影像處理 Assignment1 學號:311605015 姓名:張哲源

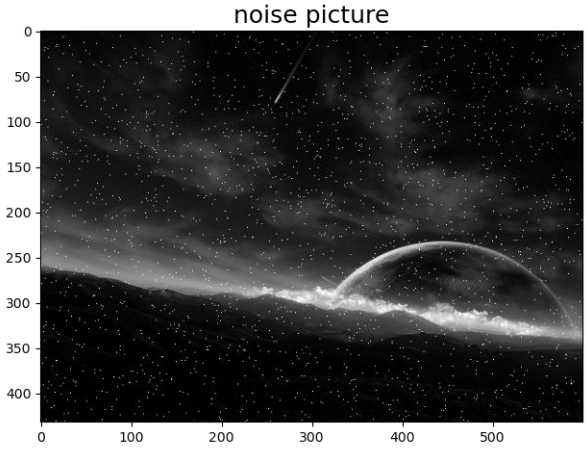
1.Noise reduction:

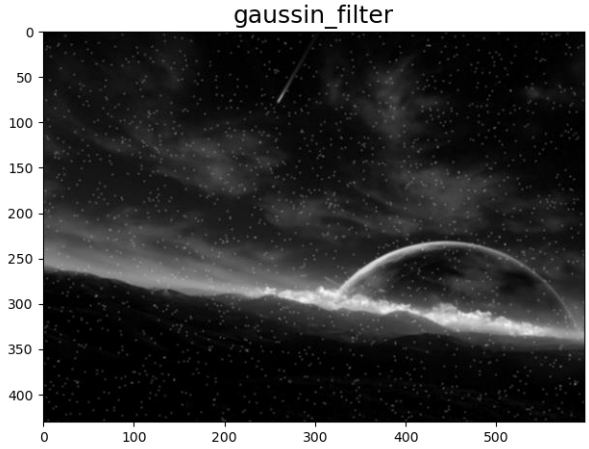
首先為了更好的比對不同降噪方法所帶來的影響，我手動增加了不同類型的噪點如椒鹽點等，並將圖片透過單通道明亮度進行讀取。

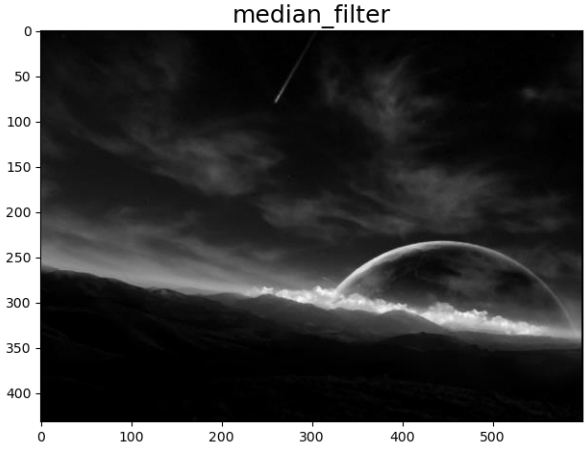
在第一組比對中我使用了四種不同的降噪濾波器，分別是 gaussian_filter,median_filter 與 mean_filter，其中 gaussian_filter 屬於一種 spatial_filter，在此我用卷機進行處理具有濾除雜訊、低通、模糊化等效果，公式如下

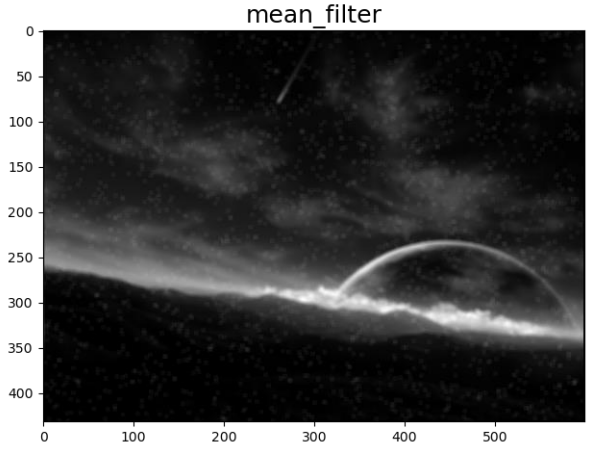
$$\text{Gaussian Filter}=1/16\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

median 與 mean_filter 則是在一個空間 filter 內分別取中位數與平均值並分配到新圖片中，下面範例都是以 3*3 的 filter_sieze 作為範例



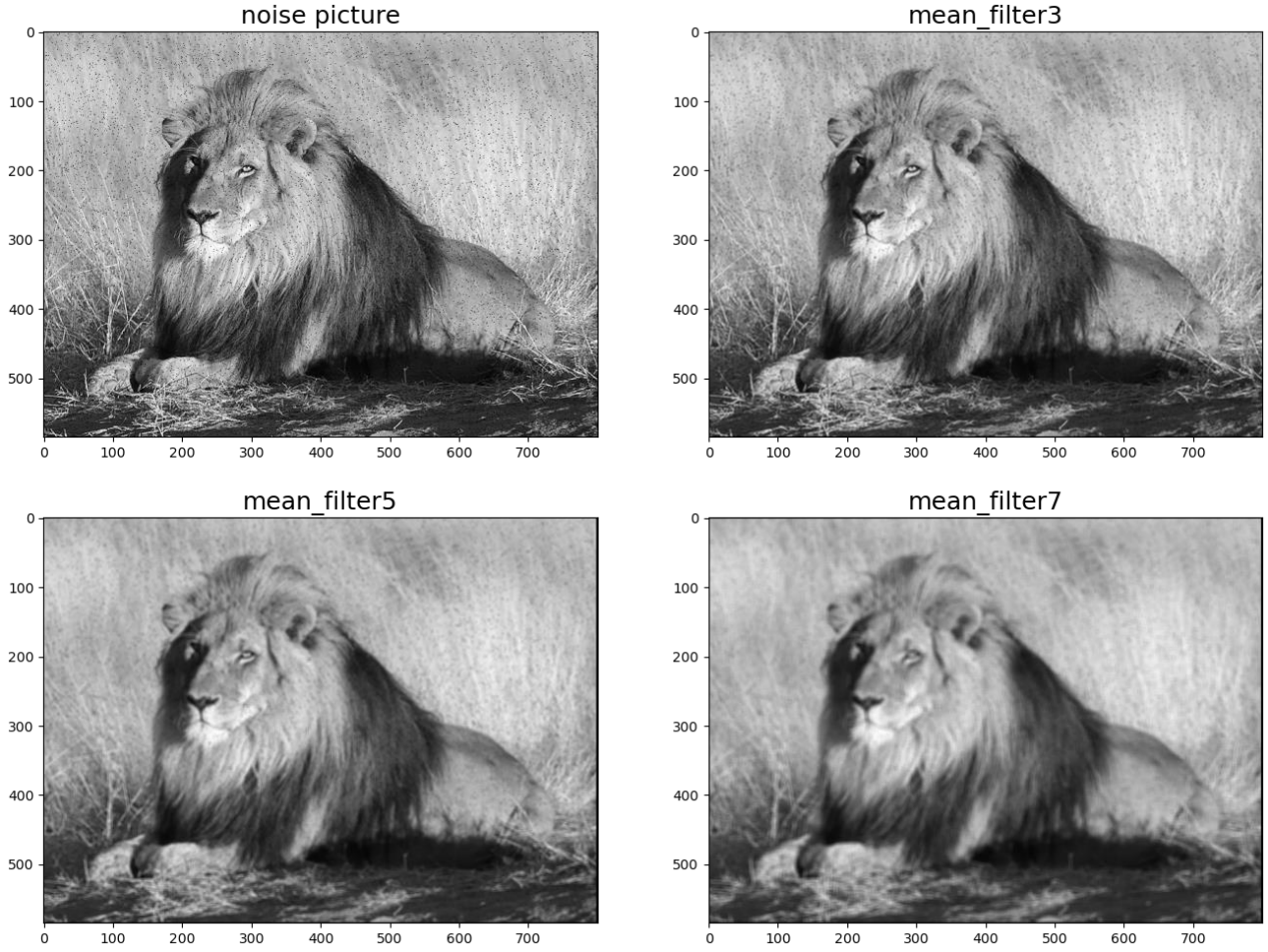






在加入椒鹽噪點的情況下 median filter 的效果最為明顯，其他兩種 filter 在此情況對於降噪能力有限，並將圖片弄得較為模糊

接著我以不同大小的 mean filter 觀察其效果，看是否不同 size 的 filter 影響如何，mean filter 主要是將 filter 下內所有值相加後取平均，median filter 則是取中位數，放大後可以更好的觀察結果，可以觀察到 filter size 越高噪點閱不明顯，但相對的圖片也稍微更模糊，這也是在預測之中的



2.邊緣偵測

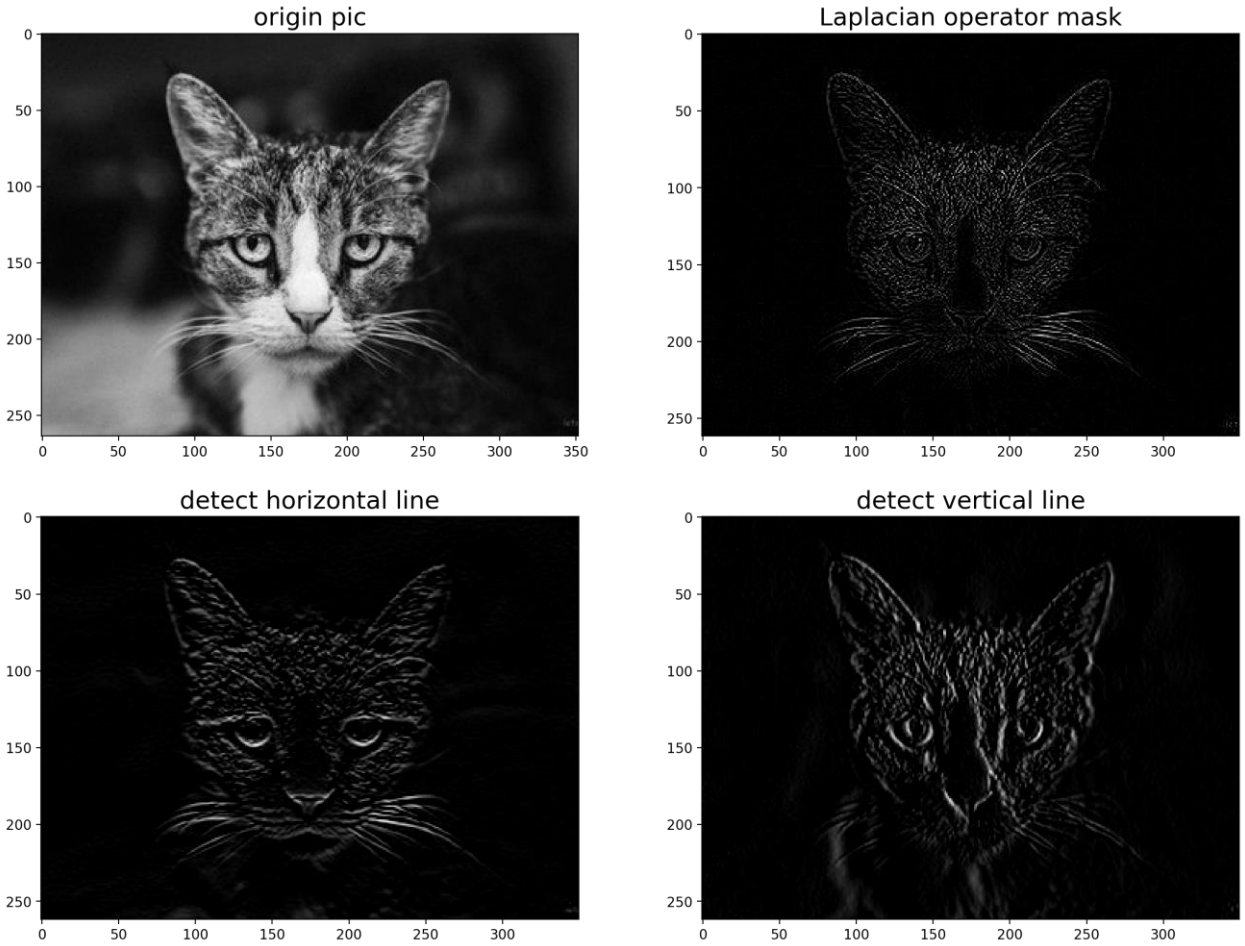
下面比對我用了三種邊緣強化濾波器，分別是 Laplacian Operator 與 Sobel Operator，相對於 Laplacian Operator 來說，Soble operator 用圖像的梯度作為判斷依據，並可以判斷水平與垂直方向的邊緣，三組公式如下

Laplacian Operator Mask = $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

Sobel Operator Mask(horizontal)= $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$

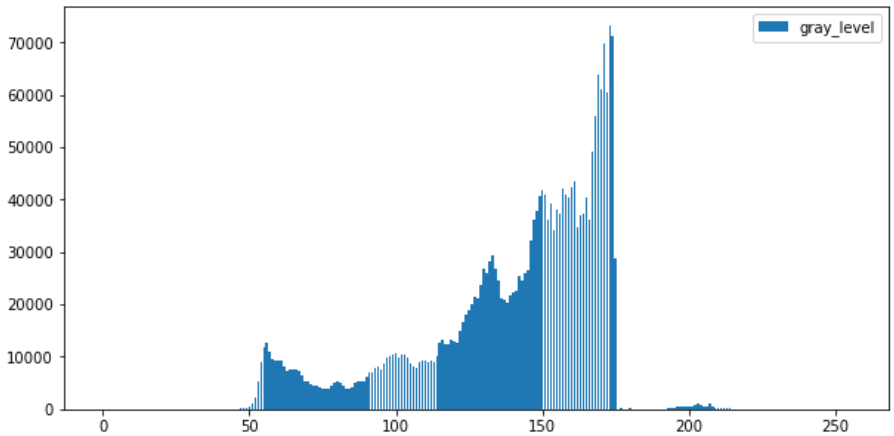
Sobel Operator Mask(vertical) = $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

實驗結果如下，利用空間捲機套用上面三種 filter，可以看出水平及垂直線條偵測效果明顯，拉普拉絲運算子也把貓的輪廓很好的抓出來，將貓的各種邊緣特徵提取的效果很好



3.Contrast adjustment

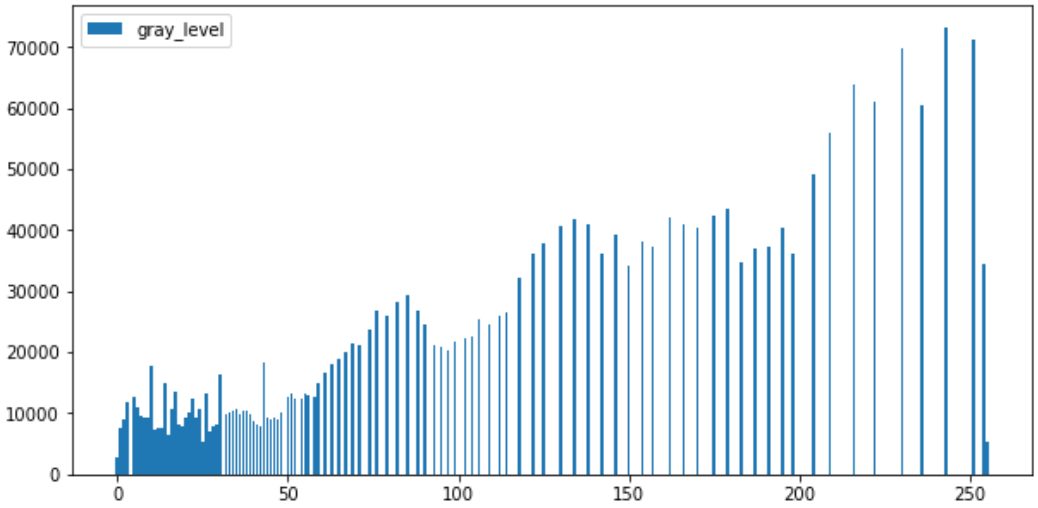
接下來我要使用 histogram equalization，首先先計算出各圖的 histogram，這邊先以黑白圖為例，後面再用彩色圖去做測試，首先我將圖片上各灰度值出現次數統計起來



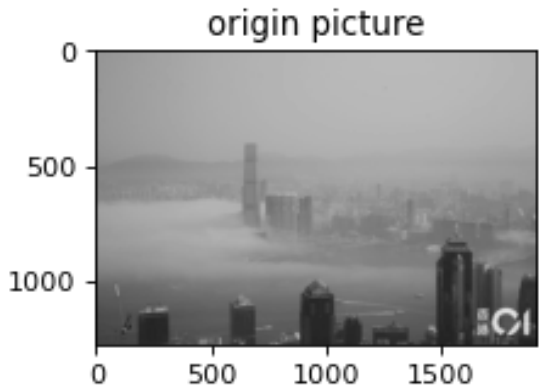
並透過以下算法將 hitogram equzilation

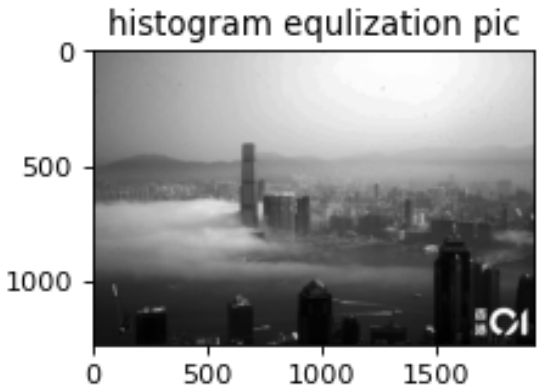
```
convert_pic=np.zeros((pic.shape[0],pic.shape[1]))
image_pdf=histogram/pic.size
#pic.size=height*weight pdf 為每個亮度出現的機率
image_cdf=np.cumsum(image_pdf)
#將 pdf 轉成累積機率
image_equ_value=np.around(image_cdf*255).astype('uint8')
#代表原本的亮度應該對應到的亮度
convert_pic = image_equ_value[pic]
#在 pic 座標的亮度對應到 equal_value 後應該為多少亮度
```

Equlization 後其灰度值如下圖，可以看到明顯平均了不少

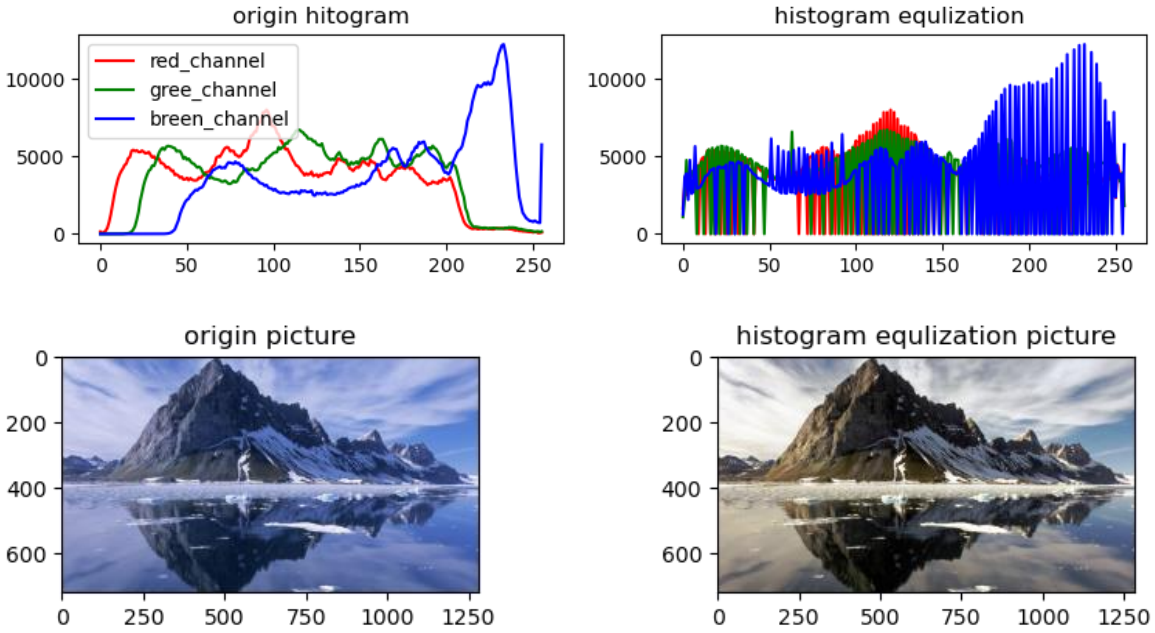


成果如下圖經過 histogram equalization 後，能將較為霧化的圖片增加了其對比度，圖片也變得較為清晰。

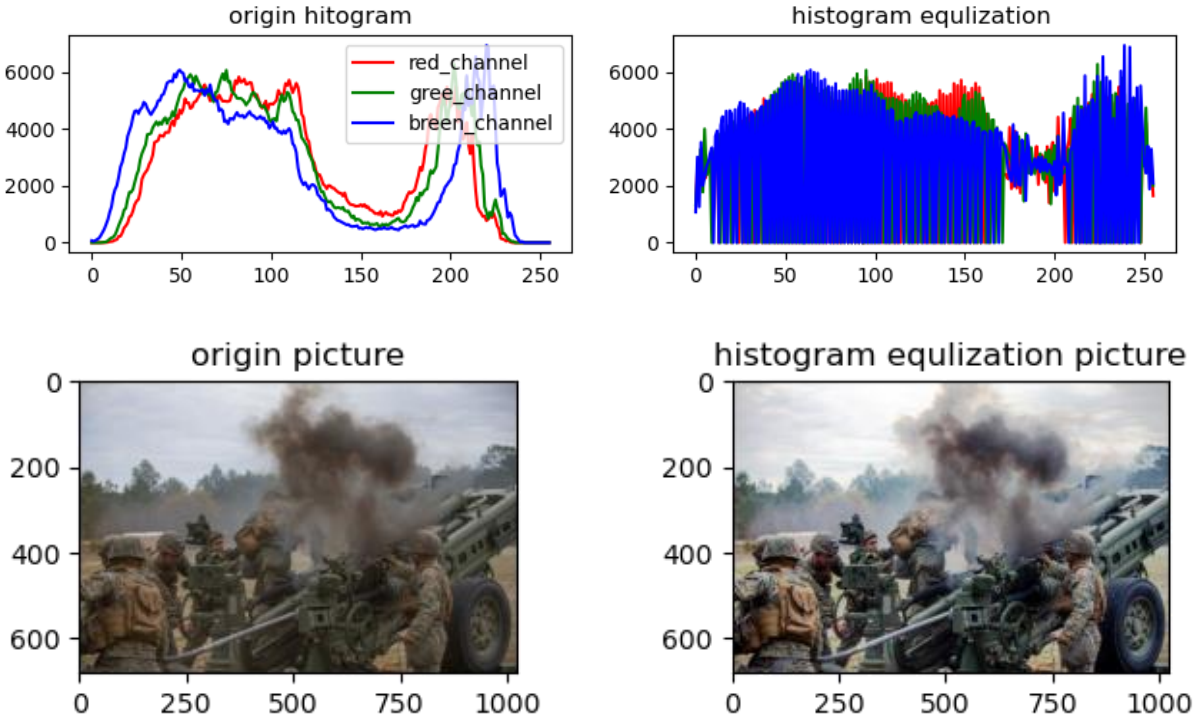




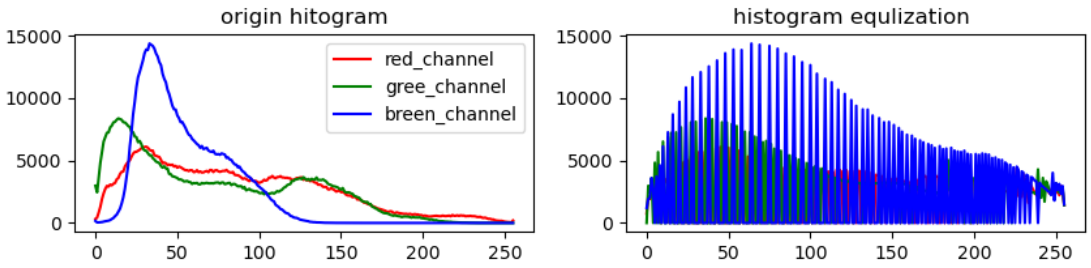
RGB 三通道也同理，分別對每一個通道進行 equalization，其色度表與轉換後的圖片如下面範例，圖片原本色調偏藍，經過 equalization 後，看起來效果不錯，有著春意盎然的感覺

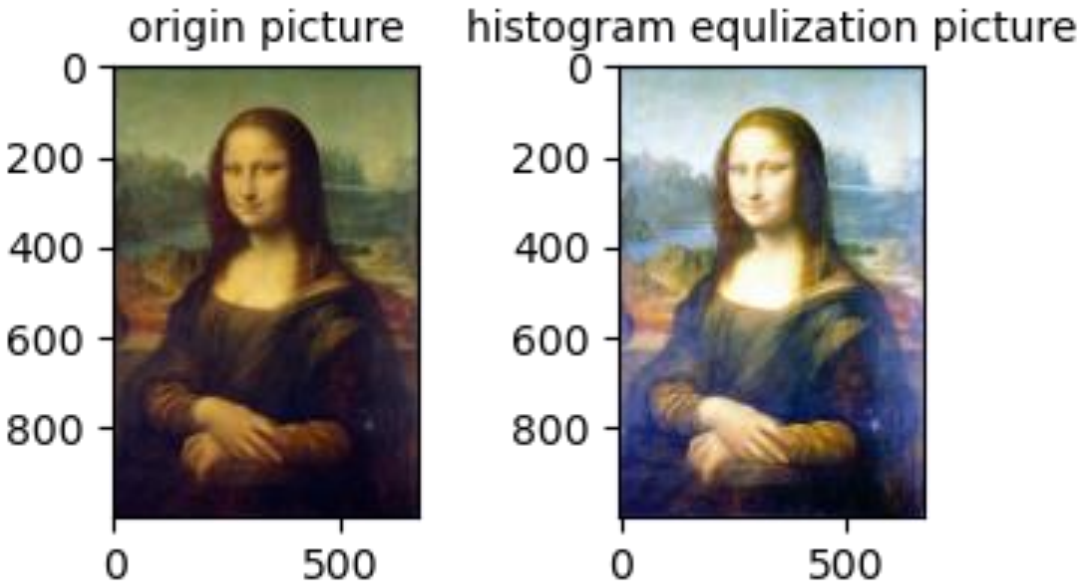


接著第二組對比圖效果也不錯，將原本較為色調較為灰暗的戰爭圖片變得具有現代感，色彩較為鮮豔



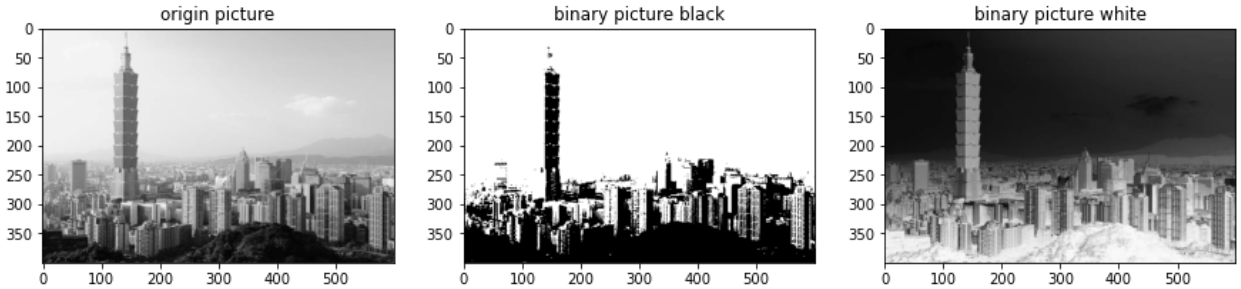
接著來看失敗的例子，我拿蒙娜麗莎的微笑作為輸入，可以看出 equalization 後的圖片明顯過曝，顏色也怪怪的，我認為是這種大師級的藝術品本身在顏色上的選用就已經趨近完美，因此在對他做顏色上的處理反而會適得其反



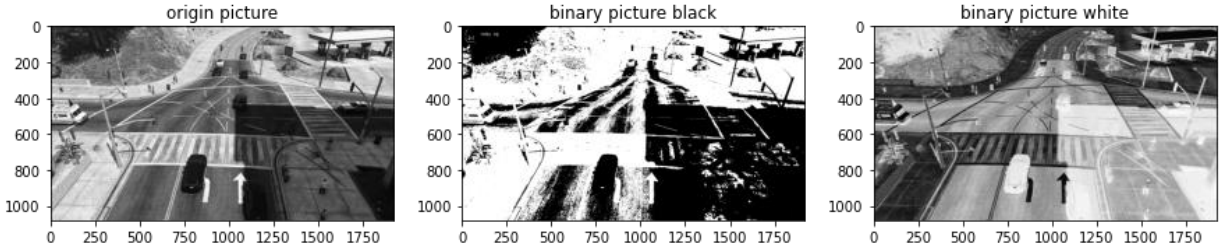


4.二值化

設定一個 threshold，並將高於這個 threshold 的灰度值都設為 255，反之都設為 0，反二值化則是與上面相反。其成果如下，效果還不錯，只用兩種灰度值便將城市之輪廓勾勒了出來。

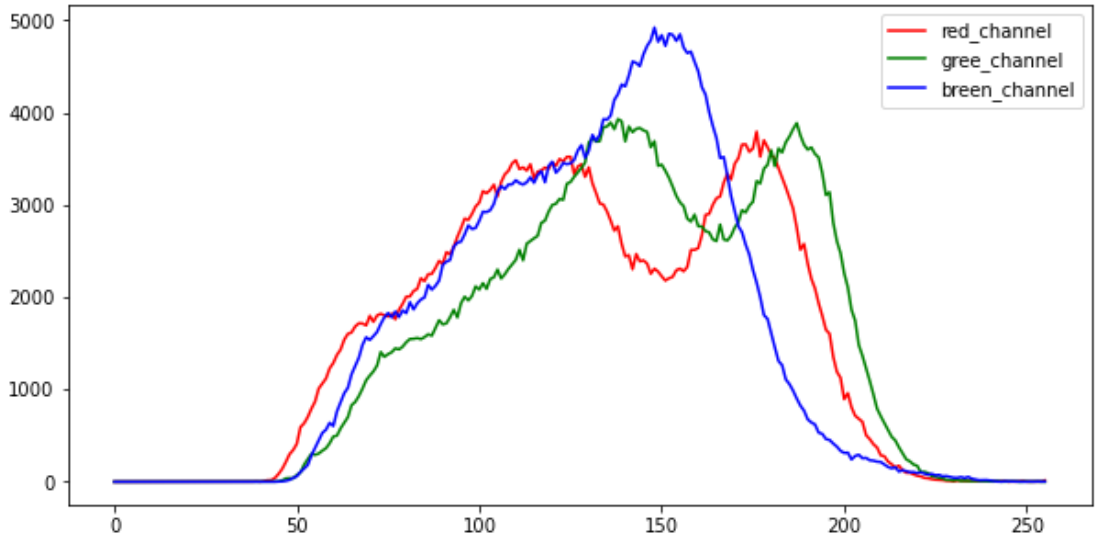


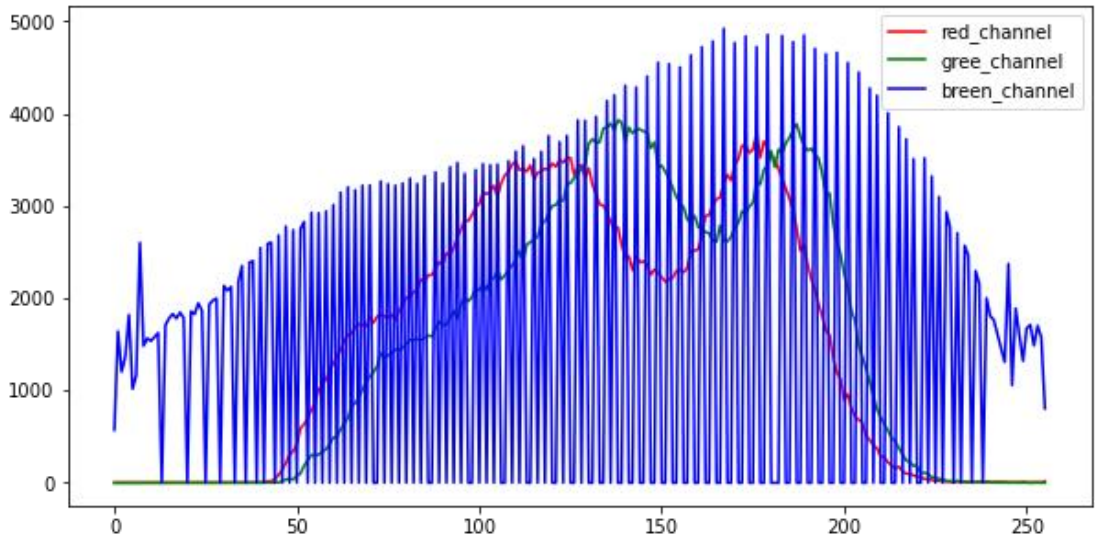
接下來看到一組效果比較不好的圖片，可以看到原圖有一些陰影的部分，在二值化後的效果很不好，其原因是陰影部分很容易被分成同一類，並都轉換成 255 的灰階值，這在視覺上就會造成效果不太好，無法使我們分辨陰影的位置。



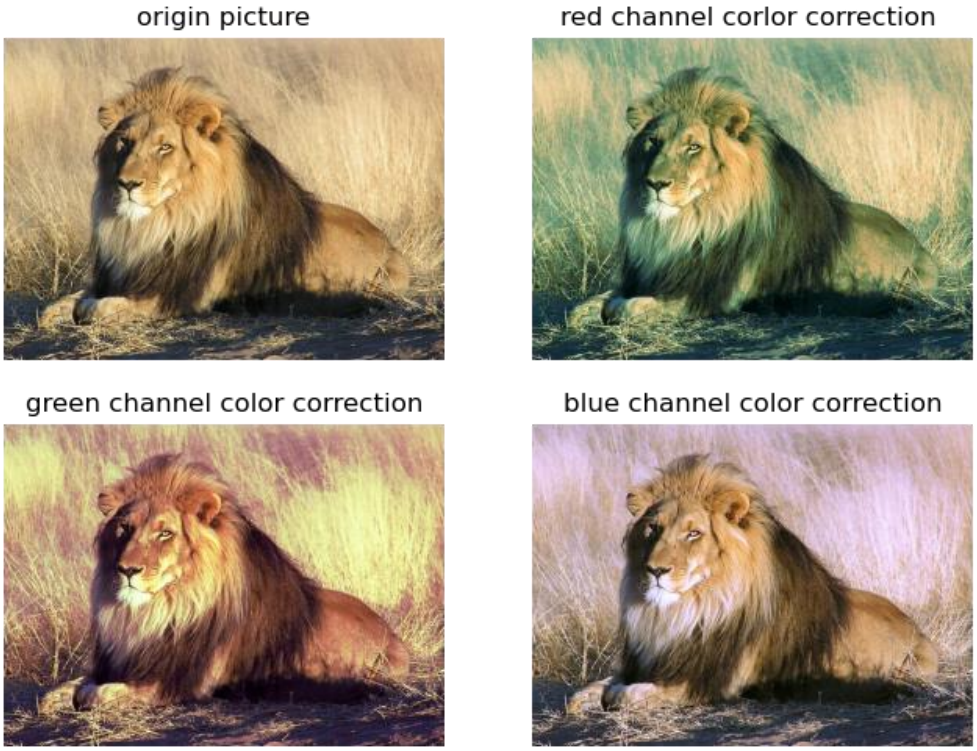
5.color correction

我先試著對各個單一通道進行 histogram equlization，試試看如果單純對某一通道調整其顏色比例能否使圖片顯色效果更好，以下圖為例，就是只對然色通道進行 equlization





其成果對比可以看到如下圖，可以看到只對紅色通道做處理的圖片，其整體偏綠，對綠色通道做處理，整體顏色更為亮麗，顏色分布的也較為均衡，整體顏色偏亮黃，對藍色通道做處理，整體顏色發紫，但卻較為銳利的感覺



接著拿人像圖做實驗，，不同顏色通道達到的效果都不一樣，針對紅色通道，人臉整體氣色更加紅潤，並且也更加均衡，針對綠色通道處理，整體顏色偏黃，也添加一股復古的感覺，針對藍色通道，整體感覺篇冰冷，皮膚也更顯白。

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
def read_picmode(figpath,mode='rgb',resize=None):#決定讀取黑白還彩色
    if (mode=='rgb'):
        pic=cv2.imread(figpath)
        pic = cv2.cvtColor(pic, cv2.COLOR_BGR2RGB)
    elif (mode=='gray'):
        pic=cv2.imread(figpath, cv2.IMREAD_GRAYSCALE)
    if resize!=None: #resize 是否
        pic =
cv2.resize(pic,(int(pic.shape[1]/resize),int(pic.shape[0]/resize)) ,
interpolation=cv2.INTER_AREA)

    return pic
pic=read_picmode('picdir/animal2.jpg',mode='gray')

#各種不同的 filter
box_filter=1/9*np.array([[1,1,1],[1,1,1],[1,1,1]])
Gaussian_Filter=1/16*np.array([[1,2,1],[2,4,2],[1,2,1]])
sobel_filter_h=np.array([[1,0,-1],[2,0,-2],[1,0,-1]])
sobel_filter_w=np.array([[1,2,1],[0,0,0],[-1,-2,-1]])
Laplacian_Filters=np.array([[0,-1,0],[-1,8,-1],[0,-1,0]])


#增加隨機噪聲
def add_noise(img,n,mode='white'):
    noice_pic=np.array(img)
    for _ in range(n):
        x = int(np.random.random() * img.shape[0])
        y = int(np.random.random() * img.shape[1])
        if mode=='white':
            noice_pic[x, y] = 255 # 白色的灰階值是 255
        elif mode=='black':
            noice_pic[x, y] = 0
    return noice_pic
noice_pic=add_noise(pic,10000,mode='black')
#卷積 可套用在各種空間濾波器
def convolution(img, kernel):
```

```
    row_vector=kernel.flatten()
    kernel_size=kernel.shape[0]
    img_hsize=img.shape[0]
    img_wsize=img.shape[1]
    new_img_hsize=img_hsize-kernel_size+1
    new_img_wsize=img_wsize-kernel_size+1
    new_img=np.zeros((new_img_hsize,new_img_wsize)) #準備新圖片
    for i in range (new_img.shape[0]):#垂直方向
        for j in range (new_img.shape[1]):#水平方向
            value = 0
            column_vector = []
            for ki in range(kernel_size):
                for kj in range(kernel_size):
                    column_vector.append(img[i+ki][j+kj]) # 從 img 中取出數值放入
column_vector
            new_img[i][j] = row_vector @ column_vector #@為內積運算
    new_img=new_img.astype(int)
    new_img=np.maximum(new_img,0)
    return new_img
# 中位數濾波器
def median_filter(img,filter_size):
    temp = []
    indexer = filter_size // 2 #filter 寬度
    data_final = np.zeros(((img.shape[0]),(img.shape[1]))) #準備新圖片
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):

            for z in  range(filter_size):
                if i+z-indexer<0 or i+z-indexer>(img.shape[0])-1 :
#超過圖像長寬 append0
                    for c in range(filter_size):
                        temp.append(0)
                else :
                    if j+z-indexer<0 or j+indexer>(img.shape[1])-1:
                        temp.append(0)
                    else :
                        for k in range(filter_size):
                            temp.append(img[i+z-indexer][j+k-indexer])
#將 filter 所到之值 apeend 到 list 中
            temp.sort() #按造順序排序
```

```
            data_final[i][j]=temp[len(temp)//2] #挑取中位數並添加到新圖片中
            temp.clear()
    return data_final
#平均值濾波 跟中位數濾波作法差不多
def mean_filter(img,filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = np.zeros(((img.shape[0]),(img.shape[1])))
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):

            for z in  range(filter_size):
                if i+z-indexer<0 or i+z-indexer>(img.shape[0])-1 :#超過圖像寬度
                    for c in range(filter_size):
                        temp.append(0)
                else :
                    if j+z-indexer<0 or j+indexer>(img.shape[1])-1:
                        temp.append(0)
                    else :
                        for k in range(filter_size):
                            temp.append(img[i+z-indexer][j+k-indexer])
            mean=sum(temp)/len(temp)
            data_final[i][j]=mean
            temp.clear()
    return data_final

#套用各種 filter 並將結果可視化
median_filter_pic=median_filter(noice_pic,3)
mean_filter_pic3=mean_filter(noice_pic,3)

laplace_pic=convolution(pic,Laplacian_Filters)
sobel_h_pic=convolution(pic,sobel_filter_h)
sobel_w_pic=convolution(pic,sobel_filter_w)
fig=plt.figure(figsize=(16,12),dpi=100)
plt.subplot(3,2,1)
plt.imshow(noice_pic,cmap='gray')
plt.title('noise picture',fontsize=18)
plt.subplot(3,2,2)
plt.imshow(mean_filter_pic3,cmap='gray')
plt.title('mean_filter 3*3',fontsize=18)
```

```
plt.subplot(3,2,5)
plt.imshow(gaussin_pic,cmap='gray')
plt.title('Gaussin filter',fontsize=18)
plt.subplot(3,2,6)
plt.imshow(median_filter_pic,cmap='gray')
plt.title('median_filter',fontsize=18)

laplace_pic=convolution(pic,Laplacian_Filters)
sobel_h_pic=convolution(pic,sobel_filter_h)
sobel_w_pic=convolution(pic,sobel_filter_w)

fig=plt.figure(figsize=(16,12),dpi=200)
plt.subplot(2,2,3)
plt.imshow(sobel_w_pic,cmap='gray')
plt.title('detect horizontal line',fontsize=18)
plt.subplot(2,2,4)
plt.imshow(sobel_h_pic,cmap='gray')
plt.title('detect vertical line',fontsize=18)
plt.subplot(2,2,2)
plt.imshow(laplace_pic,cmap='gray')
plt.title('Laplacian operator mask',fontsize=18)
plt.subplot(2,2,1)
plt.imshow(pic,cmap='gray')
plt.title('origin pic',fontsize=18)

#二值化
def binary(pic,value):
    pic_height=pic.shape[0]
    pic_width=pic.shape[1]
    new_pic =np.zeros((pic_height,pic_width))
    for i in range (pic_height):
        for j in range(pic_width):
            if pic[i][j]>value: #高於 threshold
                new_pic[i][j]=255
            else :
                new_pic[i][j]=0
    return new_pic
new_pic=binary(test_pic,100)

#反向二值化
```

```
def image_negative(pic,value=0):
    pic_height=pic.shape[0]
    pic_width=pic.shape[1]
    #二值化
    new_pic =np.zeros((pic_height,pic_width))
    for i in range (pic_height):
        for j in range(pic_width):
            new_pic[i][j]=255-pic[i][j]
    return new_pic
negative_pic=image_negative(test_pic,80)

#計算 image histogram
def cal_plot_histogram(pic,mode='rgb'):
    height=pic.shape[0]
    weight=pic.shape[1]
    image_histogram=np.zeros(256)
    rgb_histogram=np.zeros((3,256))
    plt.figure(figsize=(10,5))#繪出 histogram
    if (mode=='rgb'):
        for i in range(height):
            for j in range(weight):
                value=pic[i][j]
                rgb_histogram[0][value[0]]=rgb_histogram[0][value[0]]+1 #r
                rgb_histogram[1][value[1]]=rgb_histogram[1][value[1]]+1 #g
                rgb_histogram[2][value[2]]=rgb_histogram[2][value[2]]+1 #b
        plt.plot(rgb_histogram[0],c='r',label='red_channel')
        plt.plot(rgb_histogram[1],c='g',label='gree_channel')
        plt.plot(rgb_histogram[2],c='b',label='breen_channel')
        plt.legend()
        return rgb_histogram
    else:
        for i in range(height):
            for j in range(weight):
                value=pic[i][j]
                image_histogram[value]=image_histogram[value]+1 #計算每個亮度值得
個數
        plt.bar(range(256),image_histogram,label='gray_level')
        plt.legend()
        return image_histogram
histogram=cal_plot_histogram(pic,mode)
```

```
# 將 histogram equalization
def histogram_equlization(histogram,mode):
    if (mode=='rgb'):
        image_pdf=histogram/(pic.size/3)
        image_cdf=np.zeros((3,256))
        convert_pic=np.zeros((pic.shape[0],pic.shape[1],3))
        for i in range(3):
            image_cdf[i]=np.cumsum(image_pdf[i])
        image_equ_value=np.around(image_cdf*255).astype('uint8')
        for i in range(3):
            convert_pic[:,:,i]=(image_equ_value[i][pic[:,:,i]])
        convert_pic=convert_pic.astype('uint8')
    else:
        convert_pic=np.zeros((pic.shape[0],pic.shape[1]))
        image_pdf=histogram/pic.size
#pic.size=height*weight pdf 為每個亮度出線的機率
        image_cdf=np.cumsum(image_pdf)#累積機率並用於 normalization
        image_equ_value=np.around(image_cdf*255).astype('uint8')
#返回四捨五入的值 #代表原本的亮度應該對應到的亮度
        convert_pic = image_equ_value[pic]
#在 pic 座標的亮度對應到 equal_value 後應該為多少亮度 輸出 shape 為 pic 的 shape
        convert_pic=convert_pic.astype('uint8')
    return convert_pic

convert_pic=histogram_equlization(histogram,mode)

newhis=cal_plot_histogram(convert_pic,mode)
#對轉換後的圖片將其 histogram 可視化出來

#將成果對比圖可視化出來
fig=plt.figure(figsize=(10,5),dpi=100)
plt.subplot(1,2,1)
plt.imshow(pic)
plt.title('origin picture',fontsize=10)
plt.subplot(1,2,2)
plt.imshow(convert_pic)
plt.title("histogram equlization picture",fontsize=10)

#color correction
```

```
def power_transform(mode,pic):
    if (mode=='gray'):
        height=pic.shape[0]
        width=pic.shape[1]
        new_pic=np.zeros((height,width))
        for i in range(height):
            for j in range(width):
                new_pic[i][j]=pic[i][j]**0.7 #對灰階做轉換
    else:
        new_pic=np.array(pic)
        height=pic.shape[0]
        width=pic.shape[1]
        for i in range(height):
            for j in range(width):
                new_pic[i][j][0]=pic[i][j][0]**0.8 #只對紅色 channel 做調整
    return new_pic
new_pic=power_transform(mode,pic)
#可以針對某一個顏色通道做 equlization
def onechannel_equlization(histogram,channel):
    image_pdf=histogram[channel]/(pic.size/3)
    stand_by_pic=np.array(pic)
    image_cdf=np.cumsum(image_pdf)
    image_equ_value=np.around(image_cdf*255).astype('uint8')
    stand_by_pic[:,:,channel] = image_equ_value[pic[:,:,channel]]
    return stand_by_pic
convert_picr=onechannel_equlization(histogram,channel=0)#針對 red 通道
convert_picg=onechannel_equlization(histogram,channel=1)#針對 green 通道
convert_picb=onechannel_equlization(histogram,channel=2)#針對 blue 通道

#用 plt 可視化
plt.figure(figsize=(8,5),dpi=100)
plt.subplot(2,2,1)
plt.axis('off')
plt.title('origin picture')
plt.imshow(pic,cmap='gray')
plt.subplot(2,2,2)
plt.axis('off')
plt.title('red channel corlor correction ')
plt.imshow(convert_picr)
plt.subplot(2,2,3)
```

```
plt.axis('off')
plt.title('green channel color correction')
plt.imshow(convert_picg)
plt.subplot(2,2,4)
plt.axis('off')
plt.imshow(convert_picb)
plt.title('blue channel color correction')
```