Handwritten Digit Recognition Using Random Forest Classifier and Convolutional Neural Network

Chintan Tikekar(N03476581) and Pratikkumar Gohil(N03497336) - Professor : Min Chen

***Abstract*—In the pattern recognition field, interest has been grown in recent years for multiple classifier systems. These methods aim at inducing an ensemble of classifiers by producing diversity at different levels. Following this principle, Our work aims at studying classifiers to recognize a digit in a strictly pragmatic approach and make it to the distributed level. For that purpose we have experimented the Random Forest algorithm and Convolutional Neural Network considered as the prime reference algorithm for the MNIST handwritten digits database. In this paper, we describe random forest classifier and convolutional neural network (CNN) classifier and review methods and how to build it in parallel or distributed system. Next we present our experimental results. We finally draw some conclusions on random forest vs CNN global behavior according to their time of execution and accuracy tuning.**

INTRODUCTION

A. Project Motivation

Despite the abundance of technological writing tools, many people still choose to take their notes traditionally: with pen and paper. However, there are drawbacks to handwriting text. Its difficult to store and access physical documents in an efficient manner, search through them efficiently and to share them with others. Thus, a lot of important knowledge gets lost or does not get reviewed because of the fact that documents never get transferred to digital format. This time our aim to develop classifier that recognize digits from images.

B. Aims and Objectives

Machine Learning issues are concerned by several learning approaches aiming at building high performance classification systems, with respect to a set of data. One of them, arousing growing interest in recent years, deals with classifiers to build Classifier Systems that can classify digits or characters. So as part our project we aim to create a classifier that can classify digits.

BACKGROUND/HISTORY OF THE STUDY

Handwriting recognition of characters has been around since the 1980s. The task of handwritten digit recognition, using a classifier has great importance and use such as recognize zip codes on mail for postal mail sorting, processing bank check amounts, numeric entries in forms filled up by hand(i.e. Tax forms) and so on. There are different challenges faced while attempting to solve this problem. The handwritten digits are not always of the same size, thickness or orientation and position relative to the margins. Our goal was to implement a pattern classification method to recognize the handwritten digits provided in the MNIST data set of images of handwritten digits (0-9).

The general problem we predicted we would face in this digit classification problem was the similarity between the digits like 1 and 7, 5 and 6, 3 and 8 etc. Moreover people also write the same digit in many different ways. Finally the uniqueness and variety in the handwriting of different individuals also influences the formation and appearance of the digit.

Our aim is to classify digit and there are many algorithms for classify digit but to classify handwritten digits by images there are several algorithms.

Data Set : In our project we have used MNIST data which is downloaded from kaggle.com There are two data files train.csv and test.csv which contain gray-scale images of hand-drawn digits, from zero through nine. A Test data set contains 28000 handwritten images. Train.csv contains 42000 different rows each raw has 785 columns. Each raw represent a handwritten image of 28*28 pixels, a total of 784 pixel per image. The first column called label, is the digit that was drawn by the user. The rest of the columns contain the pixel value of the associated image. Each pixel has single pixel-value associated with it, indicating lightness or darkness or that pixel, with higher number means darker. This pixel-value is an integer between 0 and 255 inclusive.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero). For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below. Visually, if we omit the ”pixel” prefix, the pixels make up the image like this:

000 001 002 003 ... 026 027
028 029 030 031 ... 054 055
056 057 058 059 ... 082 083
— — — — ... — —
728 729 730 731 ... 754 755
756 757 758 759 ... 782 783

Random Forest : Random forest algorithm is an ensemble classification algorithm. Ensemble classifier means a group of classifiers. Instead of using only one classifier to predict the target, In ensemble, we use multiple classifiers to predict the target. In case, of random forest, these ensemble classifiers are the randomly created decision trees. Each decision tree is a single classifier and the target prediction is based on the majority voting method.

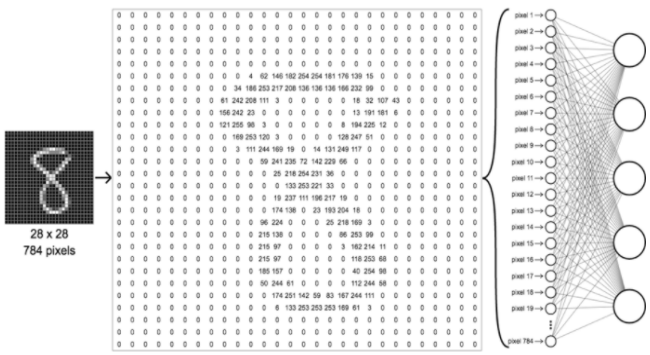

Fig. 1. Data format representation of a row

Convulutional Neural Network : In machine learning, a convolutional neural network is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. Convolutional networks were inspired by biological processes in which the connectivity pattern between neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

Approach and Implementation

I. Random Forests

Actually, Random Forest is a general term for classifier combination that uses L tree-structured classifiers h(x, k), k = 1,...L where k are independent identically distributed random vectors and x is an input. With respect to this definition, one can say that Random Forest is a family of methods in which we can find several algorithms, such as the Forest-RI algorithm proposed by Breiman in , and cited as the reference method in all RF related papers. In Forest-RI algorithm, Bagging is used in tandem with a random feature selection principle. The training stage of this method consists in building multiple trees, each one trained on a bootstrap sample of the original training set.

For N instances in the training set, sample N cases at random with replacement. The resulting set will be the training set of the tree.

For M input features, a number K ¡¡ M is specified such that at each node, a subset of K features is drawn at random, and among which the best split is selected.

Each tree is grown to its maximum size and unpruned

In this Random forest algorithm we are using sklearn library. pyplot is a feel like Math lab library for viewing images. matplotlib.pyplot.gray() set the default colormap to gray and apply to current image if any. In our train.csv file, we have a column for digit number(0-9) and other 784 columns for pixel value. And for accuracy or cross validation purpose We will split out data set to 80% training data(x_train, y_train) and 20% testing data(x_test, y_test) and see how it goes

x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state=4)

Now we have a train dataset and a test dataset so after that we created a object of random forest classifier and where n_estimators=100. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. N_estimators means the number of trees in the forest. Then we train our model by train dataset.

rf=RandomForestClassifier(n_estimators=100)

rf.fit(x_train,y_train)

To check accuracy we use test dataset and predict it.

pred=rf.predict(x_test)

We created a method which calculate number true of prediction from Pred. Thus we got accuracy. Now we have 2 options to use this model to classify digits or there are two methods to pass input in our model to classify digit.

1) Get data from csv format and pass it.

2) Scan an image file

We already did first approach and for second approach we scan an image and passed it our method for to do multiple operation on image.

Like convert it into 28*28 pixel, grayscale and mask image. And then convert it into a proper input for model and then pass it into model. In Both case we got 92% accuracy.

And to run it parallel we used MapReduced framework. see figure **??**.

II. Convolutional Neural Network

We will implement a simple Convolutional Neural Network in TensorFlow which has a classification accuracy of about 99%, or more. We are using TensorFlow, Sklearn, matplotlib libraries.

Convolutional Networks work by moving small filters across the input image. This means the filters are re-used for recognizing patterns throughout the entire input image. This makes the Convolutional Networks much more powerful than Fully-Connected networks with the same number of variables. This in turn makes the Convolutional Networks faster to train.

As shown in Figure 2 ,The input image is processed in the first convolutional layer using the filter-weights. This results in 16 new images, one for each filter in the convolutional layer. The images are also down-sampled so the image resolution is decreased from 28x28 to 14x14. These 16 smaller images are then processed in the second convolutional layer. We need filter-weights for each of these 16 channels, and we need filter-weights for each output channel of this layer. There are 36 output channels so there are a total of 16 x 36 = 576 filters in the second convolutional layer. The resulting images are down-sampled again to 7x7 pixels. The output of the second convolutional layer is 36 images of 7x7 pixels each. These are then flattened to a single vector of length 7 x 7 x 36 = 1764, which is used as the input to a fully-connected layer with 128 neurons (or elements). This feeds into another fully-connected layer with 10 neurons, one for each of the classes, which is

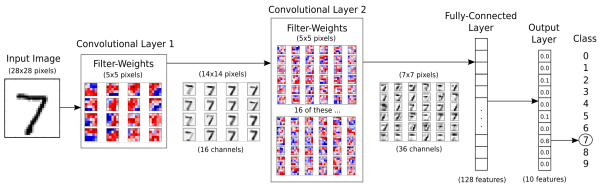

Fig. 2. CNN Implementation

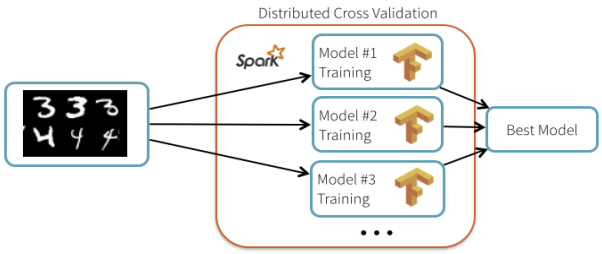

Fig. 3. TensorflowOnSpark

used to determine the class of the image, that is, which number is depicted in the image.

The convolutional filters are initially chosen at random, so the classification is done randomly. The error between the predicted and true class of the input image is measured as the so-called cross-entropy. The optimizer then automatically propagates this error back through the Convolutional Network using the chain-rule of differentiation and updates the filter-weights so as to improve the classification error. This is done iteratively thousands of times until the classification error is sufficiently low. These particular filter-weights and intermediate images are the results of one optimization run and may look different if you re-run this Notebook. Note that the computation in TensorFlow is actually done on a batch of images instead of a single image, which makes the computation more efficient. This means the flowchart actually has one more data-dimension when implemented in TensorFlow.

We can see more clearly how the filter is being moved to different positions of the image. For each position of the filter, the dot-product is being calculated between the filter and the image pixels under the filter, which results in a single pixel in the output image. So moving the filter across the entire input image results in a new image being generated.

Furthermore, the output of the convolution may be passed through a so-called Rectified Linear Unit (ReLU), which merely ensures that the output is positive because negative values are set to zero. The output may also be down-sampled by so-called max-pooling, which considers small windows of 2x2 pixels and only keeps the largest of those pixels. This halves the resolution of the input image e.g. from 28x28 to 14x14 pixels.

In the second convolutional layer is more complicated because it takes 16 input channels. We want a separate filter for each input channel, so we need 16 filters instead of just one. Furthermore, we want 36 output channels from the second convolutional layer, so in total we need 16 x 36 = 576 filters for the second convolutional layer. It can be a bit challenging to understand how this works.

The input is the flattened layer from the previous convolution. The number of neurons or nodes in the fully-connected layer is fc_size. ReLU is used so we can learn non-linear relations. The output of the first fully-connected layer is 128. Means there is an arbitrary number of images and fc_size == 128. Add another fully-connected layer that outputs vectors of length 10 for determining which of the 10 classes the input image belongs to. Note that ReLU is not used in this layer.

There are 55,000 images in the training-set. It takes a long time to calculate the gradient of the model using all these images. We therefore only use a small batch of images in each iteration of the optimizer. If your computer crashes or becomes very slow because you run out of RAM, then you may try and lower this number, but you may then need to perform more optimization iterations.

Function for performing a number of optimization iterations so as to gradually improve the variables of the network layers. In each iteration, a new batch of data is selected from the training-set and then TensorFlow executes the optimizer using those training samples. The progress is printed every 100 iterations.

In distributed system, We use Spark architechture for that we use pyspark library. Because Apache Spark and TensorFlow are both open-source projects that have made significant impact in the world of enterprise software in recent years. TensorFlow provides a foundational framework for running distributed numerical computations, such as deep learning algorithms, while Spark is a general Hadoop-like, large-scale data processing framework that's also a popular choice for more traditional machine learning algorithms using MLlib.

Spark provides a framework As shown in Figure 3 and Figure **??** for big data computations, and the type of datasets that power TensorFlow algorithms tends to be large. This leads to a possible intersection between the two frameworks: using Spark to preprocess the input to TensorFlow.

The interesting thing here is that even though TensorFlow itself is not distributed, the hyperparameter tuning process is embarrassingly parallel and can be distributed using Spark. In this case, we can use Spark to broadcast the common elements such as data and model description, and then schedule the individual repetitive computations across a cluster of machines in a fault-tolerant manner.

Experiment Results and Discussion

The handwritten digit MNIST database is made of 60,000 training samples and 10,000 test samples. The digits have been size-normalized and centered in a fixed size image. In Random forest algorithm, we would like to have an idea of the result variabilities. We have therefore divided the original training set into two subsets of a training set and a test set. According to the two different ways. We have go 92% accuracy in csv form

```
Accuracy on Test-Set: 9.7% (972 / 10000)
Optimization Iteration:      1, Training Accuracy:  12.5%
Time usage: 0:00:00
Optimization Iteration:     90, Training Accuracy:  62.5%
Optimization Iteration:     91, Training Accuracy:  59.4%
Optimization Iteration:     92, Training Accuracy:  65.6%
Optimization Iteration:     93, Training Accuracy:  70.3%
Optimization Iteration:     94, Training Accuracy:  68.8%
Optimization Iteration:     95, Training Accuracy:  65.6%
Optimization Iteration:     96, Training Accuracy:  76.6%
Optimization Iteration:     97, Training Accuracy:  76.6%
Optimization Iteration:     98, Training Accuracy:  79.7%
Optimization Iteration:     99, Training Accuracy:  68.8%
Optimization Iteration:    100, Training Accuracy:  62.5%
Time usage: 0:00:11
Accuracy on Test-Set: 73.5% (7345 / 10000)
Optimization Iteration:    201, Training Accuracy:  73.4%
Optimization Iteration:    301, Training Accuracy:  84.4%
Optimization Iteration:    401, Training Accuracy:  90.6%
Optimization Iteration:    501, Training Accuracy:  95.3%
Optimization Iteration:    601, Training Accuracy:  89.1%
Optimization Iteration:    701, Training Accuracy:  87.5%
Optimization Iteration:    801, Training Accuracy:  93.8%
Optimization Iteration:    901, Training Accuracy:  93.8%
Time usage: 0:01:34
```

Fig. 4. Output and Accuracy of CNN

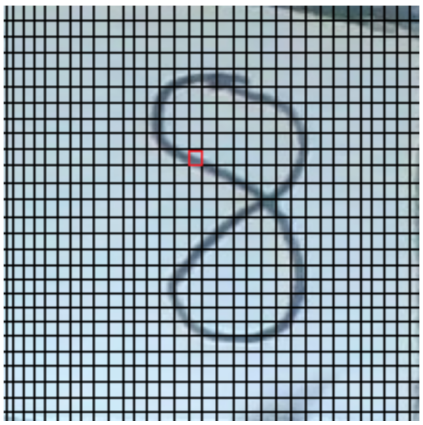

Fig. 5. Future Work Scanning image and send for processing

test data and perfectly recognized an image file. See Figure 4, The output of CNN.

Future Work

As for future work we are scanning images from camera see Figure 5. And then classify it. In this scenario We run a program which start webCam and capture frame in every five seconds and then crop digit from that frame. Now it may be possible that size of crop image is not 28*28 pixels. So, to overcome this situation we are developing a method which divide image file into 28*28 blocks. And then find mode value of that block and consider as a pixel and then that image will use as input in our classifier to classify.

CONCLUSION

In our experience we have seen that Convolutional Neural Network works much better at recognizing handwritten digits than the Random Forest. The Convolutional Network gets a classification accuracy of about 94%, or even more if you make some adjustments and iterate model several more time, compared to only 92% for random forest model. However, the Convolutional Network is also much more complicated to implement and takes more computational time, and it is not obvious from looking at the filter-weights why it works and why it sometimes fails. So we would like an easier way to program Convolutional Neural Networks of Tansorflow on Apache Spark and we would also like a better way of visualizing their inner workings.

References

[1] https://www.tensorflow.org
[2] http://scikit-learn.org/stable/index.html
[3] http://ieeexplore.ieee.org/document/4377074/
[4] http://ieeexplore.ieee.org/document/7780622/
[5] https://databricks.com/blog/2016/01/25/deep-learning-with-apache-spark-and-tensorflow.html