National University
of computer and emerging sciences

Course: Machine Learning

Title: Hand Gesture Image Processing

Introduction:

The hand gesture image processing contributions I made are summarized in this contribution report. My primary responsibility in this project was image preprocessing using Python and libraries like numpy, matplotlib, pandas and cv2 for fundamental functions.

Project Overview:

The project's goal was to analyze image of hand gesture and pull out pertinent data for further investigation. The main duties were to load, do the necessary preprocessing, and separate the foreground and background of the hand gesture.

Function & Duties I Have

1. **Hand Gesture Image Loading:** Using the cv2 library, I loaded an image from a hand gesture dataset that is available on Kaggle.
2. **Image resizing:** I resized the hand gesture image and reduced its size to optimize it for analysis.
3. **Grayscale Conversion:** I used NumPy to convert the image to grayscale using Luminance Formula, which made it two-dimensional and made the further processing easier.
4. **Seperating foreground from background:** Utilizing NumPy, I put adaptive thresholding technique into practice to separate the hand gesture from the background.
5. **Cropping:** In order to focus on the hand region, images are cropped to only maintain the relevant portions with high data points that reflect hand gesture.
6. **Extract 100 Data points:** To identify the most informative rows and columns, the cropped image was inspected. I then extracted 100 data points from the row or column that had the most valuable data (the darkest points).

Code Contributions:

1. **Loading Image:**

```
def loading_image(path,size):
    rgb_image = cv2.imread(path)
    if rgb_image is None:
        print("Error: Unable to load image.")
```

2. **Resizing Image:**

```
  else:
      resized_rgb_image=cv2.resize(rgb_image,size)
```

*Note:Also done resizing without cv2 by using simple python

3. **Grayscale Image:**

```
grayscale_image =( 0.299 *resized_rgb_image[:,:,0] + 0.587 *resized_rgb_image[:,:,1] + 0.114 *resized_rgb_image[:,:,2]).astype(np.uint8)
```

✓ **Overall Code Loading , Resizing and Gray scaling:**

```
def loading_image(path,size):

    rgb_image = cv2.imread(path)

    if rgb_image is None:
        print("Error: Unable to load image.")
```

```
    else:
        resized_rgb_image=cv2.resize(rgb_image,size)

        cv2.imshow('Loaded Image', resized_rgb_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

        grayscale_image =( 0.299 *resized_rgb_image[:,:,0] + 0.587 *resized_rgb_image[:,:,1]
+ 0.114 *resized_rgb_image[:,:,2]).astype(np.uint8)

        cv2.imshow('Grayscale Image', grayscale_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

        return grayscale_image

path = "ok.jpeg"
size=(224,224)
gray_scale_image = loading_image(path,size)
binaryimage=adaptive_thresholding(gray_scale_image)
```

4. adaptive thresholding:

```
def adaptive_thresholding(gray_scale_image):
```

```
    gridSize = 11
    offset = 2

    binary_image = np.zeros_like(gray_scale_image, dtype=np.uint8)

    for y in range(gridSize// 2, gray_scale_image.shape[0] - gridSize // 2):
        for x in range(gridSize // 2, gray_scale_image.shape[1] - gridSize // 2):

            start_r=y - gridSize // 2
            end_r=y + gridSize // 2 + 1
            start_c=x - gridSize // 2
            end_c=x + gridSize // 2 + 1

            grid = gray_scale_image[start_r:end_r,start_c:end_c ]
            gridMean = np.mean(grid)
            thresh = gridMean - offset
            binary_image[y, x] = 255 if gray_scale_image[y, x] < thresh else 0
    return binary_image
```

5. Taking out darkest data points:

```
darkest_data_points = np.where(binaryimage == 255, binaryimage, np.nan)
```

6. Cropping image:

```
hand_cropped = darkest_data_points
row_indices, col_indices = np.where(hand_cropped  == 255)
min_row=np.min(row_indices)
min_col =np.min(col_indices)
max_row=np.max(row_indices)
max_col= np.max(col_indices)

hand_cropped =hand_cropped[min_row:max_row + 1, min_col:max_col + 1]
```

7. Finding 100 data points :

```
def count_255(row_or_col):
    return np.sum(row_or_col == 255)

count_inR=[]
count_inC=[]
for row in hand_cropped:
    count_inR.append(count_255(row))
for col in hand_cropped.T:
    count_inC.append(count_255(col))
best_R_ind = np.argmax(count_inR)
best_C_ind = np.argmax(count_inC)

best_R = hand_cropped[best_R_ind, :100]
```

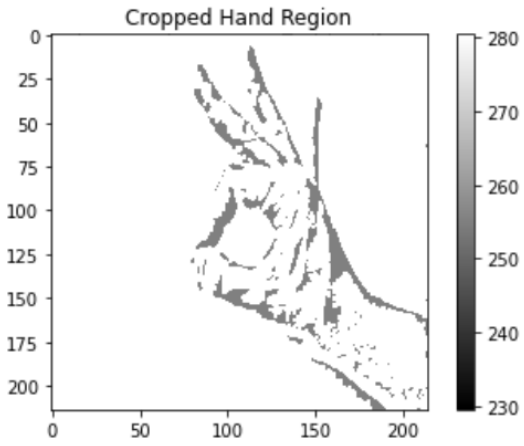
best_C =hand_cropped[:100, best_C_ind]

Results:

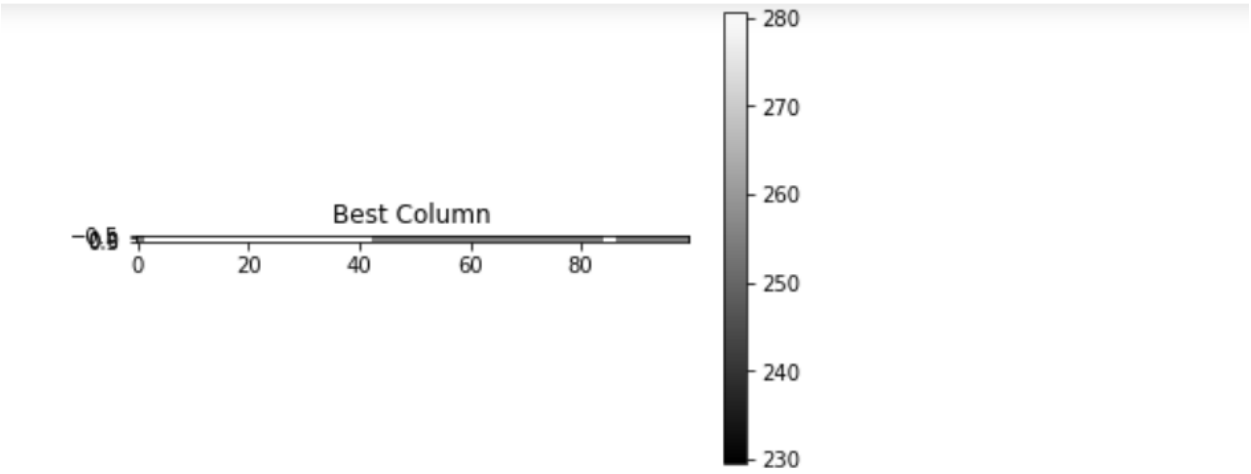
Original Image:

After applying Thresholding:

Cropping:



Taking 100 Data points:




```
]:  1 temp_data
```

```
]: array([255., 255.,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
          nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
          nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
          nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan, 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., 255., 255., 255., 255., 255., 255.,  nan, 255., 255.,
         255., 255., 255., 255., 255., 255., 255., 255., 255., 255., 255.,
         255.])
```