Classification and Detection with CNN

Manqing Mao
{GTID: mmao33}

1. Introduction

Literature Review: Recognizing multi-digit numbers in photographs captured at street level is an important component of modern-day map making. A unified approach is proposed that integrates three steps via the use of a deep convolutional neural network that operates directly on the image pixels [1]. This approach is evaluated on the publicly available SVHN dataset [2] and achieve over 96% accuracy in recognizing complete street numbers. Increases in the availability of computational resources [3], increases in the size of available training sets [4], and algorithmic advances such as the use of piecewise linear units [5] and dropout training [6] have resulted in many recent successes using deep convolutional neural networks. Convolutional neural networks have previously been used mostly for applications such as recognition of single objects in the input image. In some cases they have been used as components of systems that solve more complicated tasks. Convolutional neural networks is used as feature extractors for a system that performs object detection and localization [7].

Organization: In this project, I first showed how three datasets, namely training dataset, validation dataset and testing dataset, are prepared for CNN training and inference in Section 2. Methodology including CNN models, model selection and preprocessing for the data is described in Section 3. In Section 4, I first show accuracies for three models and then pick one with the highest validation accuracy and I show the detection results for 5 real-world images and a real-world video. Finally, I summarize this project in terms of challenges and limitations, as well as show what to do next to improve the results for digits classification.

2. Dataset Preparation

2.1 Training Dataset

SVHN is a real-world image dataset, which is very similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images). SVHN is obtained from house numbers in Google Street View images.

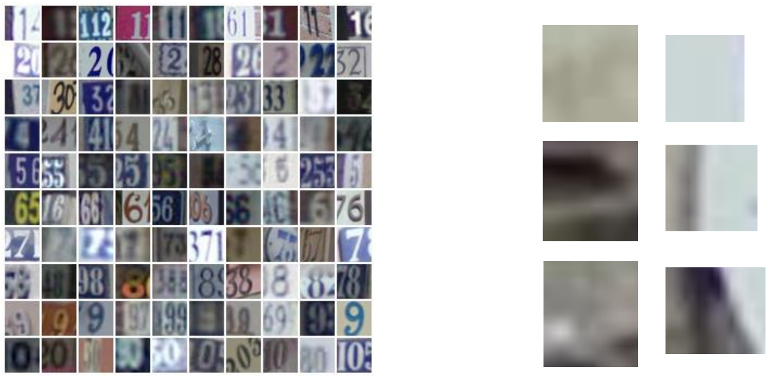
Positive Samples: In this project, I used cropped digits in Format 2 (32-by-32 pixels) in SVHN dataset and randomly selected 20,000 images from original training dataset (named *train_32x32.mat*) as the positive training samples, as shown in Fig. 1 (a).

Negative Samples: I cropped corner images in Format 1 (32-by-32 pixels) in SVHN dataset and randomly selected 10,000 images as the negative training samples and marked them as the label '11', as shown in Fig. 1 (b). More discussion is shown in Section 5.

2.2 Validation and Testing Dataset

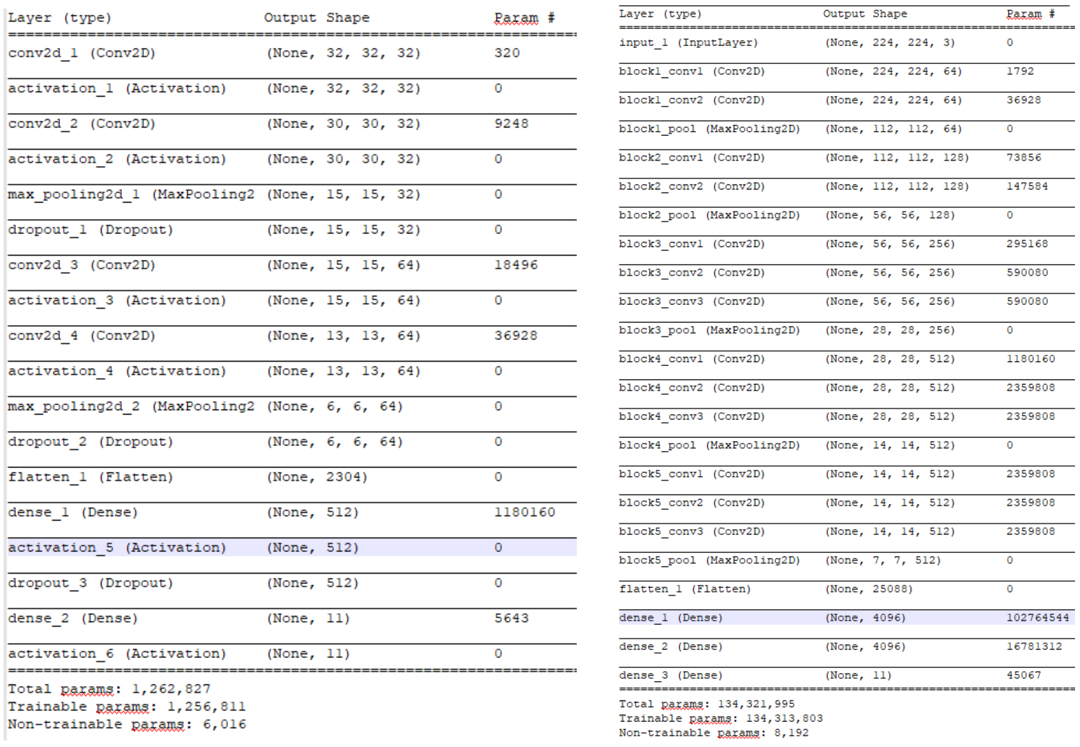
Validation Dataset: Instead of selecting CNN models by using training dataset or test dataset, I generated validation dataset and use it to select a CNN model, which leads to the highest accuracy in the validation dataset. I have chosen 10,000 images from original training dataset (named *train_32x32.mat*) as validation dataset.

Testing Dataset: I have chosen 20,000 images from original testing dataset (named *test_32x32.mat*) as test dataset.



(a) (b)

Figure 1. (a) Positive samples (32-by-32 pixels) and (b) negative samples (32-by-32 pixels) in training dataset.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 11)                5643
_________________________________________________________________
activation_6 (Activation)    (None, 11)                0
=================================================================
Total params: 1,262,827
Trainable params: 1,256,811
Non-trainable params: 6,016
```

(a)

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              102764544
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_3 (Dense)              (None, 11)                45067
=================================================================
Total params: 134,321,995
Trainable params: 134,313,803
Non-trainable params: 8,192
```

(b)

Figure 2. (a) My own CNN architecture and (b) VGG-16 architecture.

3. Methodology

3.1 CNN Models

Three models are required in this final project: my own CNN, VGG-16 and pre-trained VGG-16. I created 4 CONV layers with receptive field of 3×3 and 2 FC layers, as shown in Fig.2 (a). I also summarized VGG-16 in Fig. 2 (b). It has 16 CONV layers with receptive field of 3×3 and 3 FC layers.

3.2 Model Selection

Gradient Descent (GD) Choice: mini-batch GD. Generally speaking, there are three types of GD: stochastic GD, mini-batch GD and batch GD. Among of them, batch GD is the most expensive for large scale ML system since it goes over all training samples for each iteration. Compared to batch GD, stochastic GD is much cheaper since it fits the training sample one by one. In this way, you only fit one training point to update the model. Mini-batch fits the training samples in each mini-batch so that model is updated for each batch. I choose mini-batch GD in this project since mini-batch could be even faster than stochastic GD as mini-batch GD allows you to have a vectorized implementation, which means you can partially parallelize your computation, speeding up the training. Batch size here I have chosen is 1% of total data = (30,000 × 1% = 30).

Loss Function: In classification problem, I used cross-entropy loss, which is a convex loss function and guarantees to find global optimum.

Learning Rate (α): Firstly, I have to clearly state that learning rate I used in this project is a constant. It is not necessarily to be smaller and smaller. This intuition is actually straightforward: the steps (Δθ) are getting smaller and smaller when weights (θ) is approaching the local minimum because the slope is getting flatter and flatter. Secondly, too larger α results in overshoot and fails to converge; while too small α leads too long to converge since you do a baby step every time. So in this project, I choose $\alpha = 0.01$.

Three CNN Models: In order to compare performance of three models, I used three parameters: training accuracy (during training procedure), validation accuracy (during training procedure), and testing accuracy (during inference procedure). Then, I picked one CNN from these three models. This picked model has **the highest validation accuracy**; while its test accuracy might not be the highest.

3.3 Pre-processing for the Real World Image

I picked up 5 images about door numbers and street numbers. The original images are in the folder named images5. The video is took from my office building.

Normalize and Mean Centering: By doing this standardization, our data can achieve invariance to these irrelevant differences.

Sliding Window & Pyramid: A sliding window is a rectangular region of fixed width and height that "slides" across an image. For each of these windows, we would normally take the window region and apply an image classifier to determine if the window has an object that interests us — like digits. Combined with image pyramids we can create image classifiers that can recognize objects at varying scales and locations in the image.

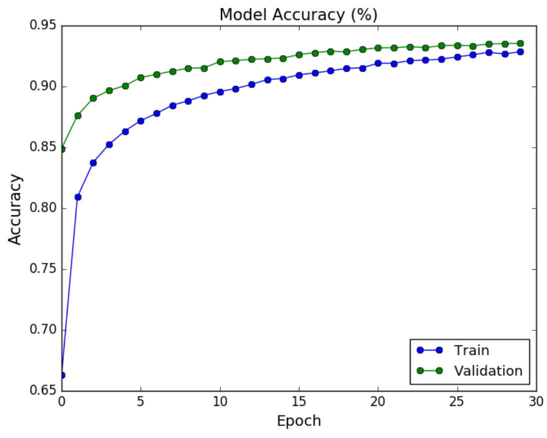
Non-maxima Suppression & Confidence Threshold: I set up a confidence threshold along with non-maxima suppression after the inference to decide which label can be output as a final result.

4. Result Analysis

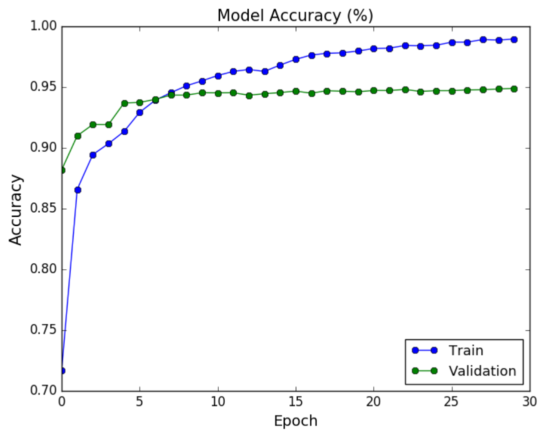
In this section, I first show accuracies for three models and then pick one with the highest validation accuracy. Next, I show the detection results for 5 real-world images and a real-world video.

4.1 Accuracy for Three Models

Accuracies: (1) Training and Validation Accuracies: During training, Fig. 3 shows the accuracies for training dataset and validation dataset as a function of the number of epoch. From the figure, we can see that both training and validation accuracy increase as increasing the number of epoch and finally converge. VGG-16 has higher training accuracy than my own network; while they have comparable validation accuracies. We can see that VGG-16 overfits since it has more layers compared to my own CNN. **(2) Testing Accuracy:** Testing accuracy for three models are listed in the table below. From the table, we can see that the VGG-16 has the highest testing accuracy; while pre-trained VGG-16 has the lowest.



(a)



(b)

Figure 3. Training and validation accuracies for (a) My own CNN architecture and (b) VGG-16 architecture.

Models	Validation Accuracy (%)	Testing Accuracy (%)
My Own CNN	*93.16*	*93.93*
VGG-16	*93.83*	*94.51*
Pre-trained VGG-16	*89.28*	*90.01*

Model Selection: VGG-16 and my own CNN have comparable validation accuracies. Thus, I picked my own CNN architecture for since it has much fewer layers and hence higher inference time-efficiency.

4.2 Digits Classification using Real-world Images

Figure 4 shows five images from real world. These five images shows different scales (IMG1 v.s. IMG3), different lighting conditions (IMG2), different locations: middle (IMG1), upper left (IMG2), middle right (IMG5) and different angles (IMG5).

Figure 5 shows the detection results for these five images. From the figure, we can see that all digits in these 5 images are properly detected. However, I choose different bounding box sizes for these 5 images. This means that different parameters are used for different images, which is main limitation of my method. This will be detailed discussed in Section 5.

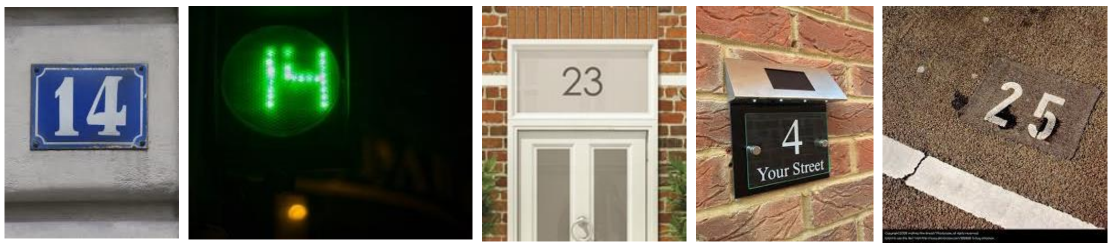


Figure 4. Five images from real world.

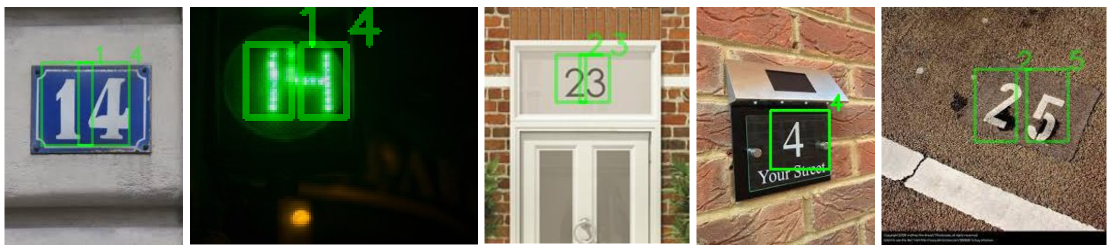


Figure 5. Digits classification for five images.

4.3 Digits Classification using a Real-world Video

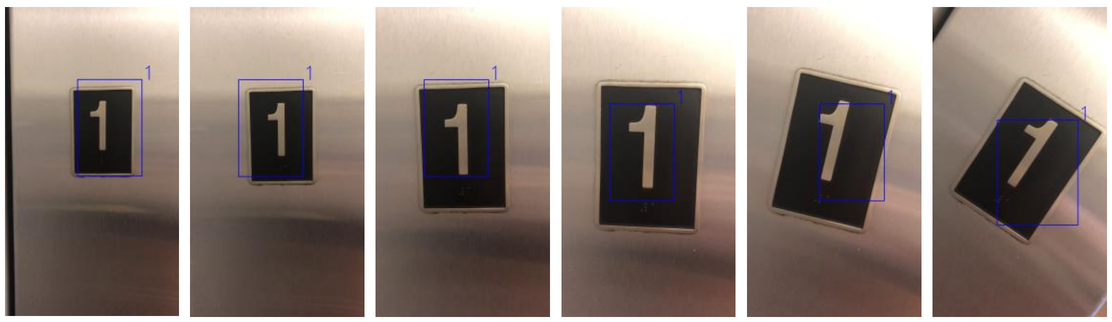


Figure 6. Digits classification for a video. Six frames are shown here.

Figure 6 shows the detection results for six frames from a real world video. 6 Frames are shown for different scales, different location and different rotation angels. From the figure, we can see that the different scales of digits also result in different bounding box. This is because the algorithm will go over the pyramids to detect the digits by using sliding window. If the digit in the image is larger than the pre-defined window, CNN will not detect the digit until it goes to next layer in the pyramid. (The image in the next layer is scaling down by 2, and thus the window can fit the digit.) Thus, we usually start with a larger window and original image. If there is no digits are detected, CNN will go to next-layer image in the pyramid.

5. Summary

5.1 Challenges

Negative Samples in Training Dataset: In this project, I found this is very important to correctly classify the digits in the images from real world. For example, in my case, there are a lot false positive classification. To be more specially, it is very easy to classify a lamppost as digit '1', a clock as digit '0', as well as a corner as digit '7'. Thus, it is very important to add those images as negative sample in the training dataset to tell CNN that those images are not digits. Also, this required more time to find a comprehensive negative samples.

Training Time: Since training actually takes a lot of time, the number of epoch is limited. Besides, the hyper-parameters, such as learning rate and batch size, dropout (pruning), are also required to choose properly to avoid overfitting issue and achieve high validation accuracy.

5.2 Comparison with Related Work

Firstly, by using the SVHN dataset, my CNN testing accuracy (93%) is not good as [3], which has over 96%. Also, my classifier is not as robust as [3] and easy to have false positive detection, as mentioned in Section 5.1.

5.3 Limitation & Improvement

Limitations: To be honest, I pick five images from real-world to avoid some situations: (1) My classifier cannot tell the difference between a real digit '1' from a lamppost as I mentioned earlier. (2) I used different parameters for these five images to achieve good performance, which is actually very unrealistic and un-practical. Imagine, in the real world, we cannot use different parameters for different images.

Improvement Method: If I have more time for this final project, (1) I will add more practical negative samples and re-train my CNN. (2) Those hyper-parameters have to be properly chosen to achieve the accuracy as high as possible. (3) More preprocessing could be applied before CNN inference.

Reference:

[1] Goodfellow I. J., et. al., "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks", in *Computer Vision and Pattern Recognition*, 2014.

[2] SVHN dataset URL: http://ufldl.stanford.edu/housenumbers/

[3] Jarrett, K., et. al., "What is the best multi-stage architecture for object recognition", in *Proc. International Conference on Computer Vision (ICCV)*, pp 2146–2153, 2009.

[4] Glorot, X., et. al., "Deep sparse rectifier neural networks", in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[5] Goodfellow I. J., et. al., "Maxout networks", in *ICML*, 2013.

[6] Hinton, G. E., et. al., "Improving neural networks by preventing co-adaptation of feature detectors. Technical report", arXiv: 1207.0580.

[7] Girshick, R., et. al., "Rich feature hierarchies for accurate object detection and semantic segmentation. Technical report", arXiv:1311. 2524.