An ensemble of CNN

Reading handwritten mathematical expressions

Joakim Sörensen
Kyung Hee University dept. of *Computer Science*
Suweon, South-Korea
joaso184@student.liu.se

***Abstract*—In order to read handwritten mathematical symbols an ensemble of Convolutional Neural Networks was implemented with the use of Google's TensorFlow library, which provides a machine learning API. The model built consists of three convolution layers, three pooling layers and two dense layers. It uses a ReLU activation function and a softmax classifier to predict one of 82 possible labels corresponding to the expression. Training data and Testing data was divided so that there were 74.7 % and 25.3 % respectively and then processed accordingly resulting in an accuracy of around 84 % with 100 epochs and a batch size of 100 images. Showing that machine learning algorithms is becoming accessible to the more general public.**

Keywords—cnn; convolutional neural network; ensemble; handwritten mathematical expression; python;tensorflow;jpeg

I. INTRODUCTION

In recent years, it has been hard not to hear the words machine learning or neural networks regardless of whether one work or study something related to computer science or not. As these concepts become known to the general public, the accessibility increases as well. Machine learning APIs have been and still are being developed by giants such as Google. In addition, the huge amount of data in circulation in our day to day life contributes to a particularly beneficial environment to apply the various models currently available. The intent of the project was to use such models in order to make an algorithm that can read and process handwritten symbols from a large variety of classes.

This paper shows how an ensemble of CNN (Convolutional Neural Network) models, built with machine learning API tensorflow, can be applied to read handwritten mathematical expressions. The dataset chosen to be processed was a set of jpeg images, consisting of 82 classes, derived from the CROHME (Competition on Recognition of Online Handwritten Mathematical Expression) dataset. Due to the significant impact an ensemble model has on performance, a final result is yet to be obtained. Therefore, the option to run a single model was also implemented. This single model delivers a result in a more reasonable fashion even if the number of epochs is high. In addition, because of the massive amount of available data, the files could be split up in to a training set and a testing set, each of considerable size. When the model is trained using the train dataset and tested using the test dataset, an accuracy of around 84% is to be expected, assuming 40 epochs and a batch size of 100. The model was designed using material dealt with in the Machine Learning class autumn 2017 at Kyung Hee University with the addition of Google's tensorflow documentation.

II. METHODOLOGY

A. Limitations

Most of the limitations of the project were the result from limited hardware. All the testing was performed on a MacBook Pro (Retina) from 2012 with the following specifications:

- Processor: 2.5 GHz Intel Core i5
- Memory: 8G 1600 MHz DDR3
- Disk: Macintosh HD
- Graphic: Intel HD Graphics 4000 1536 MB

The testing was hence adapted to these specification, and settings were set so that the computations would finish within reasonable time.

B. Choosing the dataset

The dataset chosen was a set of jpeg images derived from the CROHME (Competition on Recognition of Online Handwritten Mathematical Expression) dataset [1][2]. The reason for this choice was that, compared to other available datasets, there seemed to be a comparable abundance of sample data. Also, as building an image processing neural network was the intent of the project this dataset was deemed to be of sufficient difficulty. Sufficient meaning not too hard to obtain a fair result, but also not too easy either. This early on estimation was later shown to be surprisingly accurate. The dataset consists of 1.54GB of 45x45 jpeg images, meaning over 100 000 image samples [3]. It contains a total of 82 classes. The samples of each class, however, can differ quite a lot. The abundance of image samples in combination with limited hardware made it impossible to process all data. Figure 1 shows an example of how the expressions might look.

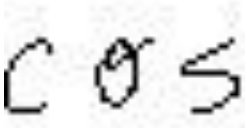

Fig. 1. Example of an image sample

C. Preprocessing

As stated in the previous paragraph, the sheer amount of data made preprocessing inevitable. As the image samples were already converted to equally sized jpeg images, all that was going to be needed was dividing the samples and converting them in to matrices readable for tensorflow. The image samples of the mathematical expressions were separated in different folders named after the expression which they contain. The ordering and selection of files is done by the script called create_train_test_data.py. This script chooses, in accordance with entered values, a number of random files from each of the 82 classes and puts them in a train and test directory respectively. An explanation of this processed will be provided in a later paragraph.

For the second part of the preprocessing step, i.e. making the data matrices, a module was created. The module is called load_data.py. This module keeps track on the amount of train and test files, provides functionality to read and convert these files to numpy arrays directly and save them in advance to npy files using the numpy library [4]. Each jpeg image is decoded and the data is added as an array to a numpy array. Then the image data are mapped to corresponding label using a one hot[1] encoded matrix. The matrix containing the image data is reshaped to 2D and then the two data matrices are returned and written to npy files. The reason for the last functionality being that reading files as the need occurs will progressively slow down the algorithm. It was noted that reading all necessary files in advance was more efficient than reading the files batch by batch. Thanks to this notion it was possible to increase the size of both the training dataset and the testing dataset, hence also increasing the credibility of the model.

D. Building the network

Since the project was started before necessary image processing theory and practice had been properly studied the first approach to the problem was to build a deep and wide neural network and not a more suitable, currently used, CNN. This would most certainly had led to a less accurate prediction. The deep and wide

[1] **One hot encoding** is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

model, however, never got to be properly tested because of a bug in the load_data.py module that resiliently remained until the deep and wide model was deprecated and substituted for a CNN model.

The CNN model consists of three convolution and pooling layers respectively, with two dense layers at the end. The first convolution layer uses a kernel size 5x5, with a step size of five; 81 of these are applied. Each convolution layer uses a ReLU activation function. The first pooling layer uses a pool size of 3x3 and three strides, reducing the output of the next layer to a vector of shape [batch_size, 15, 15, 81] with the max pooling function. The same values were used for the second and third convolution layer as well as the second pooling layer, with the number of filters accumulating for each layer. For the last pooling layer, a pooling size of 5x5 is used and the output of this layer is then reshaped and so 'flattened' as to fit the dense, fully connected, layer at the end. The first dense layer consists of 1024 units with a drop rate of 0.4, the output of which is then used for the logits parameters in the softmax classification function provided by tensorflow. The logits parameter is another dense layer itself with 82 units, which is the number of classes. This is in order to make it have a compatible shape with the one hot encoded output matrix. The loss is calculated with tensorflow's softmax_cross_entropy function and the optimizer used is the AdamOptimizer. The learning_rate is set to 0.001. Figure 1 shows an example of a typical CNN like the one used in the project.

The CNN model was implemented as a class in order to provide the option of making an ensemble of said model. The option to use the ensemble is then provided through a Boolean in the cnn_math_main.py file.

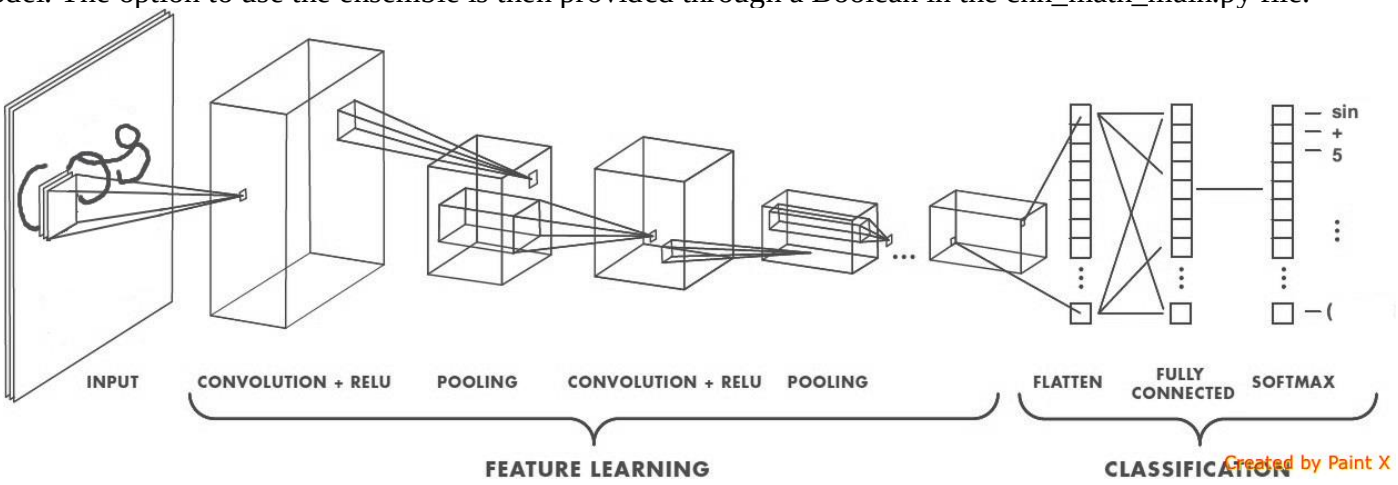

Fig. 2. Example of a Convolutional Neural Network

E. Training and testing

To ensure that the model built is sturdy and reliable, having two considerably large training and testing datasets was of high priority from the very start of the project. Picking a dataset with a vast amount of data was therefore a given. As aforementioned, the dataset chosen contains over 100 000 image samples of handwritten mathematical expressions. Due to lack of computational power, and time, all of these image samples could not be processed. A random selection was hence performed in order to create a training and testing dataset of a more preferable size. By using a python script 60, or all if less than 60 were available, randomly selected image samples of each of the 82 mathematical expressions[2] were copied to a train data folder of the same structure as the original. The same was done for the test data, but with 20 or less image samples being randomly selected. The model was then trained with different values for batch size and epochs. When training is finished the model is right away tested with the test dataset.

The tensorflow's AdamOptimizer is used along with a learning rate of 0.001. GradientDescentOptimizer with a higher learning rate was initially used but generated a less desirable output and was thus discarded. The learning rate along with batch size and epochs were altered as the model was being tuned.

[2] 4853 (74.7 %) train samples, 1640 (25.3 %) test samples

III. Results

The following section will present the results attained under different hyper parameters. The result being presented is accuracy and precision of the trained model when tested with the test data. The training and testing use different datasets to promote a more reliable prediction.

A. *Accuracy*

The model generates the accuracy shown in table I on the current data. Following figures shows the graph for the accuracy for another set of test, i.e. not necessarily the same as table I. See figure 3 and 4.

TABLE I. ACCURACY

Epochs	Batch size	Accuracy
40	20	0.832317
40	100	0.82374
40	200	0.82561
100	20	0.830488
200	100	0.84072

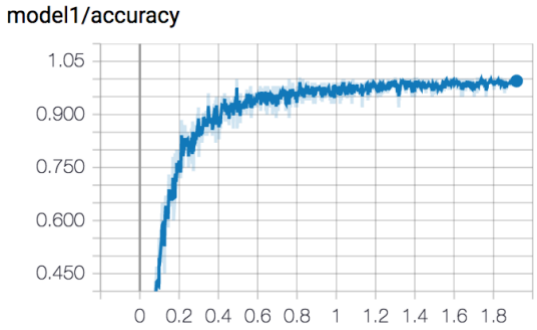

Fig. 3. The accuracy acquired with epoch= 50, batch_size = 100

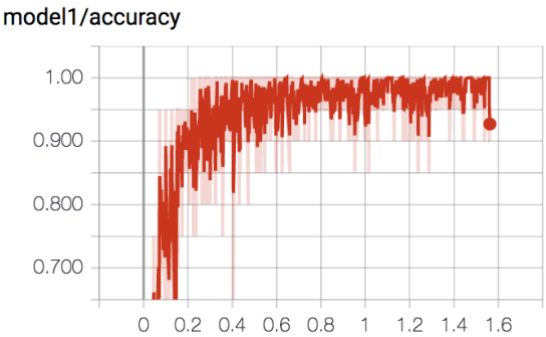

Fig. 4. The accuracy acquired with epoch= 40, batch_size = 20

B. *Precision*

Precision always comes out at around 50 %, having a graph looking like figure 5 and 6.

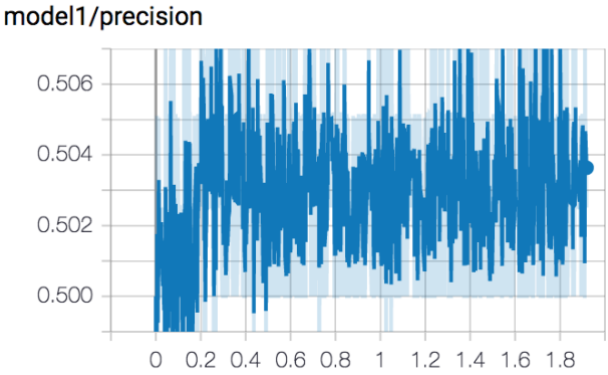

Fig. 5. The precision acquired with epoch= 50, batch_size = 100

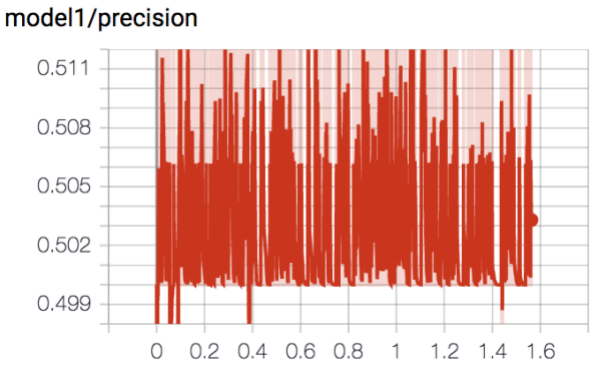

Fig. 6. The precision acquired with epoch= 40, batch_size = 20

C. *Loss*

The following pictures show the development of the loss function over time for some of the runs, see figure 7 and 8. Given enough Epochs the loss for the training dataset converges towards zero for the first and second decimals.

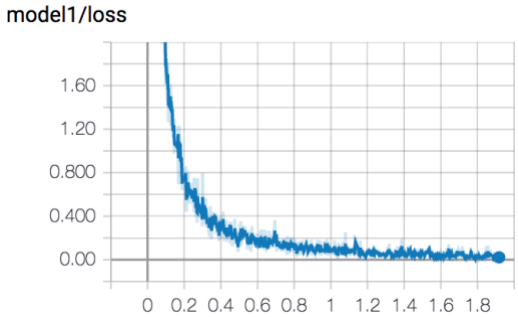

Fig. 7. The loss acquired with epoch= 50, batch_size = 100

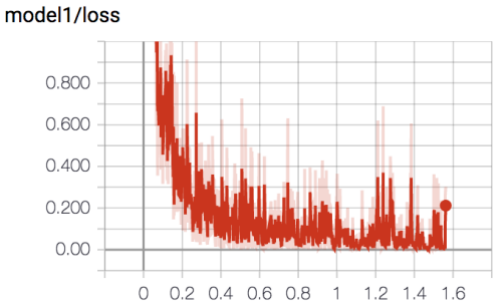

Fig. 8. The loss acquired with epoch= 40, batch_size = 20

IV. Discussion

The following sections will provide discussion about the obtained results and how to improve them.

A. *Limitations*

As stated in previous paragraph, most of the limitations were caused by insufficient hardware. The amount of data needed to make a reliable model was there, but could simply not be processed. Even with the reduced amount of data the training and testing take several hours, which is why the presented results are scarce. This problem could have been avoided by using the GPU version of tensorflow on the computers in the Kyung Hee University computer lab. Unfortunately, the majority of the testing had to be performed in Seoul, and not on campus. This resulted in the tests having to be performed on the accessible hardware, being the very limited MacBook Pro from late 2012. As future project, another round of testing would preferably be performed on the hardware provided by Kyung Hee University in order to find the models true capacity.

B. *Precision*

As can be seen from figure 5 and 6, the precision does not act as expected. There might be two reasons for this. The first being that the calculation of the precision is done is such a way that an accurate value is impossible to obtain, and the second being that there is something wrong with the model. As all other values, as well as the predictions, work as intended the first scenario is the most likely one. The precision is calculated like follows:

$$TP/(TP + FP)$$

Were TP being true positive and FP false positive. The code for this calculation can be seen in figure 9. Probabilities is a tensor that contains the result of the softmax classification performed in the last dense layer. Precision is run with the test dataset and the result is always 50 % which is odd. The possible error due to integer division was considered, hence the tf.float32 specification and use of the tensorflow divide function. This alteration did not change the outcome for the better alas, and the source of the oddity is still unknown.

```
predicted = tf.argmax(self.probabilities, 1)
true = tf.argmax(self.Y, 1)
tp = tf.count_nonzero(predicted * true, dtype=tf.float32)
fp = tf.count_nonzero(predicted * (true - 1), dtype=tf.float32)
self.precision = tf.divide(tp, tp + fp)
```

Fig. 8. The calculation of precision using tensorflow

C. Ensemble

All of the test that generated the presented results was done with the single model and not the ensemble. The reason for this being that the ensemble mode simply took too much time to train. It was far too performance heavy for the available hardware. In addition, in recent revisions the ensemble mode generates a shape error, which has not been resolved. Since the single model and the ensemble uses the same data, with same model, and the same sized placeholders it seems to be near impossible to find the potential deviation causing the error. Thus, it is highly recommended not to use the ensemble mode in the latest revisions. If the urge to try the ensemble mode gets too strong, however, it might be possible to run it in one of the early versions in the GitLab repository.

D. Improvements

Possible ways to improve the obtained results would be to use all of the available test data. As aforementioned, the hardware limitation made such a feat impossible at the time being, but it is not excluded as a possibility for future experiments. Running the tests on a GPU version of tensorflow with a high end graphic card would speed up the computations significantly. The result being, a model trained with a greater amount of data, but still with a rather high number of epochs. In addition, assuming the current bug is able to be fixed, the ensemble mode could be run as to provide an even greater improvement of accuracy and precision.

Another way to possibly improve the results would be to change the layers of the model. The reason for the current model having three layers is that, the original image being 45x45 pixels, resulted in very satisfactory reduction values during the pooling stages. This might not be optimal, however, and provided with better hardware more layers processing bigger image might result in a higher accuracy.

V. Conlusion

The intention of the project was to build a Convolutional Neural Network in order to process handwritten mathematical expressions. The CNN built provided a resulting accuracy around 84 %, and as such deemed as a relative success. Considering available hardware resources, an accuracy of around 84 % is, although not optimal, still a good result. In the future, to improve the result, the model could be trained on high end hardware, using a larger amount of the sample data. The model itself could also be the target of some improvement regarding how the layers are structured. In order to keep down the processing time the layers were kept few and the image was reduced significantly between the layers. Although an ensemble was implemented, it is unstable in recent revisions and could not be used.

Thanks to big companies like Google, pushing the development and providing resources, machine learning is becoming available to the more general public (with an interest in programming). With the use of machine learning APIs such as tensorflow, building a convolutional neural network is not as complicated as has been.

VI. Related work

In a report form 2016 the results from the Competition on Recognition of Online Handwritten Mathematical Expressions is presented. There showing that, albeit consisting of a different file format, the accurate processing of such a dataset is still hard and presented accuracy is thus quite low [5].

REFERENCES

Link to GitLab repository for project: https://gitlab.ida.liu.se/joaso184/MachineLearningProject2017

[1] http://www.isical.ac.in/~crohme/CROHME_data.html, accessed 2017-10-31

[2] https://www.kaggle.com/xainano/handwrittenmathsymbols, accessed 2017-10-31

[3] https://www.kaggle.com/xainano/handwrittenmathsymbols, accessed 2017-10-31

[4] http://www.numpy.org/, accessed 2017-11-27

[5] H. Mouchere, C. Viard-Gaudin, R. Zanibbi, U. Garain "ICFHR 2016 CROHME: Competition on recognition of handwritten mathematical expressions, IEEE, Shenzen, China, October 2016.