Digit Recognition from Street View Imagery using Convolutional Neural Networks

E4040.2019Fall.MDNR.report
Ananye Pandey ap3885, Apoorva Srinivasan as5697
Columbia University

Abstract

We classify real-world house number digits using convolutional neural networks (ConvNets). ConvNets are hierarchical feature learning neural networks whose structure is biologically inspired. We contrast our results by creating a deeper network of twelve hidden layers inspired by the original paper and achieve an accuracy of 92.46%. We also illustrate the need for depth in the training of some real life problems like number and letter recognition using neural networks.

1. Introduction

Figure 1. 32x32 cropped samples from the classification task of SVHN dataset. Each sample assigned is only a single digit label (0 to 9) corresponding to the center digits

Recognizing numbers in street view photographs is an important component of modern-day map making. It has also been a problem of interest to the optical character recognition community. Arbitrary digit recognition in photographs is highly challenging due to the wide variability in the visual appearances of text on account of large number of fonts, sizes, styles, colors, orientation and character arrangement. This problem is further complicated by environmental factors such as lighting, shadows and occlusions as well as by image acquisition factors such as blur, resolution, motion etc.

In this paper, we focus on recognizing single digit number from Street View panoramas in publicly available SVNH data. We focus on single digit because of the loss of generality in available data about sequence length, orientation and the correct order. While this minimizes the need to recognize the order of the digits, the above listed complexities still apply to this sub-domain.

In this paper, we use ConvNet and a modified version of it with added hidden layers[our best configuration had twelve layers], all with feedforward connections. We evaluate both approaches on publicly available Street View House Numbers(SNVH) dataset and achieved an accuracy of 92.46%. We then compare these methods to the original paper.

The key contributions of this paper are (a) convNet and modified ConvNet approaches to digit recognition problem from StreetView (b) a different approach to preprocessing involving image graying and normalization (c) achieve an accuracy comparable to a human operator - 92.46%

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The authors of the original paper have employed a DistBelief implementation of convolutional neural network with 11 hidden layers, and they have evaluated the performance of arbitrary multi-digit recognizing on publicly available SVNH dataset. The network had eight convolutional hidden layers, one locally connected layer and two dense layers before the softmax layer was applied at the output. The size of units in each of the hidden layers was as follows [48, 64, 128, 160, 192, 192, 192, 192, 3072, 3072, 3072]. The paper used convolutional kernels of size [5x5] in each of their convolutional layers and pooling of size [2,2] in each of their pooling layers. They used alternating strides of 2 and 1.

TensorFlow was part of a former Google product called DistBelief. DistBelief evolved into what is known today as TensorFlow to address the key requirements of scalability, flexibility and portability that is required for many deep learning models today. The network built by us was done using TensorFlow 1.13.

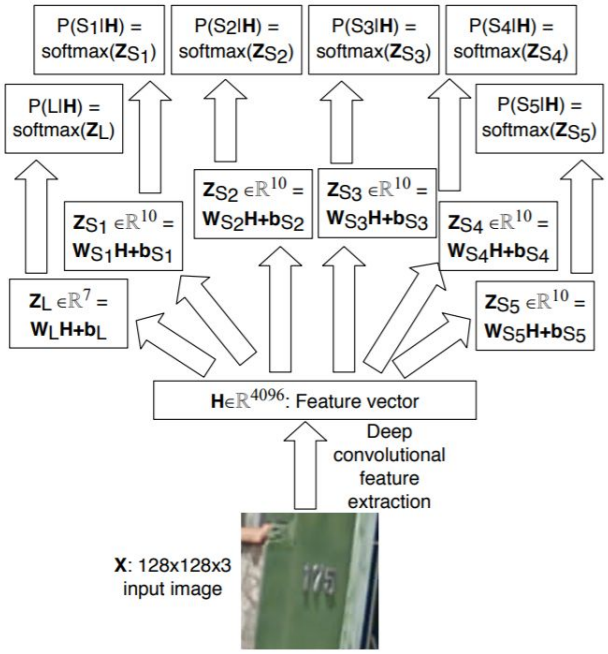

Figure 2. Flowchart from original paper showing network configuration [3]

2.2 Key Results of the Original Paper

The authors had achieved an accuracy with their proposed model of over 96% on SVHN dataset in recognizing complete street numbers. On a per-digit recognition task, they show that their model improves upon the state-of-the art, achieving 97.84% accuracy. They also evaluated their method on a more challenging dataset, Street View imagery containing several tens of millions of street number annotations and achieved over 90% accuracy. To further test the general applicability of their approach, they evaluated its performance on reCAPTCHA and achieved an accuracy of 99.8%

3. Methodology

The following section talks about each step carried out to achieve the reported accuracy.

We preprocess our data the following way; since we are interested in recognizing singular digits, the region boundaries for each digit are calculated and the digits are cropped with their corresponding labels updated for our training task While plotting random images from both testing and training dataset, we notice that one of the digits was incorrectly labeled. Digit 0 is incorrectly labelled as 10. Since we're interested in single digits, we relabeled it to 0. We also convert it to grayscale since it makes the training faster due to implementing single channel convolution and from sampling some images, we notice that the accuracy of classification remains unchanged when compared to results from a human operator. To give features equal chances and so that stochastic gradient descent can achieve convergence with improved stability, we then, normalize the dataset. Finally, we did one hot encoding on the labeled data for the purpose of calculating cross-entropy losses.

The basic ConvNet network consists of two hidden convolutional layers, after each of which there is a pooling layer, followed by two fully connected layer and a final dropout layer before applying a softmax at the output layer.

We modify the configuration of the ConvNet by introducing four additional convolutional layers and two extra densely connected layers before applying a softmax at the output layer.

We use softmax classifier to perform classification of our 10 classes of outputs (0 to 9). Softmax is a one versus all extension of a sigmoid function and returns the normalized log probabilities of each class in each training step. We then compare the probability value to a threshold (set by default as 0.5) to distinguish between whether the output of a certain class is set to 1 (positive) if its probability is greater than or equal to this threshold, and if the probability of output is less than the threshold, its output is set to 0 (negative).

We also explain the necessity of depth in real-life applications of neural networks by using networks of varying lengths to learn the problem at hand.

3.1. Objectives and Technical Challenges

Our main objective is to achieve an accuracy of at least 90% to make it comparable with the accuracy of an average human operator. We then compare deeper versions of the network to achieve higher accuracy. In order to achieve this, our approach was to implement a basic ConvNet model of 6 hidden layers to contrast the difference in results based on the depth of the network.

The main technical challenges faced are the size of the dataset required for training. We have however used data preprocessing to overcome this problem.

Another major challenge in implementation is the bias-variance tradeoff. Deeper networks have more parameters, and hence encapsulate variance in the noise as well. Such a trained network will then fail to perform well on a test dataset even after displaying high accuracies during training. This can be overcome by implementing regularization.

3.2. Problem Formulation and Design

Data Preprocessing Step:

1. cropping centered number images to size [32*32*3]
2. Splitting training data into training and validation data (87:13 split)
3. Using [0.2990 0.5870 0.2440] as a scheme to multiply the red,green and blue pixels to convert the images into [32*32*1] grayscale image.
4. convert label 10 to 0 in the data
5. one-hot-encode labels.
6. Save data locally for use by network

Creating network and training:

1. Create basic ConvNet model on Tensorflow with 6 hidden layers and ten output units.
2. Initialize all layer sizes and weight (kernel) sizes.
3. Initialize optimizer to be used (we use Adam Optimizer)
4. Create a batch of training data (we use batch size 512) and calculate the cross entropy loss between the labels and the network softmax output.
5. Train network with optimal number of epochs to converge on a global minimum.
6. Test network on test-set.
7. Create deeper network and repeat steps 1 to 6.
8. Compare results

4. Implementation

We describe each step of the network in detail along with the implementation.

4.1. Deep Learning Network

4.1.1 Architecture

The ConvNet architecture, inspired from biology, is composed of repeatedly stacked feature stages. We use three main types of layers to build ConvNet architecture: Convolutional Layer, Pooling Layer and Fully-Connected Layer.

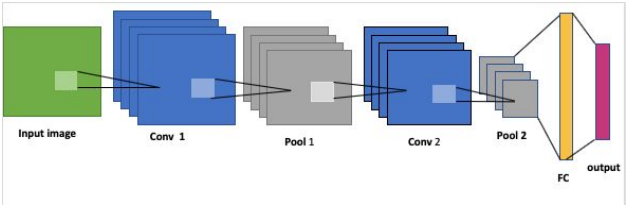

Figure 3. ConvNet architecture

In our case, input [32x32x1] will hold the preprocessed pixel values of the image, an image of height 32 and width 32 with one greyscale channel. Our convolutional kernel size was 5x5 with zero padding. We have 6 convolutional layers of size [48, 48, 64, 64, 128, 128]. The size of pooling kernel was [2,2] and we picked the strides to be equal to 2. We flattened the output of the final convolution layer before connecting it to three fully connected layers with 256 units in each layer. Finally we have a softmax layer with 10 output units to calculate the log probability of each output class. We chose Relu as the activation function for our layers to preserve long term data dependency during backpropagation training.

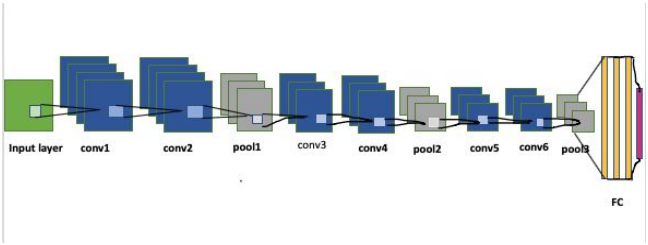

Figure 4. Architecture of network used

The output was compared to the label and the network was trained over 4 epochs using Adam Optimizer for backpropagation with momentum and in batches of 512.

The loss histories of the training and validation losses was recorded.

4.2. Software Design

The following section explains in detail using a pseudocode each section of our implementation of this project.

Data Preprocessing:

```
Data Preprocessing Pseudoalgorithm:

load training and testing data
[X_train, X_test, y_train, y_test]
y_train[y_train == 10] = 0 //convert label 10 to 0
y_test[y_test == 10] = 0
X_train, y_train, X_val, y_val = split_dataset(0.13)

function rbg2gray(input){
   return input *[0.299 0.587 0.114]
}
X_train = rbg2gray(X_train) // convert to grayscale
X_val = rbg2gray(X_val)
X_test = rbg2gray(X_test)

datamean = mean(X_train, axis = 0) //normalization
datastdev = std(X_train, axis = 0)
X_train = (X_train - datamean) / datastdev
X_val = (X_val - datamean) / datastdev
X_test = (x_test - datamean) / datastdev

y_train = one_hot_encode(y_train) //one-hot-encoding
y_val = one_hot_encode(y_val)
y_test = one_hot_encode(y_test)
```

Figure 5. Data preprocessing pseudocode

Network Pseudocode:

```
Neural Net Pseudocode:
//create network
function cnn(inputs){
 convlayer1 = layer.convolve(inputs, kernel = [5,5], units = 48)
 poollayer1 = layer.maxpool(convlayer1, size = [2,2], stride = 2)
 convlayer2 = layer.convolve(poollayer1, kernel = [5,5], units = 64)
 poollayer2 = layer.maxpool(convlayer2, size = [2,2], stride = 2)
 convlayer3 = layer.convolve(poollayer2, size = [5,5], units = 128)
 poollayer3 = layer.maxpool(convlayer3, size = [2,2], stride = 2)
 flatlater = reshape(poollayer3, [-1,4*4*128])
 denselayer1 = layer.dense(flatlayer, units = 256)
 denselayer2 = layer.dense(denselayer1, units = 256)
 denselayer3 = layer.dense(denselayer2, units = 256)
 dropoutlayer = layer.dropout(denselayer3, dropout_rate = dr)
 outputlayer = layer.softmax(dropoutlayer, units = 10)
 return outputlayer
}
//function to generate batches of data
function batch_gen(x,y,batchsize){
 for i in (0, y.shape[0],batchsize){
  end = min(x.shape[0], i+batchsize)
  yield (x[i:end], y[i:end])
 }
}
//training
output = cnn(inputs)
epochs = 4
for e in (epochs){
 loss = softmax_cross_entropy_loss(y,outputs)
 optimizer = AdamOptimizer().minimize(loss)
 accuracy = compare(outputs, y)
 initialize session
 for x, y in batch_gen(X_train,y_train,512){
  l, acc, _ = run[loss, accuracy, optimizer]
  //validation
  for x,y in batch_gen(X_val, y_val, 512){
   l, acc = run[loss, accuracy]
  }
 }
}
//testing
for x,y in batch_gen(X_test, y_test, 512){
 l, acc = run[loss, accuracy]
}
```

Figure 6. Neural Network Creation and training pseudocode

5. Results

5.1. Project Results

We use the testing dataset to calculate the reported accuracies of the networks created.

The basic ConvNet model was used on our dataset of individual preprocessed house number images and achieved an accuracy of 86%.

We added three extra convolutional hidden layers along with two fully connected layers and modified the network to achieve an accuracy of 92.46%.

Our network recorded a training time of 22 minutes using TensorFlow 1.13 on a local machine with an intel i5 CPU and an NVIDIA Geforce MX150 GPU.

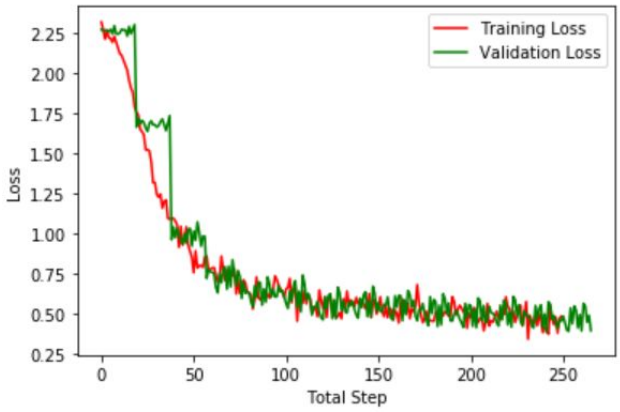

Figure 7. Training and Validation loss history of ConvNet

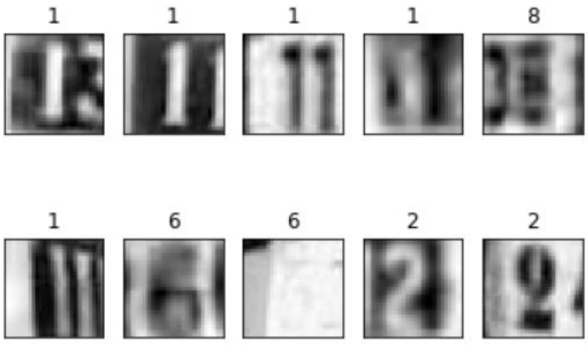

Figure 8. Correctly labeled outputs in greyscale

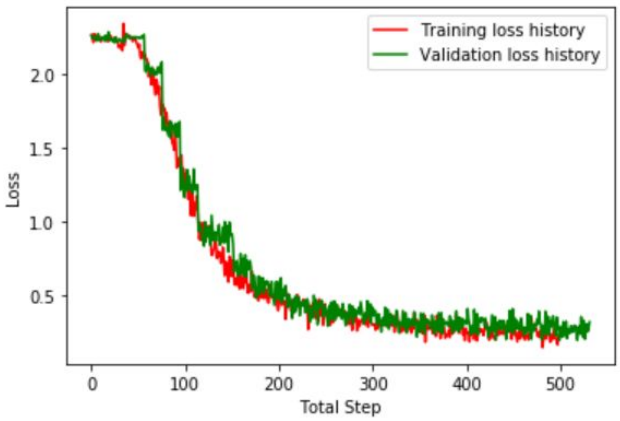

Figure 9. Training and Validation loss history of our network

5.2. Comparison of Results

By changing the number of hidden layers, we can bring about a remarkable proof of necessity of depth in our network to achieve maximum accuracy.

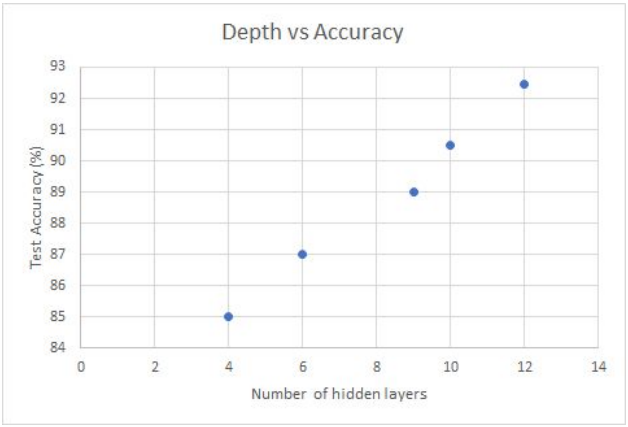

Figure 10. Comparison of network depth vs Test accuracy

The results in paper [3] achieved by creating a network of eleven hidden layers with over 1000 units in each fully connected layer achieved an accuracy of 96%.

We hypothesize that owing to some percentage loss in data during preprocessing, a network with lesser units was chosen comparable to the size of each data and the number of data available for training. An accuracy of 92.5% on a test image is quite comparable to the accuracy of an average human operator.

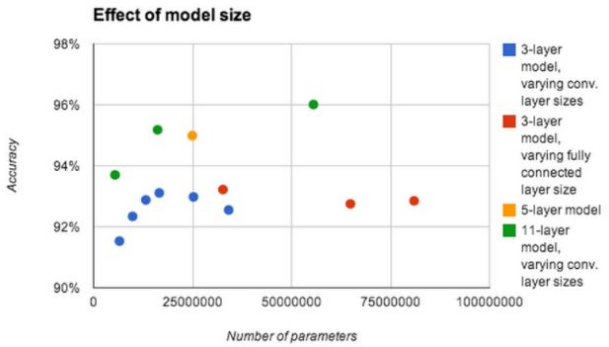

Figure 11. Effect of model size on model accuracy [3]

5.3. Discussion of Insights Gained

In this paper, we try a different approach to per digit number recognition problem from Street View Imagery using ConvNet and a modified version of it. We achieve an accuracy of 92.7% with our network of 12 layers while the authors in the original paper had achieved an accuracy of 97.84% with 11 layers for the same task. There are a number of differences in our approach compared to the authors' approach in the original paper. The fundamental difference is the adoption of multi-digit recognition problem in the original paper while we limit our focus to recognizing a single digit. In that respect, our dataset was limited to the cropped images of publicly available SVHN, while the original paper used full images. While the authors evaluated the same model on different datasets to evaluate performance, we focused on identifying the best model to recognize single digits from SVHN. We believe that with a larger dataset and a deeper network we would be able to achieve an accuracy of the authors' from the original paper. We optimize the number of epochs to allow for early stopping and cross-validation was performed at each step.

6. Conclusion

We show than an accuracy of 92.5% can be achieved with our network of modified ConvNet with 12 layers to perform single digit recognition task from Google's Street View Imagery. We then compare our models performance to that of the authors' from the original paper for the same task. We further illustrate the pattern of increasing accuracy with deeper networks.

7. Acknowledgement

We would like to thank Dr. Zoran Kostic for continued guidance and support.

8. References

[1]https://github.com/cu-zk-courses-org/e4040-2019fall-project-mdnr-ap3885-as5697

[2] H. Li, "Author Guidelines for CMPE 146/242 Project Report", *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.

[3] https://arxiv.org/abs/1312.6082

[4] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011. (PDF)

9. Appendix

9.1 Individual student contributions in fractions - table

	ap3885	as5697
Last Name	Pandey	Srinivasan
Fraction of (useful) total contribution	50%	50%
What I did 1	Data preprocessing	Data preprocessing
What I did 2	Deep network training	ConvNet training
What I did 3	Report writing Debugging	Report writing