Introduction to Image Processing – Lab01 Report

107062230 張力仁

Proj02-02: Reducing the Number of Intensity Levels in an Image

(a) Method:

$$step = \left\lfloor \frac{256}{intensityLevel} \right\rfloor$$

$$quantizedImage = \left\lfloor \frac{originalImage}{step} \right\rfloor \times \frac{maxOriginalValue}{maxQuantizedValue}$$

where $intensityLevel = 2^k$ for $k = 1 \sim 7$, $maxOriginalValue = 255$, and $maxQuantizedValue = intensityLevel - 1$.

(b) Result:

Intensity Level = 256

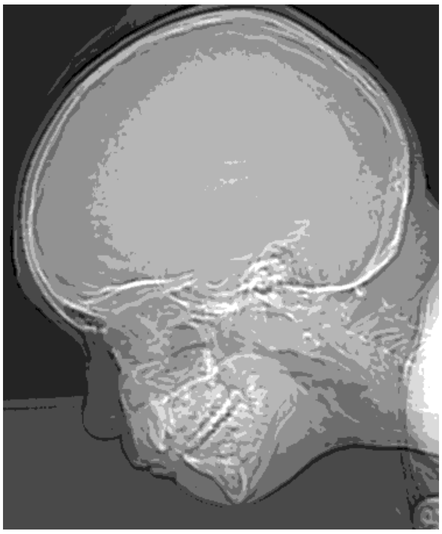
Intensity Level = 128

Intensity Level = 64

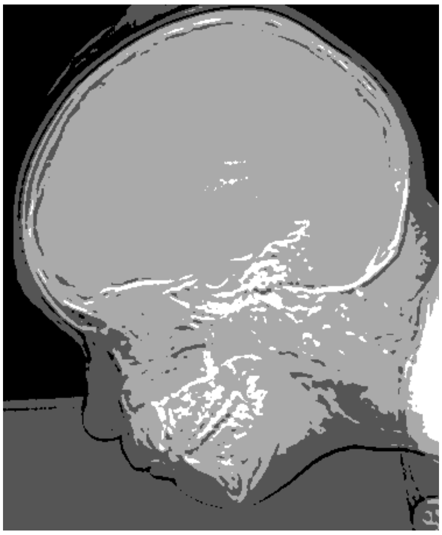
Intensity Level = 32

Intensity Level = 16

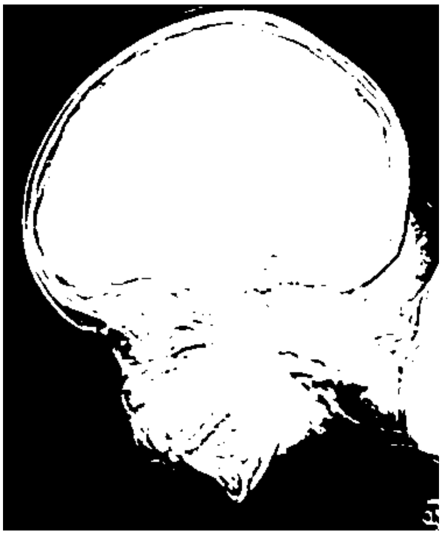
Intensity Level = 8



Intensity Level = 4



Intensity Level = 2



(c) Justification:

$\left\lfloor \frac{originalImage}{step} \right\rfloor$ transforms the image values into new intensity range, division by $maxQuantizedValue$ normalizes the intensity to $[0, 1]$, and multiplying by 255 scales the values back to $[0, 255]$. The following example shows that the quantized intensities are evenly spaced under this algorithm, and that the new minimum and maximum possible values are preserved.

Original	0 ~ 63	64 ~ 127	128 ~ 191	192 ~ 255
Intensity Level = 4	0	85	170	255

Proj02-03: Zooming and Shrinking Images by Pixel Replication

(a) Algorithm for Nearest Neighbor Interpolation: (pseudo code)

```
for i = 0 to resized_height - 1
    for j = 0 to resized_width - 1
        resized_image[i, j] = original_image[round(i / s), round(j / s)]
    end
end
(s = scaling factor)
```

(b) Shrink Fig. 2.20a by a factor of 10 (i.e., scaling factor = 0.1):

(c) Zoom the image in (b) back to the resolution of the original:

Explain the reasons for their differences:

After shrinking the image with nearest-neighbor interpolation, some information (pixel value) in the image is lost. Zooming back to the original resolution by pixel replication cannot restore the lost information.

Proj02-04: Zooming and Shrinking Images by Bilinear Interpolation

(a) Algorithm for Bilinear Interpolation: (pseudo code)

```
for i = 0 to resized_height – 1
    for j = 0 to resized_width – 1
        i' = i / scaling_factor
        j' = j / scaling_factor
        # min: avoid index out of range
        i1, j1 = min(floor(i'), H − 1), min(floor(j'), W − 1)
        i2, j2 = min(i1 + 1, H − 1), min(j1 + 1, W − 1)
        # interpolate midpoint in i-axis
        if i1 == i2
            point1 = originalImage(i1, j1)
            point2 = originalImage(i1, j2)
        else
            point1 = (i2 − i') ∗ originalImage(i1, j1)
                        + (i' − i1) ∗ originalImage(i2, j1)
            point2 = (i2 − i') ∗ originalImage(i1, j2)
                        + (i' − i1) ∗ originalImage(i2, j2)
        # interpolate midpoint in j-axis
        if j1 == j2
            resizedImage(i, j) = point1
        else
            resizedImage(i, j) = (j2 − j') ∗ point1 + (j' − j1) ∗ point2
        resizedImage(i, j) = uint8(round(resizedImage(i, j)))
    end
end
```

(b) Shrink Fig. 2.20a from 1250 dpi to 100 dpi (i.e., scaling factor = 0.08):

(c) Zoom the image in (b) back to 1250 dpi (i.e., scaling factor = 12.5):

Explain the reasons for their differences:

After shrinking the image with bilinear interpolation, some information in the image may be lost since many pixel values are replaced with weighted average of neighboring pixels. Zooming the image back to the original resolution by bilinear interpolation cannot thoroughly restore the lost information.