

This project is based on the [Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks](https://arxiv.org/abs/1312.6082) paper from Google Street View and reCAPTCHA Teams.

This project implements the multi-digit prediction model described in the paper, but with a smaller convolution neural network.

In this project we will use Public Street View House Numbers (SVHN) dataset to train a deep convolutional neural network to predict multiple digits in one image.

Outside resources:

1. The [Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks](https://arxiv.org/abs/1312.6082) paper from Google Street View and reCAPTCHA Teams. (<https://arxiv.org/abs/1312.6082>)
2. Public Street View House Numbers (SVHN) dataset (format 1) -- <http://ufldl.stanford.edu/housenumbers/>
Dataset citation: 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.

The training dataset contains 33402 house number image and a digitStruct.mat file which indicates the bounding box of digits in the image.

Also, it contains a see_bbox.m file which is a good reference to extract information from digitStruct.mat file.

The testing dataset contains 13068 house number image, a digitStruct.mat file, and see_bbox.m file.

Image in training dataset:



Images in the dataset has different orientation and layout, and have different blur and occlusions on them, which makes the classification difficult.

The digitStruct.mat file contains information of each digit for each image.

Information of digit includes the label (0-9) of digit and bounding box top, left, bottom, right of digit.

Preprocessing:

(This preprocessing procedure is from the paper)

We preprocess the training dataset by first combining the bounding boxes of all digits in the image into one large bounding box.

Then, we expand the bounding box by 30%, scale the expanded result into a 64x64 image, and sample multiple 54x54 images from the 64x64 image.

For testing set, we only preprocess with combining the bounding boxes of all digits in the image into one large bounding box.

original image (displayed in original size)	bounding box	expanded bounding box by 30%	sample1	sample2

Neural Network Model:

Training a model with 8 convolution layers and 2 linear layers costs too much resources on a single laptop computer, so we reduce the number of convolution layers into 5.

Parameter Tuning:

I first start with initial model:

model.summary() from keras

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 54, 54, 3)	0	
convolution2d_1 (Convolution2D)	(None, 54, 54, 64)	1792	input_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 27, 27, 64)	0	convolution2d_1[0][0]
dropout_1 (Dropout)	(None, 27, 27, 64)	0	maxpooling2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 27, 27, 128)	73856	dropout_1[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 13, 13, 128)	0	convolution2d_2[0][0]
dropout_2 (Dropout)	(None, 13, 13, 128)	0	maxpooling2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 13, 13, 192)	221376	dropout_2[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 6, 6, 192)	0	convolution2d_3[0][0]
dropout_3 (Dropout)	(None, 6, 6, 192)	0	maxpooling2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 6, 6, 256)	442624	dropout_3[0][0]
maxpooling2d_4 (MaxPooling2D)	(None, 3, 3, 256)	0	convolution2d_4[0][0]
dropout_4 (Dropout)	(None, 3, 3, 256)	0	maxpooling2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 3, 3, 256)	590080	dropout_4[0][0]
maxpooling2d_5 (MaxPooling2D)	(None, 1, 1, 256)	0	convolution2d_5[0][0]
dropout_5 (Dropout)	(None, 1, 1, 256)	0	maxpooling2d_5[0][0]
flatten_1 (Flatten)	(None, 256)	0	dropout_5[0][0]
dense_1 (Dense)	(None, 128)	32896	flatten_1[0][0]
dropout_6 (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 7)	903	dropout_6[0][0]
dense_3 (Dense)	(None, 11)	1419	dropout_6[0][0]
dense_4 (Dense)	(None, 11)	1419	dropout_6[0][0]
dense_5 (Dense)	(None, 11)	1419	dropout_6[0][0]

dense_6 (Dense)	(None, 11)	1419	dropout_6[0][0]
dense_7 (Dense)	(None, 11)	1419	dropout_6[0][0]
=====			
Total params: 1,370,622			
Trainable params: 1,370,622			
Non-trainable params: 0			

This model has 5 convolution layers. Each convolution layer is followed by a max pooling layer and a dropout layer. Then, the output is flattened into 256 linear unit. Followed by a dense linear layer, and 6 different output layers. The first output layer represents number of digits in the image, and the remaining 5 output layers represent digit (0-9) or invalid (10).

The structure and hyper parameter of initial model is set based on ideas of 5 convolution layers, each with max pooling and dropout, and add linear layer after flattened output with half number of units until the number of unit is roughly the same as number of output (62).

The loss function is the same as the one described in paper, which is the **sum of categorical_crossentropy of 6 outputs**.

Learning rate = 0.0001, Kernel size = 3, Activation = relu, optimizer = RMSProp.

Then, we starts tuning learning rate, kernel size of convolution layers, and activation function with other hyper parameter fixed.

We split the training set into 2 sets for validation.

Accuracy is calculated by the percentage of images that the prediction **exactly** matches the corresponding label.

Learning rate:

learning rate	training time per epoch	loss after 20 epoch	validation accuracy
0.0001 (initial)	275s	0.9546	0.7492
0.001	280s	4.2560	0.3304
0.0005	286s	2.5460	0.5876
0.00005	275s	1.0432	0.7583

Learning rate of 0.001 is too large such that the loss stay between 3.2 and 4.2 since epoch 2.

Learning rate of 0.0005 is still large such that the loss stay between 2.1 and 2.5 since epoch 4.

Learning rate of 0.00005 is too small that the loss is still decreasing after 20 epoch, the loss is around 0.81 and the accuracy is around 0.76 at epoch 40.

From the result, smaller learning rate usually yields higher accuracy if more epochs is trained. Although 0.0001 has lower loss than 0.00005 in the 20 epoch, its loss at 40 epoch is higher than that of 0.00005 .

Kernel size:

kernel size	training time per epoch	loss after 20 epoch	validation accuracy
3 (rerun initial)	275s	0.9771	0.7910
5	420s	0.6457	0.8097
7	720s	0.5987	0.8867
9	887s	0.6939	0.8562

Kernel size of 5, 7, 9 produce better result than that of 3 because of increased number of parameters.

Kernel size 7 has lower loss than Kernel size 5 may be due to its flexibility (more parameters), while Kernel size 9 has too much parameters to learn and have a higher loss in 20 epoch.

From the result, larger kernel size usually yields higher accuracy if more epochs is trained. Although most kernel size choice can still improve accuracy with more epoch trained, the significantly increased training time per epoch introduce difficulties on training with large number of epoch.

Activation function:

activation	training time per epoch	loss after 20 epoch	validation accuracy
relu (rerun initial)	282s	0.9868	0.8028
tanh	287s	0.5950	0.7823
sigmoid	276s	1.9630	0.5306

From the result, sigmoid produce worse result in both loss and accuracy than that of relu and tanh. Although tanh has significantly lower loss than that of relu, the accuracy of relu is higher.

Note that the model of relu is the initial model, we can see the the variance of validation accuracy is high from 3 running results, but the loss stays relatively close in 3 running results.

Combining parameters from those 3 tuning experiments by considering loss and training time, we made the final model.

The final model is:

model.summary() from keras

Layer (type)	Output Shape	Param #	Connected to
=====			
input_1 (InputLayer)	(None, 54, 54, 3)	0	
convolution2d_1 (Convolution2D)	(None, 54, 54, 64)	9472	input_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 27, 27, 64)	0	convolution2d_1[0][0]
dropout_1 (Dropout)	(None, 27, 27, 64)	0	maxpooling2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 27, 27, 128)	401536	dropout_1[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 13, 13, 128)	0	convolution2d_2[0][0]
dropout_2 (Dropout)	(None, 13, 13, 128)	0	maxpooling2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 13, 13, 192)	1204416	dropout_2[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 6, 6, 192)	0	convolution2d_3[0][0]
dropout_3 (Dropout)	(None, 6, 6, 192)	0	maxpooling2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 6, 6, 256)	2408704	dropout_3[0][0]
maxpooling2d_4 (MaxPooling2D)	(None, 3, 3, 256)	0	convolution2d_4[0][0]
dropout_4 (Dropout)	(None, 3, 3, 256)	0	maxpooling2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 3, 3, 256)	3211520	dropout_4[0][0]
maxpooling2d_5 (MaxPooling2D)	(None, 1, 1, 256)	0	convolution2d_5[0][0]
dropout_5 (Dropout)	(None, 1, 1, 256)	0	maxpooling2d_5[0][0]
flatten_1 (Flatten)	(None, 256)	0	dropout_5[0][0]
dense_1 (Dense)	(None, 128)	32896	flatten_1[0][0]
dropout_6 (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 7)	903	dropout_6[0][0]
dense_3 (Dense)	(None, 11)	1419	dropout_6[0][0]
dense_4 (Dense)	(None, 11)	1419	dropout_6[0][0]
dense_5 (Dense)	(None, 11)	1419	dropout_6[0][0]
dense_6 (Dense)	(None, 11)	1419	dropout_6[0][0]
dense_7 (Dense)	(None, 11)	1419	dropout_6[0][0]
=====			
Total params: 7,276,542			
Trainable params: 7,276,542			
Non-trainable params: 0			

The structure of final model is the same as the initial model, but hyper parameters are different and number of epoch is 50.

Learning rate = 0.00005, Kernel size = 7, Activation = tanh



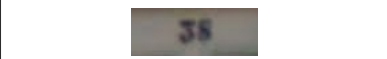

Accuracy on validation set (part of training data): 0.8933



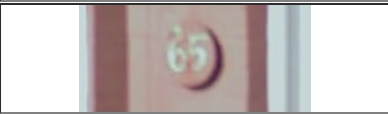
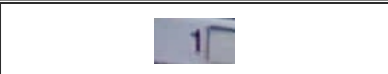

Accuracy on test set (using test dataset): 0.7068

The large difference between accuracy on validation set and accuracy on test set is caused by the procedure of splitting validation set. As we produce 5 samples for each image in training dataset, some of them will be in training set and some of them will be in validation set. Although the position of digit is different, the data in training set and validation set is highly correlated. For the test set, the data is completely new to the model.

Misclassified sample (in test set) of the final model:

For testing set, we only preprocess with combining the bounding boxes of all digits in the image into one large bounding box.

original image	processed image for classification	label	predicted
		13	19
		38	35

		81	54
		5	3
		1	7


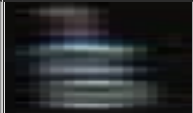


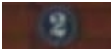


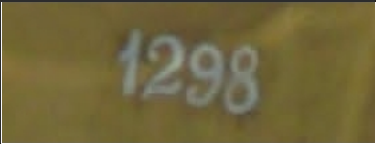



Some images after processing have different blur and occlusions which makes the classification hard. Even human may also confused by processed images with label 13, 38, 1 and make a prediction similar to that trained model. However, the processed images with label 81, 5 is clear enough for human to correctly classify.

Misclassified number of digits sample (in test set) of the final model:

Misclassified number of digits samples are those images that the model cannot even identify its correct number of digits.

To correctly classify image label, the model at least needs to correctly predict number of digits in the image.

The percentage that the model correctly predict the number of digits is **0.9321**, which means the model still make wrong prediction of 887 images out of 13068 images in test set.

original image	processed image for classification	label	predicted
		129	5
		12	122
		2	25
		114	14
		1298	228
		63	630

From the data that the model mispredict number of digits, we can see the model is weak in predicting digits with different layout as the number of digits predicted is off by 2 for the image with label 129. Also, the edge of the numbered sign distract the model to predict one more or one less digit.