Cp467 A3 report

By Phoebe Schulman, Afaq Shad, Arvind Sahota

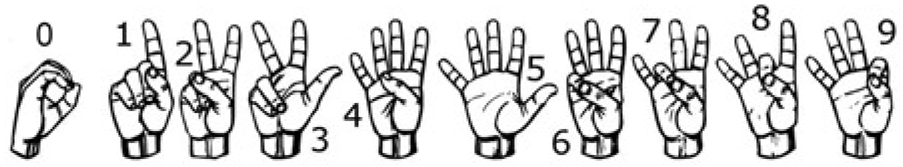
Abstract

In this project, the aim was to write a Python program to extract 3 features for each image of a sign language digit:

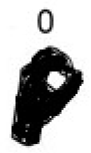
- The first feature x: the aspect ratio of an image (the ratio of its width to its height).
- The second feature y: the ratio of the left half black pixels area to the rectangle that has the image.
- The third feature z: the ratio of the right half black pixels area to the rectangle that has the image.

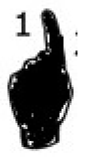
Introduction

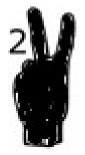
Before the image processing, we were given an image with sign language symbols.



Using the software Paint, these were manually converted into 10 images and filled in. For example:





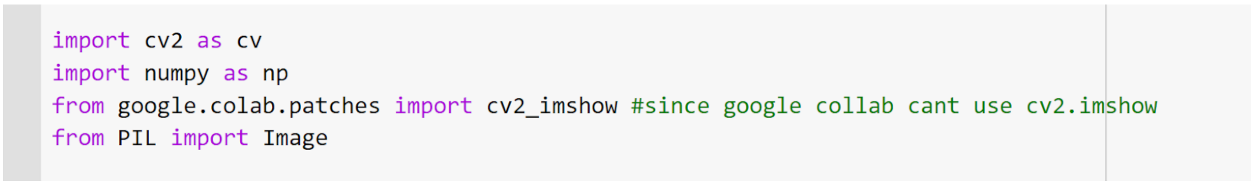


The assumptions we made were:

- Numbers and hands can stay in the picture. For example, the digit 0 is part of the image.
- Assumes images were properly filled in and split by hand.

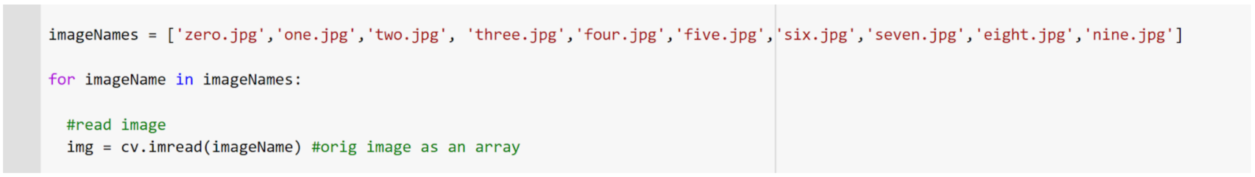
Developed system

How our program works is by using Python API's including cv2, numpy and PIL.



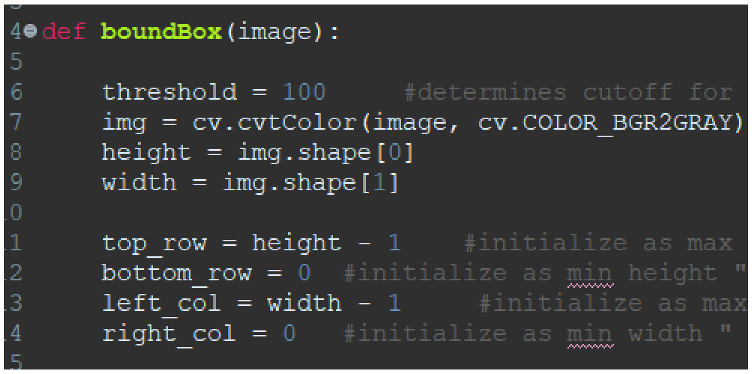
```python
import cv2 as cv
import numpy as np
from google.colab.patches import cv2_imshow #since google collab cant use cv2.imshow
from PIL import Image
```

1. First, we store the images as an array of shape (nx, ny, nz).



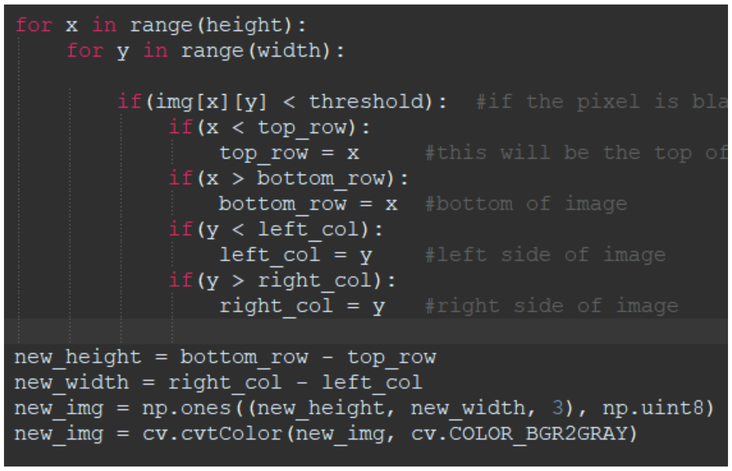
```python
imageNames = ['zero.jpg','one.jpg','two.jpg', 'three.jpg','four.jpg','five.jpg','six.jpg','seven.jpg','eight.jpg','nine.jpg']

for imageName in imageNames:

  #read image
  img = cv.imread(imageName) #orig image as an array
```

2. Next, we find the minimum bounding box (rectangle) that contains the image:



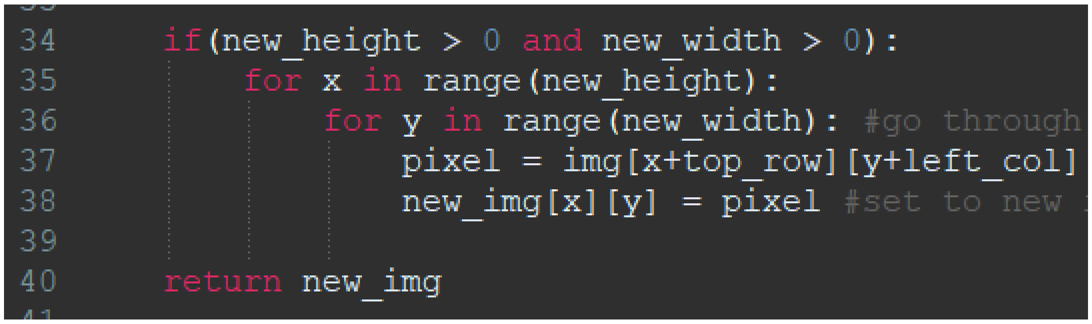
```python
def boundBox(image):

    threshold = 100     #determines cutoff for
    img = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    height = img.shape[0]
    width = img.shape[1]

    top_row = height - 1     #initialize as max
    bottom_row = 0   #initialize as min height "
    left_col = width - 1      #initialize as max
    right_col = 0   #initialize as min width "
```

We start the function by applying a greyscale and obtaining the height and width of the image. We set a threshold pixel value which will determine the bounding box of the image. The smaller the threshold, the tighter the bounding box and vice versa. We also set top_row, bottom_row, left_col, and right_col as the four values that make the shape of the bounding box. We set the final row (height - 1) to top_row since we want it to eventually be as low as possible (the smaller the row number, the closer to the top it is). We do the opposite for width. We apply the same logic to left_col and right_col but for columns instead.



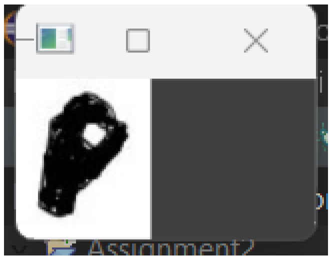
```
for x in range(height):
    for y in range(width):

        if(img[x][y] < threshold):  #if the pixel is bla
            if(x < top_row):
                top_row = x     #this will be the top of
            if(x > bottom_row):
                bottom_row = x  #bottom of image
            if(y < left_col):
                left_col = y    #left side of image
            if(y > right_col):
                right_col = y   #right side of image

new_height = bottom_row - top_row
new_width = right_col - left_col
new_img = np.ones((new_height, new_width, 3), np.uint8)
new_img = cv.cvtColor(new_img, cv.COLOR_BGR2GRAY)
```

In the for loop, we go through the image to find the four borders of what will become the bounding box. If the current pixel is below the threshold (an acceptable level for a black pixel), we update our borders if that pixel is an improvement from the current one (less than the previous top_row, more than the previous bottom_row, etc). Once the loop is done and we have our border, we obtain the new height and width for the bounding box and create the image.

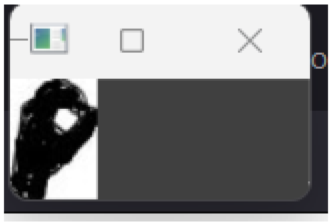


```
    if(new_height > 0 and new_width > 0):
        for x in range(new_height):
            for y in range(new_width): #go through
                pixel = img[x+top_row][y+left_col]
                new_img[x][y] = pixel #set to new

    return new_img
```

Finally, we go through the new blank image and copy and paste the corresponding pixel from the original image onto the bounding box image. Using the zero hand gesture image we get the following results:



Original image



Bounded Box

Here is how we use the bounding box function:

```
#find min bounding box
bound = boundBox(img)
img = bound
img_orig = Image.fromarray(img)
```

3. We can find the image ratio, which is simply width : height.

```
#1- The aspect ratio. (This is the first feature x). The aspect ratio of an image is the ratio of its width to its height
height = img.shape[0]
width = img.shape[1]
print("calculate aspect ratio\n",width,":",height,"\n")

s = width,":",height
x.append( s)
x_percent.append( round(width/height,3))
```

4. Now we count the number of black pixels in the left/right half, compared to the whole image.

First, we can split the image in half *(4).*

```
# Cut the image into left and right half
width_cutoff = width // 2
s1 = img[:, :width_cutoff]
s2 = img[:, width_cutoff:]

print("left pic")
cv2_imshow(s1)
print("right pic")
cv2_imshow(s2)

img_from_arr_left  = Image.fromarray(s1)
img_from_arr_right  = Image.fromarray(s2)
```

Then we can count the black pixels in the original picture, the left picture, and the right picture.

```
#count black pixels
count_black_orig = counter(img_orig)#(img)
#print("number of black pixles in orig pic", count_black_orig)


count_black_left =  counter(img_from_arr_left)#(s1)
print("number of left black pixles: all black pixles\n", count_black_left,":", count_black_orig)
s = count_black_left,":", count_black_orig
y.append( s)
y_percent.append( round(count_black_left/count_black_orig,3))

count_black_right =   counter(img_from_arr_right)
print("number of right black pixles: all black pixles\n", count_black_right,":", count_black_orig)
s = count_black_right,":", count_black_orig
z.append( s)
z_percent.append( round(count_black_right/count_black_orig,3))
```

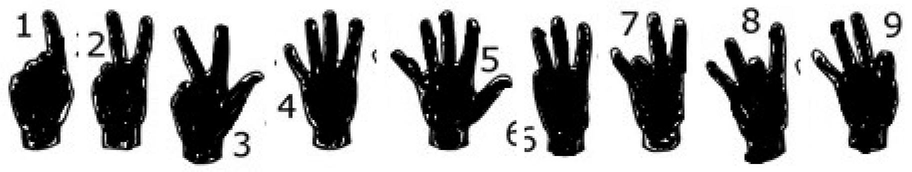
This uses 2 helper functions: counter for counting black pixels, and get_pixel for getting a pixel's colours *(5).*

```
# count num of black pixles
def counter(img):
  count_black= 0
  for i in range (img.size[0]):
    for j in range (img.size[1]):
      if get_pixel(img, i,j) == 0:#(0,0,0): #if its black pixle
        count_black +=1

  return count_black
```

```
# Get the pixel from the given image #get colours
def get_pixel(image, i, j):
    # Inside image bounds?
    width, height = image.size
    if i > width or j > height:
      return None

    # Get Pixel
    pixel = image.getpixel((i, j))
    return pixel
```

Results

Our input was these images:



Our output for the first picture:

```
orig pic zero.jpg
```

```
calculate aspect ratio
 35 : 67

left pic
```

```
right pic
```

```
number of left black pixles: all black pixles
 301 : 462
number of right black pixles: all black pixles
 161 : 462
```

Output for the rest of the digits is similar:

```
orig pic nine.jpg
```

```
calculate aspect ratio
 43 : 69

left pic
```

```
right pic
```

```
number of left black pixles: all black pixles
 235 : 911
number of right black pixles: all black pixles
 676 : 911
```

Finally, we have the x, y, and z features.

```
 X Y Z

[(35, ':', 67), (34, ':', 68), (31, ':', 68), (46, ':', 68), (50, ':', 68), (62, ':', 69), (33, ':', 69), (37, ':', 69), (46, ':', 77), (43, ':', 69)]
[(301, ':', 462), (316, ':', 707), (289, ':', 649), (619, ':', 909), (272, ':', 808), (504, ':', 928), (238, ':', 769), (152, ':', 749), (300, ':', 872), (235, ':', 911)]
[(161, ':', 462), (391, ':', 707), (360, ':', 649), (290, ':', 909), (536, ':', 808), (424, ':', 928), (531, ':', 769), (597, ':', 749), (572, ':', 872), (676, ':', 911)]

 X Y Z as percents

[0.522, 0.5, 0.456, 0.676, 0.735, 0.899, 0.478, 0.536, 0.597, 0.623]
[0.652, 0.447, 0.445, 0.681, 0.337, 0.543, 0.309, 0.203, 0.344, 0.258]
[0.348, 0.553, 0.555, 0.319, 0.663, 0.457, 0.691, 0.797, 0.656, 0.742]
```

	x	y	z
0	0.522	0.652	0.348
1	0.5	0.447	0.553
2	0.456	0.445	0.555
3	0.676	0.681	0.319
4	0.735	0.337	0.663
5	0.899	0.543	0.457

6	0.478	0.309	0.691
7	0.536	0.203	0.797
8	0.597	0.344	0.656
9	0.623	0.258	0.742

Sources

1. Aspect_ratio.py by Integralist:
https://gist.github.com/Integralist/4ca9ff94ea82b0e407f540540f1d8c6c

2. Pillow - Python Imaging Library (aka PIL):
http://2017.compciv.org/guide/topics/python-nonstandard-libraries/pillow.html

3. Calculate the pixel ratio of two halves of an image:
https://stackoverflow.com/questions/72623020/calculate-the-pixel-ratio-of-two-halves-of-an-image

4. How to Cut an Image Vertically into Two Equal Sized Images:
https://stackoverflow.com/questions/45384968/how-to-cut-an-image-vertically-into-two-equal-sized-images

5. Image Manipulation in Python by Isai B. Cicourel:
https://www.codementor.io/@isaib.cicourel/image-manipulation-in-python-du1089j1u

6. Convert a NumPy Array to an image in Python By Mohan:
https://thispointer.com/convert-a-numpy-array-to-an-image-in-python/#:~:text=The%20Approach%20to%20convert%20NumPy,using%20the%20save()%20method