Handwriting Recognition with Convolutional Neural Networks

Viswanath Pulle and Janakiram Sundaraneedi
University of Massachusetts, Lowell

Abstract—The purpose of this document portrays the application of machine learning algorithms to solving the problem of handwriting recognition. The model was developed in three stages starting with Multilayer perceptron algorithm as a baseline with a single hidden layer. It was then developed to a simple Convolutional Neural Network (CNN) and evolved into a larger CNN to achieve the best accuracy percentage. MNIST dataset was used for training and modelled by application of deep learning techniques making CNN with Keras as a backend. Results are reported using prediction error, which is nothing more than the inverted classification accuracy.
There are possible future improvisations of this task and are discussed in the end of the paper.

I. Introduction and Motivation

Motivated with the wide and interesting applications of handwriting recognition this paper will focus on how a CNN can help train a Machine to recognize handwritten characters in an accurate, efficient and faster way. Some of the various applications include Automatic Number plate recognition, postal zip code scanning, book scanning, check readers and many more.

II. Data Acquisition

The MNIST dataset is used for training the CNN. The MNIST dataset has a training set of 60,000 examples, and a test set of 10,000 examples. It is a digit recognition task with 10 digits (0 to 9) or 10 classes to predict.

The MNIST is a dataset developed by Yann LeCun, Corinna Cortes and Christopher Burges for evaluating machine learning models on the handwritten digit classification. The dataset was constructed from several scanned document datasets available from the National Institute of Standards and Technology (NIST).

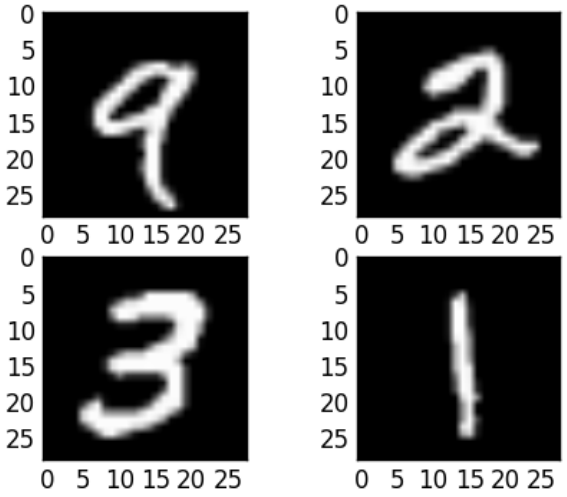


Figure 1: Excerpt from the MNIST dataset

III. Background

Multilayer Perceptron:

The field of artificial neural networks is often just called Neural Networks or Multilayer Perceptron's after perhaps the most useful type of neural network. A Perceptron is a single neuron model forming a baseline for larger neural networks. It is a field of study that investigates how simple models of biological brains can be used to solve difficult computational tasks like the predictive modeling tasks we see in machine learning. The goal is not creating realistic

models of the brain, but instead develop robust algorithms and data structures that we can use to model difficult problems.
The ability of the neural networks to learn the representation in the training data and how to best relate it to the output variable that we want to predict and build a mapping. Mathematically, they are capable of learning any mapping function and have been proven to be a universal approximation algorithm. The predictive capability of neural networks comes from the hierarchical or multilayered structure of the networks. The data structure can learn to represent features at different scales or resolutions and combine them into higher-order features. For example, from lines, to collections of lines to shapes.
The building block for neural networks are artificial neurons. These are simple computational units that have weighted input signals and produce an output signal using an activation function.

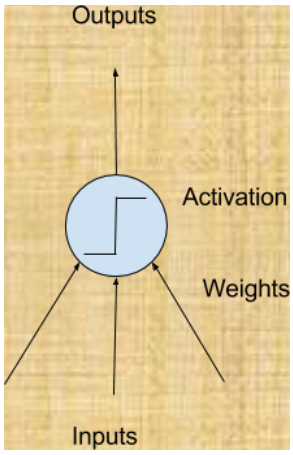


Figure 2: Model of a Simple Neuron

Stochastic Gradient Descent

The classical and still preferred training algorithm for neural networks is called stochastic gradient descent. This is where one row of data is exposed to the network at a time as input.
The network processes the input upward activating neurons as it goes to finally produce an output value. This is called a forward pass on the network. It is the type of pass that is also used after the network is trained to make predictions on new data.
The output of the network is compared to the expected output and an error is calculated.
This error is then propagated back through the network, one layer at a time, and the weights are updated according to the amount that they contributed to the error. This clever bit of math is called the Back-Propagation algorithm. The process is repeated for all the examples in your training data. One round of updating the network for the entire training dataset is called an epoch. A network may be trained for tens, hundreds or many thousands of epochs.
SGD approximates the true gradient of $E(w,b)$ by considering a single training example at a time

The algorithm iterates over the training examples and for each example updates the model parameters according to the update rule given by

$$\omega \longleftarrow \omega - \eta(\alpha \frac{\partial R(\omega)}{\partial \omega} + \frac{\partial L\left(\omega^T x_i + b, y_i\right)}{\partial \omega})$$

the learning rate is given by

$$\eta^t = \frac{1}{\alpha(t_0 + t)}$$

Prediction:

Once a neural network has been trained it can be used to make predictions. You can make predictions on test or validation data to estimate the skill of the model on unseen data.
You can also deploy it operationally and use it to make predictions continuously. The network topology and the final set of weights is all that you need to save from the model.

Predictions are made by providing the input to the network and performing a forward-pass allowing it to generate an output that you can use as a prediction.

Convolutional Neural Networks:

Convolutional Neural Networks are a powerful artificial neural network technique to get state-of-the-art results on difficult computer vision and natural language processing tasks. The main advantage with these networks is that they preserve the spatial structure of the problem and were developed for object recognition tasks such as handwritten digit recognition.

Layers of Convolutional Neural Network:

A CNN is a series of layers used repeatedly, lead to a formation of a Deep Neural Networks.
The main types of layers used to build a CNN are:

a) Input: This layer holds the raw pixel values of image
b) Convolutional Layer: This layer gets the results of the neuron layer that is connected to the input regions. We define the number of filters to be used in this layer. Each filter may be a 5x5 window that slider over the input data and gets the pixel with the maximum intensity as the output

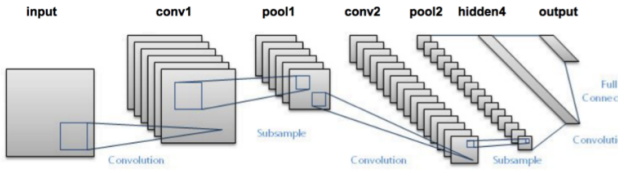


Figure 3: Convolutional Neural Network

c) Rectified Linear Unit [ReLU] Layer: This layer applies an element wise activation function on the image data. We know that a CNN uses back propagation. So, in order to retain the same values of the pixels and not being changed by the back propagation, we apply the ReLU function
d) Pooling Layer: This layer performs a down-sampling operation along the spatial dimensions (width, height), resulting in volume
e) Fully Connected Layer: This layer is used to compute the score classes i.e which class has the maximum score corresponding to the input digits

Keras:

Deep learning is supported by two of the top numerical platforms in Python, Theano and TensorFlow. Keras is a Python library that provides convenient way to create a range of deep learning models on top of Theano or TensorFlow. It is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow. It makes
developing deep learning models as fast and easy as possible for research and development. It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks.
Keras was developed and maintained to support Modularity, Minimalism, and Extensibility. It is a lightweight API and rather than providing an implementation of the required mathematical operations needed for deep learning and it provides a consistent interface to numerical libraries called backends like theanos and Tensorflow.

Data Preparation:

We must first prepare your data for training on a neural network. Data must be numerical, for example real values. If you have categorical data, such as a sex attribute with the values male and female, you can convert it to a real-valued representation called a one hot encoding. This is where one new column is added for each class value (two columns in

the case of sex of male and female) and a 0 or 1 is added for each row depending on the class value for that row.
This same one hot encoding can be used on the output variable in classification problems with more than one class. This would create a binary vector from a single column that would be easy to directly compare to the output of the neuron in the network's output layer, that as described above, would output one value for each class. Neural networks require the input to be scaled in a consistent way. You can rescale it to the range between 0 and 1 called normalization.
Another popular technique is to standardize it so that the distribution of each column has the mean of zero and the standard deviation of 1. Scaling also applies to image pixel data. Data such as words can be converted to integers, such as the frequency rank of the word in the dataset and other encoding techniques.

IV. Approach

We started to work on iris dataset to understand how to deal with classification problems. We learnt how to classify species of the iris flower taking the key attributes into account and predicting which species it belong to.

The species are classified into three types, in data pre-processing part we converted each species type to numerical value type called one Hot-encoding. Additionally, in a classification problem, if dataset of predicted output or any attribute is of the form yes or no, we need to Hot-encode it, so we fit into the machine learning algorithms. For Data Splitting, we divide the dataset into training(60%), testing(20%) and validation sets(20%). Other Measuring techniques such as accuracy levels and evaluating performance using K-cross validations.

Baseline Model with Multilayer Perceptron:

We will start with a multilayer perceptron algorithm as a baseline to compare and develop more complex convolutional neural network models. we will create a simple Multilayer Perceptron model that achieves an error rate of approximately 1.73%.

The training dataset is structured as a 3-dimensional array of instance, image width and image height. For a Multilayer Perceptron model we must reduce the images down into a vector of pixels.

In this case the 28x28 sized images will be 784 pixel input vectors. We can do this transform easily using the reshape() function on the NumPy array. The pixel values are integers, so we cast them to floating point values so that we can normalize them easily in the next step. The model is a simple neural network with one hidden layer with the same number of neurons as there are inputs (784). A rectifier activation function is used for the neurons in the hidden layer. A softmax activation function is used on the output layer to turn the outputs into probability-like values and allow one class of the 10 to be selected as the model's output prediction.
Logarithmic loss is used as the loss function (called categorical cross entropy in Keras) and the efficient ADAM gradient descent algorithm is used to learn the weights.

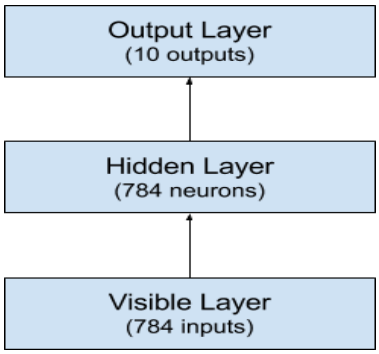


Figure 4: Summary of Multilayer Perceptron Network Structure

We proceed to fit and evaluate the model. The model is trained over 10 epochs with updates every 200 images. The test data is used as the validation dataset, allowing you to see the skill of the model as it trains. A verbose value of 2 is used to reduce the output to one line for each training epoch. Finally, the test dataset is used to evaluate the model and a classification error rate is printed. Running the model will take a few minutes when run on a CPU. This simple network in very few lines of code achieves a respectable error rate of 1.73% setting a good baseline.

Simple Convolutional Neural Network:

After having drawn our baseline we begin to develop a much complex CNN to get better performance than standard Multilayer Perceptrons. So, we will start by using a simple structure to begin with that uses all the elements for state-of-the-art results.

The Network architecture we will be using for our first CNN is summarized as follows:

a) The first hidden layer is a convolutional layer called a Convolution2D. The layer has 32 feature maps, which are scaled at the size of 5x5 and a rectifier activation function. This is the input layer, expecting images with the structure outline above
b) Pooling layer takes the maximum value called MaxPooling2D. It is configured with a pool size of 2x2
c) The next layer is a regularization layer using dropout is called Dropout layer. It is configured to randomly exclude 20% of neurons every time in the layer to reduce overfitting
d) Flatten layer converts the 2D matrix data to a vector called Flatten. It allows the output to be processed by standard fully connected layers
e) Fully connected layer relates to 128 neurons and a rectifier activation function is used
f) Final layer is the output layer that has 10 neurons for the 10 classes and a Softmax activation function to output probability-like predictions for each class.

As before, the model is trained using Logarithmic loss and the ADAM gradient descent algorithm.

We evaluate the model the same way as before with the Multilayer Perceptron. The CNN is fit over 10 epochs with a batch size of 200.

A depiction of the network structure is provided below

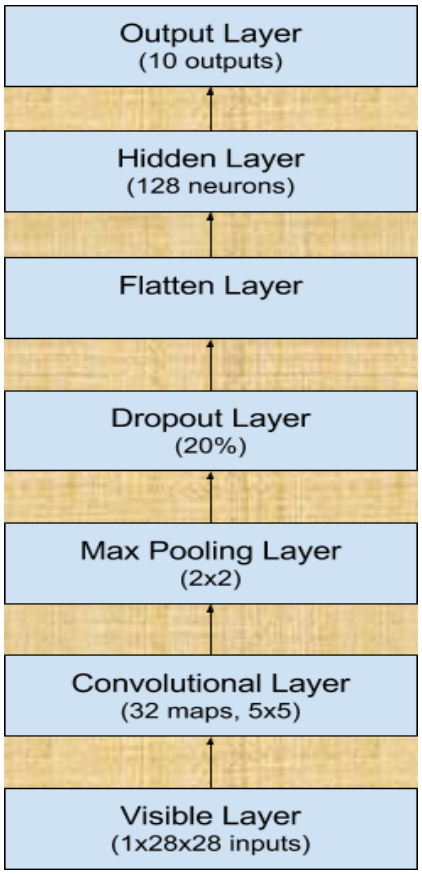


Figure 5: Summary of Simple Convolutional Neural Network Structure

Larger Convolutional Neural Network:

Our next step is to develop a better CNN to get higher accuracy than before. We start off with importing the classes and functions then load and prepare the data the same as in the previous CNN. This time we deal with a larger CNN architecture with additional convolutional, max pooling layers and fully connected layers.
The network topology can be summarized as follows:

a) Convolutional layer with 30 feature maps of size 5x5
b) Pooling layer down sampling the max over 2x2 window
c) Convolutional layer with 15 feature maps of size 3x3
d) Pooling layer down sampling the max over 2x2 window
e) Dropout layer with a probability of 20%
f) Flatten layer
g) Fully connected layer with 128 neurons and rectifier activation
h) Fully connected layer with 50 neurons and rectifier activation
i) Output layer to connect all the outputs and show the probability percentages

The Large CNN is also fit over 10 epochs with a batch size of 200. This model achieves a highly respectable classification error rate of 0.76%.

The architecture of the larger network layer is shown below:

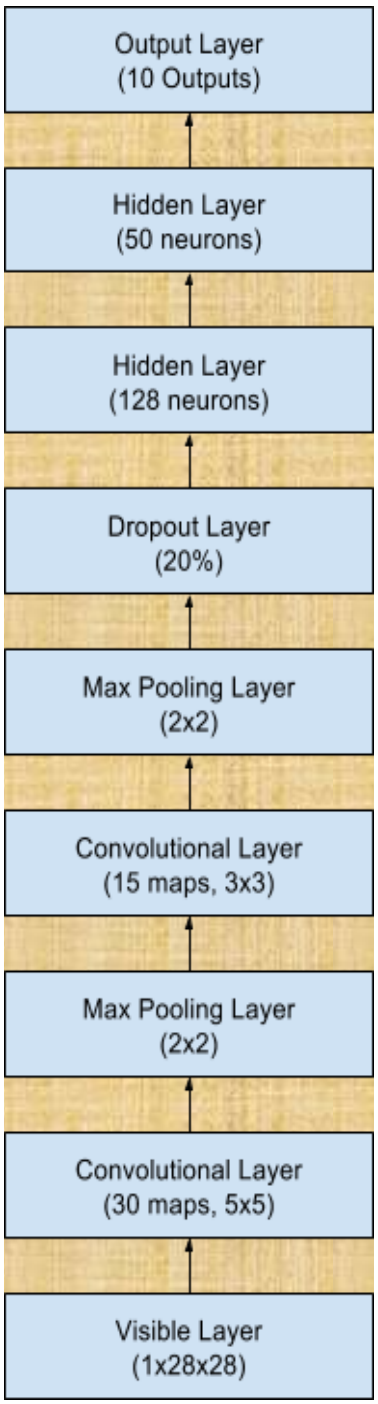


Figure 6: Summary of the Larger Convolutional Neural Network Structure.

V. Results

We ran the models to the specification mentioned above to get the following results

Model	Multilayer Perceptron	Simple CNN	Large CNN
Epochs	10	10	10
Time to run	9 secs	129 secs	155 secs
Processor	CPU	CPU	CPU
Error / Accuracy Percent	1.92%	0.95%	0.72%

Multilayer Perceptron Model:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
8s - loss: 0.2790 - acc: 0.9211 - val_loss: 0.1412 - val_acc: 0.9573
Epoch 2/10
8s - loss: 0.1120 - acc: 0.9677 - val_loss: 0.0922 - val_acc: 0.9712
Epoch 3/10
8s - loss: 0.0720 - acc: 0.9797 - val_loss: 0.0785 - val_acc: 0.9770
Epoch 4/10
8s - loss: 0.0505 - acc: 0.9859 - val_loss: 0.0744 - val_acc: 0.9770
Epoch 5/10
8s - loss: 0.0376 - acc: 0.9892 - val_loss: 0.0677 - val_acc: 0.9788
Epoch 6/10
10s - loss: 0.0270 - acc: 0.9927 - val_loss: 0.0642 - val_acc: 0.9797
Epoch 7/10
10s - loss: 0.0211 - acc: 0.9946 - val_loss: 0.0626 - val_acc: 0.9800
Epoch 8/10
10s - loss: 0.0143 - acc: 0.9968 - val_loss: 0.0642 - val_acc: 0.9797
Epoch 9/10
10s - loss: 0.0109 - acc: 0.9978 - val_loss: 0.0600 - val_acc: 0.9807
Epoch 10/10
10s - loss: 0.0077 - acc: 0.9986 - val_loss: 0.0592 - val_acc: 0.9808
Neural Network Baseline Error: 1.92%
```

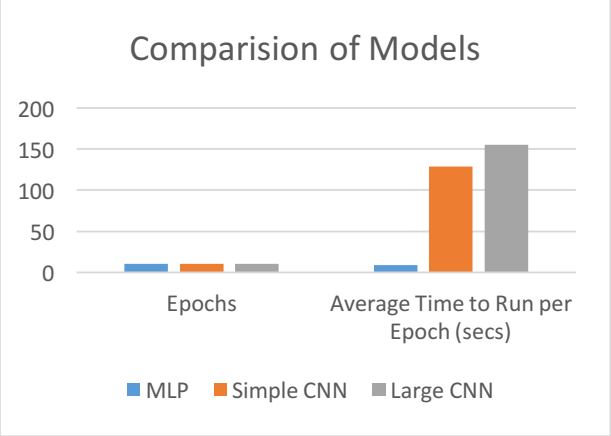
Simple CNN Model:

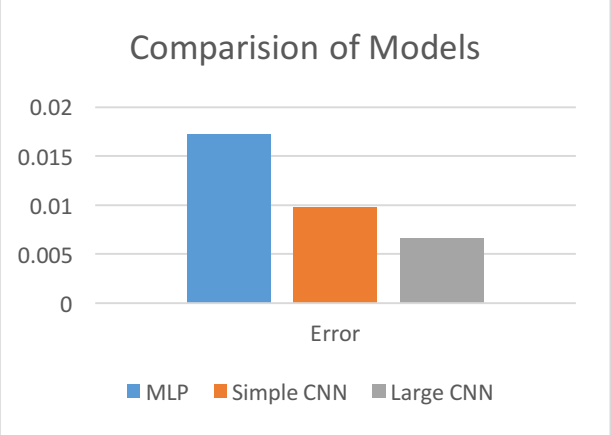
```
Train on 60000 samples, validate on 10000 samples
146s - loss: 0.2334 - acc: 0.9339 - val_loss: 0.0817 - val_acc: 0.9740
Epoch 2/10
129s - loss: 0.0736 - acc: 0.9782 - val_loss: 0.0464 - val_acc: 0.9844
Epoch 3/10
132s - loss: 0.0533 - acc: 0.9839 - val_loss: 0.0428 - val_acc: 0.9861
Epoch 4/10
130s - loss: 0.0404 - acc: 0.9875 - val_loss: 0.0399 - val_acc: 0.9867
Epoch 5/10
129s - loss: 0.0336 - acc: 0.9892 - val_loss: 0.0344 - val_acc: 0.9886
Epoch 6/10
262s - loss: 0.0277 - acc: 0.9913 - val_loss: 0.0319 - val_acc: 0.9892
Epoch 7/10
129s - loss: 0.0235 - acc: 0.9926 - val_loss: 0.0365 - val_acc: 0.9883
Epoch 8/10
130s - loss: 0.0203 - acc: 0.9937 - val_loss: 0.0316 - val_acc: 0.9897
Epoch 9/10
128s - loss: 0.0165 - acc: 0.9946 - val_loss: 0.0293 - val_acc: 0.9898
Epoch 10/10
128s - loss: 0.0144 - acc: 0.9957 - val_loss: 0.0318 - val_acc: 0.9905
Convolutional  Error: 0.95%
```

Large CNN Model:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
161s - loss: 0.3922 - acc: 0.8793 - val_loss: 0.0949 - val_acc: 0.9692
Epoch 2/10
149s - loss: 0.0933 - acc: 0.9714 - val_loss: 0.0595 - val_acc: 0.9804
Epoch 3/10
149s - loss: 0.0690 - acc: 0.9784 - val_loss: 0.0381 - val_acc: 0.9881
Epoch 4/10
148s - loss: 0.0558 - acc: 0.9826 - val_loss: 0.0320 - val_acc: 0.9887
Epoch 5/10
1612s - loss: 0.0476 - acc: 0.9851 - val_loss: 0.0298 - val_acc: 0.9904
Epoch 6/10
185s - loss: 0.0432 - acc: 0.9862 - val_loss: 0.0291 - val_acc: 0.9913
Epoch 7/10
167s - loss: 0.0380 - acc: 0.9876 - val_loss: 0.0287 - val_acc: 0.9899
Epoch 8/10
155s - loss: 0.0344 - acc: 0.9890 - val_loss: 0.0245 - val_acc: 0.9913
Epoch 9/10
142s - loss: 0.0326 - acc: 0.9899 - val_loss: 0.0217 - val_acc: 0.9935
Epoch 10/10
142s - loss: 0.0278 - acc: 0.9909 - val_loss: 0.0225 - val_acc: 0.9924
Large Convolutional Neural Netwoek Error: 0.76%
```

Comparison:





VI. Future Work

Many Applications are now presently working on neural to improve there performance and prediction. For Instance, Google Autonomous car, Amazon Echo, Robotics, stock prediction's, diseases prediction, climatic conduction's and sentiment analysis etc. Expanding this work we need to work Recurrent Neural Network's which deals with different input types to feed the neural network and classify them.

VII. Conclusion

This paper focused on building a Complex Large Convolutional Neural Network by starting with multilayer perceptron model as a baseline. We improved the error percent from 1.92% to 0.72%. we built a simple CNN as an intermediate design step. Later we developed into a complex CNN adding more layers than before to improve the predictability and reduce the error percentage. Future works have been proposed.

VIII. References

- The MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/
- Žiga Zadnik, Handwritten character Recognition: Training a Simple NN for classification using MATLAB)
- Sci-Kit Learn
- Andrew Ng cs229 course materials and Course Era
- Shaohan Xu, Qi Wu, and Siyuan Zhang, Application of Neural Network In Handwriting Recognition
- J Pradeep, E Srinivasan, and S Himavathi, Diagonal Based Feature Extraction For Handwritten Character Recognition System Using Neural Network
- Xinyi Jiang, Huy Pham, Qingyang Xu, Handwritten Digit Recognition via Unsupervised Learning
- http://neuralnetworksanddeeplearning.com/chap6.html