Img2Pdf

CS 293 Final Project Demo

November 16, 2012

Mayank Meghwanshi – 110050012

Shivam H Prasad - 110050041

Aim of the project

- Detection of possible text regions in the input image
- Extraction of characters from text regions detected by OCR (using template matching algorithms)
- Writing the extracted text information from the file to PDF file
- A nice looking GUI to give a feel of good image to PDF converter

Demo

- Our aim is to take an image containing some text information as an input and generate a PDF file made out of the text info contained in input image.
- Text info extraction from the input image involves three important steps :
  - Text Region Detection
  - Text Area Localization
  - Optical Character Recognition

Finally after implementing these steps we get all the text contained in image which we write into a PDF file.

Demo

- Now exploring each step one by one :-
- **Text Area Detection –**
  - We start with an **input image which may be colored or grayscale** containing some text information which we wish to extract. **Text detection is normally easy on grayscale images** as compared to colored ones as it throws away unnecessary color information and keeps just the intensity factors which play a key role in text detection. So we begin with converting input image (if colored) to grayscale image.

Demo

- Obviously the input image can have some noise along with the relevant text info which can create problem in text extraction. So **next step is to remove noise**. We do this using Gaussian Blur. Thus, **we apply Gaussian blur on our grayscaled image**. Though it is not possible to remove all the noise present in the image but still Gaussian blur give satisfactory results.

Demo

- It is a general observation that text always contains sharp edges therefore **in the input image text will be present in those regions which are dense in terms of sharp edges**. Thus, our next step is to detect edges in the image. This we do using **Caney's Edge Detection** algorithm. Once we get all the edges we apply **open and close image operations**. Opening of image removes small dark patches and closing removes small white patches. At the end of edge detection and after operating open and close image algorithms we get text areas in the image almost in the form of rectangles.

Demo

- Once we get the possible text areas marked next step is to detect connected components in the image which will be the text information in image like words, sentences etc. Getting connected component information involves one intermediate step which is to convert the image into binary image so that connected components can be easily extracted. **We use Balanced Histogram Thresholding for binarisation of image and a self worked out algorithm to find connected components in the input image.**
- This finishes Text Area Detection part.

Demo

- **Text Region Localization –**
  - Now that we have got all the connected components in our image which are possible text regions we start to localize the text info in form of lines and characters.
  - We have connected component information of our image in a sense that each pixel has a label as to which connected component it belongs to.
  - First we find co-ordinates for enclosing boxes of each of the connected components. We store this information so now we have in our hands rectangles in the input image each enclosing a connected component possibly containing text.

Demo

- Now we start analyzing connected components one by one and horizontally partitioning the box enclosing the corresponding connected component depending on certain thresholds. What this does is to partition a single connected component into lines if multiple lines get trapped in a single connected component.
- Finally we get rectangles enclosing single characters in the input image by partitioning the rectangles containing single lined text which we extracted in previous step. Thus we end up getting as many boxes as there are characters in the image so that portion of image enclosed by each of these box can now be processed by OCR to get character outputs.

Demo

- **Optical Character Recognition –**
  - After text localization only step left is to now recognize text in the image. We have used **template matching algorithms** for character recognition. **We have 16x16 standard font templates**. What we do is convert each of the character from the input image into a similar 16x16 template and then match this against all standard ones. **The standard template which gives best normalized 2D correlation on comparison with character template is declared as detected character**.

Demo

- Now that we can get characters corresponding to each of the character box we can process all the character boxes in one go and write all text info into a PDF file.
- **This we do by first writing all text info into a latex file and then using system commands to get a PDF out of that latex file.**
- As proposed in the project proposal we have also implemented a GUI for all this process using gtk+ library. **The interface gives option to open an image, write text to PDF file and also to preview all the intermediate steps involved in the text detections and text localization**.

Teamwork Details

- From the very beginning of the project we did not have much idea about implementation of OCR or text extraction.
- So we had to spend lot of time googling and reading different papers published in this field. So we had to discuss at every stage about algorithms that we will be using and both have read different papers and discussed together about different approaches.
- Our code was stepwise execution of different algorithms so working together was helpful in a sense that we both knew what was happening and it is easy to detect errors and change code because both the team members are doing code together so remembering part of code was also not a big problem.
- So finally we did almost equal work as most of the time we sat together for coding and reading.

Overall Contribution by Team Member 1 -Mayank	Overall Contribution by Team Member 2 - Shivam
50 %	50 %

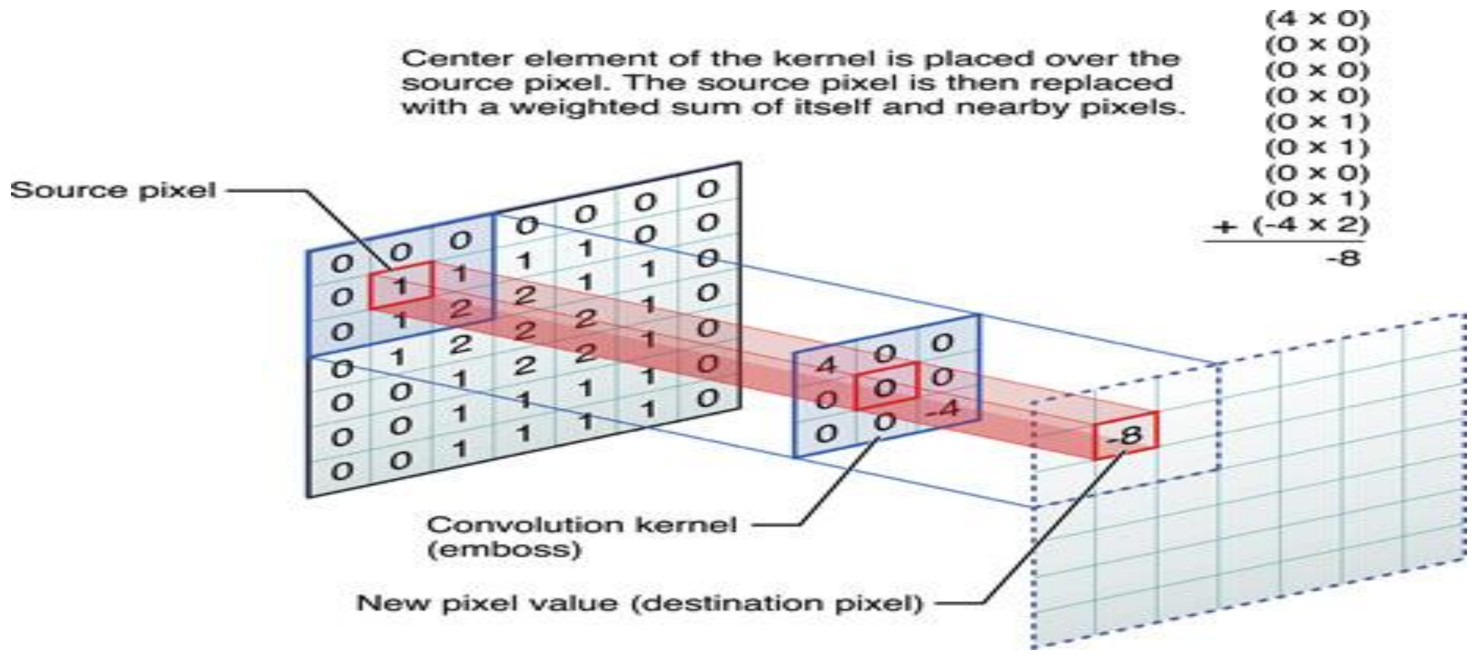
Design Details

- **Colored to Grayscale Conversion :**
  - Colored image is converted to grayscale by calculating grayscale values for each pixel using **weighted sum of RGB values** of the pixel in colored image.
  - Formula used is

    Grayscale = 0.2126*R + 0.7152*G + 0.0722*B
  - This has O(width * height) time complexity.

Design Details

- **Convolution Operators :**
  - Convolution is an image processing technique which changes the value of a pixel to reflect the intensity values of its neighboring pixels.


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 × 0)
(0 × 0)
(0 × 0)
(0 × 0)
(0 × 1)
(0 × 1)
(0 × 0)
(0 × 1)
+ (-4 × 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Design Details

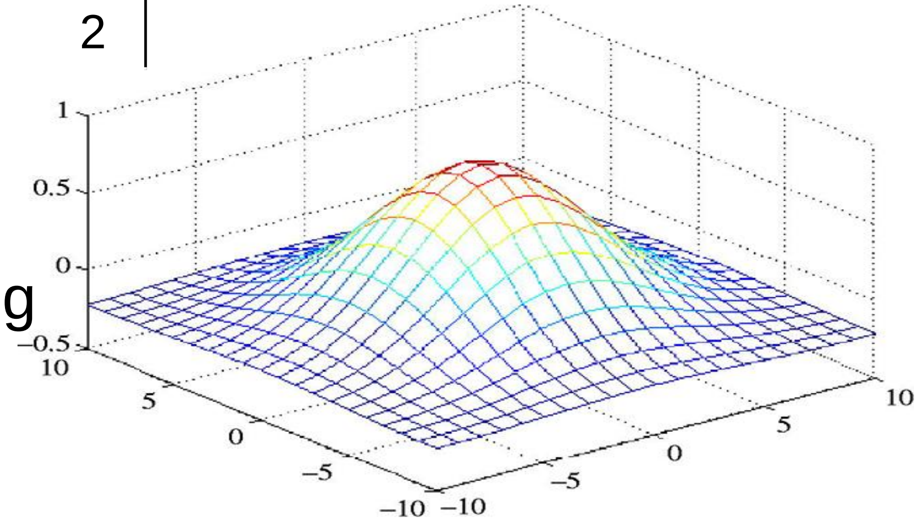
- **Convolution Algorithm:**
  - Convolution operation involves a **structuring element (nxn matrix)** which defines the weights of the neighboring pixels with respect to central pixel.
  - **Structuring element is placed on the image and pixel value corresponding to its central element is updated to weighted sum of surrounding pixels of the center**.
  - This runs in O(n^2 * width * height) but since we are using constant matrices so n^2 term turns out to be constant reducing order to O(width * height).

Design Details

- **Gaussian Blur Algorithm:**
  - Structuring element for Gaussian blur is

$$\frac{1}{159}\begin{vmatrix} \mathbf{2} & \mathbf{4} & \mathbf{5} & \mathbf{4} & \mathbf{2} \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{vmatrix}$$

  - Applying Gaussian blur has the effect of reducing image's high frequency components.



Design Details

- **Sobel Operators:**
  - **Horizontal Sobel-**

$$\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$$

  - **Vertical Sobel-**

$$\begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}$$

  - At each point in the image Sobel operator calculates gradient of intensity around a pixel.
  - This results in marking of **vertical (vertical Sobel) and horizontal (horizontal Sobel)** edges in the image

Design Details

- **Modular Addition of Sobel operated images:**
  - In order to combine the effect of vertical and horizontal sobel operators i.e. to get both vertical and horizontal edges of image we take modular addition of pixel values of both Sobel operated images at given point and assign modular sum of these values to corresponding pixel in the new image which will have both vertical and horizontal edges highlighted.
  - New pixel value is calculated as

    newImage[i][j] = sqrt(Image1[i][j]^2 + Image2[i][j]^2)
  - This runs in O(width * height) time complexity.

Design Details

- **Erosion of an Image:**
  - Erosion of image gets rid of small white patches by changing pixel value to minimum of its 3x3 neighbors.
  - This has O(width * height) time complexity.

- **Dilation of an Image:**
  - Dilation of image gets rid of black patches like small space that can be enclosed inside character **'O'** and such similar patches by changing pixel value to maximum of its 3x3 neighbors.
  - This has O(width * height) time complexity.

Design Details

- **Opening of an Image:**
  - First the image is eroded and then dilated.
  - This has O(width *height)
- **Closing of an Image:**
  - First the image is eroded and then dilated.
  - This has O(width *height)

Design Details

- **Averaging Opened and Closed Image:**
  - This creates an image which is simply arithmetic mean of opened and closed image.
  - Average image is needed to project the effects of opened and closed image in a single image.
  - Formula for calculating pixel value of averaged Image from opened and closed images is

    averagedImage[i][j] = (Image1[i][j] + Image2[i][j]) / 2
  - This is simply an O(width * height) algorithm.

Design Details

- **Binarisation of Image:**
  - This updates the value of all pixels in image to 255 if original value is greater than threshold and 0 if original value is lower than threshold.
  - First, cumulative histogram is calculated for the image and also cumulative sum i.e. sum[i]=sum[i-1]+N[i]*I
    - Where N[i] is number of pixels with pixel value I
  - Assume threshold to be 127 and create two partitions i.e. one with higher pixel values than threshold and other with lower pixel values.
  - Now mean of both parts is calculated using cumulative sum and cumulative histogram.

Design Details

- **Binarisation of Image:**
  - Now a new value of threshold is calculated by taking average of means calculated in previous step
  - This process is iteratively repeated until difference between new and old threshold is less than 0.5
  - Now using this threshold binary Image is generated
  - This runs in O(width * height) + O(iteration)
  - Since O(iteration) turns out to be smaller than width*height therefore final order is just O(w*h)

Design Details

- **Extraction of Connected Components:**
  - Corresponding to each connected component we assign a label to all the pixels belonging to that component.
  - For every white pixel we check 4 neighbors which are left, top left, top and top right which by inspection we see that will always be already visited before that pixel.
  - After these we have three cases
    - If all four neighbors are black we assign a new label to this pixel
    - If there are some white pixels with same label then the new pixel is assigned same label

Design Details

- **Extraction of Connected Components:**
  - If some pixels are white and they have different labels then this implies that they are multiple connected components which are merging to form bigger connected component at this pixel so we add all these values to same equivalence class thus all those pixels now belong to same connected component.
- **Algorithmic Steps:**
  - We have a 2D vector for storing label of a given pixel to access in O(1).
  - Also we have buckets(list of pairs) containing pixel location with same label.
  - When we add new label we add a new bucket and add that pixel in it.

Design Details

- **Extraction of Connected Components:**
  - **Algorithmic Steps:**
    - When we have same label values in neighbor we just add current pixel to its bucket.
    - When we have different label value then we just empty one bucket into the other and update the label values corresponding to each pixel.
    - Finally we have pixels corresponding to each connected components in separate buckets.

Design Details

- **Text Area Localization:**
  - We start by defining the boundaries for connected component found in previous algorithm.
  - We are trying to find minX, minY, maxX, maxY for each label in the image so that we can having enclosing rectangles for each connected components.
  - We visit each pixel in image and checks its x and y values with min and max of its label and replace min and max values if needed.
  - Finally we have box vector which contains information related to enclosing rectangles.

Design Details

- **Text Line Localization:**
  - For each connected component we calculate number of black pixels in the enclosing rectangle of the given label.
  - We partition the connected component horizontally at points where number of pixels are less than 20% of width of the connected component rectangle.
  - From here we get boxes which contain single line or sometimes single words whose enclosing rectangle component information we have from above steps.

Design Details

- **Single Character Partition:**
  - Now for every text line box we partition it in vertical partitions which are finally single character boxes.
  - We do this by partitioning it at places where in a given vertical line there are no black pixels.
  - This will finally return character boxes which are rectangle coordinates of single character which will be used in character recognition.

Design Details

- **Pre Processing of Font Templates:**
  - We have image template of each character in our database with name [ascii code].jpg.
  - We by command line argument load all the images and for each image we convert template to grayscale than binarize the template image.
  - Now, we remove extra whitespace from each side of the image and than rescale it to a 16x16 binary template.
  - We have a template struct which have 16x16 matrix, mean, horizontal and vertical projections and character it represents which is extracted from filename.

Design Details

- **Pre Processing of Font Templates:**
  - We create a binary file to write all the templates created. (char_dict.bin)
  - Now, this file is read into our main program of image processing and used for template matching of each character.

Design Details

- **Character Detection Step:**
  - In this function we take out the portion of image containing character by using enclosing rectangle coordinates.
  - We create template of this portion by rescaling and converting to 16x16 image template.
  - We now compare this template to all the standard font template by using 2d normalized correlation factor and also using horizontal and vertical projection’s correlation.

Design Details

- **Character Detection Step:**
  - Normalized 2D Correlation of x-y is given by:

$$\text{Correl} = \frac{\sum(x-xmean)\,(y-ymean)}{sqrt(\sum(x-xmean)\hat{}2 + \sum(y-ymean)\hat{}2)}$$

  - This correlation is maximized to find the best matching standard template.
  - Corresponding character is returned as recognized character.

Design Details

- **PDF Generating Step:**
  - We for each text line block (assuming it to be single word) write it to a latex file using the detected character.
  - Than using system command pdflatex convert the latex file in a pdf file and than using rm command extra files formed are deleted.

Design Details

- **GUI in gtk+ library:**

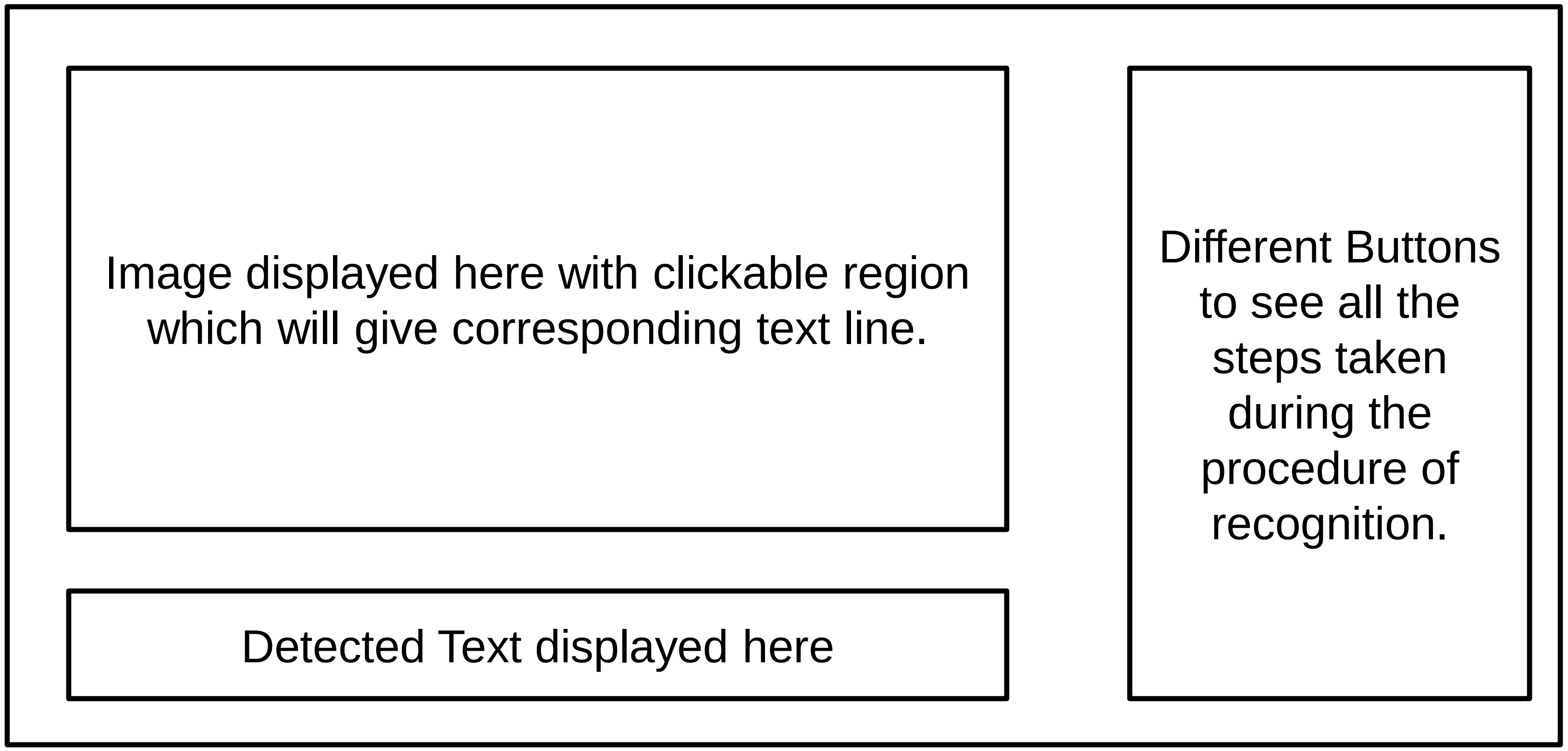


Image displayed here with clickable region which will give corresponding text line.
Detected Text displayed here
Different Buttons to see all the steps taken during the procedure of recognition.

Data Structures Used

Purpose for which data structure is used	Data Structure Used	Whether Own Implementation or STL
Vector	Storing standard font templates	STL
	In binarize to represent cumulative histogram	
	Storing enclosing rectangles at different steps	
	Storing label corresponding to each pixel value	

Data Structures Used

Purpose for which data structure is used	Data Structure Used	Whether Own Implementation or STL
Priority Queue	For storing correlation corresponding to each character for further processing	STL
List	To make buckets of labels containing pixel values	STL
	List of char boxes corresponding to each text line	

Brief Conclusion

- We successfully covered all the milestones as mentioned in project proposal except complex background text detection.
- We are able to detect 80-85% text correctly.
- Our code is very versatile and easily extendable at every stage. We can replace any algorithm used with better one without much changes in the code.

Brief Conclusion

- We also completed GUI part for the code which was mentioned as extra part in the project proposal.
- We were not able to generate properly formatted PDF as we had proposed but still we used latex to present the text and learned about using system command in c++ code.

Thank You – Questions?

Viva Time \m/