Practice 2. Image Recognition

A. Juan, R. Paredes, D. Picó and E. Vidal
Learning and Perception
Facultad de Informática
Course 2003-2004
Introduction

The goal of this practice is to know some software for the capture, processing and recognition of images. By the end of it, the student should be able to set his/her own face recognizer.

The necessary files are located in:

/labos/assignaturas/Fl/app/p2/bin : programs
/labos/assignaturas/Fl/app/p2/dat : data files
/labos/assignaturas/Fl/app/p2/doc : documentation
/labos/assignaturas/Fl/app/p2/awk : AWK programs

You should add the following lines to the file .bash_profile:

- export APP2=/labos/assignaturas/Fl/app/p2
- PATH=$PATH:$APP2/bin
Image recognition

• A very simple way of classifying an image is to compare it *pixel by pixel* with a set of images the classification of them is already known, and to classify it by the class of the closest image in this set.

• This technique has obvious limitations if we use the entire images for comparing.

• Some areas of the image are not interesting for classification.
Local Features

- The recognition systems must be robust and able to work under variations of the objects that are to be recognized. Possible variations are: viewpoint, illumination, scaling, overlapping, etc.

- The use of *local features* is a technique that helps to solve some of these problems.
Local Features: intuition

Reference objects:

Local representation of a triangle:

This local feature is only present in the triangle, therefore it is valid for distinguish the objects that have to be classified.
Local Features: intuition

Reference objects:

Local representation of the triangle using many local features:

We need these new local features to distinguish the triangle of the new object.
In the classification phase local features of the object are extracted and they are compared to the local features that were extracted from the reference objects. The new object is classified as belonging to the same class as the most similar reference object.
Local features. Complex objects

Reference objects:

Test objects:

The interesting local features for classifying are those that do not change with the time and help distinguish the object without confusion. Therefore, the hair, the clothings or the background are not adequate. We need techniques for feature extraction to find the interesting parts of the image.
Local features. Feature Selection

- The local standard deviation of each pixel in the image is analyzed.
- Those pixels with a standard deviation higher that some threshold are selected.
- We extract some windows centered in the selected pixels. These windows will be our local features.
In this example, a **skin detector** was also used. This detector analyzes the RGB components (i.e., the colors) to select the local features that should be extracted.
Local features. Feature selection. Example
Local features. Feature selection. Example
Local features. Conclusions

- The images are represented using many smaller images that define the extracted local features.

- In order to classify a new image, each of its local features is compared with all the local features that were extracted from the training set.

- The image is classified in the class that has the highest number of coincidences.
Practice 1. Previous steps

Create a temporal (local) directory with your user name:

$ cd /tmp; mkdir $USER; cd $USER

In this directory we can store the amount of data that the practice requires.

Copy the available database to your directory:

$ cp -R /labos/asignaturas/FI/app/p2/dat/base_caras

Enter this new directory and take a look to the images:

$ cd base_caras; xv */*.pgm

With the program xv we can open many images and see them sequentially pressing the space bar.
Description of the database ORL

In order to get familiar with the programs, we will use a database of face images that has already been acquired. This is the database that you have just copied to your temporary directory.

It has 10 images of 5 different persons.

The directories s1, s2, s3, s4 and s5 refer to the 5 different persons.

The images cara1.pgm, cara2.pgm, etc. in each directory correspond to the 10 different images for each person.

From now on, we will refer the persons as s1, s2, ..., s5, respectively.
**extractlf. Extraction of local features.**

*extractlf* is a program that extracts local features from a pgm image that reads from the standard input.

$ cat s1/cara1.pgm | extractlf > s1_1.vec

The following information is displayed:

112x92
427 vectores

The first line refers to the dimensions of the input image. The second line is the number of local feature vectors that have been obtained. In this example, we have represented the first face of subject s1 by means of 427 vectors.

As we will see later, extractlf can read a mask and apply it to the input image so that not all the pixels are considered as candidates for being a local feature. This mask is useful for segmenting (separating) the pixels that belong to the face from the rest of the scene.
extractlf. Extraction of local features.

Let’s take a look to the information that we have obtained with extractlf:

$ more s1_1.vec.

The first line of the file indicates the number of vectors that have been obtained. As we said above, they are 427 in this example.

Next, there are 427 lines of data, one for each feature vector. At the end of each line, there is a label indicating the class to which the vector belongs. Since we haven’t indicated any class, the label is the word unknown.

This final label should indicate the person to which the feature vector belongs. In this example, it should be “s1”.
extractlf. Extraction of local features. Options

Let's see the options of extractlf: $extractlf -h

-h : this help.
-l label : To label the vectors.
-w : Window size [11x11]
-r : Reduced [1]
-s : Sampling [2]
-d : Standard Deviation Threshold [16]
-m : Minimum Local Features [200]
-V : verbose.

Before trying the different options let's see the local features that have been obtained with the default values. We will use an awk program to do this:

$ awk -f $APP2/awk/miratodas.awk s1_1.vec.

If we make a $ ls, we’ll see that we have obtained a pgm file for each local feature.

Let's see the features:

$ xv ?.pgm ???.pgm ????.pgm -expand 8. With -expand we amplify the images 8 times. The space bar changes the image in xv.
extractlf. Associated class.

Change the name of the associated class using the opcion `-l name`

```bash
$ cat s1/cara1.pgm | extractlf -l s1 > s1_1.vec
```

See the resulting file:

```bash
$ more s1_1.vec.
```

The name should correspond to the name (or identifier) of the person.
extractlf. Size of the feature window.

The size of the window that defines the feature can be changed with the parameter “-w”, that has a default value of 11. For example:

```
$ cat s1/cara1.pgm | extractlf -w 21 -l prac2 > s1_1.vec
```

In this case, we would obtain local features of $21 \times 21$ pixels, i.e., vectors of 441 components.

See now the resulting local features:

```
$ rm *.pgm

$ awk -f $APP2/awk/miratodas.awk s1_1.vec

$ xv ?.pgm ???.pgm ????.pgm -expand 8
```

This size of window helps appreciate better where each of the local features comes from in the original image.
The threshold standard deviation that is used for selecting the local features can be modified with the parameter “-d”, that has a default value of 16.

```
$ cat s1/cara1.pgm | extractlf -d 20 -w 15 -l prac2 > s1_1.vec
```

242 vectores

With this threshold fixed to 20 there are only 242 pixels that pass this standard deviation. For a threshold of 16 we had 427 pixels.

The problem that we can have with this parameter is that if the value is too high, then too few features will be extracted. In order to avoid this, we can indicate to the program that there is a minimum number of features that we want to extract. This is done with the parameter “-m”.

```
$ cat s1/cara1.pgm | extractlf -m 300 -d 20 -w 15 -l prac2 > s1_1.vec.
```

This way, the program will reduce as much as necessary the standard deviation threshold so that at least 300 local features are obtained.
extractlf. Subsampling.

A very important problem of local features is that a very big amount of feature vectors that can be obtained and that makes the recognition algorithms very slow. In order to reduce the number of vectors, a subsampling of the original image can be done when extracting the feature vectors.

For example, if this subsampling is 2, then the candidate pixels will be those in every two columns and rows of the image.

This is done using the parameter “-s” (by default, s=2).

$ cat s1/cara1.pgm | extractlf -s 2 -l prac2 > s1_1.vec.$

Try now without subsampling, i.e., with “s=1”:

$ cat s1/cara1.pgm | extractlf -s 1 -l prac2 > s1_1.vec.$

1694 vectores

In this case, the number of vectors is too high. In order to reduce it, we
could increase the standard deviation threshold and/or increase the subsampling.
Another way of reducing the number of local features obtained is to reduce the original image. For example, if we reduce the image to a half, we will obtain approximately four times less local features.

In this case, we must take into account that we are losing resolution.

We must also take into account that the window sizes should also reflect this change. For example, if we were using $w = 15$ for the original image, then we should use $w = 7$ if we reduce the image to a half. This is necessary to obtain comparable results.

This scaling is done with the parameter “-r”.

$\texttt{cat s1/cara1.pgm | extractlf -w 21 -l prac2 > s1_1.vec.}$

$\texttt{cat s1/cara1.pgm | extractlf [-r 2] -w 11 -l prac2 > s1_1.vec.}$

Take a look to the local features for both cases. These scan approximately the same area of the image, but with “-r 2” the vectors have a lower dimension. Therefore, they occupy less space and the make the classification algorithms faster.
knn. K-Nearest Neighbors Classifier

**knn** is a program that implements the classification using the technique of the k-nearest neighbors for local features.

```
$ knn -h
```

- **t** training : Training file.
- **k** kvalue: K value for the knn algorithm.
- **d** : Print distance values.
- **a** : Print all neighbors information.

The option **-t training** is used for indicating the name of the file where we have the training vectors.

The option **-k kvalue** indicates the number of neighbors that are used for classification.

The option **-d** shows the resulting distances.

The option **-a** shows information for all the k neighbors.
knn. K-Nearest Neighbors Classifier

knn reads a file from the standard input that contains local features, and it returns from the standard output the class that has induced using the k-NN technique from the indicated training set. This output may contain more columns with some extra information if the options -a and -d are used.

An example:

$ cat s1/cara6.pgm | extractlf -w 11 | knn -t training.vec -k 3

Training: 13271 Vectores – 121 Dim – 5 Clases
...
s1

The first line contains information about the used training file, in this case “training.vec”. The 1st line simply displays the winning class, in this case “s1” (the right answer).
Test now the option `-a`:

\[
\text{\$ cat s1_1.vec | knn -t $APP2/dat/training.vec -k 3 -a}
\]

In this case, the class of the 3 more similar local features in the training set is displayed for each of the local features in the test image. At the end, the winning class “s1” is displayed.

Test now the option `-d`:

\[
\text{\$ cat s1_1.vec | knn -t $APP2/dat/training.vec -k 3 -d}
\]

In this case, the mean distance to the winning class is also shown.
Experiment design

In order to classify faces and check the performance of local features and the k-NN classifier, we should:

1. Decide what are the images that we will choose for defining the training set.

2. Decide what are the parameters that we will use for extracting the local features.

3. Decide what will be the value of $k$ that we will use in the classifier.
A practical example

1. We decide to pick the even images for the training set.

2. We decide to use “-w 11” and “-d 20” for the extraction of local features.

3. We decide to use “-k 3” for classification.

Learning phase (remember to delete the existing training.vec).

```bash
$ for i in 1 2 3 4 5; do for j in 2 4 6 8 10; do cat s$i/cara$j.pgm | extractlf -w 11 -d 20 -l s$i > > training.vec; done; done
```

All the local features of all the training images have been stored in the file training.vec. In this file there are local features of the 5 different classes.
A practical example. Classification

Classification phase:

```bash
$ for i in 1 2 3 4 5; do for j in 1 3 5 7 9; do echo s$i > > realclass; cat s$i/cara$j.pgm | extractlf -w 11 -d 20; done ; done | knn -t training.vec > > testclass
```

The file `realclass` contains the real class of each of the test images.

The file `testclass` contains the class that the classifier has inferred. This class is compared with the real class stored in the file `realclass` to obtain the error rate of the classifier.

Obtaining the error rate:

```bash
$ paste realclass testclass > resultado; confus resultado;
```

The program confun calculated the confusion matrix and the error rate (in percentage).
A practical example. Accelerating the classification

A problem that we can encounter is that the classification of each test image is too slow. This is due in part to the high number of local features that we extracted from each training image.

In order to decrease this number and accelerate the process we could for example choose a high standard deviation threshold.

However, we don’t know what’s the best value for this threshold. A trick can be to use a very high value (e.g., “-d 100”) and use the option “-m X” so that we know that we will have at least X local features.

For example, the training phase will be something such as:

```bash
$ for i in 1 2 3 4 5; do for j in 2 4 6 8 10; do cat s$i/cara$j.pgm | extractlf -w 11 -d 100 -m 150 -l s$i > > training.vec; done; done
```

Here, we would obtain a minimum of 150 local features, that will be those with a higher standard deviation.

We can also reduce the number of local features using the options for **subsampling** “-s” and for **scaling** “-r”. 
Exercise 1 (mandatory). Variation of parameters with ORL

You must design the following experiments of face recognition using the images in ORL:

- Only the images 1 and 2 for each person are selected for the training set. The images 3 to 10 are used for estimating the classification error.

- Feature windows must be used with the following sizes: 9, 11 and 15.

- Use a high standard deviation threshold and a minimum of 50, 150 and 250 features for each image.

- Scan the value of $k$ for the k-NN classifier for the values 1, 5 and 7.

For each experiment, obtain the associated confusion matrix with the following name: confus_win_min_k. Here, $win$ is the size of the feature window, $min$ is the minimum number of features to obtain and $k$ is the classifier’s parameter. Make also a table with the error rates that have been obtained for all the experiments.
Reconocimiento de caras online.

- Se trata de realizar un prototipo de reconocedor de caras.

- Las imágenes serán capturadas mediante una webcam.

- Los alumnos deberán primero capturar caras para formar un buen conjunto de aprendizaje.

- Posteriormente probarán el sistema para ver si los reconoce y diferencia con el menor error posible.
Capturar imágenes con la webcam.

Para capturar imágenes utilizando la webcam podemos usar el programa captura:

```
$ capture
```

captura abre una ventana donde aparece la imagen que se está adquiriendo mediante la webcam.

En el momento que pulsemos con el botón izquierdo del ratón dentro de dicha ventana, captura saca por la salida estandar un fichero pgm de la imagen capturada en niveles de gris.

Probar a capturar:

```
$ capture > imagen.pgm
```

```
$ xv imagen.pgm
```
Segmentación y detección de la cara.

Una vez adquirida la imagen sería deseable aislar y detectar la cara que se pueda encontrar en ella.

Este proceso de segmentación se realiza mediante el programa segmenta.

Dicho programa espera recibir por la entrada estandard una imagen en formato pgm para escribir por la salida estandard una imagen pgm con solamente la cara, si la hubiera:

$ cat ejemplo.pgm | segmenta > cara.pgm
# ejemplo.pgm se encuentra en $APP2/dat

Así mismo segmenta genera el fichero detec.ppm donde podemos comprobar en la imagen original donde se encuentra dicha detección.

$ xv detec.ppm
Segmentación y detección de la cara.

Este programa necesita que el usuario sea cooperativo y se coloque delante de la cámara de frente y sin ningún tipo de rotación de la misma.

$ captura | segmenta > cara.pgm

Comprobar la detección realizada: $ xv cara.pgm detec.pgm

Si la detección de la cara no es satisfactoria probaremos a subir o bajar el umbral de aceptación, parámetro -thr del segmentado. El valor por defecto de este parámetro es 0.5.

Valores más elevados significa ser más restrictivo a la hora de decidir si alguna parte de la imagen es una cara. Probar:

$ captura | segmenta -thr 0.6 | xv -

Valores inferiores del parámetro -thr significa ser más permisivo a la hora de decidir si alguna parte de la imagen es una cara. Probar por ejemplo:

$ captura | segmenta -thr 0.4 | xv -
Entrenamiento

Para realizar un conjunto de entrenamiento procederemos como sigue:

```
$ captura | segmenta | extractlf -w 11 -d 80 -m 50 -l roberto > > training.vec
```

En este caso hemos capturado una imagen y extraído características locales de la misma utilizando un ventana de $11 \times 11$, un umbral de varianza 80 y un mínimo de 50 características.

Todas las características locales extraídas son etiquetadas como pertenecientes a la clase roberto.

Todas estas características se han añadido al fichero training.vec.
Clasificación

Para clasificar una imagen capturada mediante la webcam procederemos como sigue:

$ captura | segmenta | extractlf -w 11 -d 80 -m 50 | knn -t training.vec -k 3

Se realiza la extracción de características locales mediante una ventana de 11, este parámetro debe ser igual que en fase de entrenamiento, un umbral de varianza 80, un mínimo de 50 características. Estos dos últimos parámetros podrían diferir de los utilizados en fase de entrenamiento.

Una vez extraídas las características locales se clasifica la imagen utilizando 3 vecinos más cercanos y el fichero de aprendizaje training.vec.
Ejercicio 2 (Voluntario). Reconocimiento de caras online.

Realizar un reconocedor de caras online con la webcam y hacer una demo al profesor de prácticas.

Para ello primero se deben de capturar unas cuantas imágenes por alumno (5 ó 6) para formar un fichero de aprendizaje training.vec.

Posteriormente lanzar el capturador para que capture 5 imágenes de cada alumno y las clasifique todas correctamente.

Para ello se tendrán que seleccionar correctamente los parámetros: tamaño de ventada, mínimo número de características, submuestreo y/o reescalado, valor de $k$ del clasificador etc...

Además hay que prestar especial atención a que el sistema reconozca lo más rápido posible pero sin perder eficiencia. Prestar atención pues a los parámetros de submuestreo, escalado y número mínimo de características.
Ejercicio 3 (Voluntario). El problema de Verificación.

Este ejercicio pretende realizar un sistema de control de acceso o verificación.

El sistema debe de conceder el acceso a los alumnos introducidos en el training, y denegarlo a cualquier persona que no esté en dicho fichero.

Para ello se debe usar la opción “-d” del clasificador por k-vecinos. Este valor de distancia se supone que debe de ser **pequeño** si la persona a reconocer está en el conjunto de aprendizaje. Así mismo, este valor de distancia debe de ser **grande** cuando se clasifiquen personas que no estaban en el conjunto de aprendizaje.

Eligiendo un **umbral apropiado** de distancia se podrá rechazar o aceptar el acceso de la persona a reconocer.

Para aprobar el ejercicio el sistema debe de aceptar a los alumnos pertenecientes al training y rechazar al profesor.