CSC 2515 Project: SVHN

Didier Landry
Department of Electrical and Computer Engineering
University of Toronto
`didier.landry@mail.utoronto.ca`

Dustin Kut Moy Cheung
Department of Electrical and Computer Engineering
University of Toronto
`dustin.kutmoycheung@mail.utoronto.ca`

Abstract

Detecting and reading text from photographs is a challenging computer vision problem that has garnered a lot of work in recent years. Being able to accurately localize, and recognize arbitrary digits from natural images is hard due to the complex scenes in those images. In this report, we apply convolutional neural networks to learn a unique set of features optimized for this task, and discuss the evolution of our neural network topology to achieve a high accuracy on the validation set. We use over 500,000 labeled digits obtained from the SVHN[4] dataset for training. Moreover, we also describe our attempt to localize digits from unconstrained images by using image processing techniques and unsupervised learning.

1 Background

Convolutional neural networks citation needed are biologically-inspired neural networks that uses identical copies of the same neuron for training, allowing the network to have a large number of units while keeping the number of parameters of those neurons small. The individual neurons are generally connected to filtered overlapping regions of the image. Those convolutional layers are connected to pooling layers. One popular choice for this kind of layer is the max-pooling layer, which extracts the maximum of features over small blocks of the previous layer.

Generally 2D convolutional neural networks are used in computer vision to learn features for extraction information. The 2D layer will look at patches of images to generate features such as the detection of the presence of an edge, or a particular texture.

The LeNet models are a popular family of models used in computer vision. They consist of multiple layers of convolutional layers and max-pooling, followed by fully-connected neural networks. We will be using one of those LeNet network for our training.

Recently, convolutional neural networks have been applied with dropout training in computer vision. When training deep neural networks with a small training set, the training generally leads to overfitting. One method to fix this is to simply stop training when the performance on the validation set starts to get worse. Another popular technique is to use L2 regularization on the neurons weights. Dropout is yet another technique used to prevent overfitting by randomly dropping out units in a neural network.

The Street View House Numbers(SVHN)[4] dataset is a popular training set consisting of Street View images cropped to either show single digits (Format 2) and multiple digits (Format 1). Those images are extracted using a combination of automated algorithms and Amazon Mechanical Turk (AMT) framework.

2 Problem Description

The project consists of classifying digits from street view images. The SVHN dataset will be used for training. All the images in the Format 2 dataset have a fixed 32x32 resolution with a digit centered at the image. There are ten classes in total. The images show vast intra-class variations due to image distortions that happen in natural scene pictures. Factors that cause those image distortions include lighting, shadows, motion, and focus blurs.

The Format 1 dataset consists of images with different resolutions with multiple digits in each image. The images are not well cropped and not centered. The task is to first detect the digits with a bounding box, then classify each digit. The images in the Format 1 dataset also show the same image variations as in the Format2 dataset.

To facilitate the process of implementing and debugging the digit recognition, the Format 2 data was used to test the digit recognition neural network. The digit segmentation was treated as a different problem because it (obviously) degrades the quality of the input images fed to the neural network. However, the digit segmentation algorithm produces 32x32 images that could, then, be fed to the neural network for recognition.

3 Process

3.1 Digit Recognition

3.1.1 Frameworks Used

Throughout the project, we used the popular Theano project[3], together with Lasagne[1], numpy, scipy[5], and NoLearn[2] libraries. Most of our experiments are executed using an EC2 machine with a GPU to exploit the speed gains of training our Neural Network with the CUDA framework.

3.1.2 Basic Neural Network

Our first approach is to train a basic Neural Network with two hidden layers with the grayscale images of the train dataset. The nonlinearity activation function used for our hidden layers is a Rectifier, whereas the output layer uses a softmax layer. The grayscale images are normalized before being used as input. The output layer consists of 10 units, each representing a particular digit. The guessed digit from the Neural Network is determined by the unit with the largest value.

3.1.3 Convolutional Neural Network (CNN)

Another more successful approach that we attempted is to add convolutional neural networks to our basic neural network. We added 2 convolutional and max-pooling layers in between the inputs and the ReLu hidden layers.

3.1.4 Improvement to CNN

One of the issues that can be experienced while using deep neural networks is overfitting, where the neural networks overtrain on the training set, while the validation set gets worse. One of the methods to rectify overlearning is by the use of Dropout layers, with a dropout probability of 0.5.

Another improvement to our Convolutional Neural Network is to employ pre-processing on our images to boost some features that could help our network train faster and better. One of the popular techniques is the Local Contrast Normalization, which is applied to each of the three color channels of the images. It has been used extensively in Neural Networks and huge improvements on the training part has been observed.

3.1.5 Augmenting the training set

Another way to decrease overfitting in our neural network is to simply augment our training set. To achieve this, we use the extra set (500k images) available from the SVHN dataset to augment our initial training set of 73k images. Another way that we could have augmented our dataset would be to add noise and slightly rotate our existing training set to produce more data.

3.2 Image Segmentation

The unsegmented data set contains images of houses address plates at different levels of zoom of zoom and isolation. Moreover, the number of digits that a given image contains is also unknown. Consequently, separating each individual digit is far from trivial.

One approach could have been to randomly take squares from the image and train a classifier to decide whether it contains a digit or not. This could have worked rather well if the goal was just to find digits in an image. However, for segmentation (one image per digit, centered), its performance would have been random at best. Since, to properly decode an address, the information contained in the number of digits and their order is as important as the digits themselves, this approach was rejected.

Accordingly, here, we wrote an algorithm loosely inspired on MATLAB ocr example. It can be found attached to this document.

First, the images are preprocessed: they are resized to a height of 200 pixels while keeping the aspect ratio the same and converted to grayscale.

The second step is to find the areas of the image that have a high probability of containing text (an approach similar to that of A. Coates et. al.s detection). For that, we apply a sobel vertical edge detection algorithm and, then, morphological closing on the found edges (very scattered) to create connected regions. This worked well because the vertical boundaries in numbers tend to create a rectangular blob of points that, when morphologically closed, completely contains the digit (Figs.2 & 3). Vertical edges such as the wall, in this example, get filtered easily because, after closing, they create really narrow regions.

Depending on the image, this preliminary analysis can produce a lot of possibilities. To solve this problem, the candidate connected zones are filtered by size, aspect ratio and solidity. Areas that are too narrow, fat or have a low solidity (level of vertical gradients in a box containing the connected zone) are discarded. The zone that has the greatest probability of containing a digit is then passed to the next step.

The third step is to detect the number of digits present in the previously found area. For that, the image is turned into binary (black and white) using Otsus [thresholding] method . Then, the after inverting the image if the background was white, the number of white pixels in each column (assumed to be the digit) is counted. If it falls below a threshold (that is a function of the maximum number of white pixels in the images column), a split point is saved (see Figs. 4 & 5).

Finally, the split zones are, again, filtered by width and cropped into 32x32 pixels images (extending laterally to get a 1:1 aspect ratio, if necessary) (Figs. 5 & 6).

4 Experimental Results

4.1 Digit Recognition

4.2 Image Segmentation

5 Discussion and Conclusion

5.1 Scaling with Millions of Examples

Scaling our current convolutional neural network to millions of examples is incredibly challenging. Luckily, in lasagne, a lot of optimizations happen in the background. For instance, the batch size is selected automatically depending on the number of examples.

Even though Lasagne and Theano ultimately produces CUDA code that can be run on a GPU, there are further optimizations that could be achieved by using a pure CUDA approach and getting rid of Python dependencies along the way.

One way to scale our training is to simply use a distributed neural network spanning across several machines. At this amount of data, the overhead involved with communication between the different machines become negligible compared to the speed gain in training in parallel. An example of such a distributed network is DistBelief, developed at Google.

Another way is to simply decrease the pixels of each image used to a smaller size. By having less pixels to train on, the training speed will increase, albeit at the expense of losing some information.

5.2 Digit Recognition

Scaling our current convolutional neural network to millions of examples is incredibly challenging. Luckily, in lasagne, a lot of optimizations happen in the background. For instance, the batch size is selected automatically depending on the number of examples.

Even though Lasagne and Theano ultimately produces CUDA code that can be run on a GPU, there are further optimizations that could be achieved by using a pure CUDA approach and getting rid of Python dependencies along the way.

One way to scale our training is to simply use a distributed neural network spanning across several machines. At this amount of data, the overhead involved with communication between the different machines become negligible compared to the speed gain in training in parallel. An example of such a distributed network is DistBelief, developed at Google.

Another way is to simply decrease the pixels of each image used to a smaller size. By having less pixels to train on, the training speed will increase, albeit at the expense of losing some information.

5.3 Image Segmentation

The image segmentation algorithm could be improved to allow for more than one zone of high probability to contain a digit. Currently, if the numbers of the address are too far apart in the image, they get boxed separately and, in the end, only one number gets isolated properly. This is because, after the morphological closing of the vertical gradients, all the numbers are assumed to be in the same connected zone (this was a simplifying assumption that proved to be correct in most cases).

It would also be interesting to continue to investigate to find machine learning techniques that could exploit some information that comes with the Format 1 images; the boxes. Indeed, each image comes with the location and size of boxes that surround the each digit. Perhaps, using the number of boxes, a classifier could be trained in a supervised manner to find the number of digits in each image.

6 Citations, figures, tables, references

These instructions apply to everyone, regardless of the formatter being used.

6.1 Citations within the text

Citations within the text should be numbered consecutively. The corresponding number is to appear enclosed in square brackets, such as [1] or [2]-[5]. The corresponding references are to be listed in the same order at the end of the paper, in the **References** section. (Note: the standard BibTeX style `unsrt` produces this.) As to the format of the references themselves, any style is acceptable as long as it is used consistently.

As submission is double blind, refer to your own published work in the third person. That is, use "In the previous work of Jones et al. [4]", not "In our previous work [4]". If you cite your other papers that are not widely available (e.g. a journal paper under review), use anonymous author names in the citation, e.g. an author of the form "A. Anonymous".

6.2 Footnotes

Indicate footnotes with a number[1] in the text. Place the footnotes at the bottom of the page on which they appear. Precede the footnote with a horizontal rule of 2 inches (12 picas).[2]

6.3 Figures

All artwork must be neat, clean, and legible. Lines should be dark enough for purposes of reproduction; art work should not be hand-drawn. The figure number and caption always appear after the figure. Place one line space before the figure caption, and one line space after the figure. The figure caption is lower case (except for first word and proper nouns); figures are numbered consecutively.

Make sure the figure caption does not get separated from the figure. Leave sufficient space to avoid splitting the figure and figure caption.

You may use color figures. However, it is best for the figure captions and the paper body to make sense if the paper is printed either in black/white or in color.

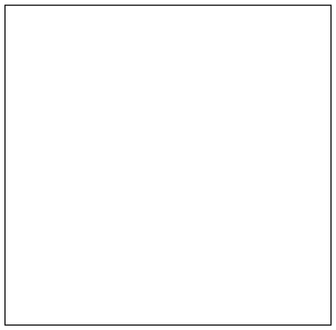

Figure 1: Sample figure caption.

6.4 Tables

All tables must be centered, neat, clean and legible. Do not use hand-drawn tables. The table number and title always appear before the table. See Table 1.

Place one line space before the table title, one line space after the table title, and one line space after the table. The table title must be lower case (except for first word and proper nouns); tables are numbered consecutively.

References

[1] Neural network tools for theano. `https://github.com/Lasagne/Lasagne`.

[1]Sample of the first footnote
[2]Sample of the second footnote

Table 1: Sample table title

PART	DESCRIPTION
Dendrite	Input terminal
Axon	Output terminal
Soma	Cell body (contains cell nucleus)

[2] scikit-learn compatible wrappers for neural net libraries. `http://pythonhosted.org/nolearn/`.

[3] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[4] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5. Granada, Spain, 2011.

[5] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.