Handwritten Mathematical Symbol Classification

Shivam Basia
Dept of Computer and Information Sciences
University of Florida
Gainesville, US
shivam.basia@ufl.edu

Divya Upadhyay
Dept of Computer and Information Sciences
University of Florida
Gainesville, US
divyaupadhyay@ufl.edu

Shruti Shivani
Dept of Computer and Information Sciences
University of Florida
Gainesville, US
sshivani@ufl.edu

Teja Harshini Matukumalli
Dept of Computer and Information Sciences
University of Florida
Gainesville, US
t.matukumalli@ufl.edu

***Abstract*—Classification of handwritten mathematical symbols is a challenging task due to the similarities between symbols. To handle this task, this application uses CNN model to classify ten different mathematical symbols. The classification is done for both images with individual symbols and equations. The classifier uses contour detection and bounded rectangles to classify symbols in an equation. Along with classification, the number of symbols present in an equation are also identified. The results show that the model classifies the symbols with a higher accuracy.**

***Index Terms*—handwritten mathematical symbol classifier, CNN, Contour Detection, Bounding Rectangle, Piecewise functions**

I. Introduction

The Handwritten Mathematical Symbol Classifier is an image reader that identifies the mathematical symbols in a given image. This classifier identifies ten different symbols and classifies the rest of the symbols as unknown or undefined. The ten symbols identified by this classifier are:

- The letter x
- Square Root
- Plus sign
- Negative sign
- Equal sign
- Percentage
- Partial differential
- Product symbol
- pi
- Summation

In addition to an image, the classifier also identifies the symbols in any given equation. This equation is given to the classifier as an image and can be handwritten or typed. The labels of the symbols are given in the order of their appearance in the equation. Any unknown symbol is labelled as "*u*". For ease of use, the images to be classified can be consolidated in a file and given as input. The final classifications of the images are also presented in a file.

A. Literature Review

Upon a detailed review of multiple researches done to classify offline mathematical symbols, the following approaches are identified.

1) Support Vector Machines: Some experiments by researchers to classify offline mathematical symbols involved the usage of Support Vector Machines. To classify multiple symbols, a non-linear decision surface was used. To implement this, the feature space is projected onto a higher dimensional space and then a linear separating hyperplane is applied there. For the multi-class classification, one-by-one and DAG-SVM (Directed Acyclic Graph SVM) are used [1] [3] [4] [5].

2) LeNet: Some researches used the LeNet architecture to classify the offline mathematical symbols. Using ReLU as the activation function for a higher accuracy, a seven layered CNN model was built with two convolution layers, two pooling layers and three fully connected layers. Softmax activation function was used at the output layer to determine the label of the input symbol [2].

II. Methodology

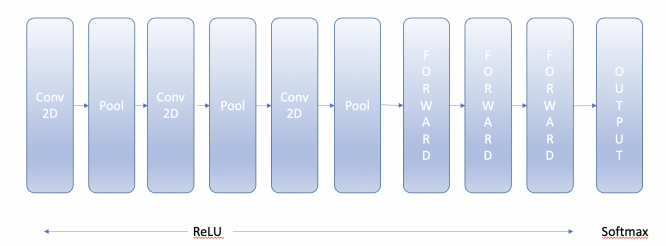
The Mathematical Symbol Classifier handles the classification in different ways depending on the input images. The approaches for each kind of input are as follows:

A. Individual Symbol Classification

For an accurate and a speedy classification of the symbols, this classifier uses CNN algorithm. The input images initially undergo pre-processing where they are first resized and then normalized. These pre-processed images, after being split into training and testing data, are then converted to tensors and divided into batches. The batches of images are given as input to the CNN model.

The CNN model used here consists of three 2D convolutional layers, three pooling layers and three fully connected

layers. ReLU activation function is used for the lower layers and softmax activation is used for the output layer. The architecture of this model is as follows:



Using the output of this model, the error is calculated. Depending on the error, the parameters used for the model are updated. The model is re-executed for the updated parameters. This process is repeated till the required number of epochs are executed. The overall training and validation accuracy are computed. The performance metrics used to evaluate this model are:

- Confusion matrix
- Accuracy
- Loss

The trained model is then stored for further usage.

B. Symbol Classification In An Equation (Extra Credit)

In addition to the above classification, this classifier also identifies the symbols present in a given equation. These equations are given as images to the model. The model uses piecewise functions methodology to handle these images.

The classifier initially detects contours in the image to identify the number of symbols in the equation. It then uses bounded rectangles to convert the image into multiple rectangles where each rectangle has a symbol. These rectangles are then trained in a linear fashion to figure out the labels of the symbols. Then these rectangles are merged back into one. The output would be the set of labels of all the symbols in the order that they are present in the equation. Any unknown symbol would be labeled as ”u”.

III. Implementation

The classifier is trained using the above approach. The classifier implements the above logic using the following steps of machine learning:

A. Data Pre-Processing

The data set provided consists of 9032 images of ten handwritten mathematical symbols and their labels. All the images were not accurately labelled. The first step of pre-processing involved manually correcting the incorrect labels of the images. The labels of unknown symbols are left as they are. The corrected dataset undergoes the following pre-processing:

- The images are resized from 300 x 300 to 150 x 150 images.
- These resized images are split into training and test split using an 80-20 split.
- The training and test data are normalized using their mean and standard deviation.
- The data is then converted to tensors from numpy array.

The tensors obtained are used as the data for the model.

B. Model Execution

Once the data is pre-processed, it is divided into batches and the model is applied on the data. The hyper parameters used to train the final model are as follows:

- Pytorch is used for implementation
- Batch size is 10
- Number of epochs is 10
- Learning rate is 0.01
- Threshold is 1e-7.
- Maximum batches are 3000 for both train and validation data

Using the above hyper parameters, the CNN model is executed. The CNN model consists of the following layers:

- 2D Convolution layer with 1x11 filter and kernel size 3. Stride is 1.
- Average pooling layer with pool size 2.
- 2D Convolution layer with 11x20 filter and kernel size 3. Stride is 1.
- Average pooling layer with pool size 2.
- 2D Convolution layer with 20x30 filter and kernel size 3. Stride is 1.
- Average pooling layer with pool size 2.
- Three forward layers.

The model summary of the generated CNN model is as follows:

```
ConvolutionalNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(20, 30, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=8670, out_features=200, bias=True)
  (fc2): Linear(in_features=200, out_features=100, bias=True)
  (fc3): Linear(in_features=100, out_features=25, bias=True)
)
Layer 1 ->      Weights: 10 x 1         Bias:  10       Parameters:  20
Layer 2 ->      Weights: 20 x 10        Bias:  20       Parameters:  220
Layer 3 ->      Weights: 30 x 20        Bias:  30       Parameters:  630
Layer 4 ->      Weights: 200 x 8670     Bias:  200      Parameters:  1734200
Layer 5 ->      Weights: 100 x 200      Bias:  100      Parameters:  20100
Layer 6 ->      Weights: 25 x 100       Bias:  25       Parameters:  2525

Total Params:  1757695
```

The model initially classifies unknown symbols as class 10. Later, class 10 is reverted to unknown or "u".

C. Model Execution for Equations(Extra Credit)

If the input image consists of equations, the images are first reshaped to 300x300. Then the threshold function of opencv is applied on the images. This threshold function applies a binary threshold to the images. Then chain approximation is used to find the contours in the images.

Once contours are found, bounding rectangle is used to separate the symbols into rectangles. These rectangles are then reshaped to 150x150 and normalized. The normalized data is then converted to tensors and is given to the model. The model performs the same operations as with individual symbols.

The output is then compared with the threshold of 1e-7 and if the predicted outcome is greater than the threshold, it is considered as unknown symbol. Further, the same performance metrics are used as before on the output of this model.

D. Optimizers and Loss Functions

The output of the above models are then given to an optimizer, loss functions and the errors are computed. This model uses

- Optimizer: Adam
- Loss function: Cross Entropy Loss
- Error: Mean Squared Error

Since it is a multi-class classification, cross entropy loss function is used for the model. Further, the accuracy of the model is computed using the confusion matrix. The final output is stored in a csv file.

IV. Experiments

Prior to finalizing on the above model, a number of experiments were performed. To identify the appropriate model for the classification, multiple models were applied on the data set. They are:

- *Support Vector Machine:*SVM model was applied on the data set to classify the individual symbols. However, due to the large amount of time taken to train the model, SVM model was not considered to be the appropriate model.
- *Random Forest:*Random Forest model was applied on the data set to classify the individual symbols. Though the training speed is fairly good, the accuracy of the model was 45% which was not satisfactory.
- *Naive Bayes:*Naive Bayes model was applied on the data set to classify the individual symbols. Though the training speed is fairly good, the accuracy of the model was 14% which was not satisfactory.
- *KNN:*KNN model was applied on the data set to classify the individual symbols. Though the training speed is fairly good, the accuracy of the model was 35% which was not satisfactory.
- *Transfer Learning:*Transfer Learning approach was applied on the data set to classify the individual symbols. Though the training speed is fairly good, the accuracy of the model was not satisfactory.

Further experiments has been done by tuning the hyper parameters and modifying the learning rate values. The best possible hyper parameters and learning rate was applied to the model and are mentioned above.

V. Steps to Execute

The following modules are required to run the project(The modules are already there in hipergator):

- torch
- sklearn
- cv2
- numpy
- pandas

The project has both .py and .ipynb files for execution. The steps to execute .ipynb files are as follows:

- Open Hipergator jupyter notebook using jhub.rc.ufl.edu
- Use the NGC-PyTorch-1.9 kernel.
- Change the filename in the variables 'images' and 'labels'(which is just after the import statements)
- Execute the notebook.

The steps to execute the .py files are as follows:

- For training: python train.py images-file-name labels-file-name
- For testing: python test.py images-file-name labels-file-name
- For testing the extra credit part: python test-extra-credit.py images-file-name labels-file-name

The training model is already saved in 'saved-model.pt'. So, the test file can be directly executed. The preprocessing.ipynb consists of the pre-processing steps and the result of this file is t-train-corrected.npy which contains the data set with corrected labels.

VI. Results and Analysis

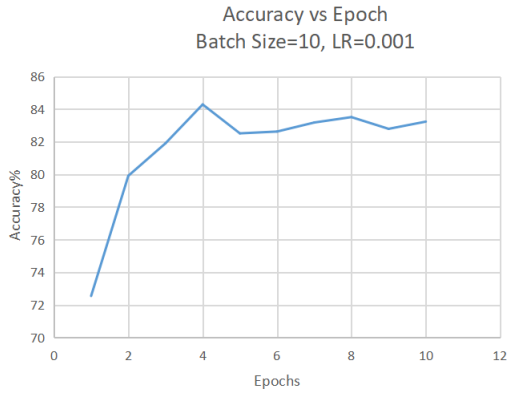
The model generates an output file 'output.csv' containing the original and predicted output for easy analysis of the model in the testing part. If the input is equations, the output file 'output-equation.csv' is generated and it contains the original file equation as well as the predicted equation. Also,'output-equation-length.csv' contains the actual number of symbols in the equation as well as the predicted number of symbols in the equation.

The training accuracy of the model is 95%. The validation accuracy of the model is 85%.

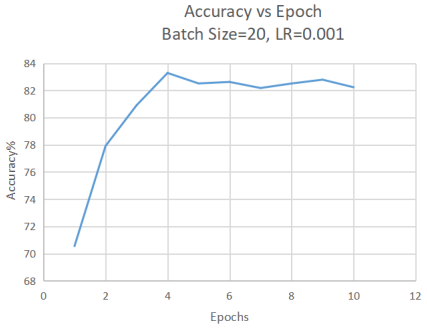
The classification report of the model is as follows:

	precision	recall	f1-score	support
0	0.90	0.95	0.92	905
1	0.97	0.98	0.97	895
2	0.97	0.97	0.97	893
3	0.96	0.95	0.96	888
4	0.96	0.95	0.95	880
5	0.95	0.95	0.95	899
6	0.96	0.96	0.96	920
7	0.92	0.93	0.92	851
8	0.91	0.89	0.90	923
9	0.98	0.95	0.96	894
10	0.80	0.70	0.75	84
accuracy			0.95	9032
macro avg	0.93	0.93	0.93	9032
weighted avg	0.95	0.95	0.95	9032

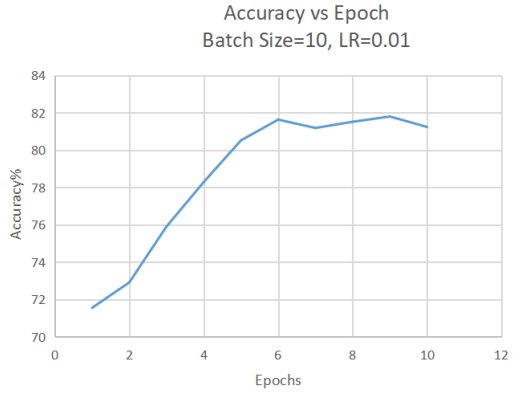
The results of the experiments with hyper parameters and learning rates are captured and the accuracies are analysed. The accuracy graph for batch size 10 and learning rate 0.001 is:



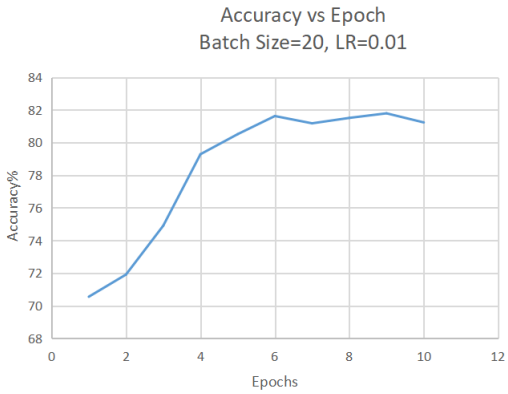
The accuracy graph for batch size 20 and learning rate 0.001 is:



The accuracy graph for batch size 10 and learning rate 0.01 is:



The accuracy graph for batch size 20 and learning rate 0.01 is:



VII. Conclusion and Future Scope

The offline mathematical symbol classifiers uses CNN algorithm to classify the ten mathematical symbols both individually and in equations. The model classifies the symbols with 95% accuracy. Various algorithms were experimented on the data set to determine the best possible algorithm. Further, various hyper paramters and learning rates were used to find the best combination of hyper parameters. The best learning rate was identified to be 0.01. The output of the model are stored in csv files for easy access.

This application can be further developed to recognize numbers and solve the equations provided in the images.

References

[1] B. Keshari and S. Watt, "Hybrid Mathematical Symbol Recognition Using Support Vector Machines," IEEE Xplore, Sep. 01, 2007. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=4377037 .

[2] A. Nazemi, N. Tavakolian, D. Fitzpatrick, C. Fernando, and C. Suen, "Offline handwritten mathematical symbol recognition utilising deep learning."

[3] C. Malon, S. Uchida, and M. Suzuki, "Mathematical symbol recognition with support vector machines," Pattern Recognition Letters, vol. 29, no. 9, pp. 1326–1332, Jul. 2008, doi: 10.1016/j.patrec.2008.02.005.

[4] S. A. Firdaus and K. Vaidehi, "Handwritten Mathematical Symbol Recognition Using Machine Learning Techniques: Review," Learning and Analytics in Intelligent Systems, pp. 658–671, Jul. 2019, doi: 10.1007/978-3-030-24318-0-75.

[5] C. Malon, S. Uchida, and M. Suzuki, "Support Vector Machines for Mathematical Symbol Recognition," Lecture Notes in Computer Science, pp. 136–144, 2006, doi: 10.1007/11815921-14.