IntensityPatches and RegionPatches for image recognition

IntensityPatches and RegionPatches for Image Recognition

Tiago B. A. de Carvalho[a,*], Maria A. A. Sibaldo[a], Ing Ren Tsang[b], George D. C. Cavalcanti[b], Jan Sijbers[c], Ing Jyh Tsang[d]

[a] *Universidade Federal Rural de Pernambuco, Unidade Acadêmica de Garanhuns, PE, Brasil*
[b] *Universidade Federal de Pernambuco, Centro de Informática, Recife, PE, Brasil*
[c] *University of Antwerp, IMEC - Vision Lab, Antwerp, Belgium*
[d] *Nokia, Antwerp, Belgium*

Abstract

In this paper, we propose a framework for defining feature extraction techniques, called Pixel Clustering. It is an extension of feature extraction with Wavelets. We propose two linear feature extraction techniques using Pixel Clustering: IntensityPatches and RegionPatches. We assess the methods in color and grayscale image datasets: two face datasets and two object datasets. The proposed methods present a short computation time for feature extraction and high accuracy compared with linear feature extraction methods and other state-of-the-art feature extraction techniques.

Keywords: Dimensionality reduction, Feature extraction, Feature learning, Linear methods, Principal Component Analysis, Unsupervised machine learning, Wavelets

1. Introduction

Image shrinkage seems to have only a minor influence on recognition tasks. This can be observed in the experiments performed using Waveletfaces [1]. Here, we investigate and present a reasonable explanation for this effect, which is

☆This work was partially supported by Brazilian agencies CNPq, CAPES, FACEPE and by the Fund for Scientific Research Flanders (FWO).
*Corresponding author
Email address: tbac@cin.ufpe.br, tiago.buarque@ufrpe.br (Tiago B. A. de Carvalho)

used to propose a dimensionality reduction method based on this idea. In Waveletfaces a 2D Discrete Wavelet Transform (2D-DWT) is applied to an image. The 2D-DWT produces 4 images, each one with 1/2 of the original height and 1/2 of the original width: a low resolution image and three detail images. Waveletfaces uses only the low resolution image (approximation image).

Various Wavelet functions can be used, Chien & Wu suggest applying the Haar Wavelet[1]. The Haar approximation image is trivially computed, since the value of each pixel in the approximation image is simply the summation of the four corresponding neighborhood pixels in the original image. If each pixel intensity value in the approximation image is multiplied by the constant value 1/4, the approximation image is a resized (or shrank) version of the original image. It is possible to compute Waveletfaces Level 3, Level 4, and so on, by applying Waveletfaces to the previous level. As described in the experiment section, it is possible to reduce the number of features by $1/1,000$ with small impact on the classification accuracy.

Assuming without loss of generality that each Level of Waveletfaces is equivalent to image shrink, the following question arises: why does image shrinkage has only little influence on the recognition task? This is a very intriguing question because information is lost when the image is rescaled to a smaller size. On the other hand, it is expected that examples can be better classified in a lower dimensional space due the curse of dimensionality [2]. Why is this kind of dimensionality reduction suitable for image recognition? The first probable explanation is that it retains the rough information while eliminating the details.

The second explanation, and the basis for the proposed method, is that each extracted feature averages similar intensity values. Most of the neighborhoods in an image are non-border pixels, see Figure 1. Averaging those values does not cause information loss. Also, information loss is not so large in some border pixels as shown in the middle image in Figure 1, because most of the pixels in those cases have similar intensity values. A relevant information loss occurs when each half of the pixels belongs to different intensity range as shown on the left image of Figure 1.

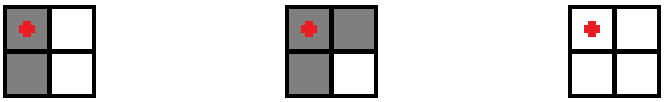

Figure 1: Border and non-border pixels highlighted within their neighborhoods.

Assuming that most of the neighborhoods are non-border pixels. The process of averaging neighboring pixels should reduce the number of correlated features retaining a more diverse set of features. This can be easily visualized in a frontal face image, in which there are various pixels with similar intensity value: skin pixels, hair pixels etc. The proposed method, Pixel Clustering, automatically finds regions of similar intensity values and extracts features by averaging the intensity values within each region. This process is an unsupervised linear feature extraction method which is discussed in the next sections.

The paper is organized as follows: the proposed method is described in Section 3. Experiment and discussion are presented in Section 4. Conclusions are presented in Section 6.

2. Unsupervised linear feature extraction methods

Feature extraction is the task that aims to find a characteristic representation for the original data. This task is also known as representation learning (or feature learning) and plays a major role in deep networks and other machine learning applications [3]. If the set of extracted features has less elements than the original set of features, feature extraction can also be called dimensionality reduction.

Attention has been given to linear feature extraction methods because they are computationally efficient and have solid mathematical properties [4]. The composition of linear operations yields another linear operation thus those methods have restricted expression power. In spite of the limitation of the linear methods, they are a base for more sophisticated non-linear feature extraction such as Autoencoders [3] and PCANet [5].

Unsupervised feature learning deals with extracting features without considering class labels. Nevertheless, it is applied to labeled data in supervised tasks, such Principal Component Analysis (PCA) for classification [6], Autoencoders for Deep Neural Networks (DNN) [3], and Convolutional Neural Networks (CNN) for image recognition [7]. Unsupervised linear feature extraction is also important in the classification of high dimensional data, especially for hyperspectral imagery [8].

PCA is a widely used unsupervised linear method for dimensionality reduction and can be interpreted as a linear Autoencoder [3]. PCA is also known as the Karhunen-Loève Transform, and one of its properties is a minimum reconstruction error. It means that using the eigenvectors of maximal eigenvalues guarantees that no other linear projection has a smaller squared error for the reconstructed data using the same number of extracted features [9].

Recently, PCA has been applied in combination with other techniques such as quantile regression for the regression task [10] and decision tree algorithm for the classification task [11]. Also, extended versions have been proposed, like Convex Sparse PCA which aims to be easier to interpret and less sensitive to noise [6]. Eigenfaces, the dual version of PCA for high dimensional data, is still a benchmark technique for face recognition [12]. Moreover, extensions of Eigenfaces, based on the fractional covariance matrix, [13, 14] has also been proposed. The Wavelet transform is a linear filtering [15] thus Waveletfaces is also a linear unsupervised linear feature extraction method. Here, we compare the proposed technique with PCA, PCANet, Waveletfaces and Autoencoder.

3. Pixel Clustering

Pixel Clustering is a framework for defining feature extraction techniques. Our inspiration comes from the assumption that each pixel has many neighbors of similar intensity (recall Figure 1), it is true for several face and object images datasets. The objective of Pixel Clustering is to cluster those pixels into regions and extract features from those regions. Such sets of pixels are not segmentation

of a single image; the image partition is unique and used for every image in the training and test sets.

The main contributions of the proposed framework are twofold: the cluster-of-pixel approach; and the pixel-vector, extracted before performing the clustering algorithm. Those procedures result in a proper segmentation that is unique for all training images.

Waveletfaces inspired the Pixel Clustering framework. Pixel Clustering is a generalization of Waveletfaces so that Waveletfaces is a method within the proposed framework. Many other feature extraction methods can be defined using Pixel Clustering. In this paper, we define two unsupervised feature extraction methods: IntensityPatches and RegionPatches.

The proposed methods are defined using a precise algebraic notation. It allows an accurate interpretation and implementation of the proposed feature extractions. However, the proposed methods have simple interpretations. RegionPatches is a direct extension of Waveletfaces. Waveletfaces sums pixel in 2×2 non-overlapping patches, RegionPatches averages pixels intensities in squared neighborhoods of arbitrary size. Every square has the same size, and they define a grid as shown in Figure 8. Each feature extracted with Region Patches is the mean intensity in a specific square.

A feature extracted with IntensyPacthes is also the average intensity for a set of pixels, but the region defined by this set of pixels is not a square. Each region represents areas of similar intensity within the training set. The example in Figure 7 shows several regions automatically defined by IntensityPatches for a set of frontal face images. Those regions are meaningful, the average intensity level within each region describes many typical facial features. Comparing the intensity of the regions on the top of the head with skin regions it is possible to define baldness levels. The same for beard, mustache, and eyebrow. It describes position and color of the eyes using specific regions. It is also possible to define many other features such the shape of the head.

Both methods are within the Pixel Clustering framework. For Intensity-Patches, each cluster contains pixels of similar intensity. For RegionPatches, the

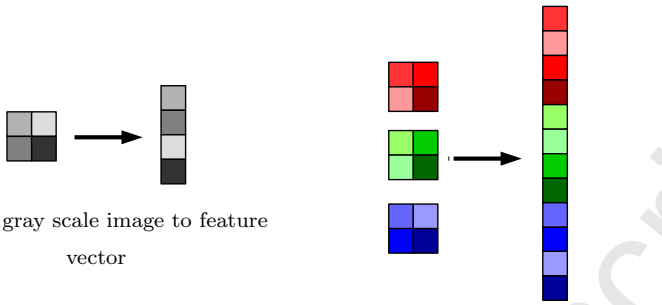

Figure 2: Converting a image (gray scale or color) into a feature vector.

cluster is a set of neighbor pixels. We proposed a preliminary version of these methods in the conference paper [16]. Before formally defining those methods in the remainder of this section, we present some preliminary concepts.

Even though the proposed technique can be used for other kind of dataset, we focus in applications that use images. We consider the use of feature extraction for two kinds of image: grayscale and color. A gray scale image is represented by a matrix, each scalar within the matrix is an intensity level. A color image is typically represented with a color model with three primary colors: red, green, and blue.

Each image (color or grayscale) can be converted into a feature vector by concatenating the columns of each intensity matrix into a single column vector, as described in Figure 2. Consider an image with height h, width w and s color, $s = 1$ for a gray scale image and $s = 3$ for a typical color image. This image can be transformed into a column vector of $h \times w \times s$ elements. Assuming a training set of n images with m pixels each, there are n raw feature vectors with dimension $m = h \times w \times s$. The proposed technique assumes, without loss of generality, that each training or test image has the same number of lines, columns, and colors. Then the same cluster indexes pixels in any image.

3.1. Pixel vector

A *pixel vector* is a vector that contains information that is representative for the same pixel on every image. It can be either: intensity pixel vector or region

Figure 3: Examples of the proposed pixel vectors for the ORL gray scale face database. The region pixel vector (bellow) represent the position (18,23) in the image. The intensity pixel vector (above) contains the intensity for position (18,23) in all the 8 images from this example.

pixel vector. Figure 3 shows examples of both pixel vectors.

Definition 1. *An intensity pixel vector is an n-dimensional vector that contains the intensity value for the same pixel position through the n training images.*

For the training set, m intensity pixel vectors are extracted. A pixel vector is constructed for every position in the feature vector. Each one representing the pixel intensity for a pixed position for a specific color.

Definition 2. *A region pixel vector is a 2-dimensional vector that contains the position (x, y) of the pixel.*

The region pixel vector ignores whether the image is in grayscale or color since it treats every image as grayscale. Then the clusters defined for a color are the same for the others. This does not create any inconsistence because every color is represented by a matrix of the same size. Also there is no inconsistencies due the pixels intensity values because those values are not considered by this definition of pixel vector. Finally, the $h \times w$ region pixel vector are extracted for the training set, in spite of the number s of colors.

3.2. Feature extraction

In order to define a feature extraction algorithm using the Pixel Clustering framework it is necessary to set: (1) a definition of a pixel vector; (2) a cluster algorithm; and (3) a linear combination of the pixels within a cluster. Once the set up is performed, the algorithm has three main steps:

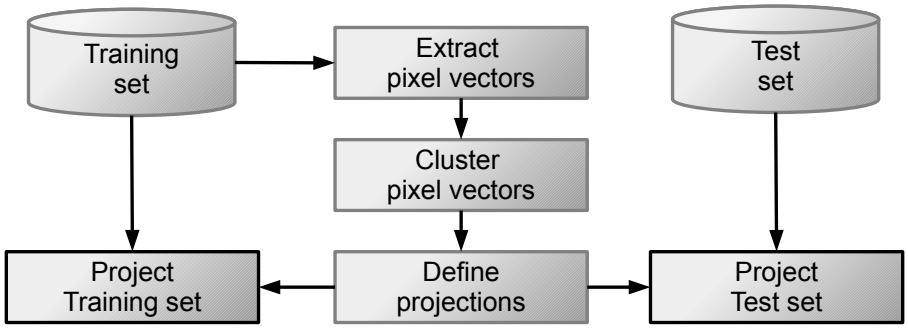

Figure 4: Flowchart of the Pixel Clustering framework.

1. Extract pixel vectors from the training set;
2. Apply a cluster algorithm to the pixel vectors;
3. Define, for each cluster, a projection of its features.

The first step is to extract pixel vectors according to the chosen definition. It can be one of the definitions described above. The second step is to apply a clustering algorithm to these pixel vectors, such as k-means. The output of this step is a set of clusters, which depend on the chosen cluster algorithm and distance metric. The third step is to extract features that use the information of the clustering. It extracts a new feature by combining the pixels within a cluster. Figure 4 shows the flowchart of the different steps of the proposed method.

3.3. Notation

Let I_a be an image with s color band, height h and width w, represented by an s-tuple of matrices, each one of dimensions $h \times w$, where $I_a(i,j,l) \in \mathbb{R}$ is the intensity at the image position (i,j,l), $0 \leq i < w$ and $0 \leq j < h$, $0 \leq l < s$. This *image* can also be represented as a vector

$$\mathbf{x}_a = [x_{a1}\ x_{a2}\ \ldots\ x_{am}]^T \tag{1}$$

where $m = h \times w \times s$ and $x_{ap} = I_a(i, j, l)$, $p = 1 + j + hl + l(hw)$, as described in Figure 2. For the pattern recognition task, x_{ap} is the p-th feature for the a-th example in a dataset, $p = 1, 2, \ldots, m$, and $a = 1, 2, \ldots, n$; m is the number of pixels intensities within an image and n is the number of training images.

The dimensionality reduction aims to find, for each image $\mathbf{x}_a$, a new feature vector

$$\mathbf{x}'_a = [x'_{a1}\ x'_{a2}\ \ldots\ x'_{ak}]^T, \tag{2}$$

with $k < m$; x'_{aq} is the new q-th feature for the a-th example, $q = 1, 2, \ldots, k$.

Each x'_{aq} is obtained by combining the features of $\mathbf{x}_a$ as a linear projection:

$$x'_{aq} = \boldsymbol{\alpha}_q^T \mathbf{x}_a, \tag{3}$$

$\boldsymbol{\alpha}_q$ is a column vector which contains the coefficients for the extraction of the q-th feature.

In order to extract the new feature vector $\mathbf{x}'_a$, it is necessary to extract all the new k features. This can be performed as follows:

$$\mathbf{x}'_a = \mathbf{A}^T \mathbf{x}_a, \tag{4}$$

where $\mathbf{A}$ is the projection matrix with columns $\boldsymbol{\alpha}_q$, $q = 1, \ldots, k$, as a projection vector of the new space of features:

$$\mathbf{A} = [\boldsymbol{\alpha}_1\ \boldsymbol{\alpha}_2\ \ldots\ \boldsymbol{\alpha}_k]. \tag{5}$$

The pixel clustering approach creates hard clusters that can be written as a vector

$$\mathbf{c}_q = [c_{q1}\ c_{q2}\ \ldots\ c_{qm}]^T, \tag{6}$$

where $c_{qv} = 1$, if the feature v belongs to the cluster q, and $c_{qv} = 0$, otherwise. A cluster is defined for each new feature. From the q-th cluster $\mathbf{c}_q$, the projection vector, $\boldsymbol{\alpha}_q$, for the q-th feature is defined. A remark on the Pixel Clustering approach is that the clustering algorithm is not performed on the examples of the training set, but on the pixel vectors that are extracted from the training set.

3.4. IntensityPatches

In the Pixel Clustering framework, we propose a feature extraction technique called IntensityPatches, that uses the traditional k-means clustering algorithm [17] applied to the pixel vectors. It combines the pixels within a cluster by taking the average of the pixels intensity values. The k-means algorithm associates each pixel vector to the cluster of the closest prototype, iteratively recalculating the prototype of each cluster as the mean vector of the cluster. Similar to PCA, IntensityPatches generates orthogonal projection vectors and enforces each new feature to be less correlated with the others by clustering similar features. The projection vectors for IntensityPatches are orthogonal because the coefficient for a pixel is nonzero only for a single projection vector, but unlike PCA, it does not combine all the features because it does not use features outside the cluster.

For an m-dimensional dataset with n samples, PCA is able to extract up to $n-1$ features [18]. However, IntensityPatches is able to extract from 1 to m features even in problems with high dimensionality (when $m >> n$). This property is particularly useful for situations where new classes are added to the dataset, as described in the experiments in [16].

The Intensity pixel vectors are extracted as lines from $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_n]$, where a column $\mathbf{x}_a$, $a = 1, \ldots, n$, is an image from the training set. Since an image is described as $\mathbf{x}_a = [x_{a1}\ x_{a2}\ \ldots\ x_{am}]^T$, an intensity pixel vector is denoted as

$$\mathbf{p}_r = [x_{1r}\ x_{2r}\ \ldots\ x_{nr}]^T, \tag{7}$$

$r = 1, \ldots, m$, which contains the value of the r-th pixel for every image in the training set.

The k-means cluster algorithm [17] is then performed over $\mathbf{p}_r$ vectors yielding hard clusters $\mathbf{c}_q = [c_{q1}\ c_{q2}\ \ldots\ c_{qm}]^T$, $q = 1, \ldots, k$.

For each cluster $\mathbf{c}_q$ a transform vector is created

$$\boldsymbol{\alpha}_q = \frac{1}{\|\mathbf{c}_q\|^2}\mathbf{c}_q, \tag{8}$$

where $\|\cdot\|$ is the norm of a vector. Once $c_{qo} \in \{0, 1\}$, $o = 1, \ldots, m$, as discussed in Equation 6, $\|\mathbf{c}_q\|^2$ is the number of pixel in the q-th cluster. Then α_q is $1/\|\mathbf{c}_q\|^2$ for the pixels position that belong to the q-cluster, or zero otherwise. In this way, $\boldsymbol{\alpha}_q^T \mathbf{x}_a$ is the average of pixels intensity values within the q-th cluster.

According to Equation 5, the transformation matrix $\mathbf{A}$ is computed by:

$$\mathbf{A} = \left[\frac{1}{\|\mathbf{c}_1\|^2}\mathbf{c}_1 \ \frac{1}{\|\mathbf{c}_2\|^2}\mathbf{c}_2 \ \ldots \ \frac{1}{\|\mathbf{c}_k\|^2}\mathbf{c}_k \right]. \tag{9}$$

The IntensityPatches dimensionality reduction is obtained by applying this matrix $\mathbf{A}$ in Equation 4.

3.5. RegionPatches

The second proposed algorithm, called RegionPatches, have a *region pixel vector* $\mathbf{p}_r = [i_r, j_r]^T$ which is the position of the pixel r, as in Definition 2. A cluster includes pixels from all color that have the same pixel positions. Similar to IntensityPatches, it averages pixel intensities values within each cluster in order to create a new feature. It fits squares of the same size on the image, whereas each pixel within a square belongs to this square cluster, which is the trivial result of a clustering algorithm using the Chebychev distance metric [19]. The Chebychev distance between two vectors, $\mathbf{a}$ and $\mathbf{b}$, is the function $d(\mathbf{a}, \mathbf{b}) = \max_{i=1}^{m} |a_i - b_i|$. Using this metric, the place of the points that have the same distance to the center is a square. We choose the Chebychev distance because its a very simple case similar to the Haar Waveletfaces. More details on the correspondence between Waveletfaces and RegionPatches are provided in the next section.

The RegionPatches clusters are extracted from spatial relations within the images. They depend on the dimension of the images whereas in the IntensityPatches they depend on the pixel values. In RegionPatches each cluster is a non-overlap squared region within the image, but for the last line and last column of these regions the squares may be cropped due to the image boundaries. Each hard cluster within the RegionPatches approach is a $h \times w$ matrix $\mathbf{C}_{(d,e)}$,

this matrix is an image mask that is 1 for the pixels the belong to the square (d, e) and zero otherwise, it is defined as:

$$\mathbf{C}_{(d,e)} = \begin{bmatrix} c_{00} & \dots & c_{0w} \\ \vdots & \ddots & \vdots \\ c_{h0} & \dots & c_{hw} \end{bmatrix} \tag{10}$$

with

$$c_{ij} = \begin{cases} 1 & \text{if } du \leq i < (d+1)u,\ eu \leq j < (e+1)u \\ 0 & \text{otherwise} \end{cases}, \tag{11}$$

where $0 \leq i < w$ and $0 \leq j < h$; u is the square side length, it is a parameter for this technique similar to k in the *IntensityPatches*.

A large value of u results in a small number of clusters. The minimum value for u is $u = 1$, which generates $h \times w$ clusters, each cluster containing a single pixel. If $u \geq \max(h, w)$ it generates a single cluster that contains every pixel. The square side u can be approximated for a fixed number k of clusters as:

$$u = \left\lfloor \sqrt{\frac{wh}{k}} \right\rfloor. \tag{12}$$

Each pair (d, e) defines a cluster, with $d = 0, ..., \lceil w/u \rceil - 1$, and $e = 0, ..., \lceil h/u \rceil - 1$. There are $k' = \lceil w/u \rceil \times \lceil h/u \rceil$ clusters, $k' \approx k$. Each $\mathbf{C}_{(d,e)}$ is converted into a vector

$$\mathbf{c}'_q = [c_{q1}, c_{q2}, \dots, c_{qm}]^T, \tag{13}$$

$q = 1, \dots, k'$, where $m = hw$, $q = 1 + e + d\lceil h/u \rceil$, and $c_{qp} = c_{ij}$ for $p = 1 + i + wj$. For color images the computed clusters $\mathbf{c}'_q$ are repeated s times in order to define a projection vector for the whole image:

$$\mathbf{c}_q = \begin{bmatrix} \mathbf{c}'_{q1} \\ \vdots \\ \mathbf{c}'_{qm} \end{bmatrix} = \begin{bmatrix} c_{1q1} \\ \vdots \\ c_{1qm} \\ c_{sq1} \\ \vdots \\ c_{sqm} \end{bmatrix}, \tag{14}$$

with $c_{lqm} = c_{qm}$, $l = 1, \ldots, s$. This means that the cluster have su^2 pixels for a region of u^2 pixels. Pixels of different color belongs to the same cluster once they all have the same position (i, j) within it own cluster.

In order to RegionPatches perfoms dimensionality reduction, it should replace the calculated clusters to compute $\mathbf{A}$ through Equation 9 and finally replacing these results into Equation 4. If RegionPatches squares have side length, $u = 2^g$, $g = 1, 2, 3, ...$, considering images with a single color, it emulates Waveletfaces [1], because this is similar the low-low band of the Wavelet transform using a Haar function. In this case, the only difference is that RegionPatches averages pixel intensities while Haar Waveletfaces sum them up.

4. Image recognition experiments

In the present experiments, we used four image datasets: a grayscale and a color face dataset, and a grayscale and a color object dataset. We compare the proposed methods with four unsupervised feature extraction techniques: two linear, Haar Waveletfaces and PCA (Eigenfaces); and two non-linear, autoencoders [3] and PCANet [5]. Waveletfaces for color image computes Waveletfaces within each band and concatenates vector of extracted features. The level of the Waveletfaces is computed as $\lceil \log_4(hw/k) \rceil$, with h defined as the image height, w the image width and k the requested number of extracted features. PCA is computed in the standard way. The parameters for PCANet are $k_1 = 5$, $k_2 = 5$, $L_1 = 2$, and the histograms are computed for 8×8 non-overlapping block size. The convolution window has dimension $k_1 \times k_2$. The number of principal components is L_1. We tuned the parameters for the ORL dataset, they differ from the authors suggestions only in L_1, which they set equal to 8 [5]. To perform PCANet on colored images, we transformed each color image to grayscale. We tuned the autoencoder parameter for the ORL dataset: the number of features is equal to the number of neurons in the hidden layer; the activation function is the logistic sigmoid; the maximum number of training epochs is 300 because we did notice a significant improvement in accuracy for a longer training.

The methods are compared for approximately 16 extracted features, depending on the restriction of the method. However, only three k-means iterations are used, which results in a lower computational time for the proposed IntensityPatches.

We compared the methods in two different scenarios: the original data and data with random Gaussian noise. The noise has zero mean and standard deviation equal to 50. This means that each feature from each sample is summed with a pseudo-random number from a Gaussian distribution. Prior the summation the pseudo-random number is multiplied by 50, *i.e.*, the standard deviation.

Accuracy, defined as the number of corrected classified instances divided by the total number of test instances, is used to evaluate the experiments. For every database 100 holdout experiments were performed, except when using autoencoder for which we perform only 10 holdouts. Then, we computed the mean and standard deviation of the accuracy.

There are two setups for training and test partition. In the first setup, each training set in a holdout experiment has a single example from each class, the other examples from each class are in the test set. It means 40 examples in the training set and 360 in the test set for the ORL dataset, 50 in the training and 700 in the test for the GT dataset, 20 in the training and 1,420 in the test for the COIL20 dataset, and 100 in training and 7,100 in the test for the COIL100 dataset. In the second setup, each training set in a holdout experiment has nine examples from each class, the other examples from each class are in the test set. It means 360 examples in the training set and 40 in the test set for the ORL dataset, 450 in training and 300 in test for the GT dataset, 180 in training and 1,260 in the test for the COIL20 dataset, and 900 in training and 6,300 in the test for the COIL100 dataset.

The results presented only for the Nearest Neighbor Classifier (1-NN) with Euclidean distance. However the same experiments were also tested using other classifiers: decision tree, linear support vector machine (SVM), and linear discriminant and the results where either very similar to the ones obtained using 1-NN or presented a very lower accuracy (around 10%) depending on the

dataset.

We used a deterministic seed for random sampling. This allows an exact replication of the results. Matlab® code is fully available upon request. Datasets and results are presented in the following subsections.

4.1. Datasets

Four datasets were used in the experiments. Two face dataset: one in gray scale (ORL) and one in color (GT). Two datasets of images from objects: one in gray scale (COIL20) and one colored (COIL100). The colored datasets are RBG, *i.e.*; they have three color spectra: red, green and blue. Table 1 summarizes the information on the datasets that are described below:

- ORL[1]: This dataset contains 10 images for each 40 subjects (a total of 400 images). Each cropped gray scale image has 92×112 pixel, consequently each image is represented by a $10,304$-dimensional vector. Samples from this dataset are shown in Figure 5.

- Georgia Tech (GT) [2]: This dataset contains 15 images for each 50 subjects (a total of 750 images). Each cropped color image has a different size, all the image were resized to 150×200 pixel, consequently each image is represented by a $(30,000 \times 3)$ $90,000$-dimensional vector.

- COIL20 database[3]: This dataset contains 1,440 images (72 for each of its 20 objects) with 256 gray levels. The size of each image is 128×128 pixels, consequently each image is represented by a 16,384-dimensional vector.

- COIL100 database[4]: This dataset contains $7,200$ images (72 for each of its 100 objects) with 256^3 color levels (256 for each color band: red, green, blue). The size of each image is 128×128 pixels, consequently each image is represented by a $(16,384 \times 3)$ $49,152$-dimensional vector.

[1] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[2] http://www.anefian.com/research/face_reco.htm
[3] http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php
[4] http://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php

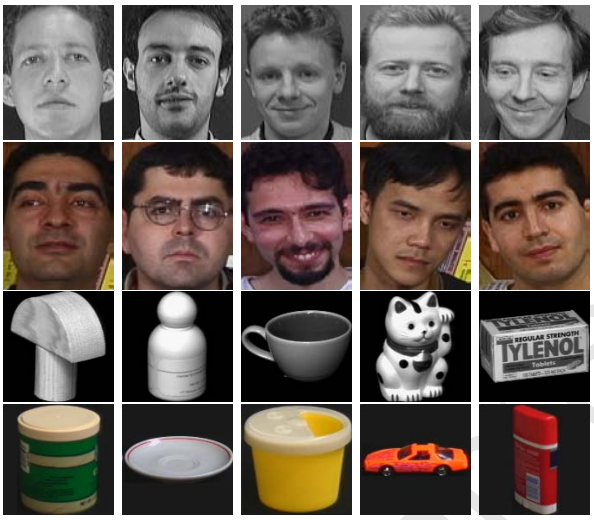

Figure 5: Each line contains randomly selected example from each dataset, from top to bottom: ORL, GT, COIL20, and COIL100.

Figure 6: Noisy examples from each dataset, from left to right: ORL, GT, COIL20, and COIL100.

4.2. Results and discussion

The results of these experiments are evaluated concerning the number of extracted features, accuracy, and computational time. As noticed, we defined for each method to extract 16 features. For Waveletfaces and RegionPatches this number can not always be defined precisely, so we used an approximated number. The same occurs for PCANet, but few features were extracted since this method has a significant impact on classification accuracy. The mean accuracy is expressed in percentage and the standard deviation is presented in round brackets. The time was measured using the same computer with the

Table 1: Description of the dataset

Dataset	Type	Color	Classes	Instances	Dimension
ORL	face	gray	40	400 (10×40)	$10,304$
GT	face	RGB	50	750 (15×50)	$90,000$
COIL20	object	gray	20	1,440 (72×20)	16,384
COIL100	object	RGB	100	7,200 (72×100)	49,152

same configuration and softwares[5]. The mean time for the feature extraction was measured in seconds; the standard deviation is also presented in round brackets.

We describe the results for the ORL dataset using a single training image per class in Table 2. First, the "original data" or data without noise was considered. The accuracy of the raw data (10,304 features) was 68.29%. Comparing to the feature extraction methods, the highest accuracy for extracted features was obtained using PCA (16 features), 62.96%. Followed by Waveletfaces 62.5% (12 features), RegionPatches (20 features) adn PCANet (672 features) $\approx 59\%$, IntensityPatches $\approx 58\%$ (16 features), the lowest accuracy is from Autoencoder $\approx 31\%$ (16 features). The proposed methods did not present great accuracy for this dataset using so few features. This behavior was expected once the proposed methods present higher accuracy for more extracted features, as described in the next section. Similarly, Autoencoder, which is widely used in Deep Neural Networks [3], has a very low accuracy using few features. Autoencoder demands more than 256 features to have accuracy close to the other methods, for this dataset.

The minimum mean time for feature extraction is achieved by RegionPatches, 0.03 seconds. Moreover, the maximum mean time by Waveletfaces, 2.19 seconds, two orders of magnitude greater than RegionPatches. IntensityPatches and PCA have the mean time of 0.21 and 0.11 seconds respectively, they are

[5] A8-5500B processor with 28GB DDR3 RAM, Windows 7® and Matlab 2016b® .

Table 2: Experiment results for ORL face dataset using a single example per class in the training set.

ORL (original)		1 example per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	57.86% (2.78%)	0.21 (0.12)
RegionPatches	20	59.43% (2.66%)	0.03 (0.00)
Waveletfaces	12	62.50% (2.29%)	2.19 (0.17)
PCA	16	62.96% (2.75%)	0.11 (0.03)
PCANet	672	59.73% (2.67%)	6.30 (1.09)
Autoencoder	16	30.58% (9.82%)	77.57 (15.88)
Raw	10,304	68.29% (2.50%)	–

ORL (noisy)		1 example per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	54.56% (2.94%)	0.18 (0.01)
RegionPatches	20	59.01% (2.62%)	0.04 (0.02)
Waveletfaces	12	61.70% (2.49%)	2.02 (0.07)
PCA	16	38.65% (4.02%)	0.10 (0.01)
PCANet	672	9.58% (1.78%)	6.38 (1.33)
Autoencoder	16	15.22% (3.31%)	76.02 (1.43)
Raw	10,304	67.51% (2.46%)	–

in the same order of magnitude. PCANet is about three times slower than Waveletfaces, training a PCANet requires computing PCA for every overlap patch in the training set. Training an Autoencoder demands a huge amount of computational time, about 1,000 more than the faster method.

For noisy data, the time analysis is very similar, but the accuracies for the recognition after feature extraction are different. Accuracy using PCA shows an enormous difference of 24.31%. PCA is very sensitive to noise, which probably caused an overfit in the learning by expecting that the noise in the test set to be

in the same pixel position of the training set. The same effect is noted PCANet, reducing the accuracy to about 1/6. The accuracy of Autoencoder is reduced form 30.58% to 15.22%.

The other methods are less sensitive to the Gaussian noise. Since, the noise inserted in the data has zero mean, the sum of the intensities from a set of pixel does not vary much from the same sum without noise. The summation of images with Gaussian noise tends cancel the effect of the noise – the sum of zero mean samples approaches zero.

RegionPatches with accuracy of 59.01%, presented the smalled difference: 0.42% less than accuracy with the original data. Waveletfaces have accuracy reduced to 61.70%, a difference of 0.80%. IntensityPatches had the accuracy reduced to 54.56%, a difference of 3.30%. IntensityPatches is probably more sensitive to noise than the other two methods because it learns its regions from the training set, similarly to PCA and its projections. So the learned regions can be biased by the noise. Nonetheless IntensityPatches presented an accuracy difference much smaller than PCA.

The Nearest Neighbor classifier is also not very sensitive to Gaussian noise. The accuracy for the raw data with noise was 67.51% and without noise 68.29%, the difference is of 0.78%. This small sensitivity for Gaussian noise is also due the summation effect. The classifier used here calculates the Euclidean distance, which sum the squares of the differences between each feature. As expected, the sum of the squares of the noise also approaches zero.

Waveletfaces showed to be the best method for the ORL dataset in this experiment, this same dataset was evaluated in the Waveletfaces paper [1]. However the proposed method also presented a compatible accuracy, small sensitivity to noise, and very small time for feature extraction.

When using nine training examples per class for the ORL dataset (3), most of the methods have accuracy higher than 96% even in the presence of noise. The accuracy for IntensityPacthes is close to 93% (with or without noise). The accuracy for autoencoder are 62% (noise free) and 47% (with noise). PCANet accuracy is greatly reduced to 14% for noise data. The computational time did

Table 3: Experiment results for ORL face dataset using nine examples per class in the training set.

ORL (original)		9 examples per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	92.92% (3.88%)	1.12 (0.07)
RegionPatches	20	96.38% (2.83%)	0.03 (0.00)
Waveletfaces	12	95.59% (3.19%)	2.14 (0.11)
PCA	16	97.13% (2.43%)	1.86 (0.29)
PCANet	672	96.28% (2.83%)	15.34 (1.05)
Autoencoder	16	62.05% (16.21%)	254.61 (4.74)
Raw	10,304	97.41% (2.32%)	–

ORL (noisy)		9 examples per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	92.82% (3.64%)	1.10 (0.01)
RegionPatches	20	96.08% (2.88%)	0.03 (0.00)
Waveletfaces	12	95.05% (3.14%)	2.08 (0.04)
PCA	16	96.74% (2.52%)	1.63 (0.04)
PCANet	672	14.46% (4.50%)	16.71 (1.57)
Autoencoder	16	47.44% (11.67%)	256.67 (2.01)
Raw	10,304	96.90% (2.71%)	–

not increase for RegionPatches and Waveletfaces. It is about five times greater for IntensityPatches, 16 times higher for PCA, and the triple for PCANet and autoencoder. The proposed method still presents low computational time and high accuracy even in the presence of noise.

The results for the GT color face dataset, using a single example per class in the training set, are shown in Table 4. Similar to the ORL dataset, the feature extraction time has three different orders of magnitude: 10^{-1} for RegionPacthes, 10^{0} for IntensityPatches and PCA, and 10^{1} for Waveletfaces. The reason is the same. The time to evaluate the original and noisy data were, respectively, 0.40

Table 4: Experiment results for GT face dataset using a single example per class in the training set.

GT (original)		1 example per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	41.73% (2.65%)	1.64 (0.03)
RegionPatches	20	47.67% (2.69%)	0.40 (0.04)
Waveletfaces	12	35.33% (2.21%)	17.45 (0.10)
PCA	16	42.20% (3.07%)	1.72 (0.09)
PCANet	1,900	2.09% (0.13%)	30.04 (1.21)
Autoencoder	16	10.97% (4.01%)	695.81 (12.08)
Raw	90,000	44.63% (3.28%)	–

GT (noisy)		1 example per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	40.11% (2.56%)	1.68 (0.06)
RegionPatches	20	47.57% (2.72%)	0.40 (0.05)
Waveletfaces	12	35.25% (2.21%)	17.95 (1.27)
PCA	16	19.98% (2.97%)	1.67 (0.03)
PCANet	1,900	30.16% (2.18%)	30.90 (1.12)
Autoencoder	16	5.77% (2.41%)	678.50 (11.19)
Raw	90,000	44.45% (3.18%)	–

and 0.40 for RegionPacthes, 1.64 and 1.68 for IntensityPatches, 1.72 and 1.67 for PCA, 17.45 and 17.95 for Waveletfaces. These times were approximately 10 times larger than for the ORL dataset because there were approximately 10 times more features to be evaluated, as can be seen in the difference in the raw images from the two datasets.

The highest accuracy was achieved by RegionPatches (20 features), 47.67% in the original data, with a small difference with the noisy features (47.57%). For the original data, the accuracies of IntensityPatches, 41.73%, and PCA, 42.20%, are on par. The PCA accuracy decreases to about half 19.98% with noisy

Table 5: Experiment results for GT face dataset using nine examples per class in the training set.

GT (original)		9 examples per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	71.74% (2.23%)	13.13 (0.44)
RegionPatches	20	79.98% (1.88%)	0.39 (0.03)
Waveletfaces	12	67.65% (2.22%)	18.51 (0.52)
PCA	16	76.77% (1.97%)	19.38 (0.19)
PCANet	1,900	2.66% (0.31%)	58.94 (1.99)
Autoencoder	16	8.87% (8.54%)	2,528.86 (20.55)
Raw	90,000	77.96% (1.88%)	–

GT (noisy)		9 examples per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	73.67% (1.74%)	12.69 (0.04)
RegionPatches	20	79.90% (1.87%)	0.40 (0.01)
Waveletfaces	12	67.01% (2.09%)	17.77 (0.17)
PCA	16	73.99% (2.13%)	19.25 (0.17)
PCANet	1,900	54.77% (2.16%)	60.37 (2.08)
Autoencoder	16	13.47% (5.11%)	2,435.73 (37.43)
Raw	90,000	77.75% (1.87%)	–

features while the accuracy for IntensityPatches presented a small reduction to 40.11%. Waveletfaces was restricted to extract 12 features and showed an accuracy of 35.33% and 35.25% for original and noisy data, respectively.

An important issue for the Waveletfaces in this dataset is that it extracted only four features for each of the three color band. Wavelet level 7 for an 150×200 image defines four regions and level 6, 12 regions. The number of features for a color image for level 7 is 12 features, for level 6, 36 features. RegionPatches is more flexible than Waveletfaces and allows more options for the number of extracted features. Using the same example, RegionPatches

allows choosing 12, 20, 24, 30, 35, or 42 in the same range.

To compare RegionPatches and Waveletfaces for the same number of features, the accuracy of RegionPatches for 12 extracted features was 44.47%(2.27%) for original data and 44.46%(2.43%) for noisy data. The computational time for feature extraction was 0.31 (0.03) seconds. RegionPatches also differs from Waveletfaces dealing with color images. It explains the difference for the same number of extracted features. While Waveletfaces extracts features from each color band, RegionPatches averages the pixel intensities from all color within the same position. For this dataset, the RegionPatches approach achieved a higher accuracy.

For the noiseless GT dataset, PCANet has the accuracy close theoretical minimum (the inverse of the number of classes). Such result reveals another advantage of proposed methods: few parameters. PCANet is 30 times slower than proposed methods; it takes too long for tuning to many parameters. Strangely, its accuracy increased from 2% to 30% in the noisy data. Autoencoder also was not able to extract discriminant features in this case. Its accuracy is significantly low, mainly with noise data. Moreover, its training time is hundred times larger compared with proposed methods.

A larger training set for the GT dataset 5 increases the accuracy for most of the methods. PCANet and autoencoder presented a significantly low accuracy. The other methods have accuracy close or higher than 70%. The greatest accuracy of 80% is reached by RegionPatches. Computational time increased for most of the methods compared with smaller training set case. We conclude for the GT dataset that the proposed methods are simple, fast, demand few parameters, and present high accuracy even in the presence of noise.

The COIL20 dataset is composed of a grayscale object images, the experimental results for COIL20 are shown in Table 6. This dataset is very different from the datasets of the previous experiments. For the face datasets, each image has a similar structure. However, for object datasets, the shapes and background areas vary depending on the class. For the COIL20 the shapes also vary within the class because the images of each object are rotations of the original image.

Table 6: Experiment results for COIL20 object dataset using a single example per class in the training set.

COI20 (original)		1 example per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	63.15% (3.09%)	0.29 (0.02)
RegionPatches	16	58.42% (2.28%)	0.14 (0.02)
Waveletfaces	16	58.31% (2.28%)	8.43 (0.08)
PCA	16	60.51% (2.41%)	0.20 (0.02)
PCANet	1,024	55.14% (2.25%)	28.07 (1.00)
Autoencoder	16	38.01% (9.51%)	113.34 (3.86)
Raw	16,384	61.39% (2.21%)	–

COIL20 (noisy)		1 example per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	61.53% (3.02%)	0.28 (0.02)
RegionPatches	16	58.37% (2.27%)	0.13 (0.02)
Waveletfaces	16	58.22% (2.29%)	8.36 (0.06)
PCA	16	50.69% (4.75%)	0.19 (0.02)
PCANet	1,024	19.46% (2.31%)	28.87 (0.90)
Autoencoder	16	33.44% (6.34%)	115.32 (3.92)
Raw	16,384	61.39% (2.21%)	–

RegionPatches, IntensityPatches, and PCA presented similar average time. IntensityPatches and PCA timing increase with more training examples (7) but still at least 8 times faster than Waveletfaces, 30 faster than PCANet, and 300 faster than Autoencoder.

The mean accuracy for raw images (16,384 dimensions) is 61% with or without noise; with more training examples it goes to 89%. The greatest mean accuracy was obtained when the features are extracted with the proposed method IntensityPatches: 63% for the original data and 61% for the noisy data (1 example

Table 7: Experiment results for COIL20 object dataset using nine examples per class in the training set.

COIL20 (original)		9 examples per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	91.08% (1.31%)	1.07 (0.03)
RegionPatches	16	87.94% (1.23%)	0.13 (0.01)
Waveletfaces	16	87.85% (1.25%)	8.63 (0.09)
PCA	16	90.62% (1.29%)	1.60 (0.13)
PCANet	1,024	84.33% (1.30%)	33.88 (1.64)
Autoencoder	16	53.71% (8.19%)	356.18 (4.24)
Raw	16,384	88.73% (1.18%)	–

COIL20 (noisy)		9 examples per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	90.76% (1.44%)	1.07 (0.03)
RegionPatches	16	87.89% (1.24%)	0.13 (0.01)
Waveletfaces	16	87.75% (1.25%)	8.54 (0.08)
PCA	16	90.01% (1.33%)	1.23 (0.02)
PCANet	1,024	30.90% (1.68%)	35.38 (1.88)
Autoencoder	16	46.30% (7.43%)	365.73 (19.47)
Raw	16,384	88.68% (1.21%)	–

per class) and 91% for 9 examples per class in training (noisy or noiseless data). The accuracy for IntensityPatches was higher than for the raw data. It probably occurs because IntensityPatches reduces the influence of the background image in the classification. The proposed methods presented low computational time and accuracy comparable to or greater than other methods. They also suffer small influence of noise for classification.

The COIL100 is a color object image dataset, which is a colored and extended version of COIL20. While COIL20 has 20 objects (classes), COIL100 has 100

Table 8: Experiment results for COIL100 object dataset using a single example per class in the training set.

COIL100 (original)		1 example per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	63.62% (2.23%)	3.63 (0.12)
RegionPatches	16	49.60% (1.08%)	2.06 (0.20)
Waveletfaces	12	52.22% (1.19%)	131.09 (0.91)
PCA	16	62.87% (1.21%)	4.28 (0.18)
PCANet	1,024	2.00% (0.00%)	92.22 (6.06)
Autoencoder	16	21.74% (8.89%)	663.50 (10.14)
Raw	49,152	59.91% (1.13%)	–

COIL100 (noisy)		1 example per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	63.36% (2.13%)	3.41 (0.16)
RegionPatches	16	49.52% (1.05%)	2.06 (0.35)
Waveletfaces	12	52.12% (1.17%)	131.83 (0.49)
PCA	16	58.70% (1.24%)	4.16 (0.15)
PCANet	1,024	38.88% (1.30%)	92.07 (0.36)
Autoencoder	16	17.16% (6.43%)	667.89 (8.81)
Raw	49,152	59.86% (1.13%)	–

objects. Results for the experiments with COIL100 dataset are presented in Table 8. The mean accuracy for the raw data (19,152 features) is 60% (noisy or noiseless data) with a single training example per class; and about 89% with nine training examples per class (noisy or noiseless).

The highest mean accuracy for COIL100 within the small training set is also achieved using IntensityPatches 63.62% (original data) 63.36% (noisy data). This accuracy did not alter much when using noisy data. PCA is on par with the original data with an accuracy of 62.87%, but the mean accuracy decreases

Table 9: Experiment results for COIL100 object dataset using nine examples per class in the training set.

COIL100 (original)		9 examples per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	90.35% (0.80%)	17.03 (0.14)
RegionPatches	16	81.64% (0.61%)	2.02 (0.12)
Waveletfaces	12	83.57% (0.70%)	135.28 (2.35)
PCA	16	90.74% (0.55%)	26.52 (0.38)
PCANet	1,024	2.00% (0.00%)	106.76 (0.37)
Autoencoder	16	39.67% (12.84%)	3,877.87 (50.83)
Raw	49,152	89.13% (0.59%)	–

COIL100 (noisy)		9 examples per class in training	
Method	Features	Accuracy	Time (s)
IntensityPatches	16	90.56% (0.98%)	16.33 (0.15)
RegionPatches	16	81.46% (0.60%)	1.81 (0.07)
Waveletfaces	12	83.38% (0.68%)	133.27 (0.54)
PCA	16	90.53% (0.55%)	26.04 (0.36)
PCANet	1,024	65.96% (0.64%)	107.66 (0.45)
Autoencoder	16	43.06% (7.62%)	3,964.22 (46.34)
Raw	49,152	89.00% (0.59%)	–

to 58.70% in the noisy version. RegionPatches and Waveletfaces presents a similar result. Also, the mean accuracy did not alter much in the presence of noise. RegionPacthes showed a mean accuracy of 49.60% and 49.52%, and Waveletfaces 52.22% and 52.12%, both for original and noisy dataset respectively. For the larger traning set, results are similar. IntensityPatches and PCA have the greatest accuracies, about 90%. RegionPatches and Waveletfaces have around 80% accuracy.

PCANet and Autoencoder presented significantly lower accuracies in every

case. It is worth to note that advantage of using linear feature extraction methods: they are simpler, faster, and can achieve higher accuracy in many cases, as in these experiments. Also, for this dataset, the proposed methods presented the benefit os speed, high accuracy, simplicity, few parameters, and tolerance to noise.

In the next section, we analyze the specific problem of face recognition using IntensityPatches and extracting more features.

5. Face recognition experiments

In this section, two experiments and their results are described to evaluate Intensity Patches for the face recognition task. We compare IntensityPatches and PCA accuracies using the Nearest Neighbor (1-NN) classifier is applied using Euclidean distance. For each dataset, we randomly select half of the images as the training set, and the other half as the test set. These experiments were performed using three well-known face data sets: Yale, ORL, and UMIST. The Yale dataset has 165 images, 11 images for each one of the 15 subjects. These images are well registered and have illumination variation and several face expressions. The ORL dataset contains 10 images for each of its 40 subjects. The UMIST face database consists of 574 images of 20 subjects. The images from ORL are all frontal faces, the images in UMIST are faces rotated up to 90 degrees. We resize the images in all datasets to have dimensions of $h = 112$ and $w = 92$.

Figure 7 shows examples of extracted regions for IntensityPatches for the used databases. It defines regions of the face such as eyes, eyebrows, chin, lips, cheek, hair, ear, and nose. It is worth to note that these regions do not optimally fit each image because they are not defined for a individual image. But these regions are unique for the entire training set. Figure 8 shows the squared clusters defined by the RegionPatches algorithm, the squares can be cropped in the rightmost and the bottommost clusters. The smaller the square side u the greater the number of clusters k, because more squares are needed to

Figure 7: The region of similar intensity. The features extracted by IntensityPatches are the mean intensity of these regions. From left to right: ORL, Yale, and UMIST datasets.

Figure 8: Square clusters defined by RegionPatches. From left to right: ORL and UMIST datasets. For ORL square side $u = 16$, number of clusters $k = 42$. For UMIST $u = 23$ adn $k = 20$.

fill the image. In the following subsections two experiments are described along with their results and discussion.

5.1. Choosing the number of clusters for IntensityPatches

In this experiments, the choice for the number k of extracted features is defined. We verify the mean accuracy for the three evaluated face datasets (ORL, UMIST, and Yale) using different number of clusters (2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, and 2048). The results for the mean accuracy for 10 holdouts are summarized in Figure 9. For the UMIST dataset, the mean accuracy has a maximum of 64 or more extracted feature. For the ORL and Yale datasets, the mean accuracy has a maximum with 512 or more extracted features. Therefore, we set $k = 512$ in the following experiment, since we are looking for a low value for the number of features but also gives a high accuracy.

5.2. Inserting new classes after training

This experiment evaluates the robustness of a face recognition system if new subjects are added to the system. Normally the classifier have to be retrained,

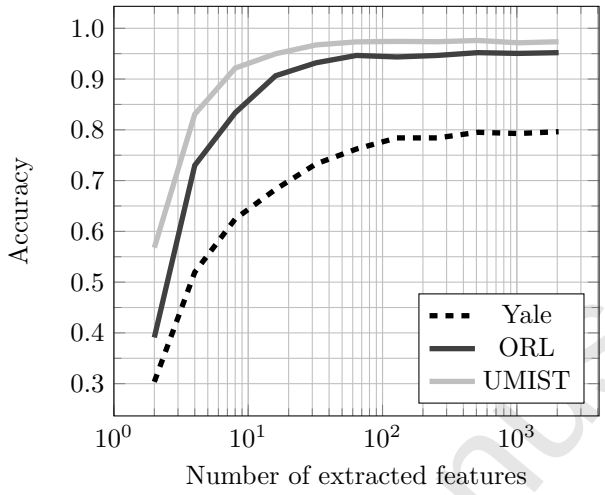

Figure 9: Accuracy rate of the proposed IntensityPatches per number of features for the three dataset: Yale, ORL, and UMIST.

however, what about the feature extraction method? We evaluate the system by using images from only a few classes to train the feature extraction method that projects every sample for all classes using the trained projections.

The face recognition accuracy is compared for 512 features extracted with the proposed method and the maximum number of features extracted with Eigenfaces. The results are described in Table 10. For each dataset, three experiment were performed using 1, 2, or 3 random selected classes. Mean accuracy and its standard deviation were calculated for 10 holdouts. Training projection with only a few classes has minor effects for IntensityPatches but not for Eigenfaces. The proposed method has accuracy from 11 to 23% greater than Eigenfaces for the Yale dataset, from 12 to 38% greater for the ORL dataset, and from 3 to 10% greater for the UMIST dataset.

The main advantage of the proposed method is that it does not have the same limitation for the number of extracted features as Eigenfaces. IntensityPatches is a suitable feature extraction method for face recognition, since it showed a relevant advantage in datasets with only a few samples. Also, it generates projection that is discriminant even for classes that have not being evaluated

Table 10: Results of the mean accuracy and (standard deviation) for ORL, YALE and UMIST datasets extracting features with PCA and IntensityPatches if only 1, 2 or 3 classes are used to generate data projection. N. C. stands for the number of classes.

	N.C.	PCA	IntensityPatches
YALE	1	52.80% (6.51%)	75.98% (2.88%)
	2	63.78% (3.90%)	76.34% (4.50%)
	3	67.56% (2.52%)	78.66% (3.95%)
ORL	1	55.95% (6.23%)	94.15% (1.31%)
	2	77.15% (5.15%)	95.05% (1.42%)
	3	82.45% (3.69%)	95.00% (1.37%)
UMIST	1	87.84% (2.59%)	97.21% (0.79%)
	2	92.37% (1.83%)	97.46% (0.99%)
	3	94.25% (1.53%)	97.35% (0.87%)

during training.

6. Conclusion

We proposed a framework for an unsupervised feature extraction. The basic idea is the clustering of the pixels so to diminish the multicollinearity issue. Two methods were proposed: IntensityPatches and RegionPatches. They were inspired in Waveletfaces and present similar property of being robust to Gaussian noise. RegionPatches is a direct extension of Waveletfaces, but it is more flexible and deal differently with color images. For image dataset were evaluated in the experiments. The results showed that RegionPatches have similar or greater accuracy compared to Waveletfaces. Also, RegionPatches and Waveletfaces were the methods that presented the highest accuracy for the face recognition task, mainly in the presence of noise.

Extra experiment with IntensityPatches for face recognition shows that this method demands about 500 extracted features for a maximum accuracy in the face recognition task. But it shows one of the highest accuracy for object recog-

nition in these experiments, for color or grayscale images, with or without noise. The proposed the framework is a powerful linear feature extraction method and opens the possibility for other combinations of clustering algorithms, distance metrics, and combination rules, to create new feature extraction techniques.

References

[1] J.-T. Chien, C.-C. Wu, Discriminant waveletfaces and nearest feature classifiers for face recognition, Pattern Analysis and Machine Intelligence, IEEE Transactions on 24 (12) (2002) 1644–1649.

[2] T. J. Hastie, R. J. Tibshirani, J. H. Friedman, The elements of statistical learning : data mining, inference, and prediction, Springer series in statistics, Springer, New York, 2009, autres impressions : 2011 (corr.), 2013 (7e corr.).

[3] Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (8) (2013) 1798–1828.

[4] J. P. Cunningham, Z. Ghahramani, Linear dimensionality reduction: Survey, insights, and generalizations, J. Mach. Learn. Res. 16 (1) (2015) 2859–2900.

[5] T. H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, Y. Ma, PCANet: A simple deep learning baseline for image classification?, IEEE Transactions on Image Processing 24 (12) (2015) 5017–5032.

[6] X. Chang, F. Nie, Y. Yang, C. Zhang, H. Huang, Convex sparse PCA for unsupervised feature learning, ACM Trans. Knowl. Discov. Data 11 (1) (2016) 3:1–3:16.

[7] A. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, A. Ng, On random weights and unsupervised feature learning, in: L. Getoor, T. Scheffer (Eds.), Proceedings of the 28th International Conference on Machine

Learning (ICML-11), ICML '11, ACM, New York, NY, USA, 2011, pp. 1089–1096.

[8] L. O. Jimenez-Rodriguez, E. Arzuaga-Cruz, M. Velez-Reyes, Unsupervised linear feature-extraction methods and their effects in the classification of high-dimensional data, IEEE Transactions on Geoscience and Remote Sensing 45 (2) (2007) 469–483.

[9] S. Theodoridis, K. Koutroumbas, Pattern Recognition, Fourth Edition, 4th Edition, Academic Press, 2008.

[10] Y. Fan, G. Huang, Y. Li, X. Wang, Z. Li, L. Jin, Development of PCA-based cluster quantile regression (PCA-CQR) framework for streamflow prediction: Application to the xiangxi river watershed, china, Applied Soft Computing (2016) –.

[11] V. V. Kamadi, A. R. Allam, S. M. Thummala, V. N. R. P., A computational intelligence technique for the effective diagnosis of diabetic patients using principal component analysis (PCA) and modified fuzzy SLIQ decision tree approach, Applied Soft Computing 49 (2016) 137 – 145.

[12] A. K. Jain, K. Nandakumar, A. Ross, 50 years of biometric research: Accomplishments, challenges, and opportunities, Pattern Recognition Letters 79 (2016) 80 – 105.

[13] T. B. A. de Carvalho, M. A. A. Sibaldo, I. R. Tsang, G. D. C. Cavalcanti, I. J. Tsang, J. Sijbers, Fractional eigenfaces, in: 2014 IEEE International Conference on Image Processing (ICIP), 2014, pp. 258–262. doi:10.1109/ICIP.2014.7025051.

[14] T. B. A. de Carvalho, A. M. Costa, M. A. A. Sibaldo, I. R. Tsang, G. D. C. Cavalcanti, Supervised fractional eigenfaces, in: 2015 IEEE International Conference on Image Processing (ICIP), 2015, pp. 552–555.

[15] R. C. Gonzalez, R. E. Woods, Digital Image Processing (3rd Edition), Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[16] T. B. A. de Carvalho, M. A. A. Sibaldo, I. R. Tsang, G. D. C. Cavalcanti, I. J. Tsang, J. Sijbers, Pixel clustering for face recognition, in: 5th Brazilian Conference on Intelligent Systems (BRACIS), 2016, pp. 121–126.

[17] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu, An efficient k-means clustering algorithm: analysis and implementation, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (7) (2002) 881–892.

[18] M. Turk, A. Pentland, Eigenfaces for recognition, J. Cognitive Neuroscience 3 (1) (1991) 71–86.

[19] D. R. Wilson, T. R. Martinez, Improved heterogeneous distance functions, J. Artif. Int. Res. 6 (1) (1997) 1–34.

- We propose a framework for defining feature extraction techniques
- This frameworks is an extension of feature extraction with Wavelets
- The proposed method is applied on face and object recognition
- Performance and accuracy compared to other methods