Hand Digit Recognition

Abdulmajeed Kabir[1]

[1]Systems Engineering, King Fahd University of Petroleum and Minerals, 31260, Dhahran, Saudi Arabia

January 7, 2018

Abstract

In this work, the automatic recognition of handwritten digits was treated using a classical and modern machine learning approach; both based on neural networks. First, we present a simple softmax regression model and achieved a recognition accuracy of 91.91%. Then, we implement a convolutional neural network model with a custom KNet architecture and achieved an accuracy of 99.13%. In this work, we make use of make use of the MNIST Dataset which comprises of cleaned train, test, and validation categories of hand-written digits as grayscale images.

1.0 Introduction

Automatic recognition of hand written digits involves the use of image processing, computer vision, and, or, artificial intelligence translation from a digit or string of digits written by hand, to a set of characters accurately represented in memory by a computer without explicit human guidance. This has numerous applications, for example, automatic handling of cheque notes in the banking industry, translation of old handwritten texts to digital forms, handwriting input modes in smartphones, fraud detection, and so on. In this work, we present two methods: (1) A softmax regression (SR) model, and, (2) A custom KNet convolutional neural network (CNN) model.

1.1 Handwritten Digits Dataset (MNIST)

The MNIST database was built by modifying the original NIST database containing digits 0 to 9 as mentioned earlier. It has 60,000 training images (the mnist tensorflow train database has 55,000 examples however), 10,000 test images, and 5,000 validation images all drawn from the same distribution. All these images are in grayscale and normalized. The center of gravity of the intensity lies at the center of the image. Each image is 28 x 28 pixels in dimension thus each image has a total of 784 intensity values between 0 and 1 when flattened. When flattened, it can become a 784 x 1 or a 1 x 784 vector. Unfortunately, during flattening (or reshaping) the relationship of pixels with neighboring pixels is lost. Image pixels are highly correlated with their neighbors and losing this information is detrimental. The solution to this can be found in convolutional neural networks which would be discussed in Section 4.0. The major categories of machine learning techniques used in solving the recognition tasks with MNIST dataset include: Linear classifiers, k-nearest neighbors, boosted stumps, nonlinear classifiers, support vector machines(SVMs), neural nets, convolutional nets. Using tensorflow for our machine learning task, we utilize "One-Hot" encoding for the Classification Labels.

2.0 Literature Survey

The automatic recognition of hand written digits can be safely regarded as a classical machine learning problem. Numerous methods, architechures, tricks, and techniques, have been proposed and good results have been achieved. Evidence of this can be found in the prevalence of different digit datasets used for benchmarking and analyzing such algorithms. Common classical datasets in this work include USPS (US Postal Service), NIST (US National Institute of Standards and Technology) dataset, and its variants such as the MNIST (modified NIST) and EMNIST (Extended MNIST) [1] datasets. The CIFAR-10 and CIFAR-100 [2] dataset, STL-10 dataset [3], Street View House Numbers (SVHN) dataset [4] and CEDAR dataset are commonly sought too.

One of the early works utilizing a relatively modern technique on this subject was [5]. In [6] however, the authors studied the performance of different classifier algorithms on the MNIST

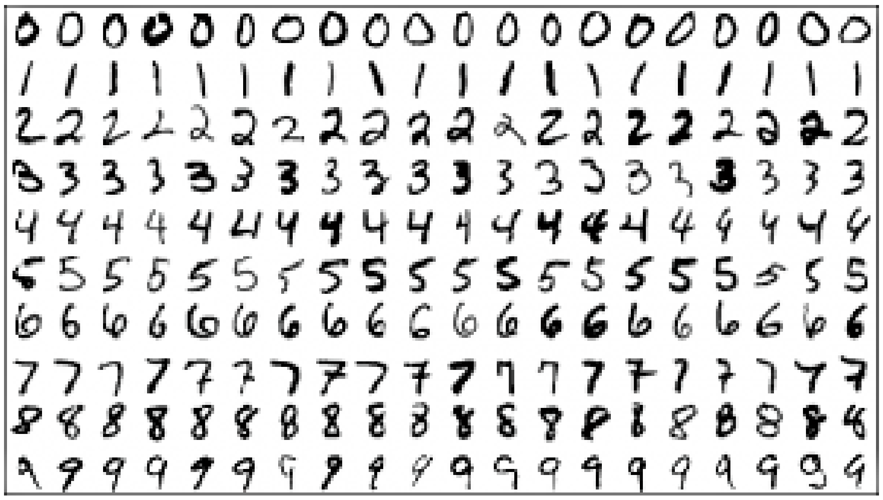

Figure 1: An Overview of the MNIST Dataset

database of handwritten digits. They discussed measures that affect algorithmic implementation such as training time, run time, and memory requirements. In their work, they presented a Baseline Linear Classifier, Nearest Neighbor Classifier, Large Fully Connected Multi-Layer Neural Network, LeNet1, LeNet4, Boosted LeNet4 (based on the idea of Convolutional Networks), Tangent Distance Classifier (TDC), LeNet4 with K-NN, Local Learning with LeNet4, Optimal Margin Classifier (OMC). From these, the Boosted LeNet4 achieved the best error performance while LeNet4 required the least memory. LeCun et al.'s paper proposed a ConvNet approach to digit recognition problem, achieving 96% accuracy. The method involved considerable preprocessing without which accuracy falls to 60%.

Similarly, among the early works on Handwritten digit recognition was [7]. The authors developed a backpropagation network constrained for digit recognition on zipcode digits provided by USPS. The aimed to show that a large BP network can be applied to real image recognition tasts without extensive preprocessing. The method produced a 1% error rate and about a 9% reject rate. Work by Yawei Hou and Huailin Zhao utilized an Improved BP Neural Network for Handwritten Digit Recognition. The authors claim results obtained converged faster and the classification results were more accurate compared to results at that time [8]. In [9], the authors presented that Feedforward neural networks utilizing Extreme Learning machine algorithm had faster weight optimization, however, required larger number of hidden units to provide comparable results with a Backpropaogation based algorithm. In [10], the authors presented a method for recognizing handwritten digits by fitting generative spline models which would then be tuned by an Expectation Maximization Algorithm. While the method has it advntages, the main advantage is higher computational requirements compared to standard OCR techniques. Work by [11] on the other hand involved the combination of classifiers for digit recognition. This work was based on the idea that either by Bayesian combination, Dempster-Shafer evidential reasoning, and Dynamic classifier selection, the independent decisions by two high performance nearest-neighbor hand-printed digit classifiers can be combined to obtain improved digit classification systems.

Loo-Nin Teow and Kia-Fock Loe in [14], presented a method based on biological vision to solve automatic recognition of handwritten digits. They extracted linearly separable features from MNIST dataset and used a linear discriminant system for recognition, with the triowise linear support vector machines with soft voting yielding the best results. It doesn't end there however. In 1999, [15] proposed contour information and Fourier descriptors for digit recognition. Models were built based on contour features, then test digits were analyzed by comparing the test digit's features with built models. The recognition rate achieved as around 99.04%. In [16], a three-stage classifier was developed comprising of 2 Neural Networks and one Support Vector Machine (SVM). The two Neural Networks in tandem help to provide low misclassification rate, more complex features, and, a well-balanced rejection criterion. The SVM was optimized to take the top classes ranked by the Neural network. The authors claimed their work to achieve competitive results at the time. In 2003, Cheng-Lin Liu et al [12] summarized the performance of then state-of-the-art feature extraction and classifier techniques on three image databases: CEDAR, MNIST, CENPARMI. In total, 10 feature vectors and 8 classifiers were combined to give 80 accuracies to the test data sets used. Results obtained can be found in [12]. Similar work by the same author(s) evaluated normalization methods and direction feature extraction techniques with existing methods useful in digit recognition [13].

Numerous works, tricks, approaches, techniques, and systems can also be found on this subject. For instance: The use of Self organizing maps [17], Shape matching [18], [19] A method involving the division of the image into grids and computing the Hu moments as features was proposed. Artificial neural network was then implemented as classifier. The method yielded good processing times and accuracy. Methods such as Restricted Boltzmann Machines (RBMs) [28], SVM with inverse fringe feature [29], Echo state networks [30], Discrete Cosine S-Transform (DCST) features with Artificial Neural Networks classifier [31], Neural Dynamics Classification algorithm [32], Bat Algorithm-Optimized SVM [33] have been applied. Similarly, promising results from numerous algorithms have prompted the extension to numerous languages and characters. Indian numerals were treated in [23], Persian digits in [24], Bangla Digits [25], Hindu and Arabic digits in [26], Sindhi Numerals [27].

Most recently, due to the advent of powerful computational systems such as GPUs and TPUs, more solutions have been proposed, especially, with Deep learning. In [21] for instance, the authors made a case for Online digit recognition using deep learning. They developed a software application to record a dataset which included user information such as age, sex, nationality, and handedness. Thereafter they presented a 1D and 2D ConvNet model which obtained results of 95.86% (using distance and angle), and 98.50% respectively. Unfortunately, as deep learning methods have yielded exceptional results, they have also empowered Adversarial systems. It was shown by [22] that the changing of 1 pixel can lead to significant misclassification rates. The authors showed that 70.97% of the natural images can be perturbed to at least one target class simply by modifying a single pixel with 97.47% confidence on average. Further information can be found from academic resources.

3.0 Neural Network (Softmax Regression)

In this section, we describe the first approach to solving the handwritten digit classification problem. Here, we present the Softmax Regression.

The 28 x 28 image matrix is flattened to a column vector of length 784. Each entry in the 784 rows represents a part of the training image. It is noteworthy that by flattening the input image matrix, spatial relationship information between pixels is lost. This issue is addressed by our KNet CNN model. Invariance to rotation and scaling can be dealt with by a more recent technology known as Capsule networks. In Section 3, we'll discuss the CNN model employed. Capsule networks, on the other hand, are beyond the scope of this study.

The Arabic numerals consists of ten digits: 0,1,2,3,4,5,6,7,8,9. Each image is unpacked to a column vector $x_j$and multiplied by weights $W_{i,j}$ with biases b_i. This results into a tensor of `logits` defined by Equation 1.0.

$z_i = \sum_j W_{i,j}x_j + b_i$

Solving a recognition problem on this system requires techniques from multi-class classification. This is where the `softmax` function comes in (Equation 2.0 below). Basically, the softmax regression (also known as the multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes. It replaces the common sigmoid activation function used in perceptrons

$\sigma(z_j) = \frac{\exp(z_j)}{\sum_j \exp(z_j)}$

The Softmax function takes an array of numbers and returns a set of numbers in the range 0 and 1. These set of numbers add up to one and are used in determining the probailisitc appropriateness of a learning outcome. Figure 2.0 below shows the softmax regression graphically.

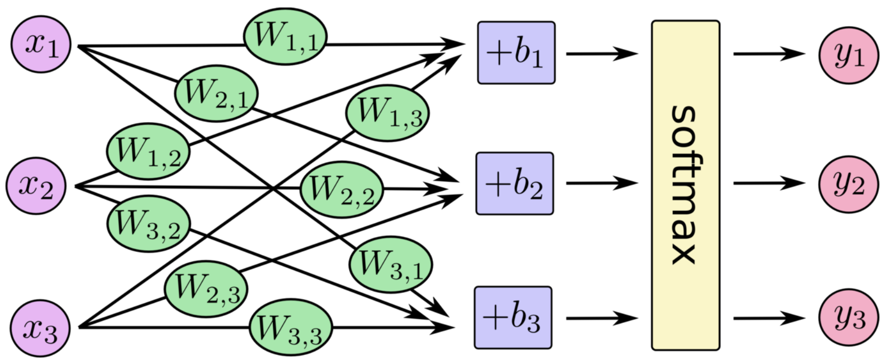

Figure 2: Graph of Softmax Regression

This can be collapsed into the matrix form shown in Figure 3.0

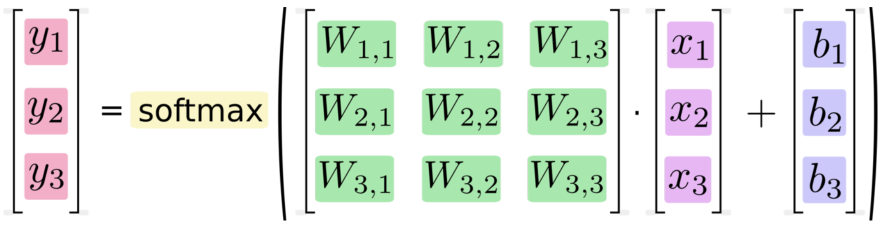
y_1
y_2
y_3
= softmax
$W_{1,1}$ $W_{1,2}$ $W_{1,3}$
$W_{2,1}$ $W_{2,2}$ $W_{2,3}$
$W_{3,1}$ $W_{3,2}$ $W_{3,3}$
x_1
x_2
x_3
$+$
b_1
b_2
b_3

Figure 3: Tensor Representation of Softmax Regression

$D(S, L) = -\sum_j L_j \log(\sigma(z_j))$

Then the softmax cross-entropy (Equation 3.0 above) is defined and gradient descent optimizer with a learning rate of 0.5 is used to perform training. 1000 epochs were used with a batch size of 100. The weights of the network were initialized as zeros.

```
#MNIST Digit Classification
#(c) Abdulmajeed Muhammad Kabir, December 2017
#Libraries: Tensorflow Matplotlib
#Dataset: MNIST Dataset
#Softmax Regression on MNIST Dataset

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt
%matplotlib inline

mnist = input_data.read_data_sets("MNIST_data/",one_hot=True)

#single_image = mnist.train.images[1].reshape(28,28)
#plt.imshow(single_image,cmap='gist_gray')

x = tf.placeholder(tf.float32, shape=[None,784])
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x,W) + b

y_true = tf.placeholder(tf.float32, [None,10])
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_true, logits=y
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.5)
train = optimizer.minimize(cross_entropy)

init = tf.global_variables_initializer()

with tf.Session() as sess:

    sess.run(init)

    for step in range(1000):
        batch_x, batch_y = mnist.train.next_batch(100)

        sess.run(train, feed_dict={x:batch_x, y_true:batch_y})

    #Evaluate the Model
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_true,1))

    acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    print(sess.run(acc, feed_dict = {x:mnist.test.images, y_true:mnist.test.labels}))
```

4.0 Convolutional Neural Network

CNNs are a much better approach to solving image classification problems. CNNs are based on the visual cortex of mammals and therefore find their origins in biological research just like simple perceptrons. It is based on the realization that neurons in the visual cortex have a small local receptive field which can overlap to create a large robust visual system. This inspired an ANN architecture that later became the CNN. Famously implemented by Yann LeCun et al (See Section 2.0). The LeNet-5 architecture was first used to classify the MNIST dataset. Other famous CNN architectures include AlexNet by Alex Krizhevsjy, GoogLeNet by Szegedy, and ResNet by Kaiming He et. al. Compared to Densely Connected Neural Networks, the CNN uses lesser parameters and thus scales well over larger input data and is easier to deploy as applications. The CNN's architecture gives its strength. There are basically two main activities: (1) feature extraction via convolutions, and (2)classification. Input data (Images) are represented as tensors of dimensions: H-Height of image in pixels, W-Width of image in pixels, Color channel dimensions - 1 for Gray color and 3 for RGB color. Figure 4 below is an outlook of a CNN model.

The KNet CNN architecture to be discussed is built using the following elements: (1) an input layer, (2) a convolutional layer, (3) Nonlinearity function, (4) Pooling layer, (5) Fully connected layer, (6) Drop out layer, (7) Softmax layer. There are two layers in this model. The layers contain the following elements.

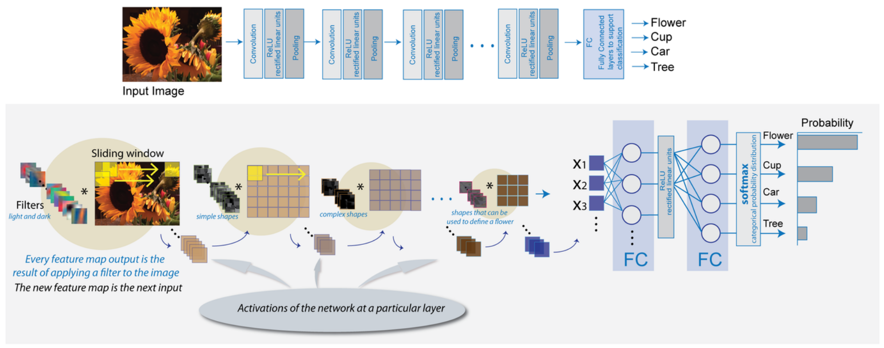
Input Image
Convolution
ReLU rectified linear units
Pooling
FC Fully Connected layers to support classification
Flower
Cup
Car
Tree
Sliding window
Filters
light and dark
simple shapes
complex shapes
shapes that can be used to define a flower
Every feature map output is the result of applying a filter to the image
The new feature map is the next input
Activations of the network at a particular layer
X1
X2
X3
FC
FC
softmax
categorical probability distribution
Probability

Figure 4: Schematic of a Convolutional Neural Network

The input layer takes in a 28 x 28 image as a matrix with pixels normalized between 0 and 1.

The convolution layer is abbreviated as `Conv`. In this layer, kernels on the input image are learnt. Its description includes three parts: number of channels, kernel spatial extent (kernel size), padding and stride size. In KNet, a kernel size (receptive field) of 5x5 with a stride of 1 pixel. Zero padding ('SAME' padding) is used. At the first layer, it computes 32 features for each 5x5 patch. At the second layer, it computes 64 features for each 5x5 patch.

The `ReLU` stands for rectified linear unit and serves to introduce nonlinearity in the model. It is the activation function between the layers of the CNN (after convolution operation). The ReLU can be seen in Figure 3.0.

The pooling layer is abbreviated as `Pool`. Pooling is a subsampling procedure that helps to reduce the dimensionality of each feature map and at the same time, the number of parameters required for the model. It removes a lot of information from the image yet helps agains overfitting. Max pooling (largest element from a window) is used, the kernel size is 2x2 and the stride is set as 2x2 in KNet. The pooling layers will reduce the input size by a factor 2 pixels. We make use of zero padding here ('SAME').

The fully connected layer is abbreviated as `FC`". Is more or less a traditional multilayer perceptron and uses a softmax function at the output. Every neuron in the previous layer is connected to every neuron in the next layer. It uses the learned features to classify the input image into various classes based on the training dataset.

The dropout layer is abbreviated as `Drop`". Dropout is a technique to improve the generalization of deep learning methods. It sets the weights connected to a certain percentage of nodes in the network to 0. In KNet, the drop out percentage is set to 50% in the dropout layer).

Softmax is abbreviated as $\sigma(z_j)$.

Finally, KNet is trained with the MNIST Dataset, which is a hand digit recognition problem with 10 classes. The last fully connected layer outputs a length 10 vector for every input image and the softmax layer converts this length 10 vector into the estimated posterior probability for the 10 classes (i.e. digits 0 through 9).

Then the softmax cross entropy is defined and Adam optimizer is used to perform training with a Learning rate of 0.001. Epochs of 5000 were used with a batch size of 50. The weights of the kernels were initialized randomly from a trucated normal distribution with mean 0 and standard deviation 0.1.

```
#MNIST Digit Classification
#(c) Abdulmajeed Muhammad Kabir, December 2017
#Libraries: Tensorflow Matplotlib
#Dataset: MNIST Dataset
#Convolutional Neural Network on MNIST Dataset

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

def init_weights(shape):
    init_random_dist = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(init_random_dist)

def init_bias(shape):
    init_bias_vals = tf.constant(0.1, shape=shape)
    return tf.Variable(init_bias_vals)

def conv2d(x,W):
    #x --> [batch, H, W, Channels]
    #W --> [filter H, filter W, Channels IN, Channels OUT]
    return tf.nn.conv2d(x,W,strides=[1,1,1,1], padding='SAME')

def max_pool_2by2(x):
    #x --> [batch, h, w, c]
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

def convolutional_layer(input_x, shape):
    W = init_weights(shape)
    b = init_bias([shape[3]])
    return tf.nn.relu(conv2d(input_x,W)+b)

def normal_full_layer(input_layer, size):
    input_size = int(input_layer.get_shape()[1])
    W = init_weights([input_size, size])
    b = init_bias([size])
    return tf.matmul(input_layer, W) + b

x = tf.placeholder(tf.float32, shape=[None, 784])
x_image = tf.reshape(x,[-1,28,28,1])

y_true = tf.placeholder(tf.float32, shape=[None,10])

# CONVOLUTIONAL LAYER 1
convo_1 = convolutional_layer(x_image, shape=[5,5,1,32])
convo_1_pooling = max_pool_2by2(convo_1)

# CONVOLUTIONAL LAYER 2
convo_2 = convolutional_layer(convo_1_pooling, shape=[5,5,32,64])
convo_2_pooling = max_pool_2by2(convo_2)

convo_2_flat = tf.reshape(convo_2_pooling, [-1,7*7*64])
full_layer_one = tf.nn.relu(normal_full_layer(convo_2_flat,1024))

hold_prob = tf.placeholder(tf.float32)
full_one_dropout = tf.nn.dropout(full_layer_one, keep_prob=hold_prob)

#PREDICTOR
y_pred = normal_full_layer(full_one_dropout, 10)

# LOSS FUNCTION
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_true,logits=y_

# OPTIMIZER
optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
train = optimizer.minimize(cross_entropy)

#INITIALIZE
init = tf.global_variables_initializer()

steps = 5000

with tf.Session() as sess:

    sess.run(init)

    for i in range(steps):

        batch_x, batch_y = mnist.train.next_batch(50)

        sess.run(train, feed_dict={x:batch_x, y_true:batch_y, hold_prob:0.5})
        if i%100 == 0:
            print("ON STEP: {}".format(i))
            print("ACCURACY: ")
            matches = tf.equal(tf.argmax(y_pred,1), tf.argmax(y_true,1))
            acc = tf.reduce_mean(tf.cast(matches, tf.float32))
            print(sess.run(acc,feed_dict={x:mnist.test.images, y_true:mnist.test.labels, hold_
            print('\n')
```

5.0 Results

The following results were obtained:

5.1 Neural Network (Softmax Regression) Results

Tables 1 and 2 show parameters used and model performance

Neural Network Softmax Regression (Settings 1)

Parameter	Value
Optimizer	Gradient Descent on Cross-Entropy
Learning Rate	0.5
Batch Size	100
Initialized Weights	Zeros
Initialized Biases	Zeros
Epochs	1000
Training Time	0.5hrs
Accuracy	0.92

Table 1: Neural Network Softmax Regression Parameters Type 1 and Results

5.2 Convolutional Neural Network Results

Tables 3 and 4 show parameters used and model performance

Neural Network Softmax Regression (Settings 2)

Parameter	Value
Optimizer	Gradient Descent on Cross-Entropy
Learning Rate	0.1
Batch Size	100
Initialized Weights	Ones
Initialized Biases	Ones
Epochs	1000
Training Time	0.5hrs
Accuracy	0.91

Table 2: Neural Network Softmax Regression Parameters Type 2and Results

Convolutional Neural Network (KNet Settings 1)

Parameter	Value
Optimizer	Adam optimizer on Cross-Entropy
Initialized Weights	Random Normal std.dev=0.1
Initialized Biases	0.1
Padding	Zero Padding
Pooling	2x2 Stride=2
Batch Size	50
Learning Rate	0.0
Epochs	5000
Dropout	0.5
Training Time	3hrs
Accuracy	0.99

Table 3: Convolutional Neural Network Parameters Type 1 and Results

6.0 Conclusion

In conclusion, one might be tempted to asking the question: “was the computational costs really worth it since the results of the neural network softmax regression model were close to that of the convolutional neural network model?”. In this case, it is worthy to note two things: (1) A 1% difference in accuracy is a big deal in the field of machine learning and there are numerous research groups trying their best to squeeze out as little as a 0.5% improvement in machine learning models. (2) The testing of these models is based on properly cleaned, and normalized datasets. In the real-world, this is not the case. The CNN does well at learning complex features and abstracting higher levels of information from input data - this has been proven in literature. Finally, regardless of the final accuracy of either model, the CNN has better potential in computer vision and image recognition tasks.

Convolutional Neural Network (KNet Settings 2)

Parameter	Value
Optimizer	Adam optimizer on Cross-Entropy
Initialized Weights	Random Normal std.dev=0.5
Initialized Biases	0.1
Padding	Zero Padding
Pooling	2x2 Stride=1
Batch Size	50
Learning Rate	0.0
Epochs	5000
Dropout	0.5
Training Time	3hrs
Accuracy	0.99

Table 4: Convolutional Neural Network Parameters Type 2 and Results

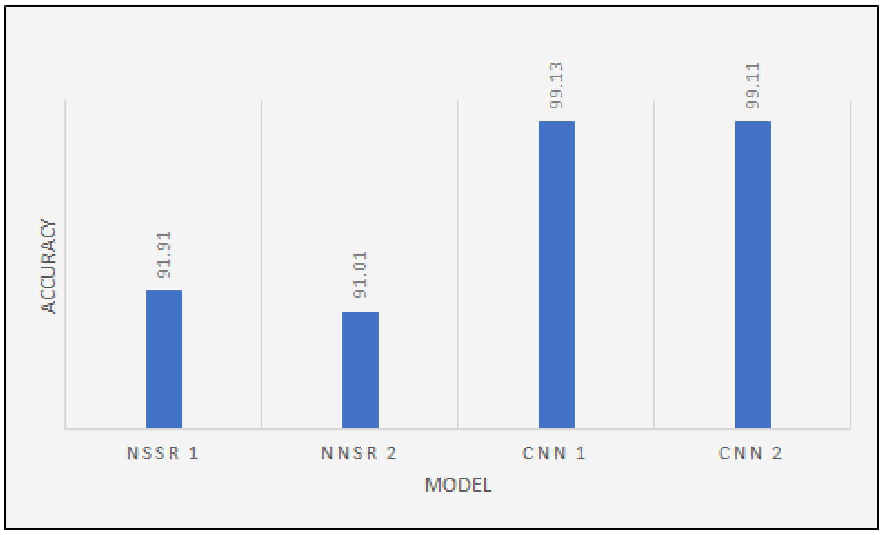

Figure 5: Summary of Results for the four models

References

[1] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373.*

[2] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[3] Coates, A., Ng, A., & Lee, H. (2011, June). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 215-223).

[4] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011, December). Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning* (Vol. 2011, No. 2, p. 5).

[5]

[6] LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., ... & Vapnik, V. (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, *261*, 276.

[7] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems* (pp. 396-404).

[8] Hou, Y., & Zhao, H. (2017, October). Handwritten Digit Recognition Based on Improved BP Neural Network. In *Chinese Intelligent Systems Conference* (pp. 63-70). Springer, Singapore.

[9] De Chazal, P., Tapson, J., & van Schaik, A. (2015, April). A comparison of extreme learning machines and back-propagation trained feed-forward networks processing the mnist database. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on* (pp. 2165-2168). IEEE.

[10] Revow, M., Williams, C. K., & Hinton, G. E. (1996). Using generative models for handwritten digit recognition. *IEEE transactions on pattern analysis and machine intelligence*, *18*(6), 592-606.

[11] Sabourin, M., Mitiche, A., Thomas, D., & Nagy, G. (1993, October). Classifier combination for hand-printed digit recognition. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on* (pp. 163-166). IEEE.

[12] Liu, C. L., Nakashima, K., Sako, H., & Fujisawa, H. (2003). Handwritten digit recognition: benchmarking of state-of-the-art techniques. *Pattern recognition*, *36*(10), 2271-2285.

[13] Liu, C. L., Nakashima, K., Sako, H., & Fujisawa, H. (2004). Handwritten digit recognition: investigation of normalization and feature extraction techniques. *Pattern Recognition*, *37*(2), 265-279.

[14] Teow, L. N., & Loe, K. F. (2002). Robust vision-based features and classification schemes for off-line handwritten digit recognition. *Pattern Recognition*, *35*(11), 2355-2364.

[15] Jeong, C. S., & Jeong, D. S. (1999, December). Hand-written digit recognition using Fourier descriptors and contour information. In *TENCON 99. Proceedings of the IEEE Region 10 Conference* (Vol. 2, pp. 1283-1286). IEEE.

[16] Gorgevik, D., & Cakmakov, D. (2004, August). An efficient three-stage classifier for handwritten digit recognition. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on* (Vol. 4, pp. 507-510). IEEE.

[17] Zhang, B., Fu, M., Yan, H., & Jabri, M. A. (1999). Handwritten digit recognition by adaptive-subspace self-organizing map (ASSOM). *IEEE transactions on neural networks*, *10*(4), 939-945.

[18] Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, *24*(4), 509-522.

[19] Leem, S., & Kim, S. (2016). High Speed Digit Recognition for Serial Numbers of Home Appliances.

[20-21] Corr, P. J., Silvestre, G. C., & Bleakley, C. J. (2017). Open Source Dataset and Deep Learning Models for Online Digit Gesture Recognition on Touchscreens. *arXiv preprint arXiv:1709.06871.*

[22] Su, J., Vargas, D. V., & Kouichi, S. (2017). One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864.*

[23] Mathur, G., & Rikhari, S. (2017). A Review on Recognition of Indian Handwritten Numerals.

[24] Parseh, M. J., & Meftahi, M. (2017). A New Combined Feature Extraction Method for Persian Handwritten Digit Recognition. *International Journal of Image and Graphics*, *17*(02), 1750012.

[25] Alom, M. Z., Sidike, P., Taha, T. M., & Asari, V. K. (2017). Handwritten Bangla Digit Recognition Using Deep Learning. *arXiv preprint arXiv:1705.02680.*

[26] El Hindi, K., Khayyat, M., & Abu Kar, A. (2017). Comparing the machine ability to recognize hand-written Hindu and Arabic digits. *Intelligent Automation & Soft Computing*, *23*(2), 295-301.

[27] Chandio, A. A., Jalbani, A. H., Laghari, M., & Awan, S. A. (2017). Multi-Digit Handwritten Sindhi Numerals Recognition using SOM Neural Network. *Mehran University Research Journal of Engineering & Technology, 36*(4), 8.

[28] Toulgaridis, N., Bougioukou, E., & Antonakopoulos, T. (2017, May). Architecture and implementation of a Restricted Boltzmann Machine for handwritten digits recognition. In *Modern Circuits and Systems Technologies (MOCAST), 2017 6th International Conference on* (pp. 1-4). IEEE.

[29] Patel, A., & Kalyani, T. V. (2016, February). Support Vector Machine with Inverse Fringe as Feature for MNIST Dataset. In *Advanced Computing (IACC), 2016 IEEE 6th International Conference on* (pp. 123-126). IEEE.

[30] Schaetti, N., Salomon, M., & Couturier, R. Echo State Networks-based Reservoir Computing for MNIST Handwritten Digits Recognition.

[31] Mohapatra, R. K., Majhi, B., & Jena, S. K. (2015, December). Classification performance analysis of MNIST Dataset utilizing a Multi-resolution Technique. In *Computing, Communication and Security (ICCCS), 2015 International Conference on* (pp. 1-5). IEEE.

[32] Rafiei, M. H., & Adeli, H. (2017). A new neural dynamic classification algorithm. *IEEE transactions on neural networks and learning systems*, *28*(12), 3074-3083.

[33] Tuba, E., Tuba, M., Simian, D., & Street, I. R. (2016). Handwritten digit recognition by support vector machine optimized by bat algorithm. In *24th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision,(WSCG 2016)* (pp. 369-376).

[34] Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, *29*(6), 141-142.