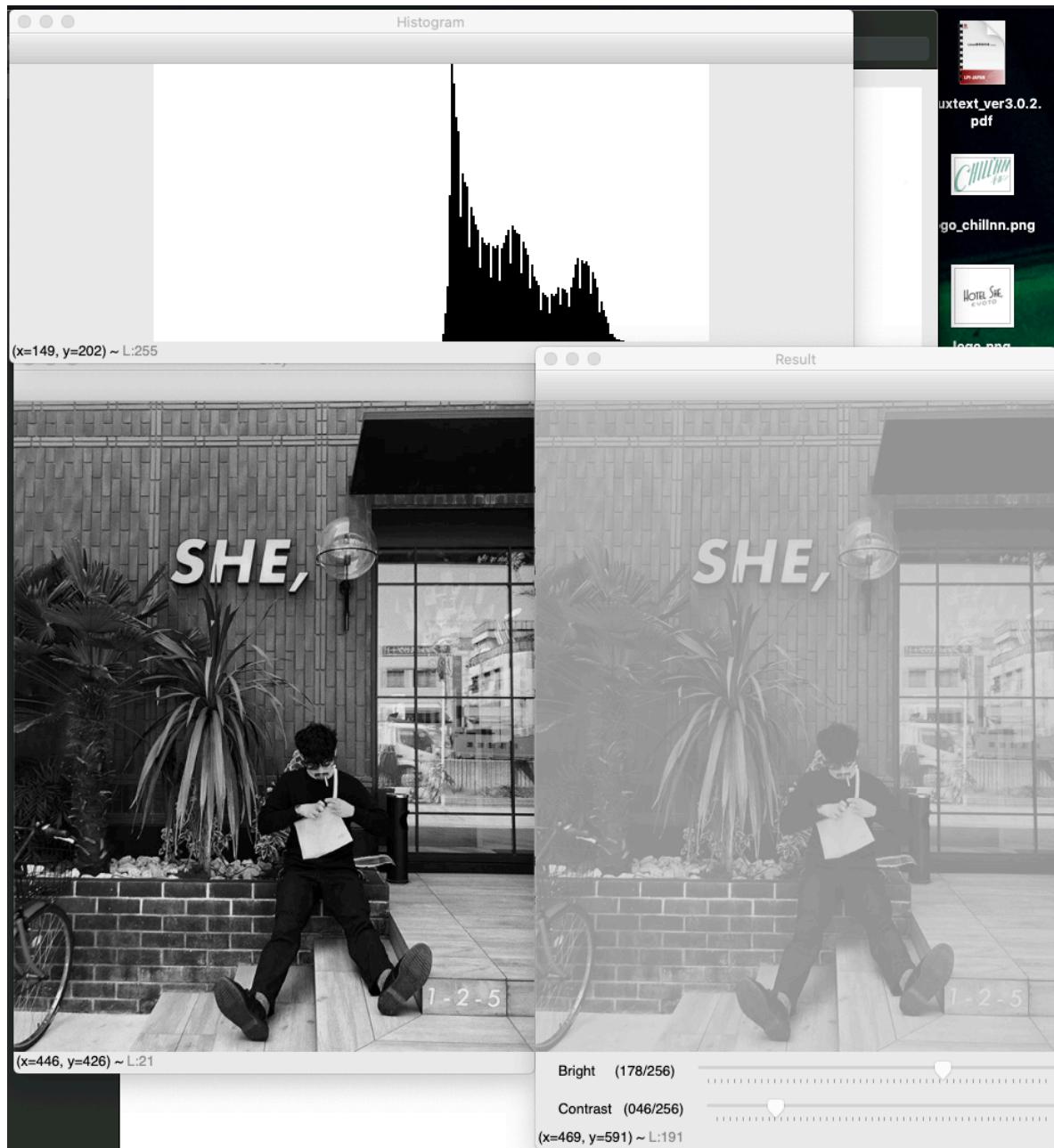


レポート2  
08-172022  
永田 謙

histgram.py

実行結果



コード：

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:


import cv2
import numpy as np
from grayscale import readImage


# In[5]:


def createLUT(win, trackB, trackC, trackHalf, histSize):
    trackMax = trackHalf * 2
    bright = cv2.getTrackbarPos(trackB, win) / trackMax
    delta = (cv2.getTrackbarPos(trackC, win) - trackHalf) / trackHalf

    if delta >= 0:
        a = 1.0 / (1.0 - delta)
        b = bright - 0.5 * a
    else:
        a = 1.0 + delta
        b = bright - 0.5*a
    lookUpTable = np.zeros(histSize, np.uint8)
    for i in range(histSize):
        newVal = round((a * i / (histSize - 1) + b) * (histSize - 1))

        if newVal < 0:
            newVal = 0
        if newVal > (histSize - 1):
            newVal = histSize - 1
        lookUpTable[i] = newVal
    return lookUpTable
```

```
# In[6]:
```

```
def getHistogram(img, histSize):
    histogram = np.zeros(histSize, np.float64)
    for row in range(len(img)):
        for col in range(len(img[row])):
            histogram[img[row, col]] += 1
    return histogram
```

```
# In[7]:
```

```
def histImage(histogram):
    histSize = histogram.shape[0]
    barWidth = 2
    histHeight = histSize
    histWidth = histSize * barWidth
    histogram = histogram * (histHeight / max(histogram))
    histImg = np.ones((histHeight, histWidth), np.uint8) * 255

    for val in range(histSize):
        cv2.rectangle(histImg, (val * barWidth, histHeight),
                     ((val+1)*barWidth-1, histHeight-
int(histogram[val])), 0)

    return histImg
```

```
# In[8]:
```

```
winRes = 'Result'
trackB = 'Bright'
trackC = 'Contrast'
trackHalf = 128
```

```
# In[9]:
```

```
def onChange(val):
    global gray
    histSize = 256
    lookUpTable = createLUT(winRes, trackB, trackC, trackHalf,
histSize)

    result = np.zeros(gray.shape[:2], np.uint8)

    for row in range(len(result)):
        for col in range(len(result[row])):
            result[row, col] = lookUpTable[gray[row, col]]
    cv2.imshow(winRes, result)
    hist = histImage(getHistogram(result, histSize))
    cv2.imshow('Histogram', hist)
```

```
# In[10]:
```

```
def main():
    global gray
    img = readImage()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Gray', gray)
    trackMax = trackHalf * 2
    cv2.namedWindow(winRes)
    cv2.createTrackbar(trackB, winRes, 0, trackMax, onChange)

    cv2.setTrackbarPos(trackB, winRes, trackHalf)
    cv2.createTrackbar(trackC, winRes, 0, trackMax, onChange)

    cv2.setTrackbarPos(trackC, winRes, trackHalf)
    onChange(trackHalf)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
# In[11]:
```

```
if __name__ == '__main__':
    main()
```

```
# In[ ]:
```

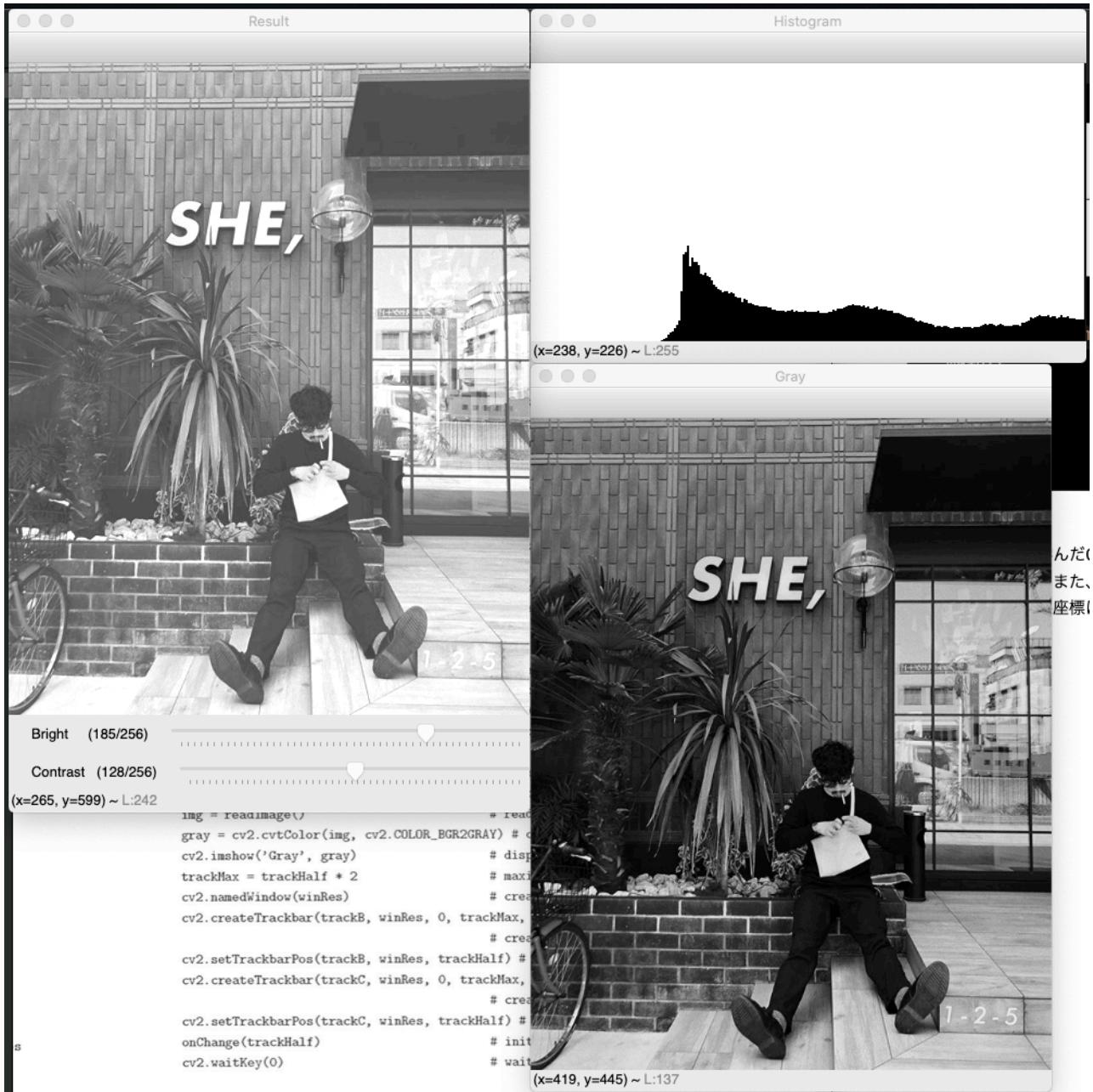
考察：

前回レポート時に学んだGUIでのパラメーターの変更方法を輝度とコントラストの二変数に対して適応した。また、cv2のrectangle関数を使ってヒストグラムを作成した。ヒストグラムは、各座標に得られたデータの高さと固定の横幅を持つ四角形を配置することで描かれている。

今回の主題はトーンマッピングであるが、あらかじめルックアップテーブルを定義しておくことで、ある値が取られた時に、どのトーンにマッピングすべきかをピクセル毎に計算する必要がなく、少ない計算量で変換が可能になっている。

トラックバーでいじっているのは、画像がとりうる画素値のマッピングに用いる曲線である。

## histgramCV.py



コード：

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:


import cv2
from grayscale import readImage
```

```
from histogram import createLUT, histImage, winRes, trackB,
trackC, trackHalf

# In[5]:


def getHistogram(img, histSize):
    """
    img      -given image
    histSize -size of histogram (number of bins)
    create and return a histogram with OpenCV
    """
    histogram = cv2.calcHist([img], [0], None, [histSize], [0,
histSize])

    return histogram
# In[6]:


def onChange(val):
    """
    val      -changed trackbar value (not used directly)
    """
    global gray, result
    histSize = 256
    lookUpTable = createLUT(winRes, trackB, trackC, trackHalf,
histSize)

    result = cv2.LUT(gray, lookUpTable)
    cv2.imshow(winRes, result)
    hist = histImage(getHistogram(result, histSize))
    cv2.imshow('Histogram', hist)

# In[7]:


def main():
    global gray, result
    img = readImage()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
cv2.imshow('Gray', gray)
trackMax = trackHalf * 2
cv2.namedWindow(winRes)
cv2.createTrackbar(trackB, winRes, 0, trackMax, onChange)

cv2.setTrackbarPos(trackB, winRes, trackHalf)
cv2.createTrackbar(trackC, winRes, 0, trackMax, onChange)

cv2.setTrackbarPos(trackC, winRes, trackHalf)
onChange(trackHalf)
cv2.waitKey(0)
cv2.imwrite('mapped.jpg', result)
cv2.destroyAllWindows()

# In[8]:
```

```
if __name__ == '__main__':
    main()

# In[ ]:
```

出力

考察：

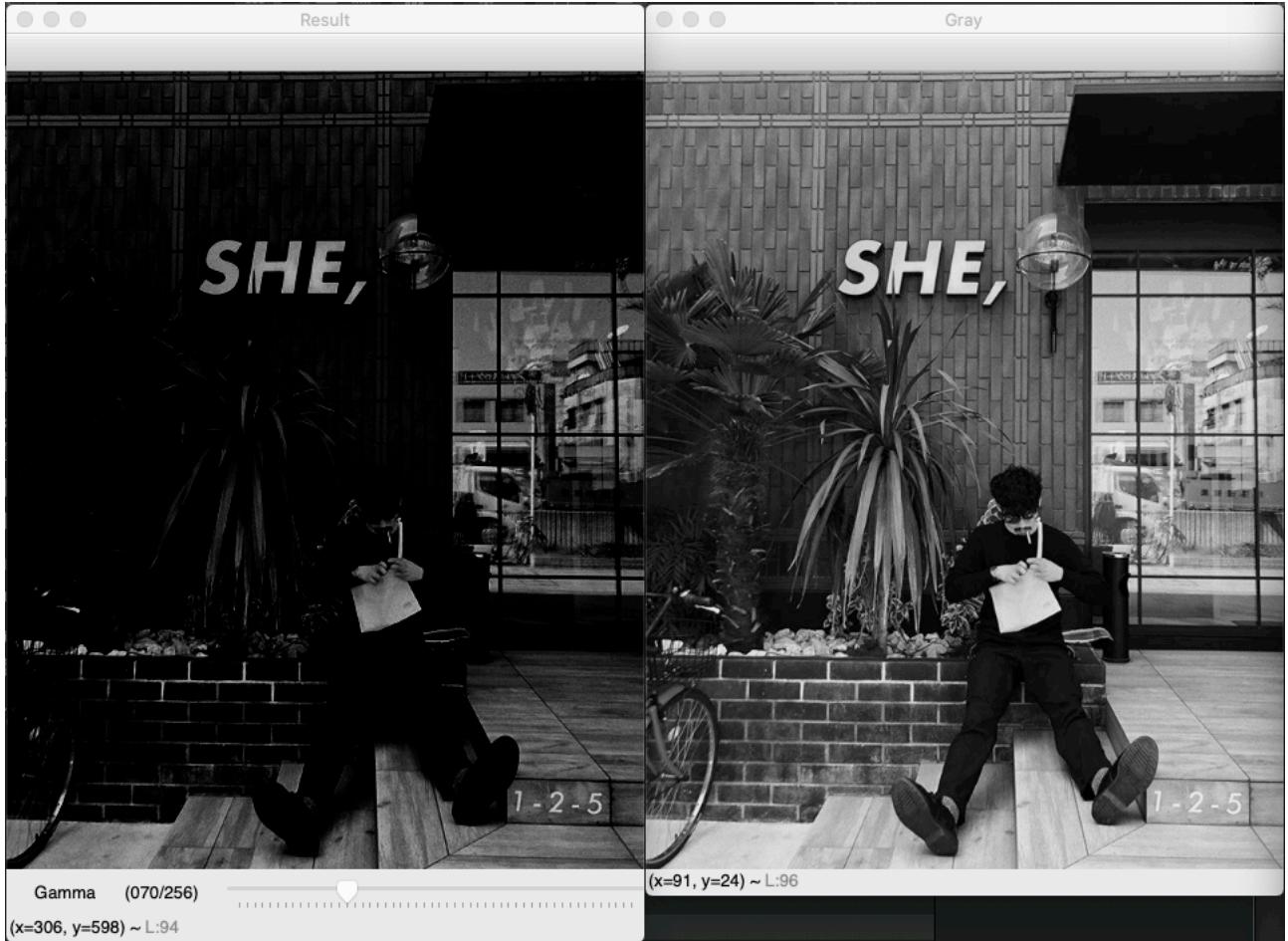
まず動作がめちゃくちゃ早くなっていた。

教科書での説明通り、cv2.LUTを用いて階調変換を行うことで処理速度が向上している。  
numpyで並列で計算を行なっているためだと考えられる。

出力



## gammaCV.py



コード：

```
import cv2
import numpy as np
from grayscale import readImage

def gammaLUT(trackHalf, trackMax):
    base, ratio = 2.0, 32.0
    histSize = 256
    lookUpTable = np.zeros((trackMax+1, histSize), np.uint8)
    for trackVal in range(trackMax+1):
        gamma = base*((trackHalf-trackVal) / ratio)
        for val in range(histSize):
            lookUpTable[trackVal, val] = \
                round((val/(histSize-1))**gamma * (histSize-1))
    return lookUpTable
```

```
winRes = 'Result'
track = 'Gamma'

def changeGamma(val):
    global gray, lookUpTable
    val = cv2.getTrackbarPos(track, winRes)
    result = cv2.LUT(gray, lookUpTable[val])
    cv2.imshow(winRes, result)

def main():
    global gray, lookUpTable
    img = readImage()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Gray', gray)

    trackHalf = 128
    trackMax = trackHalf * 2
    lookUpTable = gammaLUT(trackHalf, trackMax)

    cv2.namedWindow(winRes)
    cv2.createTrackbar(track, winRes, 0, trackMax, changeGamma)

    cv2.setTrackbarPos(track, winRes, trackHalf)
    changeGamma(trackHalf)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main()

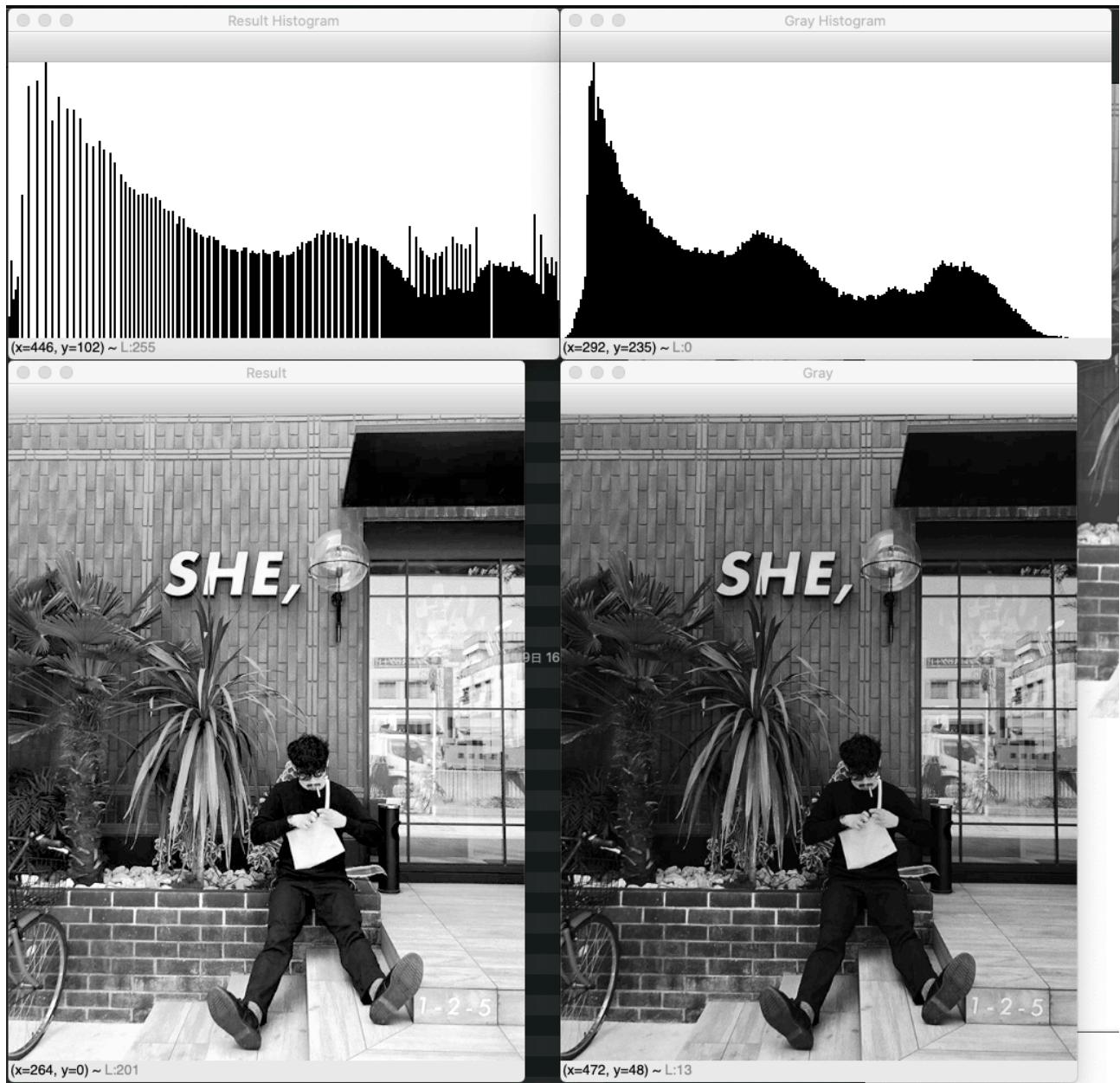
# In[ ]:
```

考察：

ページを跨いだインデントに惑わされ、しばらく時間がかかった。  
ガンマ変換に関しては第三回のレポートにてまとめたのでそちらを参照されたい。

## equalizeCV.py

### 実行結果



コード：

```
import cv2
from grayscale import readImage
from histogram import histImage
from histogramCV import getHistogram

def main():
    img = readImage()
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('Gray', gray)
histSize = 256
hist = histImage(getHistogram(gray, histSize))
cv2.imshow('Gray Histogram', hist)
equal = cv2.equalizeHist(gray)
cv2.imshow('Result', equal)
hist = histImage(getHistogram(equal, histSize))
cv2.imshow('Result Histogram', hist)
cv2.waitKey(0)
cv2.imwrite('equalized.jpg', equal)
cv2.destroyAllWindows()
```

```
if __name__ == '__main__':
    main()
```

考察：

ヒストグラムの均一化を行うことで、なんとなく明るい雰囲気の画像になった。  
実際にはヒストグラム均一化で行なっているのは、コントラストが低い領域においても、  
高いコントラストにすることで、ものをはっきり見えるようにすることができる。  
内部実装は課題にて確認する。

余談だが、プリントのサンプルコードのファイル名の拡張子がccになっていた。

## ヒストグラム均一化

### 実行結果



コード：

```
import cv2
import numpy as np
from grayscale import readImage
from histogram import histImage
from histogramCV import getHistogram

def main():
    img = readImage()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Gray', gray)
    histSize = 256
    hist = histImage(getHistogram(gray, histSize))
    cv2.imshow('Gray Histogram', hist)
    # ここを変更する
    ....
    ヒストグラムを均一化するルックアップテーブルを作成し、そのルックアップテーブルを用いてトーンマッピングする
    ヒストグラムが均一化されている場合には、累積相対頻度と明度が等しくなる。
```

=> ヒストグラムから累積ヒストグラムを作成し、それをトーンマッピングのルックアップテーブルとして用いればいい。

```
....  
# histogramの取得  
histogram = getHistogram(gray, histSize)  
# 累積histogramを作成  
ruiseki = getRuikeki(histogram)  
print('ruiseki', ruseki)  
hist = histImage(ruseki)  
cv2.imshow('Ruikeki Histogram', hist)  
# 累積histogramをトーンマッピングのルックアップテーブルとして使う  
lookUpTable = getLookUp(ruseki)  
equal = cv2.LUT(gray, lookUpTable)  
cv2.imshow('Result', equal)  
hist = histImage(getHistogram(equal, histSize))  
cv2.imshow('Result Histogram', hist)  
cv2.waitKey(0)  
cv2.imwrite('equalized.jpg', equal)  
cv2.destroyAllWindows()
```

```
def getRuikeki(histogram):  
    ruseki = np.zeros(histogram.shape, np.float64)  
    sum = 0  
    for i in range(len(histogram)):  
        ruseki[i, 0] = sum + histogram[i, 0]  
        sum += histogram[i, 0]  
    # 最大値  
    max = ruseki[len(histogram)-1, 0]  
    ruseki = (ruseki * (255.0 / max)).astype(np.uint8)  
    # 正規化  
    return ruseki
```

```
def getLookUp(ruseki):  
    lookUpTable = np.zeros(len(ruseki), np.uint8)  
    for i in range(len(ruseki)):  
        lookUpTable[i] = ruseki[i, 0]  
    return lookUpTable
```

```
if __name__ == '__main__':
    main()
```

考察：

丁寧に誘導に従うことで、実装することができた。

ヒストグラム均一化で行なっていることは特定の範囲内に集中してしまっている画素値を、ヒストグラム上で全体に広げている。

つまり、写真全体が暗かったとしても、明るかったとしても、結果的にヒストグラム均一化によって得られる画像は同じような光源環境で撮影されたものになる。