the faces of the new images in the probes. This is done with the following snippet:

```
label, confidence = face_recognizer.predict(face)
```

So we will have in output the confidence describing the distance between the image in the current probe, and the identity images corresponding to the label in output.
To understand the accuracy with which the label is predicted, thus identity recognition, the confidence obtained is compared with a set of thresholds. The purpose is to figure out which threshold is best suited to achieve better performance.

Setting up a loop to input a substantial number of thresholds, the process involves calling the *calculate_metrics* function for each threshold and every image in the probes. The *calculate_metrics* function is designed to compute the number of Genuine Acceptances, Genuine Rejections, False Acceptances, and False Rejections for each threshold. This systematic approach allows for a comprehensive evaluation of the system's performance.

- **Genuine Acceptance** is incremented when the confidence value is less than or equal to the current threshold and the label in the output of the predict is equal to the label of the subject of the probe being analyzed.
- **False Acceptance** is incremented when the confidence value is less than or equal to the current threshold but the label in the output of the predict is different from the label of the probe being analyzed.
- **Genuine Rejection** is incremented when the value of the confidence is greater than the current threshold and the labels returned by the predict and of the subject of the current probes are actually different.
- **False Rejection** is calculated through DIR.

If we are in the case of the Genuine Acceptance, so if the recognition is successful, a rectangle identifying the face and the identity label belonging to the face will be visible on the probes image (as you can see on Figure 4 where there are only few of recognized probes).

Here we would like to mention that we did multiple trials, for multiple times and for multiple subjects, changing the pictures in the probes and in the gallery. An interesting case was the one of Calista_Flockhart, using a picture (Picture 5) in which the position is not ideal and the face is partially turned around, the identity is not recognized while, by changing the picture as in Figure 4, the face is found and recognized correctly by the face recognizer.

3.2 Deep Learning with DeepFace

About deep learning instead, we chose to use the DeepFace module of Python's Deepface library. The development of this part of the project was entirely carried out by us taking

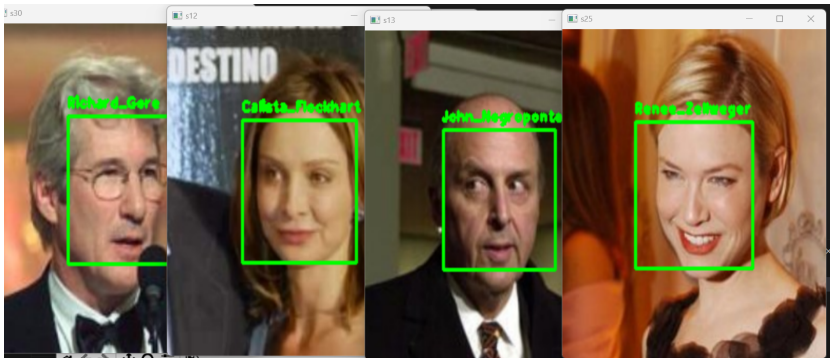

Figure 4: Examples of Recognized People

Figure 5: Calista is not recognized with LPB

as reference only the official deepface documentation [3] and a few other resources. [6]. In this case, a *face_recogn()* function was written to realize both the Detection and Recognition functions. The deepface tool turned out to be very useful and easy to use since it provides various functions to accomplish what in objective was to be done. In fact, we decided for the face recognizer to be implemented to use the existing **find** function of the library. Basically this function already has default settings to which we adjusted, for example:

- deepface uses several deep learning models for facial recognition, the default one being the **VGG-Face** model. VGG-Face is a convolutional neural network (CNN) model designed for facial recognition. The advantage of using models like VGG-Face is that they have been trained on large datasets of facial images and have learned to automatically extract discriminative features useful for facial recognition.

- Face detection and alignment are important early stages of a modern face recognition pipeline. The default face detector for deepface is **OpenCV** and we decided

to use it as is the same we used also in LBP.

- Another decision to take before going on is the type of distance metric. Face recognition models are regular convolutional neural networks and they are responsible to represent faces as vectors. We expect that a face pair of same person should be more similar than a face pair of different persons. Similarity could be calculated by different metrics such as Cosine Similarity, Euclidean Distance and L2 form. In our case we used the metrics of **cosine distance**, it means that the cosine varies between 0 and 1, where 0 indicates that the vectors are perfectly similar and 1 indicates that the vectors are completely different so, a lower cosine distance value indicates closer or similarity between the vectors.

The organization of probes and gallery considered for deep learning is the same as the one we used for LBP therefore the logic for the realization of the face recognizer is more or less the same as the one used with LBP, with the only variation in handling and reprocessing the data received from the face recognizer. In fact, the **find** function, after searching for the indentity of the input image in the gallery, will return list of pandas data frame as output.

Once the results of the face recognizer have been reprocessed, it is possible to calculate the number of Genuine Acceptance, False Rejection, False Acceptance and Genuine Rejection by cycling over all the possible probes and over a range of thresholds (for example in this case [0.01,0.99]) and going for each threshold to check whether the distance with the true identity is below the current threshold exactly as we reasoned in the case of LBP. We only checked the first result in the list of images and distance from the images without considering the distances from the other images so, as we will see in the next section, the Detection and Identification Rate will be calculated only for rank 1.

Speaking about our previous example in Fig. 5, LPB wasn't able to find out a face for that subject.

Due to the different nature of DeepFace, we're now able to correctly detect the face of Calista, as shown below.

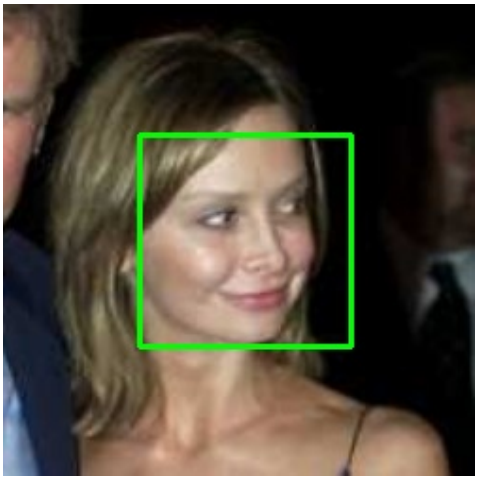

Figure 6: Deepface at least correctly detects Calista

Moreover, all the images have been merged into a single one, to show Deepface results (Figure 7

In this case many more faces than in LBP are not only located but also recognized, although if we look more closely not all faces are detected correctly.

Figure 7: Deepface results

4 Performance Evaluation

The evaluation part was not developed by the authors of the starting project so it was realized entirely from us. Our aim in this phase is to plot and analyze different graphics for both Deep Learning and LBP and highlights differences when the same dataset is used.

To achieve a comprehensive and consistent overall assessment, it is not enough to merely calculate the total number of true positives, false negatives, etc., instead it is necessary to calculate the following rates:

- **Detection and Identification Rate DIR(t,k)**: the probability of correct identification at rank k (the correct subject is returned at position k).
- **False Acceptance Rate (FAR)**: False Acceptance Rate is the rate at which impostor faces are incorrectly accepted. It is defined as the ratio of the number of False Acceptances to the number of Impostor Probes.
- **False Rejection Rate (FRR)**: False Rejection Rate is the rate at which genuine faces are incorrectly rejected. It is defined as the ratio of the number of False Rejections to the numberof Genuine Probes. Notice that FRR(t) = 1-DIR(t,1)
- **Equal Error Rate (ERR**: Error Equal Rate is the threshold value at which both False Rejection Rate (FRR) and False Acceptance Rate (FAR) are equal. It provides a measure of the performance of the face recognition system when the threshold is set such that both error rates are equal.

Once these values have been calculated, it will be possible to plot graphs that will be an indication of the rate changes as a result of threshold changes.

4.1 Results

The purpose of this section is to see how rate errors vary as thresholds change.

The implementation of this phase was also possible thanks to the NumPy and matplotlib libraries of OpenCV.

Let's start by plotting the Receiver Operating Characteristic (ROC) graph, which is a useful tool for evaluating the performance of a model. The ROC curve displays the relationship between FAR and 1-FRR as the decision threshold changes. The area under the curve measures the overall discrimination capability of the model, with a larger area indicating better performance. In the case of LBP, the ROC is shown in the following picture (Figure 8):

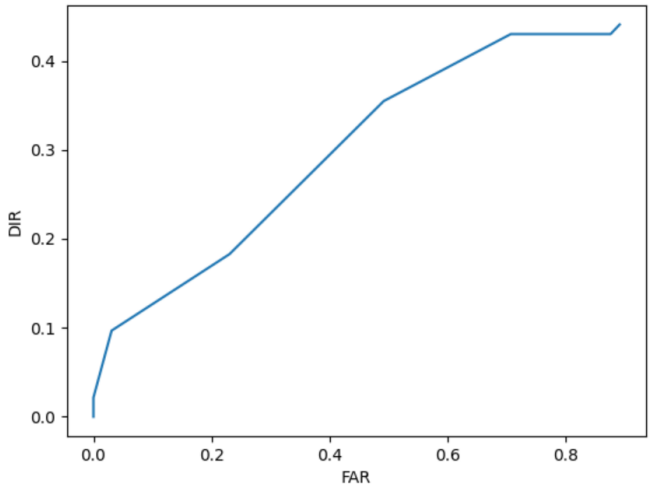

Figure 8: ROC Curve for LBP

The previous ROC curve compared with the ROC curve obtained from deepface (Figure 9), indicates that using LBP on a dataset like LFW does not yield optimal results due to the low area under the curve, in comparison with deepface which shows a significantly greater area underneath.

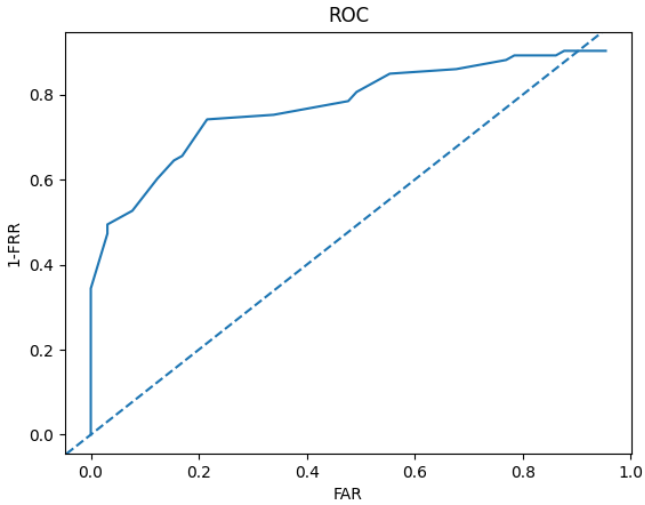

Figure 9: ROC Curve for DeepFace

Now let's consider the Error rate/threshold graphs which plot the curves representing FAR and FRR at varying thresholds. Let us examine the case of LBP in Figure 10. The green curve represents the FAR while the blue curve represents the FRR. As the confidence of LBP is estimated as 'distances', increasing the threshold decreases the False Rejection Rate but increases the False Acceptance Rate until they intersect at the Equal Error Rate (EER) where FRR=FAR. It is important to note that having

a low false acceptance rate is preferable to having a low false rejection rate. This is because a high false rejection rate can negatively impact the user experience, whereas a false acceptance rate can be more harmful as it may allow malicious users to bypass security measures.

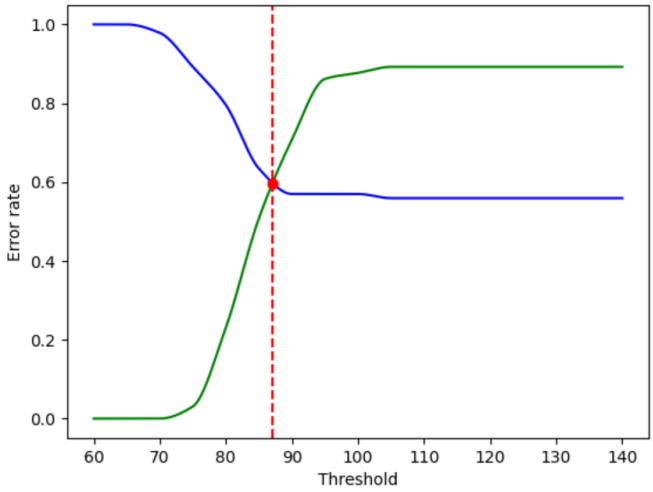

Figure 10: FAR and FRR curves for LBP

Regarding the deepface, the Error Rate on Threshold graph produced the following results:

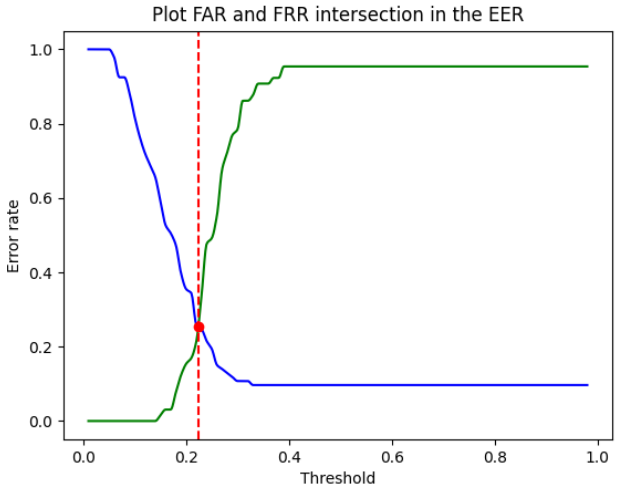

Figure 11: FAR and FRR curves for Deep Learning

Note that in this case, the thresholds have different values from those of LBP because DeepFace uses different distance values than those used by LBP.

As can be seen by comparing the two graphs, the use of deep learning methodologies achieves an error rate (between 0.2 and 0.3) that is much lower than the error rate achieved by LBP (about 0.6) at the point where false rejections and false acceptance

are minimized and this is for us indicative of better results from the use of deep learning methods in the specific case of this dataset.

5 Conclusions

Our system doesn't offer a detector for a real specific application but underlines the importance of the approach used and it highlights the great difference between the 2 approaches taken into account when dealing with dataset like LFW; it is evident that both approaches exhibit advantages and limitations, which can be assessed based on the specific needs of the application.
The LBP approach, based on local texture descriptions, proved effective in scenarios where training data availability is limited, and computational resources are more constrained. However, its ability to address complex variations in facial features may be limited, especially in contexts with diverse lighting and angles.

Deep Learning, particularly with models like VGG-Face, excelled in representing and recognizing complex patterns in facial images.
The dataset size and problem complexity play a crucial role in performance differences. In scenarios where the LFW dataset is limited or facing particularly challenging conditions, the Deep Learning approach can offer significant improvements.
The LBP approach is considerably faster and requires fewer computational resources compared to Deep Learning models. This feature makes it attractive for real-time applications or environments with resource constraints.

On the other hand, Deep Learning models provide greater flexibility, making them ideal for contexts where precision and the ability to handle complex variations is crucial. In conclusion, the choice between LBP and Deep Learning depends on the specific needs of the application, problem complexity, and available resources.

6 Future Development

We would like to suggests that a strategic combination of approaches might offer optimal results in certain contexts. Integrating LBP techniques for quick and lightweight initial identification, followed by Deep Learning models for refined recognition, could represent a possible perspective for future developments.
The final consideration pertains to the prospect of incorporating a live capture option. While acknowledging that its implementation would necessitate additional testing and code adjustments, this enhancement could introduce significant functionality. It would be possible to do some tests with live samples and perhaps propose a kind of competition between LBP and Deep Learning in terms of speed of response and adequacy of response.

References

[1] https://docs.opencv.org/.

[2] https://github.com/informramiz/opencv-face-recognition python. opencv-face-recognition-python.

[3] https://pypi.org/project/deepface/.

[4] https://www.superdatascience.com/blogs/opencv-face detection.

[5] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. (07-49), October 2007.

[6] Sefik Ilkin Serengil and Alper Ozpinar. Lightface: A hybrid deep face recognition framework. pages 23–27, 2020.