Capstone Project Report

Examine and assess the business value of the MNIST Database of Handwritten Digits

CSC-591 Algorithms for Data Guided Business Intelligence

North Carolina State University

Aravind Anantha 200154605 aananth3@ncsu.edu

Ruthvik Mandava 200132414 rmandav@ncsu.edu

Raghavendra Prasad Potluri 200132312 rpotlur@ncsu.edu

***Abstract*—** There are several approaches that were implemented to classify handwritten images and recognize digits. As a part of capstone project, we tried to learn the way each of these works and made sure the report covers all the preprocessing required, parameters that are to be set and pulled out the accuracies. Firstly we built a series of base classifiers such as convolutional neural networks, random forests, support vector machines, k-nearest neighbors and tabulated the results. A naive implementation of Radial Basis Function Neural networks with initial K Means clusters fed and logarithmic derivatives used as activation functions is developed and run with small dataset available which achieved 90% accuracy on test set. Understanding the importance of a lot of factors that needs to be added to the data such as elastic distortions, stroke direction, and affine deformations we started to use the existing packages adding different pre-processing methods before classifying. Even seemingly simple additions to existing classifiers made significant changes in the results. This report covers many of the topics that were taught in the course applied to the digit recognition problem. All codes were developed in python and many classifiers were imported from sklearn

Keywords- Digit recognition, Rbfnn, ANN, SVM, KMeans, KNN, CNN

I. Introduction

Automatic handwriting recognition has been of great academic and commercial interest for decades now. Current implementations are able to solve this problem with an accuracy of 98%. Some of the most popular approaches to solving this problem are Multi Layered Perceptrons, Support Vector Machines and Convolutional Neural Networks. We were able to use all of these classifiers with right pre-processing options added such as normalization, stroke and elastic deformations to get an accuracy of 99.3%. The base classifier Rbfnn which was implemented from scratch achieved 90% accuracy.

One should start solving this problem of digit recognition to get an idea of how, many deep learning techniques work and make themselves acquainted with a lot of efficient packages available. There are lot of applications specifically to this problem such as

a) Data entry from documents: Some documents or forms require manual written dates and information which needs to be recognised and be stored in database.

b) Applications of this include scanning cheques for deposit in a mobile banking application, scanning information from passports etc.

c) Optical Character Recognition: Quick conversion of printed documents into PC docs

d) To assist Pen Computing (an initiative to convert written documents into real time word docs)

e) Number extraction from number plate (image)

II. Related work

In our research, we came across multiple effective algorithms to classify digits including convolutional neural networks, support vector classifiers, and k nearest neighbors classifiers, random forest classifiers, and naive bayes classifiers. As we further explored through more research and code implementation, we observed several strengths and weaknesses of each of these approaches.

Random Forests

One of the first methods we were interested in at the beginning of the project was random forests. Random forests, in general, are a popular choice to implement recommendation systems because of their efficiency, interpretability, and flexibility in handling various attributes as input. We quickly realized random forests are very efficient to build and the accuracy achieved was good. Random forests provide a good balance between build efficiency and accuracy achieved. We wanted to implement a few more classifiers to increase the prediction accuracy and see how the models fare.

Radial Basis Function Neural Networks

There are various ways a single digit is written by humans and therefore hundreds of instances that needs to be classified to the same label or digit. RBFNN is known to accommodate minute variations in the data dispersed across many instances. It is a three layer network comprised of input layer, hidden layer and output layer. Based on the number of input features, number of neurons in the input layer are fixed. The output neurons specify the labels. Hidden or middle layer is where the activation or kernel function is calculated. Many papers suggested to feed to the input layer, the clusters formed from the K Means to make it time efficient. There are many ways an activation function could be implemented such as Least Mean square, logarithmic functions, exponential functions etc. The most efficient implementations could get to the accuracy of 97.3%.

Convolutional Neural Networks

They are same as any other neural networks trained with a back propagation algorithm, but differ in the architecture which involves a five layer network designed specifically to recognize patterns from digitized or pixel images which led to achieving 98% accuracy with minimal preprocessing. We will demonstrate the additional efforts put in to make CNN output the predictions that achieve 99.4% accuracy in the methodology section. The architecture at the input layer takes in the pixel data of handwritten image and the output layer consists of 10 neurons that represents each digit where one of them needs to be +1 for correct digit and all others must be -1 after classification.

Support Vector Machines (SVM)

Linear SVM tries to find out the hyperplane that classifiers the training data into two class with maximum possible margin. Each image with 784 pixels (28 * 28) is considered as a point and the hyperplane divides the data points into two classes. But digit recognition is a multi class problem as we have 10 digits. Therefore the sklearn package forms ten linear support vector machines, each of which classifies a training instance as either a '0' or not a '0', turning into binary classifier. The model classifies an instance based on the highest confidence score given by all the ten svm's.

III. Methodology

Convolutional Neural Networks

We started off by implementing already available packages in python (sklearn and keras) to see what kind of accuracies we are getting. One of the most widely used approach in Handwritten Digit Recognition is Convolutional Neural Networks. We implemented convolutional neural networks with the following layers. There are two convolutional layers. Both the layers have a kernel size of 3. The general strategy of a convolutional neural network is to extract simple features at a high resolution, and then to convert them into more complex features at a coarser resolution. The simplest method to generate coarse feature is by sub-sampling a layer by a factor of 2. The simplest of this implementation is to have only a one unit overlap. This is achieved by using a kernel size of 3. We used the simplest method for convolutional neural networks since they are computationally very intensive compared to the other methods. These two layers can be thought of as a trainable feature extractor. Since these layers work as feature extractors, hence we pass the entire dataset to the algorithm directly. The next two layers are fully connected layers which form the classification layers. By changing the number of hidden units we can control the capacity and generalization of the overall model. For MNIST (10 classes), the optimal capacity was reached with around 100 layers [4] and for our implementation we used 128 hidden layers in the first fully connected layer. The second fully connected layer is the output layer and has 10 hidden units corresponding to the digits 0 to 9.

Support Vector Machines

There are a lot of implementations of support vector machines. By comparing previous research done on SVM for digit recognition we felt that SVM's with and RBF kernel provided high accuracies. We decided to use this SVM as one of our models to compare performance since it's performance comes close to the performance of convolutional neural networks. There is an implementation of SVM with RBF kernel in sklearn. We implemented SVM without modifying most of the parameters since the in-built model using the RBF kernel gives high accuracy.

K-Nearest Neighbors

We used the KNeighborsClassifier from sklearn for our implementation of K-NN. Going through previous work done on the same topic by [6], the best model obtained was a tie between number of neighbors set to 3 and 5. So, in our experiments we tried fitting models with 3, 4, and 5 neighbors. The best results were obtained when number of neighbors is 4. The accuracy shown for K-NN is for the model fitted with 4 neighbors.

Random Forest Classifier

We decided to add Random Forests into our experiments because the models are easy to build and have a low complexity and running time compared to other models. In general we can say that by increasing the number of trees in the forest the results get better. But, this is true only upto a certain point after which prediction performance will be lower than the cost of computing the additional trees. On the other hand if the number of trees is very small compared to the number of observations then some observations will be predicted very few times or not at all. This seriously affects the predictive power of random forests. So the main focus is to find the optimal number of trees to get the best output. Referring to [7] we went ahead with 250 trees in the forest, since the best results were obtained somewhere between 100 and 300. We fitted models with 100, 150, 200, 250 and 300 trees respectively and found that choosing 250 trees gives the best output.

Elastic distortions

The main focus of our project was to try out different, novel preprocessing approaches with the intent of increasing accuracy of the predictions. One such preprocessing method was adding elastic distortions to the images. Elastic distortions are transformations applied to the original dataset to increase the size. There are other methods for distortion as well, but on research about previous work done on these distortions we found that elastic distortions give best results [4]. The process of elastic distortion is as follows:

1. First a random displacement field between -1 and 1 is generated with a uniform distribution for the pixels of the image.
2. The displacement fields are convolved with a Gaussian of standard deviation (sigma). Points to keep in mind here are that if sigma is very large then the random displacement fields average to zero, and if sigma is very small then the field looks completely random after normalization.
3. The displacement fields are then multiplied by a scaling factor (alpha). Alpha controls the intensity of the deformation.

To implement the above method, we need to find optimal values for sigma and alpha so that the distorted fields look like elastic distortions. Research on optimal alpha and sigma was done by [5] and the optimal values were found to be 36 and 8 for alpha and sigma respectively.

Radial Basis Function Neural Networks

We also implemented our own version of a Radial Basis Function Neural Network. Details of the implementation can be found in the next section.

IV. Experiments and Results

A. Experimental setup

i) PoC for K Means clustering to predict genres
ii) Collaborative filtering in Python with Spark's Alternating Least Squares implementation (from Spark MLlib)
iii) kNN implementation in R to predict ratings to a user

B. Dataset used

For our own implementation of basic RBFNN, we used simple dataset from sklearn datasets load digits() where each image corresponds to 8 * 8 pixel image. After we gained proper insight into the problem, we shifted to the MNIST digit dataset which is originally provided by [2]. The same data can be found in kaggle. We used the training dataset in kaggle[3] for our experiments. The data consists of 60,000 labelled images in csv format. Each row contains a label which corresponds to what digit is in the image and the next columns contain the pixel values of the image. Each image is 28x28 pixels. We split the dataset into training and test to check the accuracy of our predictions. 80% of the data is used for training and the rest is for testing.

C. Implementation details

Convolutional Neural Network in Tensorflow (Keras)

1. Loading the dataset
2. Reshaping the image into a 28x28 format using pandas
3. Creating an optimal model using available models and layers in Keras
    a. A 2d convolutional layer with 32 filters and a kernel with a 3x3 window
    b. A second 2d convolutional layer with 64 filters and a 3x3 kernel
    c. A Max Pooling layer
    d. A Dropout layer with a dropout rate of 0.25
    e. A Flatten layer to implement a Dense layer next
    f. A Dense layer, a densely-connected neural network with 128 units
    g. Another Dropout layer with rate 0.5
    h. A final Dense layer with 10 units corresponding to the 10 labels which we want for our output
4. Fitting the above model on the entire train dataset
5. Calculating accuracy and confusion matrix on the test dataset

The code can be found here [9]

K-NN Classifier in sklearn

1. Loading the dataset
2. Fitting the K-NN classifier on the train data with number of neighbors 4. Using 4 neighbors gave the best results
3. Calculating accuracy and confusion matrix on the test dataset

SVC Classifier in sklearn

1. Loading the dataset
2. Fitting the support vector classifier with an rbf kernel and using balanced class weights
3. Calculating accuracy and confusion matrix on the test dataset

Random Forest Classifier in sklearn

1. Loading the dataset
2. Fitting the random forest classifier with 250 trees and balanced class weights
3. Calculating accuracy and confusion matrix on the test dataset

The code for KNN, SVC and Random Forest can be found here [10]

All the above methods were implemented on the entire dataset as train data with no preprocessing and with PCA for dimensionality reduction. Best results were obtained without dimensionality reduction.

Elastic distortions

1. Loading the dataset
2. Reshaping the image into a 28x28 format using pandas
3. Implementing the elastic distortions function mentioned in the previous section
4. Applying all the classifiers, CNN, SVM, KNN, Random Forest to the new dataset with increased training data obtained from elastic distortions

The code can be found here [11]

Rbfnn Naive Implementation

Preprocessing done

1. Load the small dataset load_digits() from sklearn.data_sets
2. Normalized the data to bring to [0,1]
3. Divide the dataset into training set and test set
4. K Means clustering is performed with 20 centers to decrease the time complexity

Implementation

1. Initial weights between hidden layer and output layer are initialized randomly
2. We now iterate epoch times, selecting each instance randomly and then back propagating the weights based on the logistic derivative formula available from numpy
    a) Forward propagation from first layer to hidden layer and store the weights
    b) Forward propagation from hidden layer to output layer and store the weights
    c) Now the initial random weights are changed accordingly by back propagation from output layer to hidden layer
3. After the iterations, we calculate the output using forward pass

The implementation is very naive and a lot of preprocessing steps and other ensemble methods can be added to achieve higher accuracy which are available in machine learning libraries.

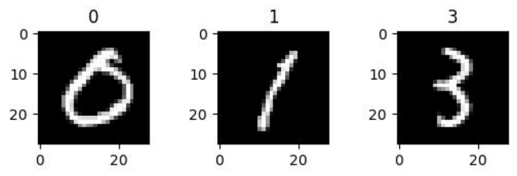
The code can be found here [12]

D. Results

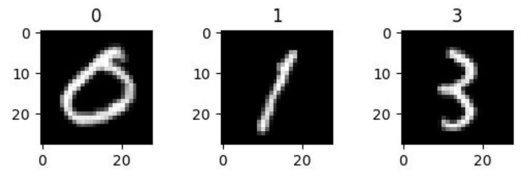
1. *Elastic Distortions*

The below two figures show some sample digits before and after elastic distortions.

Before distortion



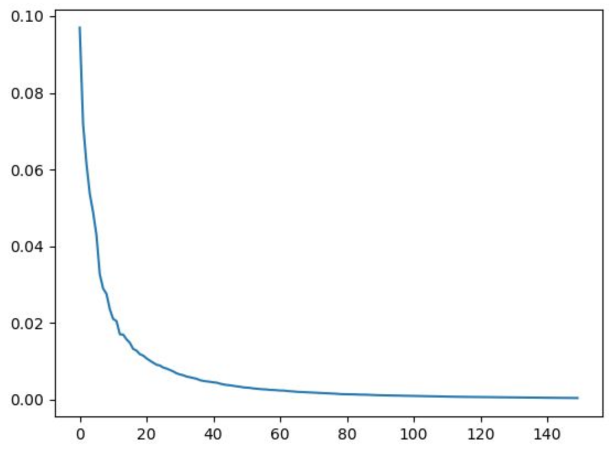
After distortion



Clearer images can be found in the github link.

2. *PCA*

We tried implementing some of the classification models using PCA to compare the accuracy achieved when run with the entire feature set. Below is the graph showing the asymptote of the principal components. From the graph we can see that there is a sharp dip in the explained variance at around 20 components. For our analysis with PCA we took the first 40 components.



3. *Accuracies of the various classifiers*

Classifier	Without preprocessing	With elastic distortions

K-NN	0.9672	0.9692
CNN	**0.9876**	**0.9915**
SVM	0.9555	0.9646
Random Forest	0.9669	0.97333

V. Conclusions and future directions

We have gained an insight into how a classifiers work using Deep Learning and the importance of tuning the parameters to get optimal results. We got to understand the basics of Deep Learning and how Tensorflow works, especially the packages available in Keras. By running experiments on various classifiers which build models of different complexities, we got a first-hand understanding about the problem of getting a better accuracy vs building a simple model. Here, Random Forests gave a good accuracy and difference between the best accuracy we achieved and the accuracy from Random Forests is comparable. But, Random Forests are much easier to implement and it only takes a fraction of the time needed to fit a convolutional neural network. We have to choose between a better accuracy and a simpler model depending on our requirements.

Our future aim is to try out other preprocessing steps which could further increase the accuracy. A lot of work has already been done in building optimal models, but trying out different preprocessing techniques on the existing models could significantly improve accuracy.

References

[1] http://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/

[2] http://yann.lecun.com/exdb/mnist/

[3] https://www.kaggle.com/c/digit-recognizer/data

[4] Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. By Patrice Y. Simard, Dave Steinkraus, John C. Platt

[5] Handwritten Digit Recognition with a Committee of Deep Neural Nets on GPUs. By Dan C Ciresan, Ueli Meier, Luca M. Gambardella and Jurgen Schmidhuber.

[6] Handwritten digit recognition: Benchmarking of state-of-the-art techniques by Cheng-Lin Liu, Kazuki Nakashima, Hiroshi Sako, Hiromichi Fujisawa.

[7] Using Random Forests for Handwritten Digit Recognition. By Simon Bernard, Laurent Heutte, Sebastien Adam.

[8] Recognition of Unconstrained Handwritten Numerals by a Radial Basis Function Neural Network Classifier Hwang, Young-Sup and Bang, Sung-Yang.

[9] https://github.com/kira0992/Handwritten-Digit-Recognition---BI-Capstone/blob/master/mnist_cnn.py

[10] https://github.com/kira0992/Handwritten-Digit-Recognition---BI-Capstone/blob/master/mnist_knn-svc.py

[11] https://github.com/kira0992/Handwritten-Digit-Recognition---BI-Capstone/blob/master/mnist_elastic_deformations.py

[12] https://github.com/kira0992/Handwritten-Digit-Recognition---BI-Capstone/blob/master/rbfnn.py