Handwritten Equation Identification

Introduction:

The problem that is trying to be solved is classifying and identifying mathematical symbols from equations. The paper being implemented is *Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Networks* by Catherine Lu and Karanveer Mohan. The objective of this paper is to provide a faster alternative for recognizing math symbols and equations instead of using typesetting systems. This paper was chosen because of the applicable purpose of the results. Having experienced the long hours necessary using typesetting systems and editors, developing a way that would speed up the process seemed like a worthwhile endeavor. This problem is a classification problem because the model is trying to identify the different math symbols in an handwritten equation and present

Related Work:

This research paper could help us achieve our extended/reach goal of solving complex mathematical equations. This research paper not only addresses the identifying mathematical symbols, but it also implements a method to solve these equations, specifically quadratic equations.

List of Useful Links

Uses an SVM: https://github.com/anushreedas/Handwritten_Math_Expression_Recognition

Only Data Extraction: https://github.com/ThomasLech/CROHME_extractor

Data Extraction:
https://github.com/karan1149/crohme-data-extractor/blob/master/extract.py

Data:

Link to dataset :
http://www.iapr-tc11.org/mediawiki/index.php/CROHME:_Competition_on_Recognition_of_Online_Handwritten_Mathematical_Expressions

The CROHME dataset consists of inkml files representing different mathematical equations consisting of over 100 different mathematical symbols.
The size of the whole CROHME file is 55mb consisting of 3 different datasets from 3 different years. This data will require some preprocessing as all the images are .inkml files so they need to be converted to a form that can be used for testing/training.

For preprocessing we used the library INKML extractor to help us extract the INKML images because it was a very complicated process. 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.

We had to change the Extractor class of the library linked above to adapt to our preprocessing. We changed the get_data and parse_inkml functions so that we can get 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.

For testing expressions, we have an array of arrays, where each array has a different size consisting of the symbols of each equation. We also get an array of all the symbols to test out symbol identification first. We call these functions in preprocess.py, where we divide the list of symbols into inputs and labels. We also convert them to numpy arrays of the necessary size to pass it into the convolutional layers. Another step added in preprocess.py was to one-hot encode the labels, so that each class of symbols can be classified during training and testing.

We use 32x32 pixel arrays to represent the symbols of the equations, these can be changed to any suitable matrix. For example, 24x24, 48x48, etc.

Methodology:

We will be training our model using CNNs. Taking the paper linked above as reference, we will be tuning a 2-layer fully connected neural network. The architecture for the CNNs suggested in the paper is of the form [Conv-Relu-Pool] x N - [FC - Relu] x M - [FC] - [Softmax], where the parameters for our values of N and M are filter size (F) and number of hidden neurons per layer (H). The number of filters used is suggested to be 32 with a 2x2 max pool with stride of 2 for the pooling step.

Model architecture:
[Conv2D] with 32 filters, 5 kernel size, 1 stride, SAME padding and RELU activation
[Conv2D] with 32 filters, 5 kernel size, 1 stride, SAME padding and RELU activation
[MaxPool2D] with 2 pool size
[Conv2D] with 128 filters, 5 kernel size, 1 stride, SAME padding and RELU activation
[Conv2D] with 128 filters, 5 kernel size, 1 stride, SAME padding and RELU activation
[MaxPool2D] with 2 pool size
[Flatten]
[Dense] with RELU activation acting as a hidden layer of size 500
[Dropout] with a 0.3 rate
[Dense] with SOFTMAX activation and output size equal to the number of classes(101)

Training and Testing:

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.

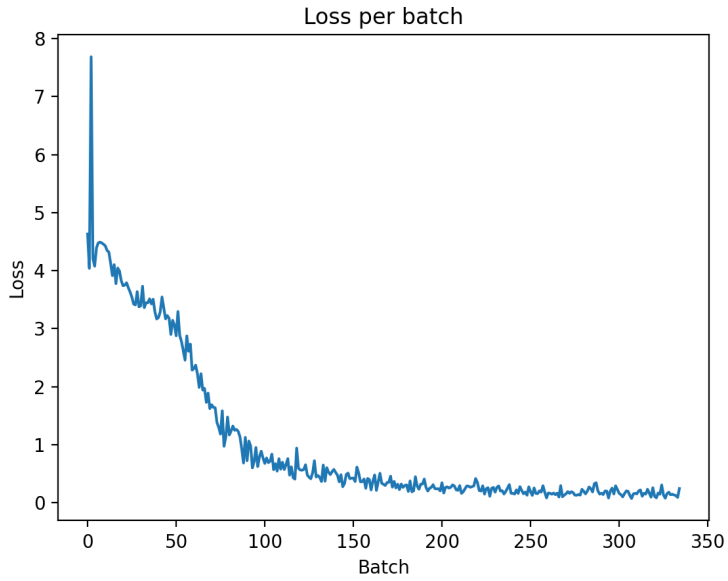
Results:

Before our model was only giving us an accuracy of around 8% collectively and then after we found the issue and ran it in a working environment we got a much better accuracy.

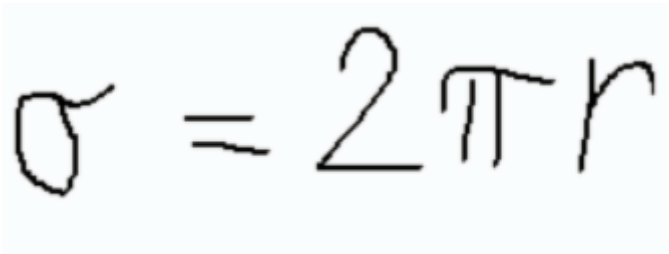
- Training Accuracy: 95.12%
- Individual Symbols Accuracy: 87.24%
- Expression Accuracy: 85.01%

```
 activation and thus does not represent logits. Was this intended?"
  warnings.warn(
Epoch 0   tf.Tensor(0.17062685, shape=(), dtype=float32)
Loss: tf.Tensor(3.5193357, shape=(), dtype=float32)
Epoch 1   tf.Tensor(0.7592836, shape=(), dtype=float32)
Loss: tf.Tensor(2.226553, shape=(), dtype=float32)
Epoch 2   tf.Tensor(0.9021492, shape=(), dtype=float32)
Loss: tf.Tensor(1.6000229, shape=(), dtype=float32)
Epoch 3   tf.Tensor(0.9361792, shape=(), dtype=float32)
Loss: tf.Tensor(1.253871, shape=(), dtype=float32)
Epoch 4   tf.Tensor(0.9512835, shape=(), dtype=float32)
Loss: tf.Tensor(1.0345805, shape=(), dtype=float32)
Accuracy for Training tf.Tensor(0.9512835, shape=(), dtype=float32)
Accuracy for testing characters:  tf.Tensor(0.8724801, shape=(), dtype=float32)
Accuracy for testing expression tf.Tensor(0.85016775, shape=(), dtype=float32)
(environment) (base) devansh@devanshs-mbp CSCI1470_Final_Project %
```

- Loss graph:



- Identification Results:
  - Predicted Labels: ['\\sigma', '=', '2', '\\pi', 'r']



Challenges:

- One of the challenges that we have encountered is with the preprocessing of the INKML file. We originally planned on using a PNG version of our dataset instead of the INKML version, but ran into trouble trying to encode the location of the equations from the images and splitting up the ground-truth labels cleanly. We reverted to using the INKML files and 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. This is the library we used https://github.com/karan1149/crohme-data-extractor/blob/master/extract.py
- Another challenge that we faced was that our model was not training very well, thus yielding a very low accuracy. The loss did not decrease as we trained more. At first we thought that it was our accuracy function that was not calculating correctly however, we changed the accuracy function and it did not produce better results. We also changed the loss function from Softmax Cross Entropy with Logits to Categorical Cross Entropy loss and the loss still did not decrease, it still hovers around 4.45 throughout the duration of training. We also tried changing the architecture of the model, however it did not seem to provide a big improvement to the loss or the accuracy. We also played with the learning rate and the size of the hidden dimensions in hopes of trying to increase the accuracy and get our model to train. It took us a long time to figure out the issue and it turns out that some of us were running it in an environment and some of us thought that we were running it in an environment but the environment was broken. So, the code worked when it was run in the CSCI1470 environment given to us in class.

Ethics:

Why is Deep Learning a good approach to this problem?

Deep Learning is a good approach to this problem because it involves recognition and classification. CNNs are the optimal method when it comes to training a model to recognize and learn the data that is being inputted. CNNs are also more efficient and accurate in producing results when the problem involves classifications.

How are you planning to quantify or measure error or success? What implications does your quantification have?

The method for quantifying success is to identify the number of correct symbols classified and divide that over the number of total symbols. A success would be if the accuracy of the model was over a certain threshold. There should be no major implications with this method of quantification because it simply takes into account the number of correct symbols the model identified over the total numbers of symbols in the dataset.

Reflection:

- How do you feel your project ultimately turned out? How did you do relative to your base/target/stretch goals?
    - The project ultimately turned out to be okay since we reached a sufficiently high accuracy and the code worked and produced a result. We did not implement our stretch goal of solving the equations after they had been identified. But we did complete our base and target goals of completing a model that identifies handwritten equations.
- Did your model work out the way you expected it to?
    - We followed the architecture given to us in the research paper we took inspiration from; however, initially, the model did not work out the way we expected to. By fine tuning the parameters like learning rate, the different layers in the model, the number of filters, and running the proper environment etc., we were able to slowly build up a better accuracy. Concluding the project, we are satisfied by the final losses and accuracies as they are consistent with the accuracies provided in the research paper.
- How did your approach change over time? What kind of pivots did you make, if any? Would you have done differently if you could do your project over again?

    - In the beginning, our approach was to use a PNG form of the CHROME dataset so we did not have to deal with the INKML file. However, we soon realized that we did not have a way to encode the location of each of the math symbols so we had to work with the INKML version. This changed our approach as we had to change the way we preprocessed and the layout our data was in. Something that we would have done differently if we could redo our entire project was to change the dataset. Since we used a dataset that was in a form that we had never used before, we had to spend a lot of time figuring out how to use it and ended up using a library online to help us preprocess our code.
- What do you think you can further improve on if you had more time?
    - Something that we could further improve upon if we had more time would be the accuracy of the model. We could change the model to give a greater testing accuracy. We could also improve upon the extensibility of this code. Right now the code only works with INKML files which is not commonly used by most people so we could make this code more accessible if it were able to be used on other image types.
- What are your biggest takeaways from this project/what did you learn?
    - Some of the biggest takeaways from this project are
        1. Learning how to work with INKML Files
        2. Learning how to tweak models in order to try and raise accuracy
        3. Time management - planning when to finish certain components and to account for extra time in case we run into problems was really important.