**Figure 1.** The architecture of this tool. The input image is processed by two independent pipelines to segment the UI components and detect the text region. Then the results are merged to get the final result.

**Architecture**

We assume the input to our tool to be an image of an interface, which can be either a screenshot of real web or mobile application, or a conceptual design drawing. We focus on segmenting and classifying the possible human-computer interface components on the image. Three categories including totally five types of graphic user interface components are defined in this process: interactive elements (button, input box), static resource (image, icon) and layout structure (block). To perform precise segmentation, we implement this tool as a three-phase pipeline consisting of preprocessing, components detection and classification, combined with text detection achieved by CTPN.

We utilize image processing and computer vision techniques instead of popular deep learning object detection methods (such as Faster RCNN and YOLO) to complete this task, because of the particularity of interface elements.

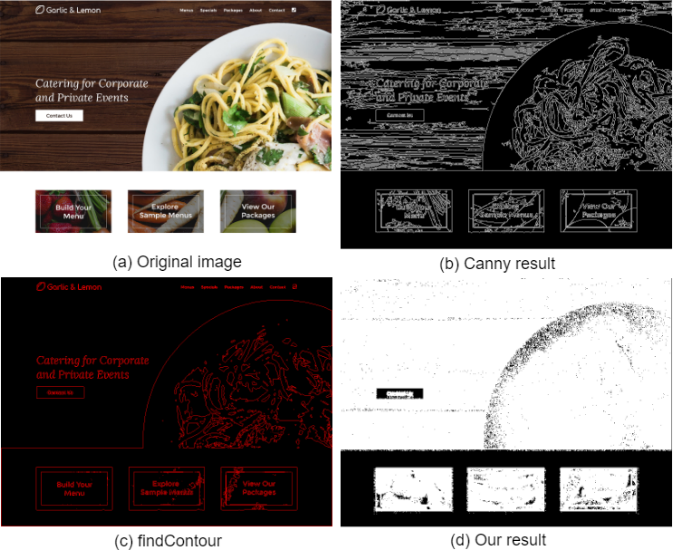
Figure 1 presents the overall technical structure of this pipeline.

**Preprocessing**

The first step is to preprocess the input image to acquire desirable information for further processing. Three substeps are conducted here: gray-scale image conversion, gradient calculation and binary image conversion.

In components detection, we treat all contents on the image as part of individual components without consideration of their own detailed information. For example, an image on an interface should be regarded as a single element instead of a combination of the real contents in it.

To this end, we try to find a means to convert the colorful and complicated image into an integrated object that does not contain redundant information we do not need. The popular related algorithms, such as Canny edge detection and findContour method in OpenCV based on techniques proposed by Satoshi et al., do not work well in this task, because those processing always leave the texture details and disconnect the contents in an image, as shown in Figure 2. So, we propose a new method to meet this need.



**Figure 2.** The picture (a) is the original image; the image (b) is the result of Canny algorithm, which contains too much details of texture; the picture (c) is result of findContour function in OpenCV library; (d) is the binary image processed by our method, which convert the components to an integrated object without too many redundant texture information.

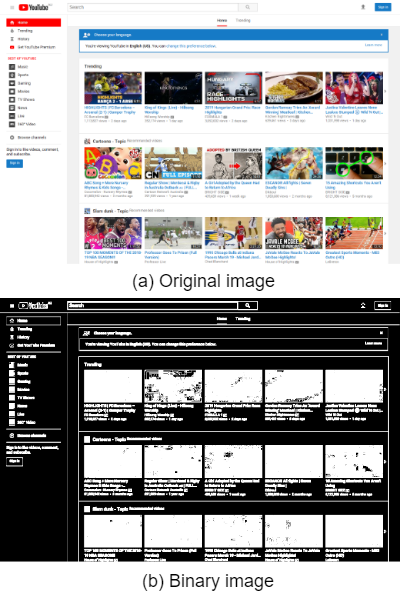
*Gradient Calculation*

The gradient of an image measures how it is changing. Popular techniques of image gradient calculation are Roberts cross operator, Prewitt operator, and Sobel operator. We can acquire two pieces of information from it, the direction of change in terms of pixel value for each pixel, and the magnitude of this change. However, unlike other common computer vision tasks that deal with the natural scene, we do not care about the changing direction as much as about the magnitude because we are only interested in detecting the potential components from the background where the gradient is zero. Therefore, we calculate the magnitude of gradient by formula below:

where: *f(x,y)* is the pixel vale for point *(x,y)* in the image; *gx* and *gy* are the gradient in the x direction and y direction respectively; *G(x,y)* is the gradient value for this point.

*Image Segmentation*

The goal of image segmentation is to assign a label to every pixel in an image. In our case, this step assigns either 255 (white) or 0 (black) to each pixel; whose value is 255 means this point could be part of an interface component; value 0, on the contrary, means this point is part of the background, as demonstrated in Figure 3.



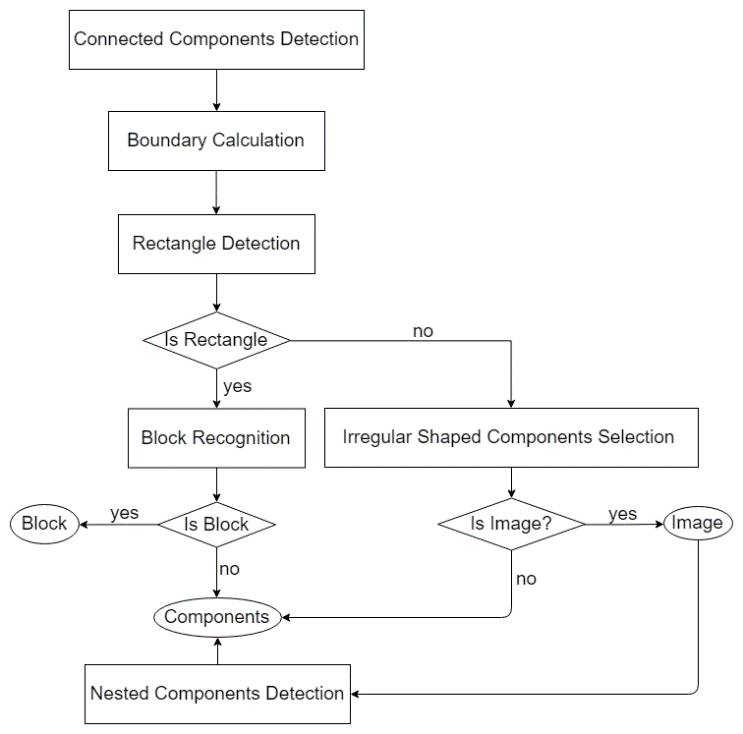
**Figure 3.** (a) is the original image and (b) presentsthe foreground (white) and background (black) for a web graphic user interface.

One observation on graphic user interface is that the regions where there is little or no gradient change are likely to be background. On the other hand, pixels with high gradient are likely to be part the foreground objects (interface components). Based on this, we set a small gradient threshold to label each pixel.

For different datasets, the gradient property would be slightly different, which requires adjustment of the threshold. For instance, the images in Rico dataset are more compact than the screenshots of real webpages, so the threshold should be slightly higher to better segment regions.

**Component Detection**

The preprocessing generates a binary image consisting of foreground points and background; the result is then processed to detect individual objects. This step contains several substeps: connected components detection, boundary calculation, rectangle recognition, block recognition, irregular components recognition and nested components detection.



**Figure 4.** The flow chart of component detection.

*Connected Components Detection*

This step is similar to the connected-component labeling algorithm, the purpose is to assign each pixel a label identifying the connected component to which this pixel belongs.

We implement this process by a simple method similar to the seed filling algorithm in computer graphic.

*Boundary Calculation*

For each connected component, we calculate its boundary by scanning it vertically and horizontally. The resulting boundary consist of four borders: border-top, border-bottom, border-left and border-right. Unlike the popular contour detection algorithm, we do not care much about the precise outer borders and hole borders for each object, because the purpose here is to roughly select the potential graphic interface components instead of acquiring their detailed texture information. Thus, the four-border boundary detection algorithm is sufficient and more efficient in this case.

*Rectangle Detection*

Another observation on human-computer interface is that most of the elements have regular shape. For example, pictures on a website are always rectangle regions, and buttons are usually round or rectangular. Therefore, we introduce the rectangle detection as a heuristic process for interface component detection.

Existing techniques, such as approxPolyDP in OpenCV library and Hough transform, are too complicated and rather unnecessary in our task. We only estimate whether the component is a rectangle or not, but the approxPolyDP method based on Douglas-Peucker algorithm involves too much computation to calculate the precise polygonal curves. Hough transform is also too computationally expensive because it examines four parameters to detect a rectangle, which projects the information into four-dimension computation space.

Therefore, we propose a simple and efficient method to detect rectangle by calculating the smoothness of boundary. In addition, we also measure the magnificent of change to filter out objects that are smooth but concave.

*Block Recognition*

We define a bordered region comprising various elements as block. Block is a layout structure concept here, which could be regarded as a frame or a box. Because it is usually rectangular and hollow, a specific algorithm is designed to recognize it.

*Irregular Shaped Components Recognition*

As mentioned before, the graphic interface elements always have regular shapes (rectangle or round or oval), the irregular objects are more likely to be the content in pictures, or images with transparent background. This step checks all irregular objects and estimates if they are image or potential components.

*Nested Components Detection*

In the previous stages, we do not inspect into components to see their contents, but some functional elements, such as button and input box, might be superimposed on an image. Therefore, the images detected are required to be further process to examine whether there are graphic interface components on them.

**Classification**

We build a convolutional neural network to perform classification of the selected components from the previous step. As shown, we have totally five classes: button, input box, image, icon and block. Block is a layout structure, a bordered region containing several other kinds of components, which is recognized in the last step. Thus, we build a four-class classifier to classify the rest of components.

**Text Processing**

Regarding text, we process it separately by utilizing a popular method called Connectionist Text Proposal Network (CTPN) proposed by Zhi Tian et al. As state-of-the-art natural scene text detection technique, CTPN performs well in the case that text combines with graphic interface design.

After we have the result of image processing pipeline and CTPN respectively, we merge the detection results to remove some misrecognitions, in particular, that the text is misidentified as graphic interface components. To enhance accuracy, we design a method inspired by non-maximum suppression to refine the detection.