Evaluating Handwritten Math Expressions from Images using Machine Learning models

Jose Joy
A53230620
University of California, San Diego
jojoy@eng.ucsd.edu

Ian Colbert
A11440921
University of California, San Diego
icolbert@eng.ucsd.edu

Mathew Sam
A53241787
University of California, San Diego
m1sam@eng.ucsd.edu

Abstract

The MNIST dataset is one of the most commonly used databases for training and testing machine learning models. Our project aims to extend this dataset containing 60,000 handwritten digits ranging from 0 to 9, to include mathematical symbols and, ultimately, test the effectiveness of various trained machine learning models to generalize to real handwritten mathematical expressions. We compare and analyze four models by their classification accuracy on a homemade set of handwritten mathematical expressions - Random Forest, AdaBoost, Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN).

1. Introduction

Our work aims to evaluate handwritten mathematical expressions using open source databases such as the MNIST dataset [12], the HaSy dataset [19] and the handwritten math symbols dataset (kaggle) [2].

For an accurate evaluation of the handwritten expressions, a network has to correctly classify every digit and symbol. This increases the complexity of our problem. While individual models perform well classifying data samples, the overall performance of our system is dependent on how well that model generalizes. To analyze the effectiveness of our system, we tested the models with the database of handwritten expressions.

2. Project Description

The main objective of our project was to evaluate handwritten mathematical expressions from images, using various machine learning models. For this purposes, we experimented with various classification models. Our project has been developed as a logical extension of ECE 271B course content. It aims to compare the effectiveness of the different machine learning algorithms such as bagging, ensemble learning, fully-connected and sparsely-connected feed-forward Artificial Neural Networks.

To do so, we implement the Random Forest Algorithm, the AdaBoost Algorithm, the Multi-Layer Perceptron and Convolutional Neural Network as respective examples from each class of learning algorithm. We analyze their performance in digit and symbol classification and compare their ability to generalize to our homegrown equation dataset.

We started with expressions involving many mathematical symbols like greater than, less than, division, exponents, π etc., but ultimately decided to go with just three symbols, addition, subtraction and multiplication. This was because, as after initial experiments, we realized that we didn't have enough data to properly identify the other symbols.

3. Related Work

Our project spans the field of image classification. By comparing the effectiveness of each method of learning multi-class discriminative models, we link together years of image classification research.

Bernard et. al [3] proposed the Random Forest classifier. We extend this research on the MNIST dataset to our own MNIST, HaSy, and Kaggle composite dataset.

Zhu et al. [21] extended the standard AdaBoost binary classification problem to a multi-class classification problem. We combine this with the work of Schapire et al. [16] in our two-stage AdaBoost classifier.

One of the best works in handwritten digit recognition comes from LeCunn et al [10] in his seminal work on hand-

written character recognition from

Springenberg et al. [17] attempts to simplify and generalize CNN architecture by replacing max pooling layers with more convolutional layers. Clevert et al. [7] proposes the Exponential Linear Unit and compares it to its counterparts. We also apply the research of batch normalization by Ioffe et al. [8] and dropout by Srivastava et al. [18] to optimize and accelerate training our deeper networks without over-fitting.

4. Data

Our test set of handwritten mathematical expressions is all homegrown. To expedite the process, we wrote multiple expressions, evenly separated in sheets of paper to be processed and evaluated. After dividing the expressions into single images, we separate individual digits and symbols using connected components. These individual characters are rescaled to a 28×28 image (dimensions of images in MNIST dataset [12]). We also used morphological operations like dilation and erosion to improve the segments corresponding to handwritten expressions, along with zero padding to make the images match the images in training set.

4.1. Preprocessing

Efficiently preprocessing the data took a degree of creativity, since the data from each source had different sizes, shapes and image resolutions. The insights gained from the experiments pertaining to preprocessing are listed below:

- Padding of images fed into the system is very important for models. We observed that the average image in the MNIST dataset has a padding of 5 on each edge. Since the MNIST data forms the largest subsection of the data we use, we bring the other images used for training to the same standards as the MNIST.

- The characters in the handwritten expressions made from scratch required a lot of preprocessing. The amount of padding and size of the kernels used for morphological operations required depended on the handwriting of the person who wrote the expression and the degree of resolution with which the image was taken.

4.2. Versions of models generated

The MNIST dataset is one of the most common open sourced databases for image classification and, as such, is our source for numbers in our training set for mathematical expressions [12].

We extend our training set to include mathematical symbols by adding the HaSy dataset [19]. Unfortunately, while there are roughly 6000 images for each number in the MNIST dataset, in HaSy, there are at best 500 images for each symbol. Although the images were of high quality, training models on this merged dataset resulted led to heavy bias on digits and misclassification on symbols.

In an attempt to balance the class distribution, we added the Kaggle dataset [2], which is of poorer quality and requires slightly more preprocessing than the images pulled from the HaSy dataset but contains far more images for each set of mathematical symbols.

Ultimately, we ended up with an exploration versus exploitation problem. To better understand the effects of quantity of data versus quality of data, we fit two models of each architecture and algorithm using two different subsets of training data. For both subsets, the data was randomly sampled from each dataset, combined and shuffled.

- **Subset version1**: 20,000 samples from the MNIST dataset and all the images belonging to symbols +, - and $\times$ from the HaSy dataset

- **Subset version2**: full 60,000 samples from the MNIST dataset and all the images belonging to symbols +, - and $\times$ from the Kaggle dataset

5. Experiments

This section delves into how we tune our models to maximize network performance across each algorithm.

5.1. Random Forests

The Random Forest algorithm is a form of bagging. It fits a number of decision trees on various subsamples of the data, drawn with replacement, and uses a majority voting system for network prediction [3, 14]. While the randomness of the split slightly increases the bias, using multiple trees reduces the overall variance. For this machine learning model, we experimented with class weights and number of trees [3, 14].

5.1.1 Structure Used

We experimented with multiple structures by changing the number of decision trees in the forest and maximum depth of each decision tree. By selecting the optimum value of maximum depth we can prevent over-fitting and under-fitting by a single decision tree. By limiting the number of decision trees in the forest, we can prevent over-fitting to noise in the dataset [14]. For this project, an ensemble of 2000 trees were used with maximum depth of 4 gave us the best results in the validation set.

5.1.2 Inferences from experiments

- **Class Weight**:
  When class weights weren't balanced, the model was heavily over fitting to digits.

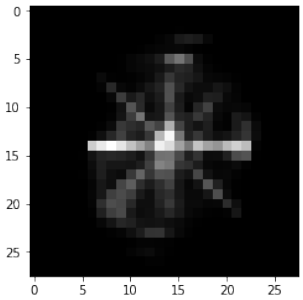

Figure 1: Feature Importance plot of Version1 model

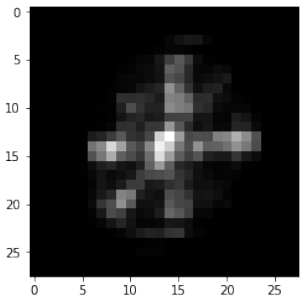

Figure 2: Feature Importance plot of Version2 model

Version used	Error in validation set
Model version1	19.4%
Model version2	18.75%

Table 1: Comparison of models generated using random forests

- **Number of Trees**:
  Increasing the number of trees improved validation accuracy, till a point after which the model wouldn't show much improvement in accuracy. Between 2000 and 5000, the accuracy of the model on validation set didn't improve much.

5.2. Adaboost

Our AdaBoost model leverages the work of Zhu et al. [21] to implement a multi-class discriminative model. At every iteration, decision tree stumps are added to improve accuracy [16, 21]. The stumps are added based on re-weighted misclassified examples. The more times an example is misclassified, the larger its weight.

5.2.1 Structure

For our ensemble learner, we fit a 2 stage AdaBoost algorithm to classify digits and symbols. The first stage was a symbol-digit detection network, attempting to correctly classify whether an image was either a symbol or a digit. The second stage was trained to recognize the specific digit/symbol. The structure is explained more clearly below:

- **Stage 1**: This stage discriminates between digits and symbols, achieving an accuracy of 100%. This stage fits 250 decision stumps.
- **Stage 2**: This stage uses 2 different models, one to recognize/discriminate between digits and one to discriminate between symbols.
- **Stage 2.1**: The model used for recognizing digits has 500 stumps
- **Stage 2.2**: The model used for recognizing symbols has 500 stumps.

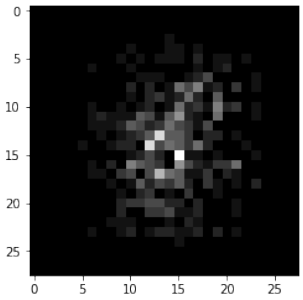

Figure 3: Feature Importance plot of Version1 model (stage1)

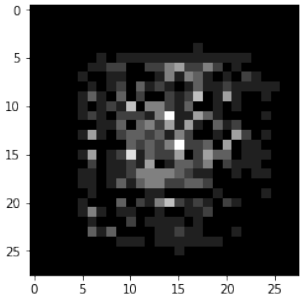

Figure 4: Feature Importance plot of Version2 model (stage1)

5.2.2 Inferences from experiments

- **Structure**:
  Multistage structure was able to give us better results in the validation set. The first stage achieved 100 % accuracy on validation set

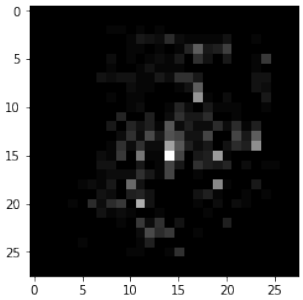

Figure 5: Feature Importance plot of Version1 model (stage2.1)

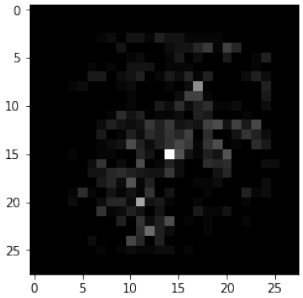

Figure 6: Feature Importance plot of Version2 model (stage2.1)

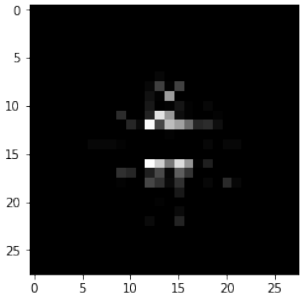

Figure 7: Feature Importance plot of Version1 model (stage2.2)

Version used	Error in validation set
Model version1	18.1%
Model version2	17.75%

Table 2: Comparison of models generated using adaboost classifiers

- **Number of Classifiers**:
  Validation accuracy improved with number of classifiers, after a point which the accuracy stagnates. After increasing the number of classifiers in stage 2 from 500 to 1000, we didn't observe significant increase in validation accuracy.

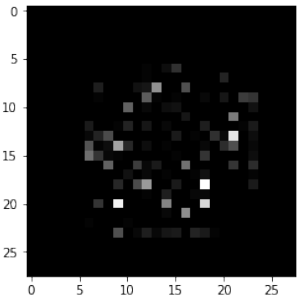

Figure 8: Feature Importance plot of Version2 model (stage2.2)

5.3. Multi-Layer Perceptron

A Multi-Layer Perceptron aims to map the input space to a different representation so as to find a hyperplane that best separates the data in that representation using linear and non linear transformations.

The data used for each model was split into a 80:20 ration for training and validation.

5.3.1 Structure

- Used a 4 layer structure with 3 hidden layers and one output layer.
- Each hidden layer has batch normalization,relu activation and a dropout of 0.3
- There were 256 neurons in the first layer, 128 in the second and 64 in the last hidden layer. The output layer used softmax activation and had 13 neurons for the 13 different classes.
- An adam optimizer was used with cross entropy used as a measure of loss.
- The model was trained with a batch size of 256 for 50 epochs.

Version used	Error in validation set
Model version1	2%
Model version2	3%

Table 3: Comparison of models generated using MLP

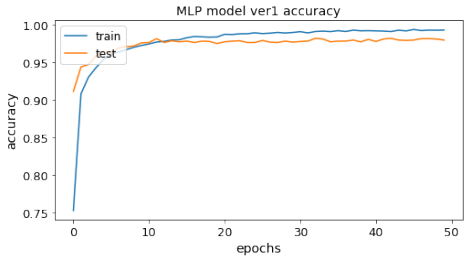

Figure 9: Accuracy vs iterations for version1

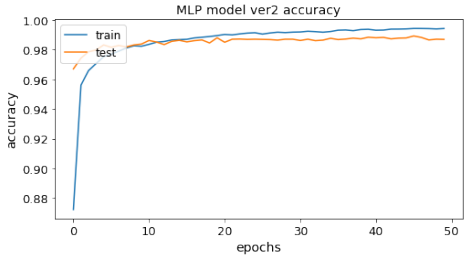

Figure 10: Accuracy vs iterations for version2

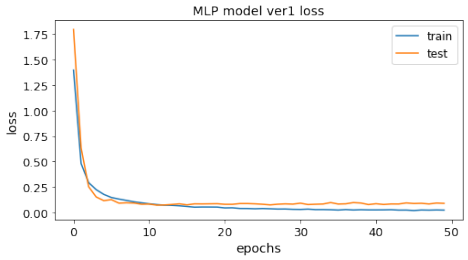

Figure 11: Loss vs iterations for version1

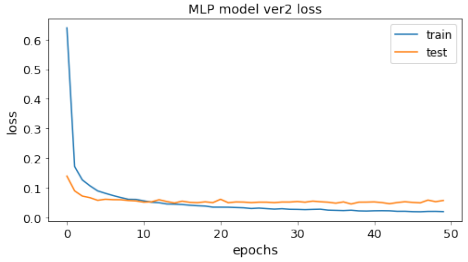

Figure 12: Loss vs iterations for version2

5.3.2 Inferences from experiments

- **Depth**:
  We noticed that by increasing the number of hidden layers from 1 to 3, we were able to increase our accuracy substantially from 93% to close to 97%. However, increasing the number of hidden layers to 4 seemed to be overkill and did not improve performance.
  As we increase the number of hidden layers, we observe that the MLP can fit more complex functions allowing an increase in accuracy. Increasing it too much can lead to over fitting and must be avoided.

- **Activation**:
  We see that the ReLu activation function performs better than the sigmoid activation function as it no longer faces the problem of having a flat region in its activation function in the positive region.

- **Batch Normalization**:
  Batch normalization creates an obvious improvement to the performance of the network. Batch normalization selectively scales the inputs to a layer to a region that best benefits the network. The scaling parameters used in each layer are learnt by the network [8].

5.4. Convolutional Neural Networks

Convolutional Neural Networks(CNN) are sparsely-connected artificial neural networks that attempt to learn the contours and features of an input image by learning weighted filters. CNNs often have a lot of advantages over MLP as they leverage pixel locality and introduce a degree of translational invariance [1, 5, 10].

Lecun et al. [10] was able to use a three layer neural network with back propagation for digit recognition. Of the three layers two were convolutional layers and the other was a fully connected layer. While this network achieved 5% error on the test set tailored to digit recognition, we have achieved 1.5% error on our test set combining both digits and symbols by using a more complicated network.

For training, our dataset was split in an 85:15 ration for training and validation.

5.4.1 Structure

- Our highest performing CNN architecture uses 3 convolutional layers and 3 fully connected layers.

- The convolutional layers use 256, 128, and 64 filters, respectively, with a 5x5 filter size on the input layer and 3x3 for both of the other convolutional layers.

- The Leaky ReLu activation function was used for each convolutional layer while the ReLu was used for all fully-connected layer other than the output layer which used the SoftMax activation function. Max pooling was only used on the last convolutional layer.

- Batch normalization was used on all layers except the output layer.

- Dropout was used on all layers except the activation function. Each convolutional layer used a dropout of 20% while each fully-connected layers used a dropout of 10%.

- The model was trained with a batch size of 128.

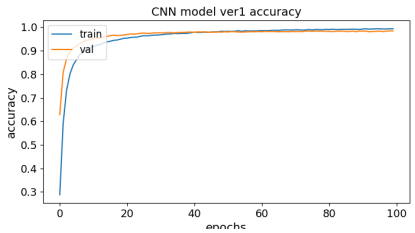

Figure 13: Accuracy vs iterations for version1

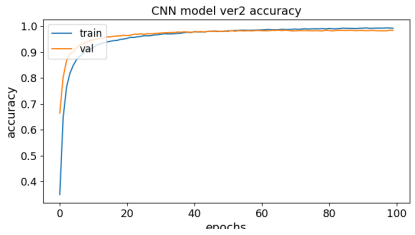

Figure 14: Accuracy vs iterations for version2

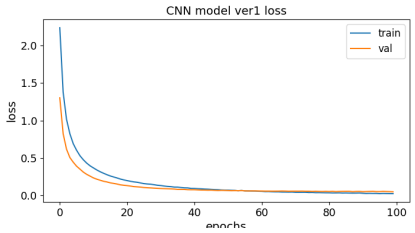

Figure 15: Loss vs iterations for version1

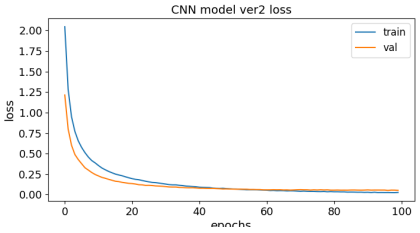

Figure 16: Loss vs iterations for version2

Version used	Error in validation set
Model version1	1.5%
Model version2	1.3%

Table 4: Comparison of models generated using CNN classifiers

5.4.2 Inferences from experiments

- **Network Strength**:
  We found that, much like the MLP, deeper networks performed better. However, too many filters in the convolutional layers led to over-fitting and actually decreased network performance on inference. To reduce over-fitting, we added dropout layers of 20% to each of the convolutional layers and 10% to each of the fully-connected layers [18, 9].

- **Activation Function**:
  We used the Leaky ReLu function on each of the convolutional layers and the ReLu on each of the fully-connected layers. Using the Leaky ReLu for convolutional layers resulted in higher accuracy and faster convergence, compared to tanh and sigmoid activation functions. We experimented with using the ELU proposed by Clevert et al. [7].

- **Batch Normalization**:
  Similar to the MLP, we used batch normalization in all convolutional and fully-connected layers. According to Ioeffe et al. [8], batch normalization acts as a regularizer and makes training more invariant to initializations. Implementing batch normalization led to a significant improvement in classification performance with deeper networks as there are more parameters to initialize and a greater risk of over-fitting.

- **Training Epochs**:
  Over-fitting to our data was a common problem across all of our networks. When training the CNN, we found that training networks with more than 3 layers for over 50 epochs led to over-fitting and a degredation in network performance.

- **Pooling**:
  CNNs have been conventionally implemented using information pooling after the applying layer filters [1, 9]. According to Krizhevsky et al. [9], the addition of the max pooling layers "summarize the outputs of neighboring groups of neurons." We found this to be harmful in the first few convolutional layers as there is a substantial amount of information loss in pooling functions. Because of this, we only used Max Pooling on the final convolutional layer.

6. Results

The test data was acquired using our own hand written expressions as well as that of colleagues and friends. Images of these expressions were taken and preprocessed to get separate blocks with the image of each digit and symbol. Once we were successful in extracting these images, we passed the extracted images to trained models for each machine learning algorithm. There were 96 equation images in test set, with around 450 symbols and digits. The symbols and digits were first extracted from the images and fed to the models. The model output was converted to strings, which were then converted to equations and evaluated. The code and data can be found in the following github repository `https://github.com/JoseJoy249/Handwritten-Expression-Evaluation`

6.1. Performance on test set

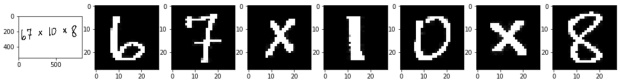

Figure 17: Original equation and extracted segments

`Equation = 6 7 x 1 0 x 8`

	RF	AB	MLP	CNN
ver1	02x+0xx	23+8086	67x10x8	67x10x8
ver2	0+x+0xx	53x80x5	67x10x8	67x10x8

Figure 18: Predictions of various models on equation

The best performance we have achieved is tabulated below

Performance	RF	AB	MLP	CNN
overall	0	1.0	22.6	52.5
digit	29	40.6	74	84.0
symbol	72.1	7.6	55	86

Table 5: Performance of each model version1 in percentage

Performance	RF	AB	MLP	CNN
overall	1.0	3.0	39.1	68.0
digit	25.5	40.4	79.0	90.1
symbol	92.1	50	74.0	88.4

Table 6: Performance of each model version2 in percentage

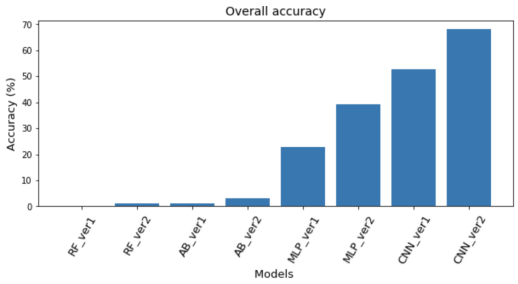

Figure 19: Overall accuracy of models in evaluating expressions

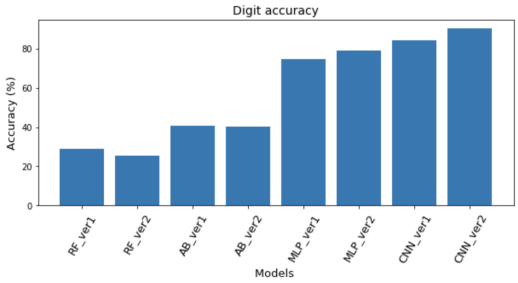

Figure 20: Accuracy of models in identifying digits

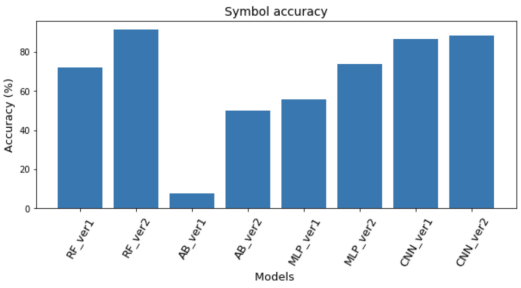

Figure 21: Accuracy of models in identifying symbols

6.1.1 CNN

What we observe is that the CNN performs the best when it comes to evaluating entire expressions as it is the most consistent regarding both numbers and symbols. This is because of its ability to view connected components through its filters. As the number of convolutional layers increase, the receptive field size increases allowing the network to recognize structures far more easily.
When we observe exploration versus exploitation, we see that as we get more data for a deep network, the better it learns. allowing it to learn more complex functions with better accuracy. However, the version 1 model of CNN comes very close to version 2 model of CNN for digits and symbols.This is because of the superior quality of images in version 1.

6.1.2 MLP

Multilayer perceptron(MLP) performs better than Random forests and Adaboost. We noticed that varying the depth improves performance. This is expected as a deeper network can better approximate a more complex functions. The MLP takes pixel value information and uses it to find the best features. However, it fails to acquire structure information and lacks the degree of flexibility of CNNs.Falling short of the performance seen in CNN models.
We also observe that more data does drastically improves the performance of the model. However, the increase in the quantity of data drastically improves the performance on symbols from version 1 models to version 2 models by about 19%. What we can conclude from the above is that quality of images affect performance of CNNs more than MLP.

6.1.3 Adaboost

While the adaboost model was able to perform well in validation set, its performance in the test set of handwritten images was poor. This is probably due to the fact that the model wasn't able to generalize well to test samples.

6.1.4 Random Forests

While random forests perform extremely well on symbols (even better than CNN), due to its poor performance on digits, it is unable to correctly evaluate the expression.

7. Conclusion

We see from the results that more data with reasonable quality is better than very low amount of good quality data, as the version2 models were able to perform much better than version1 models. The CNN model trained on the most data had the best performance. MLP model was able to achieve good performance due to the superior quality of features extracted. Random Forest and Adaboost models weren't able to achieve good results due to the poor performance in identifying digits and(or) symbols. It can also be inferred from the results that MNIST is a good dataset that can generalize well for handwritten digits.

8. Contribution of each member

Everyone contributed equally to the project.

References

[1] Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io/convolutional-networks/`. Accessed 2018-03-20.

[2] Handwritten math symbols dataset. `https://www.kaggle.com/xainano/handwrittenmathsymbols/data`. Accessed 2018-03-21.

[3] S. Bernard, S. Adam, and L. Heutte. Using random forests for handwritten digit recognition. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 1043–1047. IEEE, 2007.

[4] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 77–82. IEEE, 1994.

[5] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1135–1139. IEEE, 2011.

[6] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big multilayer perceptrons for digit recognition. In *Neural networks: tricks of the trade*, pages 581–598. Springer, 2012.

[7] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[11] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[12] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[13] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

[14] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[15] H. Mouchere, C. Viard-Gaudin, R. Zanibbi, U. Garain, D. H. Kim, and J. H. Kim. Icdar 2013 crohme: Third international competition on recognition of online handwritten mathemat-

ical expressions. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1428–1432. IEEE, 2013.

[16] R. E. Schapire. *Explaining AdaBoost*, pages 37–52. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[17] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[19] M. Thoma. The hasyv2 dataset. *CoRR*, abs/1701.08380, 2017.

[20] R. Zanibbi and D. Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357, 2012.

[21] J. Zhu, S. Rosset, H. Zou, and T. Hastie. Multi-class adaboost. *Ann Arbor*, 1001(48109):1612, 2006.