EE554/CSE586 Spring 2016 Project 2 Due Date: Fri Apr 29, 11:59PM

We have discussed convolutional neural networks in class. Now it is time for some practical experience. We will train a CNN to recognize the Chinese characters for numbers 1,2,3,...,10 .

Nobody writes CNNs from scratch, although I hope you agree that with what we have learned in class you **could** write and train a simple CNN composed of the computational modules we have discussed. It would run slowly however, unless spent a lot of effort vectorizing and optimizing the code for efficiency. Luckily, existing CNN libraries have done all that optimization work already! We will use matconvnet, written by the by the same people who developed vlfeat, because it plays well with matlab and thus can be used for rapid proof-of-concept prototyping.

1. Download matconvnet from http://www.vlfeat.org/matconvnet/ . Alternatively, it is also available as part of the download package for the Oxford VGG CNN practical (tutorial) available at http://www.robots.ox.ac.uk/~vgg/practicals/cnn/ .

2. Since matconvnet does not come with precompiled binaries, you will need to compile it for your machine. Unfortunately, how to perform this step depends on your operating system and your version of matlab. If you have a recent OS and version of matlab, and if your matlab is configured to use a good compiler, this may be as simple as running vl_compilenn in matlab, following the instructions at http://www.vlfeat.org/matconvnet/install/ . I have an old mac OS, and old version of matlab, and had to compile from the command line using the instructions at http://www.vlfeat.org/matconvnet/install-alt/ . Since this step of compiling and setting up matconvnet may involve system tinkering and some trial and error, try to do it right away – don't wait till the last moment to find that you have no idea how to install matconvnet. To make the process more manageable, I have opened a discussion forum on our Angel course site, to exchange tips and tricks for getting matconvnet installed on various machine configurations. If we are all willing to help each other, it is likely that we only need to compile for a handful of common OS/matlab configurations, and in the worst case perhaps the more tech savvy folks can share their compiled binaries with other groups as needed.

3. Although not strictly necessary, I would strongly suggest stepping through the tutorial examples from the Oxford CNN practical. It gives a good introduction to how the code is used. Also, you might consider downloading and trying the pretrained object recognition CNNs from the "Quick Start" guide at http://www.vlfeat.org/matconvnet/quick/ . One of them is VGG and one is Googlenet – both were extensively trained using the imagenet database to recognize 1000 different object categories. Is it very entertaining to play with these using your own photographs.

4. I have download 156 samples of different Chinese font styles from the website http://www.freechinesefont.com. These samples are stored as png files. I have placed them in a zip file on our course web site, along with sample matlab code to read them in and to use their stored color maps to convert to greyscale floating point images. After you run my code you will have a 3D array of size 407x305x156, containing single float values between 0 and 1.

5. Each sample image contains the same characters in the same order, as far as I can tell. Part of your job is to locate and crop out the numeric characters and manipulate them into a training set with 10 classes corresponding to 1,2,3,4,5,6,7,8,9,10. The mean image across the dataset is shown below, and is annotated with which line contains the characters for numbers. The image on the right shows some samples cropped from the dataset images.

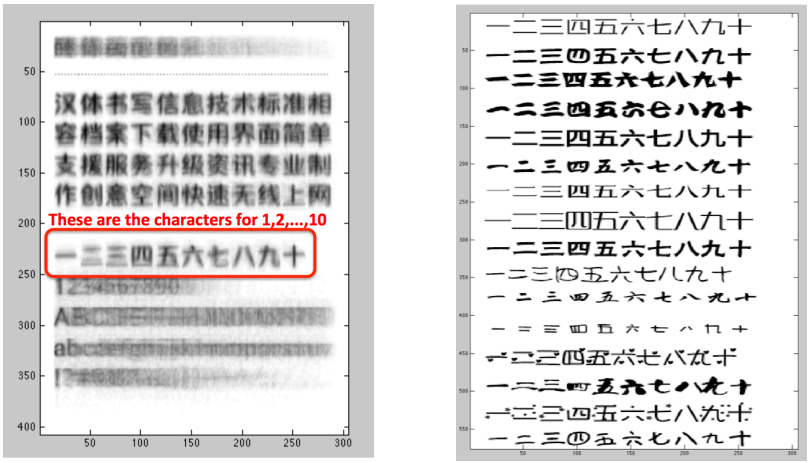

Figure 1. Left: mean of the 156 font images, with the characters for numbers indicated in red. Right: some samples cropped from the dataset, illustrating some of the variability in fonts.

Since the characters in the mean image look pretty clear, one might be tempted to choose absolute row,col values for cropping each character across all 156 images. That might work reasonably well. On the other hand, further inspection of sample characters in the image to the right shows that not all of the font styles line up in neat columns, so you may need local adjustments for cropping the characters for some of the fonts. The ultimate success of your classifier is going to be due in part to how clean you can make your training data, in terms of well-centered and well-aligned characters. At the end of this step, you should have 156 small character images corresponding to class "1", 156 corresponding to class "2", and so on, for a total of 1560 images in your dataset.

6. Randomly split your dataset into training and validation sets. Maybe a 60-40 split between training and testing? It's up to you.

7. Build and train a CNN for recognizing these characters. Exercise 4 from the VGG CNN practical would be a very good guide to follow for doing this. It is trying to handle a similar task, namely, learning how to recognize western fonts for characters a,b,...,z. If you follow this approach, you will need to massage your dataset into conformance with the imdb data structure that they use. You can also try to use their CNN configuration, which should work reasonably well since this is a similar problem. However, I'd like you to also explore other configurations modules for your CNN, in terms of number of conv filters and layers, when to do relu, when to do maxpool, etc. This is a place where you can get creative. Smaller CNNs will train much faster, larger CNNs will tend to have more accuracy. Play around a little to get a feel for what works and what doesn't. Finally, because of the relatively small number of training examples (with respect to degrees of freedom of the CNN learning problem), you very likely want to augment the data by at least "jittering" the samples. Again, refer to exercise 4 to see how they do it.

8. Test your classifier on some test examples and discuss how well it works, or why it might not be working. You could try handwriting and scanning in some of these characters yourself, to see if it can recognize them. You also could try finding some images on the web that contain Chinese number characters. One example below is from a set of Mahjong tiles.

Keep in mind that you have learned how to recognize characters at only a single scale. You will undoubtedly need to resize any images that you generate or find to have characters at the correct scale for your classifier.

What to hand in: Upload in the Angel dropbox a zip file containing your cropped character dataset, your code, and a pdf of a written report that explains what you did and what your results are.

I'm not planning on running your code, and it doesn't have to run as an end to end demo, but I do want to look through it so make it clear by documentation and program structure where the main computations for each of the steps are happening.

Your report, which counts as equal weight with the code towards the grade, should contain at least the following:

a) Summarize *in your own words* what the project was about, what tasks you performed and what results you expected to achieve;

b) Explain any design decisions you had to make. How did you decide how to crop and align the characters to make your dataset? What training-validation split did you use. What is the configuration of your CNN (how many layers, what modules comprise each layer, how many parameters are being learned at each layer). What other configurations did you try? What worked best (are there any rules of thumb you observed on what works and what doesn't)? What kind of data augmentation / jittering did you use? What is your overall accuracy on the validation set? The cnn_train program generates a nice plot showing training and validation set error decreasing over time, show that. Present your testing data, your classifier's results on that data, and discuss.

c) Document what each team member did to contribute to the project (if you are working in a team). It is OK if you divide up the labor into different tasks, and it is OK if not everyone contributes precisely equal amounts of time, as long as the effort split isn't terribly imbalanced.