CP 468

ASL Number Recognition

CP-468: ARTIFICIAL INTELLEGENCE

DHRUV SAGAR - 186802500

Introduction

ASL (American Sign Language) is the most used sign language. A huge portion of the population has not learnt this language creating a language barrier for the mute people. This project aims to recognize the numbers of the ASL by comparing the pixels of the image being tested to the images' dataset.

Dataset

The Dataset which I used is a collection of images found on a GitHub repo (link in the bibliography section). The images are a collection of ASL numbers in a huge quantity. The method I have used for the comparison of the images required me to transform each of these images into a black and white image manually. Thus, making it so that I have to reduce the size of the dataset as the images had to be in black and white. The program converts this dataset into a huge array of RGB codes which are then compared with the array of the image being tested.

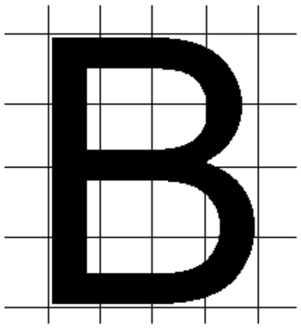
Methodology

The method I selected for this project is called the Zoning method. I had considered the density and directed density methods which are also mentioned below.

Zoning Method

The zoning method divides the image into zones while utilising the density feature. Calculating the percentage of black pixels is similar to calculating density. But in this case, it is determined for each zone. During the training phase, the symbol's features are retrieved, and the associated symbol is then found. Later, the database will record both the features and the symbol. The symbol is partitioned into 16 (4x4) areas in the example below. Next to the picture is the appropriate feature vector.

Figure 1. Symbol partitioned into 16 unique regions of their own.



[0.79, 0.46, 0.51, 0.42,

0.77, 0.31, 0.40, 0.47,

0.72, 0.14, 0.20, 0.67,

0.80, 0.46, 0.47, 0.53]

The normalised pixel density for each zone has been established. The zone-making process is:

By analysing the density each time, the computer can easily accommodate for changing angles, which helps it learn new images in a better way.

The total number of zones can be computed as follows if Z_H and Z_v are the total number of zones formed:

$$f^z(i) = \sum_{x=x_s(i)}^{x_e(i)} \sum_{y=y_s(i)}^{y_e(i)} im(x,y)$$

Where,

$$x_s(i) = \left(i - \left\lfloor \frac{i}{Z_H} \right\rfloor Z_H\right) \frac{x_{max}}{Z_H} \quad , \quad x_e(i) = \left(i - \left\lfloor \frac{i}{Z_H} \right\rfloor Z_H + 1\right) \frac{x_{max}}{Z_H}$$

And

$$y_s(i) = \left(\left\lfloor \frac{i}{Z_H} \right\rfloor \right) \frac{y_{max}}{Z_V} \quad , \quad y_e(i) = \left(\left\lfloor \frac{i}{Z_H} \right\rfloor + 1\right) \frac{x_{max}}{Z_V}$$

The following formula, where d is defined as, allows us to determine the density function of a zone.

$$d = \frac{Total\ number\ of\ information\ pixels\ in\ a\ zone}{Total\ number\ of\ pixels\ in\ a\ zone}$$

Directed Density

Another feature employed in zoning is directed density, which is quite similar to the density function computed during zoning. In this instance, the rectangle is pointed in the direction of the main axis. Direct density reachability is closed transitively to density reachability, which is an unbalanced connection. This asymmetry density is called density connectivity.

Density:

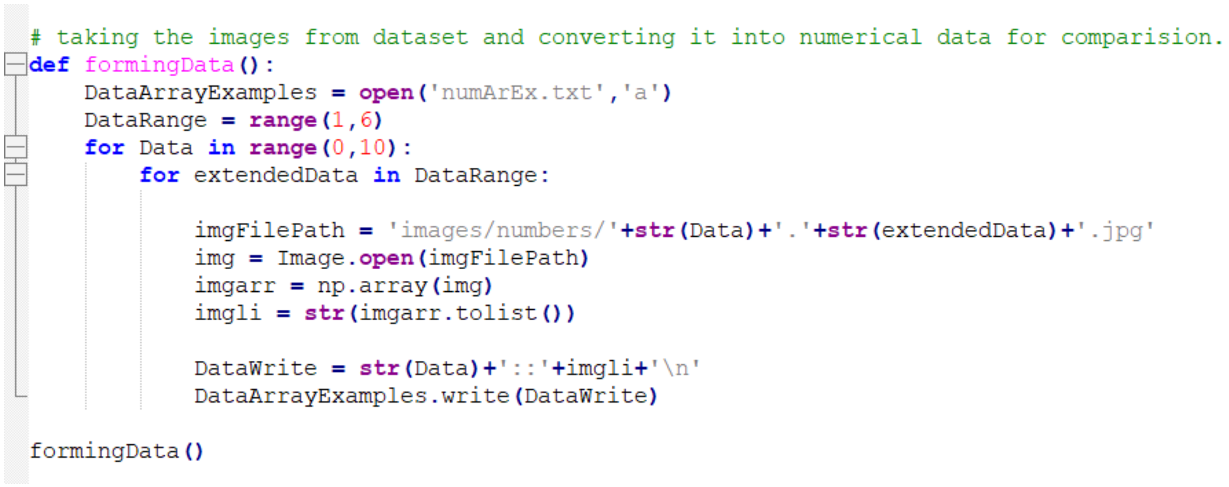
The density is a property that identifies an image by using the ratio of its pixels. The density is determined as a percentage of the total number of pixels in the image divided by the number of black pixels.

$$d = \frac{Total\ number\ of\ information\ pixels\ in\ a\ zone}{Total\ number\ of\ pixels\ in\ a\ zone}$$

With the use of this feature, we have attempted to decode the numbers in sign language.

My implemented method:

To train the code and build the arrays of the dataset's characteristics, we first downloaded a dataset from the web (see link below). We determined how many black pixels were present in each image in the collection. To accomplish this, we defined formingData() and kept the values in a text file



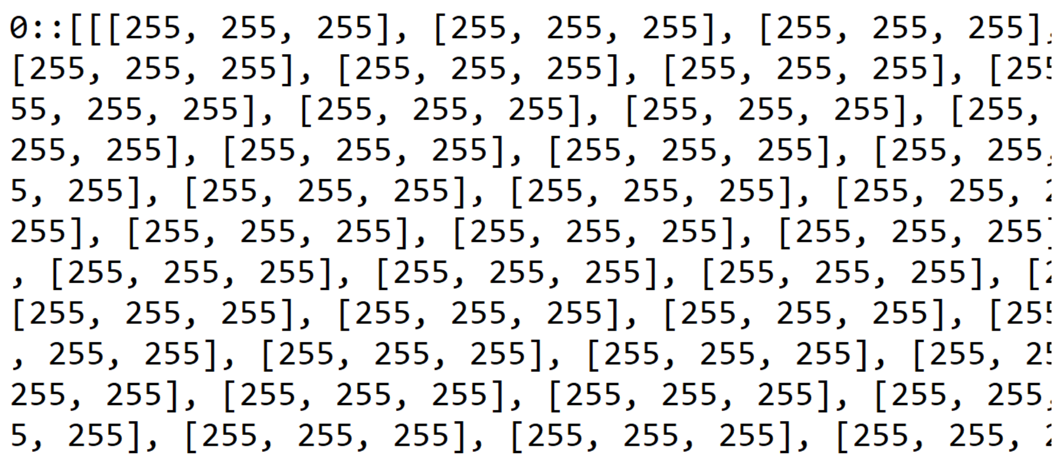
```
# taking the images from dataset and converting it into numerical data for comparision.
def formingData():
    DataArrayExamples = open('numArEx.txt','a')
    DataRange = range(1,6)
    for Data in range(0,10):
        for extendedData in DataRange:

            imgFilePath = 'images/numbers/'+str(Data)+'.'+str(extendedData)+'.jpg'
            img = Image.open(imgFilePath)
            imgarr = np.array(img)
            imgli = str(imgarr.tolist())

            DataWrite = str(Data)+'::'+imgli+'\n'
            DataArrayExamples.write(DataWrite)

formingData()
```

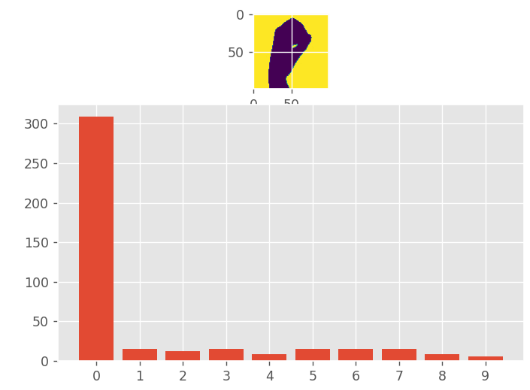
The information was presented as following:

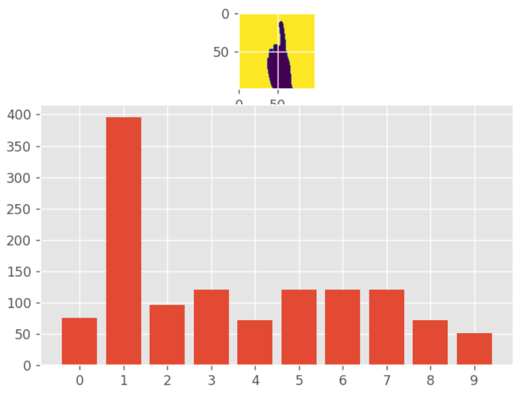


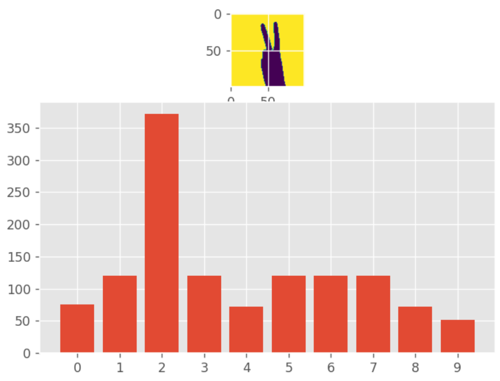
```
0::[[[255, 255, 255], [255, 255, 255], [255, 255, 255]
[255, 255, 255], [255, 255, 255], [255, 255, 255], [25
55, 255, 255], [255, 255, 255], [255, 255, 255], [255,
255, 255], [255, 255, 255], [255, 255, 255], [255, 255
5, 255], [255, 255, 255], [255, 255, 255], [255, 255,
255], [255, 255, 255], [255, 255, 255], [255, 255, 255
, [255, 255, 255], [255, 255, 255], [255, 255, 255], [
[255, 255, 255], [255, 255, 255], [255, 255, 255], [25
, 255, 255], [255, 255, 255], [255, 255, 255], [255, 2
255, 255], [255, 255, 255], [255, 255, 255], [255, 255
5, 255], [255, 255, 255], [255, 255, 255], [255, 255,
```

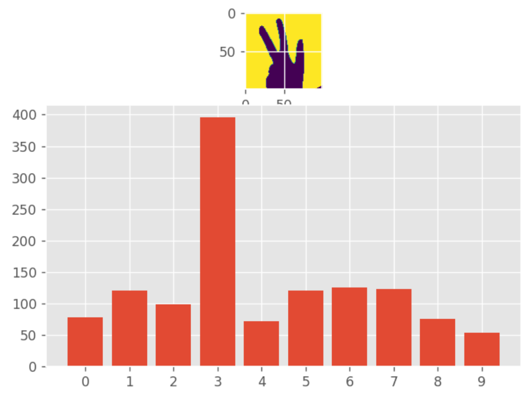
Results:

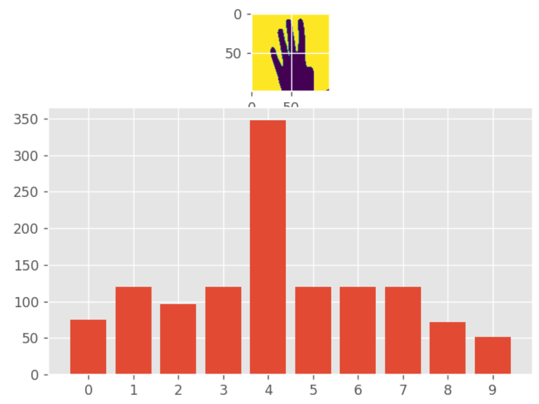
We then determined the data's average and compared it to the inputted test image. It would compare the values and show which dataset values correspond to those in the test image. With the use of a bar graph, we displayed the data and demonstrated which values were most close to the picture.

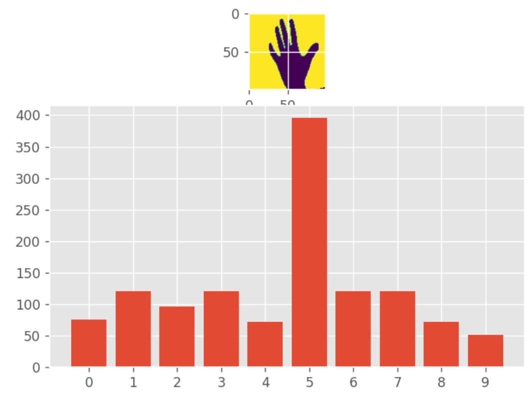

0
50
0 50
0
50
100
150
200
250
300
0 1 2 3 4 5 6 7 8 9

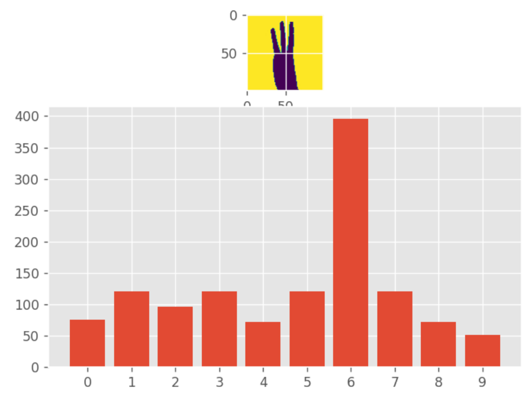

0
50
0 50
0
50
100
150
200
250
300
350
400
0 1 2 3 4 5 6 7 8 9

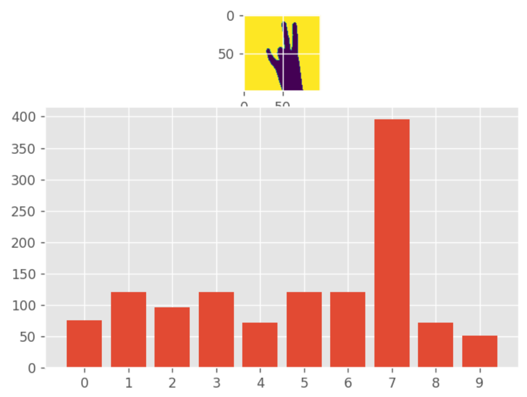

0
50
0 50
0
50
100
150
200
250
300
350
0 1 2 3 4 5 6 7 8 9

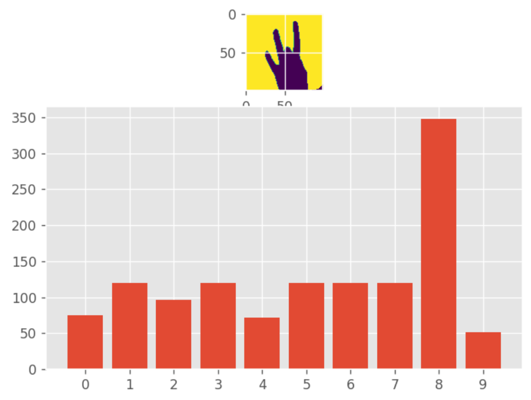

0
50
0 50
0
50
100
150
200
250
300
350
400
0 1 2 3 4 5 6 7 8 9

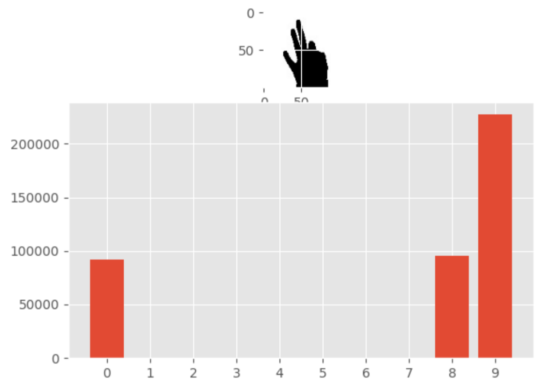

0
50
0 50
0
50
100
150
200
250
300
350
0 1 2 3 4 5 6 7 8 9


0
50
0 50
0
50
100
150
200
250
300
350
400
0 1 2 3 4 5 6 7 8 9



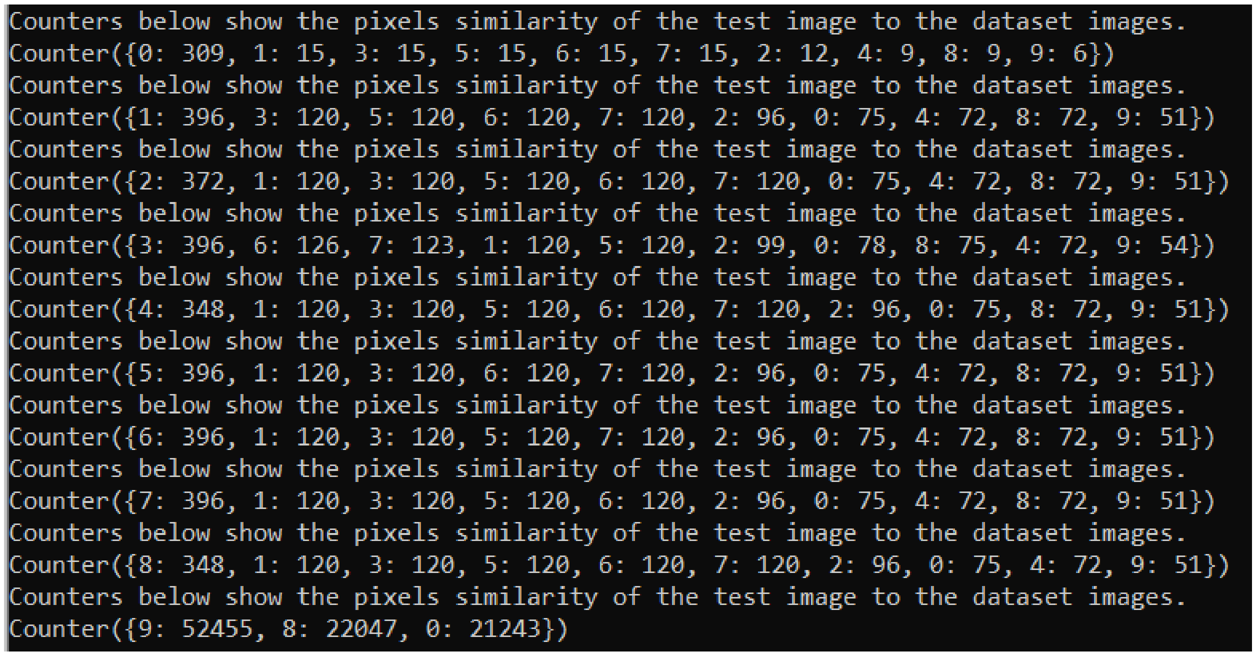






After testing for values between 0 and 9, we discovered that the numbers 6, 7, and 8 contain enormous mistakes. This happened because we only made a 2D comparison under the assumption that the black photos would be used. This also shows that the density function is not very accurate and contains many symbols that are not correctly recognised.

The counters for the comparison to each number can also be found in the terminal like below:



```
Counters below show the pixels similarity of the test image to the dataset images.
Counter({0: 309, 1: 15, 3: 15, 5: 15, 6: 15, 7: 15, 2: 12, 4: 9, 8: 9, 9: 6})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({1: 396, 3: 120, 5: 120, 6: 120, 7: 120, 2: 96, 0: 75, 4: 72, 8: 72, 9: 51})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({2: 372, 1: 120, 3: 120, 5: 120, 6: 120, 7: 120, 0: 75, 4: 72, 8: 72, 9: 51})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({3: 396, 6: 126, 7: 123, 1: 120, 5: 120, 2: 99, 0: 78, 8: 75, 4: 72, 9: 54})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({4: 348, 1: 120, 3: 120, 5: 120, 6: 120, 7: 120, 2: 96, 0: 75, 8: 72, 9: 51})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({5: 396, 1: 120, 3: 120, 6: 120, 7: 120, 2: 96, 0: 75, 4: 72, 8: 72, 9: 51})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({6: 396, 1: 120, 3: 120, 5: 120, 7: 120, 2: 96, 0: 75, 4: 72, 8: 72, 9: 51})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({7: 396, 1: 120, 3: 120, 5: 120, 6: 120, 2: 96, 0: 75, 4: 72, 8: 72, 9: 51})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({8: 348, 1: 120, 3: 120, 5: 120, 6: 120, 7: 120, 2: 96, 0: 75, 4: 72, 9: 51})
Counters below show the pixels similarity of the test image to the dataset images.
Counter({9: 52455, 8: 22047, 0: 21243})
```

Conclusion:

In conclusion, the accuracy of the algorithm is pretty solid, where the results are almost always accurate. The results seem to be extremely accurate for the digit 0 and least accurate for the digit 9, where we see a huge spike for the values 8 & 0. This is due to having limited data set since the program only accepts images that are black and white. Since the zoning method considers the density and the digit 0 having the most amount of directed density, this explains why it is the most accurate out of all. Since all the peaks within the results point out to the correct values, it can easily be used for real life applications without having to worry about inaccuracies.

Citations

- Vamvakas, G. & Gatos, B. & Pratikakis, Ioannis & Stamatopoulos, Nikolaos & Roniotis, Alexandros & Perantonis, Stavros. (2007). Hybrid off-line OCR for isolated handwritten Greek characters. 197-202.
- Ye, Q., Zeng, W., & Gao, W. (2003, August). Document Not Found. Retrieved July 03, 2022, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.901.7252
- Mavi, A. (2017, December 18). Ardamavi/Sign-Language-Digits-Dataset. Retrieved July 04, 2022, from https://github.com/ardamavi/Sign-Language-Digits-Dataset
- S. (Director). (2018). Image Recognition and Python Part 1 [Video file]. Youtube. Retrieved July 08, 2022, from https://www.youtube.com/watch?v=hbL_FTEZSyY