CGDI - Project Report

Remi Coudert and Loïs Paulin

May 2017

Abstract

We present here our work on the Computer Graphics and Digital Images project which consisted in the creation of an image recognition system learning a set of 70 classes from an database of 15 pictures per class. Our classifier is a KNN algorithm working on features derived from Hu moments. We obtained satisfying results on our test protocol on close shapes for the human eye such some devices class, bricks and some simple shapes such as children. But we had big drawbacks on classes with more details such as guitars, keys and butterflies.

Contents

1 Introduction

Computer Vision is a very challenging part of image analysis in which many problems have to be solved in order to determine what is the image representing. In real life situations, the first problem faced is to be able to determine what is part of the background and is part of the object or what are the interest points in the picture. In this project, we consider that this part is already done and that all we have to do left is the analysis of the object shape in order to classify it.

The main challenge in this project is to be able to characterise objects which can come in any rotation or size, or even with some noise on it. This require us to find features which are scale and rotate invariant, and which still allow us to discriminate close classes such all the round and square shaped classes and to detect the obvious characteristics of classes such as the bugs antenna and legs.

As we present it in Section 2 our approach was to use KNN[2] a classifier that compares the test image to all the training samples and select the most frequent class in the K Nearest Neighbours. We decided to use Hu moments [3] and the ratio between the area and the square of the perimeter of the shape as features. All those features respect the condition of being scale and rotate invariant.

2 Chosen Features

2.1 Hu Moments

Hu moments[3] are a set of 7 image features derived from image moments.

if $f : \mathbb{R}^2 \to \mathbb{R}$ is the (continuous) function mapping pixel positions to their value, an image moment is defined as

$$M_{pq} = \int x^p y^q f(x, y)\ dxdy$$

and for the discrete verrsion, where I is the discrete version of f (i.e. an array of pixel values)

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

Let's define $(\overline{x}, \overline{y})$ to be the centroid as follows :

$$\overline{x} = \frac{M_{10}}{M_{00}}$$
$$\overline{y} = \frac{M_{01}}{M_{00}}$$

and define (only the discrete case)

$\mu_{pq} = \sum_x \sum_y (x - \overline{x})^p (y - \overline{y})^q f(x, y))$

μ_{pq} is translation invariant

from here let

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{(1+\frac{i+j}{2})}}$$

η_{ij} is scale (and translation) invariant.

Hu moments are then defines as mathematical combinations of different n_{ij}.

For example the first Hu moment is

$$I_1 = \eta_{20} + \eta_{02}$$

All the hu moments are rotation (and scale, translation) invariants.

2.2 Area Over Squared Perimeter

2.2.1 Feature Properties

With this feature, we tried to get a measure of how folded the contour of our picture was. Since for a given area, the higher the perimeter is, the more folded our contour is.

In order to compute this feature, we implemented estimators for both area and perimeter which are convergent. And thus are rotate invariant with a sufficient sampling precision. Moreover, by definition, when scaling by a factor n, the area is multiplied by n^2 and the perimeter by n. Thus we have:

$$\frac{A(S(X,n))}{P(S(X,n))^2} = \frac{A(X) * n^2}{P(X)^2 * n^2} = \frac{A(X)}{P(X)^2}$$

Thus this feature is scale invariant.

2.2.2 Implementation Details

Our estimator for the area is simply the number of pixel in our shape but our perimeter estimator involves more complex objects such as DSS.

Data: S : Shape
Result: P : Perimeter
DSS_Set = DSS_Cover(S);
P = 0;
forall *d in DSS_Set* **do**
  P += Distance(d.begin, d.end);
end

Algorithm 1: Perimeter Estimator

We used the DSS structure contained in the DGtal librairy to implement the DSS_Cover function which computes a set of non extendable DSSs which covers the whole contour of our shape.

3 unimplemented features

3.1 SIFT descriptor

The SIFT[4] (Scale-invariant Feature Transform) algorithm, is a very famous algorithm used in shape indexing. This algorithm computes 'SIFT descriptors' that can be seen as local characterisation of the images.

We could add the SIFT image descriptors to our feature vectors to have more precise image descriptions ans thus a better classification.

3.2 SURF descriptor

The SURF[1] (Speeded Up Robust Features) algorithm was presented (in 2006) as an improvement of the SIFT algorithm in terms of computational speed and robustness.

As for SIFT, we could add these image descriptors in our features vectors to get better results.

4 C++ Implementation

4.1 General Layout

We defined the following classes :

- **Image** : This is the main class, representing an image where we compute features and read images from files.
- **Pixel** : This class simply represents a pixel and is used for Image
- **ImageClass** : This class represent an image class, it holds the value of its name and its feature vector. This class overrides the '$<$' operator for its use in **std::sort**, an imageClass is 'less than' another if its distance to the current evaluating class is smaller.
- **KNearestNeighbours** : This class holds the value of k (3 by default) and the functions used to compute the KNN algorithm.

The feature vectors are computed insinde **ImageClass**'s constructor. We then use the KNN[2] algorithm.

For simplicity and speed, we already computed feature vectors, they are stored in the file 'classes.csv'.

4.2 Implemented functions unused

Here's a list of function we implemented but that are not used in the current code :

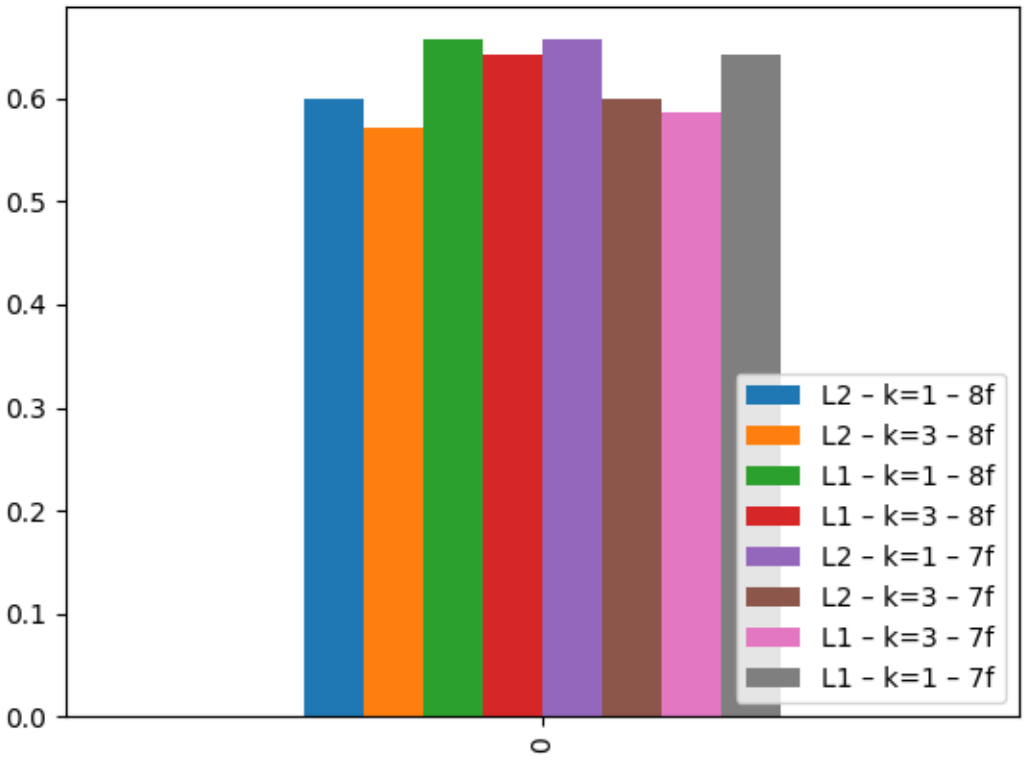
- **closing** : We implemented the closing algorithm seen in class to be able to correctly classify some images such as the cup (which can have two connected components). We dropped this functionality because it did not change much the result while beging computationaly intensive. Perhaps with some more experimentation with the parameters of the closing (such as the size of the closing element) we could have done something better.
- **standardization** : We implemented this function as an alternative to normalization, it worked well with only hu moments as features. The implementation consists in finding the mean μ and variance σ^2 of the vector and for each x in the vector do $x = \frac{(x-\mu)}{\sigma^2}$

5 Experimental Protocol and Results

The given database contains 70 different classes, each holding 15 different examples. We chose to learn on the first 14 and test on the 15th one for each class. We then simply ran our knn algorithm on each 15th class to see whether it was correctly classified, each time changing some parameters. The 3 different parameters were

1. k as the argument for the knn algorithm

Figure 1: Correct classification frequency against different testing parameters



2. The choice of Euclidian (L2 norm) or Manhattan (L1 norm) distance
3. The use (or not) of the last feature (i.e. use only the 7 hu moments or the full 8 moments)

It yielded the results shown in Figure 1.

These results can be seen in the files $l1, 2\_k1, 3\_7, 8f$ along with knn votes and classification result for each image.

6 Conclusion

Our results show that our choice our features, even if not complete, is somewhat relevant. We were able to correctly classify about 60% of the images we tested, some classes being extremely well classified some being quite badly classified.

Finding good image features is a difficult task and depends a lot on the data. Finding and using the right classifier and use it well is a challenge as

well. We chose to use KNN for its simplicity but we are aware of its limitations, such as its high sensitivity to data range and dimension of the space and its relative slowness. For instance we could have used a deep learning classifier or a decision tree classifier, which would most certainly yields better results, but with the drawback of implementation complexity.

Data preprocessing is an aspect that could be rethought as well, a lot of choices are possible such as smoothing input images or using different kinds of data normalisation.

Cross-validation could have been beneficial as well. Indeed, if we had more data, we could have split it in 3 sets (training, testing, and validation) to be able to learn parameters such as the best value of k for KNN or to learn a good metric to use since l1 and l2 norms might not be optimal distance functions. Dimensionality reduction could have been done as well.

To sum up, our implementation permits to decently classify images even though we could have had a more complete recognition pipeline.

References

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *Computer vision–ECCV 2006* (2006), pp. 404–417.

[2] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[3] Ming-Kuei Hu. "Visual pattern recognition by moment invariants". In: *IRE transactions on information theory* 8.2 (1962), pp. 179–187.

[4] D.G. Lowe. *Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image*. US Patent 6,711,293. Mar. 2004. URL: `http://www.google.fr/patents/US6711293`.