1 Example: photo OCR

In the last section, we will use photo OCR as an example to illustrate how a complex machine learning is put together, and also to explain the concept of machine learning pipeline.

1.1 Problem description

Photo OCR (abbr. for optical character recognization) is the process to recognize characters in a photo with machine learning algorithms. A basic photo OCR process includes the following steps:

Text detection Pick out rectangle regions in the photo where characters appear.

Character segmentation Divide each region found in the previous step to small rectangle regions that each contain one single character.

Character classification Classifier that recognizes each small region as the character it contains.

By dividing the task of photo OCR into a few steps as shown above, we have built the pipeline of this problem. In a pipeline, all or some of the steps may involve machine learning. The division into separate steps facilitates the split of work load among groups of engineers.

1.2 Sliding windows

The method used to solve the text detection problem is called sliding windows. A simpler example to illustrate the ides is pedestrian detection.

Regions containing pedestrian have similar length-width ratios. Here we will use the ratio of 82*36. We will collect a series of image patches of this size that contain ($y = 1$) or do not contain ($y = 0$) pedestrians, and use them as examples to train a neural network as the classifier. Then we will silde a 82*36 window, at some step size, along the width and length of the photo and use the trained classifier to detect whether the region inside the window contains a pedestrian or not. If we have a 1000*1000 photo and use 2 as the step size, totally 460×460 windows of size 82*36 will be checked ($\frac{1000-82}{2}+1 = 460$). Of course pedestrians could be of other size, so the window will be resized, say to 164*72, and another sliding windows process will be carried out. In the end, hopefully we will pick out all regions that contain a pedestrian.

Text detection works in a similar way, except that regions containing text are not of the same length-width ratio. Thus besides resizing the window, we must also try different length-width ratios.

The character segementation problem can also be solved by sliding windows. Instead of training a classifier that tells whether an image patch contains texts, we will train a classifier that tells whether an image patch contains the separation of characters. When such regions are detected, we will draw lines in

their middle to split the regions picked out in the text detection step into small regions that each contains only one character.

Finally, the character classification problem can be solved by one of the machine learning algorithms we have introduced, e.g. neural network.

1.3 Artificial data synthesis

Large amount of training data may in general improve the performance of a machine learning algorithm. For some machine learning problems, character recognization being one example, it is possible to produce large amount of synthetic data and use it in the training process, which will save us a lot of time and effort to collect real data.

For the character recognization problem, one way to synthesize data is to take characters from different fonts and put them against random backgrounds. Another way is to distort an existing image patch containing a character and use the distorted image patch as a new example.

Another problem in which artificial data synthesis may help is speech recognization. From a real piece of audio containing some content to be recognized, we can add different noisy backgrounds, such as photo connection sound, crowd of people talking, sound of machinery, etc, to obtain synthetic audio that can be used as new training examples.

Note that synthetic data do not always helps. In the character recognization case, rather than distort the original image patches, if we add random noises on pixels inside the image patch, the new examples will not help to improve the accuracy of the classifier.

Before expending the effort to get more data, it is indispensable to make sure that the problem has low bias. Afterwards, it is worthwhile to ask ourselves "How much work would it require to get $10\times$ as much data as we currently have, with by collecting real data or synthesizing artificial data?" The actual amount of work may often be quite small, while the extra data obtained will significantly improve the performance of our algorithm.

1.4 Ceiling analysis

Time of the engineers working on the machine learning problem is the most precious resource we have. It would be indubitably helpful if we could figure out which part of the pipeline is the most promising to provide significant improvement. Ceiling analysis is the tool that gives us a clue on this.

Take the photo OCR pipeline as an example. Suppose we currently have an overall accuracy of 72%. Ceiling analysis will now be carried out to decide which step of the pipeline is the worthiest of working on for improvement of accuracy.

Imagine that we substitute the text detection unit of our pipeline with a perfect one that has 100% accuracy, and leave the rest of the pipeline as it is. This could be achieved by providing manually obtained text detection results, which are 100% correct, for the subsequent character segmentation step. Now

the system has an overall accuracy of 89%. We continue to do the same thing to the character segmentation step and the character recognization step, and finally end up with a system that features 100% accuracy. The procedure is shown in Table 1.

Table 1: Ceiling analysis of photo OCR pipeline

Component	Accuracy
Overall system	72%
Text detection	89%
Character segmentation	90%
Character recognization	100%

From the analysis result, it would be worthwhile to devote the effort to improving the accuracy of text detection, while it won't help much to build a better character segmentation unit. Ceiling analysis has provided us with a guideline on judicious allocation of our development resources.