Using Neural Networks for Character Recognition

Kevin Luikey
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
kluikey@gatech.edu

Emily Chang
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
echang60@gatech.edu

Thao Dinh
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
thaolbdinh@gatech.edu

Yinghui Dong
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
ydong99@gatech.edu

Abstract

Neural Networks are a powerful tool for learning complex, nonlinear relationships between inputs. Vaguely inspired by the biological neural networks present in the human brain, neural networks are particularly useful in image recognition. Given sufficient numbers of labeled training examples, it can learn to recognize patterns and generalize to new data. Convolutional neural networks are the most popular neural network approach when it comes to computer vision, and are capable of classifying images with little or no preprocessing of the data.

1 Motivation

Optical character recognition has a number of practical applications, such as, bank-check processing, postal address interpretation, license plate scanning, book scanning, digitizing paper records, etc. So many records in the world are transmitted via paper, like passport documents, invoices, bank statements, receipts, business cards, mail, etc. Essentially any task which requires interpreting printed or handwritten records can be streamlined and improved by employing optical character recognition techniques.

2 Related Works

Feature extraction: Feature extraction relies on humans identifying features that they believe to be relevant to the image's classification, such as aspect ratio, percent of pixels above horizontal half point, percent of pixels to the right of vertical half point, average distance from image centre, etc. then training on those features [2]. This approach is more labor intensive to set up because humans have to manually identify features that may be relevant to the classification. This differs from our approach that does not involve any human preprocessing of the data.

Clustering: The purpose of clustering is to divide a dataset into different clusters (subsets) given a prior number of clusters. The clustering method has two properties- homogeneity and heterogeneity. Homogeneity means that data belonging to one cluster should be as similar as possible, while heterogeneity means that data belonging to different clusters should be as different as possible. K-means clustering has been used to group sets of characters. Clustering is an unsupervised learning approach which does not use class labels and also does not return a prediction of a class label [3].

Preprint. Work in progress.

Ours is a supervised learning approach which trains on labeled data, and can output a predicted class for new data points.

T-SNE: T-distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm for data visualization. It is used to reduce the dimensionality of data to two or three dimensions, such that similar data points are close together in the lower dimensional space and dissimilar data points are farther apart. Thus, the data can be seen as clusters in 2D or 3D space as is shown below with a limited example of just digits 0-9.

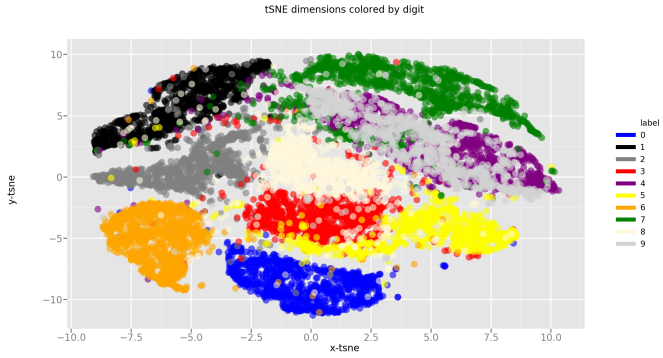


Figure 1: t-SNE 2-Dimensional data visualization

3 Approach

3.1 Data Description

The expanded mnist dataset is a set of over 800,000 labeled examples of handwritten characters a to z, A to Z, and 0-9 for a total of 62 unbalanced classes.

3.2 Techniques

3.2.1 Fully connected neural network

Our first approach was to build and train a fully connected neural net to classify the characters. There was no preprocessing of the data other than to reshape it to fit as an input to our model. We structured our neural net so that every pixel was an input neuron to the model, taking in the intensity of that pixel, a floating point value from 0 to 1. We initialized the starting weights randomly with a bias toward smaller values in hopes of reducing the complexity of the trained model, making it less likely to overfit. We primarily used two activation functions, sigmoid and reLU. Sigmoid was originally very popular in neural nets, but recently reLU, defined as f(x) = max(0, x), has been more commonly used. reLU has some useful properties that make it more appealing than sigmoid, such as being a more computationally efficient calculation, and dropping out useless neurons, as it outputs 0 when its input is negative, and will not change due to its derivative also being 0 when negative. We also experimented with different numbers of neurons per layer. Finally, our output layer contained one neuron for each class, where the label vector for a given class would be a 1 for the corresponding neuron and 0 for all others.

3.2.2 Convolutional neural network

We also experimented with the effectiveness of convolutional neural network models, comparing their effectiveness with fully connected artificial neural networks in optical character recognition. Our convolutional neural network consisted of 7 layers - input layer, convolutional layer, pooling layer, regularization layer, flatten layer, fully connected layer, and output layer. The convolutional layer creates feature maps, which are created by taking a filter matrix, smaller than the image, and doing element-wise multiplication between the filter and the image and summing the outputs. After

doing this over the whole image, we get a feature map. After that, the pooling layer reduces the dimensionality of the feature maps from the previous layer in a number of ways (e.g. maxing, averaging, summing, etc.) For our implementation, our regularization layer used dropout. Dropout randomly ignores neurons in the training phase to prevent overfitting. The flatten layer is used to convert the 2D matrix into a 1D matrix to be classified and processed in the fully connected layer. The fully connected layer connects every neuron from the previous layer into every neuron in this layer. Conceptually, this layer holds the information that is most important from all of the convolutional layers. This way, our model is able to better classify the images by simplifying the features from the convolutional layers and, at the same time, learning non-linear combinations of those features [1]. In our case, we had 128 neurons in our fully connected layer. Our model used a softmax activation function which makes all the outputs sum to 1, making each output neuron have a probability-like value, with the output layer consisting of one neuron for each class as with our first approach.

3.2.3 Principal component analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components, thereby reducing the dimensionality of the training data. We also experimented with performing PCA on the training examples in order to reduce the dimensionality of our data before training our fully connected neural network, as training with a smaller input dimension would speed up computation time, allowing the model to train faster.

3.2.4 Cross validation

We trained our models using 5 fold cross validation. We divided the data set into 5 random groups and trained our model 5 separate times, once for each combination of 4 groups as the training set with the 1 group not used to train as the validation set. This ensures that the model generalizes to new data it didn't train on and that it doesn't overfit, as if it does start to overfit, we would see the accuracy on the validation set decrease while the accuracy on the training set increases, and we could stop the training process there.

4 Implementation details

For our experiments we used cPickle in python to extract the training data from the data files into numpy arrays. Then we used the sklearn package in python for PCA and t-SNE. For our fully connected neural network we coded the training and cross validation algorithms in python without any libraries. For our convolutional neural network models, we used a python package called Keras, a high level neural network API backed by, in our case, TensorFlow, that allowed us to create convolutional neural networks.

5 Experiments

5.1 Cross validation

In each of our approaches, we used cross validation to ensure our model can generalize, to reduce overfitting, and to get a more certain measure of accuracy, as just one run could produce better or worse results due to luck. To do this, we shuffled our training data into a random order, then divided it into k folds, where each fold was 1/kth of the data. Then we would run our training algorithm n=k times with the nth fold serving as the validation set for that run, and the rest of the data as the training set. For this experiment, we decided to use 5 fold cross validation.

5.2 Fully connected neural network

Our training algorithm used stochastic gradient descent to find a local minimum, calculating the gradient for one training example at a time, then stepping the weights and biases in that direction. After trying multiple different network structures, we found that 128x96x64 neurons in the hidden layers using the reLU activation function led to the best validation accuracy.

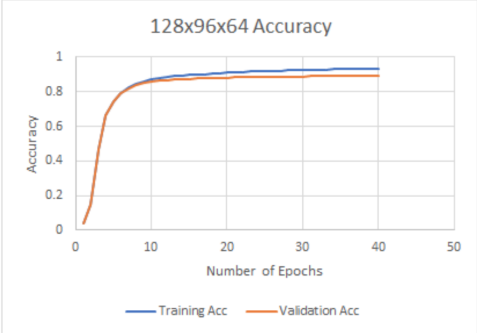


Figure 2: accuracy vs. epochs for 128x96x64 model

(# of neurons in h1)x(# of neurons in h2)x(# of neurons in h3)

model	128x96x64	128x64x32	64x48x32
sigmoid	88.24%	88.58%	85.99%
reLU	89.15%	87.46%	86.31%

Table 1: average validation accuracy over 5 folds for fully connected neural network

5.3 PCA

To test our results with PCA preprocessing of the data, we ran multiple trials with different principal numbers: 4, 8, 16, 32, 48, 64, 96, 128, 256, and 784. The input data will be reduced to these dimensions and then our neural net will be trained on the new, dimensionally reduced data. Below are the results from this experiment: the accuracy rate and the average computation time per epoch for each PCA reduced dataset.

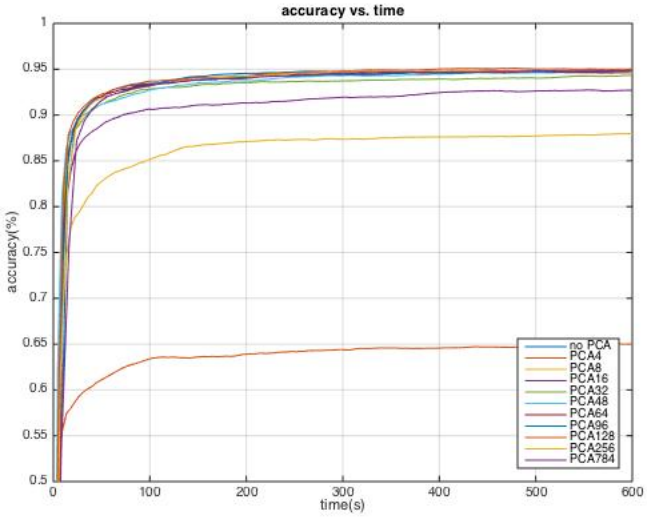


Figure 3: accuracy vs. time

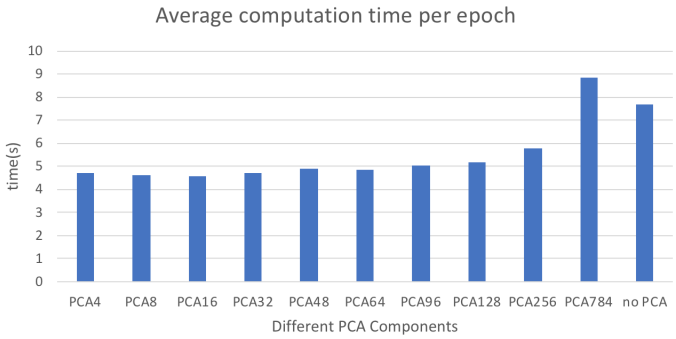
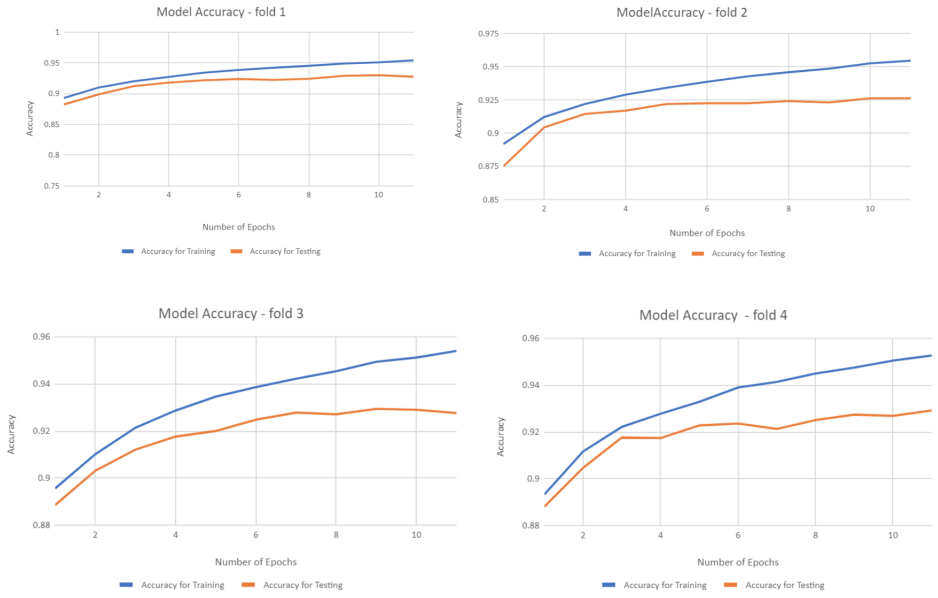


Figure 4: Average computation time per epoch

5.4 Convolutional Neural Network

Our model consisted of one convolutional layer that creates 32 5x5 feature maps using the reLU activation function. We also use max pooling with a pool size of 2x2. A dropout layer with a probability of 0.20 for each neuron to "drop out". We also used the reLU activation function inside of our fully connected layer, but used the softmax activation function in the output layer to give us a nice probability output.



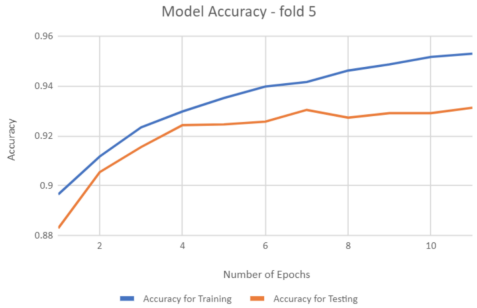


Figure 5: Accuracy vs number of epochs for convolutional neural network with 5 folds

Average training accuracy over 5 folds	Average validation accuracy over 5 folds
95.37%	92.84%

Table 2: training and validation accuracy

6 Analysis

6.1 Fully connected neural network analysis

Our model did not seem to overfit at all as evidenced by our validation accuracy continually increasing, and not dropping off even after many epochs. This is likely due to the large number of inputs relative to the number of hidden layer neurons. We had 784 inputs, and usually around 128 to 32 neurons in the hidden layers. As the number of parameters decreases, the complexity of the functions it is able to model decreases, and since we had many more input nodes, 784, the actual function is much more complex than the model is able to perfectly capture, and so the limited number of neurons in the hidden layers reduced the likelihood of our model overfitting the data, while still being able to capture the important patterns present in the data necessary to generalize to new examples.

6.2 PCA analysis

As seen from the results in the experiments section, PCA reduced the average computation time per epoch and had little negative effect on the accuracy of the model for larger principle numbers. It reduced the computation time because it lowered the dimension of inputs, making the matrix multiplication for back propagation faster. However, as the data was reduced to really small dimensions, too much information was lost to correctly classify many of the points and so the accuracy rate decreased.

6.3 CNN analysis

As seen in the above graphs, our model tended not to overfit. As the number of epochs increased and the training accuracy increased, the validation accuracy did start to level off, but did not decrease. This could be because of our regularization layer where we implemented dropout. By ?dropping out? each hidden neuron with a probability of 0.20 throughout training, the training data went through a different architecture with each pass [4]. This reduced the ability for the neurons to learn specific relationships between each other, reducing co-dependencies, and forcing them to learn more robust features of the image. Pooling also reduced overfitting by decreasing the dimensionality of the data early, forcing the model to look at more generalized features of the image.

7 Conclusion

In this paper, we presented three different models for our handwritten recognition problem - fully connected neural networks, fully connected neural network with PCA and convolution neural network. Our convolutional neural network outperformed our fully connected neural network significantly in

terms of accuracy. The validation accuracy for our convolutional neural network across 5 folds was 92.84% while the test accuracy for the best fully connected neural network model across 5 folds was 89.15%. Convolutional neural networks perform especially well on images because they form local connections between pixels in the pooling layer, and in images pixels are more closely related to other pixels in their locality than those far away. Our CNN has a high enough accuracy and generalization that it can be deemed an effective model for character recognition, and it could be used to identify any new characters by scaling an image into 28 by 28 pixels, then querying our model.

References

[1] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.

[2] Suruchi G Dedgaonkar, Anjali A Chandavale, and Ashok M Sapkal. Survey of methods for character recognition. *International Journal of Engineering and Innovative Technology (IJEIT)*, 1(5):180–189, 2012.

[3] John Loeber. K-means clustering on handwritten digits. `http://johnloeber.com/docs/kmeans.html`, 2015. [Online; accessed 25-July-2018].

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Meigarom Diego Fernandes. Does pca really improve classification outcome. `towardsdatascience.com/dimensionality-reduction-does-pca-really-improve-classification-outcome-6e9ba21f0a32`. [Online; accessed 25-July-2018].

[6] Patrice Y Simard, Dave Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *null*, page 958. IEEE, 2003.

[7] Vineet Singh and Sunil Pranit Lal. Digit recognition using single layer neural network with principal component analysis. In *Computer Science and Engineering (APWC on CSE), 2014 Asia-Pacific World Congress on*, pages 1–7. IEEE, 2014.

[8] Wikipedia contributors. Principal component analysis. `https://en.wikipedia.org/wiki/Principal_component_analysis`, 2018. [Online; accessed 25-July-2018].