Project report: Handwritten Mathematical Expression Recognition from Photographs

Alicia Rakotonirainy
CentraleSupélec
Gif-sur-Yvette, France
alicia.rakotonirainy@student-cs.fr

Jean Meunier-Pion
CentraleSupélec
Gif-sur-Yvette, France
jean.meunier-pion@student-cs.fr

Abstract

Scientific documents are mainly shared on computers nowadays. However, to avoid spending all day on a computer writing LaTeX code, and to be able to easily transfer handwritten notes to digital devices, computer vision can be applied for Handwritten Mathematical Expression Recognition. Our work is an attempt to develop a method for handwritten mathematical expression recognition, in a framework where expressions are written on paper and photographed. To the best of our knowledge, existing solutions for such a task are very rare, not open source and are specialized for fonts. We aim, here, to develop a method that is oriented towards photographed handwritten mathematical expressions.

The Git repository with all data and code is accessible here.

1. Introduction

Automatic recognition of math expressions is a complex task, especially when dealing with handwritten data. Developing a system for this task would be a huge time saver, allowing people to simply scan a handwritten equation instead of typing it into LaTeX. It also has other possible applications, such as retrieving a mathematical expression in a document containing it, or using it as input of a program evaluating the expression automatically.

The input of a math recognition system can take three forms [20] : vector graphics (such as PDF), strokes (such as pen strokes on tablet) or images. One can also classify them into two different modes : online mode, where tracing information is available, and offline mode, where only the final drawings are provided. The methods used to recognize mathematical expressions are highly dependent on these various input formats. Here, we will focus on offline Handwritten Mathematical Expression Recognition (HMER) based on photographs, which constitutes the most general use case. It is often viewed as a much harder problem as compared to online HMER because of variability in writing style and the impossibility to exploit temporal information.

HMER has been investigated for several decades. The first work that aimed at recognizing mathematical handwritten expression was published in 1999 [12]. But the author relied solely on online data, and the tool was designed to detect the handwriting of only one person. Moreover, they did not use photographs but clean text written on a digital device, suppressing much of the noise in the data. More recent work, such as [7] can process photographs of handwriting, but still only with online data. On top of that, their tool only recognizes single symbols, not whole equations.

In recent years, several works based on Deep Learning techniques (encoder-decoder frameworks [17], adversarial learning [16], etc.) were proposed and achieve very high performances. Still, the ecological impact of deep learning methods lead us to develop a new method based on th wide variety of classical computer vision techniques. Several commercial applications have also emerged, such as MathPix or Solvio. These apps are very efficient for the task at hand, but they are paying services. We therefore want to address this lack of open-source offline HMER tools.

2. Problem Definition

The problem we consider in this project is handwritten mathematical expression recognition from paper documents.

Handwritten Mathematical Expression Recognition (HMER) is the task consisting of automatically recognizing handwritten mathematical expressions. It is an optical character recognition (OCR) task where the characters under study compose mathematical expressions. Moreover, another level of difficulty is added as characters are handwritten and do not come from a font.

Several contests involved HMER in the past. Especially, the CROHME challenges in 2011, 2012 and 2013, provided datasets for HMER. However, the data used in these challenges were expressions written on numerical tablets, not on paper, so there were no issues with background noise. Here, in our project, we want to develop an HMER system that aims at converting mathematical expressions written on paper and photographed into LaTeX expressions, so an important part of the system is the image preprocessing and detection of characters, while images have noisy backgrounds.

There are two different kinds of OCR tasks: off-line OCR and on-line OCR. Off-line OCR only uses images while on-line OCR gives access to the pen strokes obtained during writing acquisition. In this project, we work in an off-line mode.

Here is a short problem statement: given an image I which is a photograph of a handwritten mathematical expression written on a sheet of paper, we want to get a LaTeX transcript of all mathematical expressions in I.

This problem can be divided into two main parts: (1) character extraction and (2) character recognition.

2.1. Character extraction

The first part consists of finding mathematical expressions in an image and getting all characters of the mathematical expressions.

From an image I, we want to get a sequence $char(I)$ of handwritten characters that are in I.

To start with a simplified version of the problem, we consider images which contain only one line of mathematical expression.

2.2. Character recognition

Once a character sequence $char(I)$ is extracted from I, we know that each element of $char(I)$ is a character, but we do not know which character it is. To recognize characters, a model $\mathcal{M}$ is used to take a handwritten character c as input and predict what character $\mathcal{M}(c)$ it represents.

3. Related Work

A survey concerning mathematical expression recognition was published in 2001 [4].

3.1. Character extraction

For character extraction, the segmentation of symbols is performed in [8] by using two modules. A first module identifies groups of characters such as those with square-roots $\surd$ and it segments characters by using projections on the x-axis and the y-axis. It generates a data structure with a tree to represent the mathematical expression. The second module is used to correct the expression tree, especially for characters which have several connected components as i, j and $=$.

Projection profile cutting, which consists of recursively computing histogram intensities along the x-axis and the y-axis to get the profile of a mathematical expression was used in [19].

In our work, we assume that there is only one line of mathematical expression per image and we simply use the histogram of intensities along the x-axis to do character segmentation, which is a simplified version of projection profile cutting.

3.2. Character recognition

For character recognition, some systems use template matching approaches, i.e. compare reference characters to characters to identify, as in [6]. In this paper, they resized images to a standardized shape and used a nearest neighbor classifier to classify images.

This approach is similar to ours as we do template matching, but we do an average pooling and compute a cosine similarity, whereas [6] uses nearest neighbor wih the Euclidean distance on the whole images.

Other systems use structural approaches as [3] which uses graph grammars to model mathematical expressions, in an attempt to better get the complexity of mathematical expressions, with superscripts for example.

4. Methodology

4.1. Data collection

In order to perform HMER from photographs, we needed first a dataset to evaluate our methods. To the best of our knowledge, there are too few HMER datasets, and none of them corresponded to the task we wanted to address. The HME100K dataset [18] could suit our needs but is not accessible for free.

For HMER, the CROHME datasets [13–15] are widely used, but they consist of expressions directly written on a screen, not on paper and then photographed.

We want to consider a real-world study case where mathematical expressions are written on sheets of paper. We thus decided to make our own dataset.

A first option to create this dataset would consist of taking photographs of several matheatical expressions. However, this approach is very time-consuming, especially for the labelisation part.

We thus decided to make a dataset through another approach. Building upon the existing CROHME datasets, we followed the methodology from [11] to create a synthetic dataset made of mathematical expressions to which a realistic background is added.

To reduce the task complexity, we decided to only use characters from the following set of characters:

$$\mathcal{C} = \{(,),+,-,0,1,2,3,4,5,6,7,8,9,=,[,]\} \quad (1)$$

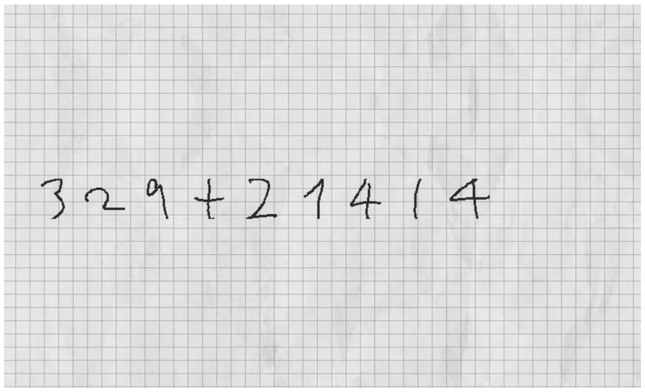
Figure 1. Example of original input image

The equations used in this project thus only contain characters from the set $\mathcal{C}$ defined in 1, which correspond to digits and simple operators that can be used in primary school. In particular, this allows us to avoid one of the specific difficulties in the task of recognizing mathematical expressions : the 2D-structure (for instance x^2 or $\frac{x}{y}$). We restricted this work to one-line expression, written in 1D.

We generated thirty different expressions, each composed of 9 characters randomly sampled from the CROHME dataset and belonging to the set $\mathcal{C}$. We used four different background images representing a sheet of paper. From each equation, we generated four images by adding each background to the equation. Therefore, we ended up with 120 images.

4.2. Image preprocessing

Raw data are photographs of handwritten mathematical equations written on paper, with background lines or shadow. First, we need to have a preprocessing phase before extracting characters from those images.

This preprocessing phase is composed of the following steps:

1. RGB to grayscale conversion
2. Edge detection with the Canny edge detector
3. Background lines detection with the Hough transform and background lines removal
4. Gaussian blur and Balanced Histogram Thresholding
5. Median filtering

Each of those steps are described in the following paragraphs in more details. For illustration, one image of the dataset (Fig 1) is printed at each of the step.

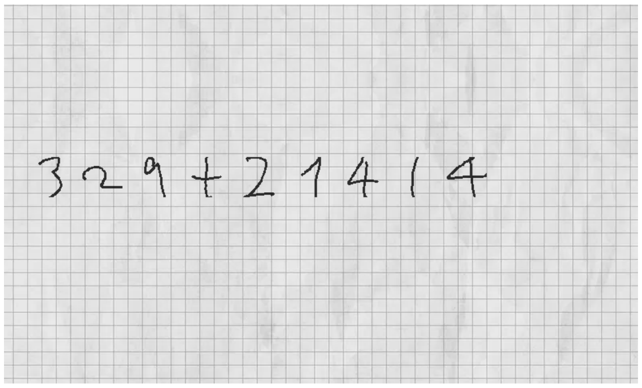
Figure 2. Step 1 : Grayscale conversion (there is not much difference for this particular background)

4.2.1 RGB to Grayscale conversion

First, raw images are in RGB format. We project them onto a grayscale color space ; see Fig 2.

To do so, we used the OpenCV library from Python, with the cvtColor method and the constant COLOR_BGR2GRAY.

The purpose of converting images to grayscale is to have only one channel. Indeed, for character recognition, images are divided into two parts: background and foreground. So we only need two colors, black and white, to detect writing.

4.2.2 Edge detection with the Canny edge detector

Once images are converted to grayscale, we used a Canny edge detector to detect edges of the images [2] ; see Fig 3.

The Canny edge detector works as follows. In a first step, a 5×5 Gaussian filter is applied to reduce noise. Then, we apply a Sobel operator [10] on the obtained image with an aperture size equal to 3. The Sobel operator lets us get the first derivative of the intensity along the x-axis G_x and along the y-axis G_y. From that, we can derive the edge gradient for each pixel:

$$\begin{aligned} Gradient\_Magnitude(I) &= \sqrt{G_x^2 + G_y^2} \\ Gradient\_Direction(I) &= \sqrt{\arctan\left(\frac{G_y}{G_x}\right)} \end{aligned} \tag{2}$$

Once we have the edge gradient, we do non-maximum suppression, i.e. only pixels whose gradient magnitude is a local maximum in their gradient direction are kept.

Finally, on the remaining pixels, we do hysteresis thresholding. We define two threshold values t_{min} and t_{max}, which are set to 20 and 150 in this work. Then, for all remaining pixels, if their intensity is below t_{min}, they are not edges. If their intensity is above t_{max}, they are edges. And

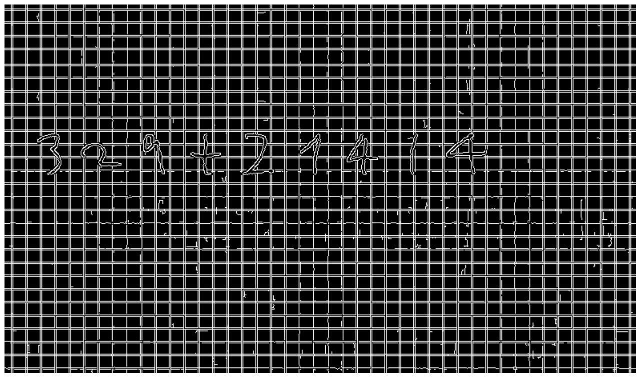

Figure 3. Step 2 : Canny edge detector

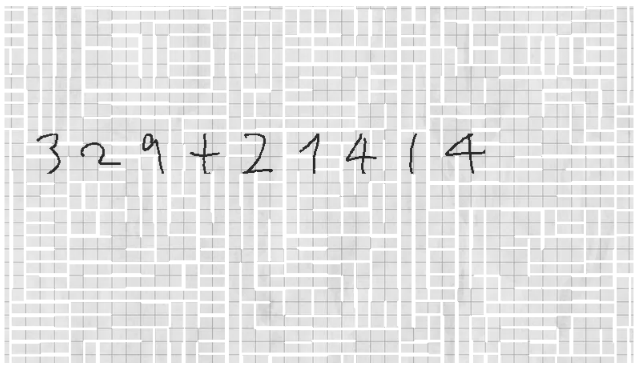

Figure 4. Step 3 : Background lines removal with the Hough Transform

if their intensity is between t_{min} and t_{max}, they are edges if and only if they are connected to pixels that are edges.

4.2.3 Background lines detection with the Hough transform and removal of the lines

Background images contain horizontal and sometimes vertical lines. They are harder to remove by a simple thresholding since they are almost as dark as our written equation. To detect those straight lines, we used the Hough transform on the previously computed canny edges.

Once we detected background lines, we removed them by covering them with high intensity pixels, which will then be easier to remove in the next step.

4.2.4 Gaussian blur and Balanced Histogram Thresholding

To divide the images into two parts, namely background and foreground, we need to find a threshold that separates background and foreground intensities.

We want to do it automatically, as the intensity distribution may vary from one image to another. We thus resorted to Balanced Histogram Thresholding [1].

Balanced Histogram Thresholding is an easy-to-implement method that provides good results with few computational resources. It is used to find the threshold between 2 modes in a bimodal distribution.

This method works as follows. We have three positional indices i_l, i_r and i_m that represents the index of the lowest intensity and the highest intensities in the histogram, and i_m is the mean index, equal to $\frac{i_l+i_r}{2}$ and rounded when necessary.

We cut the intensity histogram in two parts: the left part which contains the lower intensity values, ranging from i_l to i_{m-1} and the right part that contains the higher intensity values, ranging from i_m to i_r. We compute the weight of the left part w_l as being the sum of the left part intensities and the weight of the right part w_r as the sum of the intensities of the right part. Then, we compare w_l and w_r.

If $w_l < w_r$ (resp. $w_r \leq w_l$), it means that there is more weight on high (resp. low) intensities, so we drop the highest (resp. lowest) bin of the right (resp. left) part and remove its contribution from w_r (resp. w_l). Then, we update the center of the histogram and adjust the weights in that respect. It means that now we have $i_r = i_r - 1$ (resp. $i_l = i_l + 1$), and we update i_m.

When $i_l = i_r$, the procedure is stopped and the common value of i_l, i_m and i_r is the threshold obtained by Balanced Histogram Thresholding.

We tested this method on the image with and without a previous Gaussian Blur. We obtained far better results in the determination of the threshold when the image was first blurred (see Fig 6 and Fig 7 for a comparison), and even more when the size of the kernel was very high, resulting in a highly blurred image. This is surely due to the fact that the background of the image represents an overwhelming majority of the pixels, compared to the few dark pixels in the equation. This makes the Balanced Histogram Thresholding method work less well. The dark pixels are too few, and therefore considered as noise and not as a mode of the intensity distribution. We found that the Gaussian blur allows to increase the number of dark pixels in the image by smoothing the edges, and thus to amplify the signal associated with the dark pixels. The method then recognizes them as a modality in its own right.

Once the threshold is found on the blured image, it is applied on the non-blurred image. All pixels with intensity higher than this threshold are set to white, and the other to black.

We also tested other thresholding methods, such as the Otsu Thresholding. By tuning the parameters, we achieved good removal of the background on one particular image. But as soon as we applied it to an image with a different background, the results were very poor. This lack of generalization made us opt for the Balanced Histogram Thresholding method.

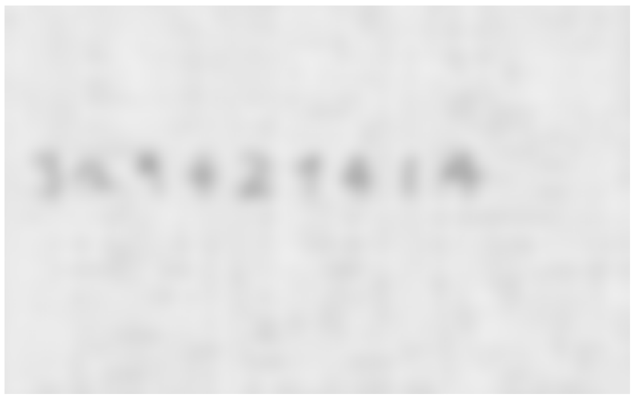
Figure 5. Step 4a : High gaussian blur to increase the dark signal

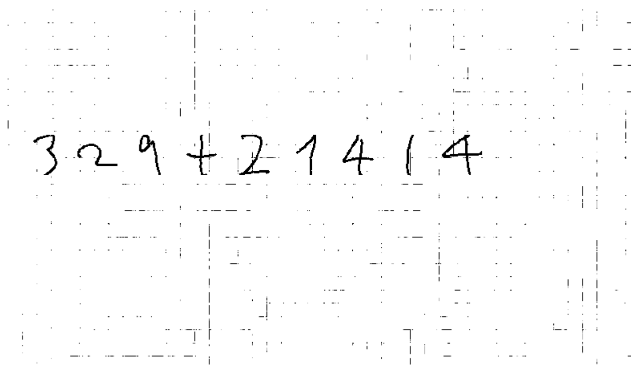
Figure 6. Step 4b : Balanced Histogram Thresholding of the image of step 3 based on the threshold found on step 4a

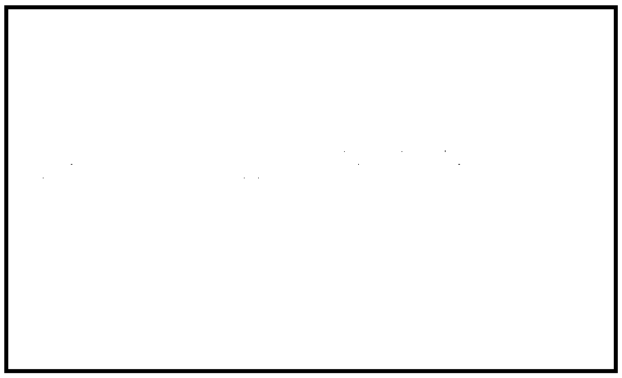
Figure 7. Step 4b : Example of Balanced Histogram Thresholding of the image of step 3 without blurring the image to obtain the threshold. We see that we keep only background and loose the text completely.

4.2.5 Median filtering

We finish by applying a median filter to remove the remaining noise of the resulting image. Median filter consists of replacing each pixel intensity by the median of its neighbors in a certain range. Here, we consider all neighbors that are at most three steps away.

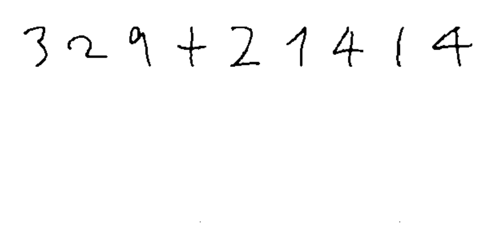
Figure 8. Step 5 : Median filtering

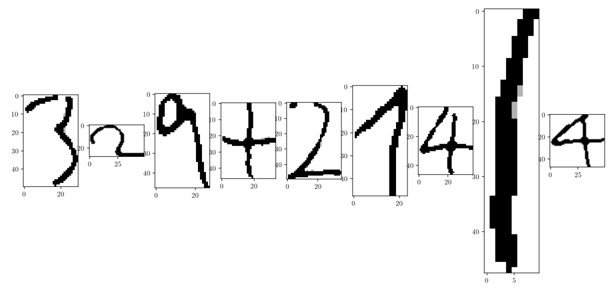
Figure 9. Segmented characters

4.3. Character segmentation

Once the images are preprocessed, we want to extract characters. The preprocessing phase lets us obtain images whose background and foreground are clearly identified, with foreground pixels being set to 1, and background pixels set to 0.

Building upon the preprocessed images, we extract characters by considering that text is written horizontally and that each character is separated from its neighbors by a background space, i.e. there is at least one vertical line of background pixels between the character and the previous and following characters.

To make sure that the extracted characters are relevant, and are not just isolated pixels for example, we do not keep characters of less than one pixel of width.

Then, before moving on to the features extraction phase, we crop the character images so that there remain no extra background horizontal or vertical lines on the boundaries.

4.4. Features extraction

Once images are preprocessed and character images extracted from the original images, we need to perform character recognition to identify characters.

To do so, we decided to use two types of features.

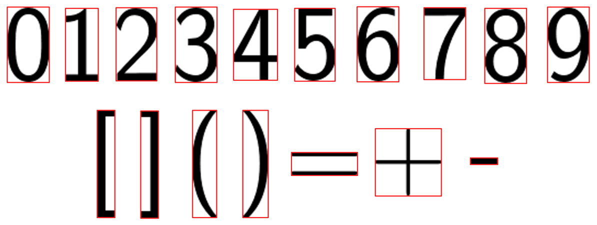

Figure 10. Reference characters of our dataset, to which handwritten characters are compared.

First, we used reference characters, namely images of corresponding LaTeX characters, to compute similarity scores. And we used signatures of characters, which consist of a 1D representation of the boundary of a character.

4.4.1 Reference characters

In order to recognize characters and decide which character from $\mathcal{C}$ handwritten characters correspond to, we resort to reference characters which we will compare extracted characters to. For each element of $\mathcal{C}$, we took images from black LaTeX characters written on a white background. The reference characters used in our dataset are provided in Figure 10.

4.4.2 Similarity score

Extracted characters have variable size. First, we resize each of them to a standard shape which is arbitrarily set to 50×50.

Then, we split the character images into a partition of 25 regions of size 10×10.

We denote by $\mathcal{R}_{i,j}$ the region composed of pixels whose coordinates are in $[\![10 \times (i-1)+1, 10 \times i]\!] \times [\![10 \times (j-1)+1, 10 \times j]\!]$.

The intensity range is usually 0-255 for grayscale images. Here, we only keep two levels, background and foreground, so the only intensity values are 0 and 255. To make computations easier and to polarize the background and the foreground, we set all background pixels to -1 instead of 255, and all foreground pixels to 1 instead of 0. Indeed, when computing cosine similarity, it will provide much more meaningful results to have the same absolute value for background and foreground pixels, and to have opposite values.

For each region $\mathcal{R}_{i,j}$, we compute the mean intensity $\mu_{i,j} = \underset{p \in \mathcal{R}_{i,j}}{mean} I(p)$.

We then obtain a 25-dimensional feature vector for each character image, that we can denote $\mu(c)$ where c is the handwritten character image.

Then, for each reference character image c_{ref} in $\mathcal{C}$ as defined in 1, we compute $\mu(c_{ref})$.

For each character image c, we want to compute a similarity score that tells how close to reference characters the handwritten character is. To do so, we compute the cosine similarity between each handwritten character c and each reference character c_{ref}, that we denote

$$sim(c, c_{ref}) = \frac{c \cdot c_{ref}}{||c|| \times ||c_{ref}||} \tag{3}$$

where $\cdot$ denotes the scalar product and $||.||$ denotes the Euclidean norm.

The values of $sim(.,.)$ range from -1 to 1 and the higher, the more similar feature vectors are, i.e. the more similar images are.

For each character c, we define the similarity feature vector as

$$similarity(c) = (sim(c, c_{ref}))_{c_{ref} \in \mathcal{C}} \tag{4}$$

which contains the similarity scores with all the reference characters.

Additionally, we tried to use the L_2 norm instead of the cosine similarity to compute similarity scores, but it did not work as well as cosine similarity.

4.4.3 Signature

The signature is a one-dimensional representation of the boundary of an object. This is computed by computing the distance from the centroid of the object to the boundary as a function of angles, from zero to 360 in any chosen increment.

For instance, the signature of a perfect circle would be the constant function equals to r, the radius of the circle, on the interval $[0, 360]$. An example of the shape of the signature for a handwritten (, the reference (and a comparison with the signature of a handwritten 1 is given in Figure 11. Let us denote by $S(c)$ the signature of the character c.

Once $S(c)$ is computed, we compute the distance to the signatures of each reference character $S(c_{ref})$.

To compute the distance between the 2 curves, we used the area method, an algorithm that calculates the area between 2 curves in 2D space. It is implemented in the similaritymeasures package [9].

At the end, for each character c, we obtain a feature vector $signature(c)$ defined as the distance of $S(c)$ to the signature of each reference :

$$signature(c) = (\text{areadist}(S(c), S(c_{ref}))_{c_{ref} \in \mathcal{C}} \tag{5}$$

4.5. Model for character recognition

To make predictions, we concatenate the feature vectors defined in section 4.4.2 and section 4.4.3, to define a final feature vector $f(c)$ for each character c:

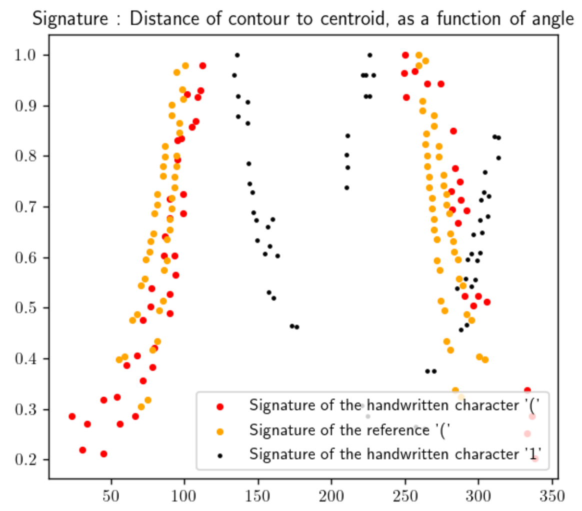

Figure 11. Signature feature for a handwritten "(", compared to the signature of its reference image, and to the one of a handwritten "1".

$$f(c) = (similarity(c), signature(c)) \tag{6}$$

We then use these feature vectors to feed a machine learning model.

The machine learning model we use is an XGBoost model [5].

4.6. Prediction

To make predictions, we run the XGBoost model on the character sequence and let it predict each character of the sequence, based on the features presented in section 4.4.

In order to train the XGBoost, we used the images of the CROHME 2012 dataset. This was not straightforward, since this dataset was designed for on-line character recognition, and we had to convert the files in "inkml" format (that gives the pen strokes informations) into "png" format. At this end, we used a publicly available package : crohme-extractor. We obtained images of dimension 50×50, each representing 1 black handwritten symbol on a white background.

We balanced our train dataset, so that each of the 17 classes in $\mathcal{C}$ are represented 500 times, which makes a total of 8500 samples.

We trained our XGBoost on this trainset, using a GridSearch to optimize the hyperparameters.

5. Evaluation

A fixed number of isolated symbols (from the digits and operators class) were randomly drawn and superimposed on 5 different paper backgrounds at a random position to form an equation of one line per image.

We generated 30 different false equations, each superimposed on 5 different backgrounds, i.e. 120 artificial images.

We then ran these images through our pipeline. There are actually two steps to validate:

- The correct extraction of the right number of characters which the equation is composed of
- The recognition of each of these characters

We have therefore measured :

- The percentage of images where the correct number of characters was extracted
- In the images where the correct number of characters was detected, the percentage of correctly recognized characters

We obtained disappointing results on these two metrics:

- Percentage of images where the correct number of characters was detected: 75.83%.
- Percentage of characters correctly detected: 13.92%.

We tried to understand these very low results. Firstly, the rate of equations where the number of characters is correctly detected is surprisingly low. We had difficulty in finding a preprocessing that suited the diversity of the backgrounds tested: gridded backgrounds, plain backgrounds, various luminosity... If one element of the image is wrongly detected as a character, the whole detection is wrong.

We therefore computed another metric to assess the performance of the whole pipeline : the Levenshtein edit distance. The edit distance is a way of counting how different two strings are, by counting the minimum number of operations (allowing insertion, deletion and substitution) required to transform one string into the other. This is a more accurate metric since it can compare strings from different lengths.

On the same test dataset, we obtained an average Levenshtein edit distance of : 8.00. This means that our classifier is not good enough.

We conducted an ablation study to see how differently features from Sections 4.4.2 and 4.4.3 contribute to the predictions. The results are reported in Table 1.

In general, it turns out that both types of features provide a similar performance. However, signature features prove to perform better than similarity score features, with 13.92% against 11.97% and a Levenshtein distance of 7.59 against 8.18.

There is a clear room for improvement for character recognition as well as some room for improvement for character extraction.

Features	Accuracy	Levenshtein
Sim	11.97%	8.18
Sig	**13.92%**	**7.59**
Sim + Sig	12.82%	8.00

Table 1. Ablation study for different cmbination of features. ”Sim” stands for features from Section 4.4.2, while ”Sig” stands for features from Section 4.4.3. We report both the character recognition accuracy and the Levenshtein distance.

One explanation is that, for similarity score features for example, features are scale invariant but not invariant to the character thickness. Besides, there is only one template per character, which is very limited, and data augmentation, with italicized characters for example, could be a way to improve the predictions.

6. Conclusion

In this work, we developed a system to perform handwritten mathematical expression recognition on photographed expressions. It involved three phases, namely preprocessing, character extraction and character recognition. The results we obtained for character extraction show that there is still room for improvement, while the results for character recognition are definitely not satisfactory. There are two reasons for this poor performance. First, the features we used, based on template matching and signatures do not get all the complexity of handwritten characters. Secondly, we decided to tackle handwritten mathematical expression with background noise, which can take several forms, like background lines or varying luminosity. However, it turns out that our system is still not robust enough to correctly retrieve all characters from the input images.

References

[1] António Anjos and Hamid Shahbazkia. Bi-level image thresholding - a fast method. volume 2, pages 70–76, 01 2008. 4

[2] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679 – 698, 12 1986. 3

[3] Mehmet Celik and Berrin Yanikoglu. Handwritten mathematical formula recognition using a statistical approach. 04 2011. 2

[4] Kam-fai Chan and Dit-Yan Yeung. Mathematical expression recognition: A survey. *International Journal on Document Analysis and Recognition*, 3, 02 2001. 2

[5] Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. 7

[6] P. A. Chou. Recognition of Equations Using a Two-Dimensional Stochastic Context-Free Grammar. In William A. Pearlman, editor, *Visual Communications and Image Processing IV*, volume 1199, pages 852 – 865. International Society for Optics and Photonics, SPIE, 1989. 2

[7] Kenny Davila, Stephanie Ludi, and Richard Zanibbi. Using off-line features and synthetic data for on-line handwritten math symbol recognition. volume 2014, pages 323–328, 09 2014. 1

[8] Claudie Faure and Zi-Xiong Wang. Automatic perception of the structure of handwritten mathematical expressions. 1990. 2

[9] Charles F Jekel, Gerhard Venter, Martin P Venter, Nielen Stander, and Raphael T Haftka. Similarity measures for identifying material parameters from hysteresis loops using inverse analysis. *International Journal of Material Forming*, may 2019. 6

[10] N. Kanopoulos, N. Vasanthavada, and R.L. Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of Solid-State Circuits*, 23(2):358–367, 1988. 3

[11] Seunghyun Chae Keechang Choi. Handwritten Math Formula Recognition. 2

[12] Nicholas Matsakis. Recognition of handwritten mathematical expressions. 05 2000. 1

[13] Harold Mouchère, Christian Viard-Gaudin, Utpal Garain, Dae Hwan Kim, and Jin Hyung Kim. CROHME2011: Competition on Recognition of Online Handwritten Mathematical Expressions. In *11th International Conference on Document Analysis and Recognition, ICDAR 2011*, page 4 p., Beijing, China, Sept. 2011. 2

[14] Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. ICFHR 2012 - Competition on Recognition of On-line Mathematical Expressions (CROHME 2012). In *ICFHR 2012*, pages 1–6, Bari, Italy, 2012. 2

[15] Harold Mouchère, Christian Viard-Gaudin, Richard Zanibbi, Utpal Garain, Dae Hwan Kim, and Jin Hyung Kim. Icdar 2013 crohme: Third international competition on recognition of online handwritten mathematical expressions. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1428–1432, 2013. 2

[16] Guangcun Shan, Hongyu Wang, and Wei Liang. Robust encoder-decoder learning framework towards offline handwritten mathematical expression recognition based on multi-scale deep neural network. *CoRR*, abs/1902.05376, 2019. 1

[17] Jiaming Wang, Jun Du, Jianshu Zhang, and Zi-Rui Wang. Multi-modal attention network for handwritten mathematical expression recognition. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1181–1186, 2019. 1

[18] Ye Yuan, Xiao Liu, Wondimu Dikubab, Hui Liu, Zhilong Ji, Zhongqin Wu, and Xiang Bai. Syntax-aware network for handwritten mathematical expression recognition, 2022. 2

[19] R. Zanibbi, D. Blostein, and J.R. Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1455–1467, 2002. 2

[20] Richard Zanibbi, Dorothea Blostein, R Zanibbi, and D Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15, 12 2011. 1