Letter Localization

Markus Köhler
University of Konstanz
Konstanz, Germany
markus.koehler@uni-konstanz.de

***Abstract*—The abstract of this paper.**

***Index Terms*—letter, localization, SVM, HOG, IoU**

I. Introduction

Computer-printed letters can be found everywhere in our cities, e.g. on traffic signs, stores and advertisement posters. The variety of those letters is huge. Although the same letter can appear in different sizes, colors, fonts, text styles and so on, humans are usually very good at locating and classifying them.

A. Goal

This paper is dedicated to the question how to locate these letters on any input image using Machine Learning. We do not care about classifying these letters afterwards. To keep it more simple, we only want to detect the 62 characters 0-9, A-Z, a-z. Although it is true that our detector might also be able to detect some hand-written letters, we restrict our search space to the computer-printed versions of the letters. Another restriction is that we only use rectangular bounding boxes and we do not take into account that a letter may be rotated.

B. Challenges

In spite of those restrictions in I-A, there are still some challenges to handle:

- As mentioned in I, there are many properties of the letters themselves which make them differently.
- The aspect ratio of a letter may also vary, especially, if the perspective of view changes. Then also the letter may also be distorted.
- The 62 letters have very different shapes and aspect ratios. So it might not be possible only to train one model for all letters in order to get good results.
- The background of the bounding boxes containing the letter may also vary or even contain parts of other letters.

So one way to handle those challenges is to find similarities of letters in all varieties which we can use to extract features on it. In the following chapter, we will do this by using HOG features and image segmentation.

II. Data

Firstly, we describe where we get the data from and how we organize and transform it before we use it in our models.

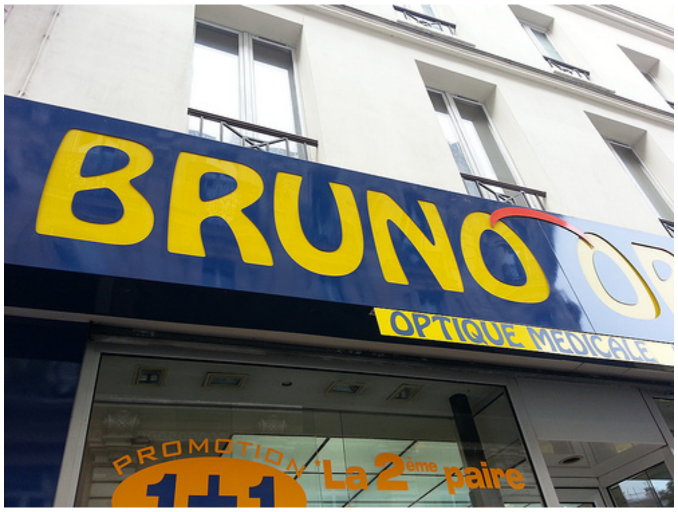

Fig. 1. A positive example by “ctw1500” [4].

A. Image selection

The selection of the data consists of 3 parts. The GitHub page “image-text-localization-recognition” [1] was very helpful for image selection.

Firstly, we need positive training examples which contain letters and we also know their positions within the images. It is an advantage if those letters already fulfill most of the properties described in I-B. Otherwise, we need to construct them out of the available examples or we have a feature extraction technique which removes those properties.

We used a part of the data set “Chars74k” [2] of 62992 computer-generated images. Each image has the size 128x128 and contains exactly one character of 0-9, A-Z, a-z with a specific font and may be italic, bold, both or normal. Each character occurs in 1016 images.

Secondly, we also need negative training examples which should not contain any kind of letters. One single letter usually contains a small space in a whole image where we want to locate the letters. So we can use a part of the image as one negative training example which means that we can reuse the same image for different parts of it and we need less initial images if they are large enough.

We used two subsets of the data set “Google Open Images V4” [3] which contains more than 9 million images with a high diversity of shown scenes. One subset contains 100

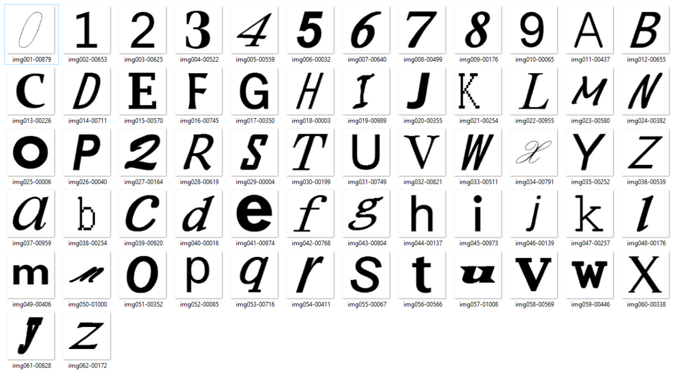
Fig. 2. Some positive examples by "Chars74k" [2].

images which do not contain any letters and the other one contains the first 2627 images.

Thirdly, we need test data which are images which we could also have added to the training examples. There should be easier and more difficult examples such that we may easier see where the evaluated model has its problems.

We used some images of "Google Open Images V4" [3] again and some of "ctw1500" [4]. Additionally, we created a very simple test example just consisting of some randomly distributed letters with white background.

B. HOG features

One technique to get a more abstract representation of the images, which still contains the information about the shape of the letters, is HOG feature extraction. Since the contrast between the border of the images and their background is usually pretty high, this piece of information is also contained in the HOG features.

An advantage of this technique is that we do not have to care about the colors of the images. We also have reduced the dimensions of the data set by a lot.

An issue in this technique is that differences in the fonts of the same letter are still visible in the HOG features. So it is necessary that the data set [2] contains a lot of different fonts. Also the differences between the letters are still visible. Since we later want to use an SVM with Gaussian kernel, we have to train a separate model for each letter since the differences between the letters are too huge to fit all in one model. Another issue is that we have to use a fixed aspect ratio of a model, but it is no problem for the data set [2] since all images have the same size.

C. Image manipulation

We still are able to do some manipulation of the images before we extract HOG features from them. Since it is very easy to find the most narrow bounding boxes for the letters in data set [2], we can easily create new training examples out of them. Firstly, the images may be rescaled by a certain factor (with fixed aspect ratio). Secondly, we implemented 4 options for further operations:

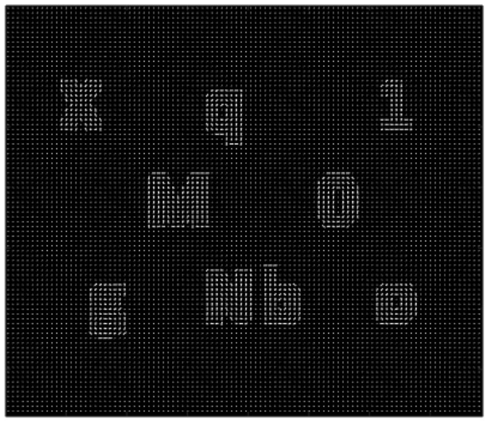
Fig. 3. A visualization of HOG features.

1) plain: Do not change the images. Only check if all images have the same size.
2) crop: Crop the bounding box of each image and set it to the center of a white image with the maximum height and width among all bounding boxes.
3) resize: Do the same as in 'crop', but also rescale the image until height or width of the bounding box hits the border of the white image.
4) locate: Relocate the bounding box somewhere randomly within the original image. Here is also the option to make more than one copy for each image.

There would have even been the possibility to use a negative example as a background and lay all the black pixels over it.

As already mentioned in II-A, we use every negative example more than once by cropping a random part with the size of the positive examples. The other positive examples, that are not containing the letter for a model, are also used as negative examples.

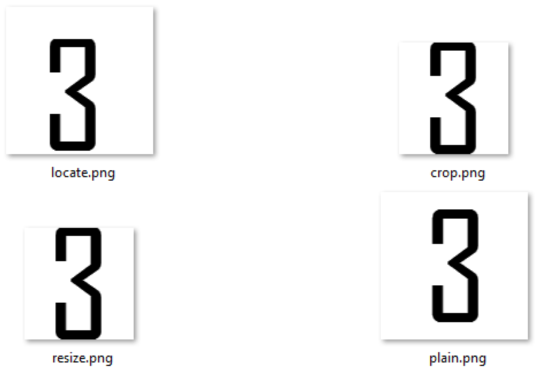

Fig. 4. The 4 options for image manipulation with rescale factor 1.

III. Models

Now, we can use the data to train several models.

A. Support Vector Machine (SVM) [5]

As a first model, we use a binary SVM with Gaussian kernel. As mentioned in II-B before, we train a separate model for each letter. For this model, it is important that we have enough data on the boundary of between positive and negative examples such that we have enough support vectors.

B. Cascade Object Detector [6]

As a second model, we use the Cascade Object Detector provided by MATLAB. We also train a separate model for each letter here.

IV. Evaluation

After we have trained our models, we can now use them on the test data. Firstly, we scan through the whole image and predict the output for each segment (IV-A). Then we need to remove overlapping bounding boxes which do not give us more information (IV-B). In the next step, we have to reduce the results of all 62 models to one final result (IV-C) which we can finally evaluate with some metric (IV-D). The Cascade Object Detector does the first 2 steps for us automatically. We only need to set the hyper-parameters.

A. Pyramid and sliding windows

We have a slider with a fixed size here. We use it to slide over the whole test image and predict the output in the SVM for each segment of the image where the slider stopped. For every segment, which the SVM has predicted as positive example, we remember its borders as a bounding box and how high the score was. We repeat this process for the whole pyramid which is created by creating smaller copies of the test image.

B. Non-maximum suppression

Since many predicted bounding boxes from the previous step may overlap, we have to reduce them. We do this by first selecting the bounding boxes with the highest scores and later only choosing one with a lower score if it overlaps with a previous one by a maximum percentage which we set to 50%.

C. Many to one

We now can predict the occurance for every of the 62 letters. But this is not that what we wanted. We wanted just to know if there is a letter in some position or not. So we have to put all bounding boxes together in a way that the true-positive predicted letters still remain, but the maybe many false-positive predicted letters are removed. However, this part is not implemented so we cannot do further analysis here.

D. Intersection over Union (IoU) [7]

The metric we use is called “Intersection over Union”. As the name says, we define

$$\text{Precision} = \frac{|TP|}{|TP \cup FP|}$$

and

$$\text{Recall} = \frac{|TP|}{|TP \cup FN|}$$

where $TP = T \cap P$, $FN = F \cap pos$, $FP = F \cap neg$. T is the union of all ground truth bounding boxes and $F = \overline{T}$ is the remaining area. pos is the union of all predicted bounding boxes and $neg = \overline{pos}$ is again the remaining area.

V. Framework

At the end, we will give a short overview about our framework which can be found at https://github.com/Markimk13/ml-project/tree/master/project.

A. Support Vector Machine (SVM)

The executable script is named `svm_project.m`. Some parameters for this can be controlled in `svm_params.m`. The paths for the .mat files, where data, features and model can be stored, are controlled in `svm_filenames.m`. If you want to delete some of the older values, you can control this in `init_params.m`.

The script loads and manipulates the data, extracts the HOG features, trains one model for one letter, predicts the bounding boxes for the test data (which needs more time than training the model) and visualizes the first test example.

B. Cascade Object Detector

The executable script is named `cod_project.m`. All parameters are controlled inside the script here.

There are also other deprecated scripts, namely `cod_project2.m`, `cod_project3.m`, `cod_projectGrid.m`, `cod_evaluateGrid.m`.

References

[1] https://github.com/whitelok/image-text-localization-recognition
[2] T. E. de Campos, B. R. Babu and M. Varma, “Character recognition in natural images”, In Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP), Lisbon, Portugal, February 2009. http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/.
[3] https://storage.googleapis.com/openimages/web/download.html
[4] https://github.com/Yuliang-Liu/Curve-Text-Detector
[5] https://de.mathworks.com/help/stats/fitcsvm.html
[6] https://de.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html
[7] https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/