California State University, Long Beach
Department of Computer Engineering and Computer Science
CECS 553 Sec 02 11792 (Machine Vision) – Fall 2022
Assignment 04 – Tuesday, 09/27/2022

Histogram and Intensity Transformations

Estimated time needed: **40** minutes

Objectives

Pixel Transforms are operations you perform one pixel at a time. In this lab, you will start by creating histograms. Histograms display the intensity of the image and can be used to optimize image characteristics. You will then apply Intensity Transformations, making objects easier to see by improving image contrast and brightness. In the last portion of the lab, you will use thresholding to segment objects from images.

- Pixel Transforms
  - Histograms
  - Intensity Transformations
  - Thresholding and Simple Segmentation

Download the image for the lab

```
In [ ]: !wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
        !wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
        !wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
        !wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
        !wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
        !wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
```

We will be using these imported functions in the lab

```
In [ ]: import matplotlib.pyplot as plt
        import cv2
        import numpy as np
```

First, lets define a helper function to plot two images side-by-side. You will not need to understand this code at this moment, but this function will be used repeatedly in this tutorial to showcase the results.

```
In [ ]: def plot_image(image_1, image_2,title_1="Orignal", title_2="New Image"):
            plt.figure(figsize=(10,10))
            plt.subplot(1, 2, 1)
            plt.imshow(image_1,cmap="gray")
            plt.title(title_1)
            plt.subplot(1, 2, 2)
            plt.imshow(image_2,cmap="gray")
            plt.title(title_2)
            plt.show()
```

Lets define another helper function. The following one will plot two histograms side-by-side. Again, you do not need to understand the body of this function at this moment.

```
In [ ]: def plot_hist(old_image, new_image,title_old="Orignal", title_new="New Image
            intensity_values=np.array([x for x in range(256)])
            plt.subplot(1, 2, 1)
            plt.bar(intensity_values, cv2.calcHist([old_image],[0],None,[256],[0,256
            plt.title(title_old)
            plt.xlabel('intensity')
            plt.subplot(1, 2, 2)
            plt.bar(intensity_values, cv2.calcHist([new_image],[0],None,[256],[0,256
            plt.title(title_new)
            plt.xlabel('intensity')
            plt.show()
```

Histograms

A histogram counts the number of occurrences of the intensity values of pixels, and it's a useful tool for understanding and manipulating images. We use `cv.calcHist()` to generate the histogram. Here are the parameter values:

`cv2.calcHist(CV array:`**`[image]`**` this is the image channel:`**`[0]`**`,for this course it will always be `**`[None]`**`,the number of bins:`**`[L]`**`,the range of index of bins:`**`[0,L-1]`**`)`

For real images, L is 256.

Toy Example

Consider the toy array with intensity values ranging from 0 to 2. We can create a histogram. Its first element is the number of zeros in the image (in this case, 1); its second element is the number of ones in the image (in this case, 5), and so on.

```
In [ ]: toy_image = np.array([[0,2,2],[1,1,1],[1,1,2]],dtype=np.uint8)
        plt.imshow(toy_image, cmap="gray")
        plt.show()
        print("toy_image:",toy_image)
```

We can use the `caclHist` function, in this case, we use only three bins as there are only three values, and the index of the bins are from 1 to 3.

TODO: @Joe

```
In [ ]: plt.bar([x for x in range(6)],[1,5,2,0,0,0])
        plt.show()
```

```
In [ ]: plt.bar([x for x in range(6)],[0,1,0,5,0,2])
        plt.show()
```

The histogram is a function where $h\[r]$ where $r \in {0,1,2} $. In the above example $h\[0]=1$,$h\[1]=5$ and $h\[2]=3$

Gray Scale Histograms

Histograms are used in grayscale images. Grayscale images are used in many applications, including medical and industrial. Color images are split into luminance and chrominance. The luminance is the grayscale portion and is usually processed in many applications. Consider the following "Gold Hill" image:

```
In [ ]: goldhill = cv2.imread("goldhill.bmp",cv2.IMREAD_GRAYSCALE)
        plt.figure(figsize=(10,10))
        plt.imshow(goldhill,cmap="gray")
        plt.show()
```

We can calculate the histogram using the `calcHist` function from the `cv2` module as follows, the shape is 256.

```
In [ ]: hist = cv2.calcHist([goldhill],[0], None, [256], [0,256])
```

We can plot it as a bar graph, the x-axis are the pixel intensities and the y-axis is the number of times of occurrences that the corresponding pixel intensity value on x-axis occurred.

```
In [ ]: intensity_values = np.array([x for x in range(hist.shape[0])])
        plt.bar(intensity_values, hist[:,0], width = 5)
        plt.title("Bar histogram")
        plt.show()
```

The histogram is a function where $h\[r]$ where $r \in {0,1,..,255} $.

We can convert it to a probability mass function by normalizing it by the number of pixels:

```
In [ ]: PMF = hist / (goldhill.shape[0] * goldhill.shape[1])
```

We can plot as a continuous function:

```
In [ ]: plt.plot(intensity_values,hist)
        plt.title("histogram")
        plt.show()
```

We can also apply a histogram to each image color channel:

```
In [ ]: baboon = cv2.imread("baboon.png")
        plt.imshow(cv2.cvtColor(baboon,cv2.COLOR_BGR2RGB))
        plt.show()
```

In the loop, the value for `i` specifies what color channel `calcHist` is going to calculate the histogram for.

```
In [ ]: color = ('blue','green','red')
        for i,col in enumerate(color):
            histr = cv2.calcHist([baboon],[i],None,[256],[0,256])
            plt.plot(intensity_values,histr,color = col,label=col+" channel")

            plt.xlim([0,256])
        plt.legend()
        plt.title("Histogram Channels")
        plt.show()
```

Intensity Transformations

It's helpful to think of an image as a function $f(x,y)$ instead of an array at this point, where `x` is the row index and `y` is the column index. You can apply a transformation T to the image and get a new image: $$ g(x,y)=T(f(x,y)) $$

An Intensity Transformation depends on only one single point (x,y). For example, you can apply a linear transform $g(x,y) = 2f(x,y) + 1$; this will multiply each image pixel by two and add one.

As the Intensity transforms only depend on one value; as a result, it is sometimes referred to as a gray-level mapping. The variable if r is the gray level intensity, similar to the histogram values. The new output s is given by:

$$ s=T(r) $$

Image Negatives

Consider an image with L intensity values ranging from $\[0,L-1]$. We can reverse the intensity levels by applying the following: $$ g(x,y)=L-1-f(x,y) $$

Using the intensity transformation function notation $$ s = L - 1 - r $$

This is called the image negative. For $L= 256$ the formulas simplifys to: $$ g(x,y)=255-f(x,y) \qquad \mbox{and} \qquad s=255-r $$

We can perform intensity transformation on the toy image where $L = 3$:

In [ ]:
```
neg_toy_image = -1 * toy_image + 255

print("toy image\n", neg_toy_image)
print("image negatives\n", neg_toy_image)
```

We see darker intensity’s become brighter and brighter become darker, middle intensity’s remain the same.

In [ ]:
```
plt.figure(figsize=(10,10))
plt.subplot(1, 2, 1)
plt.imshow(toy_image,cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(neg_toy_image,cmap="gray")
plt.show()
print("toy_image:",toy_image)
```

Reversing image intensity has many applications, including making it simpler to analyze medical images. Consider the mammogram with micro-calcifications on the upper quadrant:

```
In [ ]: image = cv2.imread("mammogram.png", cv2.IMREAD_GRAYSCALE)
        cv2.rectangle(image, pt1=(160, 212), pt2=(250, 289), color = (255), thicknes

        plt.figure(figsize = (10,10))
        plt.imshow(image, cmap="gray")
        plt.show()
```

We can apply the intensity transformation:

```
In [ ]: img_neg = -1 * image + 255
```

We see the micro-calcifications in the image negatives is easier it is to analyze:

```
In [ ]: plt.figure(figsize=(10,10))
        plt.imshow(img_neg, cmap = "gray")
        plt.show()
```

Brightness and contrast adjustments

We can use multiplication by α for contrast control and addition by β to improve brightness control. This applies the Intensity Transformation as well. The image is $f(x,y)$ and the transformed image is $g(x,y)$, where $g(x,y) = \alpha f(x,y) + \beta$.

Rather than implementing via array operations, we use the function `convertScaleAbs`. It scales, calculates absolute values, and converts the result to 8-bit so the values fall between $\[0,255]$. For brightness control, we can set α to 1 and β to 100: Remember the Good Hill image, it’s dark and hazy so let's see if we can improve it.

```
In [ ]: alpha = 1 # Simple contrast control
        beta = 100   # Simple brightness control
        new_image = cv2.convertScaleAbs(goldhill, alpha=alpha, beta=beta)
```

We can plot the brighter image, it's much brighter :

```
In [ ]: plot_image(goldhill, new_image, title_1 = "Orignal", title_2 = "brightness c
```

We see the brighter image's histogram is shifted:

```
In [ ]: plt.figure(figsize=(10,5))
        plot_hist(goldhill, new_image, "Orignal", "brightness control")
```

We can increase the contrast by increasing α:

```
In [ ]: plt.figure(figsize=(10,5))
        alpha = 2# Simple contrast control
        beta = 0 # Simple brightness control   # Simple brightness control
        new_image = cv2.convertScaleAbs(goldhill, alpha=alpha, beta=beta)
```

We can plot the image and its corresponding histogram:

```
In [ ]: plot_image(goldhill,new_image,"Orignal","contrast control")
```

```
In [ ]: plt.figure(figsize=(10,5))
        plot_hist(goldhill, new_image,"Orignal","contrast control")
```

When plotting the image we see it's too bright. We can adapt the brightness by making the image darker and increasing the contrast at the same time.

```
In [ ]: plt.figure(figsize=(10,5))
        alpha = 3 # Simple contrast control
        beta = -200  # Simple brightness control
        new_image = cv2.convertScaleAbs(goldhill, alpha=alpha, beta=beta)
```

```
In [ ]: plot_image(goldhill, new_image, "Orignal", "brightness & contrast control")
```

```
In [ ]: plt.figure(figsize=(10,5))
        plot_hist(goldhill, new_image, "Orignal", "brightness & contrast control")
```

There are other nonlinear methods to improve contrast and brightness, these methods have different sets of parameters. In general, it's difficult to manually adjust the contrast and brightness parameter, but there are algorithms that improve contrast automatically.

Histogram Equalization

Histogram Equalization increases the contrast of images, by stretching out the range of the grayscale pixels; It does this by flatting the histogram. We simply apply the function `cv2.equalizeHist`.

```
In [ ]: zelda = cv2.imread("zelda.png",cv2.IMREAD_GRAYSCALE)
        new_image = cv2.equalizeHist(zelda)
```

We can compare the image before and after Histogram Equalization, we see the contrast is improved. We see after the Histogram Equalization is applied, the histogram is stretched out:

In [ ]:
```
plot_image(zelda,new_image,"Orignal","Histogram Equalization")
```

In [ ]:
```
plt.figure(figsize=(10,5))
plot_hist(zelda, new_image,"Orignal","Histogram Equalization")
```

Thresholding and Simple Segmentation

Thresholding is used in image segmentation this means extracting objects from an image. Image segmentation is used in many applications including extracting text, medical imaging, and industrial imaging. Thresholding an image takes a threshold; If a particular pixel (i,j) is greater than that threshold it will set that pixel to some value usually 1 or 255, otherwise, it will set it to another value, usually zero. We can write a Python function that will perform thresholding and output a new image given some input grayscale image:

In [ ]:
```
def thresholding(input_img,threshold,max_value=255, min_value=0):
    N,M=input_img.shape
    image_out=np.zeros((N,M),dtype=np.uint8)

    for i  in range(N):
        for j in range(M):
            if input_img[i,j]> threshold:
                image_out[i,j]=max_value
            else:
                image_out[i,j]=min_value

    return image_out
```

Consider the following toy image:

In [ ]:
```
toy_image
```

We can apply thresholding, by setting all the values less than two to zero.

In [ ]:
```
threshold = 1
max_value = 2
min_value = 0
thresholding_toy = thresholding(toy_image, threshold=threshold, max_value=ma
thresholding_toy
```

We can compare the two images. In the new image we see all the gray values are now black:

```
In [ ]: plt.figure(figsize=(10, 10))
        plt.subplot(1, 2, 1)
        plt.imshow(toy_image, cmap="gray")
        plt.title("Original Image")
        plt.subplot(1, 2, 2)
        plt.imshow(thresholding_toy, cmap="gray")
        plt.title("Image After Thresholding")
        plt.show()
```

Consider the cameraman image:

```
In [ ]: image = cv2.imread("cameraman.jpeg", cv2.IMREAD_GRAYSCALE)
        plt.figure(figsize=(10, 10))
        plt.imshow(image, cmap="gray")
        plt.show()
```

We can see the histogram as two peeks, this means that there is a large proportion of pixels in those two ranges:

```
In [ ]: hist = cv2.calcHist([goldhill], [0], None, [256], [0, 256])
        plt.bar(intensity_values, hist[:, 0], width=5)
        plt.title("Bar histogram")
        plt.show()
```

The cameraman corresponds to the darker pixels, therefore we can set the Threshold in such a way as to segment the cameraman. In this case, it looks to be slightly less than 90, let's give it a try:

```
In [ ]: threshold = 87
        max_value = 255
        min_value = 0
        new_image = thresholding(image, threshold=threshold, max_value=max_value, mi
```

We see the pixels corresponding to the cameraman are now zero and the rest are set to 255:

```
In [ ]: plot_image(image, new_image, "Orignal", "Image After Thresholding")
```

```
In [ ]: plt.figure(figsize=(10,5))
        plot_hist(image, new_image, "Orignal", "Image After Thresholding")
```

The function `cv.threshold` Applies a threshold to the gray image, with the following parameters:

`cv.threshold(grayscale image, threshold value, maximum value to use, thresholding type )`

The parameter thresholding type is the type of thresholding we would like to perform. For example, we have basic thresholding: `cv2.THRESH_BINARY` this is the type we implemented in the function `thresholding`, it just a number:

In [ ]:
```
cv2.THRESH_BINARY
```

We can apply thresholding to the image as follows:

In [ ]:
```
ret, new_image = cv2.threshold(image,threshold,max_value,cv2.THRESH_BINARY)
plot_image(image,new_image,"Orignal","Image After Thresholding")
plot_hist(image, new_image,"Orignal","Image After Thresholding")
```

`ret` is the threshold value and `new_image` is the image after thresholding has been applied. There are different threshold types, for example, cv2.THRESH_TRUNC will not change the values if the pixels are less than the threshold value:

In [ ]:
```
ret, new_image = cv2.threshold(image,86,255,cv2.THRESH_TRUNC)
plot_image(image,new_image,"Orignal","Image After Thresholding")
plot_hist(image, new_image,"Orignal","Image After Thresholding")
```

We see that the darker elements have not changed and the lighter values are set to 255.

Otsu's method `cv2.THRESH_OTSU` avoids having to choose a value and determines it automatically, using the histogram.

In [ ]:
```
ret, otsu = cv2.threshold(image,0,255,cv2.THRESH_OTSU)
plot_image(image,otsu,"Orignal","Otsu")
plot_hist(image, otsu,"Orignal"," Otsu's method")
```

We assign the first row of pixels of the original array to the new array's last row. We repeat the process for every row, incrementing the row number for the original array and decreasing the new array's row index assigning the pixels accordingly.

In [ ]:
```
ret
```