Handwritten recognition

Image analysis project

VIVIER Thibault

Note: This report detail the pipeline of the "main.ipynb" file, which uses all the additional python scripts of the project.

Database construction

Raw data

The raw data are pictures of pages containing lines of the same character, for example the letter "a". To build the database, it is required to extract the contours of the shapes in those images using OpenCV. Then, write a new image file for each of the character identified with the contours. For each letter, there are now the following architecture:

```
data
  |_ characters
         |_ a
            |_ char_1.png
            |_ char_2.png
            |_ ...
         |_ b
         |_ ...
```

Figure 1

However, this automatic process of character extraction using OpenCV is not enough to have the best data as possible. There are a lot of false positive: shapes extracted and identified as characters, which are not. It is important to clean manually the database and remove the bad shapes from the database to avoid mislabeling in the next steps.

Data augmentation

Now, there is a full database. But due to the low level of confidence into the OpenCV character extractor, and the need of a large volume of data. It is important to increase the volume of data available. To do that, it is possible to artificially increase this data volume by applying simple transformations like: zoom, rotation, translation, image crop, or apply a blur on the image.

Applying each of those transformations to all the characters in the database multiply the data volume by 4, 5 times.

Labeling

Then, to build a classifier, it is important to label the data. Due to the specific architecture detailed in the *Figure 1*, it is easy to label the data using the folders' name. The final dataset is composed of two fields:

- File path: the file path of the image to be able to read its content.
- Label: the label of the associated letter (a, b, c, …)

Note: you can find sample of original and sampled letter in the Git repository.

Features extraction

The next step is to extract features from the images of the database. To do that, feature descriptor are needed. They are mathematical functions that map raw data into feature vector. Direct learning from image can be inefficient, noise sensitive and involve computational issues. Instead of training classifiers directly on images and pixels value, using feature vector is better. Those feature vectors contains an image representation that can be used to discriminate and classify the image among others.

There are many different feature descriptors. Here are the main three tested:

- HOG
- HU – HU Moments
- GEOMETRIC

HOG

It works by calculating the distribution (histogram) of gradient orientations in localized portions of an image. The image is divided into small connected regions called cells, and for each cell, a histogram of gradient directions is computed. These histograms are then normalized across larger spatial regions (blocks) to improve robustness against variations in lighting and shadow.

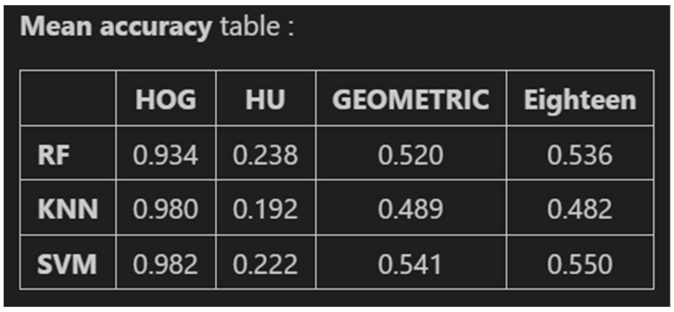
HU

Hu Moments are a set of seven numerical values derived from central moments that are invariant to image translation, scale, and rotation. They are based on the theory of algebraic invariants and are used to describe the shape of objects in an image. These moments capture global information about the object's shape, such as symmetry, elongation, and complexity.

GEOMETRIC

Those descriptors focus on the geometric properties of objects within an image, such as area, perimeter, compactness, eccentricity, and aspect ratio. These features are intuitive and directly related to the physical structure of the objects. For example, compactness measures how "tightly" an object is packed, while eccentricity describes how elongated it is. Geometric descriptors are simple to compute and interpret.

After testing the three features descriptors with the classifiers, only HOG have relevant results. It will be the one used to train the main classifier.



Mean accuracy table :

	HOG	HU	GEOMETRIC	Eighteen
RF	0.934	0.238	0.520	0.536
KNN	0.980	0.192	0.489	0.482
SVM	0.982	0.222	0.541	0.550

Figure 2

Classification

To perform the classification, I decided to test three different classifiers:

- SVM (Support Vector Machine): to find the optimal hyperplane to separate classes in the feature space.

- Random Forest: builds multiple decision trees during training. Each tree is trained on a random subset of data and features, and the final prediction is made by majority voting.

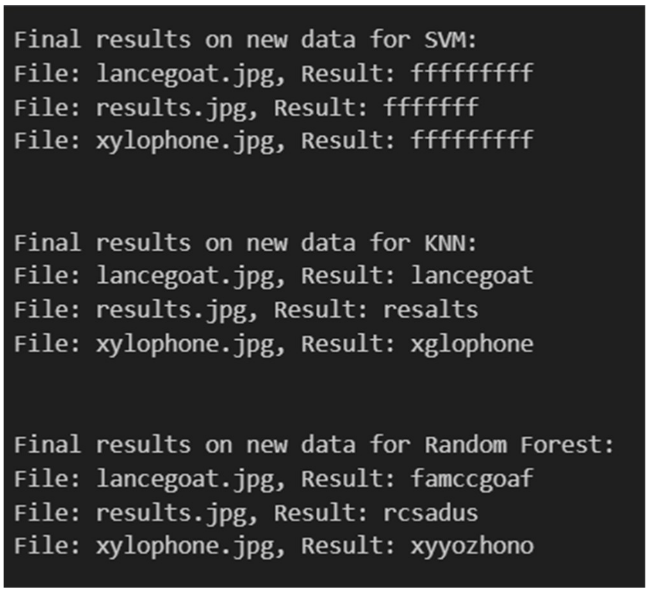
- K-NN (K-Nearest Neighbors): classifies new data points based on the majority class of their k nearest neighbors in the feature space, using a distance metric like the Euclidian's distance.

Regarding the mean accuracy table (Figure 2), KNN and SVM classifier have similar results, and Random Forest have a little lower mean accuracy compare to the others.

Testing the three algorithms is a good idea to find the most performant one on real data.

This mean accuracy table is based on characters recognition of a sample of the database, using the split into train and test sample to assess our classifiers. However, the real data are not characters but words. All the difficulty is to process the character recognition using OpenCV (like in the database part), and then classify the extracted characters.

Due to this particularity, results are not such good as it could be expected.



```
Final results on new data for SVM:
File: lancegoat.jpg, Result: fffffffff
File: results.jpg, Result: fffffff
File: xylophone.jpg, Result: fffffffff


Final results on new data for KNN:
File: lancegoat.jpg, Result: lancegoat
File: results.jpg, Result: resalts
File: xylophone.jpg, Result: xglophone


Final results on new data for Random Forest:
File: lancegoat.jpg, Result: famccgoaf
File: results.jpg, Result: rcsadus
File: xylophone.jpg, Result: xyyozhono
```

Figure 3

As shown in the Figure 3, results are bad. The best results are for the K-NN classifier. We can suppose that, with a larger volume of data, the efficacy of the results would be probably better, even if the main issue is probably coming from the OpenCV character extraction. Regarding the SVM model, the results are surprising but I do not have any explanations about it.