Project: Classification and Detection with Convolutional Neural Networks

Xiaodong Tan (xtan74@gatech.edu)

Abstract

I build a digit classification and detection pipeline, which is a combination of a single digit classifier and a sliding window detection method. The classifier itself is inspired by the VGG16 architecture and achieves over 90% accuracy and 4% false positive rate. The final pipeline is able to correctly locate and and classify numbers in a variety of scenarios outside the original training dataset.

Introduction

A CNN classifier contains a sequence of layers that transform input image volumes to output volumes layer by layer and eventually into class scores. CNNs typically consist of convolutional, pooling, fully connected, activation, dropout and batch-normalization layers. A number of CNN architectures have been proven to have good performance in classification tasks. For example, GoogLeNet/Inception [1] uses an inception module with several small convolutions to reduce the number of parameters. VGGNet [2] uses homogenous architecture that performs 3*3 convolutions and 2*2 poolings. ResNet [3], the state-of-art CNN architecture skips some connections between layers, can be trained with up to a thousand layers and makes heavy use of batch normalization.

Two common object detection methods are based on sliding windows and connected components. Sliding window based methods slide a sub window through an image of multiple scales and apply a classifier on each sub-window. Connected component based methods, such as Maximally Stable Regions (MSERs) first separate object and non-object pixels by a fast low-level filter, then group the pixels with similar features[4]. MSER is more efficient than sliding-window method as it does not need to deal with a large number of sub windows[5].

More recently, a number of CNN based algorithms have been developed for object detections. R-CNN (region-based CNN) proposes regions using selective search algorithm and then applies a CNN on each region. To speed up the process, Fast R-CNN performs feature extraction before proposing regions and Faster R-CNN uses a region proposal network to replaced the searching algorithm. [6, 7]. YOLO (You Only Look Once) [8] and SSD (Single Shot Detector) [9] are state-of-art algorithms using regression. They apply a single neural network model to the whole image and then predict coordinates of a bounding box and their possibilities. Of note, there is a trade-off between accuracy and speed for these algorithms.

Dataset

The main dataset used for this project is the SVHN dataset which contains 32*32*3 images of house numbers. To enrich the dataset with negative examples background images are randomly cropped from the full images as 'non-digit' class. The digits and non-digits trainset are then randomly divided into 80% (85,250 images) for training and 20% for validation (21,314 images).

To evaluate the model's performance on images outside the SVHN dataset, a custom dataset is created from 7 images I collected in the wild the same way as images are processed in the final detection-classification pipeline. In total 18,928 patches of size 32*32 are extracted from the image pyramids of 7 images.

Of note, technically the distribution of different classes in the training set should be similar to that of the test set. In this project which uses a sliding window method, the majority of images fed into the model contain no digits. As only 18 digits are included in the 7 images for final testing, it is estimated less than 1% of these patches contain digits. However, as the proportion of non-digits are only ⅓ in the training set, the model might tend to label non-digit as digit, leading to a high false positive rate.

Image preprocessing

In order to facilitate the training process, the images are centered by subtracting the mean and scaled by dividing by 255. To combat overfitting each training example is augmented using random shift, shear, zoom and channel shift transformation. To deal with the invariance requirements of the project, the training examples are further augmented with random rotations, random brightness changes and the addition of gaussian noise.

Training a classifier

Model Architecture

The custom models have a similar structure as VGG, which is composed of repeated blocks of convolutional layers followed by a max pooling layer. The number of filters, number of CNN blocks and units of dense layers are set to be smaller as the input image is only 32 * 32 pixels. The top dense layer has 11 units for the 11 classes. Experiments are done mainly on the depths of the models, that is, different combination of the number of CNN blocks (3 or 4) and number of dense layers on top of them (1 or 2). The deepest model has the following architecture.

	Conv layer (relu activation)			MaxPooling (stride 2*2)
	No. of layers	No. of filters	Filter size	
Block 1	2	32	4*4	3*3
Block 2	2	64	4*4	3*3
Block 3	3	128	3*3	2*2
Block 4	3	160	3*3	2*2
Flatten layer				
Dense Layer 1 (128 units with Relu activation)				
Dense Layer 2 (128 units with Relu activation)				
Dense layer (11 units, softmax activation)				

Batch normalization and regularization (dropout/L2) are not included in the architecture. I also stick to relu activation for the convolutional layers and the dense layers (except for the last one). Batch norm/regularization/other activation function (such as leakyRelu) would only be explored if the current architectures do not have good performance or have overfitting issues.

The performance of a VGG16 model provided by the *Keras* library is compared with the performance of the custom model. To this end the top dense layer is replaced by a dense layer with 11 units, to match the number of classes for this project. Three model variations are explored. 1) all the weights except for the ones in the top layer are fixed. The input size is kept as the original 224 * 224 as the weights were trained on the images with that resolution; 2) the same as 1) but all the weights are retained; 3) same as 2) but the input size is changed from 224 * 224 to 32 * 32.

Training

The loss function measures the difference between the prediction and the ground truth label. As this project is a multi-class classification problem, cross-entropy loss is chosen. It measures the distance between the empirical data distribution and the model predictive distribution. The latter is obtained by transforming the output of the top layer with a softmax function.

Training the model is then done by minimizing the loss over a given dataset. For neural networks this is typically done with gradient descent. In gradient descent the gradient of the loss function is used to find the direction of parameter adjustment. In stochastic gradient descent (SGD) only a subset of examples (a minibatch) is used to calculate the gradient at each step. To provide more stable estimates of the gradient a technique called "momentum" is often employed. The factor that scales values of the parameter updates, the learning rate, is an important hyperparameter. If it is too big, it might overstep and give a bigger instead of smaller loss. If it is very big, the loss might fail to converge. If it is too small, the loss might converge too slowly. In this project I tried both SGD and Adam[10] optimizers with various learning rates.

For a project with a large dataset, on one hand it is not possible to feed data into the memory all at once; on the other hand, it is inefficient to update the parameters only when the whole loss is calculated. I used a batch size of 128 as a trade-off between training speed and memory limit.

To make the training process more efficient and to reduce overfitting we use "early stopping". The training process is stopped if the validation loss does not improve for five passes over the entire dataset (epochs). The model with the best validation loss is saved as the final model.

The weights are initialized using Keras' default initializer, which is glorot_uniform[11].

Evaluation metrics

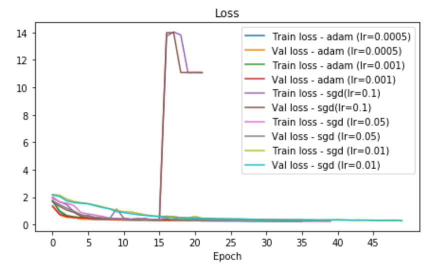
In the training, validation and test set created from the SVHN dataset, accuracy is used as evaluation metrics. This metric is potentially problematic here because the distribution of different classes (0-9 and non-digits) is not even and non-digits account for around one third of the dataset. Even if an algorithm predicts all the images as 'non-digit', it still has an accuracy (⅓) much higher than random guess (1/11).

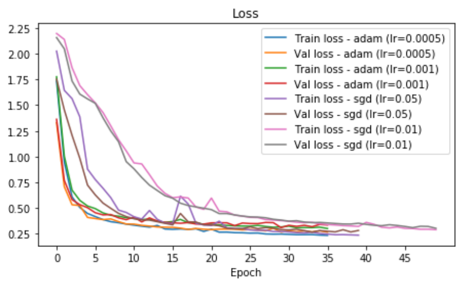
When testing the model on the custom dataset, as the labels are not available, the percentage of digits predicted by the model is used as an indication of false positive rate to evaluate the model. The ground-truth positive rate is estimated to be less than 1% as discussed above.

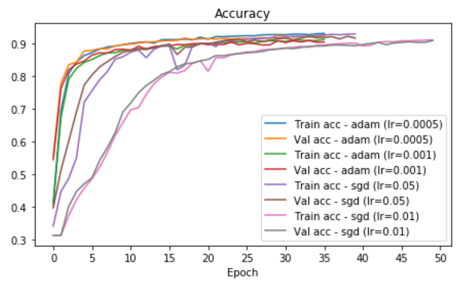
Performance of the classifier

Training model with different parameters

When using sgd optimizer, with a very high learning rate (0.1), the loss stops to converge and suddenly jumps to over 10 after a number of epochs. When the learning rate is very low (0.01), the loss converges very slowly (more than 50 epochs) . The Adam optimizer with 0.001 and 0.0005 learning rate performs better than SGD with a learning rate of 0.05 . The loss drop quickly and the accuracy reaches over 90% in 20 - 30 epochs.







Different model architectures

Except for the VGG-16 (fixed weights) model, all the other models reach over 90% accuracy on training, validation, test sets (non-digits, digits), as well as similar positive rate in the custom dataset. The model with 4 CNN blocks and 2 dense layer has the best result among the custom models. No overfitting or strong underfitting is observed so other activation function, regularization or batch normalization were not tried.

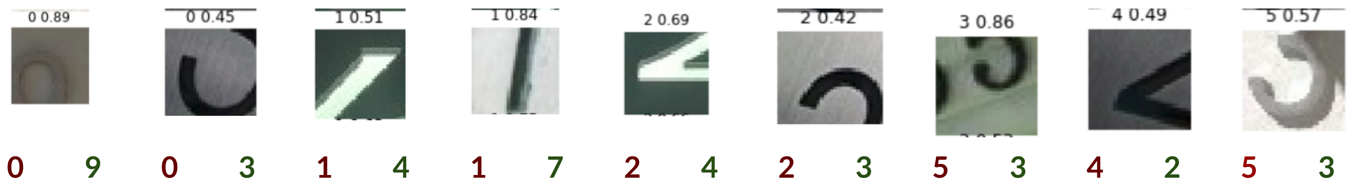
The VGG-16 model with pre-trained weights, however, has a poor performance in classifying a given digit (28.5%). Although it has a low false positive rate (1.1%) in the custom set, analysis on the images reveal that it is because the model lacks the ability to identify a digit. This is perhaps because the features that make up digits are not very useful for the imagenet classification task of natural images.

Model 4 is chosen as the final model as its performance on the test set and custom dataset is similar to that of the re-trained VGG but the model is much smaller.

Accuracy	Training	Validation	Test (non-digits)	Test (digits)	Custom %positive
Model 1 (3 blocks + 1 dense layer)	92.4%	91.6%	94.9%	93.6%	4.7%
Model 2 (3 blocks + 2 dense layers)	93.5%	92.5%	95.9%	94.9%	4.8%
Model 3 (4 blocks + 1 dense layer)	92.7%	91.9%	95.5%	93.9%	4.6%
Model 4 (4 blocks + 2 dense layers)	93.6%	92.6%	96.1%	95.0%	4.8%
VGG-16 (retrain) input: 32 * 32	96.1%	94.1%	96.5%	95.1%	4.8%
VGG-16 (retrain) input: 224 * 224	95.7%	94.4%	95.7%	96.1%	4.6%
VGG-16 (pre-trained weights)	55.1%	55.0%	91.2%	28.5%	1.1%

Analysis of the positive results

An image might be falsely classified when part of a digit looks similar to a whole or part of another digit. For example, part of 2, 4, 6, 7, 8, 9 may all be classified as 1; part of 6, 8 and 9 may be classified as 0; 2 and 3 have similar upper part while 3 and 5 have similar lower part. Some of the examples are shown below (red number: the label predicted; green number: ground truth).



In other cases, a background image without a digit in the middle is labeled as a digit. It might because some edges in the image are labeled as digits (the image 1 below) or the image is very noisy so some possible shapes are detected (the other images below).

If the predicted possibility is high, it is usually very accurate at predicting the digit in the middle of the image. Images predicted with over 98% possibilities in the whole dataset are shown as follows.

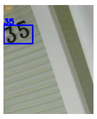
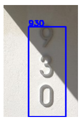
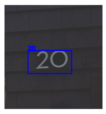
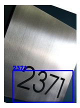
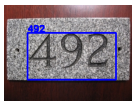
Digit identification pipeline

Step 1: for a given image, an image pyramid is created. The smallest image should be equal to or bigger than the sliding window (32*32). Then slide the window across each resized image to get a patch. The step size is chosen to be 4. If it is too small, too many patches will be created, which makes the process slower. If it is too big, the patch with a digit might be skipped.

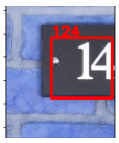
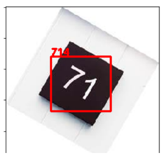
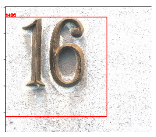
Step 2, each patch obtained above is preprocessed with mean subtraction and rescaling. If patch is predicted as a digit by the classifier with a possibility of over 98%, it is concluded that the patch contains a digit and the location of the patch inside the resized image is resized to the location in the original image. The high 98% possibility is chosen to deal with the false positive issues discussed in the previous section.

By repeating step 1 – 2, a list of digits, their predicted locations in the original image and possibilities are obtained. For each digit, their locations and possibilities are fed into a non-maximum suppression function to get a single location with highest possibility. In the end, a single big bounding box is calculated based on the best predictions of locations of each digit.

The pipeline successfully detects and identifies the digits in the following images with different location, scale, light, orientation and background situation.

 Digits are small, rotated and at the corner	 Digits are in the different light condition and different locations	 Digits are small in a dark image	 Digits are rotated and in one edge of the image.	 Digits are big and with a noisy background

The pipeline does not work well on some images, in which 1) part of a digit is detected as another digit with a high possibility 2) the background is too noisy and random shape is detected.

	Labeled as 124 An extra 2 is detected during to the similarity between 2 and 4 at the edge		Labeled as 714 An extra 4 is detected due to the edges in the image		Labeled as 1406 The background of the image seems to have other shapes in between 1 and 6

The pipeline is also used to track digits in a video. Some frames are as follows and the full video can be found at https://www.dropbox.com/s/keja6j4wntyuwgs/digit_detection.avi?dl=0

Discussion

Compared with the original VGG-16 model, the smaller VGG-style model I implemented achieves similarly good performance. This is perhaps because the resolution of the images in the training set is relatively low and a deeper network with more parameters does not necessarily extract more information from the data. In fact, after a certain number of downsampling, the output size is already 1*1 and more convolutional layers and pooling layers do not make a difference.

In this project only VGG is explored. As the state-of-art classification CNN architecture is Resnet which uses back normalization and skips some connections, a Resnet-style smaller network might achieve better results.

Compared with the state-of-art methods of object detection, the sliding window method used in this project is very slow as the whole image need to be slided over several times and all the sub windows need to be examined. A better region proposal method would significantly increase the speed. It can be either MSER, or even another neural network as used in Faster R-CNN.

My algorithm also suffers from false positive issues. In particular, if part of a digit is in a image, it might be classified as a wrong digit instead of a non-digit. This is related to the composition of the training set, in which the non-digits images only include background images, not partial digits. A possible improvement is to include partial digits and more a larger variety of background images in the training.

In this project, I get more familiar with different cnn models and object detection techniques and I find techniques such as non-maximum suppression very interesting. One difficulty is that some digits and digit parts are very difficult to distinguish even for human beings, such as the upper part of 9, the lower part of 6 and 0.

A final presentation of the project is available at https://www.dropbox.com/s/kiysauzfxzl0bkg/presentation.webm?dl=0

References

[1] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. *Going Deeper with Convolution*. arXiv:1409.4842 [cs.CV] 2014.
[2] Karen Simonyan and Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556 [cs.CV]. 2014
[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs.CV], 2015.
[4] L. Neumann and J. Matas. *A method for text localization and recognition in real-world images*. In ACCV, 2010.
[5] Weiling Huang, Yu Qiao, Xiaoou Tang: *Robust Scene Text Detection with Convolutional Neural Network Induced MSER Trees*. ECCV 2014, Part IV, LNCS 8692, pp. 497–511, 2014.
[6] Ross Girshick. *Fast R-CNN*. In International Conference on Computer Vision ({ICCV}), 2015
[7] Shaoqing Ren and Kaiming He and Ross Girshick and Jian Sun. *Faster {R-CNN}: Towards Real-Time Object Detection with Region Proposal Networks*. Advances in Neural Information Processing Systems ({NIPS}), 2015.
[8] Redmon, Joseph and Farhadi, Ali, *YOLO9000: Better, Faster, Stronger*. arXiv preprint arXiv:1612.08242},2016}
[9] Wei Liu etc. SSD: Single Shot MultiBox Detector. ECCV 2016.
[10] Diederik P. Kingma, Jimmy Ba. Adam: *A Method for Stochastic Optimization*. 3rd International Conference for Learning Representations, San Diego, 2015.
[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010.

- Python library: *Keras*
  https://keras.io
- Stanford CS231n: *Convolutional Neural Networks for Visual Recognition*
  http://cs231n.github.io
- Coursera course: *Deep Learning*
  https://www.coursera.org/specializations/deep-learning
- Blog Post: Joyce Xu, *Deep Learning for Object Detection: A Comprehensive Review*
  https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9
- Blog Post: *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD*
  http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd