Recognition of Handwritten Mathematical Symbols with PHOG Features

Nicolas D. Jimenez,Lan Nguyen

***Abstract*— Converting handwritten formulas to LaTex is a challenging machine learning problem. An essential step in the recognition of mathematical formulas is the symbol recognition. In this paper we show that pyramids of oriented gradients (PHOG) are effective features for recognizing mathematical symbols. Our best results are obtained using PHOG features along with a one-against-one SVM classifier. We train our classifier using images extracted from XY coordinates of online data from the CHROHME dataset, which contains 22000 character samples. We limit our analysis to 59 characters. The classifier achieves a 96% generalization accuracy on these characters and makes reasonable mistakes. We also demonstrate that our classifier is able to generalize gracefully to phone images of mathematical symbols written by a new user. On a small experiment performed on images of 75 handwritten symbols, the symbol recognition rates is 92 %. The code is available at: `https://github.com/nicodjimenez/hand2tex`**

INTRO

Most previous research on mathematical formula recognition has focused on *online* data, i.e. the writer's strokes are recorded on a computer or tablet as a sequence of xy coordinates. Using such datasets presents many advantages when trying to recognize full formulas. Most importantly, strokes can be individually labelled using a GUI based data collection engine. This makes the location of each symbol easy to parse for a learning algorithm. The CHROHME dataset (downloadable at: `http://www.isical.ac.in/~crohme/`), which has become a standard for mathematical equation recognition, uses online data.

On the other hand, the restriction of mathematical formula recognition to online data limits the usefulness of a mathematical formula recognition system. Transcribing LaTex expressions from handwritten sources can be a painful experience even for seasoned veterans. It would be very desirable for university students to be able to take pictures of their homework using their phone and have an engine return a typeset document. Potentially, such an engine could also automate mathematical note taking in classrooms so that students can focus on the content without the distraction of copying formulas.

In this paper we take a first step towards fulfilling such an objective. We show that PHOG features used in conjunction with a linear SVM classifier is a highly effective approach to recognizing mathematical symbols extracted from online data. We then present encouraging preliminary results of an OpenCV based pipeline that uses images taken from a phone.

DATA PREPARATION

We use the CROHME2012 dataset (downloadable at: `http://www.isical.ac.in/~crohme/`) for our project, which has become a standard for mathematical equation recognition. This dataset contains 22000 characters and 1400 equations. Since the data is "online", it provides a list of xy coordinates drawn by the user for each symbol as well as the corresponding labels. Each symbol is comprised of multiple strokes. The dataset is given in InkML format, which is an XML variant. A description of the dataset was published in [6].

In order to create images from these XY coordinates, intermediate XY coordinates were first linearly extrapolated from consecutive XY pairs. A properly scaled bitmap was then created from these XY coordinates and scaled to fit in the middle 45 pixel by 45 pixel subbox of a 50 pixel by 50 pixel box. We do this so that if the strokes are smoothed or dilated, the expanded symbol will still be contained in the larger 50 pixel by 50 pixel box. The symbol was centered in the 45 pixel by 45 pixel subbox via its own bounding box. We note that in some cases it may be beneficial to center the symbol via its center of mass [4]. These data preprocessing steps are shown in Figures 1 - Figures 3. Finally, we apply a Gaussian smoothing filter with $\sigma = 1.0$ to the pixels and then normalize the pixel intensities to norm 1. This process results in images such as the ones seen in Figure . In this paper, we do not consider image deskewing, another common normalization step considered in other papers.

We note that certain symbols such as $\sin, \cos, \tan$ were excluded from this analysis. Thus, we consider a total of 59 symbols.

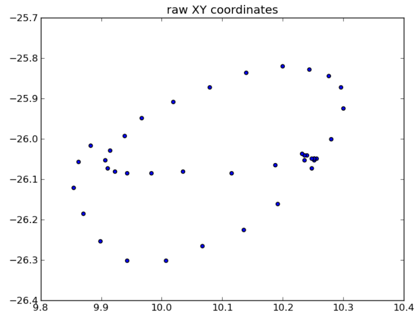


Fig. 1. Raw XY coordinates for θ example

RECOGNITION

We explored three different feature representations for the symbol images.

The first feature representation we consider is simply the raw pixel intensities. To represent this as a feature

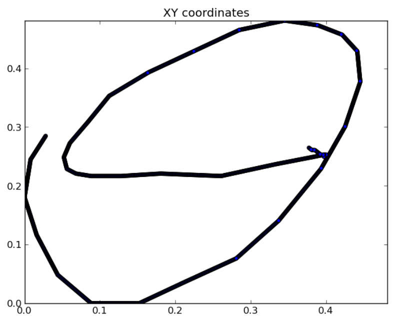


Fig. 2. Linearly interpolated XY coordinates for θ example

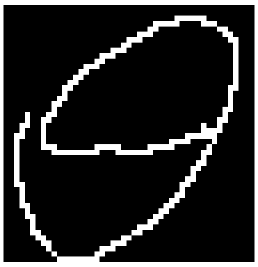


Fig. 3. Bitmap created from interpolated XY coordinates. The symbol is centered in a 45 by 45 bitmap via its bounding box

vector, the bitmap representing the image in concatenated into a single vector. Since we are using 50×50 pixels to represent the images, the total size of the feature vector is 2500.

The second feature we explore is histograms of oriented gradients (HOG). The idea behind HOG feature [3] is that an image can be represented by a localized distribution of intensity gradients. Thus HOG features appear well suited to handwriting recognition, as the intensity gradients in a image of a handwritten symbol will be due to the strokes themselves. In particular, HOG is implemented by dividing an image into small connected regions known as cells. For each cell a histogram of oriented gradients is created by compiling the gradients of each pixel in a particular cell and using these gradients to vote in an edge orientation bin. In this study we use 8 edge orientations. The histograms for all cells is then concatenated into a single feature vector. An important parameter affecting this computation is the *cell size* which is simply the pixel dimensions across which pixel gradients will be pooled. For the HOG features, we found that a cell pooling size of 5×5 worked best, We used an implementation of HOG features which is part of the scikit-image python library. This cell pooling size resulted in feature vectors of length 800.

The HOG features are easily generalized to pyramids of HOG features (PHOG) [5]. For PHOG features, HOG feature maps using different cell sizes are concatenated into a single feature vector. This method enables us to compute HOG maps at different image scales. We found that the PHOG features worked best when using we were combining cell pooling sizes of 5×5, 10×10 and 20×20. This resulted in feature vectors of length 1032

We test a linear one-vs-one SVM classifier as well as a k-NN classifier as implemented by the scikit-learn python library [7]. To evaluate the performance of our SVM classifier, we perform three fold cross validation on the data using a 2% hold out set. The results are shown in Table I. Since PHOG features exhibit superior performance, we compute the generalization error as a function of the training set size using PHOG with a linear SVM as well as a k-NN classifier and three fold cross validation; these results are shown in Figure 4.

	Accuracy (%)
Raw pixels & SVM (45×45 pixels)(C = 1000)	85
HOG & SVM (cell size $= 5 \times 5$ pixels)(C=1)	93
PHOG & SVM(cell sizes $= 5^2, 10^2, 20^2$ pixels) (C=1)	96
PHOG & k-NN (k=10)	91

TABLE I

TEST SET ACCURACY WITH THREE FOLD CROSS VALIDATION USING DIFFERENT REPRESENTATIONS OF THE SYMBOL BITMAPS. FOR ALL SVM'S, THE ONE VS ALL APPROACH WAS USED.

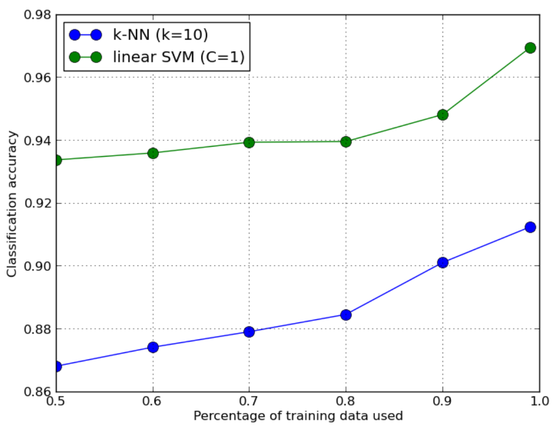
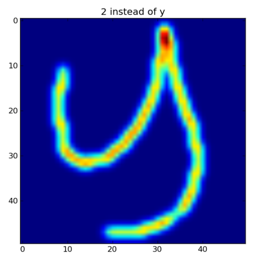


Fig. 4. Classification accuracy using PHOG features when using different training set sizes.

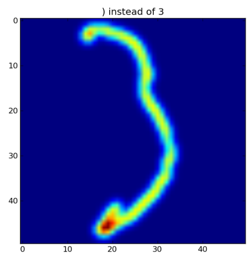
ANALYSIS

From Figure 4 we see that there is a strong trend towards greater performance as the training set becomes larger. This trend is apparent even as nearly all the data is used for training the classifier. Thus the generalization error does not appear to have approached an asymptotic limit. This suggests that incorporating more data would continue to improve the performance of our classifier. It is also unclear from this Figure whether one would expect k-NN or linear SVM to have better asymptotic performance, although k-NN has stronger theoretical asymptotic guarantees and scales more naturally to massive training sets.

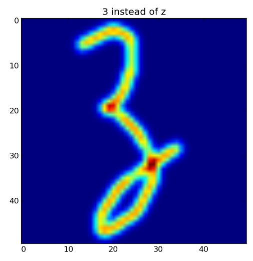
In Figures we show the first few test set errors on a particular iteration of the cross validation trials used to generate the above table for the SVM classifier. It appears that the classifier generally makes *reasonable* mistakes.



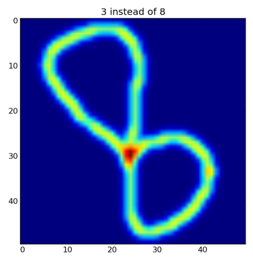
(a) 2 instead of y



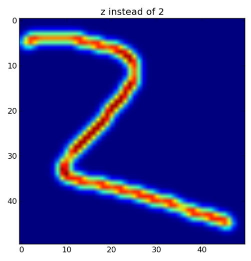
(b)) instead of 3



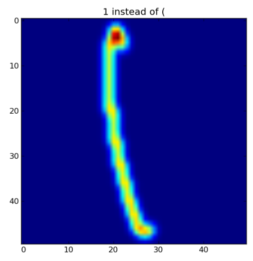
(c) 3 instead of z



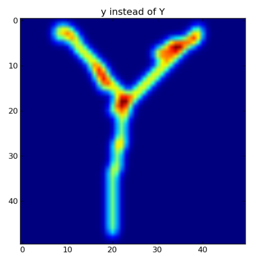
(d) 3 instead of 8



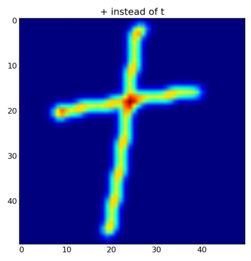
(e) z instead of 2



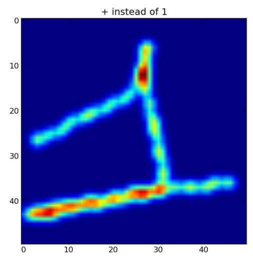
(f) 1 instead of [



(g) y instead of Y



(h) $+$ instead of t



(i) $+$ instead of 1

Fig. 5. Examples of mistakes made by the algorithm.

Application to Phone Camera Images with New User

To see how well our best PHOG based classifier generalizes to phone images, we generate 75 handwritten symbols by sampling randomly and with replacement from the the symbols in the test data. The symbol strings were written down on white paper using a black fountain pen and pictures were taken using a HTC Sensation smart phone. The person writing down these equations did not contribute any training examples to the dataset. We then apply the following steps using the OpenCV [1] and scikit-image [2] open source libaries to segment the images into constituent symbols:

- We blur the image to remove noise (OpenCV2 function: GaussianBlur)
- We apply an adaptive threshold to binarize the image (OpenCV2 function: adaptiveThreshold)
- We dilate the image to ensure symbols are fully connected (OpenCV2 function: Dilate)
- Segment image by finding connected components (OpenCV2 function: findContours)
- Skeletonize the image (scikit-image function: morphology.Skeletonize)

We perform this last step so that the symbol examples taken from pictures can be properly compared to the symbols extracted from the CHROHME dataset, which indeed consist of 1-pixel-thick skeletons before the application of the Gaussian filter and normalization. The steps are illustrated in Figures 6 - 8. We were able to easily attain 100% segmentation accuracy with the images taken although this task was substantially facilitated by the fact we use clean images with similar lighting and writing conditions. Our symbol classifier attained 92% accuracy when classifying the 75 symbols making up our dataset. Although this is a small sample, the results suggest that PHOG features are a viable method for recognizing mathematical symbols in images as well. We make the following observations.

- Some training data from the person writing the symbols would probably improve the classification accuracy. It is reasonable to expect the classifier would have better performance if it could have samples of the current writer's handwriting.
- More data seems necessary to build a robust system. Given the great variety of ways a single symbol may be written, a larger training set would be necessary to train a classifier able to recognize symbols in the variety of ways they may be written.

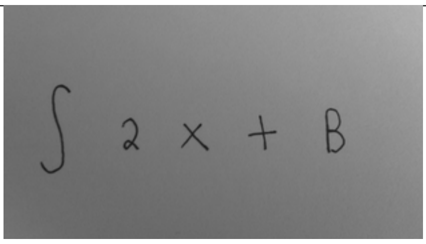


Fig. 6. Raw image of equation taken with a HTC smartphone camera representing $\int 2x + B$

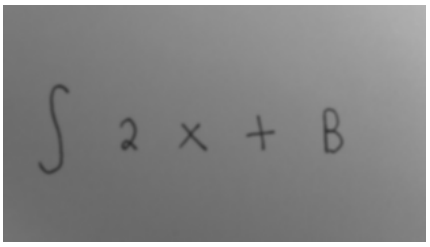


Fig. 7. Image after blurring step.

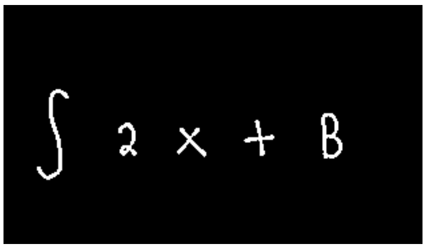


Fig. 8. Image after adaptive thresholding step.

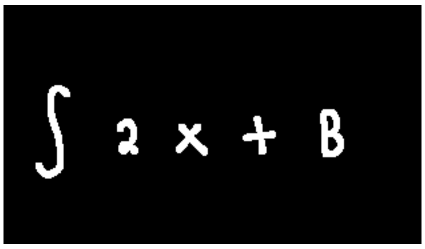


Fig. 9. Image following dilation step.

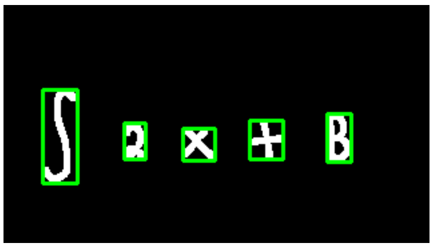


Fig. 10. Image segmentation as computed by OpenCV2 findContours function.

Conclusion

Using PHOG features in tandem with a one against one linear SVM classifier appear to be effective method of classifying mathematical symbols. Our best results are obtained on online data such as the data that makes up the CHROHME dataset, where the generalization error is approximately 96%. When the classifier is used on symbols in images of a new user's handwriting, the generalization performance drops to approximately 92%.

In future work, image deskewing and incorporating new features and classifiers should be explored. While PHOG features appear to work well, classification accuracy could probably be improved if other feature maps were considered simultaneously. For example, we have observed that a common error is the confusion between 1 and (or [(see Figures). It seems likely that using the central moments of the image would be useful information for disambiguating between these symbols. The central moments are invariant to bounding box repositioning and are able to properly characterize the orientations of such symbols in the presence of noise. On the other hand, PHOG feature vectors may be very similar for a bitmap of a 1 and a), as these will contain much overlap.

In addition, modern unsupervised feature learning methods such as restricted boltzmann machines and sparse autoencoders should be explored. Recently, such methods have been shown to exhibit classification performance that is superior to hand crafted features such as HOG in many visual recognition tasks. It would be interesting to explore HOG variants with parameters learned in an unsupervised manner.

Overall, we noticed the classifier was quite sensitive to perturbations of the positioning of symbols within their bounding box. This is highly undesirable. To circumvent this, many authors expand the training set by "jittering", i.e. applying random small shifts of images in the training set within a single bounding box. This strategy should be explored in future work on mathematical equation recognition. Another approach may be to use a distance measure between HOG feature vectors that is invariant to small translations.

The next step in building a full system for recognizing handwritten mathematical equations is to infer the intended LaTex expressions from the spatial relationships between the predicted symbols. Approaches using 2D context free grammars are presented in [?] and [?] and will be explored in future work.

References

[1] Opencv - open source computer vision. `http://opencv.org/`.

[2] scikit-image - image processing in python. `http://scikit-image.org/`.

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Subhransu Maji and Jitendra Malik. Fast and accurate digit classification. Technical Report UCB/EECS-2009-159, EECS Department, University of California, Berkeley, Nov 2009.

[6] 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 *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 811–816. IEEE, 2012.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.