Street View House Number Recognition using Deep Convolutional Neural Networks

Group Members:
Jyoti Narwariya (1611CS05)
Nikhil Jaiswal (1611CS09)

Problem Statement

- To correctly detect a series of numbers given an image of house numbers by training a convolutional neural network with multiple layers.

Dataset

- Google’s The Street View House Numbers (SVHN) dataset
- SVHN is obtained from house numbers in Google Street View images.

SVHN Features

- 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.
- 73257 digits for training, 26032 digits for testing, and 531131 additional to use as extra training data
- Comes in two formats:
    - Original images with character level bounding boxes.
    - MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides).

We have used the full number format dataset

Fig.1: Samples from Full number format of dataset with bounded boxes for digits.

Fig Source: Google

Exploring Dataset

- Colored house-number images with character level bounding boxes - train.tar.gz, test.tar.gz , extra.tar.gz
- Each tar.gz file contains the original images in png format, together with a digitStruct.mat file
- The digitStruct.mat file contains a struct called digitStruct with the same length as the number of original images.
- Each element in digitStruct has the following fields: name which is a string containing the filename of the corresponding image. bbox which is a struct array that contains the position, size and label of each digit bounding box in the image

Algorithmic Approach

- We have used Convolutional Neural Network due to their ability to work on the raw pixels of the image.
- Benefits of CNN:
  - Fewer weights
  - Preserve spatial representation amongst pixels

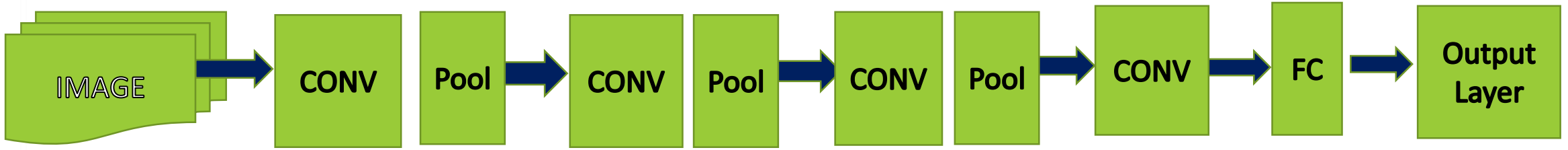


Fig Source: Google

Model Description

Layer	Description
Input Layer	InputLayerShape(30607,48,48,1)
Convolution Layer 1	Receptive Field: 3X3 ,Padding: VALID, Strides:1 Filter:16, Activation:RELU
Pooling Layer 1	Strides:[1,2,2,1],pooling size:2
Convolution Layer 2	Receptive Field: 3X3 ,Padding: VALID, Strides:1 Filter:32, Activation:RELU
Pooling Layer 2	Strides:[1,2,2,1],pooling size:2
Convolution Layer 3	Receptive Field: 3X3 ,Padding: VALID, Strides:1 Filter:48, Activation:RELU
Pooling Layer 3	Strides:[1,2,2,1],pooling size:2
Convolution Layer 4	Receptive Field: 3X3 ,Padding: VALID, Strides:1 Filter:64, Activation:RELU
Dropout	Prob:0.25
Fully Connected Layer	Nodes:64
Softmax	4 softmax layer for 4 digit

Proposed Architecture

- Download and extract the entire SVHN multi dataset
- Crop images to bounded region 48x48 to find digits.
- Find Images with more than 4 digits and delete it from training set and test set.
- Generate validation set from training set.
- Use the preprocessed dataset and train using multi layer Convolution Neural Network.
- Use test data to check for accuracy of the trained model to detect number from street house number image.

Works done so far

- Detailed study of Convolutional Neural Networks from online video tutorials(CS231n Convolution Neural Network for Visual Recognition)
- Gone through different CNN architectures (LeNet / AlexNet / ZFNet / GoogLeNet / VGGNet)
- Read some of the research papers based on this topic
  - Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks – Google Research 2014
  - Reading digits in natural images with unsupervised feature learning – NIPS Workshop 2011

Literature Survey

Algorithms	Paper description	SVHN-test (Accuracy)
Binary Features	Reading Digits in Natural Images with Unsupervised Feature Learning, NIPS Workshop 2011	63.3%
K-Means (feature learning algorithm)	Reading Digits in Natural Images with Unsupervised Feature Learning, NIPS Workshop 2011	90.6%
Stacked sparse Auto-Encoder (fixed length feature vector)	Reading Digits in Natural Images with Unsupervised Feature Learning, NIPS Workshop 2011	89.7%
Human Performance	Reading Digits in Natural Images with Unsupervised Feature Learning, NIPS Workshop 2011	98%
Conditional probabilistic model of sequences	Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks, Google Research 2013.	96.03%

Data Preprocessing

1.first generate dataset and their labels from different folder train and test with image size 48/32.

2 “ generateDataset Method”- this method first take one image file name and find all boundary boxes information from digitstruct box.

- Then calculate approximate upper left most corner co-ordinate (x1,y1) and lower right most corner co-ordinate (x2,y2).After that crop image by x1,y1,x2,y2 co-ordinates information then convert 3 channel image in one channel using dot product.
- Finally store the image and its label information in dataset and labels variable and return these variables.

3.In this step delete all images from training and testing dataset that has more than 4 digits in image.

4.Create validation set from training set(about 10% of training data).

Data Preprocessing

Store box information in respective list of features and store the labels in label array and filename in name variable

#calculate left,top(x1,y1) and right,bottom(x2,y2) using bbox height of different digits presents in a image and crop the image

Train_data (30607, 48, 48, 1)
train_lables(30607,5)

Store the object of train and test datasets in numpy array

Delete all images data from dataset that has more than 4 digits

Data Preprocessing Output

```
train dataset label shape: (33402, 5)
After delete train dataset shape: (33391, 48, 48, 1)
train dataset label shape: (33391, 5)

test data loading.........................

test dataset shape: (13068, 48, 48, 1)
test dataset label shape: (13068, 5)
After detele test dataset shape: (13066, 48, 48, 1)
test dataset label shape: (13066, 5)
test: (2718, 48, 48, 1) (2718, 5)
validation: (2784, 48, 48, 1) (2784, 5)
train: (30607, 48, 48, 1) (30607, 5)
```

Training of preprocessed data

- Use the preprocessed dataset and train a multi layer Convolution Neural Network.
- First compute all weights and bias variables for convolution layer, fullyconnect layer and for softmax .
- Create model using 8 layer : conv-->pool-->conv-->pool-->conv-->pool-->conv-->dropout--->fullyConnected
- In final architecture, GradientDescentOptimizer is used that provide higher accuracy than Adam optimizer.
- Run the optimizer using active session and find the accuracy and loss for different dataset store in the list and print it on terminal window and At last plot the error and accuracy using corresponding list of loss and accuracy.

Output After training and testing

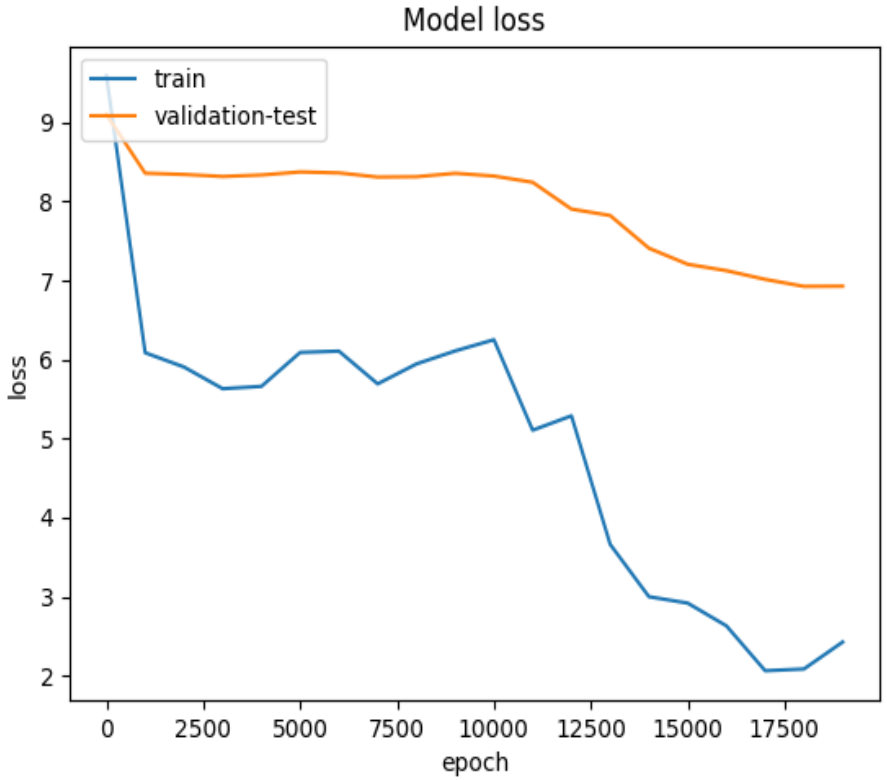
Output-

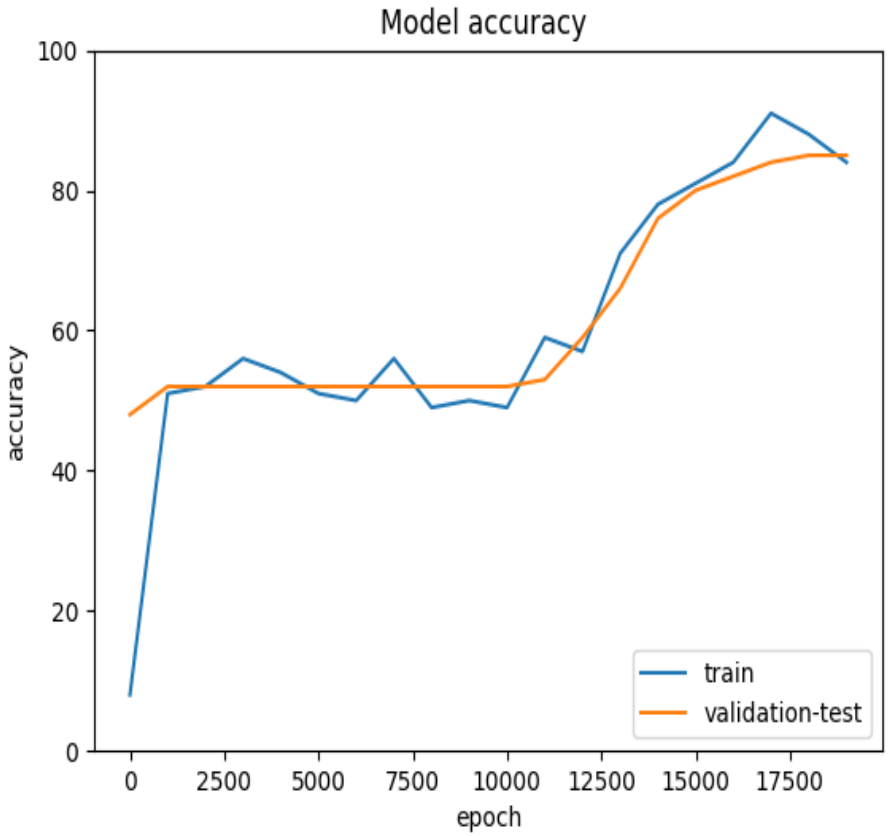
Final Accuracy for testing = 88% and Final Loss for testing = 6.93%

```
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine and could speed up CPU comp
utations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, but these are available on your machine and could speed up CPU compu
tations.
at epoch[==============================] 0/20000...batch loss:9.59152698517, batch accuracy: 8%,Validation loss:9.10105133057% Validation accuracy: 48%
at epoch[==============================] 1000/20000...batch loss:6.08597707748, batch accuracy: 51%,Validation loss:8.35824298859% Validation accuracy: 52%
at epoch[==============================] 2000/20000...batch loss:5.90569210052, batch accuracy: 52%,Validation loss:8.34221076965% Validation accuracy: 52%
at epoch[==============================] 3000/20000...batch loss:5.63084840775, batch accuracy: 56%,Validation loss:8.31690979004% Validation accuracy: 52%
at epoch[==============================] 4000/20000...batch loss:5.66035699844, batch accuracy: 54%,Validation loss:8.33533573151% Validation accuracy: 52%
at epoch[==============================] 5000/20000...batch loss:6.08997392654, batch accuracy: 51%,Validation loss:8.3739566803% Validation accuracy: 52%
at epoch[==============================] 6000/20000...batch loss:6.10619974136, batch accuracy: 50%,Validation loss:8.36196231842% Validation accuracy: 52%
at epoch[==============================] 7000/20000...batch loss:5.69216012955, batch accuracy: 56%,Validation loss:8.3093214035% Validation accuracy: 52%
at epoch[==============================] 8000/20000...batch loss:5.9442691803, batch accuracy: 49%,Validation loss:8.31357002258% Validation accuracy: 52%
at epoch[==============================] 9000/20000...batch loss:6.10888957977, batch accuracy: 50%,Validation loss:8.3569078705% Validation accuracy: 52%
at epoch[==============================] 10000/20000...batch loss:6.25065040588, batch accuracy: 49%,Validation loss:8.32213783264% Validation accuracy: 52%
at epoch[==============================] 11000/20000...batch loss:5.10578632355, batch accuracy: 59%,Validation loss:8.24301242828% Validation accuracy: 53%
at epoch[==============================] 12000/20000...batch loss:5.28755092621, batch accuracy: 57%,Validation loss:7.90429592133% Validation accuracy: 59%
at epoch[==============================] 13000/20000...batch loss:3.66077613831, batch accuracy: 71%,Validation loss:7.82266759872% Validation accuracy: 66%
at epoch[==============================] 14000/20000...batch loss:2.99836802483, batch accuracy: 78%,Validation loss:7.40903186798% Validation accuracy: 76%
at epoch[==============================] 15000/20000...batch loss:2.91745138168, batch accuracy: 81%,Validation loss:7.20571517944% Validation accuracy: 80%
at epoch[==============================] 16000/20000...batch loss:2.62941789627, batch accuracy: 84%,Validation loss:7.12474679947% Validation accuracy: 82%
at epoch[==============================] 17000/20000...batch loss:2.06134486198, batch accuracy: 91%,Validation loss:7.01433849335% Validation accuracy: 84%
at epoch[==============================] 18000/20000...batch loss:2.08369922638, batch accuracy: 88%,Validation loss:6.92606639862% Validation accuracy: 85%
at epoch[==============================] 19000/20000...batch loss:2.42488503456, batch accuracy: 84%,Validation loss:6.92834091187% Validation accuracy: 85%
Test accuracy: 88%
nik@ubuntu:~/Desktop/digit$
```

Graph for Error and Accuracy

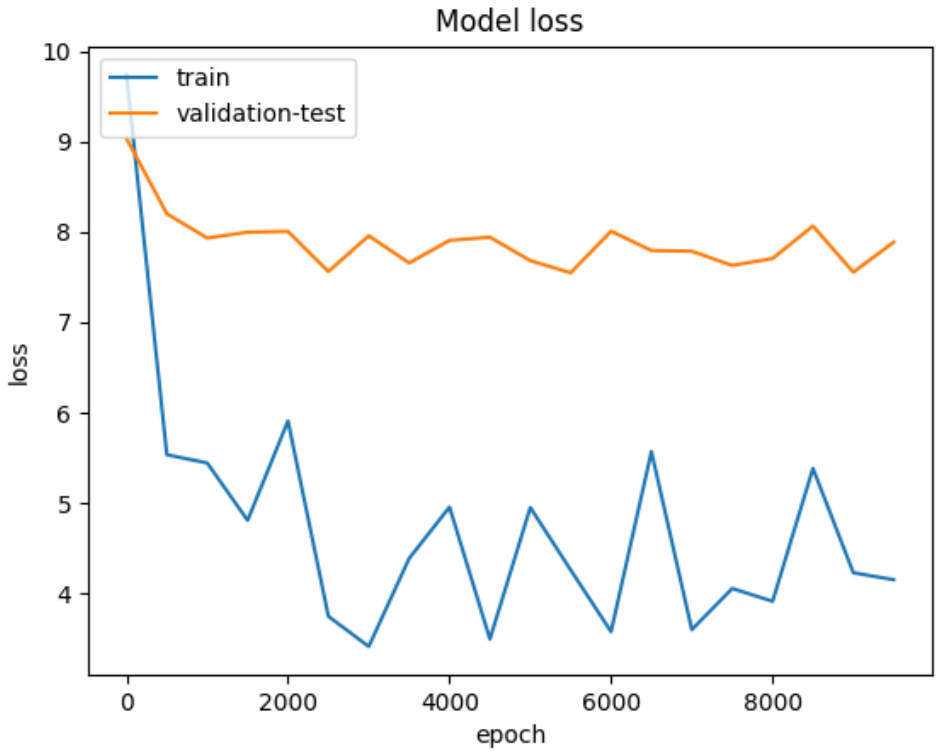
*Conv→Pool→Conv→Pool→Conv→Pool→Conv→dropout→FullyConnected	GradientDescent (filter size =3)

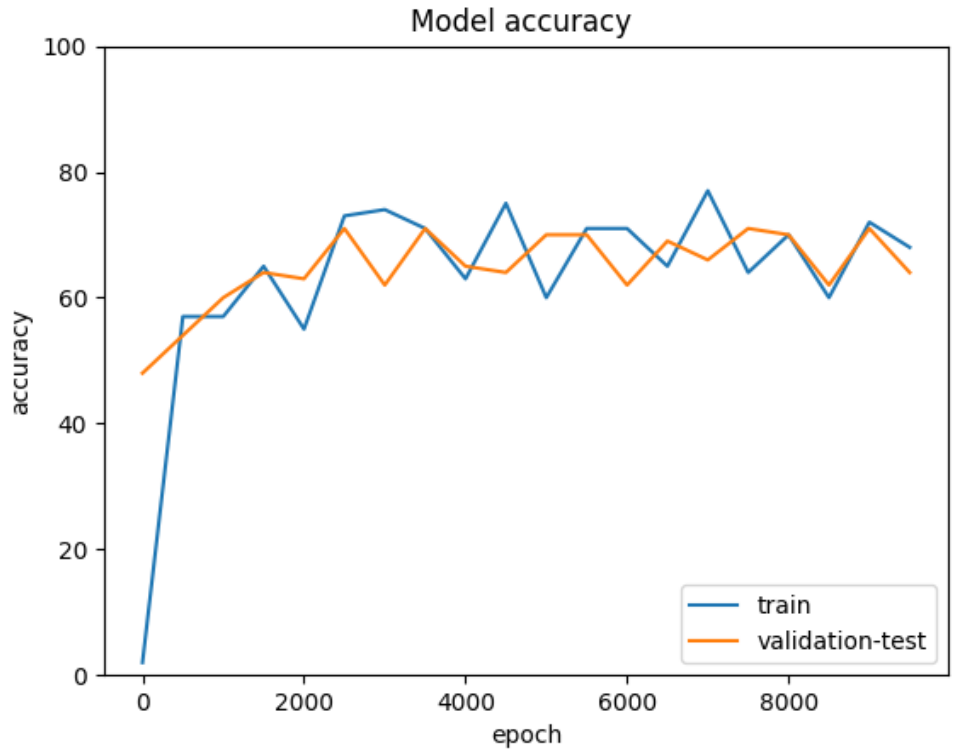




Contd:

Conv→Pool→Conv→dropout→FullyConnected Optimizer : GradientDescent Loss:8.36% Accuracy:56%





Experimental Results

Layer Configuration	Optimizer	Number of Node	Number of Epoch	Loss	Accuracy (on testset)
Conv→Pool→Conv→Pool→Conv→dropout→FullyConnected	Adam	64	5000	9.66%	66%
Conv→Pool→Conv→Pool→Conv→dropout→FullyConnected	GradientDescent	64	5000	9.23%	76%
Conv→Pool→Conv→dropout→FullyConnected	GradientDescent	64	5000	8.36%	56%
Conv→Pool→Conv→Pool→Conv→dropout→FullyConnected	GradientDescent	64	5000	9.66%	66%
Conv→Pool→Conv→Pool→Conv→Conv→FullyConnected	GradientDescent (filter size =3)	32	7000	8.32%	52%
*Conv→Pool→Conv→Pool→Conv→Pool→Conv→dropout→FullyConnected	GradientDescent (filter size =3)	64	20000	6.93%	88%
Conv→Pool→Conv→dropout→FullyConnected→dropout→FullyConnected	GradientDescent (filter size =3)	64	5000	8.72% till 3000 epoch	2% (till 3000 epoch 52%)

Future Work

The final conclusion which we inferred are that when our model consisted of few layers, the accuracy was quite low, on increasing the number of convolution and pooling layers, the accuracy increased to certain extent then finally started decreasing when we further increased the layers. We got the best accuracy of 88% when the total layers were 8.

There can be further improvements in our approach. Firstly, we can improve our accuracy by training our model by using the extra dataset which are available in extra.tar.gz folder. We could also vary the number of layers and number of epochs to improve the accuracy.

References

- 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*.
- Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks, 2014.
- http://ufldl.stanford.edu/housenumbers (dataset link)
- http://cs231n.github.io/

Thank You