

Pattern Recognition and Neural Networks

Writer Identification System

Mohamed Shawky Zaky AbdelAal Sabae

Section:2,BN:15

mohamed.sabae99@eng-st.cu.edu.eg

Mohamed Ahmed Mohamed Ahmed

Section:2,BN:10

mohamed.dardir98@eng-st.cu.edu.eg

Remonda Talaat Eskarous

Section:1,BN:19

remonda.bastawres99@eng-st.cu.edu.eg

Mohamed Ramzy Helmy Ibrahim

Section:2,BN:13

mohamed.ibrahim98@eng-st.cu.edu.eg

Abstract—In this work, we present our work pipeline for writer identification from handwriting. Our system uses *LBP* texture descriptors along with *SVM* classifier on form-level of *IAM Handwriting Database*. The system, also, uses form preprocessing techniques to extract separate lines from a single form. The combination of these techniques enables us to achieve up to 93.5% accuracy on the complete dataset and an accuracy between 98.9% to 100% with sampled test. The system can maintain fast execution, while achieving such high accuracy. Furthermore, we compare our approach to other different approaches to illustrate its advantages.

I. INTRODUCTION

Writer identification from handwriting is a challenging problem. Historically, experts with domain knowledge were required to tackle such tricky problem. However, with the rise of *AI* and *machine learning* techniques, systems can be built to solve the handwriting identification problem. In *machine learning* systems, the choice of good features and robust classifiers is the core challenge. For such problem, domain-based features can be used such as *codebooks* and *grapheme signatures*. However, with the evolution of general purpose texture descriptors like *local binary pattern* and *local phase quantization*, it turns out that these features can perform even better in most cases. For this reason, we decided to adopt the fast and well-known *local binary pattern* texture descriptor, inspired by [1]. We, also, considered multiple classifiers and decided on *support vector machine* classifier, which is the best in our case.

II. APPROACH

In this section , we discuss the overall system pipeline. The exact details of each module is discussed in subsequent sections.

Our system can be divided into 3 main modules, shown as follows :

- **Preprocessor** : this module takes the *complete form* image as an input, performs *denoise* and *extracts the written parts* only. Then, it *segment out* each written lines in the document.
- **Feature Extractor** : this module takes each line extracted by the *preprocessor* and perform *local binary*

pattern texture descriptor on it and calculates the *normalized LBP histogram*.

- **Classifier** : this module contains the training and inference of *SVM* classifier. The training is done on each line as *separate train sample*, while inference is done on each line separately and then a *majority vote* is taken.

III. PREPROCESSING MODULE

A. Form Clipping

In *Form Clipping*, we remove the upper and lower part of the image to take the hand written part only. This is done by extracting the edges of the image using *Canny edge detection* the detect the lines in the edge image using **OpenCV's** *houghlineP* function. After that, we only consider the horizontal line that are not at the beginning of the image, sort them, then take the part of the image that lies inside the first and last detected horizontal line. sample results are shown in figure 1 and figure 2.

B. Line Segmentation

As extracting features from lines is better than from the whole document, and classifying on each line helps to take a majority vote, We implemented a vectorized module to segment the document into lines using row histogram as shown in 3. The docs are scanned vertically, so as shown in 4, Detecting minima on a row histogram can get the boundaries to cut the lines on, as they are almost horizontal. We did not use the concept of local minima in Detecting peaks as there is a lot of local minima that are not considered as line boundaries. We considered that the line boundary exists when there is an abrupt change in the histogram that makes it jump over 1/5 of its mean. Some lines can be splitted into two lines, one of them has a small height. This happens because of different hand-writings. To tackle this problem, we filtered all lines that have black-pixel count less than 1/6 of the mean of this count on all extracted lines.

IV. FEATURE EXTRACTION MODULE

The feature extraction module includes **local binary pattern** (*LBP*) texture descriptor. Other feature extractor were considered, as well. However, after many experiments, we found out *LBP* texture descriptor performs the best

TEXT III – FRENCH

La France a remporté la deuxième Coupe du monde de son histoire après le sacre de 1998, en battant la Croatie 4 à 2, en finale du Mondial-2018, dimanche à Moscou, dans un match fou. Ce match a été suivi par des centaines de milliers de fans dans les rues et les bars de Paris. Et les Champs-Élysées ont été envahis avant même le coup de sifflet final. Didier Deschamps, 49 ans, capitaine des champions du monde 1998, est devenu le troisième homme de l'histoire du foot à remporter la Coupe du monde comme joueur puis comme sélectionneur, après le Brésilien Mario Zagallo et l'Allemand Franz Beckenbauer.

La France a remporté la deuxième Coupe du monde de son histoire après le sacre de 1998, en battant la Croatie 4 à 2, en finale du Mondial - 2018, dimanche à Moscou, dans un match fou. Ce match a été suivi par des centaines de milliers de fans dans les rues et les bars de Paris. Didier Deschamps, 49 ans, capitaine des champions du monde 1998, est devenu le troisième homme de l'histoire du foot à remporter la Coupe du monde comme joueur puis comme

Fig. 1: Image before clipping

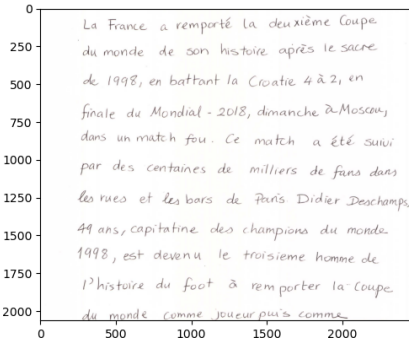


Fig. 2: Image after clipping

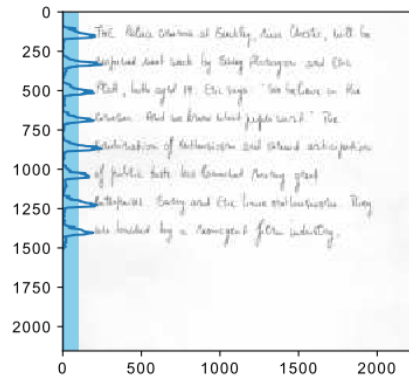


Fig. 3: Row Histogram on Document to segment lines

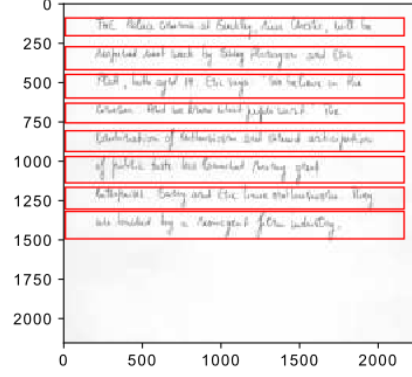


Fig. 4: Resulting Bounding Boxes generated using Histogram

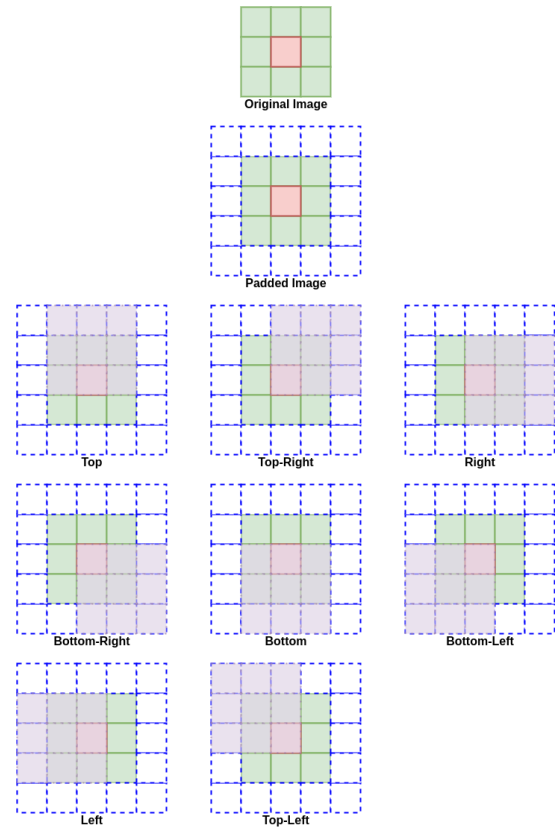


Fig. 5: Illustration of our vectorized implementation of LBP texture descriptor.

in our case. The other feature extractors are discussed in later sections. Although *LBP* features offer high accuracy, **skimage** implementation is not vectorized and heavily depends on *loops*. The extraction of *LBP* features for a single form can take up to 0.5 second. That's why, we come up with a vectorized implementation that speeds up processing up to 0.02 second per form.

Figure 5 shows the **vectorized implementation** on a simple 3X3 image matrix. The implementation goes as

follows :

- 1) The input image is *padded* with *zeros* from all directions with the *LBP* radius size.
- 2) An *LBP* map with the same dimensions as the input image is initialized with zeros.
- 3) The whole original image is displaced to the top and compared to the padded image. Using this method, we compare all pixels in parallel instead of looping over each pixel.
- 4) The resultant map is, then, multiplied by 2 raised to the power of *number of iteration*, then added to the *LBP* map. **Note that**, *number of iteration* ranges from 0 to 7, as only 8 directions are considered to speed up the implementation.
- 5) Steps 3 and 4 are repeated for *top-right*, *right*, *bottom-right*, *bottom*, *bottom-left*, *left* and *top-left* directions.
- 6) A histogram is calculated for the output *LBP* map with 256 bins. The histogram is normalized by its mean, according to the original *LBP* implementation.

V. CLASSIFICATION MODULE

For this work, 5 classifiers are considered for experimentation, which are **Support Vector Machines (SVM)**, **k-Nearest Neighbors (KNN)**, **Random Forests (RF)**, **Logistic Regression (LR)** and **Naive Bayes (NB)**. However, for our system, we choose *SVM*, as it's the best performing classifier based on accuracy. At the same time, due to the small size of test cases, it is slightly slower than other classifiers like *KNNs*, which can be acceptable.

The *classification module* can have 3 modes based on our implementation :

- 1) **Complete train** : where the system is trained on complete data of authors with a specific number of forms. This is used as an initial experiment with the dataset and helped us determine the initial set of features.
- 2) **Sampled train** : this mode mimics the test environment. It reads the whole dataset and generate a specific number of random test cases of 3 writers, 2 samples per writer and only *one* test sample. This enables us to tune our parameters and choose our final techniques, which are, then, implemented and refined.
- 3) **Test** : *main pipeline* for the provided test environment, where the system takes the test directory and generates the output labels and time in text files.

VI. PERFORMANCE ANALYSIS

Classifier	100 test samples	1000 test samples
Support Vector Machine	100%	99.7%
k-Nearest Neighbors	99%	99.4%
Random Forest	99%	99.6%
Logistic Regression	100%	99.5%
Naive Bayes	100%	98.9%

TABLE I: Comparison between accuracies of different classifiers using LBP feature and different sample size.

Component	Execution Time
Form Clipping	0.09
Line Segmentation	0.06
LBP Features	0.12
Classifier Training	0.01
Complete Test Case	2.00

TABLE II: Average execution time of different system components (measured in seconds).

A. Accuracy Analysis

As mentioned before, different approaches are considered for both feature extraction and classification.

We start by examining different feature extractors. According to [1], the most promising texture descriptors are *LBP*, *LPQ* and *GLCM*. We tried these texture descriptors, however we find that *LBP* offers the most accurate and fast results, so we considered it for further experimentation. *LBP* offers 99% average accuracy with all classifiers in **sampled train mode**, however *LPQ* and *GLCM* offers around 88% and 81%, respectively.

Regarding the classifiers, table I shows the accuracy of different classifiers using *LBP* features on 100 and 1000 random test cases. We can see that *SVM* and *RF* offer comparable results, however we choose *SVM*, as it is more robust to *preprocessor* failures and offers more consistent accuracy.

B. Time Analysis

We try to maintain our system accuracy within reasonable execution time. Different components are implemented and optimized for time. The execution times for different components are shown in table II.

VII. OTHER APPROACHES

As mentioned before, we considered other different approaches in our experiments. In this section, we discuss, in some details, other considered approaches.

A. Preprocessing

We considered training the classifier on the whole form instead of separate lines, however this resulted in a huge performance impact, due to the small test cases. Moreover, we tried to segment out **connected components** from lines in the *preprocessor*, in order to provide more training samples. Unfortunately, this does not work, as well, as the extracted features are not as good as that of complete lines. That's why, we decided to continue using separate lines in our system.

B. Feature Extraction

We, also, considered other feature extraction methods based on *texture descriptors*, including :

- **Local Phase Quantization (LPQ)** : offers around 88% accuracy on randomly-sampled test cases, which is worse than *LBP*. This is mainly because the form images do not contain heavy blurring and scanning issues.

- **Gray Level Co-occurrence Matrix (GLCM)** : very primitive texture descriptor, offers humble performance.
- **CSLBCoP** : integrates *GLCM* with **Center Symmetric Local Binary Pattern (CSLBP)** of an image to obtain the feature pattern. It offers even lower performance than *LPQ* with around 81% on randomly-sampled test cases.
- **Principal Component Analysis (PCA)** : we tried extracting principal components from *LBP* features, in order to reduce the feature vector size and avoid overfitting. Unfortunately, this does not improve performance over complete *LBP* features.

C. Classification

As mentioned before, other classifiers are considered as well. However, based on the previous analysis, we choose *SVM* as our best classifier. We use **sklearn** for training all mentioned classifiers. Also, we use **sklearn grid search** for model parameters tuning.

VIII. WORKLOAD DISTRIBUTION

Name	Workload
Mohamed Shawky	- Vectorized LBP Implementation. - Classifier Experimentation. - Code Integration.
Remonda Talaat	- Form Clipping. - Vectorized LPQ Implementation. - Code Review.
Mohamed Ahmed	- Line Segmentation. - GLCM CSLBCoP Implementation. - Code Review.
Mohamed Ramzy	- Iterative LPQ Implementation. - Iterative GLCM Implementation. - PCA Implementation.

IX. CONCLUSION AND FUTURE WORK

Classical approaches can be used to build a very robust writer identification system. In this work, we adopt different *texture descriptors* as feature extractors. We show that *LBP* features are simple yet powerful feature extractor for handwriting encoding. Also, we discuss the performance of different classifiers and illustrate that *SVM* offers the best consistent performance. We, also, introduced the idea of dividing the form into lines and taking the majority vote, which greatly improves the performance in our test environment, because of small test cases.

This work can be improved in the following ways :

- 1) Improve *preprocessor* to handle different illuminations and perform color correction.
- 2) Expand *LBP* features to include more than 8 neighbors.
- 3) Try to ensemble multiple classifiers output for better and more stable performance.

REFERENCES

- [1] Writer identification using texture features: A comparative study.
- [2] Text independent writer recognition using redundant writing patterns with contour-based orientation and curvature features.
- [3] An improved online writer identification framework using codebook descriptors.