Image Processing Project 5

Project report by Yash Lad – u1414500

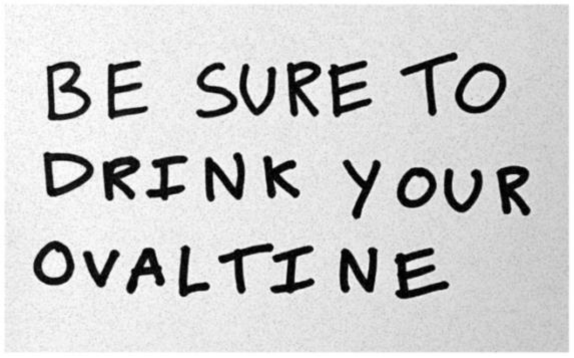
Image Transcription Pipeline =>

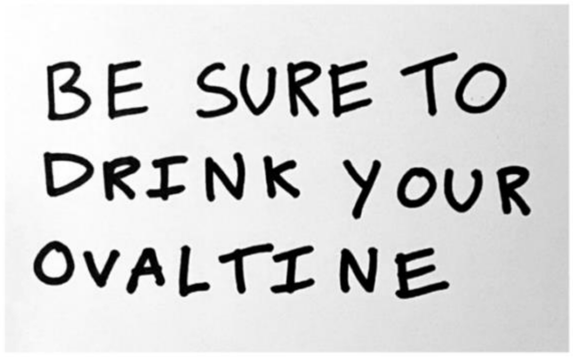
The process of converting the characters in a given image to a text file, can be divided into the following parts:

1. Extract individual characters:

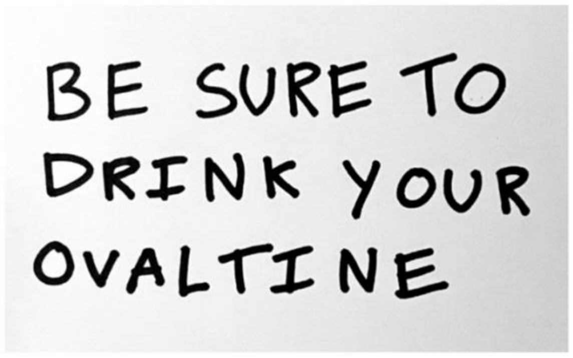
The given input images are noisy, in particular I noticed that every image is adulterated with salt and pepper noise. The best way to remove this type of noise is to use **median filtering**.

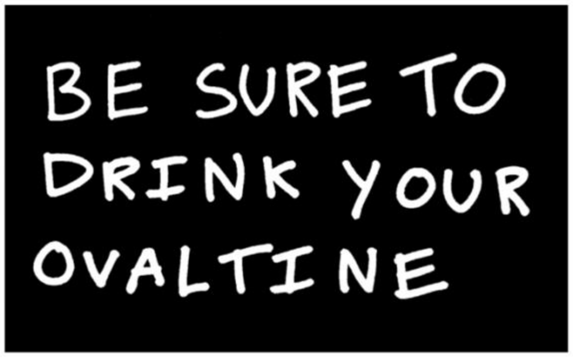
Results after median filtering.





After removing the noise from the image, the next task is to mark the individual characters as foreground and the rest of the image as background for connected components analysis. I have used the **otsu** algorithm to automatically determine the threshold for the image and based on this threshold, created a binary image.



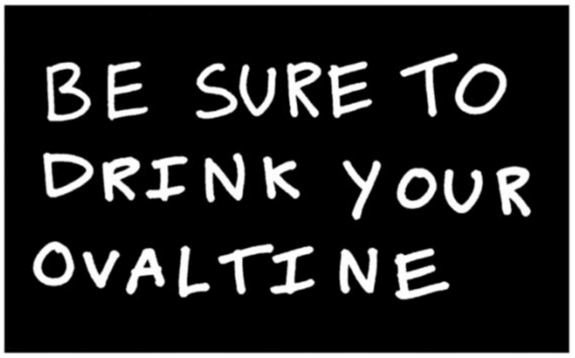


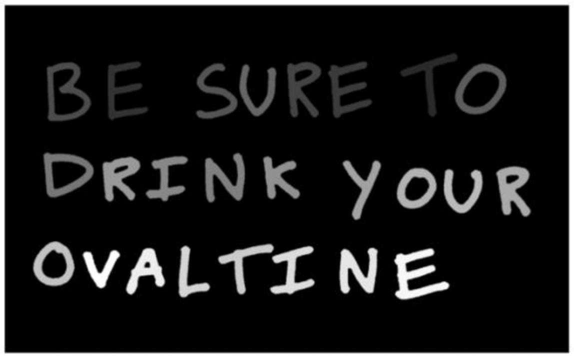
The next task is to use the connected components analysis to mark the labels for each individual characters. Before this step I found it useful to apply the **closing** function from **skimage.morphology**. This function can remove small dark spots and connect small bright cracks. This was especially useful for images containing typed characters, it helped fill the gap between letters such as L, C, T such that the letters are marked as one label and not further divided up into smaller regions.

After this I used the **skimage.measure.label** function to mark the labels for each individual characters.
After this I get the properties of each label like so:

```
region_properties = measure.regionprops(labelled_image)
```

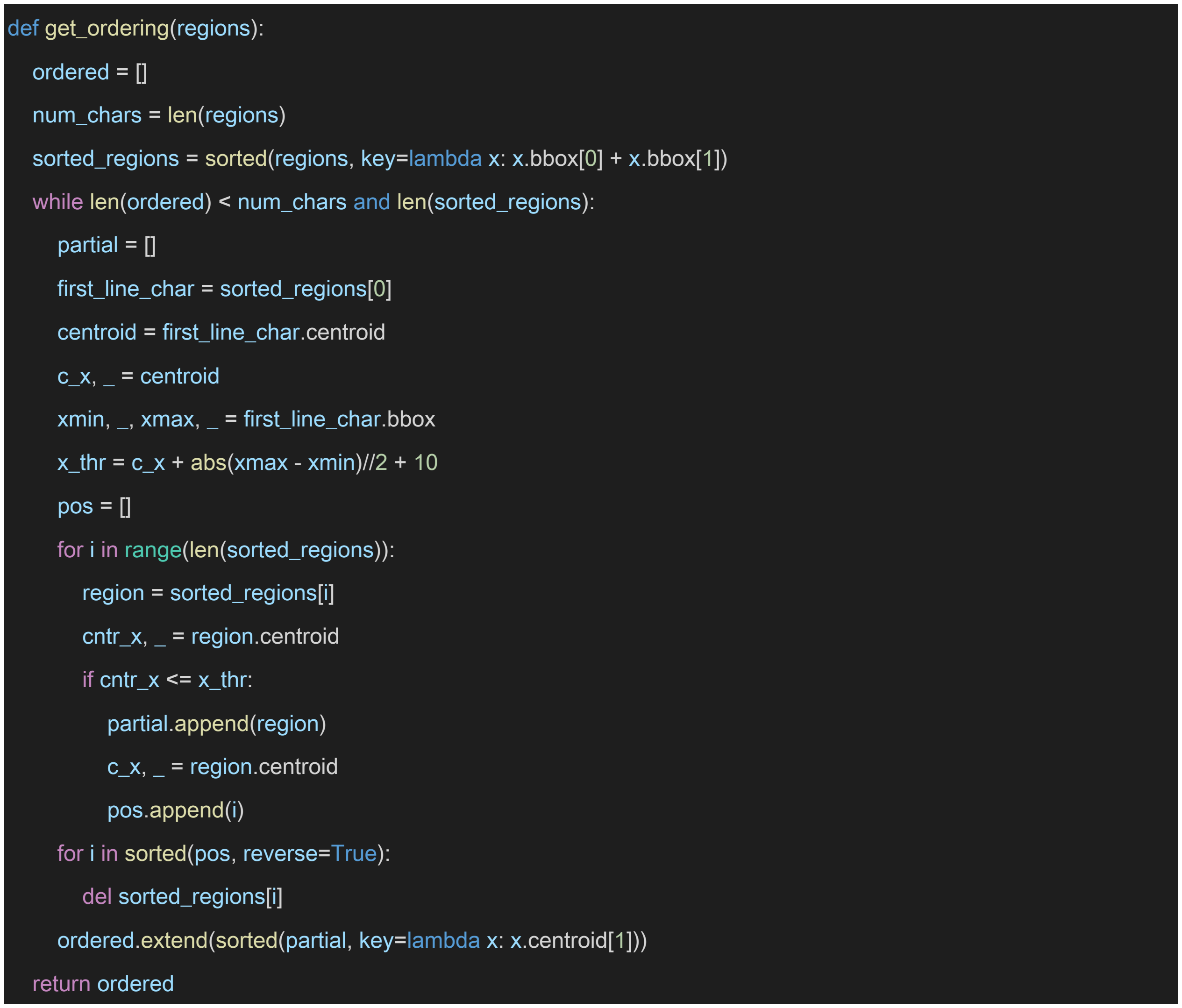
I then filter any regions which have an area lesser than a predetermined threshold for each image.





The most challenging of all task was to order the labelled characters as they appear in the image. To accomplish this, wrote an algorithm which sorts labelled images based on their centroids line by line.

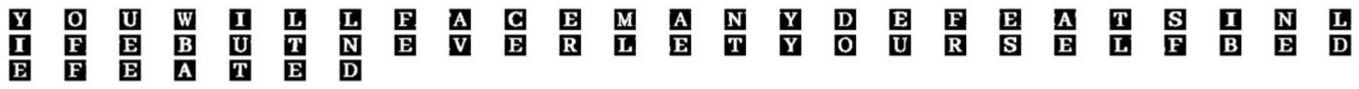
Ordering Algorithm code:

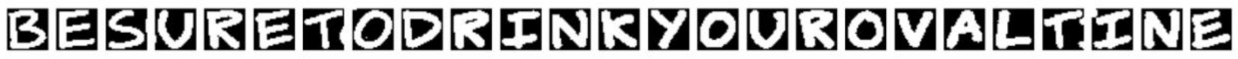


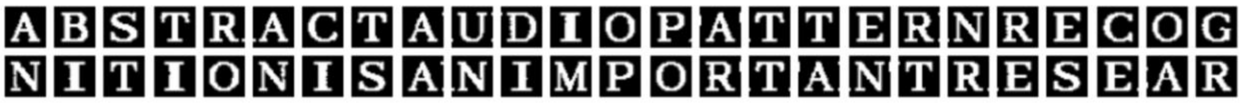
```python
def get_ordering(regions):
    ordered = []
    num_chars = len(regions)
    sorted_regions = sorted(regions, key=lambda x: x.bbox[0] + x.bbox[1])
    while len(ordered) < num_chars and len(sorted_regions):
        partial = []
        first_line_char = sorted_regions[0]
        centroid = first_line_char.centroid
        c_x, _ = centroid
        xmin, _, xmax, _ = first_line_char.bbox
        x_thr = c_x + abs(xmax - xmin)//2 + 10
        pos = []
        for i in range(len(sorted_regions)):
            region = sorted_regions[i]
            cntr_x, _ = region.centroid
            if cntr_x <= x_thr:
                partial.append(region)
                c_x, _ = region.centroid
                pos.append(i)
        for i in sorted(pos, reverse=True):
            del sorted_regions[i]
        ordered.extend(sorted(partial, key=lambda x: x.centroid[1]))
    return ordered
```

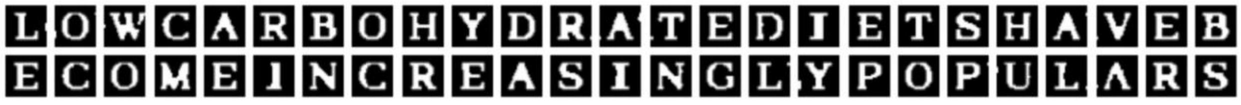
Having sorted the character images as they appear in the image, I extract the crop of each individual character and resize it to feed them to the character predictor model.

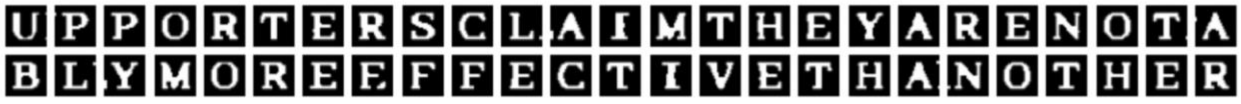
Below are some of the images which are ready to be fed to the predictor:











2. Classify the text:

Having extracted, ordered and resized each individual character in a sorted array, the classification task was fairly straightforward. Using the CharacterPredictor class and the model.pth file for the pretrained model, ran the prediction on each character image and stored the results in a list.

However, I found out that the pre-trained model makes many mistakes in classifying characters perhaps, because of the font of individual characters. I am certain the accuracy can be improved by retraining the model with a combination of different fonts and get near 100 accuracy.

3. Writing the output to the file:

For each input image file, I generate an corresponding text file containing the predicted text for the whole input image with the classifier. The text files are stored in the **output** folder.

Below are the predictions for some of the images:

GOUTES

BESURETODRINKYQUROVALTINE

XOVWIIIEACEMANXDBEEAISINIIEEBVXNEVBRDETXOURSEDEBEDEEBAIED
References:

1. https://scikit-image.org/docs/stable/api/skimage.measure.html?highlight=measure#skimage.measure.regionprops
2. https://scikit-image.org/docs/stable/api/skimage.transform.html?highlight=resize#skimage.transform.resize
3. https://scikit-image.org/docs/stable/api/skimage.measure.html?highlight=label#skimage.measure.label
4. https://scikit-image.org/docs/stable/api/skimage.morphology.html?highlight=closing#skimage.morphology.closing