Your Flowchart Secretary : Hand-Written Flowchart Converter

Team bh454nd IIIT Hyderabad December 3, 2024

Outline

1. Introduction
2. Assumptions
3. Pipeline
4. Preprocessing
5. Component Extraction
6. Digital Reconstruction
7. Future Works

Introduction

- Flowcharts are invaluable tools for organizing and visualizing ideas during the prototyping phase.
- However, manually creating well-structured flowcharts for documentation or presentations can be time-consuming and cumbersome.
- To address this challenge, we developed an image processing based pipeline capable of converting hand-drawn flowcharts into digital versions.

Assumptions

1. **Closed Graph Components** : Every shape in the handwritten flowchart must form a closed graph, ensuring that boundaries are well-defined.
2. **Straight Arrows** : All arrows connecting the components must be straight lines, simplifying the detection of directional flow. Moreover, there are 0 intersections between any 2 arrows
3. **Recognizable Shapes** : The system currently supports the recognition of only four specific shapes: rectangles, diamonds, circles, and arrows.

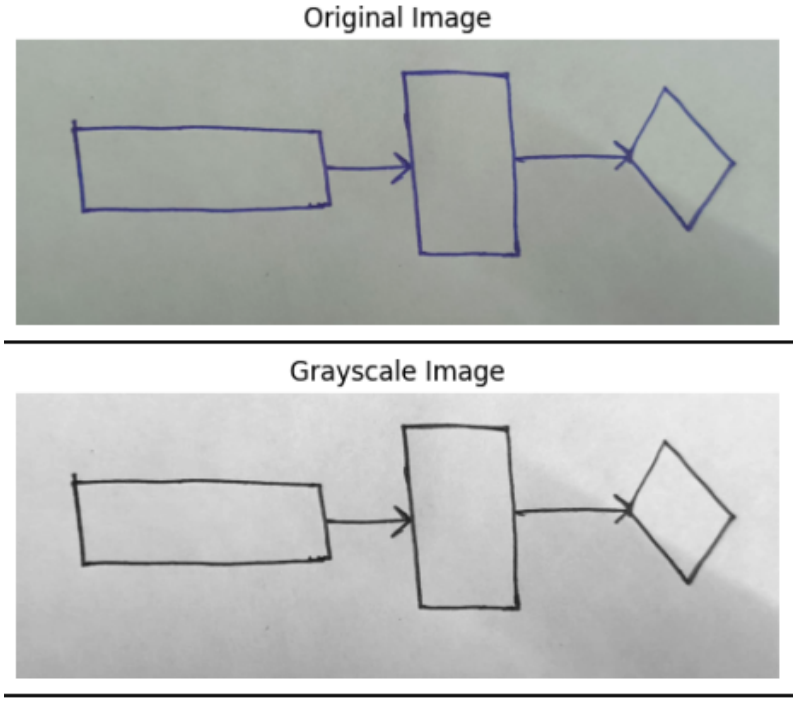
Pipeline

1. **Input and Preprocessing** : The image is loaded and basic operations such as grayscale conversion, denoising and rotation are performed to clean flowchart components.
2. **Component Extraction** : From the clean flowchart, firstly, all the arrows are separated from the shapes. Then, the 3 remaining shapes are identified and stored along with their coordinates..
3. **Digital Reconstruction** : Using coordinate data, relationships are identified between the various components. With these detected connections, a flowchart structure is build by aligning the elements.

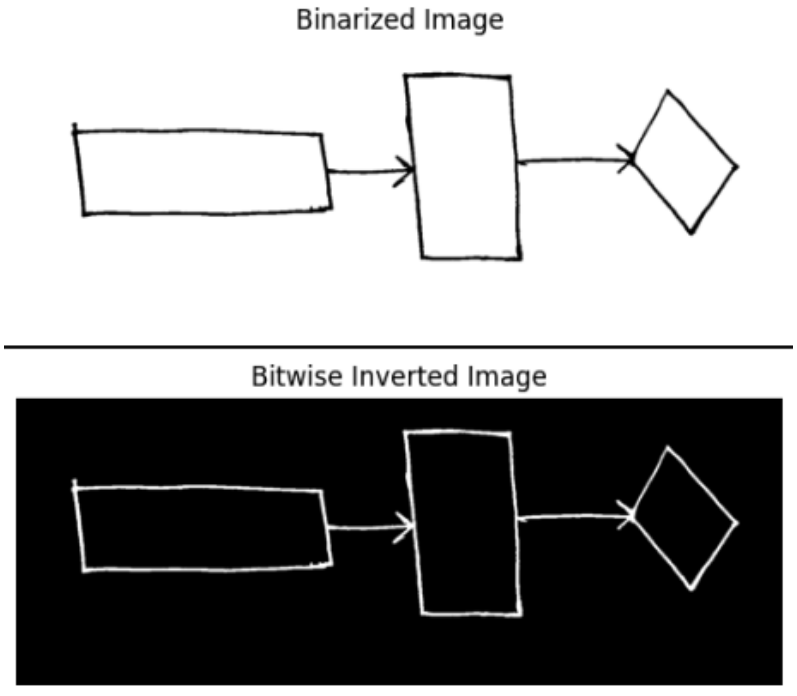
Preprocessing - Image Preparation and Binarization

- We start off by loading in the input image. Large images can be resized to a smaller size for faster computations.
- The RGB image is converted to grayscale to reduce its complexity. We then apply a Gaussian filter to remove all the salt and pepper noise.
- Then, we use adaptive thresholding to clearly separate the flowchart from its background.
- Finally, bitwise inversion is performed to swap the foreground and background colors.

Preprocessing - Image Preparation and Binarization



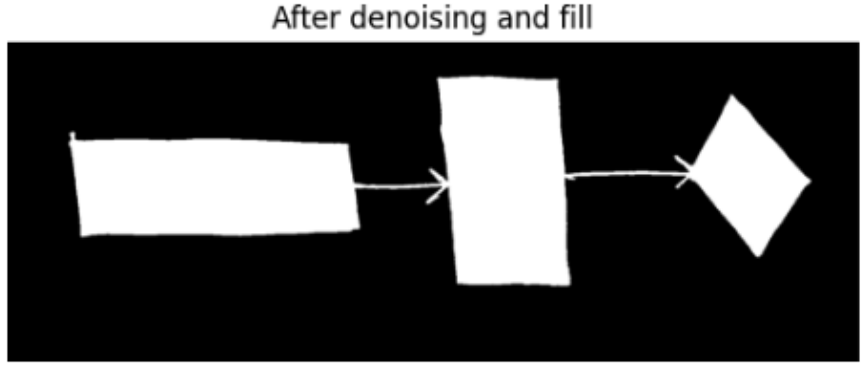
Preprocessing - Image Preparation and Binarization



Preprocessing - De-noising and Small Region Removal

- Denoising starts off by targetting small artifacts left after binarization. These can't solely be dealt with by gaussian filters.
- The small regions are identified by finding contours of the shape and filling black into all the small regions.
- Then, all the shapes and arrows in the denoised image are filled with white.

Preprocessing - De-noising and Small Region Removal



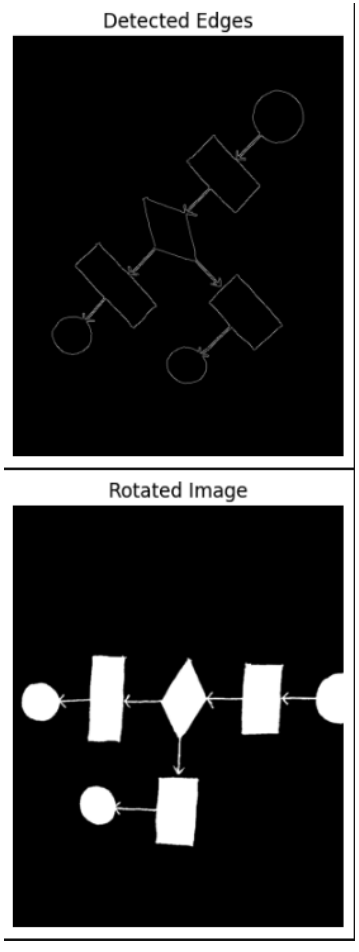
Preprocessing - Hough Transform and Rotation

- To improve the accuracy of the shape recognition and classification, we would prefer having the flowchart in a more favourable vertical or horizontal orientation.
- For this, we first detect all the edges in the image using edge detection techniques and extract all angles between those.
- The primary angle (highest frequency) can then be identified using a histogram of all the angles using a simple 2-part search.

Preprocessing - Hough Transform and Rotation

- This 2-part search first broadly searches by dividing 180 into 10 bins of 18 each. Then it narrows down on the most frequent bins to accurately find the dominant angle of the flowchart.
- Before we can rotate, we also want to increase the overall size of the image since the rotation operation also ends up changing the size.
- By using information about the primary angle and how the transformation matrix would change the coordinates, we enlarge the image into desired size and then perform rotation to retain all the components.

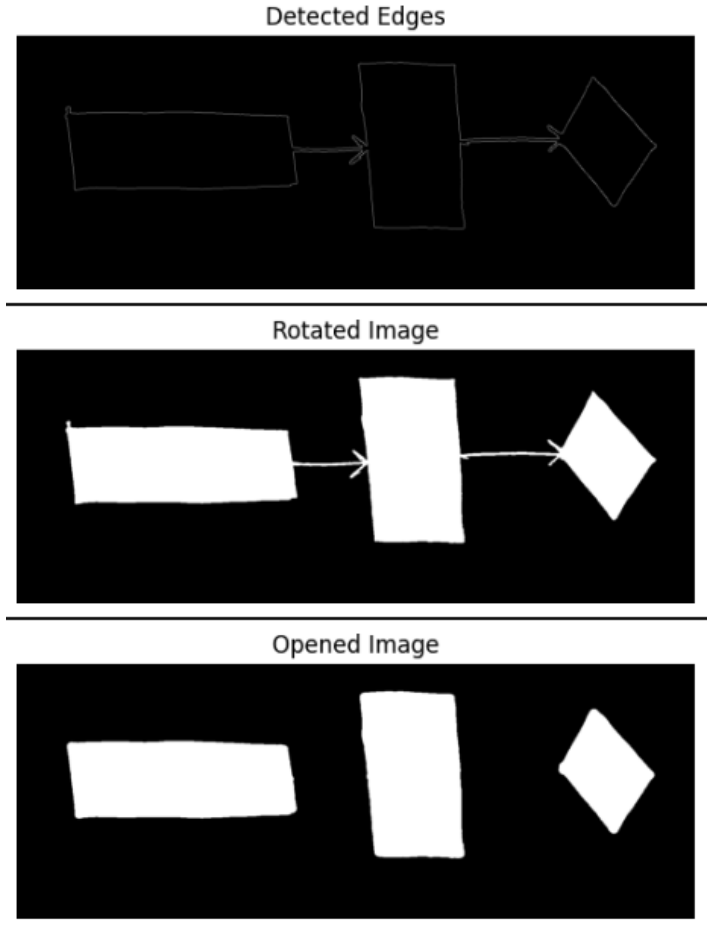
Preprocessing - Hough Transform and Rotation



Component Extraction - Separating Arrows and Shapes

- Now, we want to isolate the shapes and arrows separately for better analysis.
- Then, we erode the white areas to get rid of the arrow and thin all the separate shapes.
- Dilation on this new image restores the shapes to their original size, to which we also perform the denoising and fill operation.
- The arrows are individually obtained by subtracting the detected shapes from the original image. The detected arrows are enhanced by dilation.
- Finally, all the shapes are detected by subtracting the detected arrows from the original image.

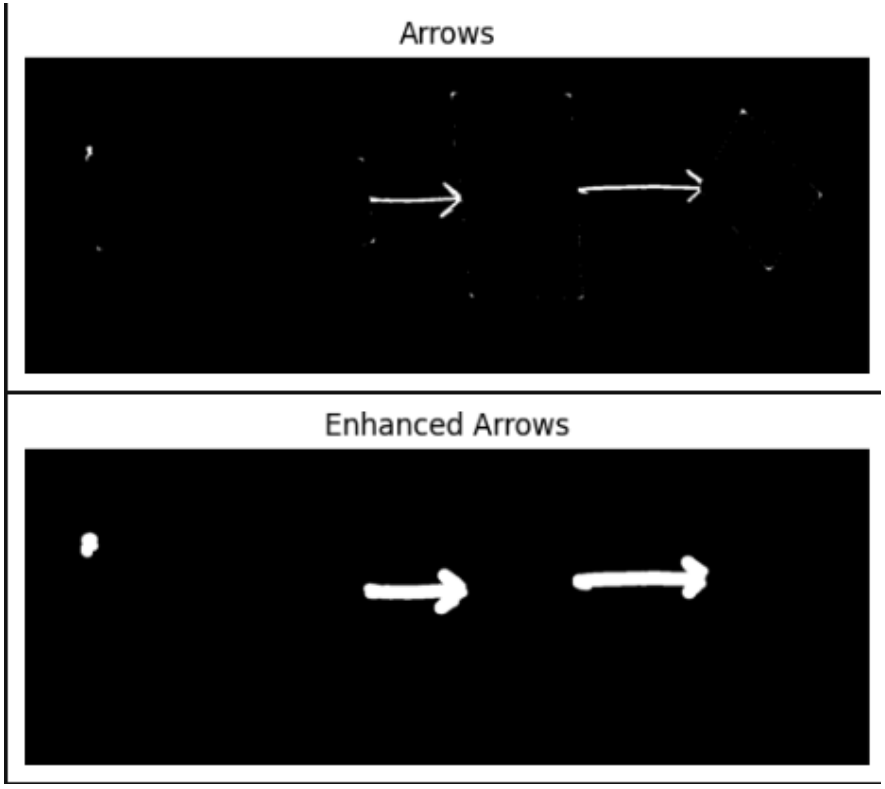
Component Extraction - Separating Arrows and Shapes



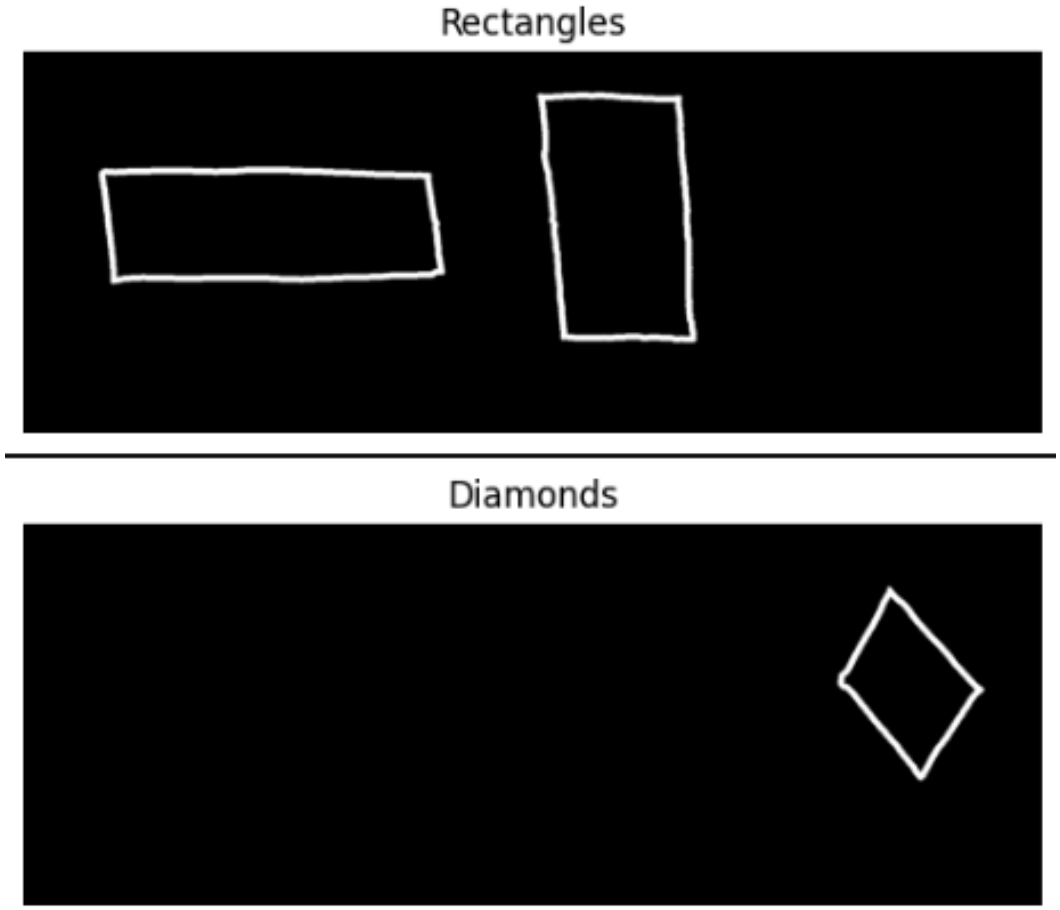
Component Extraction - Identifying Shapes

- Now, our goal is to identify the locations and identities of the 3 shapes - Circle, Rectangle and Diamond.
- Circles are detected using Hough Circles by finding the contours and calculating the value of $\frac{4*\pi*\textbf{Area}}{\textbf{(Perimeter)}^2}$. For circles, this value is near 1.
- Once we get the boundary box for each shape, we calculate the ratio of the area of the remaining shapes and the ratio of the boundary box.
- Rectangles have this ratio close to 1 whereas the shape is decided to be Diamonds if the ratio is less than 0.75.
- Finally, all the shapes are returned along with useful information such as their centroids and coordiantes of the boundary boxes.

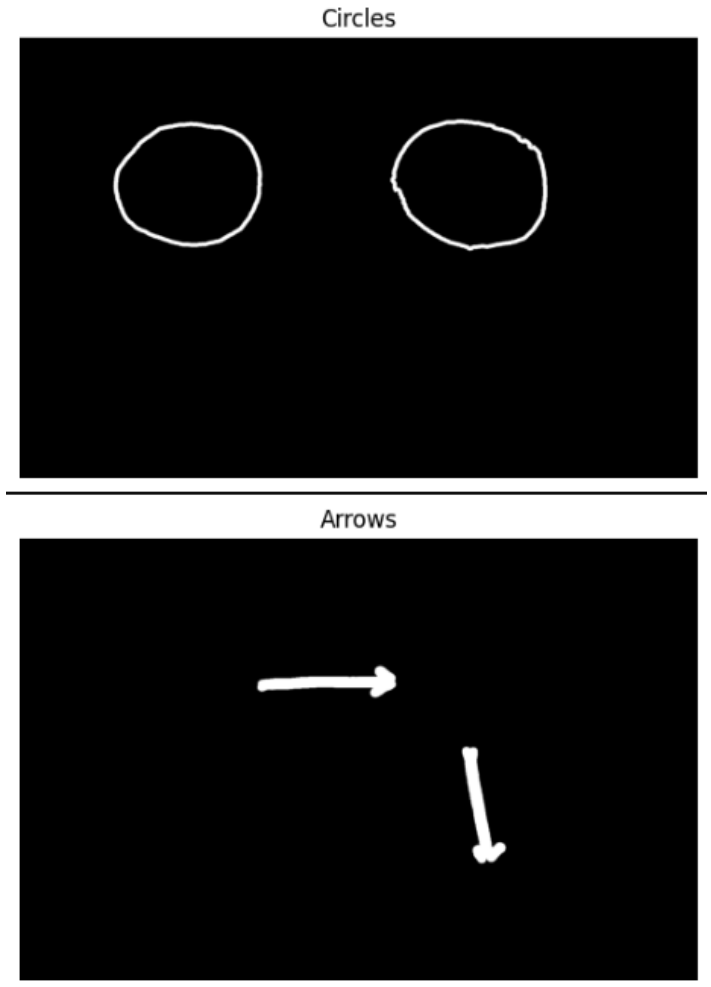
Component Extraction - Identifying Shapes



Component Extraction - Identifying Shapes

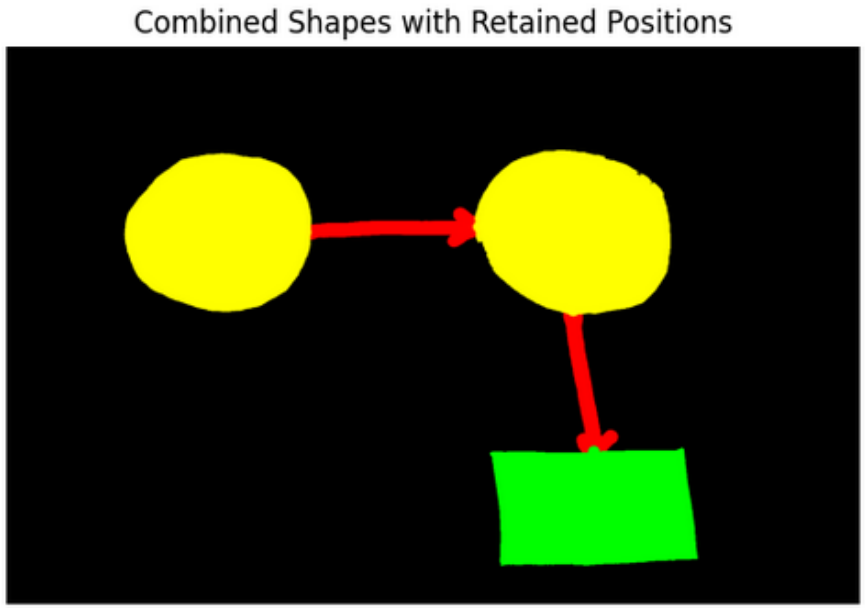


Component Extraction - Identifying Shapes



Digital Reconstruction :

- The function `detect_and_draw_contours_exact`:
  - Uses `cv2.findContours` to detect contours in a binary image.
  - Draws the contours on a canvas using `cv2.drawContours` with a specified color and thickness.



Digital Reconstruction :

- Input shapes (`circle`, `rectangle`, `diamond`, `arrows`) are overlaid onto a transparent RGBA canvas.

- Colors:
  - Yellow for circles.
  - Green for rectangles.
  - Blue for diamonds.
  - Red for arrows.

- Final output is saved as `output-flowchart.png` and displayed using Matplotlib.

Future Works

- It would be a lot more helpful if the image processing pipeline could be used in real time to directly convert any handwritten flowchart. By building an app on top of it, we should directly be able to click photos and get the digital version of the flowchart.
- Since flowcharts also usually contain text within the shapes, we could use OCR techniques to identify the handwritten text and add that to the images as well.
- It would also extend the scope of this project a lot if we could abstract away the logic of detecting shapes and rather just directly use any information (centroids, coordinates, boundaries) to recreate the shape digitally.