Master MINDS - Image Analysis

Handwritten Recognition

Student : Baptiste TRON

Github repository with the code :

https://github.com/baptwebsite/handwritten_recognition.git

1. Data Acquisition and Dataset Construction

In order to have a complete dataset, I manually wrote each letter of the alphabet 21 times on paper in order to capture the natural variability of handwriting. A photograph was taken for each letter, producing 26 source files (from a.jpg to z.jpg).
After this I developed an image extraction script to transform these photos into data that can be exploited by a model. The processing pipeline for each sample is as follows:

- Extraction: locating and cutting out each instance of the letter.
- Normalization: each letter is centered in a square of 64x64 pixels.
- Scaling: to ensure consistency, the letter is resized to occupy precisely 75% of the surface of the square.
- Binarization: conversion into black and white (white background, black line) to eliminate variations in brightness and texture of the paper

At the end of the extraction, I obtained a theoretical total of 546 images (26 text * 21 samples). I performed a visual inspection of each extracted image and I manually removed samples with segmentation defects (cut letters, holes in the plot, or significant noise).

fig 1. good quality sample

fig 2. poor quality sample

Then instead of having to handwrite other letters in addition, I perform some data augmentation with 3 ways :

- random rotation : the script applies a rotation between -15° and +15°.
- random zoom : a size variation between 0.9x and 1.1x.
- translation : a slight horizontal and vertical shift from -3 to +3 pixels.

fig 3. original letter

fig 4. augmented letter

Therefore each letter is augmented by 10. Additionally, this script creates the final dataset folder, containing the subfolder of each letter and their content.

Finally I use a script to create a csv file with the path and the label of each sample, that I will use for training and testing models. The final dataset is composed of 4906 samples.

2. Feature Extraction Strategy

Initially my input is resized in a 32x32 square in order to smooth the imperfections. So the model focuses on the fundamental structure of the letter (bars, loops, angles) rather than high frequency noise.

In order to have as much information as possible, I decided to combine the 3 possibilities of extracting features (Hu moments, HOG and geometric).

Hu moments are useful for detecting the shape and the global aspect of a letter. It doesn't matter if the letter is a bit rotated, enlarged or reduced, the final vector will be the same. They capture the general distribution of ink. It is very useful to distinguish globally different letters, like an "O" (hole in the middle, mass distributed on the edges) from an "X" (mass concentrated on two diagonals). The vector is composed of 7 values.

HOG analyzes the distribution of gradient directions (the contours) in localized areas of the image. HOG is better for capturing the local structure, it identifies precisely the arrangement of strokes. For example, it detects the horizontal bar of an "A" or the curves of an "S". It gives a vector of 144 features. The extraction of the 144 HOG features is based on a rigorous spatial decomposition adapted to the resolution of 32x32 pixels. The image is first divided into a grid of 16 cells (4x4), where each cell measures 8x8 pixels. To ensure the robustness of the extraction, the *skimage* library groups these cells into blocks of 2x2. In order not to exceed the dimensions of the image, the number of possible positions for these blocks is determined by the formula (Number of cells - Block size) + 1. In this configuration, the sliding block can only occupy 2 positions horizontally and 2 positions vertically, thus generating a total of 4 effective blocks. Each block containing 4 cells, and each cell being characterized by a histogram of 9 orientations, we obtain a vector of 36 values per block (4x9). The final concatenation of these 4 blocks therefore precisely produces a vector of 144 descriptors (4x36), offering a stable and compact structural signature of the handwritten character.

The geometric descriptor goes through an approach on proportions, using a ratio and is able to identify circularity. An "I" or an "L" are immediately distinguished from a "W" or a "M" by their width/height ratio. It gives a vector of 11 values.

The combination of all the descriptors gives a final vector of 162 values.

The robustness of the model relies on an absolute consistency of the processing pipeline between the training and inference phases. To ensure this stability, feature extraction is centralized in a single function, *extract_advanced_features(img)*, systematically applied to each sample. This approach eliminates any risk of technical

discrepancy (processing bias) and ensures that the classifier evaluates test data according to a repository of descriptors strictly identical to the one learned during learning.

3. Model Training

The pipeline starts by splitting the data with 80% for the train and 20% for the test. Moreover the *stratify=y* argument is crucial, it ensures that the proportion of each letter is identical in the training set and test set, thus avoiding evaluation bias.

The next step is the standardization of data, carried out via the StandardScaler, which is an essential pre-processing step to harmonize the 162 characteristics of the input vector. Given that the dataset combines descriptors of varied natures their scales of magnitude differ drastically. Without this standardization, the characteristics with the highest numerical values would dominate learning, thus biasing the construction of decision trees in the Random Forest. By centering the data (mean to 0) and reducing it (standard deviation to 1), the StandardScaler puts all descriptors on a mathematical equal footing.

Then we define the Random Forest model with these parameters :

- n_estimators = 500
- max_depth = 25
- class_weight = “balanced”

The last parameter asks the model to automatically give more importance to the underrepresented letters when calculating the error, ensuring a balanced accuracy over the entire alphabet.

Finally the model and the scalar are saved in the model folder using the *joblib* library.

4. Text Recognition and Evaluation

I implemented a simple test script on 982 samples, and I got an overall accuracy of 97%. Class analysis reveals a specific weakness for the letter 'h' with a recall score of 0.60, suggesting recurrent morphological confusion with other characters (such as the 'n' or the 'b'). Conversely, the letters 'a', 'i', 'm', 'p', 'r', and 's' reach a perfect score of 1.00, demonstrating the effectiveness of HOG descriptors in capturing the angular structures and unique loops of these characters. The results are saved in classification_report.txt.

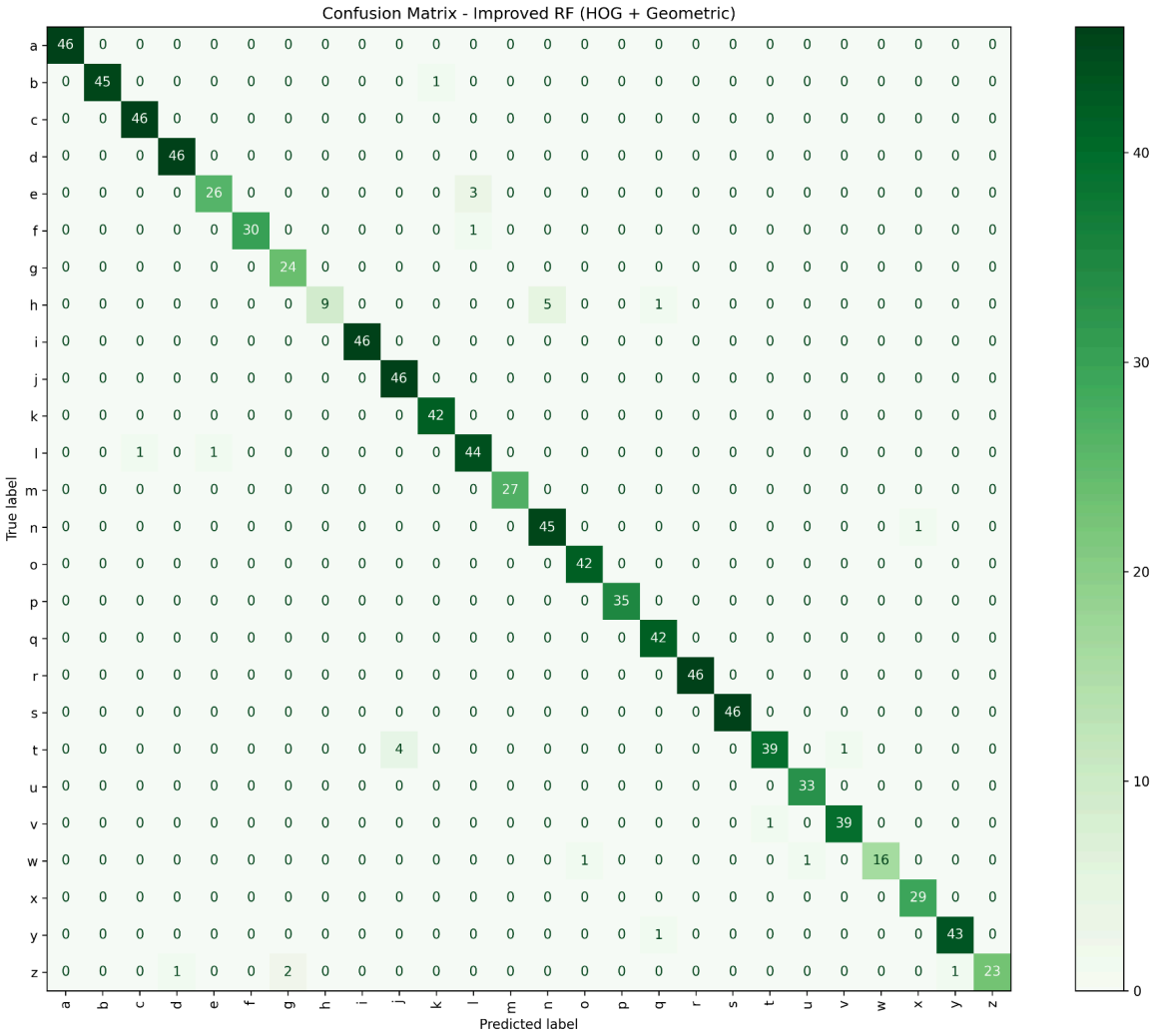

fig 5. confusion matrix

Despite an excellent precision on the isolated characters, the reconstruction of complete words was the most complex technical challenge of this project. The main difficulty lies in the segmentation step: the use of external contours to isolate the

letters proved extremely sensitive to writing variations. In several cases, letters like the ‘i’ or the ‘j’ were fragmented into two distinct entities (the body and the point), while the cursive script or overly close letters were merged into a single block, making the extraction of HOG characteristics incoherent. These segmentation errors create a discrepancy between the theoretical performance of the classifier (97%) and the practical reliability of the reading system, highlighting the fact that in an OCR (Optical Character Recognition) pipeline, the quality of preprocessing and cutting is just as decisive for the final result as the power of the classification model itself

5. Analysis and Discussion

Prior to implementation, my research into handwritten character recognition indicated that classical classifiers such as k-NN, SVM, and Random Forest often yield comparable performance levels. Consequently, I chose to focus exclusively on the Random Forest model, as its architecture was the most familiar to me, allowing for a more in-depth optimization of the feature extraction pipeline.

The efficiency of the model relies on a hybrid approach to feature extraction, combining global shape descriptors and local texture information. The use of HOG (Histogram of Oriented Gradients) is fundamental here because it allows capturing the fundamental structure of letters by analyzing the distribution of the directions of the contours, which makes the model robust to slight distortions of writing. To overcome the lack of global morphological information of the HOG, the integration of Hu Moments brings a crucial invariance in the face of scale changes, translation and rotation. Finally, the addition of geometric (circularity, aspect ratio, area) and statistical descriptors (gradients and curvature) allows to remove ambiguities between structurally close characters but morphologically distinct. This synergy of 162 features offers a rich and discriminating digital signature, much more efficient and less sensitive to noise than the raw use of image pixels.

The error analysis reveals that while the model is highly robust for most characters, specific phonetic and structural similarities lead to predictable misclassifications. The most prominent issue is the low recall for the letter ‘h’ (0.60), which is frequently confused with ‘n’ or ‘b’. This is primarily due to the morphological ambiguity of the "ascender" (the vertical stroke): if the stroke is too short, the HOG descriptors and geometric aspect ratios fail to distinguish the 'h' from an ‘n’. Similarly, letters like ‘e’ and ‘t’ show slight drops in performance, often attributed to "noise" in the handwriting, such as an incomplete loop. Beyond structural resemblance, these errors are exacerbated by the dataset's class imbalance. Indeed, of the 21 letters ‘h’ written on the paper initially. Only 7 were correctly extracted. This therefore represents 3 times less data than some letters. This is interesting because it highlights the importance of having a complete dataset.