EMNIST data classification: A Question Mark

Jacob Adley Zac Monroe Zunaeed Salahuddin

Research paper for CSCI-B351

Intro to Artificial Intelligence

Fall 2018

Abstract

With regards to classifying large-scale datasets, especially those made up of visual imagery, the modern Convolutional Neural Network reigns supreme. The combination of a feed-forward style neural network and an architecture that exclusively takes images as inputs ensures a highly efficient system that requires minimal user preprocessing. Most notably, the convolutional neural network included features that promote generality of results. This paper outlines a specific implementation of a Convolutional Neural Network, trained on the EMNIST dataset, as well as two other preliminary approaches to the classification problem. The images include handwritten characters: Arabic numerals 0-9, lowercase letters a-z, and uppercase letters A-Z. Benchmark results and runtimes are presented to further outline and provide context for the accuracy and efficiency of our implementation.

I. Introduction

We chose to implement a variety of methods for character recognition of handwritten numbers and letters (uppercase and lowercase). We are using the EMNIST data set sorted by class (62 classes: 10 numbers, 26 uppercase letters, 26 lowercase letters) containing 814,255 labeled character images of size 28px x 28px. The methods implemented were a nearest neighbor style method and two neural networks, one being a basic feed forward network and another a convolutional neural network.

II. Technique 1: Nearest Neighbor

Disclaimer: our classifier was not a proper implementation of a Nearest Neighbor method as such an implementation would be very slow. Despite perhaps being able to achieve better results from a k-th Nearest Neighbor method, for each character image that we were testing, we would have to iterate over the entire training set, which was too time complex to even warrant implementing. Instead, we simply iterate over the training set once and create an "averaged" version of each of the classifiers. Then, we can iterate over the testing data set once and for each only have to find the nearest neighbor of 62 items (one for each classifier). With this method, the time required is O(n) (where n is the size of the data set). This is because we iterate over the training set once and the testing set once, for each we iterate 62 times (which is constant). The space required is minimal, simply an array of 62 28x28 arrays. The results were lackluster though: just under 40%. This low accuracy is limited by the simplicity of the method -- we do not consider the features of the characters, but rather simply the averaged pixels that they occupy in the image. This gave us a baseline as we moved on to other methods.

III. Technique 2: Feed-Forward Neural Network

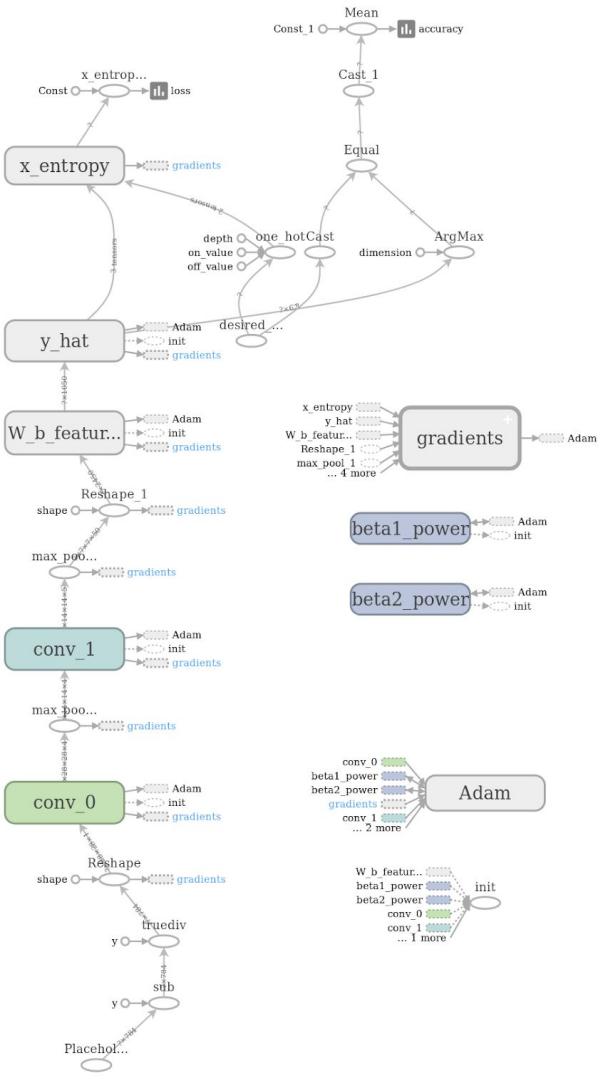
Our second stage of development was to make a vanilla feed-forward neural network classifier. We wanted to see how such a widely-used, general-purpose machine learning algorithm worked (inside and out), so we decided to build it from scratch (aside from use of numpy array data structures). Each level of the neural network first applies a linear transformation (via matrix multiplication) to the neuron vector (with the first layer being the image input), adds a bias vector of appropriate length, and pushes the result through a non-linear "activation function." This is how a class is generated from the network based on an image. When training, the final neuron layer (after being pushed through the activation function) is then compared to the expected outcome class, and the network updates itself recursively using the backpropagation algorithm implemented with stochastic gradient descent and a decaying learning rate. We used a hyperbolic tangent activation function[8] which takes numbers from (-inf, inf) → (-1, 1) in a non-linear fashion. Originally we were using the sigmoid activation function, but tanh ended up giving better results.

Our implementation of a vanilla neural network is one that trains slowly (compared to a TensorFlow- or Keras-based model), but in linear time and constant space. The time it takes the network to train is directly proportional to the size of the dataset (because each data point is used in training exactly once), and space used is constant (data points are forgotten about after the algorithm trains on them).

Some major limitations of the neural network were that it was unable to account for some subtle differences between images that seem completely transparent to humans, e.g. a character being drawn to the left by 2px or rotated by 30°. No matter how many hidden neurons we tried to

use, our error rate remained high because of these subtle differences between training points. Alternative solutions exist that help combat this problem: k-nearest-neighbors and convolutional neural networks. A k-nearest-neighbor algorithm would've been very costly (in time and space) to implement (given the size of our dataset), and a CNN is the current state-of-the-art algorithm for image recognition, so we decided on using a deep convolutional neural network for our flagship classifier.

IV. Technique 3: Convolutional Neural Network



The final stage of development entailed the implementation of a Convolutional Neural Network. We went with the TensorFlow API as our main code framework, being that it provides for fast computation with NVIDIA's CUDA parallel-computing platform and makes it comparatively simple to create convolutional layers and optimize network parameters with built-in optimizers. We used a network with the following structure/ordering:

1. Linear normalization of each input image from [0, 255] → [-1, 1]
2. Reshaping of flat input data to square tensor

3. Convolution with 40 kernels
4. Max pooling
5. Convolution with 50 kernels
6. Max pooling
7. Reshaping to flat vector
8. Fully-connected feature layer with 1050 neurons
9. Fully-connected feature layer with 62 neurons

Our reasoning for using two convolutional layers has to do with the size of our input images as well as a need for complexity. Had we used any more convolutional + pooling layers, the resulting feature images would have been too small and simple to work with; had we used only one, the network would not have been complex enough, and thus would have a much higher error rate in classification. The reasoning for having two fully-connected layers at the end is similar: we wanted some complexity, but too many layers (and having significantly fewer neurons in the second-to-last layer) gave us comparatively poor results.

Normalizing the input images drastically improved the accuracy of the CNN; our loss-value (determined by cross-entropy validation; idea to use cross-entropy for classification from source 4) decreased 100-fold immediately. This likely has to do with how convolutional filters operate, giving high results when negative pixels should be negative and positive pixels should be positive[5]. We also used a RELU activation function after each convolution, to eliminate areas in the convolved features in which there were negative values (which only made sense in the input layer).

The TensorFlow optimizer that we used was the current de-la-moda AdamOptimizer, which utilizes a unique method of updating the learning rate to get out of local minima of the loss function and more quickly find a global minimum with minimal overfitting[7].

Convolutional neural networks (or at least the one that we developed) classify an image by first passing several filters (kernels) over an image. These filters are designed and trained to look for specific features in an image (layers deeper in the network learn more abstract features); pixel values in the output of a convolutional layer are higher in places where proper features are found and lower in places where features are not found. The size of the image is reduced in this computation. After convolution, the layer is passed through a max-pooling layer, in which the maximum pixel value from a 2x2 square is taken to be the actual value of that square area (thus further reducing the size of the image). At the end, pooled-and-convolved features are flattened into a single high-dimensional vector, and the rest of network is as simple as the feed-forward implementation above.

Some limitations with this model have to do with the dataset: if we had images that were larger than 28px x 28px with more clarity, we would have a much easier time classifying images that look very similar at a low resolution (i.e. ‘o’ and ‘O’ and ‘0’; ‘Z’ and ‘2’) -- however this would make the computation/training time drastically longer (due to having more input dimensions from more pixels). Convolutional neural networks are the current state of the art of image classification, and not many alternatives can compete with their accuracy.

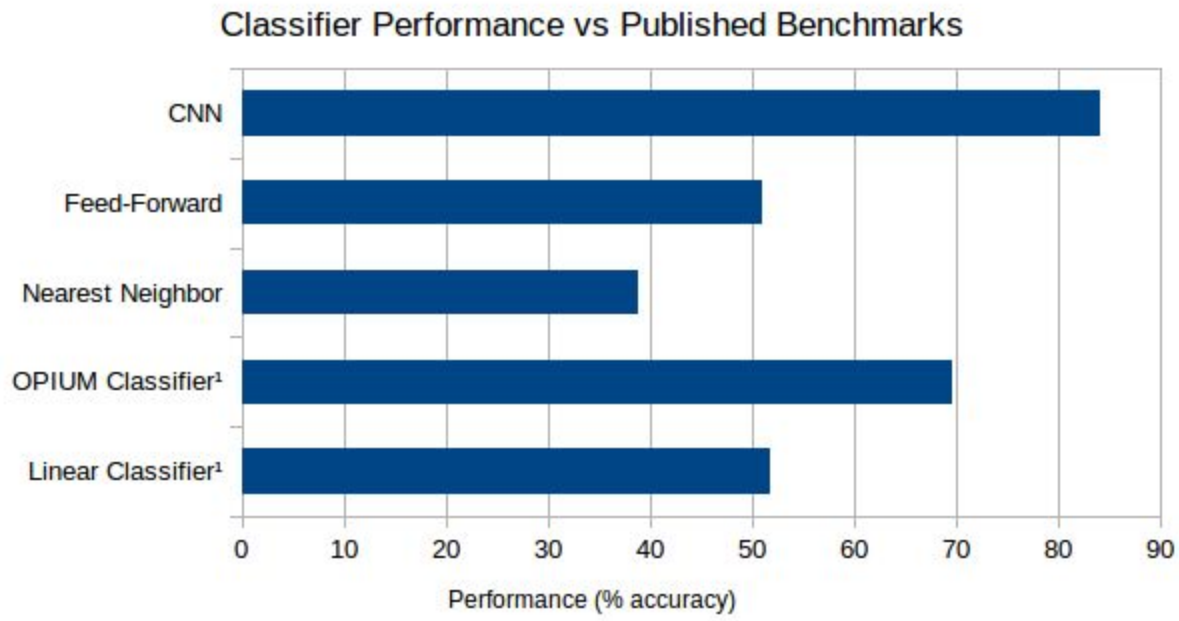
V. Results

By separating the dataset into training and testing sets, we were able to test our models:

- Nearest Neighbor:

- Accuracy: 38.98%
- Training time: 15:28 min

- Feed Forward:
  - Accuracy: 51.09%
  - Training time: 24:39 min
- Convolutional
  - Accuracy: 84.32%
  - Training time: 08:58 min



These results were as we expected. Being that the Nearest-Neighbor classifier is a a very naive method, we did not expect great results. Similarly, we were happy with the results from such a simple implementation of a Feed-Forward Network using mere stochastic gradient descent. Ultimately we were very happy with the results from the Convolutional Neural

Network, and were not surprised that it was dominative over the other two methods by such a wide margin.

VI. Expansion / Improvements

The most significant improvement that we could have made, for additional accuracy, is the combination of several Convolutional Neural Networks in parallel (5 or more). In a study conducted in Ukraine, an implementation of five 6-layer Convolutional Neural Networks, trained on the (non-extended) MNIST dataset yielded an incredible 0.21% error rate[3]. Had we had the time and computation real-estate to develop and train such a model, we could've gotten even higher accuracy than we did with our single deep neural network.

VII. Conclusion

Clearly the Convolutional Neural Network was the most effective of the methods that we implemented. We presume that this was most likely because it was the method that most accurately accounted for the features of a given character, rather than the general format of the image as a whole. Not only this, but this method would most likely do much better with added noise in the image, as well as it does not require normalizing the image into a constant form where the character is of the same size and location in the image. This point is of great importance with regards to simplicity of implementation in a more practical setting compared to such an ideal data set.

VIII. References

1. Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373
2. TensorFlow Documentation. (n.d.). Retrieved December 01, 2018, from https://www.tensorflow.org/api_docs/python/tf
3. Романюк, В В. (2016). Training Data Expansion and Boosting of Convolutional Neural Networks for Reducing the MNIST Dataset Error Rate. *Research Bulletin of the National Technical University of Ukraine "Kyiv Polytechnic Institute", 0*(6), 29-34. doi:10.20535/1810-0546.2016.6.84115
4. Loss Functions. (n.d.). Retrieved December 01, 2018, from https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html
5. Pokharna, Harsh. "The Best Explanation of Convolutional Neural Networks on the Internet!" *Medium.com*, Medium, 28 July 2016: https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8
6. Sanderson, Grant. "Deep learning" (2017, October 05). Retrieved December 01, 2018, from https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
7. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. (2018, November 25). Retrieved from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

8. Stansbury, D. (n.d.). Derivatives for Common Neural Network Activation Functions. Retrieved December 01, 2018, from https://theclevermachine.wordpress.com/tag/tanh-function/