INF250 EXAM 2020

Image Processing and Analysis

NMBU

Candidate number: 9

Index

Exercise 1 – Image sharpening

In this exercise, the goal was to sharpen a blurred image. The image was first transformed into an 8-bit image, then the “USM”-algorithm was applied to make that image sharper.

The «Unsharp masking» filter sharpens the image, by subtracting a smoothed version of image from the original, to then be added to the original image to enhance the edges there. In other words, the process follows 3 main steps. The first step is to obtain a smoothed version of the original image. The original image consists of a lot of varying pixel values, and the aim of the smoothing process is to replace each pixel value by an average of surrounding pixels in a particular local frame-window of pixels. By moving this window-operator over the whole image, one can obtain a lowpass-filtered image. The lowpass-filtered image removes the high special frequency noise, by averaging the pixelvalues according to their surrounding, neighborhood values. In this case the Gaussian Smoothing filter was used to obtain the smoothed image. The Gaussian Smoothing filter is a lowpass filter operator, which gives more weight to the central pixels than the neighboring pixels. In contrast to a more uniformly weighted average, such as a mean-filter, giving more weight to the central pixels results in gentler smoothed image and the edges are more preserved. In other words, the image is smoothed with less loss of information preserved in the edges.

The second step is to obtain a sharper image. The smoothing process is achieved by preserving the lower frequencies and filtering out the higher frequencies, the sharper edges. So, by subtracting this smoothed version from the original image, the result will consist of the higher frequencies and sharper edges which were “filtered out” in the smoothing process. So, it is possible to obtain a sharper image where the edges are intensified and more defined than in the original image. This is called a mask.

Last step is to add this mask to the original image, resulting in a sharper image. The amount of sharpening is controlled by a weight-factor.

See Appendix 1: “Python script Exercise 1” for more information about the programming aspect for the methods performed.

Original RGB image

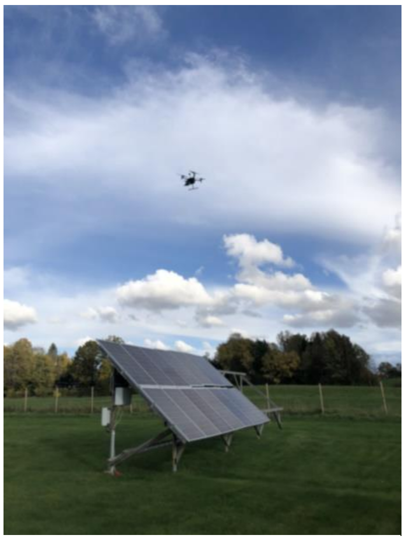


Figure 1- Blurred drone image RGB

8-bit image and the sharpened 8-bit image

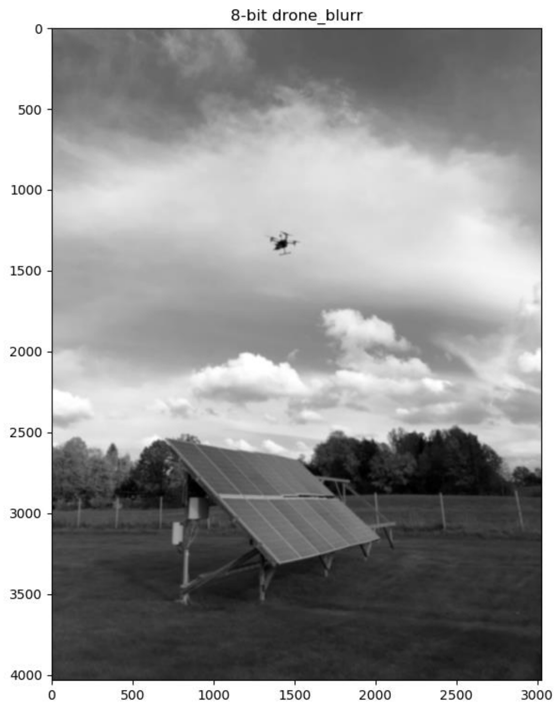


Figure 3- Blurred drone image 8 bit

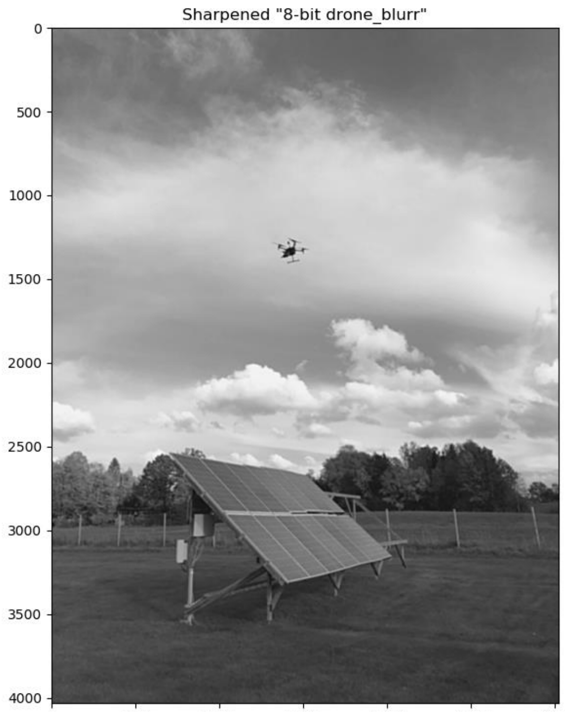


Figure 2- Sharpened drone image 8 bit

The sharpening effect are made visible when comparing the objects like the solar panel, drone and clouds, to point out a few.

Exercise 2 – Distinguish types of defects

The goal for this section is to distinguish two different types of errors. One is a string and concentrated type of defects, while the other one is more separated, individual defects. The defects differ in pattern and texture, so a texture analysis was used to achieve the goal.

Original images (String error and individual errors)

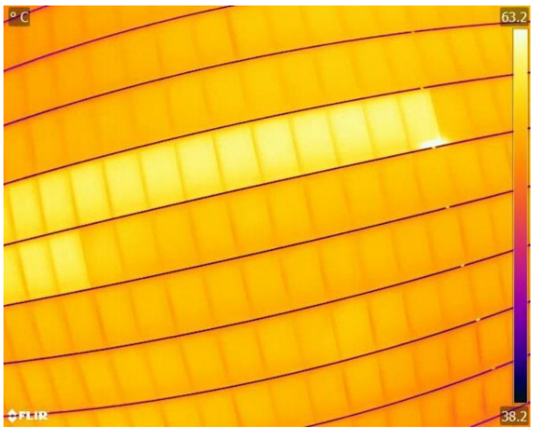


Figure 4 - String Error Original image

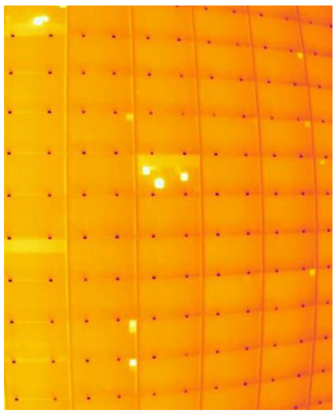


Figure 5- Individual Errors Original image

The first step was to extract the "color-channel" out of red, green and blue, which had the biggest variance in pixelvalues in the image. In this case, the blue channel was the superiour one. Here the difference in errors are more visable than in the original images.

Images (Blue channels)

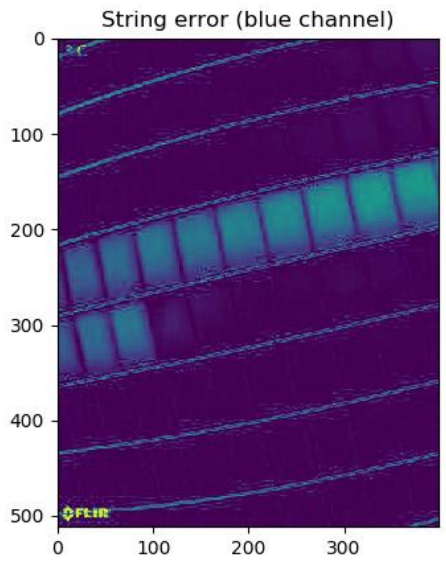


Figure 7 - String Error (Blue channel)

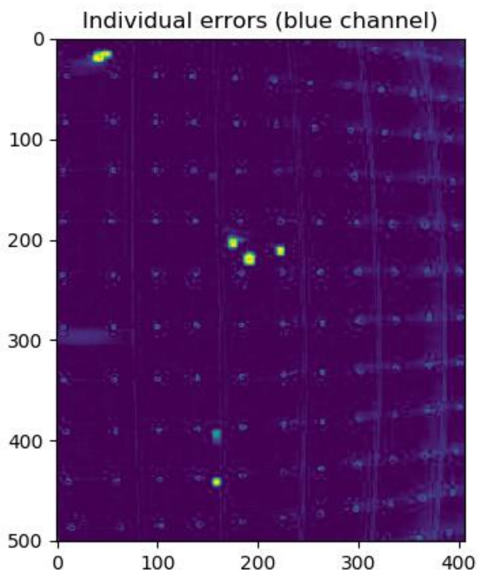


Figure 6- Individual Errors (Blue channel)

Second step was to analyse the histograms of pixelvalues and intesity for the respective, colorextracted images.

Histograms of images (blue channel)

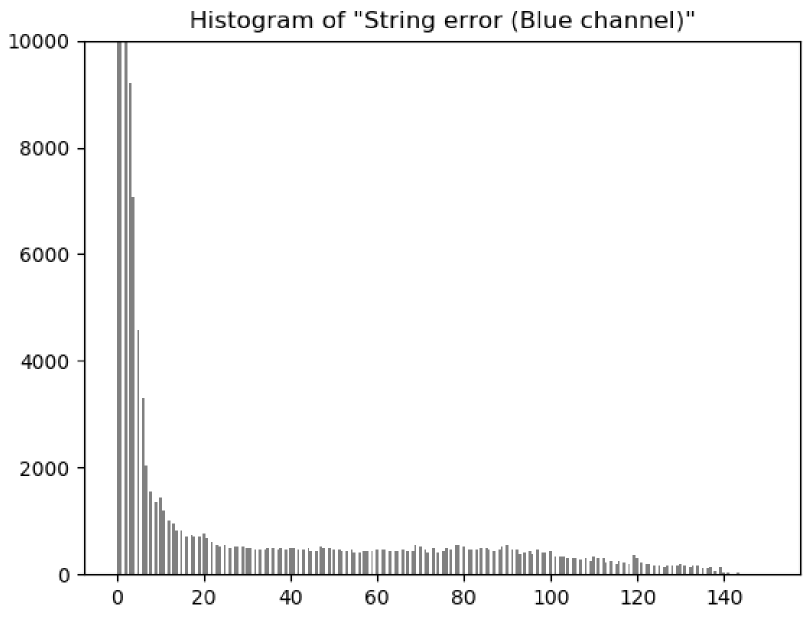


Figure 8- Histogram of "String error (Blue channel)"

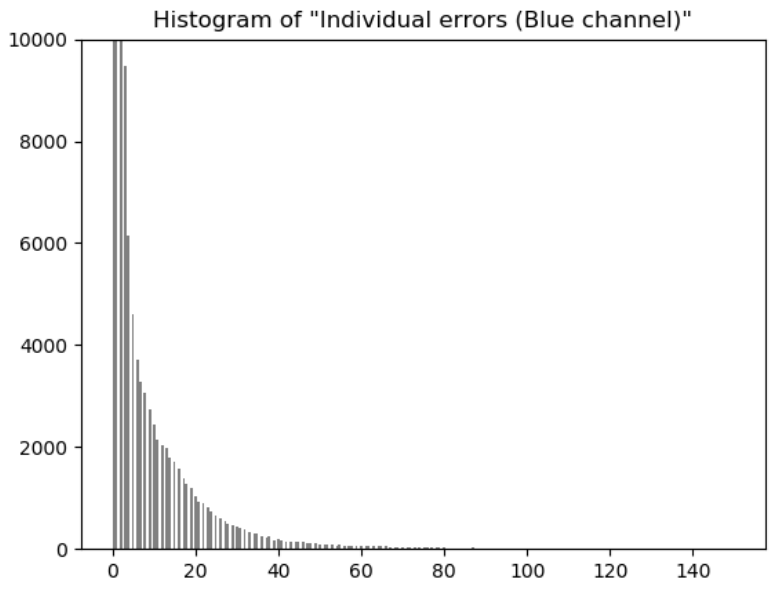


Figure 9- Histogram of "Individual errors (Blue channel)"

When comparing these two, the main indication of the difference in types of error lays in the spread in pixelvalues. The image containing the "string"-type error, has a bigger spread, more pixel in the higher values. This is a consequence of the “string”-type image, containing more defect area compared to the image of “individual” errors. The “individual”-type has a more concentrated range in the lower region of pixelvalues, a result of less area covered by defects. In this particular instance, with the “blue”-channels and the belonging histogram of pixelvalues, it is possible to distinguish between the two types of defects.

In this case, the main difference that separated these types of errors, was the distribution of defects relative to non-defected area. In another case where the area covered by defects are about the same in both cases of errors, like the distribution of errors are about the same for both cases even tough the types of errors are different, this approach would not effectively distinguish the errors. The approach chosen in this case, was 1. order statistics, which comes with limitations, like not being able to distinguish between patterns. Ideally one would benefit more from a 2. Order statistics approach, where difference in patterns and texture are more effectively detectable. 2. Order statistics -methods like “Grey Level Co-occurrence matrix” would be preferred in most cases of these kind, but there occurred some limitations in understanding and interpreting the results when this method was applied when programming. As a conclusion, the difference in 1. orders statistics was clear enough to distinguish the two types of defects presented in this case.

See Appendix 2: “Python script Exercise 2” for more information about the programming aspect for the methods performed.

Exercise 3 – Increasing contrast

There are several methods that can be applied to increase the contrast in an image. Point operation is a process involving modification of the individual pixelvalue without changing the local structure of the image. With this point operation, one can increase the contrast by individually increasing the values of each pixel or apply methods such as "Histogram equalization". Histogram equalization converts an un-linear cumulative histogram of the pixels in the original image to a modified image with approximately linear cumulative histogram of the pixels, by applying a suitable point operation. In other words, by applying a point operation, the pixels are more evenly spread by stretching out the intensity range of the original image. In this case, adaptive equalization was preferred over ordinary equalization, because of more accurate result compared to reference-image. The "Adaptive Histogram Equalization"- method differs from normal equalization by "computing several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image." By doing so the local contrast is improved and the definitions of edges in every region are more enhances in image (Sudhakar, S. 2017).

First step was to convert the image to an 8-bit image

8-bit version of image

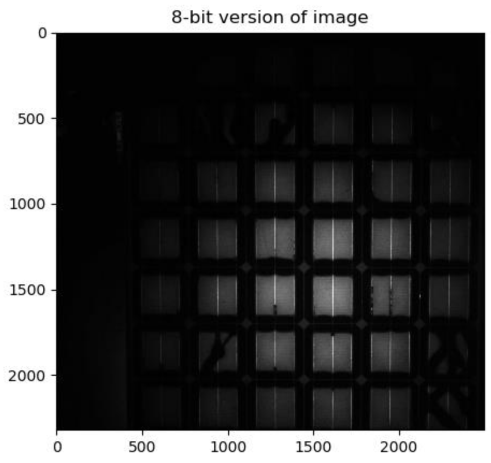


Figure 10 - 8-bit version of image

Then applying the point operators to increase the contrast. In this case three methods were being used. First method is simply increasing every single pixelvalue by the same amount to achieve increase in contrast. This method resulted in less efficient image, because the center of the image was more exposed to the modification than the surroundings. Higher weight-amount was necessary to increase the darker regions, resulting in the center getting overexposed. This could be altered by the weight-amount, but an altered image like the reference image was not achieved by this method.

Images after uniformal increase in contrast

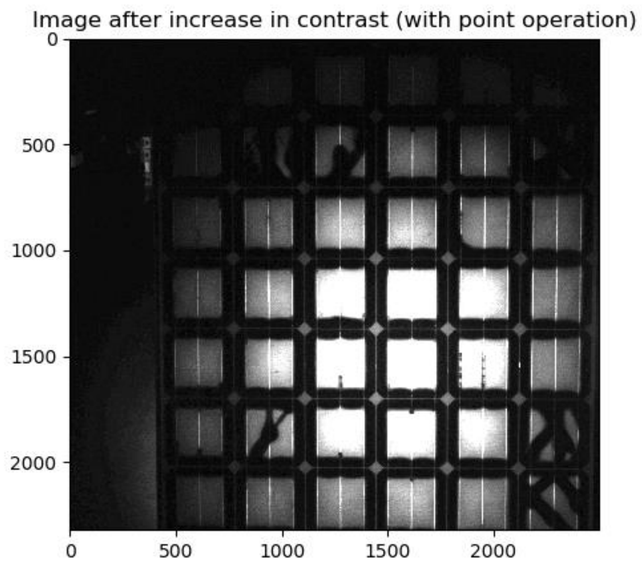


Figure 11- Increased contrast image

Therefore, the histogram equalization method was preferred.

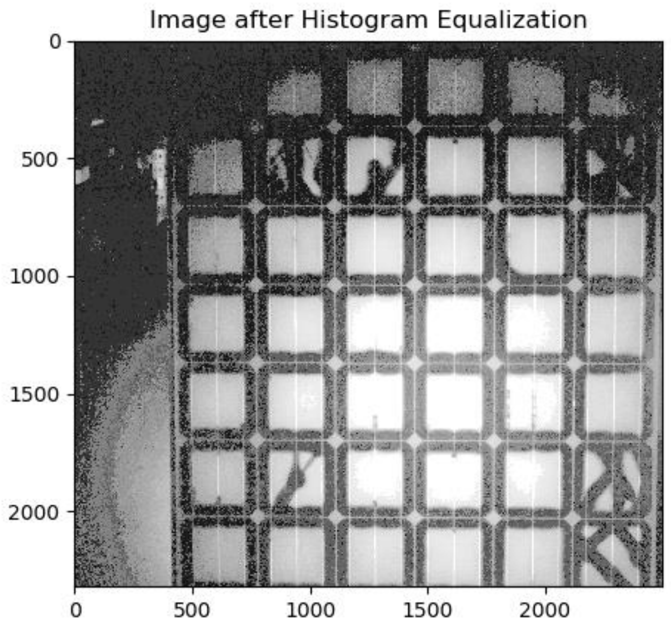


Figure 12- Image after Histogram equalization

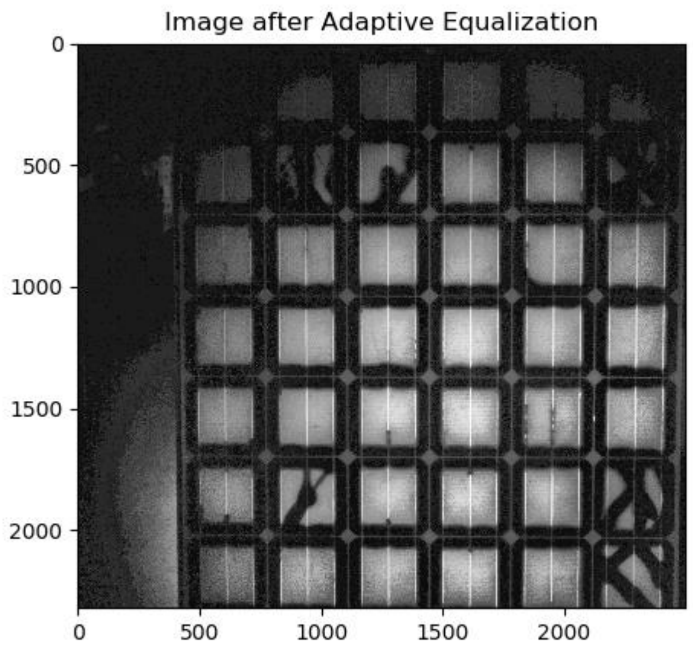


Figure 13- Histogram after Adaptive Equalization

By analyzing the respective histogram of the specific images, the increase in contrast can be visualized in terms of spread in pixel values and their intensity.

Histograms

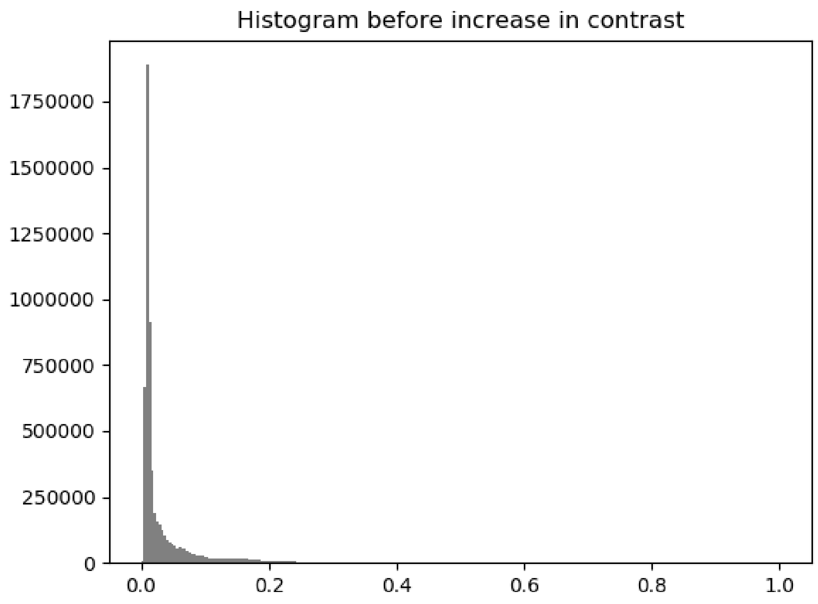


Figure 14 - Histogram of original 8-bit image

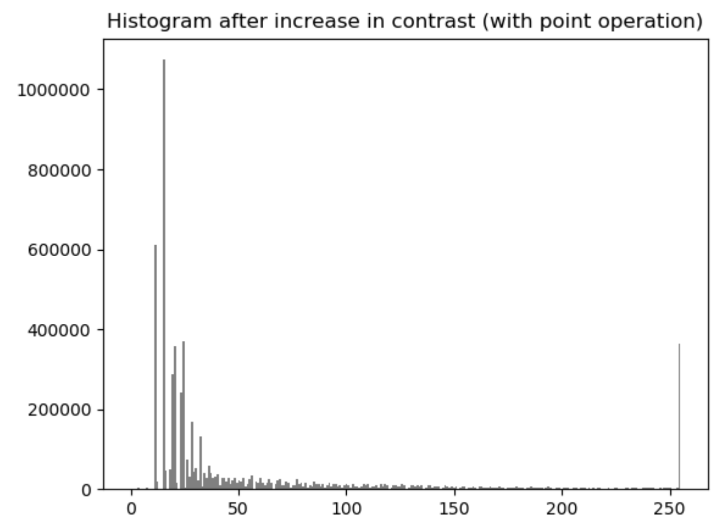


Figure 15- Histogram after increase in contrast

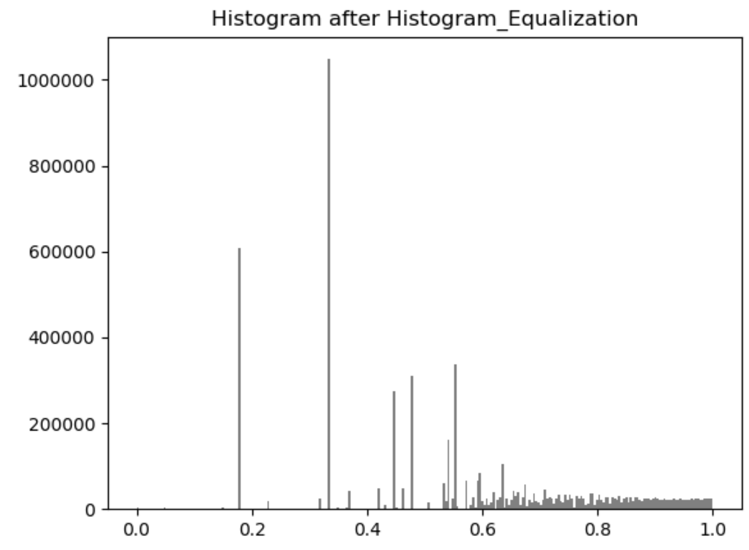


Figure 16 - Histogram after "Histogram Equalization"

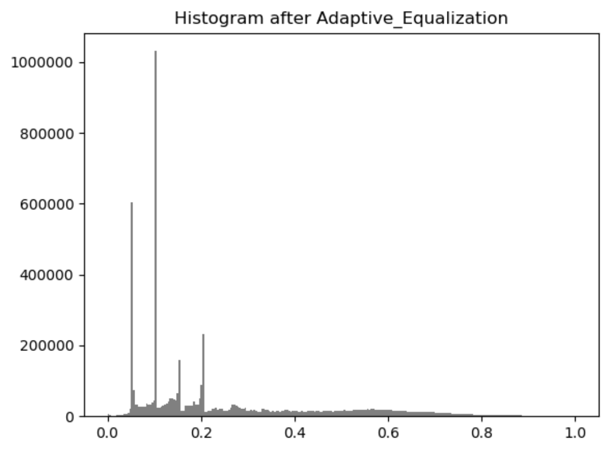


Figure 17- Histogram after "Adaptive Equalization"

One can cleary tell the difference in the spread of pixel-intensity values before and after implementation of point operations. All three operations sucsessfully increased the contrast, but comparred to the reference image, image after adaptative histogram equalization was preferred.

Lastly, the binary image was achived by applying the “Otsu”-treshold method to the “adaptive equalized”-image. By maximizing the inter-class variance, an optimal treshold was selected to separate pixels into two classes, foreground and background, resulting in a binary image.

Binary image

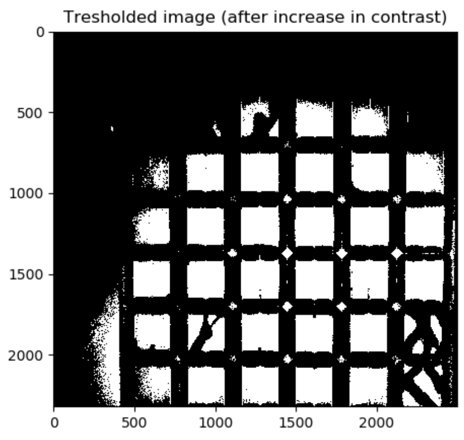


Figure 18 - Binary image after increase in contrast

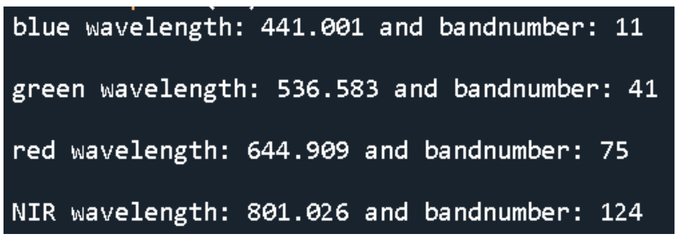
After doing the following steps, the result ended up being an image where the cracks are made way more visible than in the original image.

See Appendix 3: “Python script Exercise 3” for more information about the programming aspect for the methods performed.

Exercise 4 – Hypersprectral analysis

1) Locate 3 wavebands in red, green and blue wavelength and display a RGB image

To locate the bandnumber for the respective channels, the band corresponding to the best matching wavelength from data compared to set wavelengths, where chosen. Blue = 440 nm, Green = 535 nm, Red = 645 nm and NIR = 800 nm, were set beforehand and the wavelengts which were closest to these set wavelengts for the respective channels, where chosen. This was done by going trough every wavelenghs in given dataset and pulling out the wavelength that gave the smallest difference with the "standard" wavelengts given. By doing so for every channles, the bandnumbers were located by the found wavelengt's ordernumber in dataset. (See Appendix 4, Line 24 – 80).



```
blue wavelength: 441.001 and bandnumber: 11

green wavelength: 536.583 and bandnumber: 41

red wavelength: 644.909 and bandnumber: 75

NIR wavelength: 801.026 and bandnumber: 124
```

Figure 19- Wavelengths and bandnumber

With the found wavelengths for red, green and blue, an RGB image was comptuded by adding the bandnumbers according to the respective colorchannels.

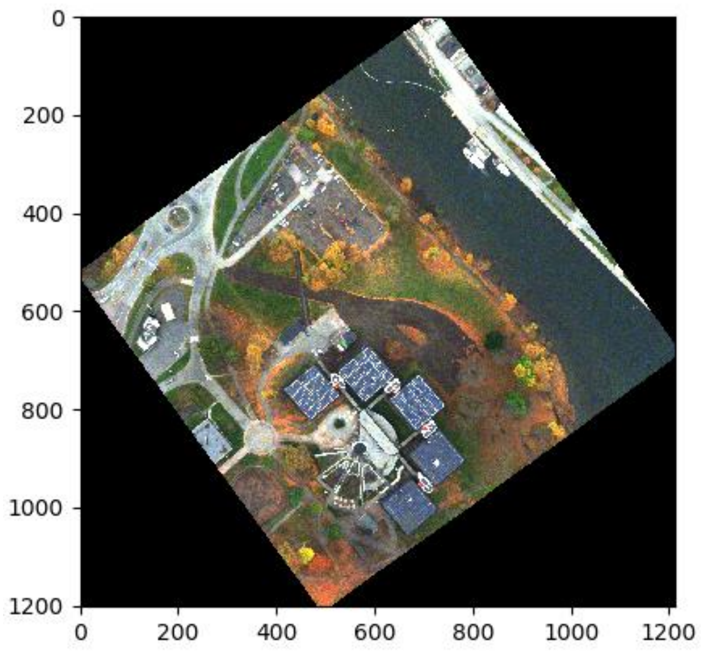


Figure 20 - RGB image

2) Spectrum from the following materials in the image: grass, solar panels, asphalt and water.

This information was obtained by selecting a particular pixel corresponding to the respective elements and plotting its values in the spectrum range.

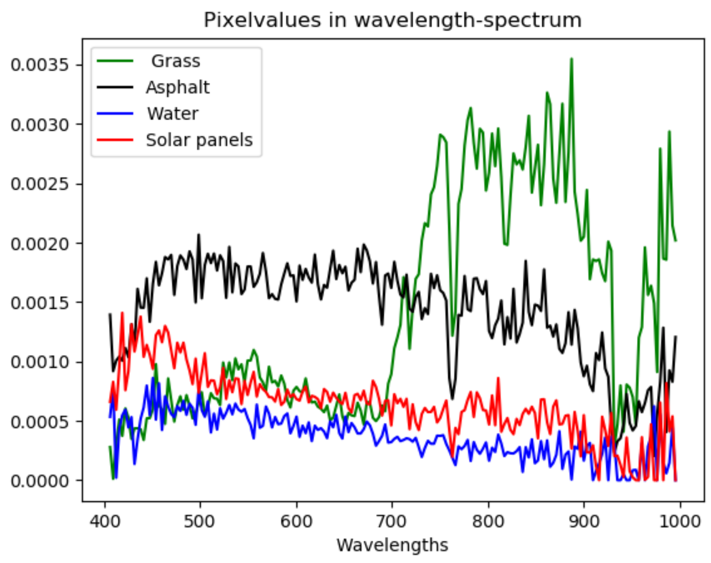


Figure 21 - Spectrum of different materials

3) Mean spectrum of each material

This information was obtained by selecting a 20x20 pixelarea corresponding to the respective elements and plotting its values in the spectrum range.

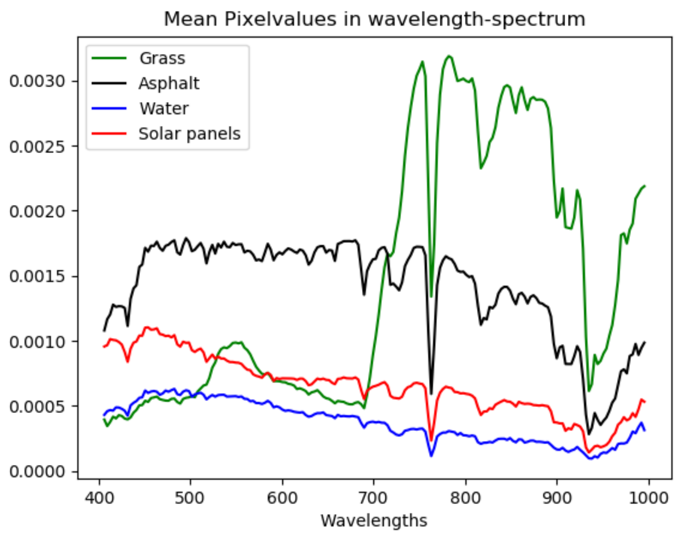


Figure 22 - Mean spectrum of each material

The grass-area is recognizable by having peaks at the green-wavelengts (500-600 nm) and the NIR-wavelengths (~800). The peak in NIR-wavrelenth is a result of vegetation reflecting the more of the infrared-radiation compared to the other regiontypes.The water-region is charactirzed by the peak at the blue-wavelengt range(400-500nm). The same goes for the blue-colored solarpanles. As for the asphalt region, there are no significant charastiristics about them in the plot, because of the mixture in colours.

4) NDVI image

The NDVI index is a graphical indicator to observe vegetation, computed by relation between reflected red wavelengths and near-infrared wavelengths. The index is calculated by (NIR-RED)/ (NIR + RED). By applying this formula to the hyperspectral image, the NDVI image was acquired.

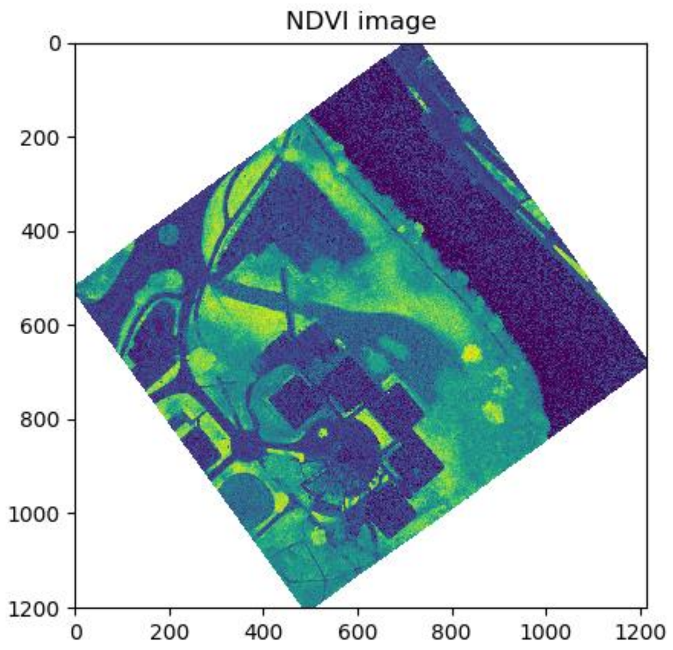


Figure 23 - NDVI image

The green-regions are mostly vegetation, that reflects the infrared-radiation. The lighter the green color is, the healthier the vegetation is supposed to be. The healthier vegetation, the more infrared-radiation is reflected. The darker regions are materials/regions that absorbs more of the radiations, such as the water region, asphalt and solar panels.

5) PCA and display the first 3 score images together with the loading plots.

Principal component analysis transforms the hyperspectral image to only contain uncorrelated features between the bands. The hyperspectral image contains of many bands that are correlated and therefore don't contribute to additional information about the image. By doing PCA, the "more important and distinct" features in every band is extracted, resulting in image containing more denser information about the image. The results of the analysis are the "Score image", containing the uncorrelated features from bands, and "Loading plot" contain information about the contribution from the bands. The higher, the more influence on the separation.

1. Score image and loading plot

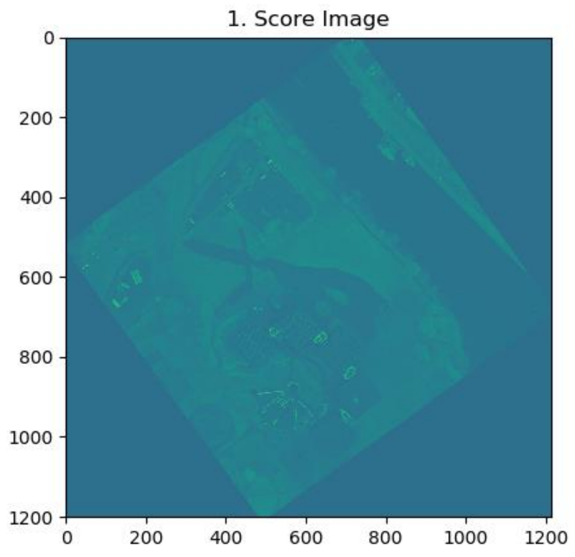


Figure 25 - 1.Score image

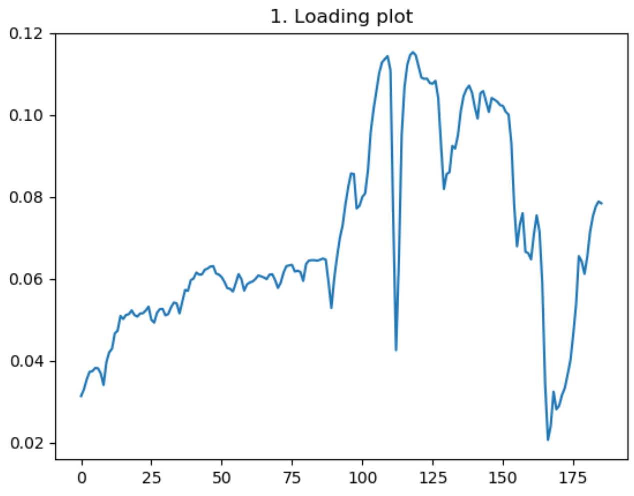


Figure 24 - 1.Loading plot

In the first score image, it seems like the metal parts are made visible by a lighter color than the surroundings. The solar panels can be identified by the yellow outlines made visible by the metal components in the panels. One can argue that vegetation can also be identified by the less lighter color, but that doesn't pop put as much as what seems to be metal parts.

2. Score image and loading plot

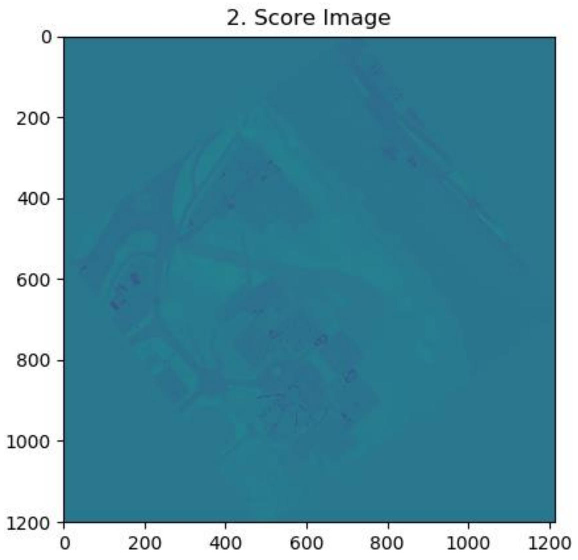


Figure 27 - 2.Score Image

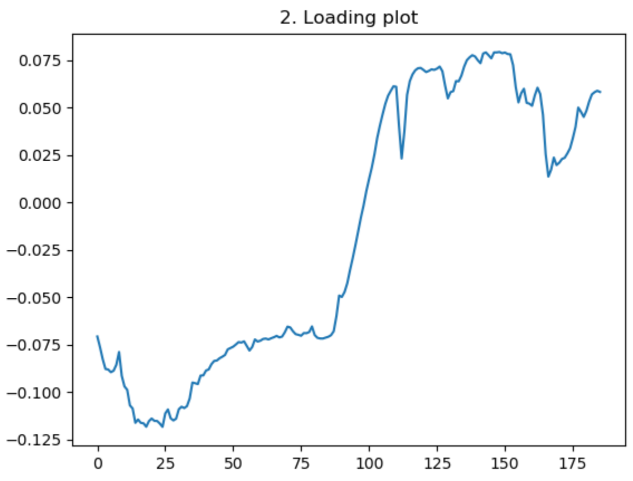


Figure 26 - 2.Loading plot

The second score image seems to give metal parts a darker tone than the surrounding regions. Though there are variations in image, they are less clear than the first score image, and doesn't add much more information than the 1. Score image.

3. Score image and loading plot

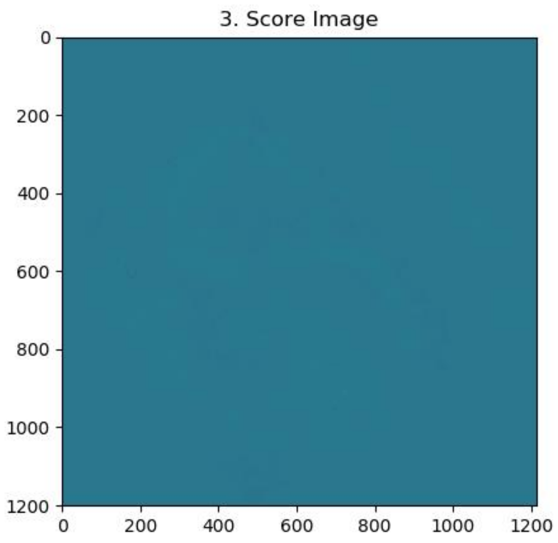


Figure 29 - 3.Score image

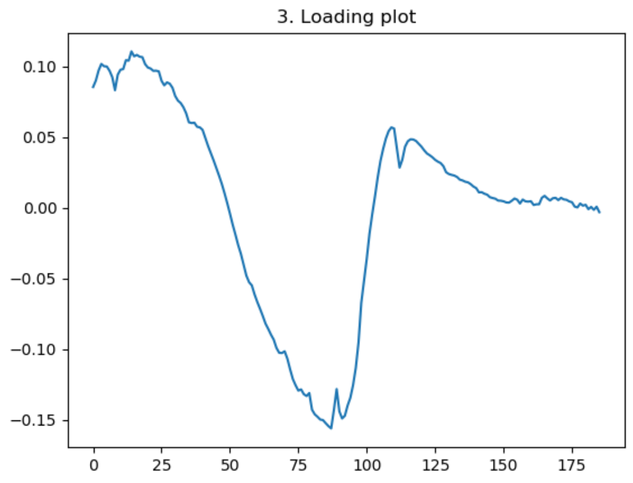


Figure 28 - 3.Loading plot

The third score image doesn't contribute to any new information.

6) K-means clustering

K-means clustering method is an unsupervised alghorithm, which means no traning or labeling are given beforehand. The method is used to image segmentation, by grouping pixels/data points which have simimalar traits and attributes. According to given number of clusters/classes and iterations, the process repeatedly assigns data poitns to nearest given cluster, resulting in groups with similar attributes. The K-means was given 5 clusters and 20 iterations, but stopped at 4 clusters and 18 iterations.

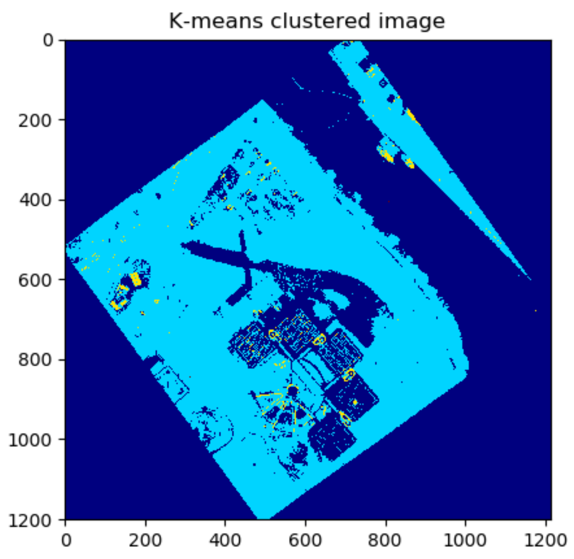


Figure 30 - K-means clustered image

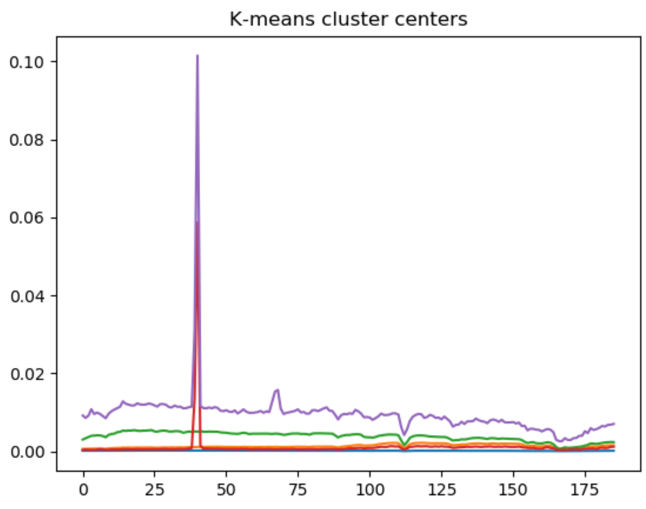


Figure 31 - K-means cluster centers

The process seems to separate metal parts, lighter regions and darker regions. Not as informative as expected but given the amount of information and different objects in image, an unsupervised classification would not effectively extract information. Supervised methods would be preferred.

7) Gaussian maximum likehood classification

GLCM is, unlike K-means, is a supervised classification method. In other words, a set of training data is provided beforehand. The training data contains labelled pixel-samples from original image. A set of pixels are assigned to a class, and with this information, the method tries to assign rest of the pixels to the given classes, based on similarity in attributes given in the original dataset.

Before performing the GMLC-method, the different regions were classified in a training image. A set of grouped pixels were given labels according to the region-type they represented. Below the training image is shown, where:

- Darker blue is "Grass"
- Lighter green is "Parking lot"
- Yellow is "Sand and Orange leaves"
- Blue is "Water"
- Green is "Darker road"
- Darker green is "Solar panel"
- Purple is "Asphalt"

Training image

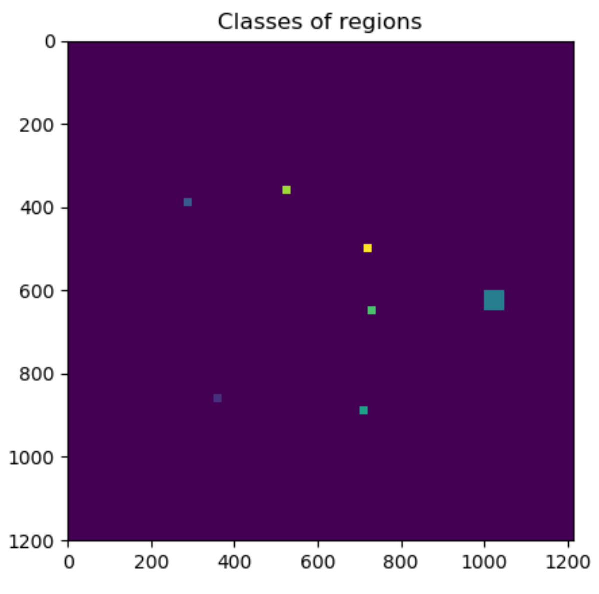


Figure 32 - GLCM training image

With these groundtruths and the training image, the GLCM method was performed and provided an image to classify the regions, using the same spectral bands as the training set.

Below is the classification map, which shows the classification results for the reference image analyzed.

The regions classified in image, can be identified by their representing colors:

- Green: 'Healthy Grass'
- Brown: 'Orange leaves and less healthy vegetation"
- Red: "Asphalt"
- Turquoise: "Parking lot"
- Pink: "Darker road"
- Dark blue: "Water"

Classification image and RGB image

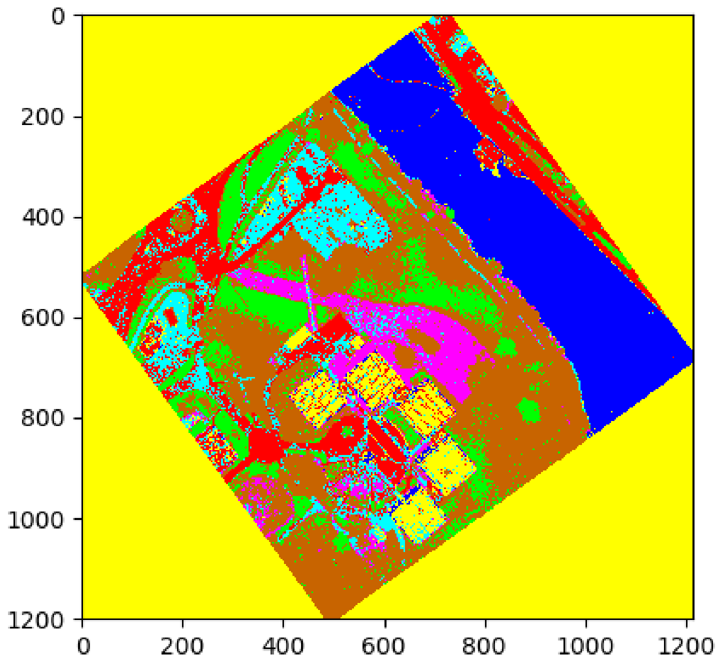


Figure 34 - GLCM classification image

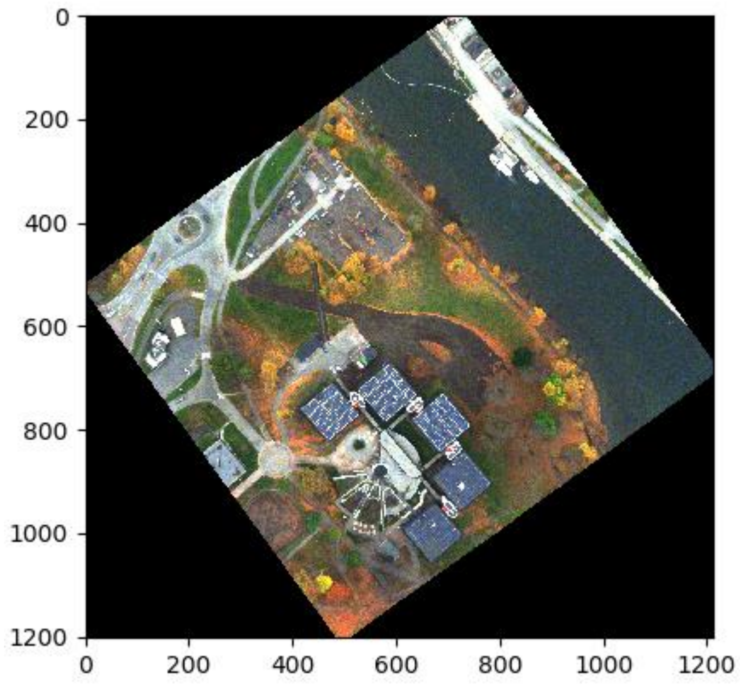


Figure 33 - RGB image

When compared to the "Original" RGB image, the result image came out well. Even though the image came out more pixelated and with lower resolution, the main sections of the respective regions are well identifiable. The classification image can give even better results, when given more classes and reference points, but based on the given classes and points, the result came better than expected.

See Appendix 4: "Python script Exercise 4" for more information about the programming aspect for the methods performed.

Sources

External

Sudhakar, S. (2017). “Histogram Equalization”. Obtained from: https://towardsdatascience.com/histogram-equalization-5d1013626e64 (14.12.2020)

Scikit-image. *Histogram equalization.* Obtained from: Histogram Equalization — skimage v0.19.0.dev0 docs (scikit-image.org) (12.12.2020)

Scikit-image. *Local Otsu Threshold.* Obtained from: Local Otsu Threshold — skimage v0.12.2 docs (scikit-image.org) (12.12.2020)

Spectral python. *Sprectral Algorithms,* Obtained from: Spectral Algorithms — Spectral Python 0.21 documentation (13.12.2020)

Internal

Burud, I. (20.09.2020). *Edges and contours* – Lecture notes

Burud, I. (15.09.2020). *Histograms and image statistics* – Lecture notes

Burud, I. (27.10.2020). *Multivariate analysis* – Lecture notes

Burud, I. (27.10.2020). *Texture analysis* – Lecture notes

Kuras, A.K. (19.11.2020). *Hyperspectral Imaging* – Lecture notes

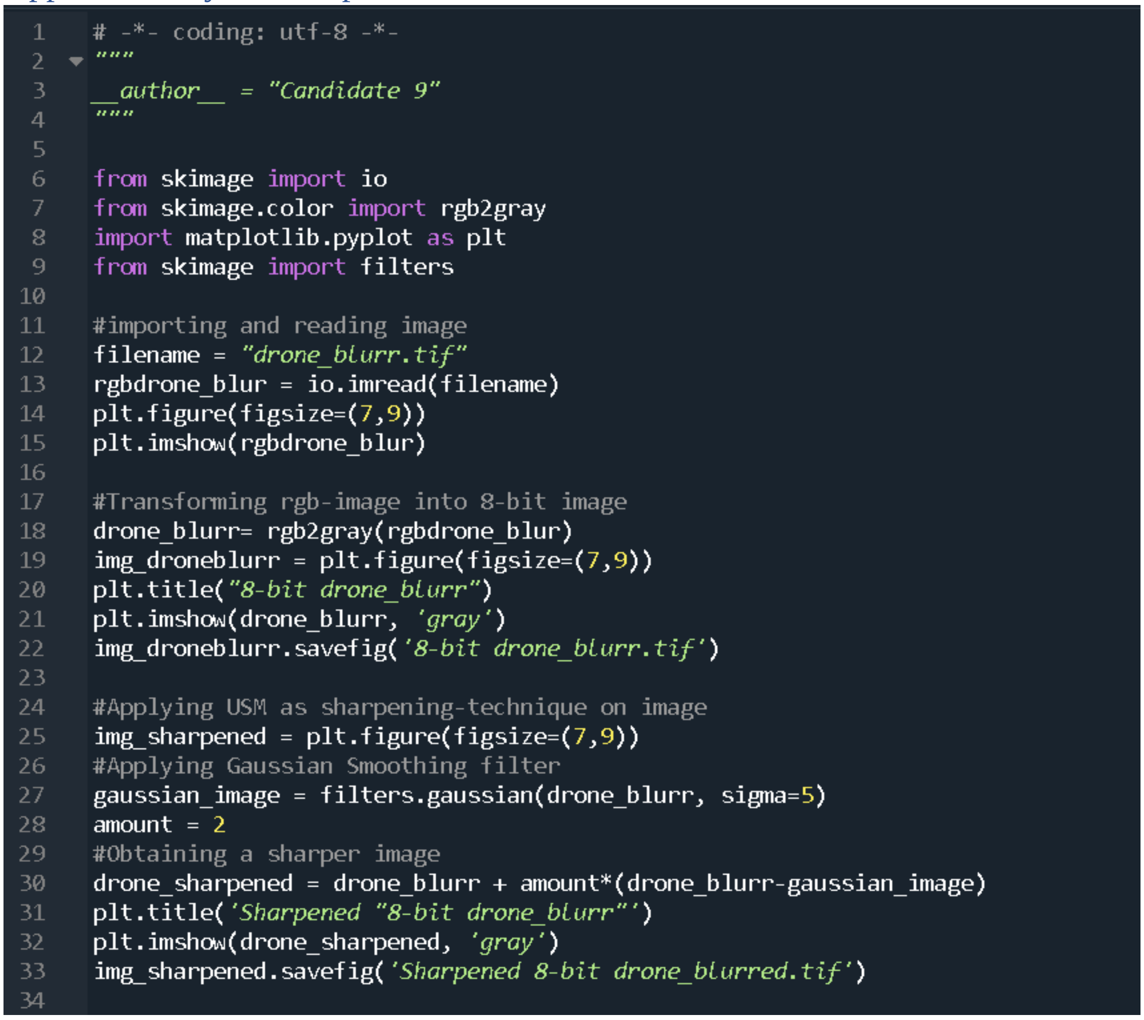
Python scripts

Burud, I. (10.11.2020) *hyperspesdemo* – Python Script

Burud, I. (10.11.2020) *scripts_lecture1509* – Python Script

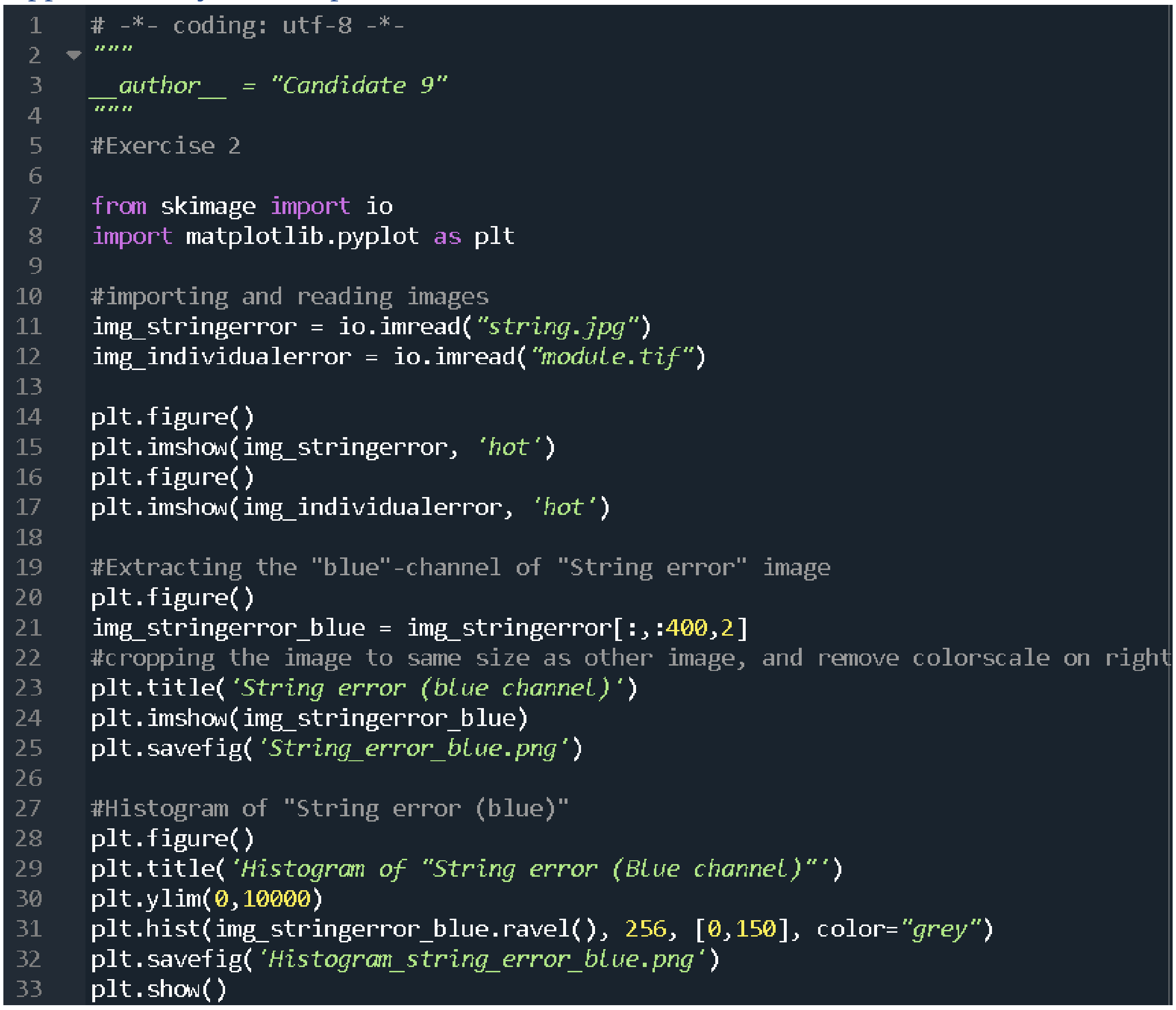
Appendix Python Scripts

Appendix 1: Python script Exercise 1

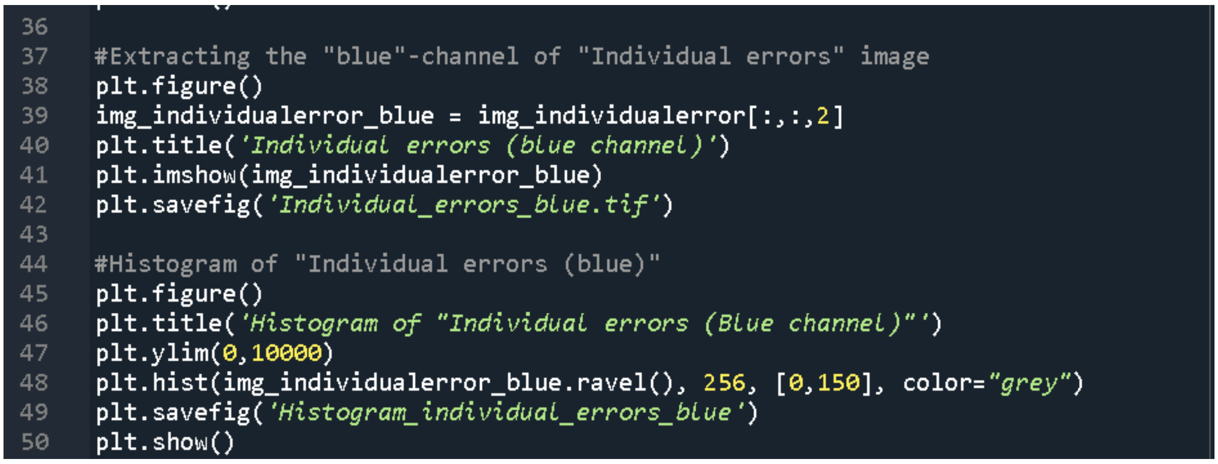


```python
# -*- coding: utf-8 -*-
"""
__author__ = "Candidate 9"
"""

from skimage import io
from skimage.color import rgb2gray
import matplotlib.pyplot as plt
from skimage import filters

#importing and reading image
filename = "drone_blurr.tif"
rgbdrone_blur = io.imread(filename)
plt.figure(figsize=(7,9))
plt.imshow(rgbdrone_blur)

#Transforming rgb-image into 8-bit image
drone_blurr= rgb2gray(rgbdrone_blur)
img_droneblurr = plt.figure(figsize=(7,9))
plt.title("8-bit drone_blurr")
plt.imshow(drone_blurr, 'gray')
img_droneblurr.savefig('8-bit drone_blurr.tif')

#Applying USM as sharpening-technique on image
img_sharpened = plt.figure(figsize=(7,9))
#Applying Gaussian Smoothing filter
gaussian_image = filters.gaussian(drone_blurr, sigma=5)
amount = 2
#Obtaining a sharper image
drone_sharpened = drone_blurr + amount*(drone_blurr-gaussian_image)
plt.title('Sharpened "8-bit drone_blurr"')
plt.imshow(drone_sharpened, 'gray')
img_sharpened.savefig('Sharpened 8-bit drone_blurred.tif')

```

Appendix 2: Python script Exercise 2

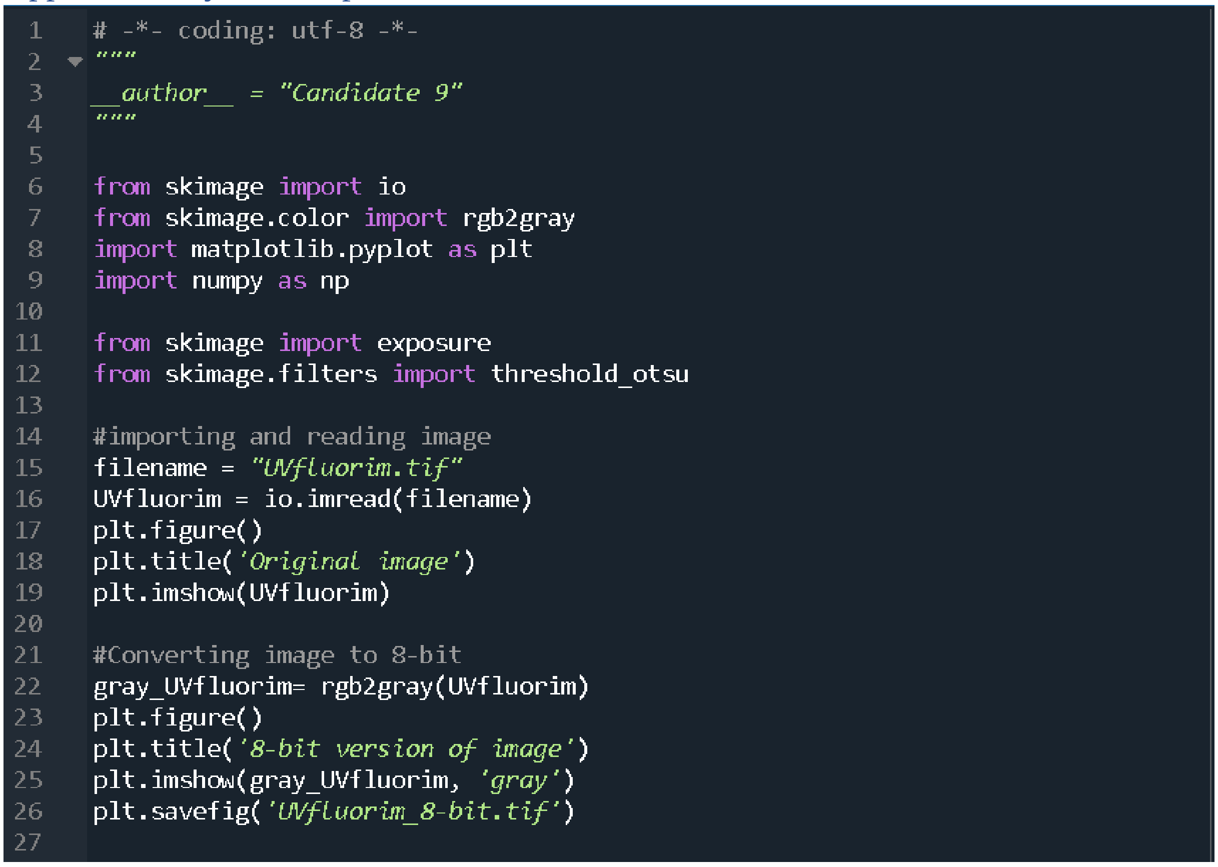


```python
# -*- coding: utf-8 -*-
"""
__author__ = "Candidate 9"
"""

#Exercise 2

from skimage import io
import matplotlib.pyplot as plt

#importing and reading images
img_stringerror = io.imread("string.jpg")
img_individualerror = io.imread("module.tif")

plt.figure()
plt.imshow(img_stringerror, 'hot')
plt.figure()
plt.imshow(img_individualerror, 'hot')

#Extracting the "blue"-channel of "String error" image
plt.figure()
img_stringerror_blue = img_stringerror[:,:400,2]
#cropping the image to same size as other image, and remove colorscale on right
plt.title('String error (blue channel)')
plt.imshow(img_stringerror_blue)
plt.savefig('String_error_blue.png')

#Histogram of "String error (blue)"
plt.figure()
plt.title('Histogram of "String error (Blue channel)"')
plt.ylim(0,10000)
plt.hist(img_stringerror_blue.ravel(), 256, [0,150], color="grey")
plt.savefig('Histogram_string_error_blue.png')
plt.show()
```

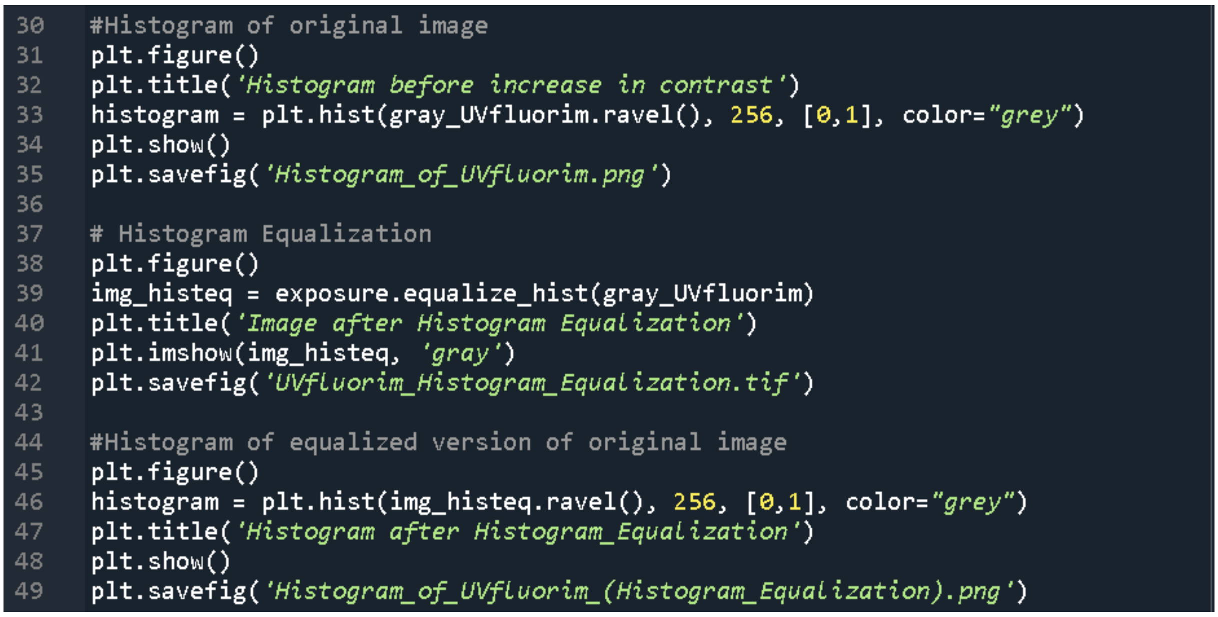


```python

#Extracting the "blue"-channel of "Individual errors" image
plt.figure()
img_individualerror_blue = img_individualerror[:,:,2]
plt.title('Individual errors (blue channel)')
plt.imshow(img_individualerror_blue)
plt.savefig('Individual_errors_blue.tif')

#Histogram of "Individual errors (blue)"
plt.figure()
plt.title('Histogram of "Individual errors (Blue channel)"')
plt.ylim(0,10000)
plt.hist(img_individualerror_blue.ravel(), 256, [0,150], color="grey")
plt.savefig('Histogram_individual_errors_blue')
plt.show()
```

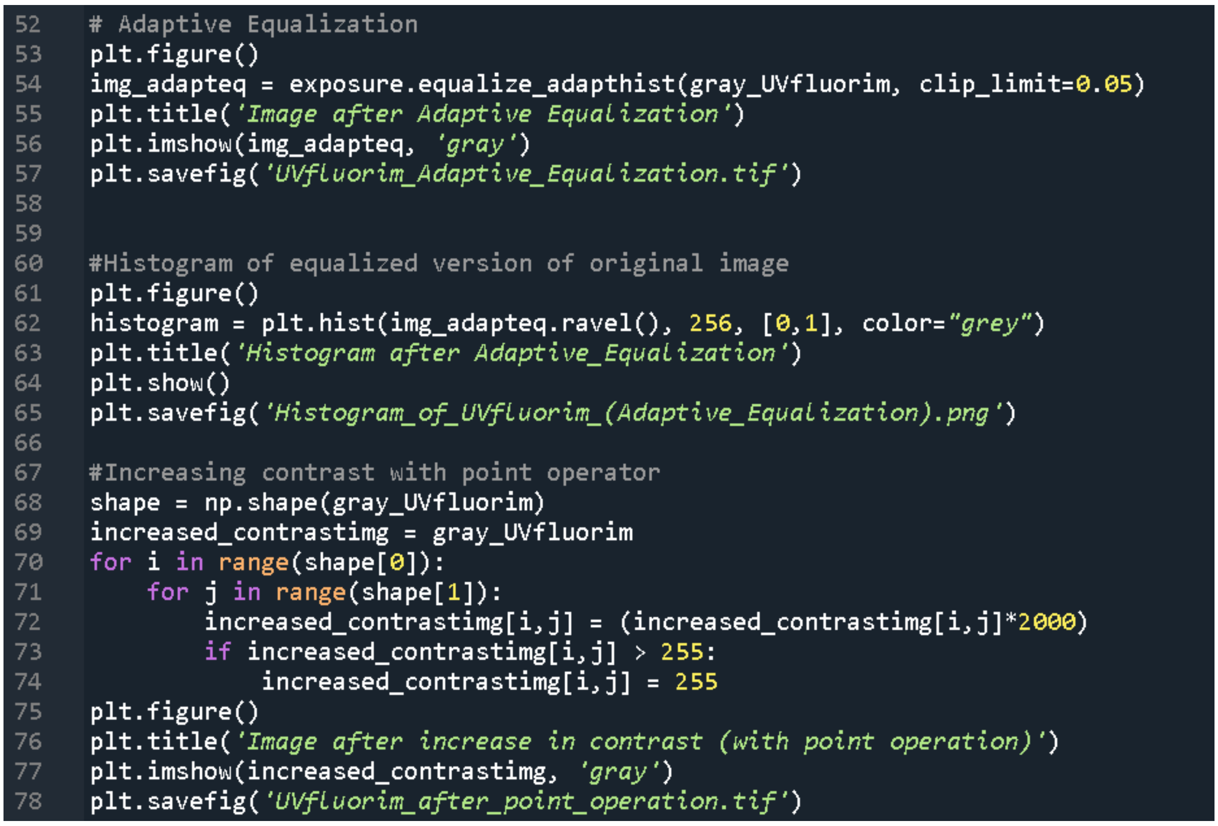
Appendix 3: Python script Exercise 3



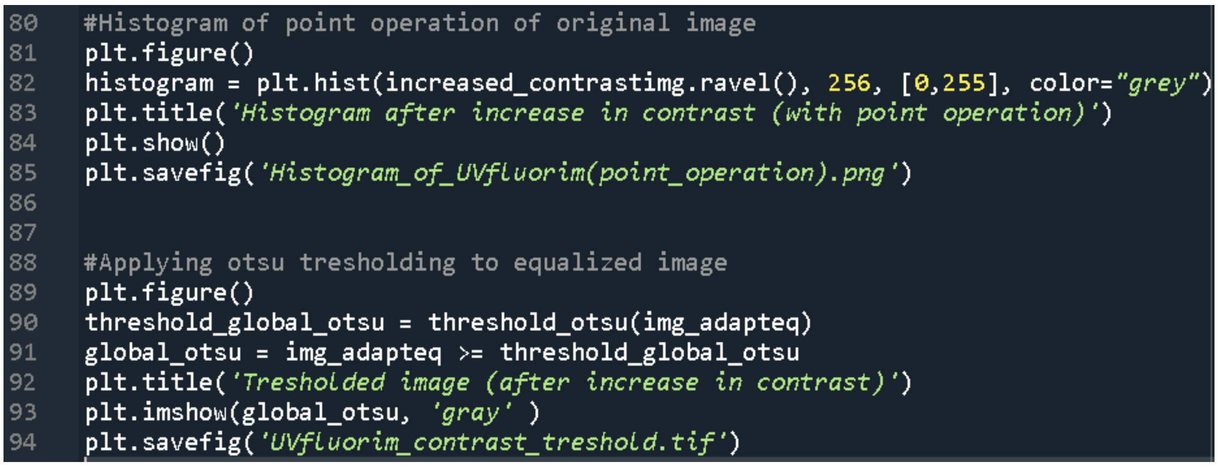
```python
# -*- coding: utf-8 -*-
"""
__author__ = "Candidate 9"
"""

from skimage import io
from skimage.color import rgb2gray
import matplotlib.pyplot as plt
import numpy as np

from skimage import exposure
from skimage.filters import threshold_otsu

#importing and reading image
filename = "UVfluorim.tif"
UVfluorim = io.imread(filename)
plt.figure()
plt.title('Original image')
plt.imshow(UVfluorim)

#Converting image to 8-bit
gray_UVfluorim= rgb2gray(UVfluorim)
plt.figure()
plt.title('8-bit version of image')
plt.imshow(gray_UVfluorim, 'gray')
plt.savefig('UVfluorim_8-bit.tif')

```



```python
#Histogram of original image
plt.figure()
plt.title('Histogram before increase in contrast')
histogram = plt.hist(gray_UVfluorim.ravel(), 256, [0,1], color="grey")
plt.show()
plt.savefig('Histogram_of_UVfluorim.png')

# Histogram Equalization
plt.figure()
img_histeq = exposure.equalize_hist(gray_UVfluorim)
plt.title('Image after Histogram Equalization')
plt.imshow(img_histeq, 'gray')
plt.savefig('UVfluorim_Histogram_Equalization.tif')

#Histogram of equalized version of original image
plt.figure()
histogram = plt.hist(img_histeq.ravel(), 256, [0,1], color="grey")
plt.title('Histogram after Histogram_Equalization')
plt.show()
plt.savefig('Histogram_of_UVfluorim_(Histogram_Equalization).png')
```

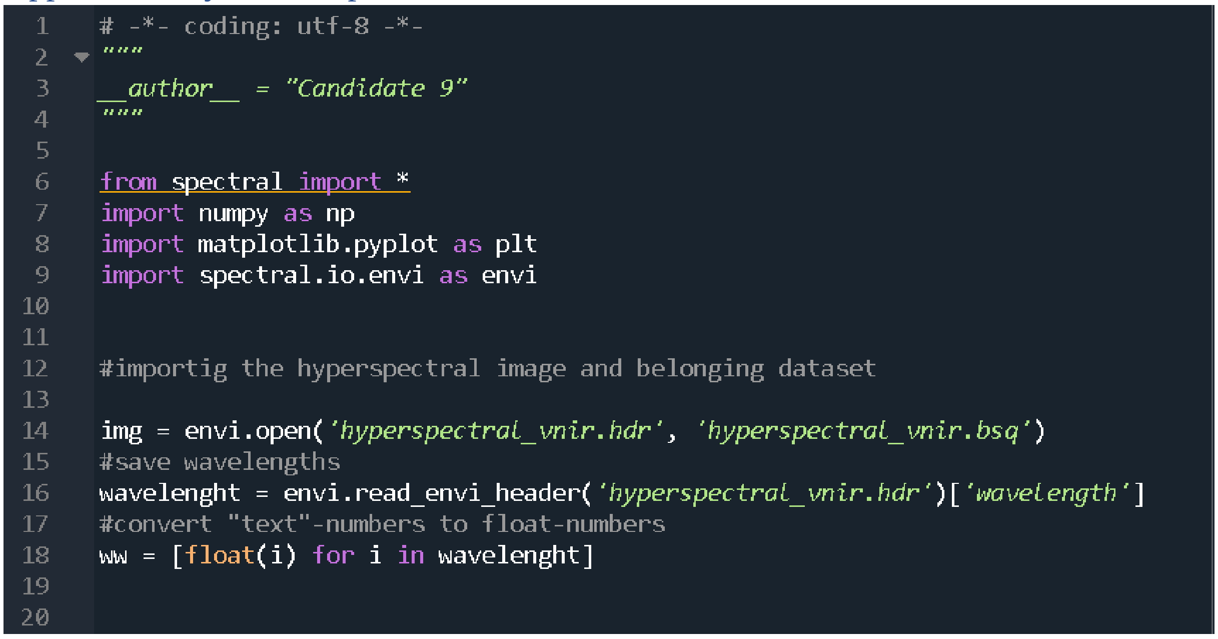


```python
# Adaptive Equalization
plt.figure()
img_adapteq = exposure.equalize_adapthist(gray_UVfluorim, clip_limit=0.05)
plt.title('Image after Adaptive Equalization')
plt.imshow(img_adapteq, 'gray')
plt.savefig('UVfluorim_Adaptive_Equalization.tif')


#Histogram of equalized version of original image
plt.figure()
histogram = plt.hist(img_adapteq.ravel(), 256, [0,1], color="grey")
plt.title('Histogram after Adaptive_Equalization')
plt.show()
plt.savefig('Histogram_of_UVfluorim_(Adaptive_Equalization).png')

#Increasing contrast with point operator
shape = np.shape(gray_UVfluorim)
increased_contrastimg = gray_UVfluorim
for i in range(shape[0]):
    for j in range(shape[1]):
        increased_contrastimg[i,j] = (increased_contrastimg[i,j]*2000)
        if increased_contrastimg[i,j] > 255:
            increased_contrastimg[i,j] = 255
plt.figure()
plt.title('Image after increase in contrast (with point operation)')
plt.imshow(increased_contrastimg, 'gray')
plt.savefig('UVfluorim_after_point_operation.tif')
```

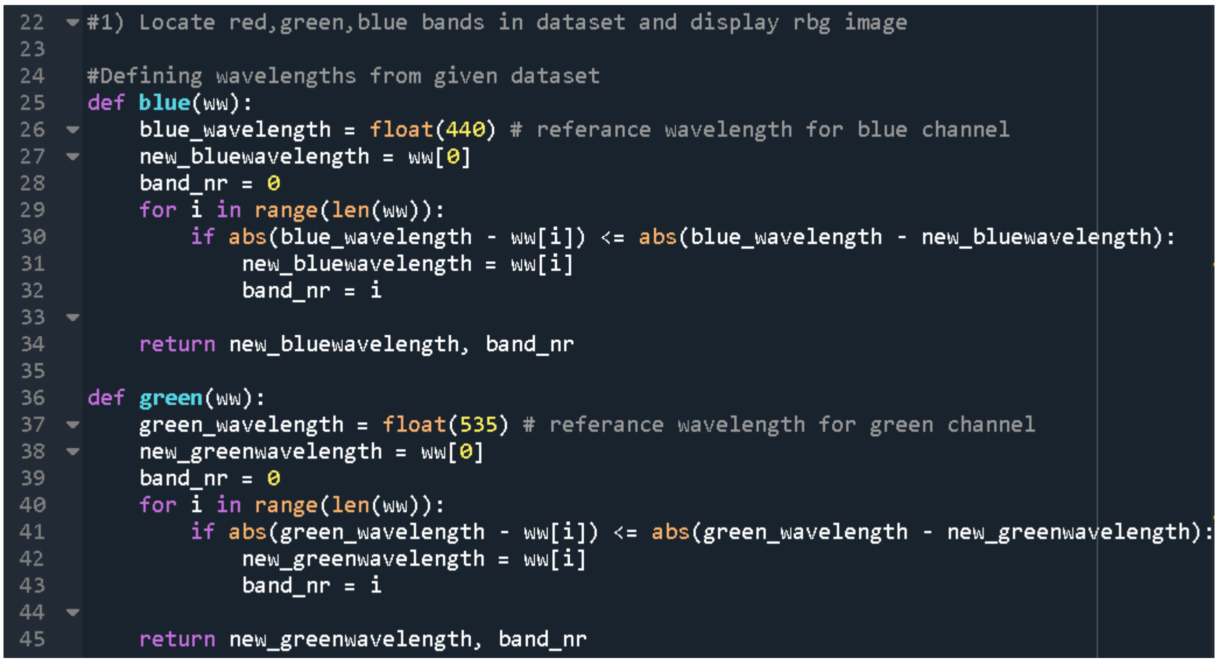


```python
#Histogram of point operation of original image
plt.figure()
histogram = plt.hist(increased_contrastimg.ravel(), 256, [0,255], color="grey")
plt.title('Histogram after increase in contrast (with point operation)')
plt.show()
plt.savefig('Histogram_of_UVfluorim(point_operation).png')


#Applying otsu tresholding to equalized image
plt.figure()
threshold_global_otsu = threshold_otsu(img_adapteq)
global_otsu = img_adapteq >= threshold_global_otsu
plt.title('Tresholded image (after increase in contrast)')
plt.imshow(global_otsu, 'gray' )
plt.savefig('UVfluorim_contrast_treshold.tif')
```

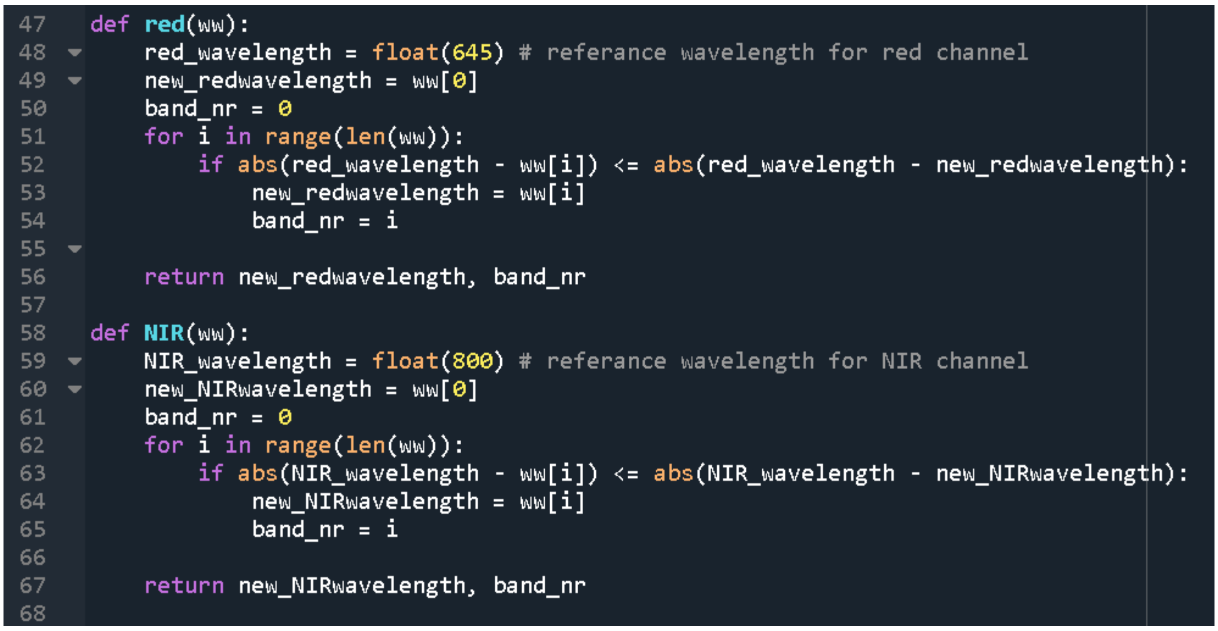
Appendix 4: Python Script Exercise 4



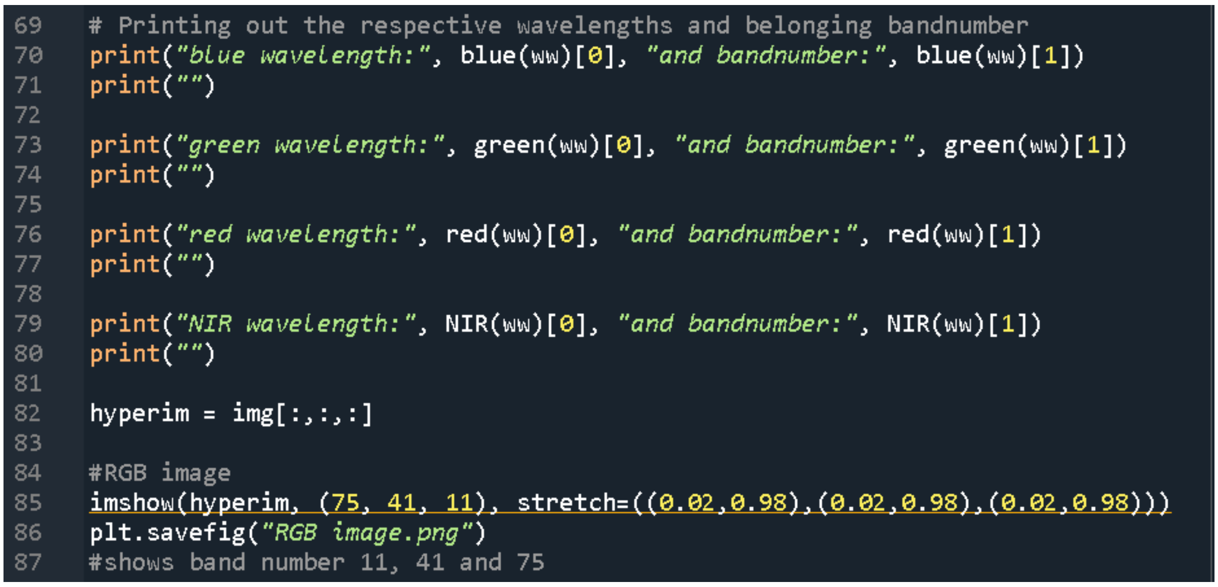
```python
# -*- coding: utf-8 -*-
"""
__author__ = "Candidate 9"
"""

from spectral import *
import numpy as np
import matplotlib.pyplot as plt
import spectral.io.envi as envi


#importig the hyperspectral image and belonging dataset

img = envi.open('hyperspectral_vnir.hdr', 'hyperspectral_vnir.bsq')
#save wavelengths
wavelenght = envi.read_envi_header('hyperspectral_vnir.hdr')['wavelength']
#convert "text"-numbers to float-numbers
ww = [float(i) for i in wavelenght]


```



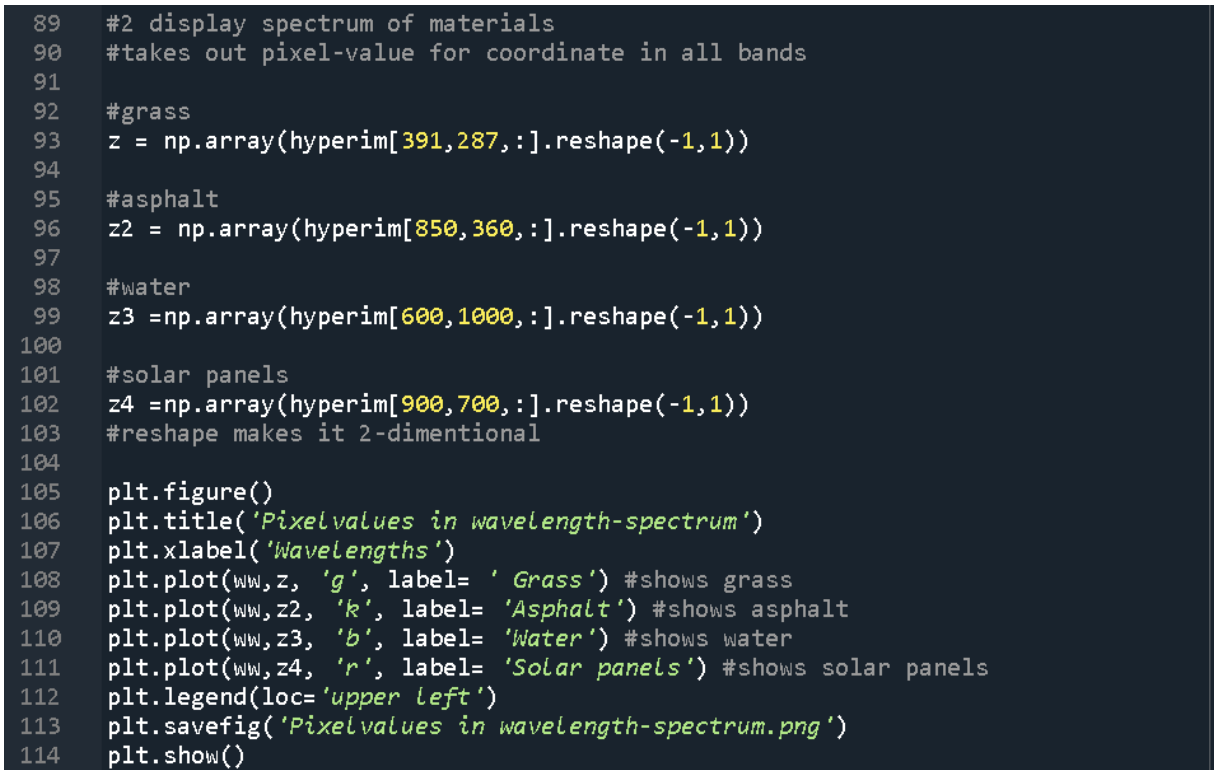
```python
#1) Locate red,green,blue bands in dataset and display rbg image

#Defining wavelengths from given dataset
def blue(ww):
    blue_wavelength = float(440) # referance wavelength for blue channel
    new_bluewavelength = ww[0]
    band_nr = 0
    for i in range(len(ww)):
        if abs(blue_wavelength - ww[i]) <= abs(blue_wavelength - new_bluewavelength):
            new_bluewavelength = ww[i]
            band_nr = i

    return new_bluewavelength, band_nr

def green(ww):
    green_wavelength = float(535) # referance wavelength for green channel
    new_greenwavelength = ww[0]
    band_nr = 0
    for i in range(len(ww)):
        if abs(green_wavelength - ww[i]) <= abs(green_wavelength - new_greenwavelength):
            new_greenwavelength = ww[i]
            band_nr = i

    return new_greenwavelength, band_nr
```



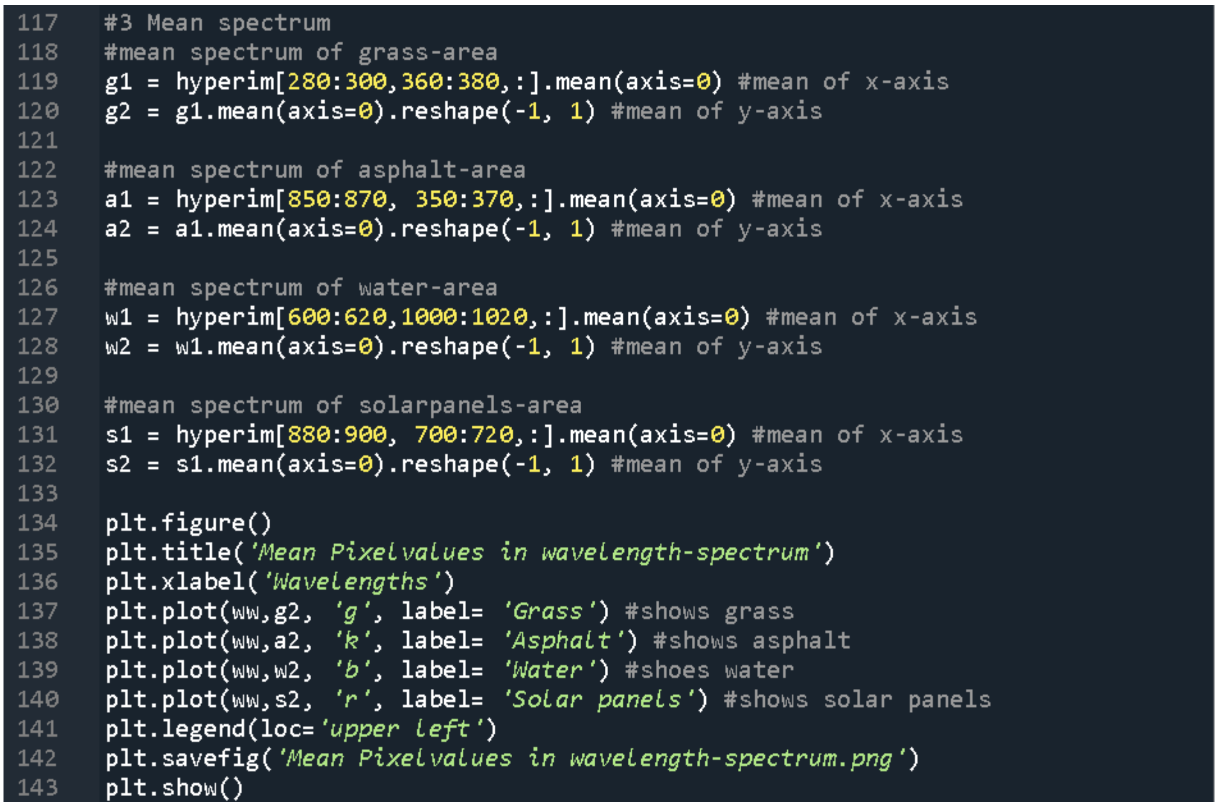
```python
def red(ww):
    red_wavelength = float(645) # referance wavelength for red channel
    new_redwavelength = ww[0]
    band_nr = 0
    for i in range(len(ww)):
        if abs(red_wavelength - ww[i]) <= abs(red_wavelength - new_redwavelength):
            new_redwavelength = ww[i]
            band_nr = i

    return new_redwavelength, band_nr

def NIR(ww):
    NIR_wavelength = float(800) # referance wavelength for NIR channel
    new_NIRwavelength = ww[0]
    band_nr = 0
    for i in range(len(ww)):
        if abs(NIR_wavelength - ww[i]) <= abs(NIR_wavelength - new_NIRwavelength):
            new_NIRwavelength = ww[i]
            band_nr = i

    return new_NIRwavelength, band_nr

```



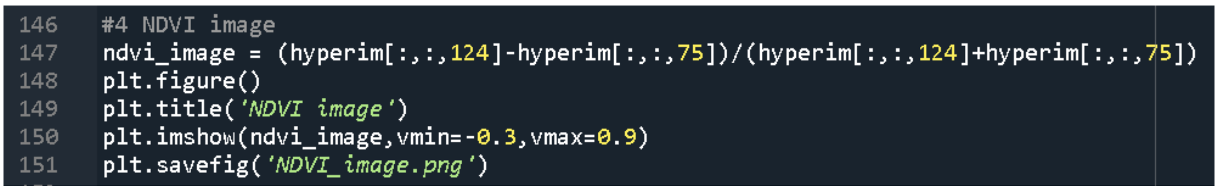
```python
# Printing out the respective wavelengths and belonging bandnumber
print("blue wavelength:", blue(ww)[0], "and bandnumber:", blue(ww)[1])
print("")

print("green wavelength:", green(ww)[0], "and bandnumber:", green(ww)[1])
print("")

print("red wavelength:", red(ww)[0], "and bandnumber:", red(ww)[1])
print("")

print("NIR wavelength:", NIR(ww)[0], "and bandnumber:", NIR(ww)[1])
print("")

hyperim = img[:,:,:]

#RGB image
imshow(hyperim, (75, 41, 11), stretch=((0.02,0.98),(0.02,0.98),(0.02,0.98)))
plt.savefig("RGB image.png")
#shows band number 11, 41 and 75
```



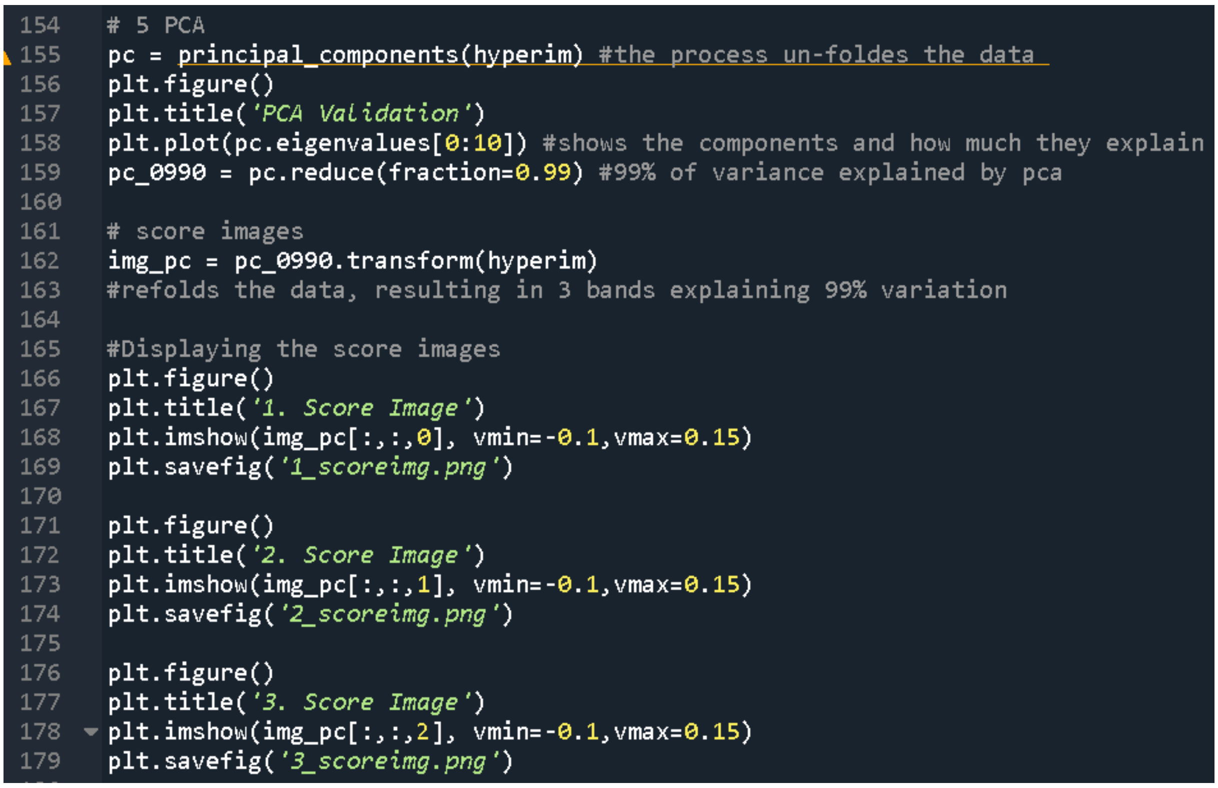
```python
#2 display spectrum of materials
#takes out pixel-value for coordinate in all bands

#grass
z = np.array(hyperim[391,287,:].reshape(-1,1))

#asphalt
z2 = np.array(hyperim[850,360,:].reshape(-1,1))

#water
z3 =np.array(hyperim[600,1000,:].reshape(-1,1))

#solar panels
z4 =np.array(hyperim[900,700,:].reshape(-1,1))
#reshape makes it 2-dimentional

plt.figure()
plt.title('Pixelvalues in wavelength-spectrum')
plt.xlabel('Wavelengths')
plt.plot(ww,z, 'g', label= ' Grass') #shows grass
plt.plot(ww,z2, 'k', label= 'Asphalt') #shows asphalt
plt.plot(ww,z3, 'b', label= 'Water') #shows water
plt.plot(ww,z4, 'r', label= 'Solar panels') #shows solar panels
plt.legend(loc='upper left')
plt.savefig('Pixelvalues in wavelength-spectrum.png')
plt.show()
```



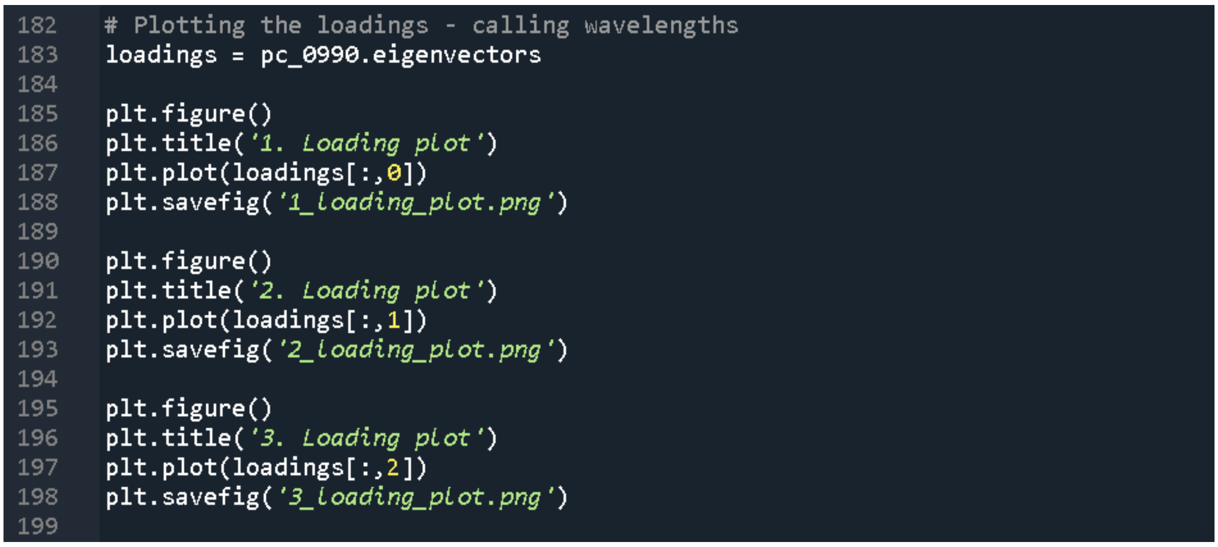
```python
#3 Mean spectrum
#mean spectrum of grass-area
g1 = hyperim[280:300,360:380,:].mean(axis=0) #mean of x-axis
g2 = g1.mean(axis=0).reshape(-1, 1) #mean of y-axis

#mean spectrum of asphalt-area
a1 = hyperim[850:870, 350:370,:].mean(axis=0) #mean of x-axis
a2 = a1.mean(axis=0).reshape(-1, 1) #mean of y-axis

#mean spectrum of water-area
w1 = hyperim[600:620,1000:1020,:].mean(axis=0) #mean of x-axis
w2 = w1.mean(axis=0).reshape(-1, 1) #mean of y-axis

#mean spectrum of solarpanels-area
s1 = hyperim[880:900, 700:720,:].mean(axis=0) #mean of x-axis
s2 = s1.mean(axis=0).reshape(-1, 1) #mean of y-axis

plt.figure()
plt.title('Mean Pixelvalues in wavelength-spectrum')
plt.xlabel('Wavelengths')
plt.plot(ww,g2, 'g', label= 'Grass') #shows grass
plt.plot(ww,a2, 'k', label= 'Asphalt') #shows asphalt
plt.plot(ww,w2, 'b', label= 'Water') #shoes water
plt.plot(ww,s2, 'r', label= 'Solar panels') #shows solar panels
plt.legend(loc='upper left')
plt.savefig('Mean Pixelvalues in wavelength-spectrum.png')
plt.show()
```



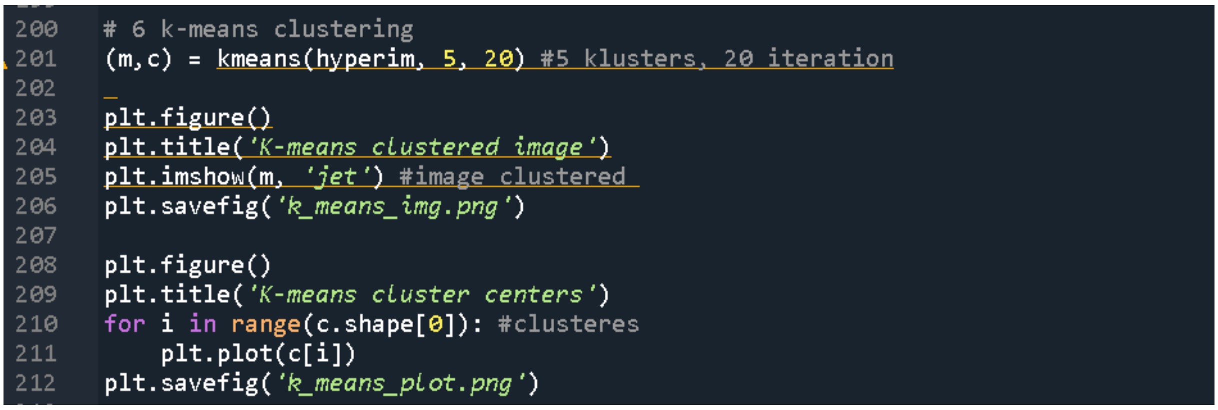
```python
#4 NDVI image
ndvi_image = (hyperim[:,:,124]-hyperim[:,:,75])/(hyperim[:,:,124]+hyperim[:,:,75])
plt.figure()
plt.title('NDVI image')
plt.imshow(ndvi_image,vmin=-0.3,vmax=0.9)
plt.savefig('NDVI_image.png')
```



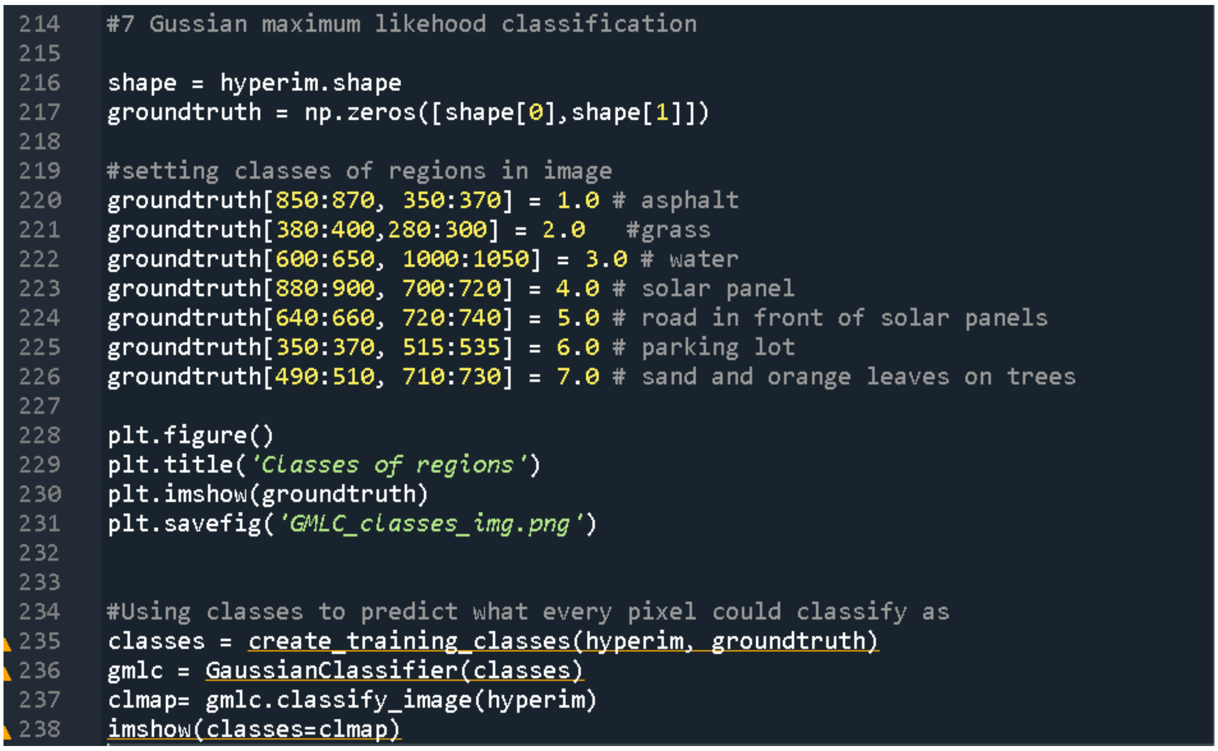
```python
# 5 PCA
pc = principal_components(hyperim) #the process un-foldes the data
plt.figure()
plt.title('PCA Validation')
plt.plot(pc.eigenvalues[0:10]) #shows the components and how much they explain
pc_0990 = pc.reduce(fraction=0.99) #99% of variance explained by pca

# score images
img_pc = pc_0990.transform(hyperim)
#refolds the data, resulting in 3 bands explaining 99% variation

#Displaying the score images
plt.figure()
plt.title('1. Score Image')
plt.imshow(img_pc[:,:,0], vmin=-0.1,vmax=0.15)
plt.savefig('1_scoreimg.png')

plt.figure()
plt.title('2. Score Image')
plt.imshow(img_pc[:,:,1], vmin=-0.1,vmax=0.15)
plt.savefig('2_scoreimg.png')

plt.figure()
plt.title('3. Score Image')
plt.imshow(img_pc[:,:,2], vmin=-0.1,vmax=0.15)
plt.savefig('3_scoreimg.png')
```



```python
# Plotting the loadings - calling wavelengths
loadings = pc_0990.eigenvectors

plt.figure()
plt.title('1. Loading plot')
plt.plot(loadings[:,0])
plt.savefig('1_loading_plot.png')

plt.figure()
plt.title('2. Loading plot')
plt.plot(loadings[:,1])
plt.savefig('2_loading_plot.png')

plt.figure()
plt.title('3. Loading plot')
plt.plot(loadings[:,2])
plt.savefig('3_loading_plot.png')

```



```python
# 6 k-means clustering
(m,c) = kmeans(hyperim, 5, 20) #5 klusters, 20 iteration

plt.figure()
plt.title('K-means clustered image')
plt.imshow(m, 'jet') #image clustered
plt.savefig('k_means_img.png')

plt.figure()
plt.title('K-means cluster centers')
for i in range(c.shape[0]): #clusteres
    plt.plot(c[i])
plt.savefig('k_means_plot.png')
```



```
#7 Gussian maximum likehood classification

shape = hyperim.shape
groundtruth = np.zeros([shape[0],shape[1]])

#setting classes of regions in image
groundtruth[850:870, 350:370] = 1.0 # asphalt
groundtruth[380:400,280:300] = 2.0   #grass
groundtruth[600:650, 1000:1050] = 3.0 # water
groundtruth[880:900, 700:720] = 4.0 # solar panel
groundtruth[640:660, 720:740] = 5.0 # road in front of solar panels
groundtruth[350:370, 515:535] = 6.0 # parking lot
groundtruth[490:510, 710:730] = 7.0 # sand and orange leaves on trees

plt.figure()
plt.title('Classes of regions')
plt.imshow(groundtruth)
plt.savefig('GMLC_classes_img.png')


#Using classes to predict what every pixel could classify as
classes = create_training_classes(hyperim, groundtruth)
gmlc = GaussianClassifier(classes)
clmap= gmlc.classify_image(hyperim)
imshow(classes=clmap)
```