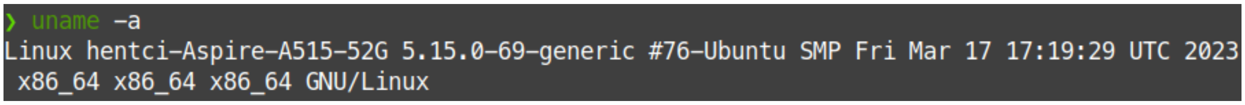
Hw1_Histogram Equalization (Due. 4/17)

409410014 資工三 柯柏旭 (hand in 4/14)

Technical description

測試環境



```
❯ uname -a
Linux hentci-Aspire-A515-52G 5.15.0-69-generic #76-Ubuntu SMP Fri Mar 17 17:19:29 UTC 2023
 x86_64 x86_64 x86_64 GNU/Linux
```

使用語言:

`python 3.10.6`

library-requirement:

```
matplotlib==3.7.1
numpy==1.23.5
opencv_python==4.7.0.72
```

如何執行

```
python3 histogram_equalization.py
```

或是

```
python histogram_equalization.py
```

便會依序顯示以下內容:

```
共12個part, 其中part6和part12各包含了16張圖
1. 原始Lena_image
2. 原始Lena_histogram
3. global approach Lena_image
4. global approach Lena histogram
5. local approach Lena_image
6. local approach Lena histogram (block * 16)
7. 原始Peppers_image
8. 原始Peppers_histogram
9. global approach Peppers_image
10. global approach Peppers histogram
11. local approach Peppers_image
12. local approach Peppers histogram (block * 16)
```

程式碼解釋

1. Histogram-Equalization (Global approach)

```python
# global approach
def histogram_equalization(img):
    [height, width, channel] = img.shape

    # sum up
    original_histogram_cnt = np.zeros(256)
    for i in range(0, height):
        for j in range(0, width):
            pixel = img[i][j][0]
            original_histogram_cnt[pixel] += 1

    # get cdf
    cdf = np.zeros(256)
    for i in range(1, 256):
        cdf[i] = cdf[i - 1] + original_histogram_cnt[i]

    # count probability
    prob = np.zeros(256)
    new_histogram = np.zeros(256)
    # total_pixels = height * width
    for i in range(0, 256):
        prob[i] = (cdf[i] - cdf.min())/ (cdf.max() - cdf.min())
        # prob[i] = cdf[i] / total_pixels
        new_histogram[i] = round(prob[i] * 255)

    # store new values
    new_img = img
    for i in range(0, height):
        for j in range(0, width):
            value = img[i][j][0]
            new_img[i][j] = new_histogram[value]

    return original_histogram_cnt, new_histogram, new_img
```

主要可以分為以下步驟:

1. 計算每個 pixel value(0~255)的出現次數 (`original_histogram_cnt`)
2. 計算累積出現的數量(`cdf`)
3. 得到`cdf`後，代入公式:

$$h(v) = \mathrm{round}\left(\frac{cdf(v) - cdf_{min}}{cdf_{max} - cdf_{min}} \times (L - 1)\right)$$

4. 將原圖經過`new_histogram`查表得到新的pixel value，將結果存入`new_img`

2. Local approach

```
# local approach
def local_approach(img):
    res = img
    # will contain 16 histogram
    blocks_histo = []
    [height, width, channel] = img.shape
    block_height = height / 4
    block_width = width / 4
    for i in range(0, 4):
        for j in range(0, 4):
            x = i * block_height
            y = j * block_width
            # comfirm the block range from original image
            block = img[int(x) : int(x + block_height - 1), \
                        int(y) : int(y + block_width - 1)]
            # local histogram equalization
            original_histo, new_histo, \
            res[int(x) : int(x + block_height - 1), int(y) : \
                int(y + block_width - 1)] = histogram_equalization(block)
            blocks_histo.append(new_histo)

    return blocks_histo, res
```

主要可以分為以下步驟:

1. 是將圖片切成16塊一樣大的blocks
2. 在每一次的iteration裡，對每個block進行histogram equalization
3. 最後將處理好的圖片(`res`)回傳

3. Histogram繪製

對於`global apprach`和`local approach`分別有兩種製圖法:

- global approach(一張圖)

```
def show_histogram(data, title):
    plt.hist(data, bins=256, range=(0, 255))
    plt.show(block=False)
    plt.waitforbuttonpress()
    plt.close()
```

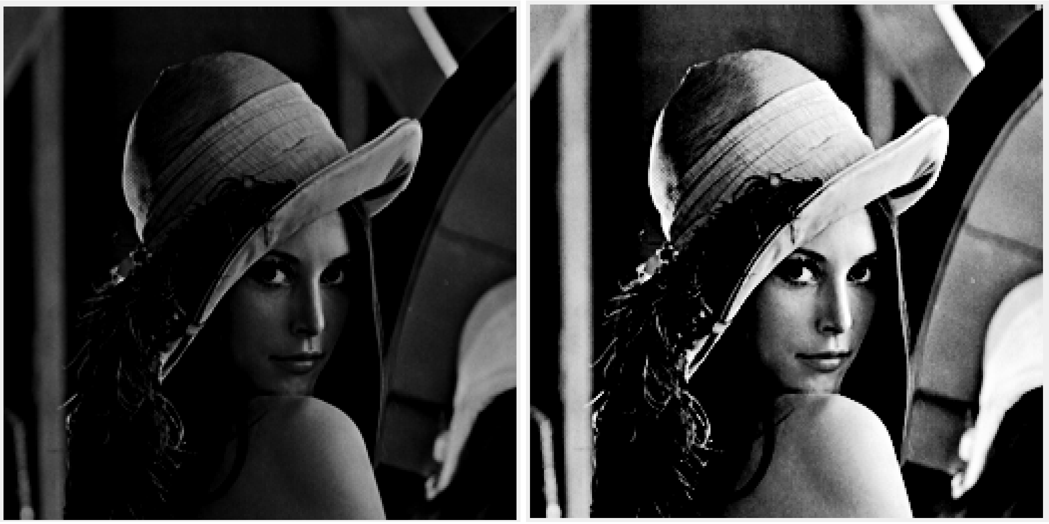
- local approach(十六張圖)

```
def show_all_local_block_histogram(block_histo):
    # Define number of blocks and plot layout
    num_blocks = 16
    num_rows = int(np.sqrt(num_blocks))
    num_cols = int(np.ceil(num_blocks / num_rows))
    # Create figure and axis objects
    fig, axs = plt.subplots(num_rows, num_cols, figsize=(12, 8))
    fig.subplots_adjust(hspace=0.5, wspace=0.3)

    # Iterate over blocks and plot histograms
    for i in range(num_blocks):
        row_idx = i // num_cols
        col_idx = i % num_cols
        axs[row_idx, col_idx].hist(block_histo[i], bins=256, range=(0, 255))
        axs[row_idx, col_idx].set_title(f"Block {i+1}")
        axs[row_idx, col_idx].set_xlim([0, 255])

    # Add overall title and axis labels
    fig.suptitle("Local Histograms")
    fig.text(0.5, 0.04, 'Pixel Value', ha='center')
    fig.text(0.04, 0.5, 'Frequency', va='center', rotation='vertical')

    # Display plot
    plt.show(block=False)
    plt.waitforbuttonpress()
    plt.close('all')
```

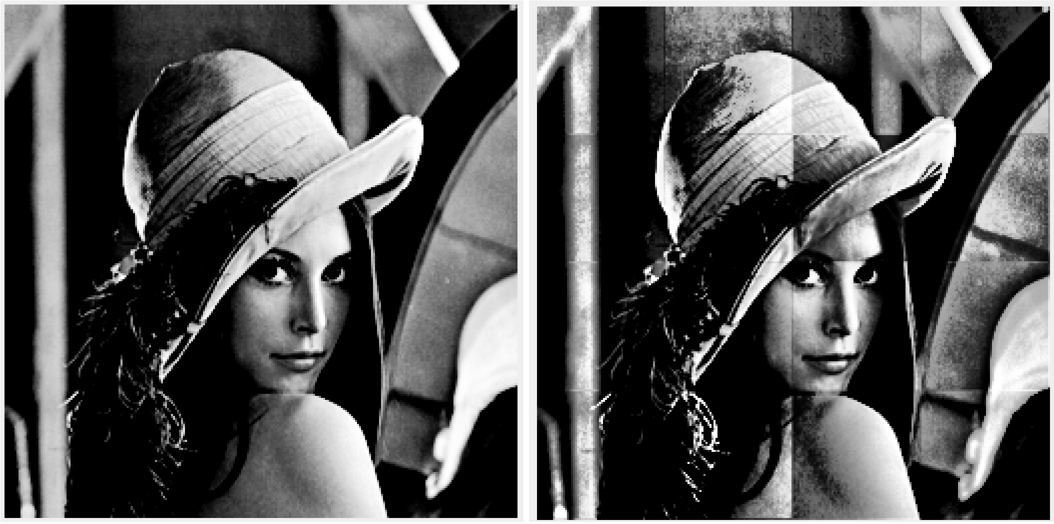
Experimental results

Lena.bmp

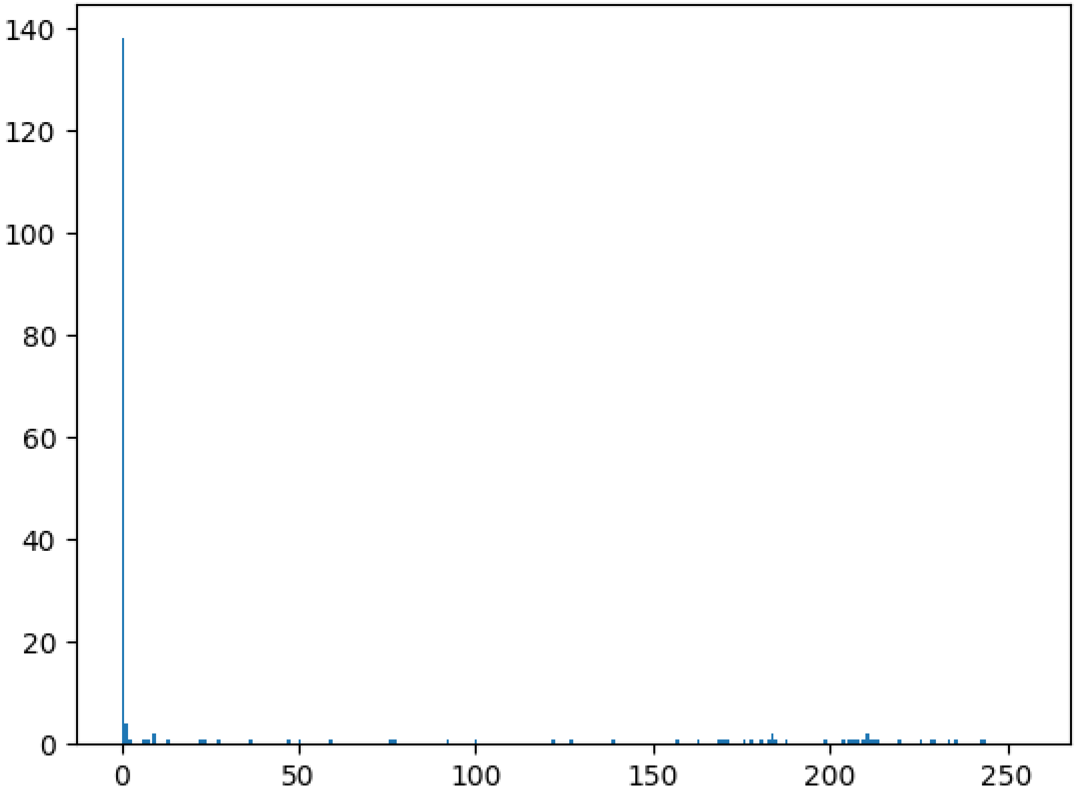
- **original** Lena v.s. **global approach** Lena

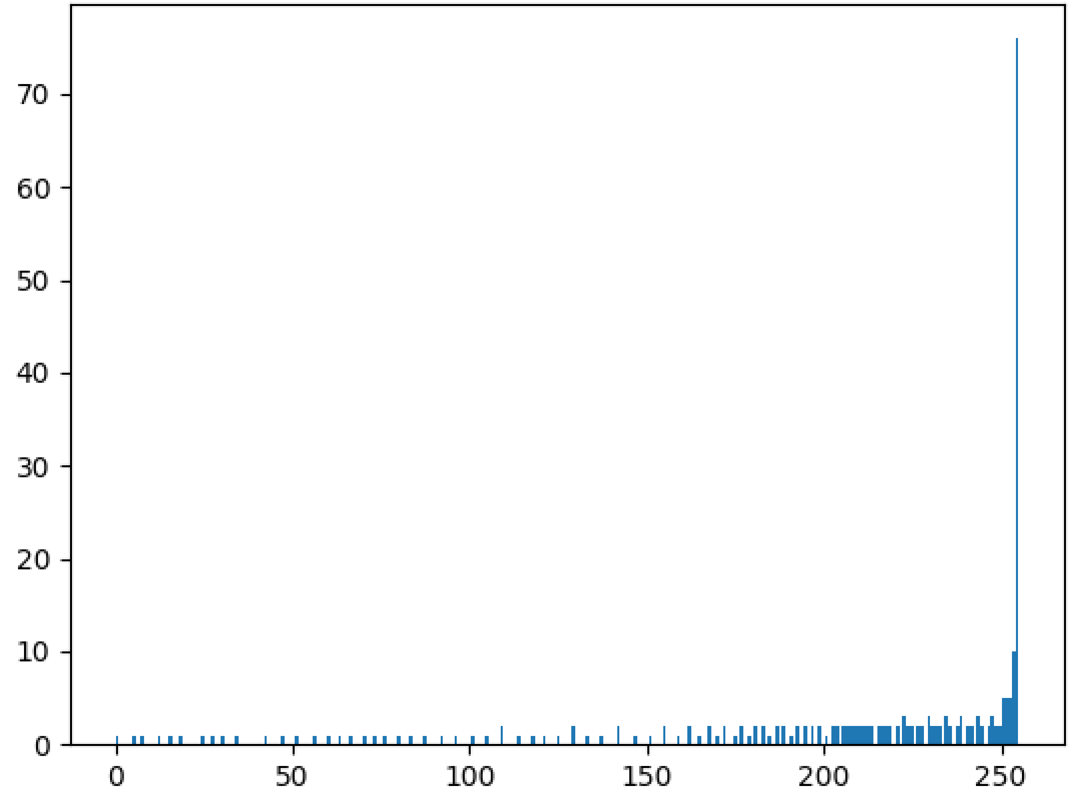


- **global approach** Lena v.s. **local approach** Lena

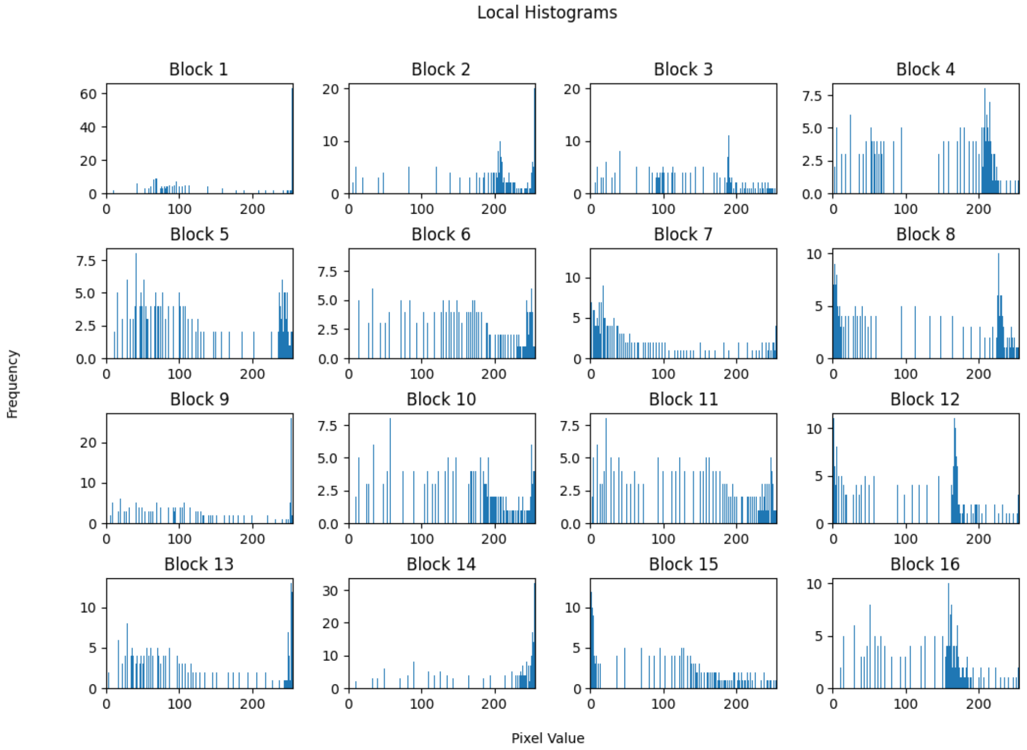


- **original** Lena histogram v.s. **global approach** Lena histogram



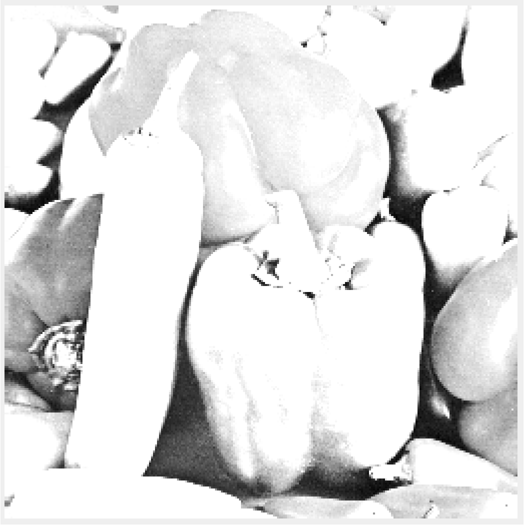


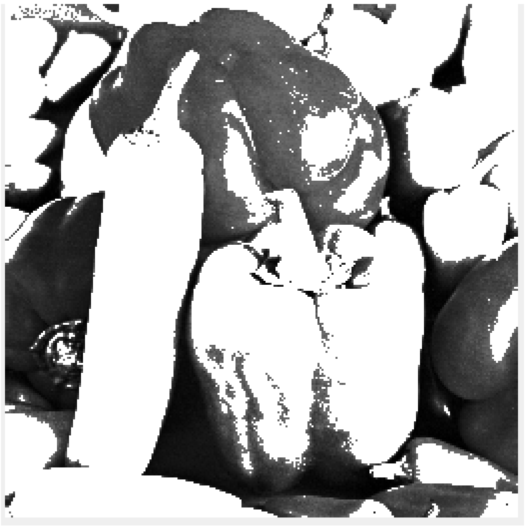
- Lena **local histogram**(block histogram)



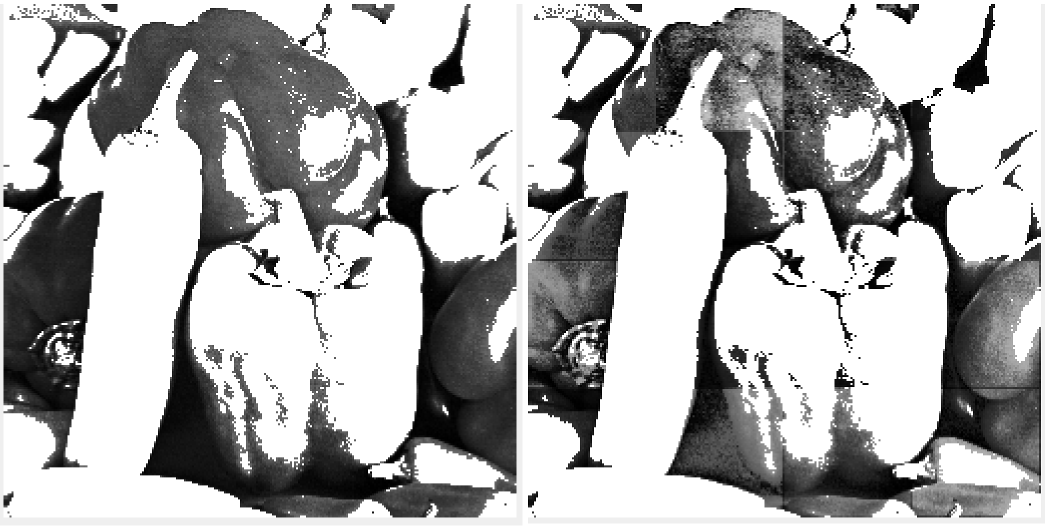
Peppers.bmp

- **original** Peppers v.s. **global approach** Peppers

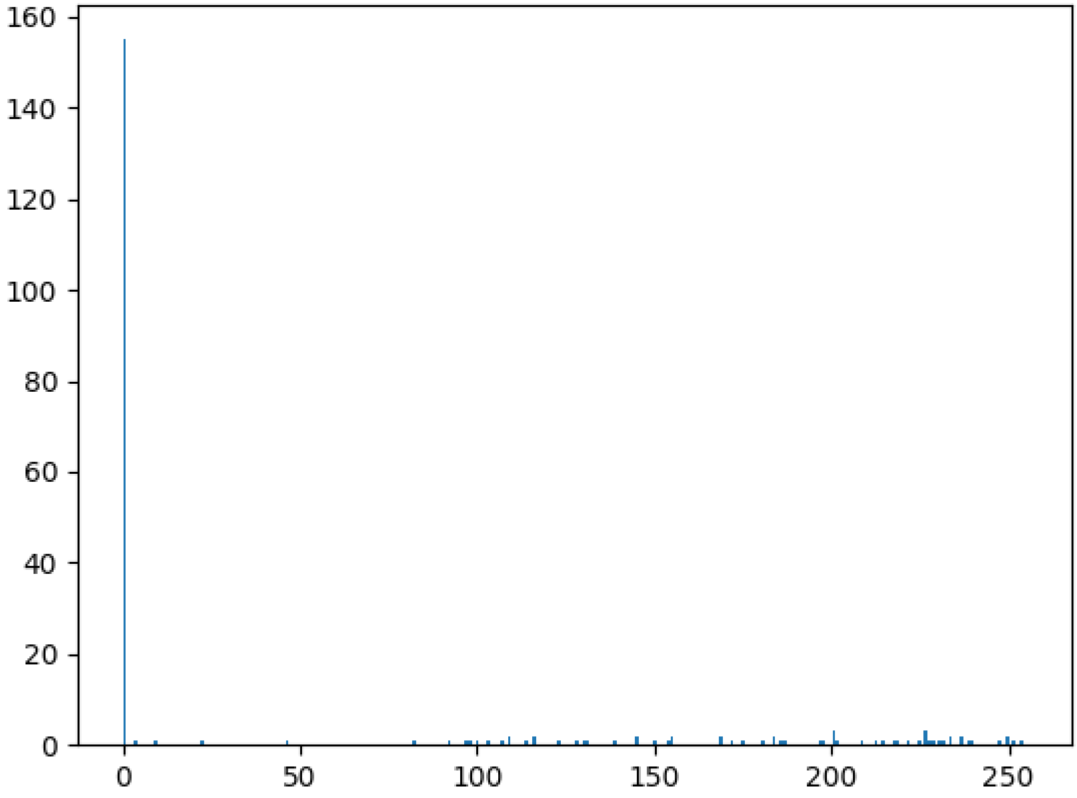


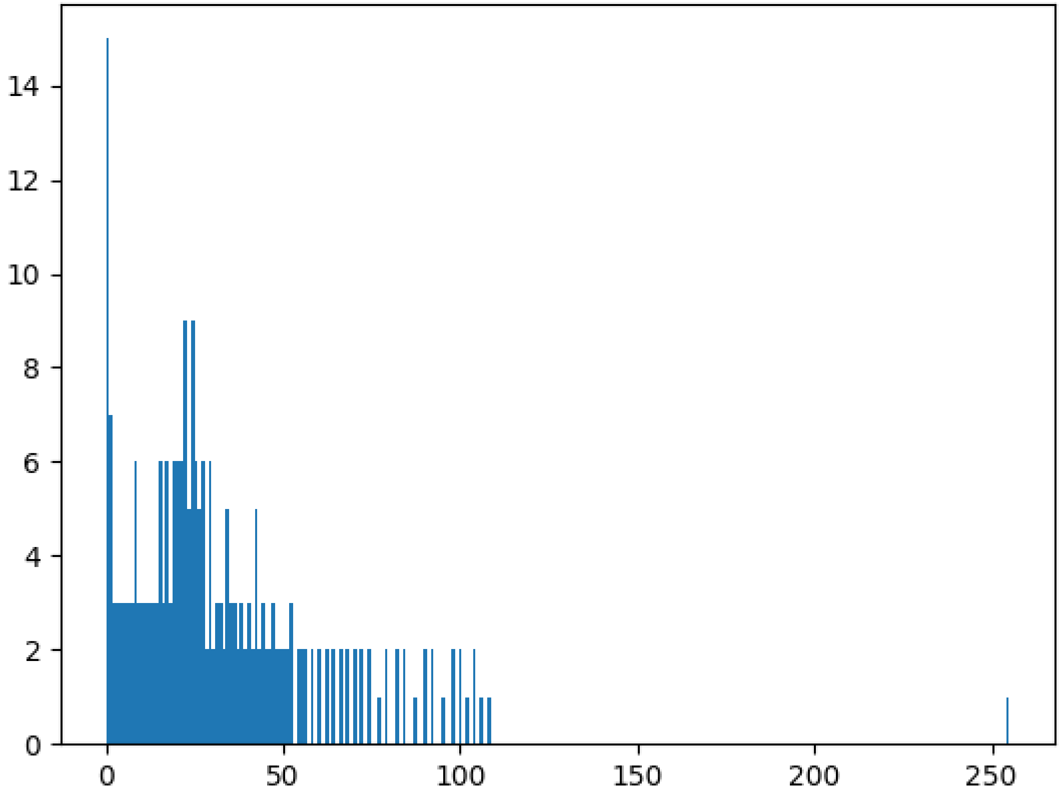


- **global approach** Peppers v.s. **local approach** Peppers

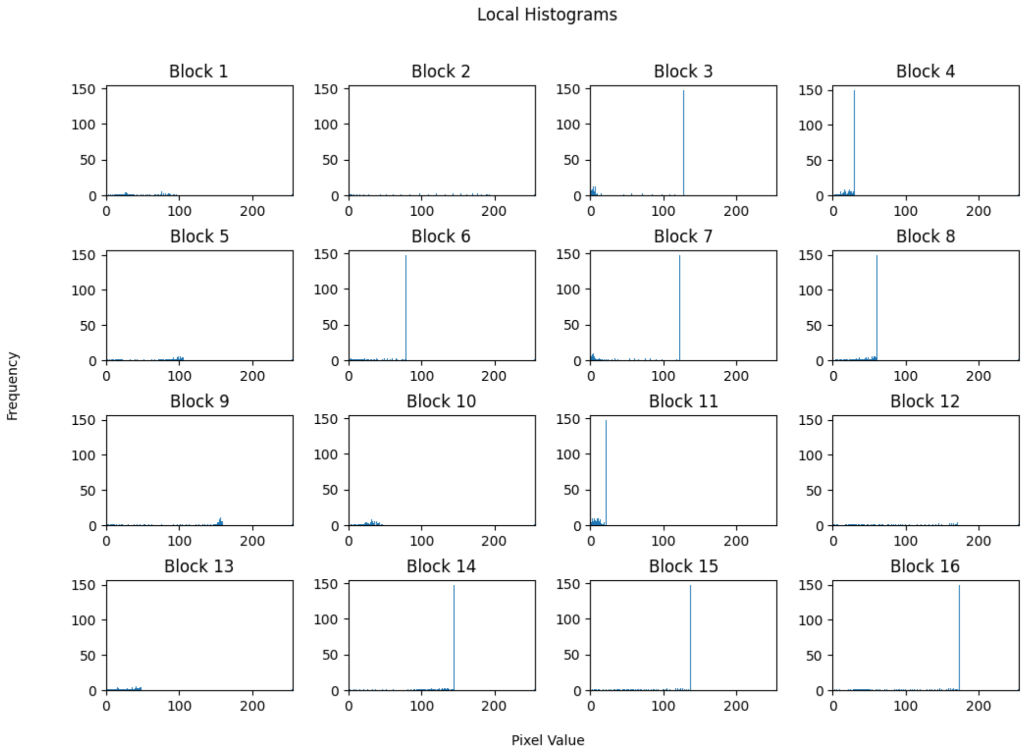


- **original** Peppers histogram v.s. **global approach** Peppers histogram





- Peppers **local histogram**(block histogram)



Discussions

從上方的實驗結果與比較可以發現:

對於Lena這張圖，histogram equalization的修復非常成功。變亮之後整張圖變得非常舒服。對於local approach的histogram equalization也能做到差不多的效果，即使他只對於單個block做統計。不過也可以發現每個block彼此都有很明顯的邊界，可以得知原始各個block之間的明亮還是有所不同。

對於Peppers這張圖，感覺修復效果就沒有Lena這麼好了。我認為原因可能在於，Peppers本身對於黑白圖片就很不搭，畢竟Peppers本身就是以鮮艷的色彩為特色。不過比起原圖，還是修復了不少。像是Pepper上面的光滑面就變得更加明顯了。

比較兩張圖的實驗結果，可以知道histogram equalization這個算法，對於將圖片由暗到亮抑或由亮到暗都可以做出不錯的結果。

心得

我覺得histogram equalization的實做本身不會太難，所以其實在算法方面很快就可以完成了。但是對於matplotlib這個library，我真的好不熟悉...。這次報告的時間大多都花在調整圖片顯示，數據範圍，邊框大小...尤其是在顯示local approach的直方圖時花最多時間。因為有16個blocks共16張圖，所以我想要一口氣將他們全部顯示來比較，最後也是費了好大功夫才實做出來。

References and Appendix

- histogram equalization算法參考
- matplotlib用法參考