Computer Vision Project
CSE 483

Spring 2023

Prepared By:

1- Tasneem Hisham Sayed 19P4152
2- Elsaeed Ahmed Elsaeed 19P1087
3- Ahmed Sameh Mourad 19P5861

Table of Contents

1. PHASE 1:

1.1 Pipeline

In phase 1, we've worked on the first format of the "train" dataset only, which has 33,402 images in total.

All the functions used in this phase are provided by the "OpenCV" python library.

1.2 Pipeline Steps

1) Read all images from the "train" dataset, using the "imread" function. The "imread" function takes the path to an image file and reads it into a NumPy array.

2) Convert each color image to its corresponding grayscale format. Grayscale images only have one channel, as opposed to RGB images which have three channels for red, green, and blue. This makes them easier to process and analyze for certain computer vision tasks. The "cvtColor" function is used here, which converts an image from one color space to another. In this case, we would use it to convert each color image to grayscale.

3) Dilating and blurring the image: Dilating using the dilate function with a 7x7 kernel, then applying a median filter using the medianBlur function with a kernel size of 21. This is done to remove any small, unwanted details and to create a background image.

4) Calculating the difference image then normalizing it: Calculating the difference between the background image using the absdiff function. It subtracts the result from 255 to invert the image, making the objects in the foreground white. Then normalizes the result using the normalize function with the NORM_MINMAX normalization type and a dtype of

CV_8UC1.

5) Apply histogram equalization to enhance the contrast or sharpness of the grayscale image. Histogram equalization is a method used to stretch the brightness levels of an image so that the entire range of brightness is utilized. This can improve the contrast of the image, making it easier to identify features. The “equalizeHist” function in OpenCV is used to perform histogram equalization on the image.

6) Apply median filtering, which is a type of noise reduction technique that works by replacing each pixel value in an image with the median value of its neighboring pixels. This helps to remove random noise from the image while preserving edges and other important features. In this code, OpenCV's fastNlMeansDenoising function is used here to apply median filtering to the image.

7) Apply a binary threshold on the equalized image, creating a binary image with white regions representing areas of the image with high intensity values, and black regions representing areas with low intensity values. The “threshold” function in OpenCV is used to create a binary image, where all pixel values greater than 200 are set to 255, and all others are set to 0.

8) Find the contours of the output threshold image. Contours are the boundaries of objects in an image. They can be used to identify and locate objects in an image. The “findContours” function in OpenCV is used to find all the contours in an image. This function takes the output of the previous step, which is the histogram equalized grayscale image, and applies a threshold to it to convert it into a binary image. Then it identifies all the contours in the binary image.

9) Draw the contours found in the previous step onto the copy of the original image. This will help visualize where the contours are in the image. The “drawContours” function in OpenCV is used to draw contours on an image. This function takes the copy to the

original image, the contours found in the previous step, the index of the contour to draw (or -1 to draw all contours), and the color and thickness of the contour.

10) Draw bounding boxes around each of the found contours, that are between a minimum & a maximum threshold that is proportionate to the size of each image. Bounding boxes are rectangles that enclose an object in an image. They can be used to crop or isolate specific objects in an image. The “rectangle” function in OpenCV is used to draw bounding boxes around the contours.

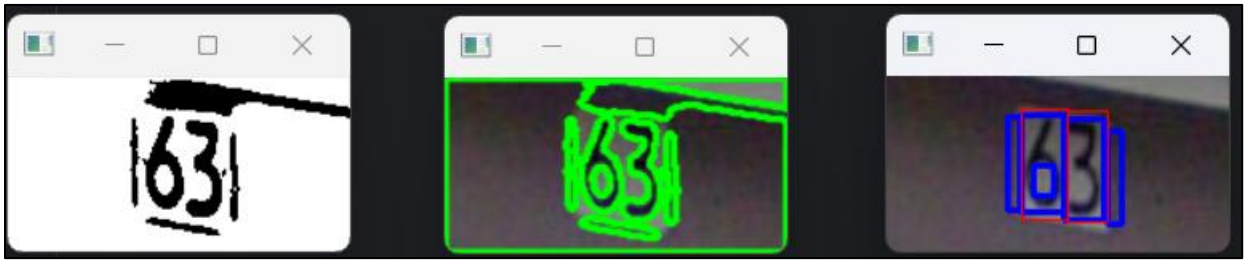
11) The final step is to calculate the accuracy of the localization by using the “intersection over union” method, by measuring the overlap between the real & predicted bounding boxes of each image. The “Intersection over Union” (IoU) is a method used to evaluate the performance of the object detection algorithms. The second method used is to calculate the accuracy uses the IoU but with a small modification that uses a black image and intersecting the found boundaries with the real boundaries.

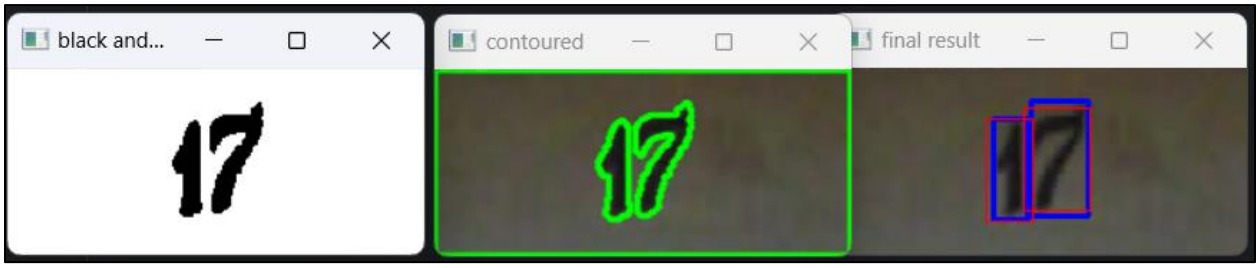
1.3 Output Screenshots:

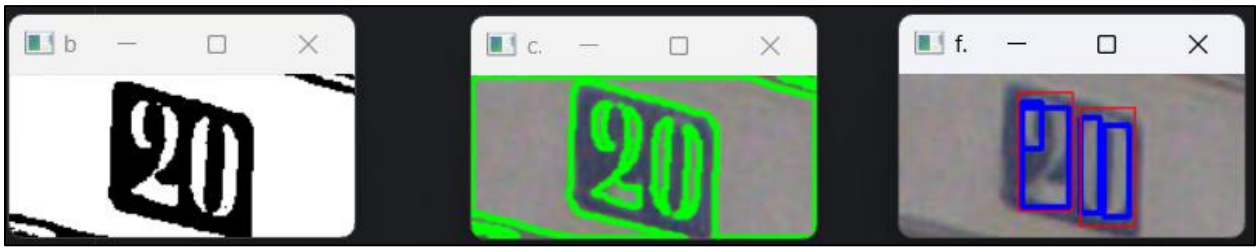
The Blue bounding boxes represent the predicted drawn boxes.

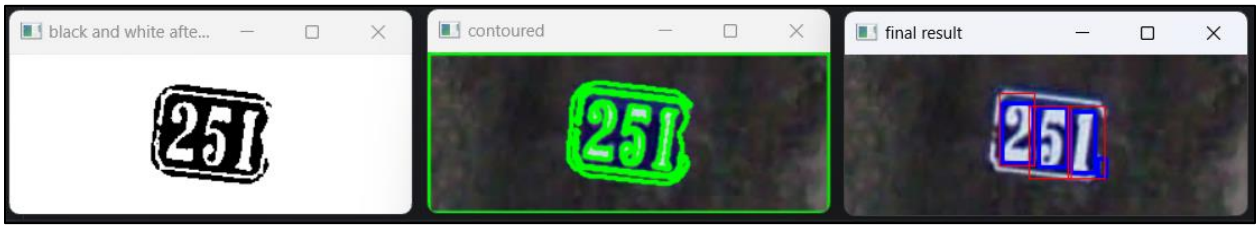
The Red bounding boxes represent the real boxes, read from the .mat file.

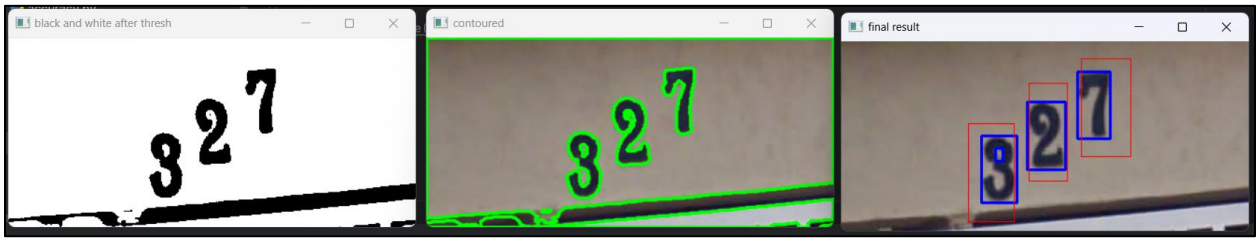
The Green outliers represent the contours drawn on the processed image.

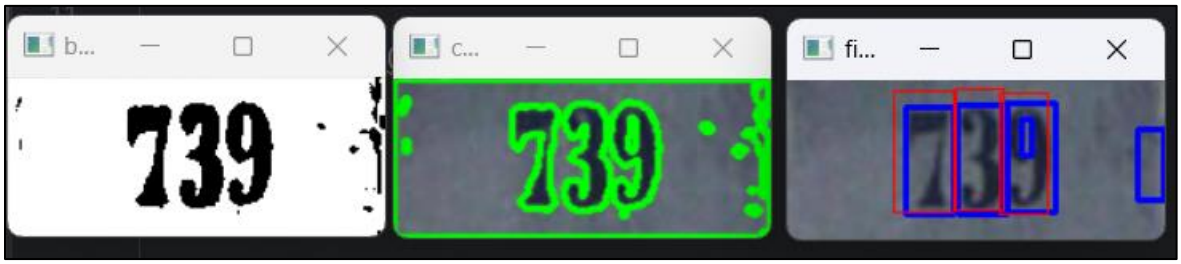



black and...
contoured
final result


b
c.
f.


black and white afte...
251
contoured
251
final result
251


black and white after thresh
327
contoured
327
final result
327


b...
739
c...
739
fi...
739

2. PHASE 2

2.1 pipeline

The code implements an image recognition pipeline that extracts features, matches them, and performs digit recognition on a set of images. It also generates images with recognized digits and calculates the accuracy of the recognition process.

2.2 pipeline steps

1. Read images:

- The `cv2.imread` function is used to read each image from the "train" dataset.
- The images are read into `image` variables.

2. Extract features:

- The `extract_features` static method is defined in the `Recognize` class.
- The SIFT (Scale-Invariant Feature Transform) algorithm is used to extract features from the images.
- If the image is in color, it is converted to grayscale using `cv2.cvtColor`.
- The SIFT algorithm detects keypoints and computes descriptors for the grayscale image.
- The keypoints and descriptors are returned from the method.

3. Match features:

- The `match_features` static method is defined in the `Recognize` class.
- The BFMatcher algorithm is used to match features between two images.
- Keypoints and descriptors are extracted from two input images using the `extract_features` method.
- The `knnMatch` function of the BFMatcher is used to find k-best matches for each descriptor.
- The matches are filtered based on a distance ratio threshold.
- Similarity scores are calculated based on the filtered matches.

- The matched features are visualized using `cv2.drawMatches`.
- The average normalized similarity score is returned.

In the `Recognize` class, the `test_images` method is defined. This method performs the following steps:

1. Load a reference image for comparison.
2. Calculate the aspect ratio of the reference image.
3. Initialize an accuracy list.

For each image, do the following:

1. Load the image and corresponding image with bounding boxes.
2. Calculate the aspect ratio and dimensions of the current image.

For each bounding box, do the following:

1. Initialize a score variable as infinity.
2. Convert the label of the bounding box to a string.
3. Extract the region of interest (ROI) based on the bounding box coordinates.

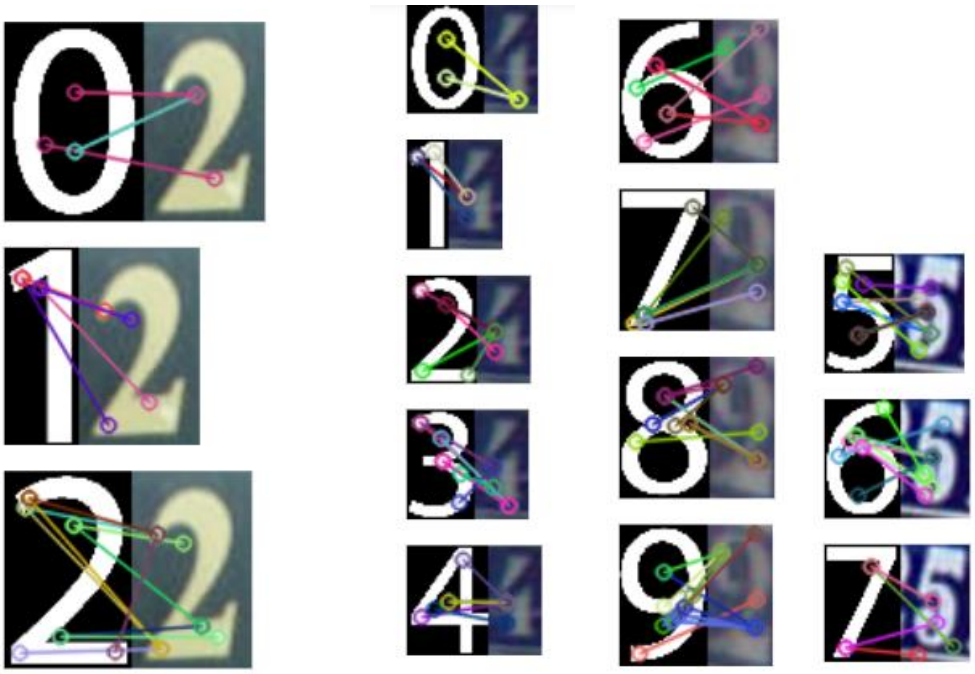
For each digit template, do the following:

1. Load the template.
2. Adjust the aspect ratio to match the ROI height.
3. Calculate the similarity between the template and ROI using the `match_features` method.
4. Update the lowest similarity score and corresponding digit label if a better match is found.

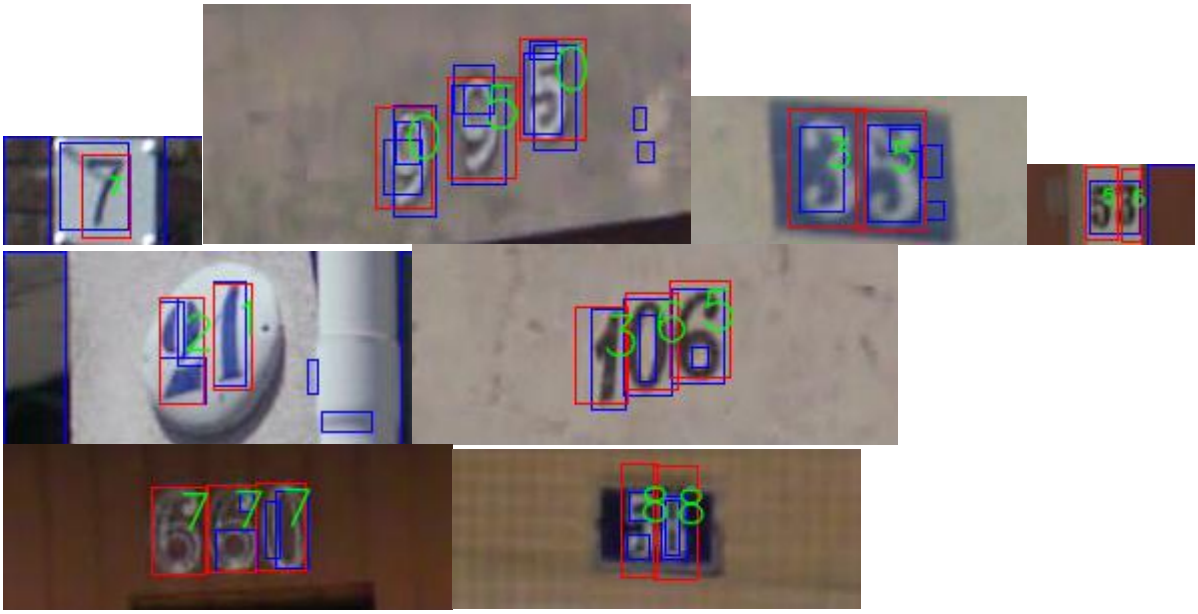
- Calculate the accuracy of digit recognition by comparing the recognized digit with the actual label.
- Draw a bounding box around the recognized digit on the image with bounding boxes using `cv2.putText`.
- Save the modified image with recognized digits.
- Calculate the accuracy of number recognition based on the accuracy list.
- Print the calculated accuracy.

2.3 Output Screenshots

Feature Matching:



Digits Recognition:



GitHub Repo Link: https://github.com/Ahmed-Sameh-MM/Computer-Vision-Project