Handwritten Equation Identification

Anubhab Bose
Anubhab Biswas
Sreeranjini TM

May 22, 2023

School of Mathematics and Statistics, University of Hyderabad, India

1 Introduction:

This project aims to recognize handwritten mathematical expressions or equations using Convolutional Neural Network. We look to provide a faster alternative for recognizing math symbols and equations instead of using type-setting systems. This problem is a classification problem because the model is trying to identify the different math symbols in an handwritten equation and present the corresponding LaTeX code for the mathematical expression.

2 Data Description:

We have considered the CROHME dataset for this project. The CROHME dataset consists of files with inkml extension representing different mathematical equations consisting of over 100 different mathematical symbols. The size of the whole CROHME file is around 55mb consisting of 3 different datasets from 3 different years (2011, 2012 and 2013). This data required some preprocessing as all the images are .inkml files so they need to be converted to a form that can be used for training or testing the deep learning model.

For preprocessing we used a pre-built inkml extractor which prepares the images for training or testing purpose and gets us the testing equations as an array of symbols. The inputs are in the form of 32x32 arrays where each array represents the pixels of each symbol. We used 2 versions of the dataset to create 3 different sets of input and labels. The first set was the training dataset which contained individual math symbols and their labels. The second set was the testing dataset, this testing dataset contains the individual math symbols as well as their labels. Third set was another testing dataset which contained a full mathematical expression and the label representing that expression.

3 Methodology:

We have trained our model using CNNs and 2-layer fully connected neural network. The architecture for the deep learning model is presented as below:

$$[\boldsymbol{ConvolutionalLayer1}]$$
$$32\ filters,\ 5\ kernel\ size,\ 1\ stride,\ SAME\ padding\ and\ RELU\ activation$$
$$\downarrow$$
$$[\boldsymbol{ConvolutionalLayer2}]$$
$$32\ filters,\ 5\ kernel\ size,\ 1\ stride,\ SAME\ padding\ and\ RELU\ activation$$
$$\downarrow$$
$$[\boldsymbol{MaxPoolLayer1}]$$
$$2\ pool\ size,\ 2\ stride$$
$$\downarrow$$
$$[\boldsymbol{ConvolutionalLayer3}]$$
$$128\ filters,\ 5\ kernel\ size,\ 1\ stride,\ SAME\ padding\ and\ RELU\ activation$$
$$\downarrow$$
$$[\boldsymbol{ConvolutionalLayer4}]$$
$$128\ filters,\ 5\ kernel\ size,\ 1\ stride,\ SAME\ padding\ and\ RELU\ activation$$
$$\downarrow$$
$$[\boldsymbol{MaxPoolLayer2}]$$
$$2\ pool\ size,\ 2\ stride$$
$$\downarrow$$
$$[\boldsymbol{Flatten}]$$
$$\downarrow$$
$$[\boldsymbol{DenseLayer1}]$$
$$RELU\ activation\ acting\ as\ a\ hidden\ layer\ of\ 500\ neurons$$
$$\downarrow$$
$$[\boldsymbol{Dropout}]$$
$$0.3\ rate$$
$$\downarrow$$
$$[\boldsymbol{DenseLayer2}]$$
$$SOFTMAX\ activation\ and\ output\ size\ equal\ to\ the\ number\ of\ classes,\ 101$$

There is one training function and two testing functions. The training function takes in the training dataset, batches the input and runs back propagation to train the model. The first testing function focuses on testing the model's accuracy on individual mathematical symbols. The second testing function focuses on testing the model's accuracy on a full mathematical expression. A brief theoretical review of the methods used in the architecture described above is discussed below:

Convolutional Neural Networks or CNNs are a type of deep learning network which is widely used to process image/video data. They are designed to recognize patterns in pixel data using various learnable filters which are applied using convolutional layers. Each of these layers takes 2D images as input, applies masks and outputs a feature map which extracts a particular feature in the input, performing an operation similar to convolution in mathematics.

CNNs also contain pooling layers and fully connected layers. Pooling layers are inserted after each/a few convolutional layers which reduce the dimention of the feature maps. The benefits of pooling is that it reduces the no. of parameters to be computed, makes the network robust to small variations in the input image and gives more importance to prominent features. A most commonly used pooling in CNNs is max pooling which tend to preserve the most important features in the input image. In max pooling, a window of pre-defined size (eg: 2×2), strides over the feature map and it replaces that region with the maximum value of the feature map inside the window.

We know that at each neuron, each input value is multiplied by a corresponding weight and then a bias term is added. Then it is passed through an activation function to introduce non-linearity, to enable the network learn complex patterns. Commonly used activation functions are sigmoid function, ReLU, tanH etc. We are using ReLU (Rectified Linear Unit) in this network because it is very straightforward, easy to compute and has been proven to be practically very efficient. What ReLU does is that keeps the positive values as they are and replaces all the negative values by zero.

i.e., $f(x) = max(0, x)$

Since we are using the CNN for classification purpose, we flatten the output from the convolutional layers and then pass it through a few fully connected layers. In the output layer, we use the softmax activation function . The no. of neurons in the output layer will be the no. of distinct classes to which we are classifying the images. The softmax activation function takes a vector of input values and produces a probability distribution over the possible output classes, It is defined as

$f(x_i) = \frac{e^{x_i}}{\sum\limits_j e^{x_j}}$

Thus, at the output layer, the network classifies the input image into the class which has the highest probability. To make the network learn, we use a loss function which gives a measure of how well the network is functioning. During thre training phase, we are using labelled data, i.e., we already know the true class of the input data. Then we compare the output predicted by the network to its true value, and the performance of the network is quantified by the loss function. This is then propogated back to the previous layers upto the input layer, and the weights are updated so as to minimize the loss function. In the widely used backpropogation algorithm, we calculate the gradient of the loss function with respect to each of the weights, using the chain rule and then update the parameters using an optimization technique such as gradient descent. In a multi-class classification problem like ours, a commonly used loss function is categorical cross entropy. It measures the dissimilarity between the true probabability distribution and the predicted probability distribution over the classes. It is defined as:

$L = -\sum\limits_i y_{true,i} * log(y_{pred,i})$

By taking the negative logarithm of the predicted probability, it encourages the network to assign weights in such a way that a high probability is given to true class.

Since our class labels are categorical, we have to use some kind of embedding to convert them to numerical. The one that we have used is one-hot encoding which creates a binary feature for each label. If we have total n labels, this creates a vector of size n for each label with the value 1 in its corresponding position and 0 in all other positions.

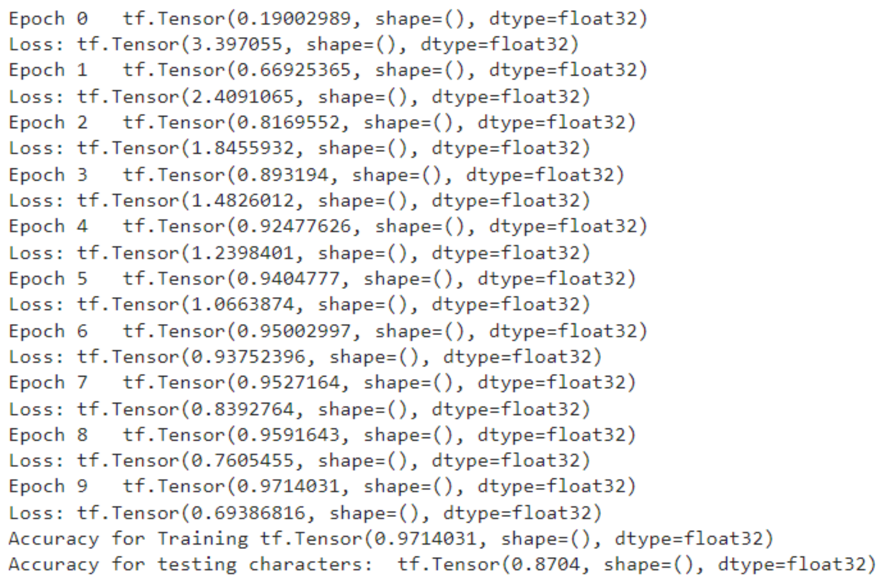
When we are using a small network to train a large network, we encounter an important problem of overfitting. A simple way to tackle this is adding a dropout layer. This layer simply cuts the connections between some of the the neurons to the next layer, thus eliminating their contribution. If we add a dropout layer with $probability = 0.3$, 30% of the neurons will be randomly dropped during each iteration. This prevents the network from learning any statistical noise present in the sample.

4 Results:

The performance of the model trained over 10 epochs is enlisted as below:

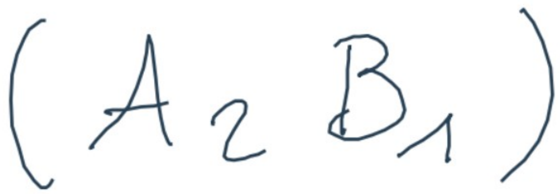
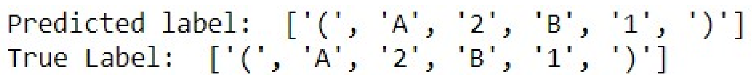
- **Training Accuracy:** 97.14%
- **Individual Symbols Accuracy:** 87.04%
- **Expression Accuracy:** 84.91%

The improvement of accuracy of the model on the training set over the 10 epochs is observed as below:

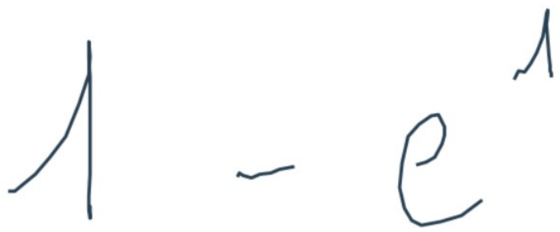
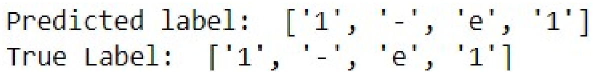


```
Epoch 0   tf.Tensor(0.19002989, shape=(), dtype=float32)
Loss: tf.Tensor(3.397055, shape=(), dtype=float32)
Epoch 1   tf.Tensor(0.66925365, shape=(), dtype=float32)
Loss: tf.Tensor(2.4091065, shape=(), dtype=float32)
Epoch 2   tf.Tensor(0.8169552, shape=(), dtype=float32)
Loss: tf.Tensor(1.8455932, shape=(), dtype=float32)
Epoch 3   tf.Tensor(0.893194, shape=(), dtype=float32)
Loss: tf.Tensor(1.4826012, shape=(), dtype=float32)
Epoch 4   tf.Tensor(0.92477626, shape=(), dtype=float32)
Loss: tf.Tensor(1.2398401, shape=(), dtype=float32)
Epoch 5   tf.Tensor(0.9404777, shape=(), dtype=float32)
Loss: tf.Tensor(1.0663874, shape=(), dtype=float32)
Epoch 6   tf.Tensor(0.95002997, shape=(), dtype=float32)
Loss: tf.Tensor(0.93752396, shape=(), dtype=float32)
Epoch 7   tf.Tensor(0.9527164, shape=(), dtype=float32)
Loss: tf.Tensor(0.8392764, shape=(), dtype=float32)
Epoch 8   tf.Tensor(0.9591643, shape=(), dtype=float32)
Loss: tf.Tensor(0.7605455, shape=(), dtype=float32)
Epoch 9   tf.Tensor(0.9714031, shape=(), dtype=float32)
Loss: tf.Tensor(0.69386816, shape=(), dtype=float32)
Accuracy for Training tf.Tensor(0.9714031, shape=(), dtype=float32)
Accuracy for testing characters:  tf.Tensor(0.8704, shape=(), dtype=float32)
```

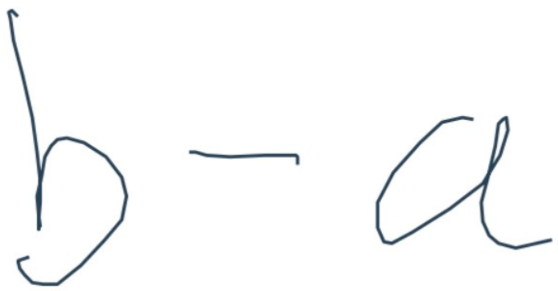
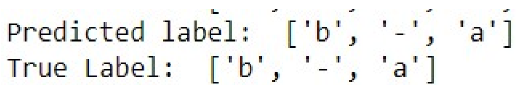
Let us look at some of the identification results where the model is predicting the LaTeX code for the handwritten equation.

- 
  

  ```
  Predicted label:  ['(', 'A', '2', 'B', '1', ')']
  True Label:  ['(', 'A', '2', 'B', '1', ')']
  ```

- 
  

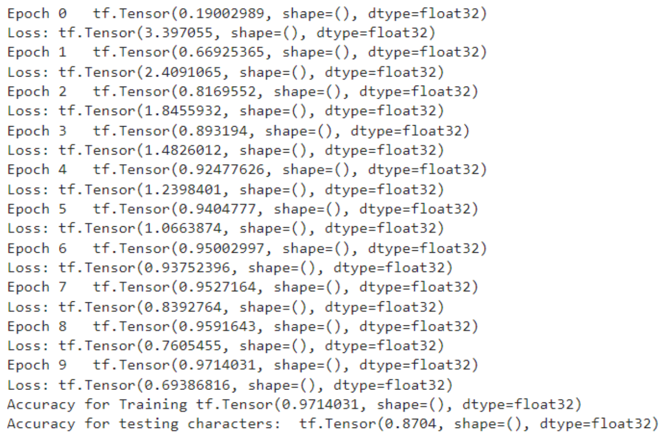
  ```
  Predicted label:  ['1', '-', 'e', '1']
  True Label:  ['1', '-', 'e', '1']
  ```

- 
  

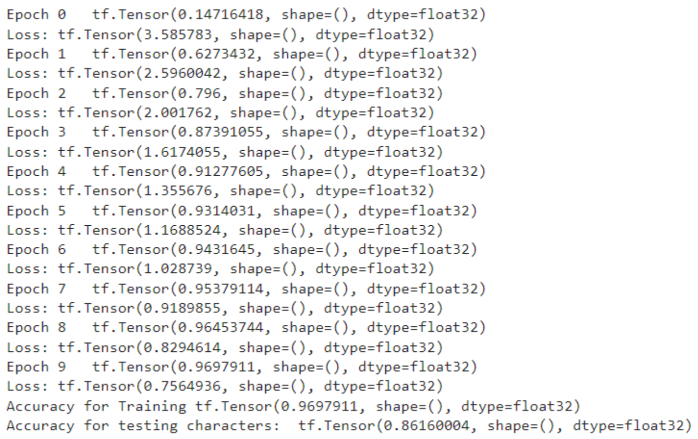
  ```
  Predicted label:  ['b', '-', 'a']
  True Label:  ['b', '-', 'a']
  ```

5 Adhoc Analysis:

In Deep Learning networks, adding more hidden layers does not neccesarily reduce the accuracy on the training set. Instead, what we observe is that the accuracy decreases as we go on adding more hidden layers. The reason behind that is the vanishing gradient problem which fails to update the initial layer weights for neural networks with many hidden layers. In this analysis, we have tried to demonstrate this phenomenon. We have trained the model over 10 epochs separately adding 1,2 hidden layers with RELU activation and 500 neurons respectively after flatenning the output of the CNN form the previously described architecture.



```
Epoch 0   tf.Tensor(0.19002989, shape=(), dtype=float32)
Loss: tf.Tensor(3.397055, shape=(), dtype=float32)
Epoch 1   tf.Tensor(0.66925365, shape=(), dtype=float32)
Loss: tf.Tensor(2.4091065, shape=(), dtype=float32)
Epoch 2   tf.Tensor(0.8169552, shape=(), dtype=float32)
Loss: tf.Tensor(1.8455932, shape=(), dtype=float32)
Epoch 3   tf.Tensor(0.893194, shape=(), dtype=float32)
Loss: tf.Tensor(1.4826012, shape=(), dtype=float32)
Epoch 4   tf.Tensor(0.92477626, shape=(), dtype=float32)
Loss: tf.Tensor(1.2398401, shape=(), dtype=float32)
Epoch 5   tf.Tensor(0.9404777, shape=(), dtype=float32)
Loss: tf.Tensor(1.0663874, shape=(), dtype=float32)
Epoch 6   tf.Tensor(0.95002997, shape=(), dtype=float32)
Loss: tf.Tensor(0.93752396, shape=(), dtype=float32)
Epoch 7   tf.Tensor(0.9527164, shape=(), dtype=float32)
Loss: tf.Tensor(0.8392764, shape=(), dtype=float32)
Epoch 8   tf.Tensor(0.9591643, shape=(), dtype=float32)
Loss: tf.Tensor(0.7605455, shape=(), dtype=float32)
Epoch 9   tf.Tensor(0.9714031, shape=(), dtype=float32)
Loss: tf.Tensor(0.69386816, shape=(), dtype=float32)
Accuracy for Training tf.Tensor(0.9714031, shape=(), dtype=float32)
Accuracy for testing characters:  tf.Tensor(0.8704, shape=(), dtype=float32)
```



```
Epoch 0   tf.Tensor(0.14716418, shape=(), dtype=float32)
Loss: tf.Tensor(3.585783, shape=(), dtype=float32)
Epoch 1   tf.Tensor(0.6273432, shape=(), dtype=float32)
Loss: tf.Tensor(2.5960042, shape=(), dtype=float32)
Epoch 2   tf.Tensor(0.796, shape=(), dtype=float32)
Loss: tf.Tensor(2.001762, shape=(), dtype=float32)
Epoch 3   tf.Tensor(0.87391055, shape=(), dtype=float32)
Loss: tf.Tensor(1.6174055, shape=(), dtype=float32)
Epoch 4   tf.Tensor(0.91277605, shape=(), dtype=float32)
Loss: tf.Tensor(1.355676, shape=(), dtype=float32)
Epoch 5   tf.Tensor(0.9314031, shape=(), dtype=float32)
Loss: tf.Tensor(1.1688524, shape=(), dtype=float32)
Epoch 6   tf.Tensor(0.9431645, shape=(), dtype=float32)
Loss: tf.Tensor(1.028739, shape=(), dtype=float32)
Epoch 7   tf.Tensor(0.95379114, shape=(), dtype=float32)
Loss: tf.Tensor(0.9189855, shape=(), dtype=float32)
Epoch 8   tf.Tensor(0.96453744, shape=(), dtype=float32)
Loss: tf.Tensor(0.8294614, shape=(), dtype=float32)
Epoch 9   tf.Tensor(0.9697911, shape=(), dtype=float32)
Loss: tf.Tensor(0.7564936, shape=(), dtype=float32)
Accuracy for Training tf.Tensor(0.9697911, shape=(), dtype=float32)
Accuracy for testing characters:  tf.Tensor(0.86160004, shape=(), dtype=float32)
```

The following output has been summarized in the following table below:

Epochs	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Hidden Layer 1	19%	66.92%	81.69%	89.31%	92.47%	94.04%	95%	95.27%	95.91%	97.14%
Hidden Layer 2	14.71%	62.73%	79.6%	87.39%	91.27%	93.14%	94.31%	95.37%	96.45%	96.97%

We compare the accuracies for the 2 cases for each epoch and observe that the model with 1 hidden layer has highest accuracy for each epoch as compared to the other case with 2 hidden layers.

6 Conclusion:

One of the challenges that we have encountered is with the preprocessing of the INKML file. We used a library that would help us convert the INKML files to images and encode the necessary location information for us. We used this library because we have not worked with INKML files before and the process to convert the file to an image was a very complicated process that seemed out of the scope of our current capability. We have reached a sufficiently high accuracy in this deep learning exercise and we are satisfied with the results. But, the only limiation of this project is that we can only work with INKML files and does not allow us to test it on jpeg or png file. Some of the biggest takeaways from this project are learning how to work with INKML Files and learning how to tweak models in order to try and raise accuracy.

7 References:

1. **Code:** Github Codes
2. **Data Source:** CROHME Dataset