CMP450 – Semester 20/21

Text Independent Writer Recognition Using LBPH & SVM

January 19, 2021

Team #1

Mahmoud Othman Adas SEC:2, BN:19* and Yosry Mohamed Yosry SEC:2, BN:33† and
Ahmad Mahmoud AbdElMen'em Afifi SEC:1, BN:5‡ and Abdulrahman Khalid Hassan SEC:1, BN:30§

Department of Computer Engineering, Cairo University
Email: *mahmoud.ibrahim97@eng-st.cu.edu.eg, †yosry.mohammad99@eng-st.cu.edu.eg,
‡Ahmed.Afifi98@eng-st.cu.edu.eg, §abdulrahman.elshafie98@eng-st.cu.edu.eg

I. Introduction

Writer identification is a problem that has lots of use cases in many fields, such as forensics and security. We created a text-independent writer identification system that is able to figure out the writer of a piece of handwritten text among several ones, given only a few handwritten samples of each writer. Our system does not require any former training to be able to identify who the writer is, it only needs two text-independent samples per writer. All the samples we worked on are full forms from the IAM Handwriting Database, and we assume that all inputs follow the same format.

II. Pipeline

Our system's input consists of a few (in our case, two) writing samples for each suspected writer, and one test image that we need to identify the writer for. The output of the system is the number (index) of the writer who the system believes is most likely to be the actual writer of the test image.

Figure 1 illustrates the system pipeline. The input images to our system undergo the following operations:

1) Preprocess the form images to extract the actual handwritten text from the form.
2) Use segmentation to separate individual handwritten lines.
3) Compute the Local Binary Patterns (LBP) of each image, line by line, and construct the LBP histogram, which represents the feature vector for each image.
4) Use a Support Vector Machine (SVM) classifier to figure out which writer most likely wrote the test image.

III. Preprocessing Module

Before starting writer recognition, a form needs to be analyzed to identify the handwritten area. For that purpose, we need to provide preliminary image pre-processing. The pre-processing algorithm includes mainly two steps:

1) Handwritten Area Detection.
2) Line Segmentation.

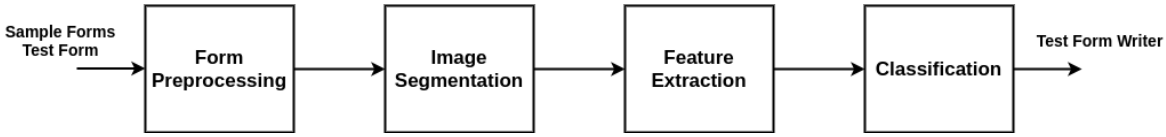

Figure (1) System Pipeline

A. Handwritten Area Detection.

The very first step of our algorithm to detect the writer of handwritten text is to detect and extract the handwritten area from the input image. It's noticed that all IAM database images have exactly three horizontal lines and The handwritten area is always contained between the second and the third lines. So when we talk about line detection, of course, we have to mention Hough Transform.

Hough Transform is used widely to find imperfect instances of objects within a certain class of shapes by a voting procedure that is carried out in a parameter space of the required shape. By using Hough transform we were able to detect the three horizontal lines in IAM database forms and by ordering them by their `y` values in pixels, the two horizontal lines that contain the handwritten text can be easily detected as shown in Figure 2.

We used some heuristic values to be able to crop the handwritten area from the IAM database forms in case of Hough transform failure to detect the horizontal lines. we noticed that the lower line is always 2800 pixels down from the image top corner and the upper line is ranged from 550 to 700 pixels down from the upper corner. Finally, we cropped the image padding using 100 pixels from the left corner and 50 pixels from the right corner.

B. Line Segmentation.

The second step in the preprocessing module is to segment the input form into a set of contours such that each contour represents one line of the handwritten text. This part depends mainly on morphological operations. It was divided into three main steps:

1) Image Dilation
2) Image Opening.
3) Find Contours.

At first, we perform the morphological dilation operation using horizontal kernel on the cropped image from the previous part so that all words in the same line are merged in one single contour.

We noticed that the dilated image has an overlapped lines merged as shown in Figure 3. So to solve this issue we performed the morphological opening operation using a vertical kernel so that we can remove all vertical lines from the image and separate the overlapped lines.

Finally, we used `openCV finCountours` function to extract the boundary boxes of the segmented lines and we add some margins above and down each line to compensate for the opening effect which may crop some pixels from the actual text line.

IV. Feature Extraction Module

The feature extraction module is to extract features to be able to distinguish one writer from the other.

We chose Local Binary Patterns as our feature extractor to extract features from the gray images of lines of each writer; LBP is a texture descriptor used to extract features by computing a local representation by comparing each pixel with its surrounding neighbors.

The first step in constructing the LBP texture descriptor is to convert the image to grayscale. We select a neighborhood of size r surrounding the center pixel for each pixel in the grayscale image. An LBP value is then calculated for this

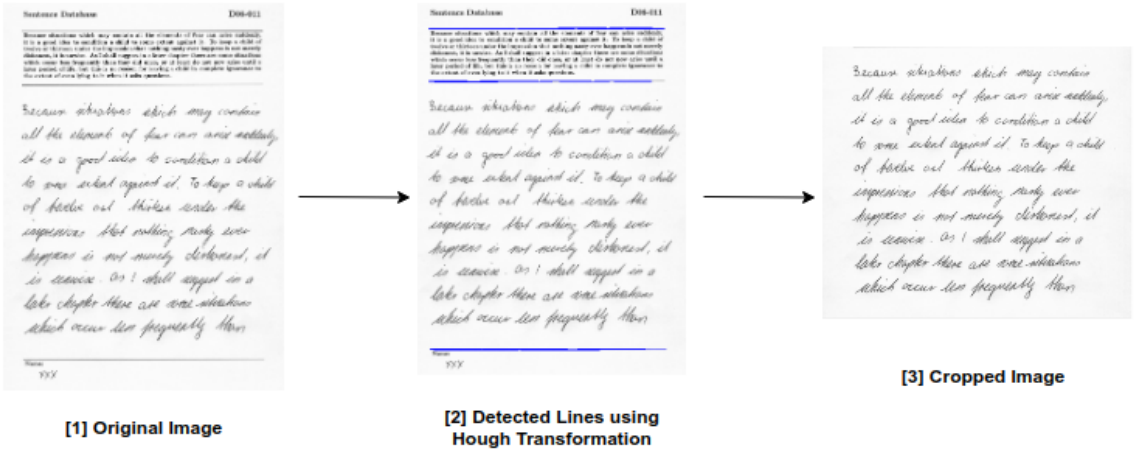

Figure (2) Handwritten Area Detection

center pixel and stored in the output 2D array with the same width and height as the input image.

Figure 4 The first step in constructing a LBP is to take the 8 pixel neighborhood surrounding a center pixel and threshold it to construct a set of 8 binary digits.

In this figure, the center colored by red, LBP checks if the intensity of the center pixel is greater-than-or-equal to its neighbor, then we set the value to 1; otherwise, we set it to 0

The second step is to calculate the LBP value for the center pixel by list the eight binary digits we got in the previous step and convert it to decimal, and we should list the binary digits by the same order and direction for every pixel in the dataset.

Figure 5 The Second step is taking the 8-bit binary neighborhood of the center pixel and converting it into a decimal.

Figure 6 Finally, yielding the LBP value and store it in the LBP output image.

By fine-tuning the parameters of LBP such as the number of points `p` in a circularly symmetric neighborhood and the radius of the circle `r` We chose radius equals 3 and 8 neighbors as it gives us the sweet spot between accuracy and speed. Another acceptable parameter will be four neighbors with a radius of 3. It also works very well with a little dropdown in the accuracy and less calculation time. After calculating the LBP image, we get our feature vector by taking its histogram as our feature vector, and by trials, we found out that masking it with the binary image to calculate only the histogram for the black pixels in the binary image gives us better accuracy. Moreover, because we have eight neighbors, we got $2^8 = 256$ possible patterns, so our LBP image has a minimum value of 0 and a maximum value of 255, allowing us to construct a 256-bin histogram of LBP codes as our final feature vector. Finally, we normalize our feature vector by dividing it by its means.

V. Classification Module

The classification module aims to train the features extracted from handwritten samples from the three different writers with their labels, then trying to predict the test handwritten sample's writer.

After research we chose support vector machine, SVM for short, to be our classification module as it has lots of pros we need such as:

- It has lots of pros, such as it is useful in cases where the number of dimensions is greater than the number of samples.

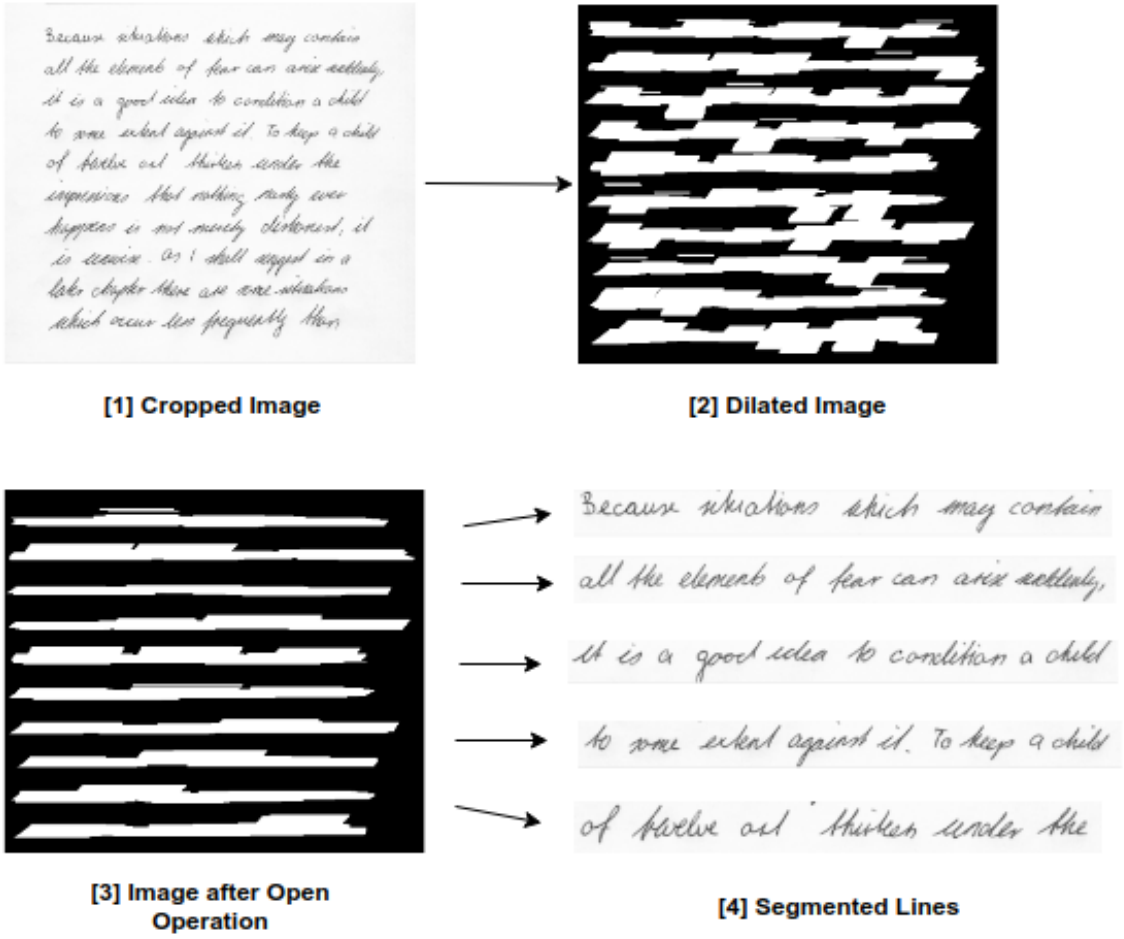

Figure (3) Line Segmentation

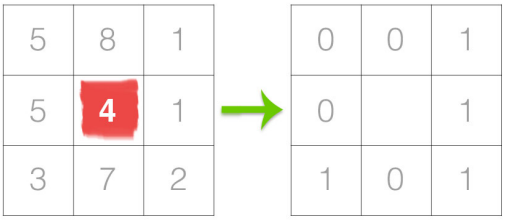

Figure (4) Constructing a set of 8 binary digits from the 8 neighbours surrounding the center.

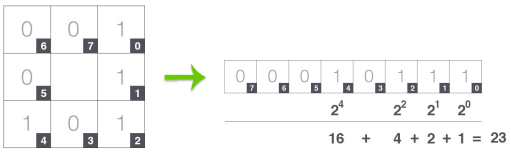

Figure (5) Taking the 8-bit binary neighborhood of the center pixel and converting it into a decimal value.

- It uses a subset of training points in the decision function. Hence, it is memory efficient.
- It is useful in high dimensional spaces.

The SVM algorithm's objective is to find a hyperplane that distinctly classifies the data points in an N-dimensional space where `N` is the number

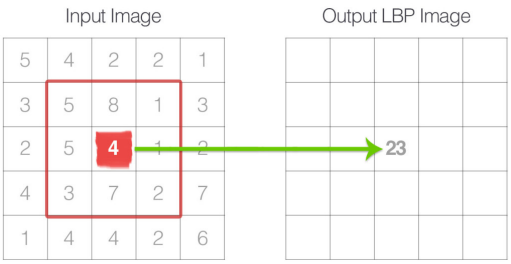

Figure (6) The calculated LBP value is then stored in an output LBP image.

of features.

We are using a `sigmoid` kernel with penalty parameter `C=1` and `ovo` decision function. Furthermore, after fitting the training images' features with their labels, we use the SVM to predict the writer from the test image features.

VI. Speed Enhancements

We put a lot of effort on speeding up the our system's pipeline. The most effective optimization was parallelizing the feature extraction by extracting each image's features in a separate process and then collecting all the features before training.

Processes are quite heavy, but python threads are totally useless, thanks to `GIL`'s locking mechanism. We believe that if we port the code to another language, the execution time would be much lower using threads and manual memory allocation.

Python lists are known to be very slow, so we replaced them all with numpy arrays, and allocated most of the needed memory ahead before the system starts operating on the images. A quite speed gain came from fine tuning `skimage` and `OpenCV` parameters.

We tried to use `Numba` and `Cython` to optimize the execution time but they didn't have an effect. It was probably because most of the code calls numpy, `skimage` and `OpenCV`, which are all written in C and well optimized for memory and CPU.

VII. Performance Analysis

To analyze the performance of our system we created two utility scripts for generating a test set and measuring performance.

The first script generates a random test set from the IAM database. For each test case, it randomly picks three writers, two sample forms for each writer, and a test form that is written by one of these writers. Since the images in the IAM database are large in size (a few MBs each), the script creates hard links to the existing images instead of copying them over to the test set. This enabled us to create huge test sets without worrying about space.

The second script measures the performance of our writer identification system by comparing it's output to the test set's expected output, and calculates the overall accuracy and average time spent per test case (excluding I/O time).

Using these two scripts, we analyzed our system's performance a computer running AMD's Ryzen 9 4000 8-core processor with 16 GBs of RAM, and produced the following results:

Test Set Size	Accuracy	Average Time
100	99%	1.03 sec
1000	98.9%	0.96 sec
10,000	98.8%	0.93 sec

The average time varies depending on the computer specifications, especially the number of cores in the processor as we parallelize the feature extraction on available cores.

VIII. Unsuccessful Trials

We started with and settled on using `LBPH` for feature extraction and `SVM` for classfication. They both gave around 80% accuracy at the beginning, and with tuning for preprocessing the accuracy reached $\tilde{9}9\%$ over large tests. During that, part of the team were testing other feature extraction methods and classfiers.

We tried to extract *Histogram of Oriented Gradients* (`HOG`) features. Using `HOG` gave very

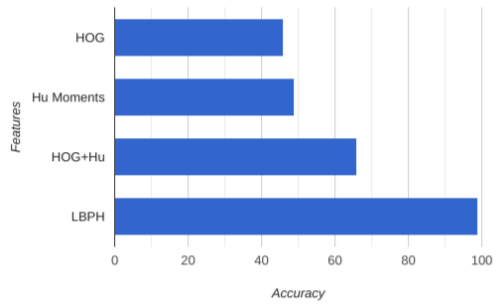

Figure (7) Feature Extraction Methods Accuracy for 15 tests.

low accuracy ($\tilde{4}6\%$) on 15 test cases. We extracted `HOG` from the binary image and then gray image, with no visible changes.

Then we tried to extract the *Hu Moments* that are used to describe the shapes. We extracted *Hu Moments* from each binary line in the image. Using *Hu Moments* with `SVM` gave accuracies lower than that of `HOG` on the same number of test cases.

Being very low made sense, because we tried to describe the shape of the whole line. So we tried to extract *Hu Moments* from a sliding window of size 13×13. This gave accuracy of $\tilde{4}8\%$ on 15 tests. On some lucky iterations, it gave $\tilde{8}0\%$.

Then we tried to mix both `HOG` features and *Hu Moments*. This gave us accuracy of $\tilde{6}6\%$ on 15 tests. It wasn't slower than only `HOG`, because we used subset of both features.

By this time, `LBPH` reached $\tilde{9}9\%$ accuracy on 200 tests. So we abandoned optimizing the feature extraction anymore.

Figure 7 shows the accuracies for the different features.

We tried another classfication method beside `SVM`. We used *K-Nearest Neighbours* (`KNN`), and it gave the same accuracies of `LBPH` but was noticably slower. It made sense that `KNN` is slower, because it iterates over the features multiple times to find the mean and cluster them.

IX. Future Work

In an upcoming work, we could try generalizing our system on different forms of handwritten text by working on other datasets. Other features (in combination with `LBPH`) could be explored if `LBPH` was not enough to produce high accuracy on other forms of handwritten text. We used `scikit-image`'s implementations of LBP to construct our feature vector. Other implementations could be explored in the hope of finding a more efficient implementation. We parallelized feature extraction of different images on different processors to lower processing time. If images must be processed sequentially, we could try parallelizing the calculations of LBP for each line in an image, then collecting the results to construct the LBP histogram.

X. Workload Distribution

A. *Mahmoud Othman Adas*

- Optimized speed with multiprocessing.
- Tried `HOG`, `Hu Moments` for feature extraction.
- Tried `KNN` for classification.
- Tried `Numba` for JIT compiling.

B. *Ahmad Mahmoud AbdElMen'em Afifi*

- Image Preprocessing
- Image Segmentation
- Image Classification
- Implemented the main module to process the test cases.

C. *Yosry Mohamed Yosry*

- Read a few papers about preprocessing.
- Implemented a script to generate test sets.
- Implemented a script to measure performance.

D. *Abdulrahman Khalid Hassan*

- Feature Extraction using LBP.
- Fine-tuning the parameters and trying different settings.
- Debugging and testing.

- Implemented the project with Cython to try to get more speed.
- Tried pypy JIT compiler to try to get more speed.

XI. Conclusion

Writer identification problem can be tackled with different approaches. We explored one of them, and found that the combination of techniques that we used in our system for preprocessing, feature extraction, and classification produced high accuracy on test sets drawn from the IAM dataset forms. By performing feature extraction of different images in parallel, we were able to greatly increase the processing speed, especially on mutli-core devices.