

Máster Universitario en Aplicaciones Multimedia

Trabajo Fin de Máster – Anexo

Application of Augmented Reality to the recognition of historical figures portraits

Eugenio P. Concepción
Universitat Oberta de Catalunya
econcepcion@uoc.edu

1 Introduction

The recent advances in mobile AR technology are bringing us new and exciting ways to discover location-related information about the environment in which we are getting on. An AR application allows us to get contextual information overprinted in an object, instead of needing to look for that information in a book, a guide, a web or any other source.

Specifically, in the field of exhibition, these technologies bring a really novel way of information presentation. If traditional means of data displaying like sign boards or graphics are passive elements and can become arid and unspectacular for the visitors, new AR applications can allow the museums bringing visitors guidance, virtual objects reconstruction, digital art performances, or even a commercial exhibition showcase.

The prototype described in this report aims to serve as a basis for a later application that will allow the visitors of a museum to get biographical data about the historical figures depicted in the exhibited paintings.

This prototype is focused on testing several face recognition techniques in order to decide which is the most suitable for painting analysis. It will apply different techniques, according to a plan, looking for perform later a survey among the users and evaluate aspects such as performance, accuracy or the global users' satisfaction.

The final result of the research process should be an application that takes advantage of the conclusions drawn after the testing of the prototype.

Ideally, this final application interface would look like the screenshot shown in figure 1. User could take a picture of a painting using a smartphone, and the system should detect the face of the portrayed figure, recognize him, and then overprint a window with biographical data.



Figure 1: Ideal screenshot of final application

2 Objectives and scope of the application

The prototype should ideally serve as sandbox for testing main face recognition techniques under real world conditions. The design counts on giving the users the ability to test a wide variety of techniques.

Apart from testing the accuracy of the different algorithms and the overall user experience, the prototype should serve for measuring the performance of the techniques and generating a statistical report.

The minimum planned scope would include the ability of the prototype to capture images, detect the existing faces in the painting, and selectively apply Eigenfaces, Fisherfaces and Linear Binary Patterns techniques. To the extent possible, the final prototype should include any other feature-based techniques and statistical algorithms from image-based category.

Finally, the selected technology for developing the prototype has been Java. The main reason for the choice is that the prototype needs to run in a mobile device, and Java is currently the most extended platform for these devices.

3 Process

3.1 Conceptual approach

Face recognition is the process of putting a name to a known face. In the same way as humans learn to recognize their people just by seeing their face, automatic recognition process needs to be trained for performing this task.

Conceptually, this process involves four main steps:

1. Face detection: It is the process of locating a face region in an image (as shown in figure 3). This step does not care who the person is, just that it is a human face.
2. Face pre-processing: It is the process of adjusting the face image to look more clear and similar to other faces (a small gray scale face in the top-centre of the following screenshot).
3. Collecting and learning faces: It is the process of saving many pre-processed faces (for each person that should be recognized), and then learning how to recognize them.
4. Face recognition: It is the process that checks which of the collected people is most similar to the face in the camera (a small rectangle on the top-right of the following screenshot).

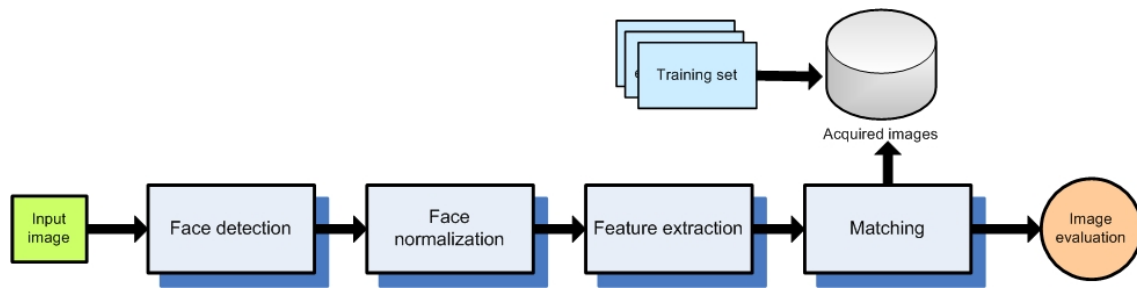


Figure 2: Abstract recognition process

To perform the recognition step accurately, the system has to be previously trained. The training process is probably the fundamental pre-requisite of successful face matching.

Figure 3 shows a visual synthesis of how training process would be in the proposed prototype. Firstly, in order to create an adequate knowledge base, it is necessary to gather as many images of the historical figure as possible. These images should ideally have been taken from different portraits, showing different poses, facial wear and even different age. It would be also useful have pictures of the same painting, but taken under diverse lightning conditions.

Once the samples have been collected, a process of normalization will be performed. In order the system can carry out the recognition all the images of the training set must be cropped to the same size and transformed into greyscale. For example, the training set of the first version of the prototype consists of a set of grey images of 70x70 pixels containing different pictures of several historical figures. Figure 3 shows some of these images, taken from Napoleon III portraits.

After having trained the system with different portraits, it is ready for performing the recognition. The first tests have been centred on images originating from the own training set. The initial results employing Eigenfaces have been satisfactory, producing a 100% of correct identifications.

The next step to be performed would be capturing real images for testing. The picture would be processed for detecting the face in the painting and cropping it in a separate image. Then, it would be resized to 70x70 pixels and transformed into greyscale. After that, the resulting image could be used as input for face recognition component.

The last stage will be performed by the face recognition module, which will apply the selected algorithm. If the face belongs to the training sample, it should be able to detect it and generate a positive response. After that, having identified the person in the portrait, the only task to do will be to show the corresponding biography. Of course, all this process will be tracked to gather statistical data about response time, performance, etc.



Figure 3: Visual summary of process applied to a portrait

3.2 Stages of the process

3.2.1 Face detection

Before the works of Viola and Jones (Viola and Jones, 2001), and Lienhart and Maydt (Lienhart and Maydt, 2002), there were many different techniques used for detecting faces, but all of them were either slow, unreliable, or both in the worst case (Baggio et al., 2012). The Haar-based cascade classifier for object detection, developed by Viola and Jones (Viola and Jones, 2001) and improved by Lienhart and Maydt (Lienhart and Maydt, 2002), is an object detector that is both fast and reliable (Baggio et al., 2012). The accuracy of this revolutionary technique reaches approximately a promising 95 percent for frontal faces detection (Lienhart et al. 2003; Baggio et al., 2012). This object detector allows real-time face detection and face recognition, working not only for frontal faces but also side-view faces (profile faces), facial features (eyes, mouths, noses), or even artificial images (for example, designs).

OpenCV offers an implementation of this cascade classifier, developed originally by Lienhart himself. Actual version was extended in OpenCV version 2.0 to also use LBP features for detection based on work by Ahonen, Hadid and Pietikäinen in 2006 (Ahonen et al. 2006), as LBP-based detectors are potentially several times faster than Haar-based detectors.

The basic idea of the Haar-based face detector is that in most frontal faces, the eyes region uses to be darker than the forehead, the nose or the cheeks; and the region with the mouth uses also to be darker than mentioned parts. The basic idea of the LBP-based face detector is similar to the Haar-based one, but it employs histograms of pixel intensity comparisons, such as edges, corners, and flat regions.

Both face detection heuristics can be automatically trained to find faces from a large set of images, storing the information as XML files to be used later. These cascade classifier detectors are usually trained using a large number of unique face and non-face images (like landscapes, trees, cars, and even text), and the training process can take a long time. Helpfully, OpenCV library contains classifiers that have been pre-trained as Haar and LBP detectors. This makes possible detecting frontal faces, profile faces, or individual features (eyes, nose) just by loading one of the different available XML files to the object detector, and choosing between the Haar or LBP detector.

3.2.2 Face pre-processing

Face pre-processing is a fundamental stage for overcoming the intrinsic obstacles associated to face recognition process: changes in illumination, pose, face expression, etc. The objective of this step is reducing differences between different images of the sample as much as possible for getting normalized image views.

The simplest way of pre-processing these images is applying a histogram equalization filter. In this case, OpenCV provides built-in functions for normalization.

Nevertheless, face pre-processing usually involves a larger number of steps. The following are supposed to be the most common:

- Geometrical transformation and cropping: this step may include scaling, rotating, and translating the image; depending on the conditions in which it has been captured.
- Independent histogram equalization for left and right side of the face. The basic idea of this step is compensating different lateral illumination.
- Smoothing the image for noise reduction.
- Elliptical mask: a final step for eliminating outside or irrelevant elements.

3.2.3 Learning step

Learning step can be as simple as collecting face images from different people and create a matrix structure with them. In order to be flexible enough, it is necessary to provide the system different images of every subject. The largest the training set is, the most accurate the system will be. Thus, images of each person can be stored in an array and each array can be allocated in another array, defining a matrix structure for representing the whole training set.

It is also very important to feed the system with images containing a wide variety of conditions (lightning, pose, expression). In the particular case of face recognition in a painting, there is an additional problem to consider. The set of available images can be quite reduced. If it is desirable to train the system using as much samples as possible, in the case of a painted portrait, the set of samples is reduced, and it is not possible to get more images for the training set. To keep a balance between the ideal initial set of samples and the available one, it has been stated that at least three images will be needed for training the system with a minimal chance of success.

3.2.3.1 Training the face recognition system from collected faces

After having collected a training set of faces for each figure to recognize, system must be trained for distinguishing them. There are several machine-learning algorithm suited for face recognition. In this prototype, there will be considered mostly those classified as image-based. The simplest one is Eigenfaces that works fine and it is easy to develop. That is the reason why it has become quite popular as the basis for new algorithms to be compared to.

3.2.3.2 Data stored from training

The results of the training process will be stored in memory. The structure of generated data will depend on the recognition algorithm to be used. In the case of Eigenfaces, training process will calculate a set of transformed images (*eigenfaces*) and blending ratios (*eigenvalues*), which can be combined in different ways to reconstruct each of the original images in the training set; and also can be used to unambiguously distinguish between the images in the training set.

Small, or not so small, differences from one face to the others will be taken on account when building the set of eigenfaces. For example, if only one of the faces in the training sample has a beard, then there would be at least one eigenface showing that, and so the

face with a beard would have a high blending ratio for that eigenface to mean that it has a beard, and the faces without a beard would have a low blending ratio for that eigenvector.



Figure 4: A set of training faces for the prototype

Let the cardinality of the people in the training set be \mathcal{N} , and let the number of faces available for each people be \mathcal{M} , then the cardinality of the set of generated eigenfaces and eigenvalues would be $\mathcal{N} \times \mathcal{M}$, to differentiate the $\mathcal{N} \times \mathcal{M}$ total faces in the training set. But usually this is an ordered set where the first few eigenfaces and eigenvalues are the most critical differentiators, and the last few eigenfaces and eigenvalues usually contain irrelevant information that do not actually help to differentiate the data. So it is common practice to discard some of the last eigenfaces and just keep the first 50 percent or so eigenfaces.



Figure 5: Calculated eigenfaces for training set

The Eigenfaces algorithm (as well as Fisherfaces does) first calculate the average face that is the mathematical average of all the training images, for subtracting later the average image from each facial image and having better face recognition results.

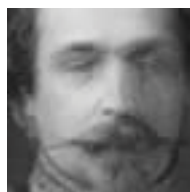


Figure 6: Average image

3.3 Difficulties to be considered

According to Baggio (Baggio et al., 2012), the current face detection techniques are quite reliable in real-world conditions, whereas current face recognition techniques are much less reliable when used in real-world conditions. For example, it is quite common to find research papers showing face recognition accuracy rates above 95 percent, but when testing those same algorithms independently, measured accuracy rate can be lower than 50 percent. This behaviour can be explained by the fact that face recognition techniques tend to be very sensitive to exact conditions in the images, such as the type of lighting, direction of lighting and shadows, face wear, pose, expression of the face,

etc. So the data set used during training is essential for the success. This point must be seriously considered when training the prototype, and it will be necessary to take pictures of the paintings from different points and under different conditions.

Face pre-processing aims to reduce these problems, such as by making sure the face always appears to have similar brightness and contrast. A good face pre-processing stage will help improving the reliability of the whole face recognition system.



Figure 7: False positives in face detection

If Figure 7 is examined, the face is divided into several pieces, or has been detected twice or even three times. There are also false face detections in regions of the picture where there is no face at all. These eventualities are usually fixed applying some kind of statistical post-processing. This point should also be taken into account when developing the final version of the prototype.

4 Prototype design

4.1 Architectural overview

From an architectural point of view, the prototype will be structured in several modules. The main module contains recognition algorithms. It will be trained for recognizing the faces, after they have been detected and cropped from the original image.

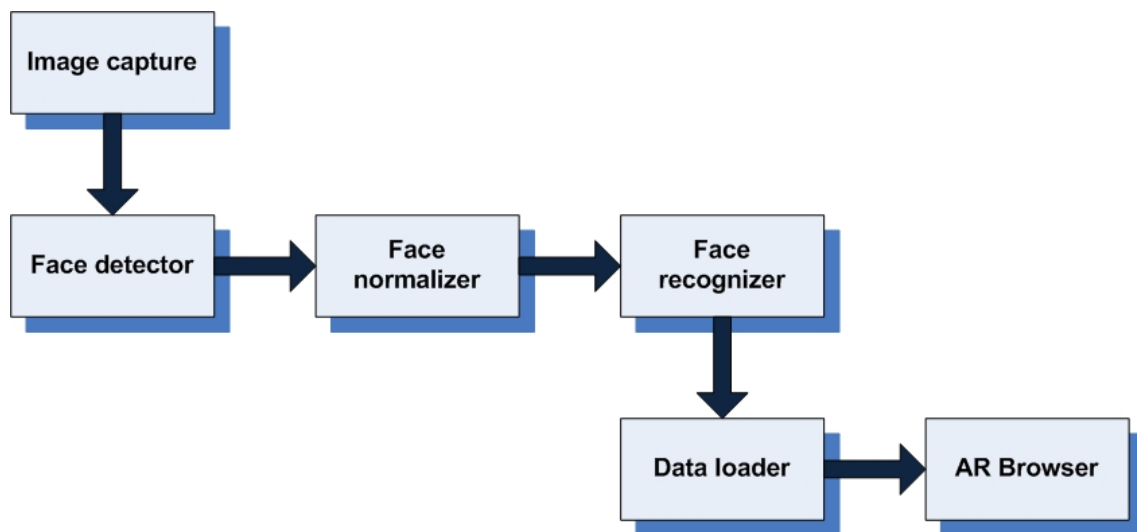


Figure 8: Overview of components as stages of a process

4.2 Components definition

Image capture module is hardware-dependent, and should be delegated on smartphone functions. All the principal smartphone platforms give programmers the ability of accessing terminal's camera and even control the capture process. For example, in Android platforms it is quite easy to get programmatic access to the camera functions after having declared program needs in a special manifest file (*AndroidManifest.xml*).

Face detector should get the image, detect the faces and return an in-memory structure containing a representation of these images. These images should be an image cropped from the original one, and should have been pre-processed for eliminating duplicates or false detections.

Face normaliser should transform the images into gray scale and perform the transformations that can be needed (scaling, rotation, etc.).

Face recognizer is the central piece of the system. It will receive normalized images and will perform a recognition using the knowledge acquired during training phase. The

design pattern applied to this component is Strategy pattern. It enables a component to select its behaviour at runtime. The strategy pattern defines a family of algorithms, encapsulates each algorithm, and makes the algorithms interchangeable within that family. It can do it by defining a common interface for calling every algorithm.

This pattern is quite recommendable when dealing with families of related algorithms. In this case, the different strategies can be defined as a hierarchy of classes offering the ability to extend and customize the existing algorithms from an application.

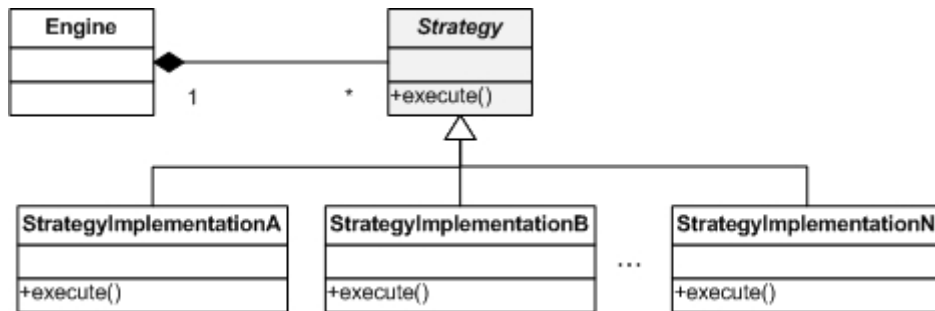


Figure 9: UML class diagram of Strategy pattern

Figure 10 shows the actual application of the pattern to face recognition context. The interface *FaceRecognizer* defines two operations: *learn* and *recognizeFileList*. The *learn* operation has one parameter that receives the name of a file containing a list of images files for training. These image files are encoded in PGM format (a portable greyscale bitmap), a format supported by OpenCV framework. The *recognizeFileList* operation performs the corresponding recognition algorithm. It receives a parameter that is the name of a file containing a list of images to be recognized.

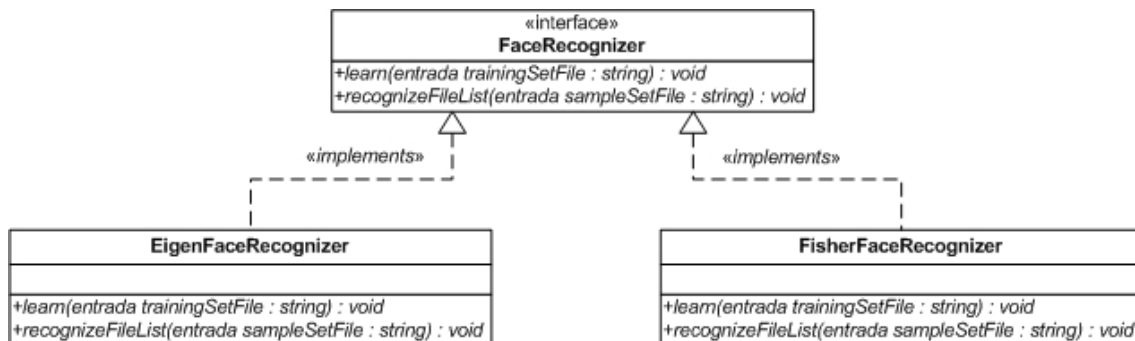


Figure 10: UML class diagram of FaceRecognizer component

Initial version of the prototype only have the Eigenfaces class, but as stated in planning, it will have a version of Fisherfaces algorithm, and also Local Binary Patterns (LBP), as they are built-in algorithms in OpenCV. Later, other implementations can be developed if convenient for research purposes, for example Gabor Wavelets or Discrete Cosinus Transform.

Data loader will receive the name of the recognized figure and will perform a query for related information. The source could be the Wikipedia or any other knowledge base available.

Finally, AR Browser will put all the information gathered in a composed view, showing the user a window associated to the portrayed figure.

Beside the main logic of the application, there will be components devoted to what Aspect-Oriented Programming calls cross-cutting concerns, that is, those elements related to transversal aspects like logging or performance measuring. Because of the application of the Strategy pattern allows the prototype to seamlessly change the algorithm for performing the recognition, the role of these cross-cutting concerns is to keep a registry of behavior and performance of each technique. The information gathered by these components will be employed in the next stage of research process, the statistical survey.

5 Annex: Source code

The following code is part of the first draft of the described prototype. It only contains the implementation of the Eigenfaces algorithm, but it must be considered as the seed for developing the final version.

File EigenfaceRecognizer.java

```
package edu.uoc.multimedia.face.recognition.test;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;
import org.bytedeco.javacpp.FloatPointer;
import org.bytedeco.javacpp.Pointer;
import edu.uoc.multimedia.face.recognition.exception.FaceRecognizerException;
import static org.bytedeco.javacpp.opencv_core.*;
import static org.bytedeco.javacpp.opencv_highgui.*;
import static org.bytedeco.javacpp.opencv_legacy.*;

/**
 * Implementation for the Eigenfaces algorithm
 * Based on JavaCV exaples
 */
public class EigenfaceRecognizer {

    /** logger */
    private static final Logger LOGGER = Logger.getLogger(FaceRecognizer.class.getName());
    /** number of training faces */
    private int numberTrainingFaces = 0;
    /** training face image array */
    IpLlImage[] trainingFacelImageArray;
    /** test face image array */
    IpLlImage[] testFacelImageArray;
    /** person number array */
    CvMat personNumTruthMat;
    /** number of persons */
    int nPersons;
    /** person names */
    final List<String> personNames = new ArrayList<String>();
    /** number of eigenvalues */
    int numberEigenvectors = 0;
    /** eigenvectors */
    IpLlImage[] eigenvectorsArray;
    /** eigenvalues */
    CvMat eigenvaluesMat;
    /** average image */
    IpLlImage averageTrainingImage;
    /** projected training faces */
    CvMat projectedTrainingFacesMat;

    /** Constructs a new FaceRecognition instance. */
    public EigenfaceRecognizer() {
```

```
}

/** Trains from the data in the given training text index file, and store the trained data into the file
'facedata.xml'.
*
* @param trainingFileName the given training text index file
*/
public void learn(final String trainingFileName) {
    int i;

    // load training data
    LOGGER.info("=====");
    LOGGER.info("Loading the training images in " + trainingFileName);

    trainingFacelImageArray = loadFacelmgArray(trainingFileName);
    numberTrainingFaces = trainingFacelImageArray.length;

    LOGGER.info("Got " + numberTrainingFaces + " training images");

    if (numberTrainingFaces < 3) {
        LOGGER.severe("Need 3 or more training faces\n"
            + "Input file contains only " + numberTrainingFaces);
        return;
    }

    // do Principal Component Analysis on the training faces
    doPCA();

    LOGGER.info("projecting the training images onto the PCA subspace");

    // project the training images onto the PCA subspace
    projectedTrainingFacesMat = cvCreateMat(
        numberTrainingFaces, // rows
        numberEigenvectors, // cols
        CV_32FC1); // type, 32-bit float, 1 channel

    // initialize the training face matrix - for ease of debugging
    for (int i1 = 0; i1 < numberTrainingFaces; i1++) {
        for (int j1 = 0; j1 < numberEigenvectors; j1++) {
            projectedTrainingFacesMat.put(i1, j1, 0.0);
        }
    }

    LOGGER.info("created projectedTrainFaceMat with " + numberTrainingFaces + "
(nTrainFaces) rows and " + numberEigenvectors + " (nEigens) columns");
    if (numberTrainingFaces < 4) {
        LOGGER.info("projectedTrainFaceMat contents:\n" +
oneChannelCvMatToString(projectedTrainingFacesMat));
    }

    final FloatPointer floatPointer = new FloatPointer(numberEigenvectors);
    for (i = 0; i < numberTrainingFaces; i++) {
        cvEigenDecomposite(
            trainingFacelImageArray[i], // obj
            numberEigenvectors, // nEigObjs
            eigenvectorsArray, // eigInput (Pointer)
```

```
        0, // ioFlags
        null, // userData (Pointer)
        averageTrainingImage, // avg
        floatPointer); // coeffs (FloatPointer)

        if (numberTrainingFaces < 5) {
            LOGGER.info("floatPointer: " + floatPointerToString(floatPointer));
        }
        for (int j1 = 0; j1 < numberEigenvectors; j1++) {
            projectedTrainingFacesMat.put(i, j1, floatPointer.get(j1));
        }
    }
    if (numberTrainingFaces < 5) {
        LOGGER.info("projectedTrainFaceMat after cvEigenDecomposite:\n" +
projectedTrainingFacesMat);
    }

    // store the recognition data as an xml file
    storeTrainingData();

    // Save all the eigenvectors as images, so that they can be checked.
    storeEigenfacelImages();
}

/** Recognizes the face in each of the test images given, and compares the results with the truth.
 *
 * @param szFileTest the index file of test images
 */
public void recognizeFileList(final String szFileTest) {
    LOGGER.info("=====");
    LOGGER.info("recognizing faces indexed from " + szFileTest);
    int i = 0;
    int nTestFaces = 0; // the number of test images
    CvMat trainPersonNumMat; // the person numbers during training
    float[] projectedTestFace;
    String answer;
    int nCorrect = 0;
    int nWrong = 0;
    double timeFaceRecognizeStart;
    double tallyFaceRecognizeTime;
    float confidence = 0.0f;

    // load test images and ground truth for person number
    testFacelImageArray = loadFacelmgArray(szFileTest);
    nTestFaces = testFacelImageArray.length;

    LOGGER.info(nTestFaces + " test faces loaded");

    // load the saved training data
    trainPersonNumMat = loadTrainingData();
    if (trainPersonNumMat == null) {
        return;
    }

    // project the test images onto the PCA subspace
    projectedTestFace = new float[numberEigenvectors];
```



```
timeFaceRecognizeStart = (double) cvGetTickCount();    // Record the timing.

for (i = 0; i < nTestFaces; i++) {
    int iNearest;
    int nearest;
    int truth;

    // project the test image onto the PCA subspace
    cvEigenDecomposite(
        testFacelmgArray[i], // obj
        numberEigenvectors, // nEigObjs
        eigenvectorsArray, // eigInput (Pointer)
        0, // ioFlags
        null, // userData
        averageTrainingImage, // avg
        projectedTestFace); // coeffs

    //LOGGER.info("projectedTestFace\n" + floatArrayToString(projectedTestFace));

    final FloatPointer pConfidence = new FloatPointer(confidence);
    iNearest = findNearestNeighbor(projectedTestFace, new
FloatPointer(pConfidence));
    confidence = pConfidence.get();
    truth = personNumTruthMat.data_i().get(i);
    nearest = trainPersonNumMat.data_i().get(iNearest);

    if (nearest == truth) {
        answer = "Correct";
        nCorrect++;
    } else {
        answer = "WRONG!";
        nWrong++;
    }
    LOGGER.info("nearest = " + nearest + ", Truth = " + truth + " (" + answer + ").
Confidence = " + confidence);
}
tallyFaceRecognizeTime = (double) cvGetTickCount() - timeFaceRecognizeStart;
if (nCorrect + nWrong > 0) {
    LOGGER.info("TOTAL ACCURACY: " + (nCorrect * 100 / (nCorrect + nWrong)) +
"% out of " + (nCorrect + nWrong) + " tests.");
    LOGGER.info("TOTAL TIME: " + (tallyFaceRecognizeTime / (cvGetTickCount()
* 1000.0 * (nCorrect + nWrong))) + " ms average.");
}
}

/** Reads the names & image filenames of people from a text file, and loads all those images
listed.
*
* @param filename the training file name
* @return the face image array
*/
private IplImage[] loadFacelmgArray(final String filename) {
    IplImage[] facelmgArr;
    BufferedReader imgListFile;
    String imgFilename;
    int iFace = 0;
```

```
int nFaces = 0;
int i;
try {
    // open the input file
    imgListFile = new BufferedReader(new FileReader(filename));

    // count the number of faces
    while (true) {
        final String line = imgListFile.readLine();
        if (line == null || line.isEmpty()) {
            break;
        }
        nFaces++;
    }
    //TODO: Check this
    imgListFile.close();

    LOGGER.info("nFaces: " + nFaces);
    imgListFile = new BufferedReader(new FileReader(filename));

    // allocate the face-image array and person number matrix
    faceImgArr = new IpLImage[nFaces];
    personNumTruthMat = cvCreateMat(
        1, // rows
        nFaces, // cols
        CV_32SC1); // type, 32-bit unsigned, one channel

    // initialize the person number matrix – for ease of debugging
    for (int j1 = 0; j1 < nFaces; j1++) {
        personNumTruthMat.put(0, j1, 0);
    }

    personNames.clear();    // Make sure it starts as empty.
    nPersons = 0;

    // store the face images in an array
    for (iFace = 0; iFace < nFaces; iFace++) {
        String personName;
        String sPersonName;
        int personNumber;

        // read person number (beginning with 1), their name and the image
filename.

        final String line = imgListFile.readLine();
        if (line.isEmpty()) {
            break;
        }
        final String[] tokens = line.split(" ");
        personNumber = Integer.parseInt(tokens[0]);
        personName = tokens[1];
        imgFilename = tokens[2];
        sPersonName = personName;
        LOGGER.info("Got " + iFace + " " + personNumber + " " + personName
+ " " + imgFilename);

        // Check if a new person is being loaded.
```

```
        if (personNumber > nPersons) {
            // Allocate memory for the extra person (or possibly multiple),
using this new person's name.
            personNames.add(sPersonName);
            nPersons = personNumber;
            LOGGER.info("Got new person " + sPersonName + " ->
nPersons = " + nPersons + " [" + personNames.size() + "]");
        }

        // Keep the data
        personNumTruthMat.put(
            0, // i
            iFace, // j
            personNumber); // v

        // load the face image
        facelmgArr[iFace] = cvLoadImage(
            imgFilename, // filename
            CV_LOAD_IMAGE_GRAYSCALE); // isColor

        if (facelmgArr[iFace] == null) {
            if (imgListFile != null) {
                imgListFile.close();
            }
            throw new FaceRecognizerException("Can't load image from "
+ imgFilename);
        }

        imgListFile.close();

    } catch (IOException ex) {
        throw new FaceRecognizerException(ex);
    }

    LOGGER.info("Data loaded from '" + filename + "': (" + nFaces + " images of " + nPersons
+ " people).");

    final StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("People: ");
    if (nPersons > 0) {
        stringBuilder.append("<").append(personNames.get(0)).append(">");
    }
    for (i = 1; i < nPersons && i < personNames.size(); i++) {
        stringBuilder.append(", <").append(personNames.get(i)).append(">");
    }
    LOGGER.info(stringBuilder.toString());

    return facelmgArr;
}

/** Does the Principal Component Analysis, finding the average image and the eigenfaces that
represent any image in the given dataset. */
private void doPCA() {
    int i;
    CvTermCriteria calcLimit;
    CvSize facelmgSize = new CvSize();
```

```
// set the number of eigenvalues to use
numberEigenvectors = numberTrainingFaces - 1;

LOGGER.info("allocating images for principal component analysis, using " +
numberEigenvectors + (numberEigenvectors == 1 ? " eigenvalue" : " eigenvalues"));

// allocate the eigenvector images
facelmgSize.width(trainingFacelImageArray[0].width());
facelmgSize.height(trainingFacelImageArray[0].height());
eigenvectorsArray = new IplImage[numberEigenvectors];
for (i = 0; i < numberEigenvectors; i++) {
    eigenvectorsArray[i] = cvCreateImage(
        facelmgSize, // size
        IPL_DEPTH_32F, // depth
        1); // channels
}

// allocate the eigenvalue array
eigenvaluesMat = cvCreateMat(
    1, // rows
    numberEigenvectors, // cols
    CV_32FC1); // type, 32-bit float, 1 channel

// allocate the averaged image
averageTrainingImage = cvCreateImage(
    facelmgSize, // size
    IPL_DEPTH_32F, // depth
    1); // channels

// set the PCA termination criterion
calcLimit = cvTermCriteria(
    CV_TERMCRIT_ITER, // type
    numberEigenvectors, // max_iter
    1); // epsilon

LOGGER.info("computing average image, eigenvalues and eigenvectors");
// compute average image, eigenvalues, and eigenvectors
cvCalcEigenObjects(
    numberTrainingFaces, // nObjects
    trainingFacelImageArray, // input
    eigenvectorsArray, // output
    CV_EIGOBJ_NO_CALLBACK, // ioFlags
    0, // ioBufSize
    null, // userData
    calcLimit,
    averageTrainingImage, // avg
    eigenvaluesMat.data_fl()); // eigVals

LOGGER.info("normalizing the eigenvectors");
cvNormalize(
    eigenvaluesMat, // src (CvArr)
    eigenvaluesMat, // dst (CvArr)
    1, // a
    0, // b
    CV_L1, // norm_type
```

```
        null); // mask
    }

    /** Stores the training data to the file 'facedata.xml'. */
    private void storeTrainingData() {
        CvFileStorage fileStorage;
        int i;

        LOGGER.info("writing facedata.xml");

        // create a file-storage interface
        fileStorage = cvOpenFileStorage(
            "facedata.xml", // filename
            null, // memstorage
            CV_STORAGE_WRITE, // flags
            null); // encoding

        // Store the person names. Added by Shervin.
        cvWriteInt(
            fileStorage, // fs
            "nPersons", // name
            nPersons); // value

        for (i = 0; i < nPersons; i++) {
            String varname = "personName_" + (i + 1);
            cvWriteString(
                fileStorage, // fs
                varname, // name
                personNames.get(i), // string
                0); // quote
        }

        // store all the data
        cvWriteInt(
            fileStorage, // fs
            "nEigens", // name
            numberEigenvectors); // value

        cvWriteInt(
            fileStorage, // fs
            "nTrainFaces", // name
            numberTrainingFaces); // value

        cvWrite(
            fileStorage, // fs
            "trainPersonNumMat", // name
            personNumTruthMat); // value

        cvWrite(
            fileStorage, // fs
            "eigenValMat", // name
            eigenvaluesMat); // value

        cvWrite(
            fileStorage, // fs
            "projectedTrainFaceMat", // name
```

```
        projectedTrainingFacesMat);

    cvWrite(fileStorage, // fs
            "avgTrainImg", // name
            averageTrainingImage); // value

    for (i = 0; i < numberEigenvectors; i++) {
        String varname = "eigenVect_" + i;
        cvWrite(
            fileStorage, // fs
            varname, // name
            eigenvectorsArray[i]); // value
    }

    // release the file-storage interface
    cvReleaseFileStorage(fileStorage);
}

/** Opens the training data from the file 'facedata.xml'.
 *
 * @param pTrainPersonNumMat
 * @return the person numbers during training, or null if not successful
 */
private CvMat loadTrainingData() {
    LOGGER.info("loading training data");
    CvMat pTrainPersonNumMat = null; // the person numbers during training
    CvFileStorage fileStorage;
    int i;

    // create a file-storage interface
    fileStorage = cvOpenFileStorage(
        "facedata.xml", // filename
        null, // memstorage
        CV_STORAGE_READ, // flags
        null); // encoding
    if (fileStorage == null) {
        LOGGER.severe("Can't open training database file 'facedata.xml'.");
        return null;
    }

    // Load the person names.
    personNames.clear(); // Make sure it starts as empty.
    nPersons = cvReadIntByName(
        fileStorage, // fs
        null, // map
        "nPersons", // name
        0); // default_value
    if (nPersons == 0) {
        LOGGER.severe("No people found in the training database 'facedata.xml'.");
        return null;
    } else {
        LOGGER.info(nPersons + " persons read from the training database");
    }

    // Load each person's name.
    for (i = 0; i < nPersons; i++) {
```

```
        String sPersonName;
        String varname = "personName_" + (i + 1);
        sPersonName = cvReadStringByName(
            fileStorage, // fs
            null, // map
            varname,
            "");
        personNames.add(sPersonName);
    }
    LOGGER.info("person names: " + personNames);

    // Load the data
    numberEigenvectors = cvReadIntByName(
        fileStorage, // fs
        null, // map
        "nEigens",
        0); // default_value
    numberTrainingFaces = cvReadIntByName(
        fileStorage,
        null, // map
        "nTrainFaces",
        0); // default_value
    Pointer pointer = cvReadByName(
        fileStorage, // fs
        null, // map
        "trainPersonNumMat"); // name
    pTrainPersonNumMat = new CvMat(pointer);

    pointer = cvReadByName(
        fileStorage, // fs
        null, // map
        "eigenValMat"); // name
    eigenvaluesMat = new CvMat(pointer);

    pointer = cvReadByName(
        fileStorage, // fs
        null, // map
        "projectedTrainFaceMat"); // name
    projectedTrainingFacesMat = new CvMat(pointer);

    pointer = cvReadByName(
        fileStorage,
        null, // map
        "avgTrainImg");
    averageTrainingImage = new IpLlImage(pointer);

    eigenvectorsArray = new IpLlImage[numberTrainingFaces];
    for (i = 0; i <= numberEigenvectors; i++) {
        String varname = "eigenVect_" + i;
        pointer = cvReadByName(
            fileStorage,
            null, // map
            varname);
        eigenvectorsArray[i] = new IpLlImage(pointer);
    }
```



```
// release the file-storage interface
cvReleaseFileStorage(fileStorage);

LOGGER.info("Training data loaded (" + numberTrainingFaces + " training images of " +
nPersons + " people)");
final StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("People: ");
if (nPersons > 0) {
    stringBuilder.append("<").append(personNames.get(0)).append(">");
}
for (i = 1; i < nPersons; i++) {
    stringBuilder.append(", <").append(personNames.get(i)).append(">");
}
LOGGER.info(stringBuilder.toString());

return pTrainPersonNumMat;
}

/** Saves all the eigenvectors as images, so that they can be checked. */
private void storeEigenfaceImages() {
    // Store the average image to a file
    LOGGER.info("Saving the image of the average face as 'out_averagelImage.bmp'");
    cvSaveImage("out_averagelImage.bmp", averageTrainingImage);

    // Create a large image made of many eigenface images.
    // Must also convert each eigenface image to a normal 8-bit UCHAR image instead of a
32-bit float image.
    LOGGER.info("Saving the " + numberEigenvectors + " eigenvector images as
'out_eigenfaces.bmp'");

    if (numberEigenvectors > 0) {
        // Put all the eigenfaces next to each other.
        int COLUMNS = 8; // Put upto 8 images on a row.
        int nCols = Math.min(numberEigenvectors, COLUMNS);
        int nRows = 1 + (numberEigenvectors / COLUMNS); // Put the rest on new
rows.

        int w = eigenvectorsArray[0].width();
        int h = eigenvectorsArray[0].height();
        CvSize size = cvSize(nCols * w, nRows * h);
        final IpLlImage bigImg = cvCreateImage(
            size,
            IPL_DEPTH_8U, // depth, 8-bit Greyscale UCHAR image
            1); // channels
        for (int i = 0; i < numberEigenvectors; i++) {
            // Get the eigenface image.
            IpLlImage bytelmg =
convertFloatImageToUcharImage(eigenvectorsArray[i]);
            // Paste it into the correct position.
            int x = w * (i % COLUMNS);
            int y = h * (i / COLUMNS);
            CvRect ROI = cvRect(x, y, w, h);
            cvSetImageROI(
                bigImg, // image
                ROI); // rect
            cvCopy(
                bytelmg, // src
```

```
        bigImg, // dst
        null); // mask
        cvResetImageROI(bigImg);
        cvReleaseImage(bytImg);
    }
    cvSaveImage(
        "out_eigenfaces.bmp", // filename
        bigImg); // image
    cvReleaseImage(bigImg);
}

/** Converts the given float image to an unsigned character image.
 *
 * @param srcImg the given float image
 * @return the unsigned character image
 */
private IplImage convertFloatImageToUcharImage(IplImage srcImg) {
    IplImage dstImg;
    if ((srcImg != null) && (srcImg.width() > 0 && srcImg.height() > 0)) {
        // Spread the 32bit floating point pixels to fit within 8bit pixel range.
        double[] minVal = new double[1];
        double[] maxVal = new double[1];
        cvMinMaxLoc(srcImg, minVal, maxVal);
        // Deal with NaN and extreme values, since the DFT seems to give some NaN
        results.
        if (minVal[0] < -1e30) {
            minVal[0] = -1e30;
        }
        if (maxVal[0] > 1e30) {
            maxVal[0] = 1e30;
        }
        if (maxVal[0] - minVal[0] == 0.0f) {
            errors.
            maxVal[0] = minVal[0] + 0.001; // remove potential divide by zero

        }
        // Convert the format
        dstImg = cvCreateImage(cvSize(srcImg.width(), srcImg.height()), 8, 1);
        cvConvertScale(srcImg, dstImg, 255.0 / (maxVal[0] - minVal[0]), -minVal[0] *
        255.0 / (maxVal[0] - minVal[0]));
        return dstImg;
    }
    return null;
}

/** Find the most likely person based on a detection. Returns the index, and stores the confidence
value into pConfidence.
 *
 * @param projectedTestFace the projected test face
 * @param pConfidencePointer a pointer containing the confidence value
 * @param iTestFace the test face index
 * @return the index
 */
private int findNearestNeighbor(float projectedTestFace[], FloatPointer pConfidencePointer) {
    double leastDistSq = Double.MAX_VALUE;
    int i = 0;
    int iTrain = 0;
```

```
int iNearest = 0;

LOGGER.info(".....");
LOGGER.info("find nearest neighbor from " + numberTrainingFaces + " training faces");
for (iTrain = 0; iTrain < numberTrainingFaces; iTrain++) {
    //LOGGER.info("considering training face " + (iTrain + 1));
    double distSq = 0;

    for (i = 0; i < numberEigenvectors; i++) {
        //LOGGER.debug(" projected test face distance from eigenface " + (i +
1) + " is " + projectedTestFace[i]);

        float projectedTrainFaceDistance = (float)
projectedTrainingFacesMat.get(iTrain, i);
        float d_i = projectedTestFace[i] - projectedTrainFaceDistance;
        distSq += d_i * d_i; // / eigenValMat.data_fl().get(i); // Mahalanobis
distance (might give better results than Eucalidean distance)
        // if (iTrain < 5) {
        //     LOGGER.info(" ** projected training face " + (iTrain + 1) + "
distance from eigenface " + (i + 1) + " is " + projectedTrainFaceDistance);
        //     LOGGER.info(" distance between them " + d_i);
        //     LOGGER.info(" distance squared " + distSq);
        // }
    }

    if (distSq < leastDistSq) {
        leastDistSq = distSq;
        iNearest = iTrain;
        LOGGER.info(" training face " + (iTrain + 1) + " is the new best match,
least squared distance: " + leastDistSq);
    }
}

// Return the confidence level based on the Euclidean distance,
// so that similar images should give a confidence between 0.5 to 1.0,
// and very different images should give a confidence between 0.0 to 0.5.
float pConfidence = (float) (1.0f - Math.sqrt(leastDistSq / (float) (numberTrainingFaces *
numberEigenvectors)) / 255.0f);
pConfidencePointer.put(pConfidence);

LOGGER.info("training face " + (iNearest + 1) + " is the final best match, confidence " +
pConfidence);
return iNearest;
}

/** Returns a string representation of the given float array.
 *
 * @param floatArray the given float array
 * @return a string representation of the given float array
 */
private String floatArrayToString(final float[] floatArray) {
    final StringBuilder stringBuilder = new StringBuilder();
    boolean isFirst = true;
    stringBuilder.append("[");
    for (int i = 0; i < floatArray.length; i++) {
        if (isFirst) {
```

```
        isFirst = false;
    } else {
        stringBuilder.append(", ");
    }
    stringBuilder.append(floatArray[i]);
}
stringBuilder.append("]");

return stringBuilder.toString();
}*/

/** Returns a string representation of the given float pointer.
 *
 * @param floatPointer the given float pointer
 * @return a string representation of the given float pointer
 */
private String floatPointerToString(final FloatPointer floatPointer) {
    final StringBuilder stringBuilder = new StringBuilder();
    boolean isFirst = true;
    stringBuilder.append("[");
    for (int i = 0; i < floatPointer.capacity(); i++) {
        if (isFirst) {
            isFirst = false;
        } else {
            stringBuilder.append(", ");
        }
        stringBuilder.append(floatPointer.get(i));
    }
    stringBuilder.append("]");

    return stringBuilder.toString();
}

/** Returns a string representation of the given one-channel CvMat object.
 *
 * @param cvMat the given CvMat object
 * @return a string representation of the given CvMat object
 */
public String oneChannelCvMatToString(final CvMat cvMat) {
    //Preconditions
    if (cvMat.channels() != 1) {
        throw new FaceRecognizerException("illegal argument - CvMat must have one
channel");
    }

    final int type = cvMat.type();
    StringBuilder s = new StringBuilder("[ ");
    for (int i = 0; i < cvMat.rows(); i++) {
        for (int j = 0; j < cvMat.cols(); j++) {
            if (type == CV_32FC1 || type == CV_32SC1) {
                s.append(cvMat.get(i, j));
            } else {
                throw new FaceRecognizerException("illegal argument - CvMat
must have one channel and type of float or signed integer");
            }
            if (j < cvMat.cols() - 1) {
```

```
                s.append(", ");
            }
        }
        if (i < cvMat.rows() - 1) {
            s.append("\n ");
        }
    }
    s.append("]");
    return s.toString();
}
}
```

File FaceRecognizerApp.java

```
package edu.uoc.multimedia.face.recognition.test;

public class FaceRecognizerApp {
    public static void main(final String[] args) {

        final EigenfaceRecognizer faceRecognizer = new EigenfaceRecognizer();
        faceRecognizer.learn("trainingsample.txt");
        faceRecognizer.recognizeFileList("faces.txt");
    }
}
```

6 References

Ahonen, T., et al. (2006). *Face description with local binary patterns: Application to face recognition*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 28(12), 2037-2041.

Baggio, D. L. et al. (2012). *Mastering OpenCV with practical computer vision projects*. Packt Publishing Ltd.

Gamma, E. et al. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

Lienhart, R. and Maydt, J. (2002). *An extended set of haar-like features for rapid object detection*. Image Processing. 2002. Proceedings. 2002 International Conference on (Vol. 1, pp. I-900). IEEE.

Lienhart, R., et al. (2003). *Empirical analysis of detection cascades of boosted classifiers for rapid object detection*. Pattern Recognition (pp. 297-304). Springer Berlin Heidelberg.

Madden, L. (2011). *Professional augmented reality browsers for smartphones: programming for junaio, layar and wiktude*. John Wiley & Sons.

Viola, P. and Jones, M. (2001). *Rapid object detection using a boosted cascade of simple features*. Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on (Vol. 1, pp. I-511). IEEE.

Virkus, R. et al. (2012). *Don't Panic, Mobile Developer's Guide to the Galaxy, 11th ed.* Enough Software GmbH+ Co. KG.

IMAGES

Portrait of Napoleon III by Franz Xaver Winterhalter

Source: Wikipedia (http://de.wikipedia.org/wiki/Zweites_Kaiserreich)

License: CC with some restrictions (Attribution and Share-alike)