Classification and Detection with CNNs

Alex Armbruster
aarmbruster3@gatech.edu

April 15, 2020

Abstract

I implemented a multi stage image processing pipeline for the detection and classification of house numbers using a CNN ensemble. This pipeline is robust to digits of different colors, fonts, sizes, scales, orientations, and positions of the digits with respect to each other. The pipeline is lightweight and accurate, inference takes 1-2 seconds for a large image on an Intel Core i7 2.6 GHz CPU, and each member of the ensemble independently achieves 88% precision and 87% recall on the test set.

1 Introduction

In recent years, Convolutional Neural Networks (CNNs) have shown excellent performance for a wide variety of computer vision tasks, from object recognition, to human pose estimation to facial recognition, eclipsing the performance of classical methods for many of these tasks. In particular, one area where CNNs are widely recognized to be the best method is digit recognition - one of the earliest and most well known applications of deep learning was handwritten character recognition [4]. CNNs perform well on this task because when trained on a large enough data set, they are robust to small variations in pose, shape, orientation and scale, that many older feature based methods are comparatively more brittle against. For my final project, I applied CNNs to the task of recognizing house numbers in pictures taken various distances away from the front of a house. The first stage uses the Maximally Stable Extremal Region (MSER) algorithm [1] as a fast way to generate Regions of Interest (ROIs). After running a non-maximum suppression algorithm to eliminate overlapping ROIs, the ROIs are fed into an ensemble of CNNs, which classify each ROI as containing a digit 0 through 9, or containing no digit. Finally, a local search is performed around the area around detected numbers, to catch any digits that were not captured as an ROI.

2 Related Work

One of the earliest applications of CNNs to computer vision was in the area of digit recognition. The well known MNIST dataset, which is a large collection of centered, single channel, labeled images of handwritten digits often serves as a benchmark for many well known supervised [3] as well as unsupervised [6] object detection and classification algorithms. A very simple architecture, such as a single layer perceptron with a 10 way softmax output can achieve good performance on the MNIST data set, because of the relative homogeneity of the images. However, the goal of my final project was to accurately detect sequences of digits of arbitrary length. This is a more complicated task than a MNIST-like digit detection setting, because not only must each of the digits must be accurately classified, but their order in the sequence with respect to each other must be correct, and the total length of the sequence must be correct.

The most well known, and first solution to this problem was proposed by Goodfellow et al in [2]. This approach takes as input a full sequence of digits, and models the value of

the i^{th} digit as a discrete random variable S_i, represented by a 10 way softmax output. The net predicts a value for 5 digits for each image, and then models the length of the sequence with another discrete random variable L. The length of the sequence, and the value of each of the digits are all assumed to be independent, thus the probability of a sequence $\mathbf{s} = s_1, s_2, ..., s_n$ given input image X is the product

$$P(\mathbf{s}|X) = P(L = n|X) \prod_{i=1}^{n} P(S_i = s_i|x) \tag{1}$$

More recent state of the art architectures for object detection, such as Faster-RCNN [7] and YOLO, which have shown good performance on very large scale, multi-class object recognition tasks would also be more than satisfactory for accurately predicting bounding boxes around individual digits, but I decided to use a simpler approach.

3 Methods

3.1 Prepossessing and ROI Detection

The first stage of my pipeline uses a high precision, low recall algorithm to detect bounding box delimited ROIs. False positives are tolerable at this stage, since all ROIs will be passed to a more accurate, but slower classifier (the CNN ensemble) that should be able to detect these false positives. The most important properties of this algorithm are its speed and precision - as false negatives will immediately make correct classification impossible. This stage pares down the number of image cutouts that need to be passed to the CNN, as running the CNN over every image patch (including clearly uninteresting regions of the image, such as blue sky) would be inefficient. I initially experimented with an SVM classifier on top of HOG features, but found that the MSER (Maximally Stable Extremal Regions) algorithm performed better. I tuned the parameters of MSER to detect each individual digit as its own ROI, as opposed to Goodfellow et al, who treated the entire sequence of house numbers as a single ROI, and classified this ROI as a whole, rather than running inference on each digit separately. An image pyramid is used at this stage to improve robustness against digits appearing at various scales. MSER is applied three times to each image, once to the full sized image, and then again after resizing the image to half its previous size twice. The hyperparameters for MSER stay constant across these two resizes, so the largest bounding boxes with respect to the original size of the image are detected at the lowest level of the pyramid, when the image is smallest. The output of these three runs of MSER generally captured most of the interesting areas in the image, but created many closely overlapping bounding boxes. To deceases the amount of redundant computation required from the CNN, I used the NMS algorithm explained here[1] to eliminate overlapping bounding boxes. I used this step aggressively, eliminating any bounding box with greater than 15% overlap with another bounding box, keeping whichever bounding box had a lower right hand corner. In practice, I found that this rather arbitrary tie breaking strategy worked fine.

[1] https://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/

3.2 CNN Detection

3.2.1 Inference

Given a set of ROIs, I used an ensemble of 4 small CNNs, each with 3 convolutional layers, three fully connected layers, and an 11 way softmax output to classify each digit independently. Each of these four CNNs was trained on cutouts of individual digits augmented differently, the first CNN was trained on regular images, the second on horizontally flipped images, the third on vertically flipped images, and the fourth on horizontally and vertically flipped images. At test time, I applied each of these transformations to each ROI, and then passed each transformed version of the original cutout to each corresponding CNN. I then used a two part criteria for classifying a cutout as containing a digit d. First, for each of the four models, the d^{th} output of their softmax layer must be the largest. Second, I defined a confidence score for each prediction made by the CNNs as the difference between the maximum activation in the softmax layer, and the next largest activation. I required that the average of this confidence score across all 4 CNNs for a digit be greater than a threshold T. I arrived at $T = 2.85$ through trial and error.

3.2.2 Training

All of my training was done on individually cropped images of digits from the Street View House Number Dataset [6], plus a number of negative images that I got from taking about 20 large screenshots from Google Street View that did not contain any housing numbers, and then taking many small crops of these. My train set had 280K images, and my validation set had 80K images. In both my training and validation sets, there was approximately a 1:4 ratio of positive examples (meaning images containing some digit) to negative samples (images containing no digit). I observed that training data augmentation improved model performance, the brightness, saturation, and contrast were all randomly jittered for each image before being trained upon. The primary challenge of this project was the fact that our training was done on small cropped images, but our pipeline needed to function on large images that taken at varying distances from the digits we are trying to recognize. So keep in mind that any validation accuracy I report is on cropped images, and consequently does not capture the additional difficulty of getting accurate image crops using MSER.

Each of the four CNNs was trained for approximately 20 epochs using the same hyper parameters, a learning rate of 0.01, batch size of 32, and Nesterov momentum of 0.9 on a single NVIDIA Titan V GPU. I arrived at these parameters using a grid search. I chose to stop training after 20 epochs because the validation loss stopped improving. I used a focal loss function [5] for all of the models to help deal with class imbalance. This loss function gave a significant performance boost over simple Cross Entropy Loss when classifying real world images, as MSERs very low recall means that at test time, the batches of ROIs that the model needs to classify are highly imbalanced towards containing negative examples (all of the false positives from MSER). I experimented with fine-tuning a pre-trained VGG16, but ended up choosing the ensemble approach instead. I made this decision because running a forward pass through VGG16 for each ROI on my laptop CPU was too slow (for some images it took almost four minutes), whereas for my ensemble of small models, the longest inference time for a single image I observed was 2 seconds. The VGG16 model achieved higher validation set accuracy than my ensemble of small models (see Figure 2), so if it were not for the specification that our model needs to run quickly on a CPU I would have probably used VGG16 in my final solution, as VGG16 forward passes with a batch size of 100 on the GPU were nearly instantaneous and slightly more accurate.

3.2.3 Post Processing

After running inference on each ROI to obtain classifications, I used a fast local search in the area classified by the CNN ensemble as containing digits. The algorithm runs the CNN ensemble on several cutouts to the left of the leftmost digit identified by the CNNs, and to the right of the rightmost digit. If any of these bounding boxes are classified as positive, this digit is added to the final result, and area to the left or right of this newly classified box is recursively searched in the same way. This heuristic eases the precision required from the MSER first pass, as the pipeline has a second chance to classify digits that were missed by the first pass. In practice, this algorithm resulted in the successful classification of digits to the immediate left and right of correctly identified ROIs, but never more than a single digit away. Further tuning of hyper parameters used by this algorithm would likely result in better results for digits further away from the originally recognized ROIs.

4 Experiment

I collected a test set of approximately 30 images taken of various houses and businesses, some in person and some from Google Street View. These images contained substantial distracting shapes and objects. My full pipeline correctly detected the correct sequence of digits in about half of these. Figure 1 shows two examples of images that the pipeline classified correctly. In Figure 1b, each of the digits above the door was correctly and individually identified by MSER as a ROI. The CNN stage correctly classified each digit. In figure 1a, note that the final two digits of the number are both contained in the same ROI. Despite this, the digits are still classified correctly - the CNN stage of the pipeline detects a 0 in this ROI, and then the post processing local search step identifies a bounding box containing a 1.

Figure 2 displays two examples of images that were classified incorrectly by the CNN. In 1a, MSER does not capture the first digit as a ROI. The other three digits are classified correctly, and the local search identifies the missing leftmost digit, but it is classified incorrectly by the CNN as a 7 rather than a 1. In 1b, each of the digits is correctly identified as a ROI by MSER, but the CNN fails to detect the 1.

Figure 3a compares the validation set precision and recall of the VGG16 network, vs. the precision and recall averaged over all of the models in the ensemble. Throughout the training process, model checkpoints were saved every 2 epochs, the dotted red line at epoch 20 shows which checkpoint I used. As previously mentioned, the VGG16 network is noticeably more accurate than the ensemble average, but this accuracy comes at the cost of inference speed. Figure 3b compares the training loss between the two runs - I used all of the same hyper parameters across both runs for consistency. Since the VGG model is larger, it seems to over fit very early in the training, whereas the small model ensemble loss goes down more gradually. If I had decided to use VGG16 as my solution, I would have used a smaller learning rate.

Despite these two failures, the system is generally robust to:

- different lighting conditions, and random noise - Heavy image augmentation, mainly brightness and saturation jitter applied randomly during training handles this
- digits at different angles - Detected digits are sorted from left to right. If the standard deviation of the x coordinates of bounding boxes is less than 2, the digits are arranged vertically, (i.e. image 2b), and detected digits are sorted from top to bottom.
- digits at different locations in the image - MSER step handles this naturally, ROIs can appear anywhere in an image

- digits with different fonts - varying digit fonts within the training set make the CNN robust to this

5 Further Work

If I had more time to spend on this project, I would probably continue to experiment with more small model architectures to create more variation within my model ensemble. Digits being misclassified (i.e. mistaking a 1 for a 7) is due to the fact that the models all share the same architecture and are trained on the same data set. I would find a more diverse set of architectures, and likely use the additional images in the SVHN dataset the create variation in the training set between the members of my ensemble. I would also spend further time tuning my local search algorithm, so that only a single digit in a house number needs to be found for correct classification to happen.

References

[1] M. Donoser and H. Bischof. Efficient maximally stable extremal region (mser) tracking. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 553–560. Ieee, 2006.

[2] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[3] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[6] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[7] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

	Precision	Recall	Forward Pass Time[2]
VGG16	95%	96%	1.77s
MySmallNet	86%	87%	13.31s

Table 1: Validation performance statistics

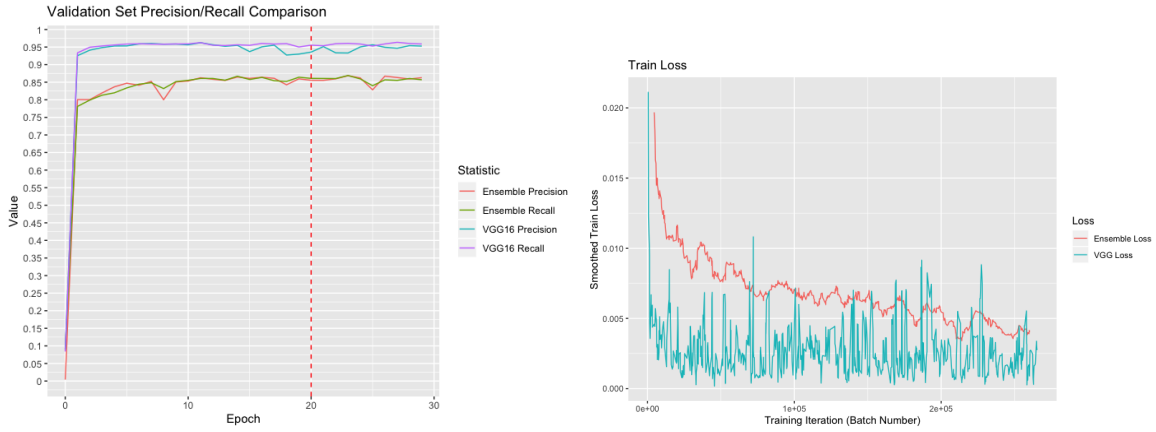
[2] Batch size of 100, 128x128 images, on Intel Core i7 2.6GHz CPU

(a) Output: 801 (b) Output: 260

Figure 1: Successful, different colors of bounding boxes correspond to ROIs identified at differing images scales

(a) Output: 7245 (b) Output: 362

Figure 2: Unsuccessful



(a) Precision/Recall Curves (b) Training Loss

Figure 3: Performance Stats, Ensemble Average vs. VGG16