Universidade Federal de Minas Gerais

Facial Detection, Tracking and Recognition in Android Devices

Report submitted to LGE as part of the Project *Scheduling heterogeneous tasks on heterogeneous devices environments.*

Belo Horizonte, MG
April, 2016

Facial Tracking and Recognition Application

1 Introduction

In this report we describe the application we're using to demonstrate the power of our syntax and framework. It's an augmented reality application that will be developed to, in a live video fed by the smartphone's camera, detect, track, recognize and display information about the faces shown. Several opportunities to paralellize data processing appear in this context of visual computing and machine learning. In the subsections below we will briefly address what have been the works in the literature regarding what we intend to do, we'll specify the application in detail and report on the progress towards it's development.

Our bibliographical focus here is a broad approach on facial detection and processing and in facial recognition - the full range of the topics we're going to address is too extensive to fit well in this report. More information regarding techniques and design choices are presented as the topics are specified later on.

2 Bibliographical Studies

Facial detection has been the subject of many studies in recent years. It's applications are numerous, such as in security systems, image search services, among others. However, the results obtained by current methods are still below those obtained by humans. Challenges such as partial occlusion, rotation at different angles and lighting differences are still a major challenge for this area of study. In this section, we present some of the main research fronts in facial detection.

2.1 AdaBoost-Based Methods for Facial Detection

The work by Viola and Jones [1] in 2001 was the first to make real time face detection possible. The method is appearance based and concentrates only on the facial detection problem, and not on the recognition one. Most modern algorithms are based on the Viola-Jones object detection framework.

The facial detector contains three main ideas that enables it running in real time: the integral image, classifier learning with AdaBoost and the cascade structure. The main idea is use simple haar-like features [1], powerful for facial/nonfacial classifiers. Haar-like features are reminiscent of Haar basis functions which have been used by [2]. The value of a feature is the difference between the sum of the pixels within rectangular regions.

The algorithm for computing and work with a haar-like feature is:

Viola and Jones built an algorithm that, in short, looks first for vertical bright bands in an image, that might be noses, it then looks for horizontal dark bands, that might be eyes, and finally

```
Pick a feature scale  (example: 24 x 24 pixels)
Slide it across the image
Compute the average pixel values under the white areas and black areas
If the difference between the areas is above some threshold, the feature matches.
```

Listing 1: Computing a haar-like feature

looks for other general patterns associated with faces. Detected all by themselves, none of these features are strongly suggestive of a face. But when they are detected one after the other in a cascade, the result is a good indication of the presence of a face in the image.

The integral image, based on the summed area table [3], is an algorithm to compute quickly and efficiently the sum of values in a rectangular subset of a grid. The value at a position x, y contains the sum of all pixels above and to the left of x, y. It allows to make the sum of the pixel values of any rectangular region in the image be calculated with only two additions and one subtraction. It leads to enormous savings in computation time for features at varying locations and scales. [4] In Viola-Jones the integral image allows haar-like features to be quickly calculated and are essential for the algorithm to be executed in real time. Since these tests, in short, are all simple to run, the resulting algorithm can work quickly in real-time.

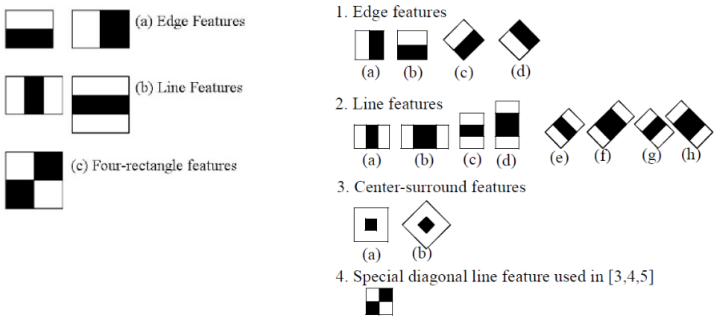

Figure 1: Viola and Jones' features, on the left, and Haar-like features proposed by Lienhart

Boosting is a method of finding a highly accurate hypothesis by combining many "weak" hypotheses, each with moderate accuracy. The AdaBoost combine the series of weak classifiers. It tries out multiple weak classifiers over several rounds, selecting the best weak classifier in each round and combining the best weak classifiers to create a strong classifier.

Cascade structure is a critical component in the Viola-Jones detector. The key insight is that smaller, and thus more efficient, boosted classifiers can be built in a way which rejects most of the negative subwindows while keeping almost all the positive examples. Consequently, the majority of the subwindows will likely be rejected in the early stages of the detector, making the detection process extremely efficient.

A number of researchers noted the limitation of the original Haar-like feature set for multi-view face detection, and proposed to extend the feature set by allowing more flexible combination of rectangular regions. Researchers have also proposed other Boosting techniques like RealBoost and GentleBoost, LogitBoost, and many others. A survey with more details can be seen in [5].

2.2 Others Methods of Facial Detection

In addition to the research inspired by the method proposed by Viola-Jones, researchers have also evaluated the use of other techniques for facial detection.

A multi-view face detection using deep convolutional neural networks is presented in the paper [6], in which the proposed method does not require pose/landmark annotation and is able to detect faces in a wide range of orientations using a single model based on deep convolutional neural networks.

The key ideas of this work are: 1) leverage the high capacity of deep convolutional networks for classification and feature extraction to learn a single classifier for detecting faces from multiple views and 2) minimize the computational complexity by simplifying the architecture of the detector.

The face classifier, based on AlexNet [7], consists of 8 layers where the first 5 layers are convolutional and the last 3 layers are fully-connected. The results are similar to state-of-the-art even without using pose annotation or information about facial landmarks.

2.3 Template Matching for Facial Tracking

Template matching [8] is a technique in digital image processing for finding small parts of an image which match a template image. It works by sliding the template image over the image. Once the patch has been tested in all the possible locations in a pixel-by-pixel basis, a matrix containing a numerical index according to how good the patch matches in each location is created.

This technique is useful when the object to be tracked, a face for example, is moving along the scene but always keeping the same appearance. If the object changes its appearance, the matrix resulting will not have a clear peek once there will not be a very clear location on where the object best matches, and this will lead to errors. The "matching error".

In any case, this algorithm always returns a value, unlike Haar cascades which is returns a position only if it finds a face, the template matching could find a matching even with some changes in its appearance.

2.4 Facial Recognition

The problem of creating a computational system that correctly identifies the face of someone in an image has been in study since the 1960's [9], and the basic idea is to use a set of specific features to represent one's face, like in an n-dimensional vector. [10] uses 21 subjective features to describe someone's face, like height of eyebrows, nose length, among others, for example.

In the last 20 years, specifically, this problem has been extensively studied [11], and several approaches to it's resolution were created, with methods based on extraction and classification of features, mainly, with much being done with the combination or extension of algorithms previously devised. The use of Machine Learning techniques to extract facial features is common [12][13], but most of these require careful manual adjustment.

In 1996, a technique of compressed image representation was proposed [14], done through reconstruction error minimization with the use of a neural network of reduced dimensionanity. This network is called autoencoder, a network built to 'encode' it's input in it's reduced dimensionality hidden layers. To put it simply, the autoencoder is a neural network that receives an image as input, codifies it into a usually smaller hidden layer, and reconstructs it using that hidden layer as basis, comparing it's result to the input itself. It then iterates to minimize the reconstruction error. This technique has been evolving [15] and has been showing promising results [16][17].

The idea behind the autoencoder is that, by trying to create a good minimized representation of an image, it learns to select the most important attributes of it automatically. In our application, it will be the technique of choice. It's capability of representing an image with good quality with reduced dimensionality makes it perfect to be used in a mobile environment, where computational resources and network capabilities are limited.

3 Specification

In this section we present the specification of the application we're developing to demonstrate the effectiveness of our framework. Putting it simply, it's an application that gets image frames from a video feed, detects and crops the faces in it, recognizes it based on a database of offline-loaded known faces and outputs the result accordingly.

At this stage, the application is partially developed in two separate modules: the part doing the video frame extraction, facial tracking and the augmented reality output is in an app, already functional, and the part regarding the facial recognition was developed as a separate app. The facial recognition itself does not yield good results yet due to lack of finetunning network parameters, as it will be explained below, but the app itself is also fully functional as a prototype.

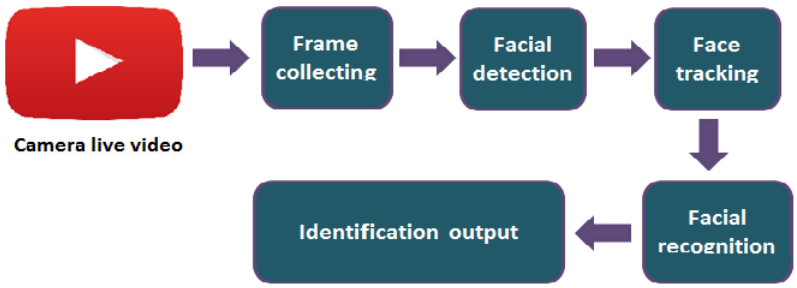

Figure 2: Application overview

The application is suited for our purposes because it relies heavily in computer vision and machine learning algorithms, both of which require heavy processing power and are not only easily parallelizable, but are fit to execute in GPUs as well, as exemplified in the sections above. It's divided in several steps that will be deeply described in the sections below.

3.1 Algorithm's Overlook

Figure 3 exhibits a scheme of the algorithm in development, as it is planned to be today.

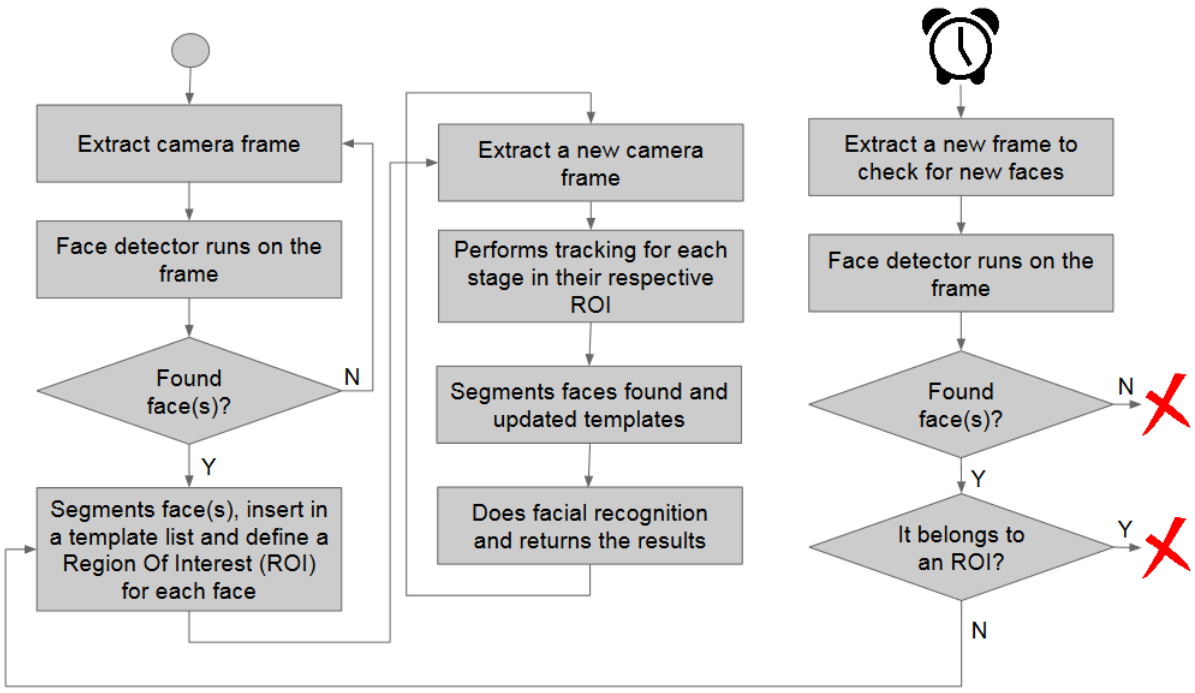

Figure 3: Application's algorithm

Basically, in the beginning the app extracts a frame from the camera, in which it runs, then, the face detector. If faces are not found, frames keep being extracted and the detector keeps looking. If faces are found, it is segmented and inserted in a template list. A Region of Interest (ROI) is defined for each face. The template stores the set of patterns that defined the face, whilst the ROI defines the region in which the face is most likely to be in in the next frames.

With faces found, the algorithm enters a loop, in which it keeps extracting frames and tracking the faces in their respective ROIs, while segmenting faces found to update the templates. On this stage, it will send the faces to the facial recognition module to get it's information to project on the screen. Meanwhile, outside of the loop, in parallel, from time to time a new frame is extracted and is checked for new faces, as it was done in the first place. New face patterns, as these only, are inserted in the loop with the others.

The facial recognition uses an autoencoder trained offline, and needs only the cropped image of a face to work. In the sections below each part of the algorithm will be thoroughly explained.

3.2 Video frames collection

The video frames are colleted from the device camera, which could be either the back or the front camera.

In this step, the frame is downscaled (initially to 320 x 240 pixels). Downscaling the frame

can speed up the detection with a trade-off is precision. The specific parameters of this operations will be subject of experimentation. The idea is to have an application as precise as possible while operating on real time.

3.3 Facial detection

A face detector based on the Viola-Jones object detection is executed in order to detect the faces of the in the frame.

The facial detector receives as input parameter a frame and as a output parameter returns a list of faces found in the frame and their positions. As long as at least one face is not found, the detector runs incessantly. After one or more faces are found, the frequency of the detector decreases, but continues running looking for new faces in the frame.

3.4 Face tracking

Once a face has been found, the area where this face is located will be taken as the template to match in the following frame and a region of interest (ROI) will be set up around this area. In the next frame, the template matching will run only in the area defined by ROI. Since the video comes from the camera in real time, processing a reasonable number of frames per second, we assume that a face does not move a lot from a given frame to the next one taken.

After a face has been found by the face detector it will be tracked by the template matching, and during that the face detector doesn't need to run. We'll have a considerable performance boost here, in comparison to the traditional face tracking algorithm, because there is much less space for the template to look.

The best area matched by the template matching algorithm, running only over the ROI, is taken as a new template, and a new ROI is set around this area. In every frame the template is updated with the best match of the previous template inside the present ROI, unless no good area is found.

3.5 Facial recognition

In the facial recognition step, the rescaled cropped image of a face will be received as input, and an ID tag or an 'unknown face' code will be returned as output. For this, the autoencoder will be used, and whether this processing will happen in the smartphone or in a server is still to be determined.

The basic idea is to create an algorithm that receives an image as input and codifies it with an autoencoder in a low dimensional vector that aggregates automatically the most determinant features of a face. The image is transformed, beforehand, in a vector of integer numbers that represent the pixels to be "codified".

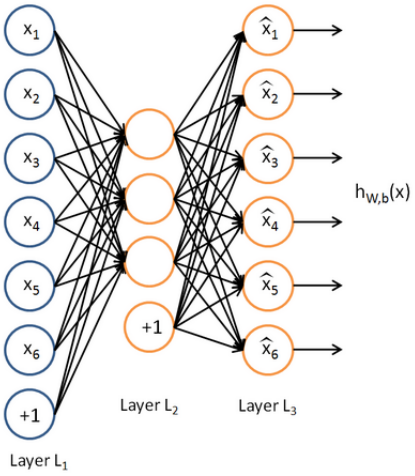

Figure 4: An Autoencoder with one hidden layer

The facial detector works in two steps: first, unsupervised training is done, offline. A thousand or so images of people's faces, in a wide variation of situations, are input in the network. The network is trained layer by layer in such a manner that it searches for the pattern in common in all pictures, which in our case is the presence of faces. It learns, therefore, to find faces in images. Then, supervised training done offline: 5 to 10 face images of several known people are fed into the algorithm, and the autoencoder iterates to minimize the reconstruction error of all of them with the same set of parameters (weights and bias). This is done so that the autoencoder learns to differentiate one face of the other becoming, therefore, a facial recognizer.

After that, online, when a new face is received, the autoencoder uses this set of parameters to encode the face into it's hidden layer representation. This encoding is then compared to the ones of known people in the database, and the closest match or, in other words, the stored picture of a known person whose hidden layer representation is closest to the one inserted, is tagged as the face in the picture. If no face inserted is similar to a certain threshold, the face is tagged as 'unknown'. An example of the Java code that generates a representation for a given face image follows below, written with our framework:

In the example referred in the listing 2, the code would've been even simpler to write if lambda notation was used. There, each pixel of the image is multiplied by an amount of weights equivalent to the number of "nodes" in the hidden layer, and all the multiplications results are summed. Our code paralellizes every operation.

In the application we have two basic design choices on how we'll do this processing: (1) the rescaled image could be sent to a server to be processed there, and the application would receive the output; (2) the image could be transformed, with the autoencoder parameters, into it's representation, and that representation could be sent into a server to be compared with the database of representations.

That design choice will be made based on the results of the experiments we'll run at a later stage. We'll use our framework to execute the model generation efficiently using CPUs and GPUs, but if that proves to be too much of a consuming task to the phone, either regarding processing power of battery consumption, we might switch to server processing. Either way, many other tasks

```
    private static void mult(final FlatArray2<LayerNode> Input, final
    WeightsTable Weights, final FlatArray2<LayerNode> Layer1) {
       Layer1.foreach(new UserFunctionWithIndex2<LayerNode>() {
           @Override //Outer loop
           public void function(final ElementWithIndex2<LayerNode> eixLayer1) {
               Input.foreach(new UserFunctionWithIndex2<LayerNode>() {
                   @Override //Inner loop
                   public void function(final ElementWithIndex2<LayerNode> eixInput) {
                       eixLayer1.element.setWeight(eixLayer1.element.getWeight() +
                       (eixInput.element.getWeight() * Weights.getField(eixInput.y,
                       eixLayer1.y))); //Input layer * Weights = Output layer
                   }
               });
               eixLayer1.element.setWeight((float) //Sigmoid function
               (1/(1+(Math.exp(-eixLayer1.element.getWeight())))));
           }
       });
   }
```

Listing 2: Creating a hidden layer representation of a given face image

to be necessarily processed in the phone we'll be done using our framework as well, so that shouldn't compromise our application's purpose of demonstrating the efficiency of our framework.

3.6 Augmented Reality Output

The output of the application is the identification of the faces seen at this moment in the video feed. Since it is a video feed, the output should be presented in a certain frequency, that reflects the changes in the video in real time. That means that performance is critical to our application, since it should be capable of processing from beginning to end in, ideally, less than a second, for a minimum refresh rate. Whether we'll be able to achieve that or not is something to be analyzed.

4 Development's Progress

Application's development has progressed well and is almost finished, save one bottleneck: the facial recognition done with the autoencoder. As described in 3.5, the autoencoder is trained is two steps: it first learns to recognize generic face templates, and then it learns to recognize specific faces. Our network is being trained in a desktop computer that runs on Python and uses theano [18] [19] for the heavy training processing. The problem we have is that, even when our training yields good results, these aren't reflected in the online recognition.

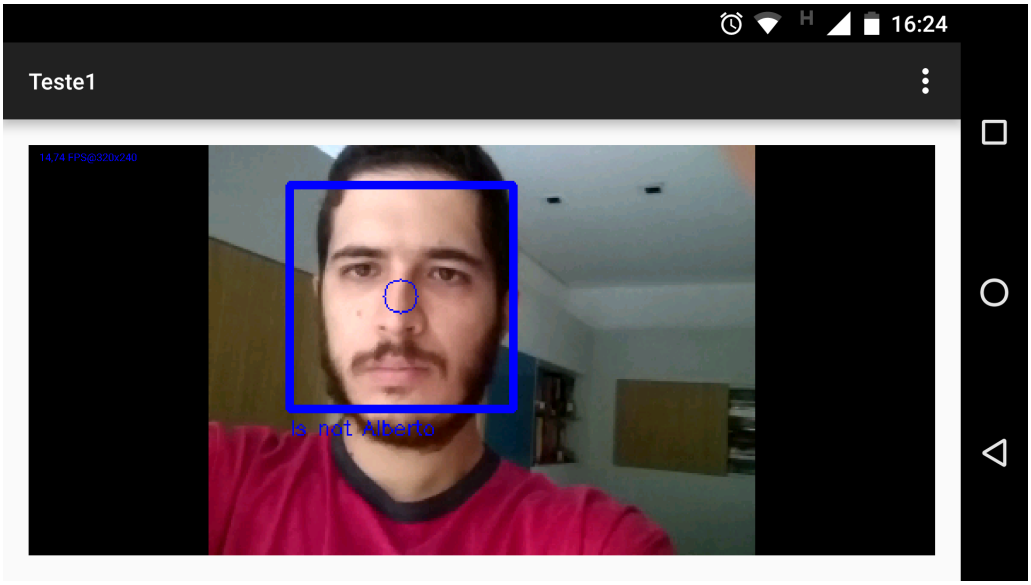

Figure 5: Application running in an Android device

The first step of the training is to treat the face images, so they are converted to a format the Python code can read: we resize them, normalize the pixel color values to a *float32* number between zero and one and, finally, convert the pixel matrix to a pixel array that is sent to the server for processing. The images to be used in the training are converted in the Android device, using the same libraries that will be used in the online recognition. This is done, naturally, so that differences in library implementations do not compromise our results. In short: the image conversion done in the application is the same one done during the training.

Next, the network is fully trained in the Python code and, as part of the training algorithm, has it's training tested, to affer the quality of the recognizer created. Being the results good enough, the parameters of the network are transferred to the network in the Android application itself, and tests are, then, executed in this network. That's where our problem lies: even though the numbers and libraries used are strictly the same, the application's network doesn't reflect the results we've had after training in the desktop computer. We have still to track the source of that problem.

It's good to emphasize, however, that the recognition's performance is irrelevant when regarding the applications purpose of being a proof of concept for our parallel programming framework. That happens because what affects the performance of the recognition are the network parameters, obtained in Python's training and transferred to the device afterwards. Regardless of whether these parameters are adequate or not, the same number of calculations is done all the times. Being this a very performance-demanding algorithm, this means that the application successfully and effectively uses the framework to increase it's performance.

4.1 Next Steps

We intend to get the bottom of the problem that is preventing the network from working. Research is being done in that regard and tests using different training databases are being done. It's

important to note that, as of now, we are considering that the autoencoder might not be an ideal network for facial recognition even though in the Python training it, apparently, performs well. Thinking of that, we'll also consider changing our network from an autoencoder to a Convolutional Neural Network [20] [21], a network model well tailored to do facial recognition.

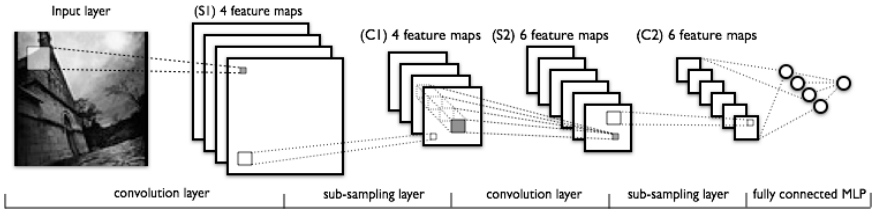

Figure 6: A convolutional neural network

Without delving too much into detail, a Convolutional Neural Network is a network whose input is the image itself and, in each layer, each neuron recieves as input a a "square" of pixels formed by the layer below. That means that, in the first conversion, a square of pixels will be united and sent as one value to the neuron above. In the next layer, a set of neurons that would represent a square in the newly formed image is united and the process repeats. Image 6 exhibits this process. The "square" each neuron recieves, also, overlaps with it's neighbors, so that less information is lost in the union of pixels. The resulting image, in the last layer, contains a specific pattern that represents the image with it's most "highlighted" aspects. If the autoencoder problem is not solved, we intend to use a CNN to do the trick.

Another feature under development is a client-server interface for the app to communicate with the network training desktop computer, so that each user could install the server in it's personal computer and, through the application, train the network to recognize it's own face.

References

[1] P. Viola and M. Jones, "Fast and robust classification using asymmetric adaboost and a detector cascade," *Advances in Neural Information Processing System*, vol. 14, 2001.

[2] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Computer vision, 1998. sixth international conference on*, pp. 555–562, IEEE, 1998.

[3] F. C. Crow, "Summed-area tables for texture mapping," *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 207–212, 1984.

[4] A. K. Jain and S. Z. Li, *Handbook of face recognition*, vol. 1. Springer, 2005.

[5] C. Zhang and Z. Zhang, "A survey of recent advances in face detection," tech. rep., Tech. rep., Microsoft Research, 2010.

[6] S. S. Farfade, M. Saberian, and L.-J. Li, "Multi-view face detection using deep convolutional neural networks," *arXiv preprint arXiv:1502.02766*, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[8] R. Brunelli, "Template matching techniques in computer vision," 2008.

[9] W. W. Bledsoe and H. Chan, "A man-machine facial recognition system—some preliminary results," *Panoramic Research, Inc, Palo Alto, California., Technical Report PRI A*, vol. 19, p. 1965, 1965.

[10] L. D. Harmon, A. B. Lesk, *et al.*, "Identification of human faces," *Proceedings of the IEEE*, vol. 59, no. 5, pp. 748–760, 1971.

[11] P. F. De Carrera and I. Marques, "Face recognition algorithms," *Master's thesis in Computer Science, Universidad Euskal Herriko*, 2010.

[12] L. Sirovich and M. Kirby, "Low-dimensional procedure for the characterization of human faces," *JOSA A*, vol. 4, no. 3, pp. 519–524, 1987.

[13] H. Fan, Z. Cao, Y. Jiang, Q. Yin, and C. Doudou, "Learning deep face representation," *arXiv preprint arXiv:1403.2802*, 2014.

[14] B. A. Olshausen *et al.*, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[15] M. Chen, K. Q. Weinberger, F. Sha, and Y. Bengio, "Marginalized denoising auto-encoders for nonlinear representations," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1476–1484, 2014.

[16] A. Krizhevsky and G. E. Hinton, "Using very deep autoencoders for content-based image retrieval.," in *ESANN*, Citeseer, 2011.

[17] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Artificial Neural Networks and Machine Learning–ICANN 2011*, pp. 44–51, Springer, 2011.

[18] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements." Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[19] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[21] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: a convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, pp. 98–113, Jan 1997.