Recognition of Mathematical Symbols using a Combination of Convolutional Neural Networks and Restricted Boltzmann machines

Dhanush Dharmaretnam
Department of Computer Science, University of Victoria, Canada.
Email: dhanushd@uvic.ca

Abstract—Recognition of mathematical expression is a widely discussed area of research. The ability to recognize and evaluate mathematical expressions in both offline (Hand Written) and online (Digital media) will be an extremely useful tool in the field of science. For example, software suits like 'maple' relies on accurate identification of mathematical symbols to perform various computations online.

In this paper, we experiment with various Machine learning algorithms and neural networks to recognize and classify these handwritten mathematical symbols. We have utilized a wide variety of algorithms such as Logistic Regression, Support Vector Machines (SVM), Simple Neural Networks and Convolutional Neural networks(CNN) for this classification task. We have also attempted to use an energy based model – Restricted Boltzmann machine (RBM) to help us improve the classification accuracy.

I. INTRODUCTION

The task of classifying mathematical symbols has been an area of immense interest over the last decade. In fact, some of the works in this area were published in the international conference on frontiers in handwriting recognition(ICFHR). The results of identifying and evaluating mathematical expressions at the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) were generally poor with the top team achieving 60% Accuracy. Such low accuracies can be attributes to the fact that there are so many mathematical handwritten symbols and it's a daunting task to classify and label them all correctly. Significantly less research has been undertaken in this area and therefore we strongly believe that improving the accuracies of classifying mathematics symbols will strongly help us in creating a drastic improvement in identifying and evaluation of mathematical expressions.

The task of classifying mathematic symbol is mainly complicated due to the fact that there are so many different symbols ranging from operators, trigonometric symbols, roman numerals, ASCII characters, MNIST numbers. Furthermore, these symbols come in different fonts, shapes (Italic etc.) and sizes. The most complicated aspect comes from the fact that different people tend to write these symbols differently and it becomes really difficult for machine learning algorithms to correctly classify and label them.

Mostly the paper focus on using neural networks and deep learning techniques to achieve its goal. We also make an attempt to use RBM as a classifier. Recent researches on combinational neural nets has prompted us to try a CNN-RBM approach towards classifying mathematical symbols. It's been shown that a simple CNN-RBM could improve accuracies on MNIST. We also implemented various other machine learning algorithms on our dataset such as Support Vector Machines and Logistic Regression. The underlying reason for implementation of different algorithms is because of the fact that this dataset is unique and not much experiments have been done using this dataset in previous papers. This approach helps us to relatively compare the effectiveness of using deep learning techniques as compared to traditional machine learning techniques like SVM and logistic regression. We also visualized activations of different convolutional layer and the patterns learned by our CNN filters.

II. Related Works

There were many papers and research done over the last four years in the area of recognition of mathematical symbols and expression especially after CHROME. Xinyan et and al [1] used a combination of Genetic algorithm and neural network to classify the mathematical symbols and got accuracies close to 90.6%. Another work in this area was done by Lu et and al [2] who used a convolutional Neural network to classify the symbols. They got accuracies close to 83% and used HMM models to evaluate the classified expression. Sometimes the symbols need to be preprocessed and features needs to selected before a classifier can be trained. Nicholas et and al [3] showed that PHOG (Pyramid of Oriented Gradients) features used in conjunction with a linear SVM classifier could achieve higher accuracies. They were able to attain accuracies close to 96% for online content while the accuracies for online handwritten symbols dropped to 92%.

There are many other methods suggested for extraction of features for this task and these are discussed in detail in the ICFHR papers. Notable features include loops, intersection, point density, Height-Width ration etc. as discussed by watt and Xie[4] where they used a preprocessing step with elastic matching technique to extract the features. The CHROME competition results over the last few years have shown significant improvement in the classification accuracies of both expressions and individual symbols. The CHROME dataset which is used for these competitions has changed drastically every year and therefore it becomes really difficult to compare the performance of different algorithms. The CHROME 2016 competition reported a maximum accuracy of 92.81% at symbol recognition and expression recognition at 67.95% [5]. Even though this paper achieves 99.1% accuracy, the experiments were performed on a much larger dataset which was uploaded to Kaggle – A google machine learning community. In the next section we will describe our dataset in detail.

III. Dataset Description

The CHROME dataset [6] is the standard dataset for the mathematical symbol recognition. This dataset which contains 82 symbols were parsed and added to kaggle Recently (This dataset was also added with more data from other sources). This dataset which is a modified version of original CHROME which is in Ink Markup Language (InkML) format was selected as our dataset for all our experiments.

The dataset consists of 375000 images each of 45 * 45 pixels. There are 82 different symbol classes such as MNIST digits, Math operators such as add, Subtract, Integral, log, some trigonometric operators such as Sin, Cos etc., Greek symbols such as alpha, beta, gamma etc. Another important aspect of this dataset is the presence of extremely skewed classes. Some of the classes had about 20,000 symbols while some classes had less than 200 symbols. We plotted a histogram to represent the skewness of our dataset under fig 1. This dataset cannot be considered as an ideal one due to its extreme skewness, however, it's really difficult and rare to find an ideal dataset for mathematical symbol classification.

The dataset images were converted into 2d NumPy arrays using Image module of the PIL library. The dataset was further divided into train, cross validation and test using Scikit-Learn train_test_split module. The dataset was normalized by dividing by 255 to improve the computational speed and to make the network converge faster. Based on our experiments, it was also discovered that absence of mean normalization causes a large gradient to flow in a neural network causing noisy updates over successive batches and causes considerable increase in the training time.

The resulting split dataset was saved as a pickle object to enable faster loading during training time. However, the pickled object was of size 9.3Gb causing further bottle necks in performance since this dataset couldn't be fit into memory(8Gb) resulting in larger loading times during training.

IV. Research Methodologies

The basic research methodology that we used was to try different machine learning algorithms to classify the mathematical symbols. We used wide variety of algorithms such as Logistic Regression, Support Vector Machines, fully connected Neural Networks (NN), Convolutional Neural Networks (CNN) and Restricted Boltzmann Machines (RBM). We will briefly describe each of these experiments below.

a. Simple Fully Connected Neural Network

We used a fully connected Neural Network with no convolutional layers to classify the symbols. A well-known framework called Keras [7] was used to train our model. Keras acts like a wrapper around popular deep learning libraries such as Theano and Tensor-Flow which enables us to create and train deep models with very few lines of code. Keras also supports loss functions such as categorical cross entropy and Binarized cross entropy which are popular for deep learning. It also supports practices like learning rate annealing, Drop Out [8], L1, L2 regularization and even adaptive learning algorithms such as ADAM [9], Adaprop, NADAM [10] etc.

The network basically consisted of two fully connected layers with a drop out layer in between. The input was flatted into a 1d array of size 2025 and fed into the input layer with 512 neurons. The output of input layer was connected to a drop out layer which randomly drops output from 20% of neurons in layer one and is finally connected to a second layer with 82 neurons. The output layer (3rd layer) was a softmax layer which calculated the probability of each classes and the class with highest probability was selected as the predicted class. The number of layers and neurons in each layer, along with drop out hyper parameter was selected using random search. The model was trained for 50 epoch using ADAM as optimizer and learning rate annealing of 20% every 5 epochs. ReLu was used as the activation function.

The model achieved a maximum accuracy of 76% which was below expected accuracy. We plotted statistics such as F1 score (A common practice used to study skewed classes) and also confusion matrix to analyze our results. We found that the network does a very bad job at classifying most of the classes. The detailed results of this experiment can be found under fig 2. It may be worth noting from the confusion matrix that most of the classes seemed to be confused with each other and F1scores are low for almost all classes.

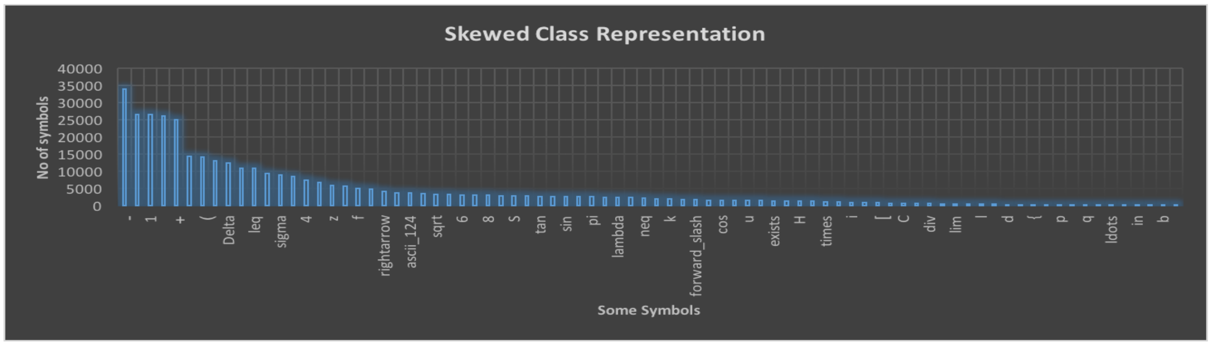


Fig 1: Representation of classes vs number of symbols in that class helps us understand the degree of the skewness in our dataset. The math operators seem to have most number of samples whereas functions such as limit, log, exists etc. tends to be skewed.

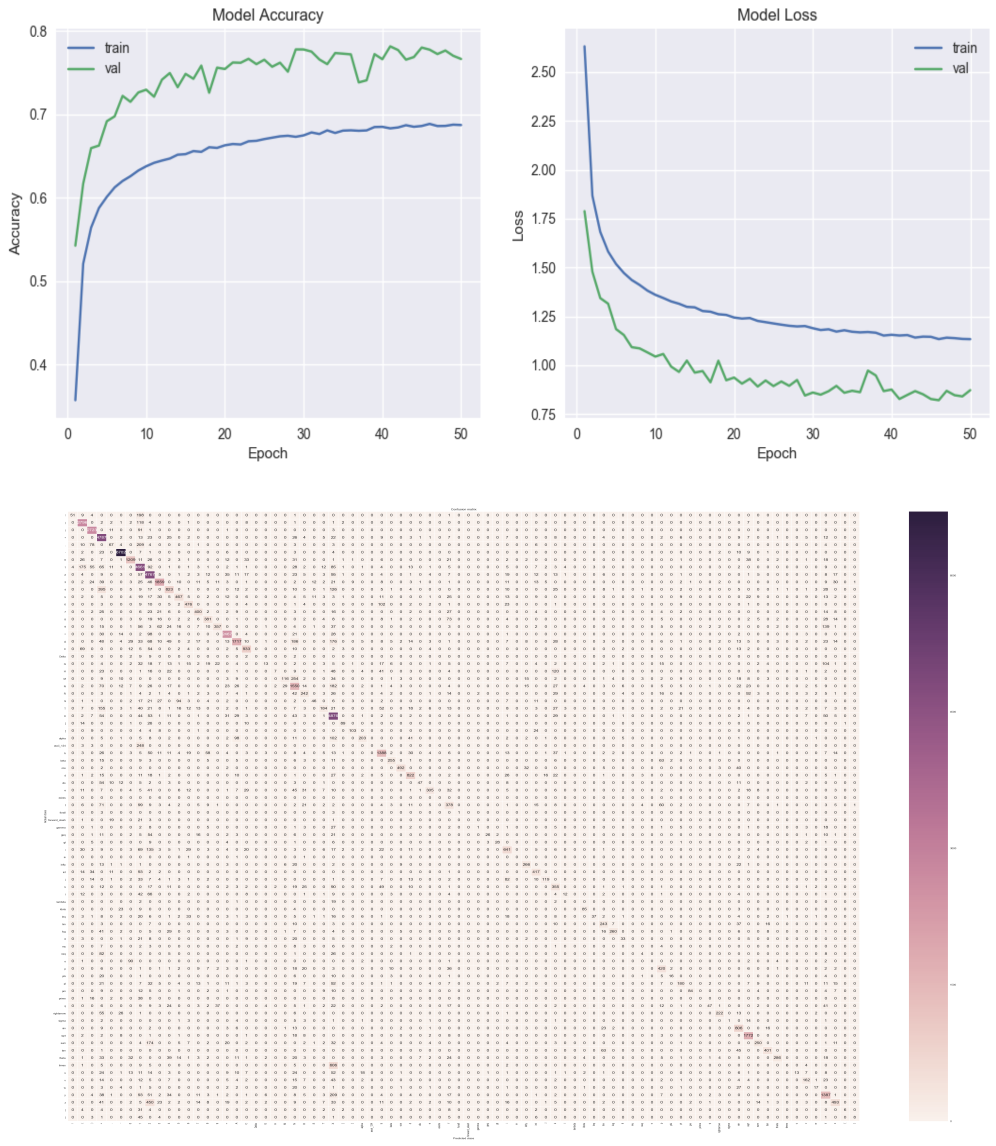


fig 2: The upper portion shows the learning curve for the simple neural net which was trained for 50 epochs and using adaptive learning rate using Adam as optimizer. The accuracy of the network peaked at 76% on cross validation even after hyper tuning various parameters such as learning rate, number of neurons and drop out. The confusion matrix doesn't have a prominent diagonal and values seems scattered all across the matrix plot. This shows a large degree of confusion (poor Precision and Recall) and the case was worst for skewed classes in the dataset.

b. Convolutional Neural Networks

Convolutional neural networks are quite popular in the field of computer vison for object recognition. There have been many experiments done on the MNIST digit dataset using CNN and the results were considerably better as compared to simple neural networks. CNN uses convolutional layers which contains filters which in turn acts as feature detectors. These filters convert the image into feature maps. These high dimensional feature maps are then max pooled to reduce the dimensionality of the feature space. Max pooling helps to improve the computational cost of the network, makes the network more tolerant to distortion and noise in the images and dataset respectively. The max pooled layers are either connected to other convolutional layer or fully connected neural layers.

We tried many different architectures for our experiments on the math symbol dataset. However, we will be discussing the results of few architectures in this paper. The most commonly used architecture had 3 convolutional layer each with 128, 64 and 32 filters respectively. These filters were of size 4 * 4, 3 * 3 and 2 * 2 respectively for each layer and had a stride of 2 * 2, 2 * 2 and 1 * 1 respectively for each layer. We experimented with different activation function such as ReLu and ELU – exponential linear Unit introduced by Clevert et al [11]. The number of filters, filter size, the filter stride, number of fully connected layers and number of neurons in the fully connected layer were all optimized using random search. We performed a global max pooling after all convolutional layer. The fully connected layers had 256 and 82 neurons. It may be worth noting that we used ELU only in the convolutional layers and not in the fully connected layers. This was mainly because our experiments revealed that using ReLu in the convolutional layer increased the problem of dead ReLu and the learning seemed to undergo plateauing after certain number of epochs. The use of a single ELU between the convolutional layers seemed to solve this problem. It's also important to note that training tends to speed up when using ELU in the network.

After we had optimized our network with grid search we performed more experiments by increasing the number of filters in each layer. We used equal number of filters in both the convolutional layers and trained for one epoch while noting the training time. We observed that there is an exponential increase in training time with the increase in number of filters without any drastic increase in accuracy of the network on the cross validation data. It's ideal to have higher number of filters to a certain extend but beyond that it just adds to the computational cost. The fig 3 shows the plot between number of filters and the training time.

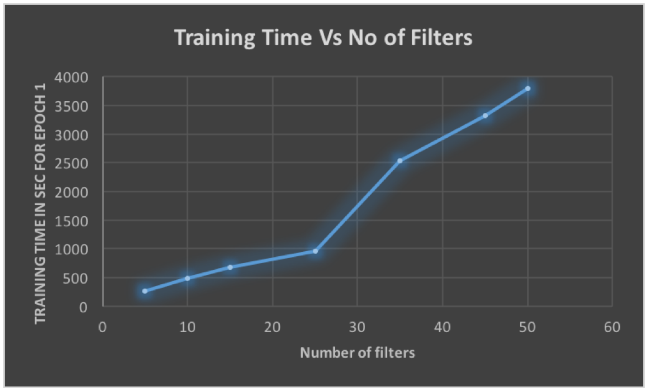


Fig 3: Number of filters Vs training time for each epoch.

The summary of our first architecture is shown in below fig 4.

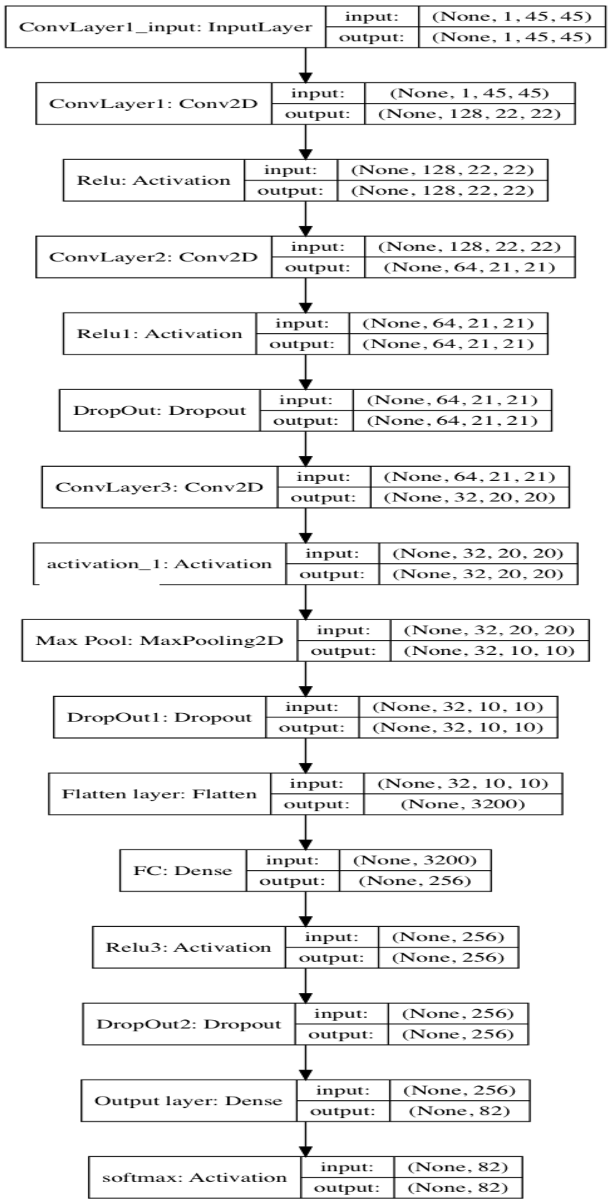
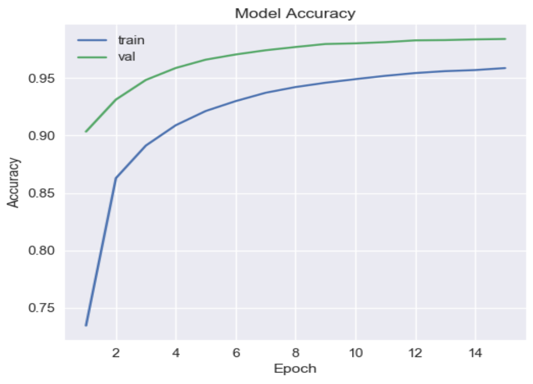


Fig 4: Summary of the CNN architecture.

The network in fig 4 gave us a total accuracy of 98.07% and the normalized confused matrix [fig 6] plotted for all the classes shows a very prominent diagonal as compared to the one with a simple fully connected neural network. The confusion among the classes is also very low but this network fails to classify some of the classes completely This network had only 882,866 trainable parameters and this doesn't seem to be sufficient to learn the skewed classes.



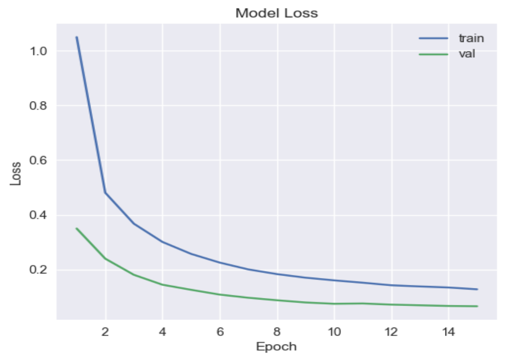


Fig 5: The upper plot shows the Train and Cross validation accuracy Vs Epochs. The network seems to have converged after 15 epochs. The below plot shows the train and cross validation loss against epochs.

Since our three convolutional layer architecture with extensive drop out, larger filter sizes and strides failed to classify skewed classes, we decided to experiment with a network with more parameters. we repeated the experiment with just two convolutional layer each with 45 filters, 3 * 3 filter size and smaller stride size of 1 * 1 and without drop out between convolutional layers. This resulted in a total of 2,333,426 trainable parameters. We were able to achieve an accuracy of 99.01 % and there was considerable improvement in the accuracy among the skewed classes. The [fig 7] shows the F1 score plot for the entire dataset.

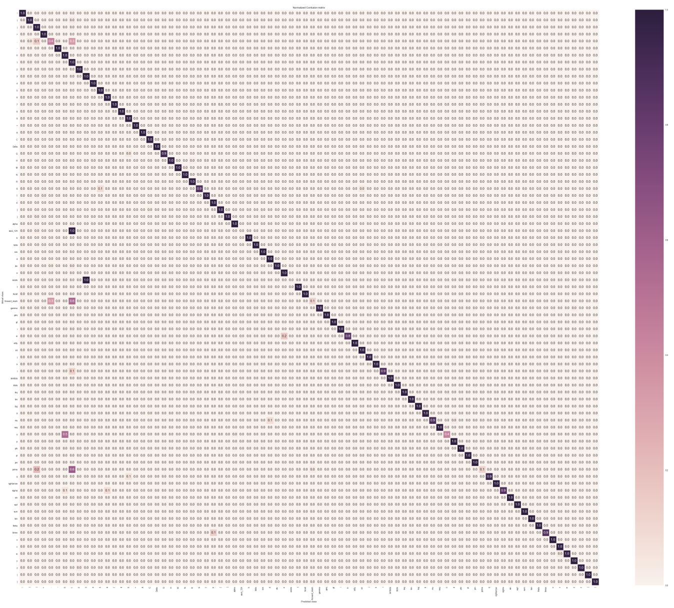


Fig 6 shows the confusion matrix for a five-layer convolutional network with three convolutional layer and two fully connected layers. The network had a total of 882,866 trainable parameters and achieved accuracy of 98.07%. Most of the classes are predicted accurately with high precision and recall, however some of the skewed classes are not classified correctly resulting in a F1 score of 0 in some cases.

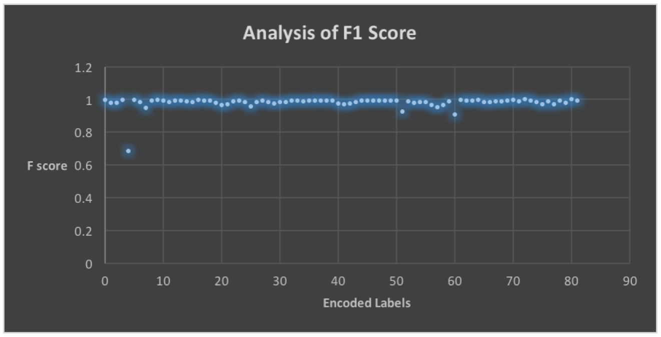
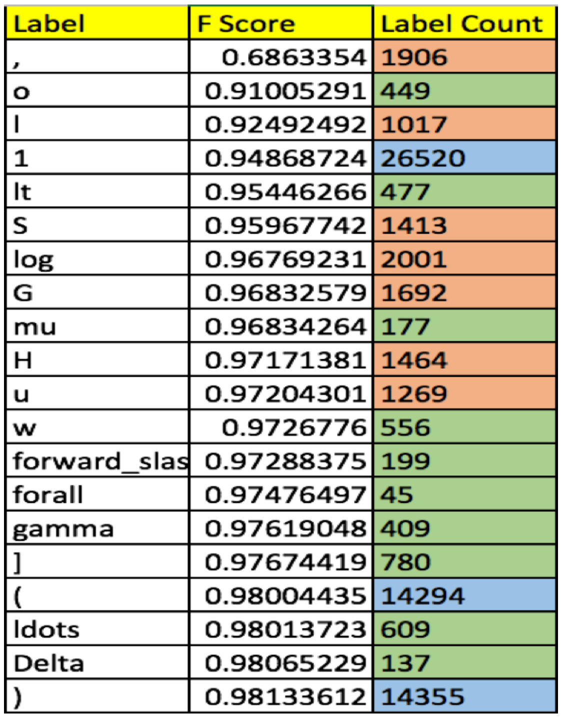


Fig 7: The scatter plot between encoded labels and F1 score. The two convolutional layer architecture with smaller filter size and strides helped to increase the number of network parameters to 2,333,426. This helped us improve the accuracy further from 98% to 99.01%. It's quite clear from the scatter plot that most of the classes included skewed seems have a boosted performance as compared to the network with less parameters.

We further conducted the analysis to identify the least performing classes in the above network. The results are summarized in the fig 8. It may be worth noting that the network successfully labels even the most skewed classes correctly in the test data.



Label	F Score	Label Count
,	0.6863354	1906
o	0.91005291	449
l	0.92492492	1017
1	0.94868724	26520
lt	0.95446266	477
S	0.95967742	1413
log	0.96769231	2001
G	0.96832579	1692
mu	0.96834264	177
H	0.97171381	1464
u	0.97204301	1269
w	0.9726776	556
forward_slas	0.97288375	199
forall	0.97476497	45
gamma	0.97619048	409
]	0.97674419	780
(	0.98004435	14294
ldots	0.98013723	609
Delta	0.98065229	137
)	0.98133612	14355

Fig 8: The table above shows the F1 score for the most wrongly predicted labels and their sample count in the dataset. It's quite clear that majority of the wrongly predicted labels are from the skewed classes.

We also attempted to incorporate Nestrov Momentum into our learning algorithm. NADAM is a variant of Adam which incorporates Nestrov momentum into the optimization algorithm. The update for the momentum variable in NADAM is performed using the current gradient updates as compared to average of previous n gradient updates in Adam. The two convolutional layer architecture was trained again replacing Adam with NADAM and performance metrics was calculated. The network performance dropped from 99.01 to 98.33%. However, the training was much faster and the network converged much faster (12 epochs as compared to 15 using Adam). Even though some of the results on MNIST seemed to argue to that NADAM is a better optimizer, our experiments prove that there is no universal optimizer and it varies from dataset to dataset. It may be also worth noting that its becomes important to hyper tune initial learning rate, initial momentum, batch size and other parameters associated with a learning algorithm before it can be effectively used to train a model. However, hyper tuning all these parameters is again computationally expensive.

We also visualized some of the filters which are used in the convolutional layer by passing random noise as input to the network. It helps us understand the patterns of these complex filters. These filters are shown in fig 9.

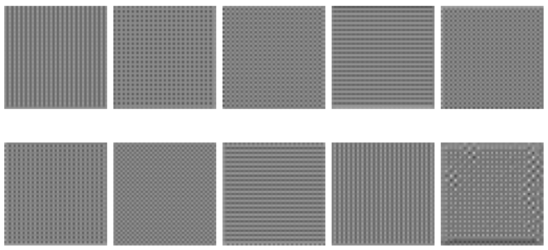


Fig 9: showing some of the filters generated in the first convolutional layer.

Summary of Convolutional Networks:

In this section, we experimented with a network with three convolutional layer and two fully connected layers along with larger filter sizes and strides resulting in fewer parameters using larger drop outs. We got results close to 98.07%. We increased the number of parameters in the network by reducing filter sizes and strides and reducing drop outs. The resulting trained networks could give results close to 99.01% accuracy on test dataset. Then we repeated our experiments with NADAM, a variant of Adam with Nestrov momentum and found that algorithm helps the network converge faster but reduces the accuracy of the network. We also observed that a network with larger parameters could learn skewed classes much better as compared to networks with smaller parameters.

c. Logistic Regression, SVM and RBM's.

We also tried to use Logistic Regression and SVM's to classify our mathematical symbols. We were successful in training a logistic regression model and a pipeline model combining a RBM with logistic regression on the same task. This was a proven method for MNIST dataset and is quoted as an example for the Scikit learn implementation of RBM's [12]. It has shown experimentally on MNIST dataset that pre training an RBM and then using logistic regression improved the classification accuracy to 93% as compared to 86% using logistic regression on raw pixels.

In our experiments with logistic regression and RBM's, we did not achieve good results as compared to CNN's. Logistic regression could only achieve a max 18% accuracy (chance accuracy: 1.2%) and model which was pre trained on a RBM with 200 hidden units with number of steps of contrastive divergence as 10 could achieve a minimum pseudo likelihood of -311 which is a considerably higher loss. When used in pipeline with a logistic regression, we could only get an accuracy of 22% approx. We believe that the low accuracy is mainly due to not optimizing the hyper parameters in pipelines especially the regularization parameter C in logistic regression and number of hidden units in RBM. The input training vector was of size 240000 * 2025 and this was highly computationally expensive and it was not feasible to grid search and tune the hyper parameter. When training and RBM with contrastive divergence, we try to minimize the pseudo likelihood loss. The fig 10 shows a plot between pseudo likelihood loss and number of epochs for a Bernoulli RBM.

It should also worth noting that when we attempted to train the SVM, the training machine eventually ran out of memory. So we were not able to achieve any results on SVM. The summary of all the experiments and their results can be found under fig 11.

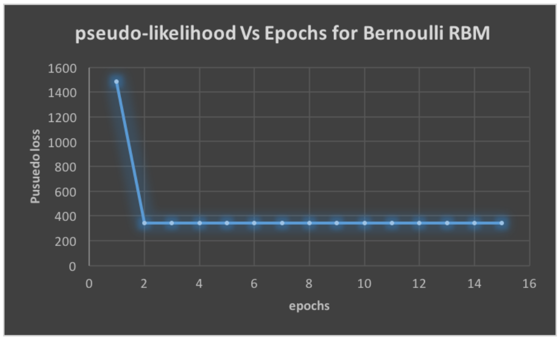


Fig 10: Pseudo likelihood loss for a Bernoulli RBM with 200 hidden units and 2025 visible units. It may be noted that the network seemed to achieve its lowest loss in one epoch and doesn't seem to make any improvements after the first epoch.

Algorithm	Details	Accuracy
Neural Net with no Conv' layer	Two layer, 512 and 82 Neurons, Adam Optimizer with drop out	76.95
CNN with 3 Conv' layer and 2 fully connected layers	Trainable parms : 882,866, Adam, Drop in every layer, large filter sizes and strides	98.07
CNN with 2 Conv' layer and 2 fully connected layers	Trainable parms : 2,333,426 , No Drop out between conv layer, smaller filter size and strides, Adam	99.01
CNN with Nadam instead of Adam as learning algorithm	Same like above but this Nadam	98.33
Logistic Regression	C = 100.0	18.37
RBM with Logistic Regression	Hidden layer : 200 Neurons, Batch size : 20, iterations : 15, Psuedolikelihood loss, C = 100.0	22
Chance	NA	1.2

Fig 11 – Shows the comparison and performance of different Machine learning algorithms and deep learning techniques on our math symbol dataset. The best network highlighted above is the one based on average F1 score and not the accuracy. We believe this is the right approach for our dataset which has many skewed classes.

V. Conclusion:

In this paper we discussed about how we used various neural network architecture such as CNN, RBM's, simple fully connected layers and also logistic regression to classify mathematical handwritten symbols. We analyzed our dataset and discovered that its extremely skewed for most classes. However, a four layer CNN was still able to achieve results close to 99% classification accuracy. The network was able to achieve perform decently well on skewed classes as well. We tried to use RBM's to classify, but did not achieve results comparable to CNN.

The biggest obstacle to our training was the large size of the dataset and very high computation and training time. This made it extremely difficult to hyper tune various parameters of logistic regression and RBM models.

VI. Future Work

There have been many new trends in symbol classification using RNN and we believe the same methodology could be applicable to math symbol generation. A committee for CNN working together to make joint prediction is widely used on MNIST dataset and this experiment could be tried on this dataset as well. We would want to improve our dataset and drastically reduce the number of skewed classes through data augmentation. There are many preprocessing steps such as elastic distortion, addition of random Gaussian noise, rotation and other image preprocessing techniques that could be applied to our dataset. We also would like to develop an efficient algorithm for parsing the symbols from mathematical expressions in the future.

VII. References

[1]. Cao Xinyan1,2, Yu Hongli,Wang Xin "Handwritten Mathematical Symbol Recognition Based on Niche Genetic Algorithm"

[2] Catherine Lu, K Mohan " Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Networks".

[3] Nicolas D. Jimenez,Lan Nguyen "Recognition of Handwritten Mathematical Symbols with PHOG Features".

[4] Stephen M. Watt and Xiaofang Xie "RECOGNITION FOR LARGE SETS OF HANDWRITTEN MATHEMATICAL SYMBOLS".

[5] CHROME 2016 Competition result: http://crohme2016eval.cs.rit.edu/rankings/.

[6] Kaggle Dataset : https://www.kaggle.com/xainano/handwrittenmathsymbols
CHROME Original: http://tc11.cvc.uab.es/datasets/CROHME-2014_2/task_2_1

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov – “Dropout: A Simple Way to Prevent Neural Networks from
Overfitting”.

[9] Diederik P. Kingma, Jimmy Ba “Adam: A Method for Stochastic Optimization”

[10] Timothy Dozat “incorporating nesterov momentum into Adam”.

[11] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter - “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”.

[12] Scikit-Learn RBM-Logistic pipeline – “http://scikit-learn.org/stable/auto_examples/neural_networks/plot_rbm_logistic_classification.html#sphx-glr-auto-examples-neural-networks-plot-rbm-logistic-classification-py”