CONVERSION OF HANDWRITTEN MATHEMATICAL EXPRESSIONS TO LaTeX

Gaurav Kumar(14240), Sharad Roy(14628)

Indian Institute of Technology, Kanpur

ABSTRACT

Since it is difficult to render mathematical equations in plain text editors, popularity of typesetting systems like LaTeX has increased. However, typing mathematical expressions in LaTeX is a cumbersome task. In this project, we present a novel algorithm for semantic analysis and conversion of handwritten mathematical expressions to LaTeX. Our algorithm includes three steps - contour based character segmentation, super/sub-script tree generation and individual character classification using CNN. Results generated on handwritten test images are promising.
Code can be found at: GitHub

Index Terms— mathematical expressions, LaTeX, OCR

1. INTRODUCTION

Although traditional OCR is a commonly researched field, recognition of mathematical expressions is an upcoming topic in the field of handwriting recognition. Automated mathematical expression recognition systems can be used by students and professors for online course materials and for conversion of old printed/handwritten mathematical documents into electronic formats. However, this task presents some significant challenges over basic handwriting recognition. First, mathematical characters are more complex than common alphabets and datasets including these symbols are rare. Second, mathematical expressions don't have a linear structure, for example - a symbol may occur in the sub-script of another symbol thus being relatively lower in position compared to its parent symbol. This non-linear structure is extensible infinitely by recursion. To tackle these challenges, we present a novel three step algorithm for complete end-to-end handwritten mathematical expression recognition.

2. RELATED WORKS

A lot of effort has been put in the past decade to address the problem of handwriting recognition. An entire conference named ICFHR (International Conference for Frontiers in Handwriting Recognition) is devoted to this problem.

Some of the early works in this field were accomplished by Le Cun et al.[5] in 1998 when they proposed a method to recognize handwritten digits using a 7 layer Convolutional Neural Network. Since then with the advent of GPU's and increased computational power, deeper networks have been consistently used for recognition. Ciresan et al.[6, 7] proposed deeper networks which could achieve accuracy close to 100 % on MNIST handwritten digit dataset.

For the specific case of mathematical expression recognition, available literature is very few. Awal et al.[8] tries to look at strokes inside text for segmentation and performs it simultaneously along with recognition and interpretation. A web application called Detexify[1] solves this problem to a small scale by identifying mathematical symbols input by the user and provides a ranking of possible outputs. Some other approaches for this problem are also being experimented with currently.

3. DATASET

We are using the *Handwritten Math Symbol* dataset from Kaggle[2] which contains over 100,000 images of size 45×45 across 82 classes including alphabets, numbers, Greek symbols, mathematical operators and set operators.

Different classes have different frequency in the dataset depending on how frequently it is used in normal day to day equations with character " $-$ " being the most frequent with 12328 data points and character "$\exists$" the least with 11 data points. Since the dataset is so skewed, we use only the top 40 most frequent classes for better classification performance by reducing class imbalance.

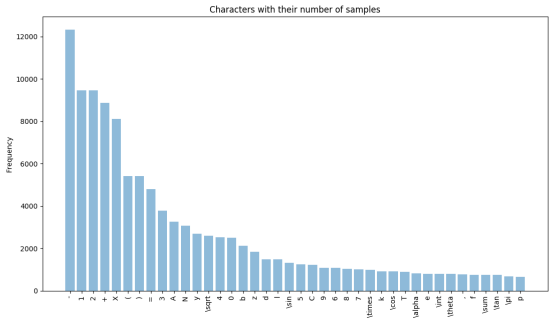
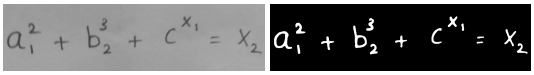


Fig. 1: Distribution of top 40 classes in the dataset

4. PREPROCESSING

The input image is expected to be black or blue ink on white paper. A median blur filter of size 5×5 is applied to remove any noise on the paper that may not be part of the target expression.

To do binary thresholding, we cannot directly use a constant threshold since lighting conditions may vary. Therefore, we must use a custom dynamic threshold. Since our image is text on white paper, its histogram is generally bimodal with maxima at the intensities of the ink and the paper. Therefore we have used the minimum thresholding algorithm which takes a histogram of the image, smooths it repeatedly until there are only two peaks in the histogram, and then selects the minimum between these two peaks as the threshold[3]. We have then used this threshold to binarize the image.



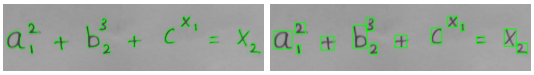
(a) Original (b) Binarized

Fig. 2: Sample result of binary thresholding

We have skeletonized the above binary image using the *cv2.dilate* and *cv2.erode* functions available in the OpenCV library[4]. This will ensure that the segmented characters have thickness of 1 pixel each similar to how they are in the dataset.

5. CHARACTER SEGMENTATION

We are using contouring of character boundaries to segment characters. To do this, we first apply Canny edge detection to the binarized image obtained after thresholding. Next, we draw curves joining all continuous points along the character boundaries. We ensure that internal contour points are not detected to take care of characters like 'a' and 'b'. This gives us the outer boundaries of each character. We then find out the rectangles enclosing these contour points for each character. This gives us the ROI for each character in the image on which we can operate individually for classification.



(a) Contour points (b) ROIs of characters

Fig. 3: Sample result of character segmentation

Since some writers may have a forward slant in their handwriting, we need to deskew the characters. To do this, we have first computed the ratio of the two central image moments using the *cv2.moments* function[4] which is equal to the skew of the image. Next, we apply an affine transformation at this angle to deskew the image.

(a) Skewed (b) Deskewed

Fig. 4: Sample result of deskewing

Finally, each character is cropped from the skeletonized image using the obtained ROIs, converted into a square image by applying horizontal or vertical padding and resized to 45×45 to make sure that they have the same structure as the dataset images.

6. TREE GENERATION

Given two adjacent characters, the second character may have one of many relations to the first - it may be a super-script, sub-script, argument for the first character or it may just be the next character on the same level.

We use a spanning tree structure to annotate the complex two dimensional relations among the characters in the target expression. Each character has three attributes - *mean line, top line and bottom line* which are defined by the bounding box detected in the character segmentation step. Every character is a node in the tree and has the following links to other nodes:

1. *super-script:* bottom line of this character appears above the mean line of current character
2. *sub-script:* top line of this character appears below the mean line of current character
3. *included:* bonding box of this character is partially or completely enclosed in bounding box of current character
4. *next:* the top and bottom line of this character enclose the mean line of current character
5. *parent:* current character is either super-script, sub-script or included of this character

Using the character ROIs generated in the previous step, we iterate over all the characters from left to right to recursively calculate the position of the current character in tree using its relative position in the x and y direction with respect to the characters already in the tree.

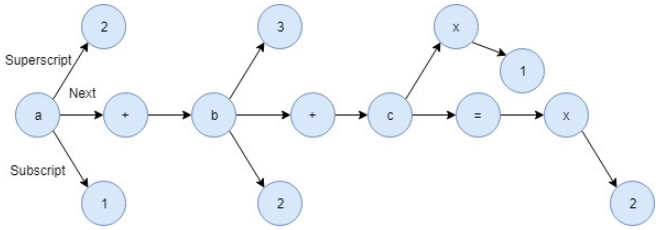


Fig. 5: Illustration of spanning super/sub-script tree

7. CLASSIFICATION

After image pre-processing and applying tree based recursive approach on the generated character images, we are now left with classifying those images. Each and every character image has been reshaped after various operations into a 45×45 image.

We have followed two different approaches for performing classification of these character images. The first approach uses handcrafted features like HOG, SIFT for character recognition. The second approach makes use of deep neural networks like Convolutional Neural Network(CNN) for performing this task. In subsequent sections, we look at details of each approach in depth and compare their performances

7.1. Handcrafted Features such as HOG, SIFT

Histogram of Oriented Gradients[10] is a feature descriptor that counts occurrences of gradient orientations in localized portions of an image. The histogram of these oriented gradients are then used as features. Image gradients are good descriptors because the gradient magnitude is large around edges and corners and it is evident that these regions are more vocal in defining the shape of an image compared to the flat regions.

The character images have been skeletonized to a width of 1 pixel to remain consistent with the training data. Two images from different classes can be differentiated by its stroke pattern which can further be captured by its gradient at any particular location. This intuition helped us to consider features like HOG and SIFT which rely heavily on image gradients for calculating the descriptors.

The following parameters were tuned and selected for finding the hog features:

1. *Cell Size* = $(10, 10)$. It is size of the local image patch taken for calculating the histogram.

2. *Block Size* =$(10, 10)$ It is the image area over which the local histogram is normalized. A large block size decreases the significance of local changes and makes it more robust.

3. *Block stride* = $(5, 5)$. It determines how much each block needs to be shifted and gives us overlapping blocks.

4. *Number of bins* =9. Number of different levels in the histogram of the image. Typically the 9 levels are at $0, 20, 40, ..., 160$ degrees.

7.1.1. Performance

We get a 576 dimensional feature vector for each image after this operation. Training data is split into 4:5 ratio with 80 % of data used for training and 20% for testing. We used 4 different traditional machine learning classifiers for learning the best fit for our data. The accuracy on each model is listed below.

Model	Accuracy on Test Data
SGD Classifier	89.5
Logistic regression	92.5
Naive Bayes Classifier	68.3
Decision Tree Classifier	64.5

Table 1: Models with their accuracies

Logistic regression performs the best on test data whereas Naive Bayes and Decision Tree Classifier had the poorest performance among all these classifiers. Since ours is a 40 class classification task, results achieved by these methods are commendable.

7.2. Convolutional Neural Networks

Convolutional Neural Networks are a variant of neural networks which are used primarily for videos and images.[9] With the use of these networks, a number of computer vision tasks have been able to achieve error rates which are close to human error. They learn features themselves in a hierarchical manner and don't need handcrafted features.

The CNN model was trained using Keras and its architecture is as follows:

1. *Input layer:* Raw pixel values of image normalized between 0 and 1 is provided as input to the CNN layer.

2. *First Convolutional layer*: 30 (5×5) filters are applied to the image to learn lower level features. It is followed by ReLU Activation followed by max pooling operation.

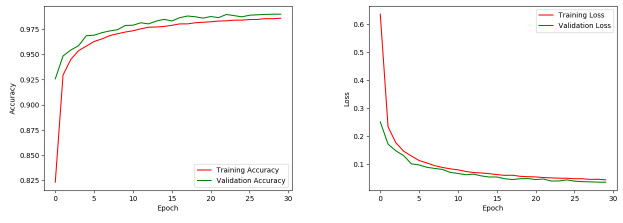
3. *Second Convolutional layer*: 15 (3×3) filters are applied followed by a ReLU Activation which in turn is followed by max pooling operation.

4. *Dropout* is applied next to it to make the system robust and reduce complexity. The dropout layer randomly drops a predefined fraction of neurons during training.

5. Flatten the layer and add two fully connected layers next to it. Apply *softmax* function at the last layer to get class probabilities for each of the 40 classes.

7.2.1. *Hyperparameter Tuning*

After hyperparameter tuning, *Categorical Cross Entropy* loss is used to train the model with an *Adam Optimizer*. *Batch size* is kept at 128 and the number of *epochs* for which the model is trained is 20.



(a) Accuracy (b) Loss

Fig. 6: CNN performance vs. number of epochs

7.2.2. *Performance*

We are getting validation accuracy of 98.59% and validation cross-entropy loss of 0.0484 at the end of 20 epochs of training. We found classification error for each class on training data. The results of the 5 classes with the highest and lowest accuracies are as follows:

Class	Total examples	misprediction	Error %
π	2332	0	0
sin	4293	0	0
cos	2986	0	0
tan	2450	0	0
9	3737	4	0.1

Table 2: Best 5 classified Classes

Clearly, visually complex classes like π, sin, cos, and tan have high classification accuracies since it is easier to distinguish them from other classes.

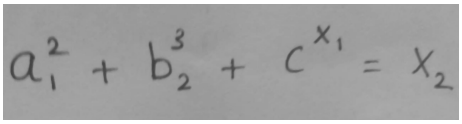
Class	Total examples	misprediction	Error %
$\times$	3251	979	30.11
l	1017	167	16.42
z	5870	327	5.57
(	14294	602	4.21
1	26520	676	2.54

Table 3: Worst 5 classified Classes

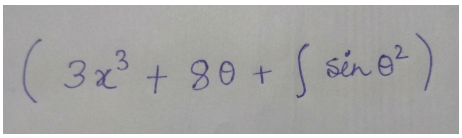
Certain classes like $\times$ and l have low classification accuracy because they look very similar to some other classes (x looks same as `\times` and l looks same as I and i)

8. RESULTS

To test our end-to-end algorithm, we have written mathematical expressions using blue and black ink on white paper and taken pictures using a mobile camera under varying lighting conditions.

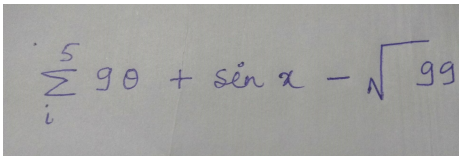


Output 1: $a_1^2 + b_2^3 + c^{x_1} = x_2$

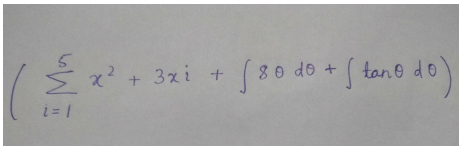


Output 2: $(3x^3 + 8\theta + \int \sin\theta^2)$

Results on the two simple expressions above show that the model works perfectly for simple expressions and characters where multi-part characters like 'i' and 'j' are not involved. As shown in the next two examples, such multi-part characters only get segmented into a single part resulting in the smaller part to be ignored thus resulting in misclassification of the character.

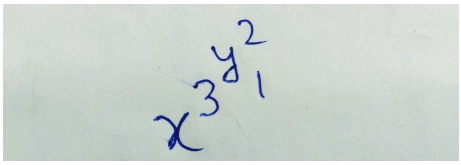


Output 3: $\Sigma_b^5 9\theta + \sin x - \sqrt{99}$



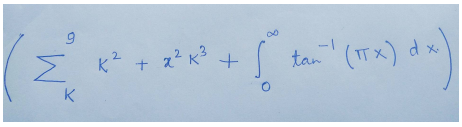
Output 4: $({}_1\Sigma_1^5 x^2 + 3x1 + \int 8\theta d\theta + \int \tan\theta d\theta)$

Output 3 also shows how the *include* feature in the tree works as the under-root only takes in the first '9' and not the second one.



Output 5: $x^{3^{y^2}1}$

Output 5 shows how the algorithm works on multiple nested super/sub-scripts. The final '1' come on the same level as '3' since its almost the same size and occurs at the same level (if it was smaller, or was a bit higher, it would appear as a subscript of 'y' instead of being on the same level as '3').



Output 6: $(\Sigma_k^9 k^2 + 2^2 k^3 + \int_0^\infty \tan^{-1}(\pi x) dx)$

Output 6 shows the performance of the algorithm on long expressions with complicated characters like integration, tan etc. (x is missclassified as 2 due to it not being very clear in the image).

9. CONCLUSION & FUTURE WORK

Our algorithm gives promising results on single line mathematical expressions and is significantly faster than algorithms using sliding window method for character segmentation. Also it can handle virtually infinite nesting of *super/subscript* and *includes* because of the recursive tree generation.

Multi-part characters like 'i' and '!' are not handled properly because of the contour based segmentation method. This can be resolved by creating special cases for these characters and extending the bounding box vertically for them, however this is a brute-force solution and not very attractive.

We are currently working on implementing a interactive web-application for deployement of this project. We can also implement handling of multi-line expressions like fractions or even complete derivations/proofs.

10. REFERENCES

[1] Detexify LaTeX handwritten symbol recognition.

[2] "Handwritten math symbols dataset" by Xai Nano on kaggle.

[3] "Thresholding" on skimage documentation.

[4] "OpenCV User Guide" by OpenCV Dev team on OpenCV Guide.

[5] Yann LeCun, Lon Bottou, Yoshua Bengio, Patrick Haffner. "Gradient-based learning applied to document recognition" Proceedings of the IEEE, Volume 86, Issue 11 (1998).

[6] Dan Ciresan, Ueli Meier, Jrgen Schmidhuber "Multi-column deep neural networks for image classification." IEEE Conference on Computer vision and pattern recognition (CVPR), Pages 3642-3649 (2012)

[7] Dan Claudiu Cirean, Ueli Meier, Luca Maria Gambardella, Jrgen Schmidhuber. "Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition" arXiv preprint arXiv:1003.0358v1 (2010)

[8] Ahmad-Montaser Awal, Harold Mouchere, Christian Viard-Gaudin. "Towards handwritten mathematical expression recognition." 10th International Conference on Document Analysis and Recognition(ICDAR), 2009

[9] " Convolutional Neural Networks for Visual Recognition" Stanford Course CS231n Page.

[10] "Wikipedia article on HOG Features" HOG.