Graph transformer networks: (Content begins Page 18 of original paper.)

The first step is to segment all the characters. We find all the candidate cuts for the input image. Once these cuts are available, they shall be used to represent a graph based model where each edge represents the image vector obtained from the cut. Here, we will focus on reading the image, applying certain processes and later obtain the vertical histogram. Using this data, we shall try to obtain image segments which can be used for recognition operations.

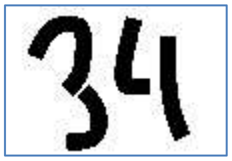
Input image:

The following input image is clipped from the original paper. Let's try to obtain multiple segments for it.

Reading image:

To read the image, we will use the Python OpenCV module. For programming purpose we will use IPython 2.7 notebook. Refer to the 'logs' file for execution details.

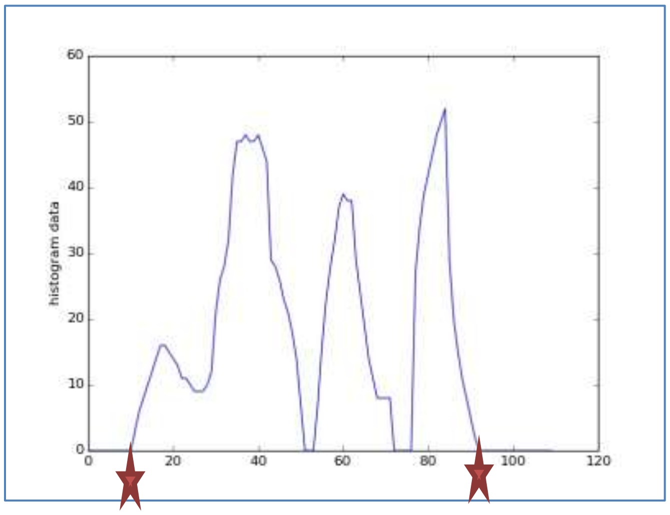
1. To read the image, we have a method 'imread' in this module. The image is read and stored as a numpy array. This is a color image with pixel data in 3 planes: R,G,B to represent colors. A pixel value of (255,255,255) represents white and (0,0,0) represents black. Image dimensions can be obtained by using the shape method.
2. We process this image, by converting it into a grayscale image: RGB planes are converted into 1 plane, representing various shades of gray from 0-255. OpenCV also provides the capability to read the color image and store it directly as a grayscale by using 'cv2.IMREAD_GRAYSCALE' argument. Processed images can be saved in memory using 'imwrite' method call.
3. Image thresholding needs to be done to obtain a vertical histogram to help us understand the image structure. For thresholding, we have kept the max value as 255 and thresh value as 128. Any pixel wit intensity more than 255, gets converted to a 1 or white, and lower values to black or 0. The thresholding function returns the threshold value and a black and white image.

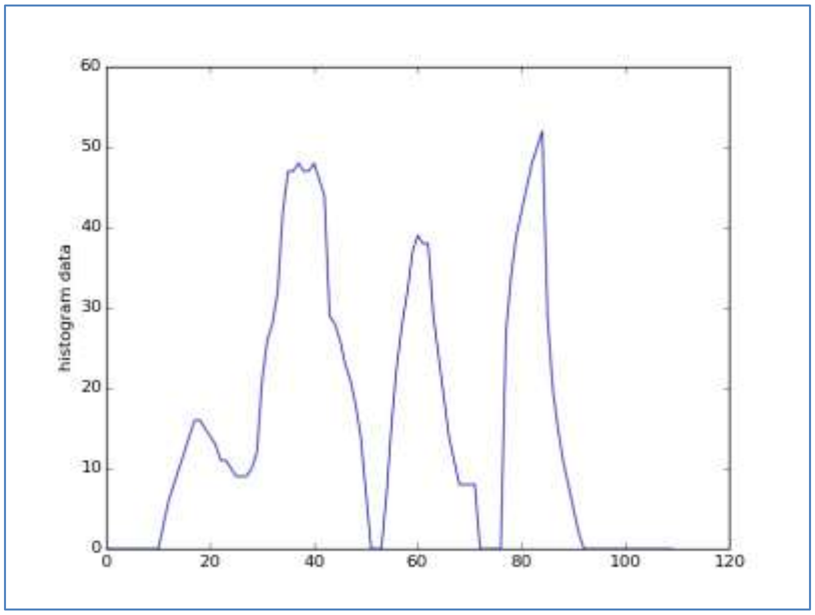


4. While thresholding, we take an inverse. This will help us in sketching the vertical histogram, explained below.

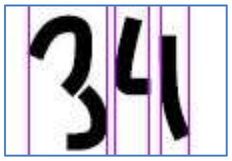
5. At this stage we are ready to draw a vertical histogram. We scan the above processed image on a per column basis and count the sum of pixel intensities for all the rows. First we scan for column 0, read [row0, row1…row_last] and add up the pixel intensities seen during this scan. Repeat the process for all the columns. If the column output is 0, it means that the column contained no white pixels. A transition from a column value 0 to a non-zero value would imply beginning of a character. Similarly a change in gradient from a non-zero column value to a zero column value would imply stop of the current character. We will store this transition data as a list and later use it for cropping images. [Code: refer to 'pr1.py' for calculation of vert-histogram]

6. Once the histogram data is available, we can plot data using matplotlib. This helps us understand the nature of gradient transformations. For eg: the first star would imply beginning of a new image segment since immediately the vertical pixel sum increases. Similarly the next star with a transition to 0 sum would mean finishing of a segment.

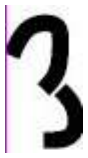




7. Now, we try to obtain the exact pixel locations where these transformations occur. Refer to 'pr2.py' for computation details. We can also draw lines to visualize the locations found over the original image.



8. Finally we crop the segments. Refer to the segment method to check this out. Segments look like:



Notice that the lines we drew at the previous step are also visible. They can be eliminated by cropping starting from the next pixel, or imagine that we never drew lines (or drew on another version and we crop the original color image).