Pattern Recognition and Neural Networks
Writer Identification System

Mohamed Shawky Zaky AbdelAal Sabae
Section:2,BN:15
mohamed.sabae99@eng-st.cu.edu.eg

Remonda Talaat Eskarous
Section:1,BN:19
remonda.bastawres99@eng-st.cu.edu.eg

Mohamed Ahmed Mohamed Ahmed
Section:2,BN:10
mohamed.dardir98@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

IV. FEATURE EXTRACTION MODULE

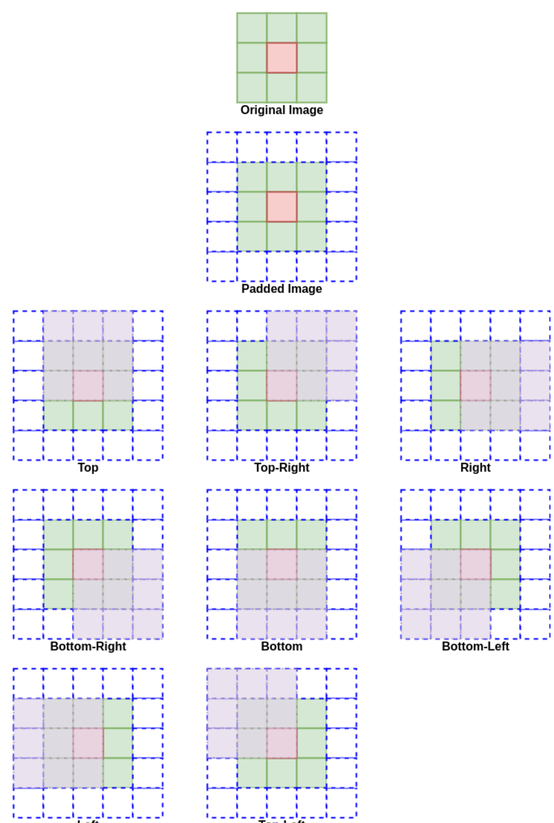

Fig. 1: Illustration of our vectorized implementation of LBP texture descriptor.

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 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 to up to 0.02 second per form.

Figure 1 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

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.

VII. OTHER APPROACHES

VIII. WORKLOAD DISTRIBUTION

Name	Workload
Mohamed Shawky Zaky	
Remonda Talaat Eskarous	
Mohamed Ahmed Mohamed Ahmed	
Mohamed Ramzy Helmy	

IX. CONCLUSION AND FUTURE WORK

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.