Handwritten Mathematical Expression Recognition

Group 27 - Abhyãsa

Abhishek Gunda
abhigun@iitk.ac.in

Krishna Karthik
jkrishna@iitk.ac.in

Harsha Nalluru
harshan@iitk.ac.in

Aravind Reddy
arareddy@iitk.ac.in

Gowtham Prudhvi
gowthamp@iitk.ac.in

1 Problem Statement

Handwritten Mathematical Expression Recognition(HMER):
Converting handwritten mathematical expressions to their LaTeX code.

In this project, we aim to build an application which is capable of taking a handwritten mathematical expression as input(in our custom scratchpad) and outputs the LaTeX code for the corresponding expression.

2 Problem Motivation

HMER is an important and challenging problem in the field of pattern recognition. As LaTeX is the de-facto mathematics rendering tool, we naturally chose it as the language of choice for expressing our recognized mathematical expressions.

Having an efficient HMER tool is useful in several situations:

- Simplifying input of mathematical notation to computers.
- Digitizing of handwritten documents.
- Assisting for visually impaired people in reading mathematical documents.
- Online education using touch screen based devices like tablets.

3 Existing Work

- Tremendous amount of work has been done to address the HMER problem. A conference exists solely for a larger case of the HMER problem - International Conference on Frontiers in Handwriting Recognition(ICFHR). Check out **ICFHR 2016**
- Dutt et.al[1] used CNNs for recognizing digits in the MNIST data set.
- Sliding window techniques for label identification - Thomas Lech, Math to Latex Blog[3]
- Contour Identification - OpenCV[4]

4 Novel Contribution

Data pruning

Chose the 10 best images from each class manually and then trained CNN using them. Tested for the images in kaggle dataset and selected around 1000 correctly classified images per class

Processing the contours

Used various techniques like Skeletonize(), medianBlur(), gaussian() and pre-processed the image to make it similar to those of the images in the training dataset.

Latexification

When we observe the image left to right, a new character can assume many positions like superscript, subscript, on the same level etc.

To handle this, we have considered characters to be nodes, with top, bottom, next, and parent attributes and we recursively look for its parent, i.e., the level in which new character is going to be present.

We have implemented a spanning-tree structure for the handwritten expressions to preserve the inherent structure of the expression.

Once we get labels for the characters, the corresponding positions are stored in a spanned-tree over the mathematical expression, then we can obtain expressions in latex.(Clearly explained in the Methodology section)

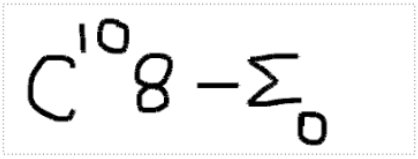
Sketchpad

Built a sketchpad using HTML, Java script. Using this, we can write mathematical expressions through the mouse.

Sketchpad : http://krishnakarthik.me/abhyasa/

5 Methodology

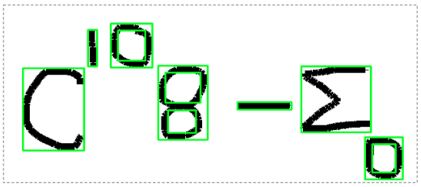
Write a mathematical expression in the sketchpad(http://krishnakarthik.me/abhyasa/) and save the image



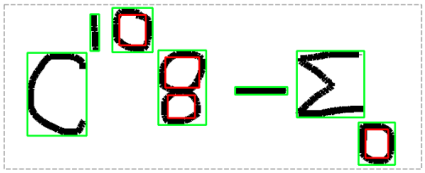
Feed the image as an input to obtain its corresponding latex expression

Finding Contours

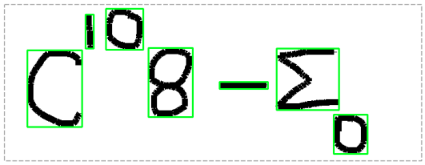
Using cv2.findContours, we obtain the contours



We ignore the boxes with more than 60% of the area overlapping with other boxes



The final contours are as follows



Processing the boxes

Cropping using PIL.Image's crop function
Squaring the image by appending whitespace to match both the height and width
Resize to (45x45) using cv2.resize()
Skeletonize() from skimage.morphology for thinning

Finding the Labels

Call the classifier(obtained by training CNN, given below $\rightarrow$ Machine Learning Model) and get the label with the highest probability

Re-labeling

$!, i, =\rightarrow$ these symbols have multiple components, for these we have increased the height of the contour by 25% above and below, and then process the contour again using the above techniques, and then classify the new boxes

Latexification

When traversed through the boxes, sorted along the X-coordinate, we observe that, a new character can assume many positions like superscript, subscript, on the same level etc.

To handle this, we have considered characters to be nodes, with top, bottom, next, and parent attributes.

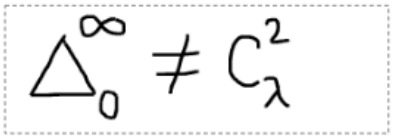
And we recursively look for its parent, i.e., the level in which new character is going to be present.

We have implemented a spanning-tree structure for the handwritten expressions to preserve the inherent structure of the expression.

Once we get labels for the characters, the corresponding positions are stored in a spanned-tree over the mathematical expression, then we can obtain expressions in latex.

Example being executed

Input



Output $ \Delta_{0}^{\infty}\neqC_{\lambda}^{2} $

```
harsha@inspiron:~/Desktop/7th_sem/cs771/abhyasa/digitRecognition$ python3 track2.py delta3.png
Using TensorFlow backend.
/home/harsha/miniconda3/lib/python3.6/importlib/_bootstrap.py:205: RuntimeWarning: compiletime ve
rk.fast_tensor_util' does not match runtime version 3.6
  return f(*args, **kwds)
2017-11-26 15:41:01.333502: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU support
 not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
Label  0 ->      \Delta
Label  1 ->      \infty
Label  2 ->      0
Label  3 ->      \neq
Label  4 ->      C
Label  5 ->      \lambda
Label  6 ->      2
Before locate
About to locate
Inside if
Inside if1
About to print
$ \Delta_{0}^{\infty}\neqC_{\lambda}^{2} $
harsha@inspiron:~/Desktop/7th_sem/cs771/abhyasa/digitRecognition$
```

Sketchpad : http://krishnakarthik.me/abhyasa/

Github Repo : https://github.com/krishnakarthik1309/abhyasa

Presentation Link

Machine Learning Model

We preferred CNN over others(SVM, kNN, RFC) to classify the dataset. This is because the accuracy on training dataset and prediction on MNIST digits dataset is more for CNN compared to other algorithms.

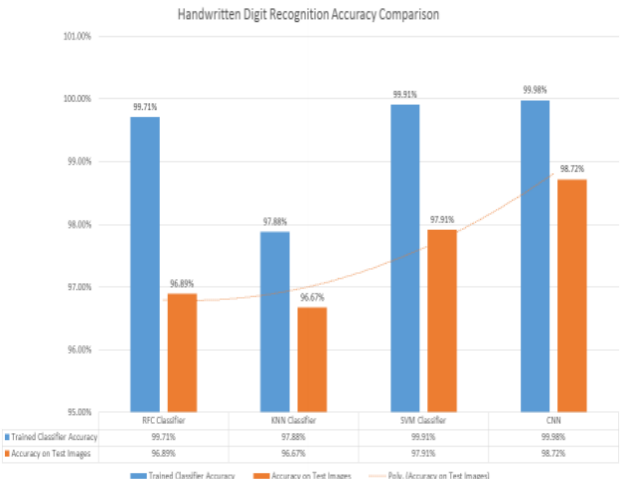


Figure 1: Handwritten digit recognition by Dutt et.al[1]

Also we trained and tested on out dataset(Kaggle dataset) with CNN and SVM algorithms. CNN performed better with an accuracy of 94%(heldout validation(validation set: 10% of dataset)). The basic structure of CNN:

- **Input Layer:** Data(Image) is converted to 45 x 45 pixel value of the image.

- **Building Network Architecture:**
  a) Convolution Layer:
  Window slider over input and gives pixel with maximum intensity as output.

  b) ReLU Function:
  ReLU function is chosen as the activation function here so that it reduces the likelihood of the vanishing gradient and avoids sparsity. This way we do not lose important data, but can get rid of redundant data like lots of 0s in the pixel matrix.

  c) Pooling Layer:
  It pools all the pixels obtained from previous layer and forms a new image(pixel matrix) of a smaller size.

- **Fully Connected Layer:** Computes the score classes using Softmax activation function.

- **Optimizer:**The loss function we used is a Categorical Cross Entropy loss function.We used Stochastic Gradient Descent as our optimizer for reducing the loss.SGD uses Nesterov Accelerated Gradient descent(NAG) with learning rate, momentum and decay(for learning rate) set to default values as it seemed to work well.We tuned other parameters i.e, Batch Size and Number of Epochs to get the values we used finally.

The flow of CNN is shown in the following figure.

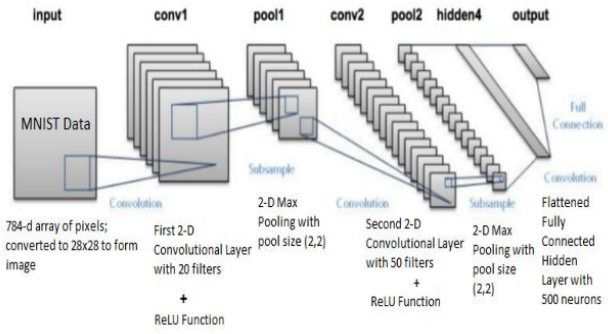


Figure 2: Handwritten digit recognition by Dutt et.al[1]

6 Experimental details

6.1 Data

Training Dataset

Kaggle Dataset(https://www.kaggle.com/xainano/handwrittenmathsymbols)

82 Classes of math symbols with 2000 - 3000 images(45x45) for each class

Data Pruning

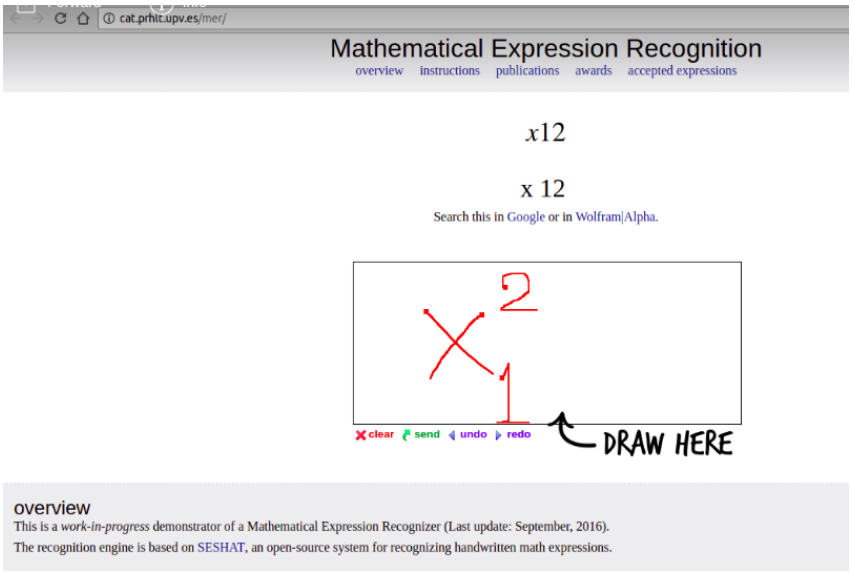
Chosen the best 10 images from each class → trained CNN and tested for the good images in kaggle dataset and eventually increased to 1000 images per class (roughly). This is done because the dataset has some conflicting characters and few bad images.

6.2 Benchmarks

On our data set, we trained and tested with CNN and SVM. CNN performed better with an accuracy of 94%.

Comparing with one tool (http://cat.prhlt.upv.es/mer/)

Our application performed better on some inputs we tested.



For example from the above figure, it seems to be that they use techniques similar to sliding window, and do not give much preference to the positioning of the characters along the vertical-axis.

We performed better in converting to latex expressions as we have implemented a spanning-tree structure for the handwritten expressions to preserve the inherent structure of the expression.

6.3 Tuning

We tuned our two main Hyper parameters Batch Size and Number of Epochs.

Tuning Batch Size

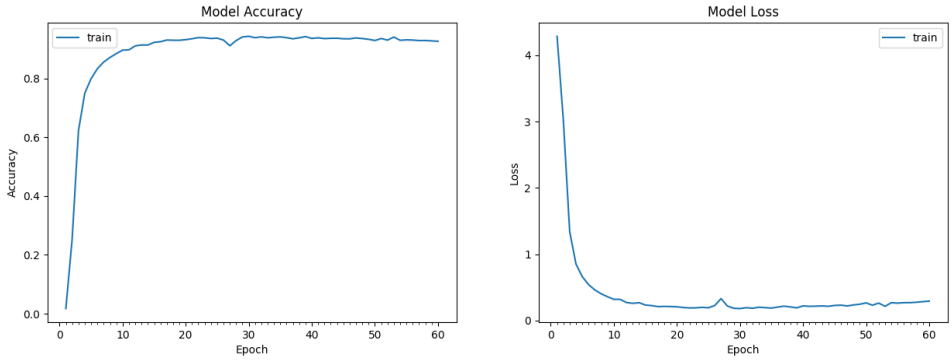
We fixed Number of Epochs=10 and then varied Batch Size checking for the best accuracy after 10 epochs.We achieved the best accuracy at Batch size=64.

Batch Size	Number of Epochs	Accuracy
2046	10	4.46%
1024	10	39.28%
256	10	88.08%
128	10	91.28%
64	10	91.75

Tuning Number of Epochs

After fixing Batch size=64 we varied Number of Epochs to get the best value for it.The following figure shows the plot of Accuracy vs Epochs.

Figure 3: Accuracy vs Epochs and Loss vs Epochs graphs



As we can see that after Number of Epochs=20 accuracy doesn't change much,

so we set Number of Epochs=10 and Batch Size=64.

6.4 Tools/Libraries used

python3

- Skimage
- sklearn
- Tensor Flow
- Keras
- PIL
- OpenCV2

6.5 Links

Sketchpad : http://krishnakarthik.me/abhyasa/

Github Repo : https://github.com/krishnakarthik1309/abhyasa

Presentation Link

7 Future work

- Adding more characters(more math symbols, devanagari symbols) to dataset. At present it contains 82 different characters.
- Taking input from paper i.e, image of math expression written on paper as input.
- Can attempt the **ICFHR 2016 CROHME** competition(It is most probably going to be repeated in ICFHR 2018, CROHME is being conducted in almost every ICFHR).

References

[1] Handwritten Digit Recognition Using Deep Learning *http://ijarcet.org/wp-content/uploads/IJARCET-VOL-6-ISSUE-7-990-997.pdf*.

[2] Kaggle dataset *https://www.kaggle.com/xainano/handwrittenmathsymbols*

[3] T. Lech *Math to Latex Blog http://blog.mathocr.com/*

[4]OpenCV