What is Histogram?

A function that depicts the distribution of intensity levels in a given image.
- The input of the function is between [0, gray_level).
- The output depicts how often the intensity is used in the image.

Calculation of Histogram:

- int H[] = new int[gray_level];
- for (int q=0 ; q<height ; q++)
  - for (int p=0 ; p<width ; p++)
    - H[F[q][p]] ++;

- float h[] = new float[gray_level];
- for (int k=0 ; k<gray_level ; k++)
  - h[k] = (float)H[k] / (width * height);

What Can We Tell from Histogram?

- Count the number of pixels that uses each of the intensity value
- Complexity: O(MxN)

Outline

- What is histogram
- How to calculate histogram
- Grayscale transformations and their effects on histograms
- Negative image
- Brightness adjustment
- Contrast adjustment
- Gamma correction
- Color balance

Histogram Function

- Unnormalized histogram:
  - H(x): # of pixels whose intensities are x
  - Values of H(x) are integers
  - \[ \sum H(x) = \text{total # of pixels} \]

- Normalized histogram:
  - h(x): % of pixels whose intensities are x
  - Essentially the probability density function (PDF) of the input image
  - Values of h(x) are float numbers between [0,1]
  - h(x) = H(x) / total # of pixels
  - \[ \sum h(x) = 1 \]
Grayscale Transformation

- Characterizes:
  - The calculation is solely based on the intensity of the current pixel
  - Convert one grayscale value to another grayscale value
  - Some transformation functions are invertible
    - The corresponding inverse function can be used to restore the original image

Negative

- The simplest grayscale transformation
- Objective:
  - Reverse the color or intensity of the image
- Input:
  - Original image: \( F[p, q] \)
- Output:
  - Negative image: \( G[p, q] \)

Implementation for Negative

```
for (int y=0; y<height; y++)
    for (int x=0; x<width; x++) {
        Color clr = new Color(source.image.getRGB(x, y));
        int red = 255 - clr.getRed();
        int green = 255 - clr.getGreen();
        int blue = 255 - clr.getBlue();
        target.image.setRGB(x, y, red<<16 | green<<8 | blue);
    }
```

Brightness Adjustment

- Objective:
  - Make the image brighter or darker than before
- Input:
  - Original image: \( F[p, q] \)
  - Brightness adjustment: \( \Delta \)
  - \( |\Delta| < \text{gray_level} \)
- Output:
  - Updated image: \( G[p, q] \)

What Happened?

```
for (int y=0; y<height; y++)
    for (int x=0; x<width; x++) {
        Color clr = new Color(source.image.getRGB(x, y));
        int red = clr.getRed() + offset;
        int green = clr.getGreen() + offset;
        int blue = clr.getBlue() + offset;
        target.image.setRGB(x, y, red << 16 | green << 8 | blue);
    }
```

Revised Implementation

```
for (int y=0; y<height; y++)
    for (int x=0; x<width; x++) {
        Color clr = new Color(source.image.getRGB(x, y));
        int red = clr.getRed() + offset;
        int green = clr.getGreen() + offset;
        int blue = clr.getBlue() + offset;
        target.image.setRGB(x, y, red << 16 | green << 8 | blue);
    }
```
Contrast Adjustment

Objective:
- Make the image contrast stronger or weaker

Input:
- Original image: \( F[p, q] \)
- Contrast adjustment: \( \text{coef} \)
  - \( \text{coef} > 0 \)

Output:
- Updated image: \( G[p, q] \)

---

Gamma Correction

Some transformations are defined using power functions:
- \( \gamma = c \cdot x^\gamma \)
  - \( c \) and \( \gamma \) are positive constants

Widely used in correcting the power-law response of CRT monitors
- Called Gamma correction

---

Log Transformations

Defined using log function
- \( \gamma = c \cdot \log(1 + x) \)
  - \( c \) is a constant
  - \( x \) is assumed to be positive

For display images with large dynamic range
- Such as Fourier spectrum
  - Input 0\( \rightarrow 10^6 \) -> output 0\( \rightarrow 6 \)

---

Color Balance

Global adjustment of the intensities of the colors
- The goal is to render specific colors (e.g. neutral colors) correctly
  - Can be achieved by adjusting RGB channels separately

---

Implementation for Contrast Adjustment

```java
for (int y=0; y<height; y++)
    for (int x=0; x<width; x++) {
        Color clr = new Color(source.image.getRGB(x, y));
        int red = (int)(clr.getRed() * coeff + offset);
        int green = (int)(clr.getGreen() * coeff + offset);
        ...
        target.image.setRGB(x, y, red << 16 | green << 8 | blue);
    }
```
Color Balance Results

Tuesday, January 16, 2018 Minglan Gong