Eksam2022

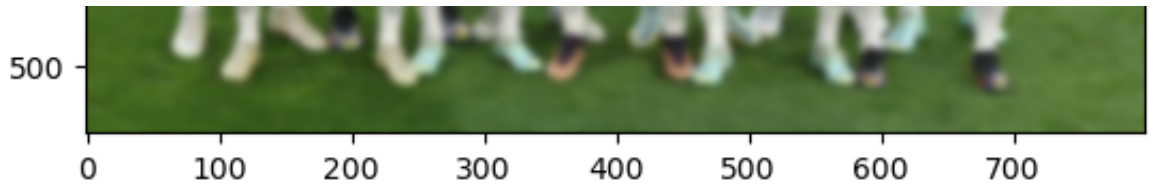
Student number: 107740

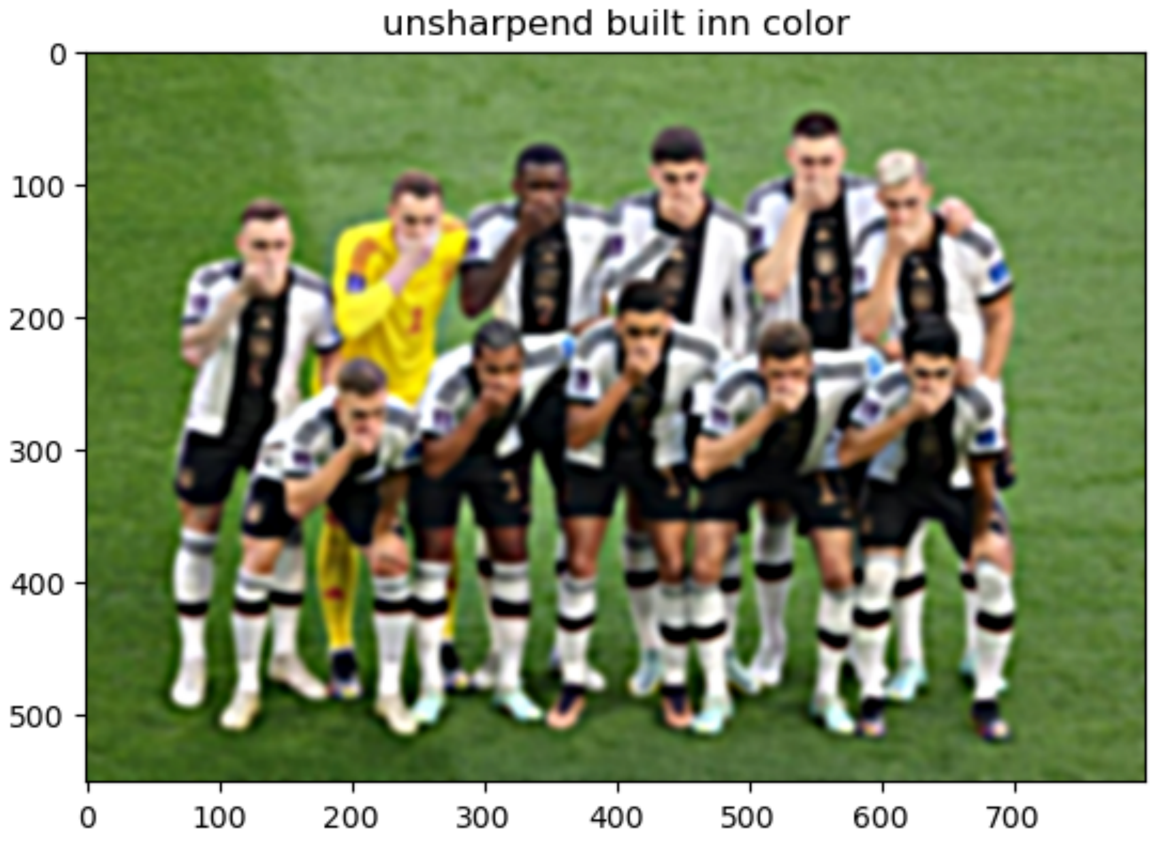
Exercise 1

a)

In [1]:
```
#Import librarys
import numpy as np
import skimage
import matplotlib.pyplot as plt

#Read image
soccerteam_image_color = skimage.io.imread('soccerteam.jpeg')

#Unsharpend mask with build inn function.
soccerteam_unsharpened_built_inn = skimage.filters.unsharp_mask(soc

#Plot results
plt.title("Original image color")
plt.imshow(soccerteam_image_color)
plt.show()

plt.imshow(soccerteam_unsharpened_built_inn)
plt.title("unsharpend built inn color")
plt.show()
```

```
/Users/Bard/anaconda3/lib/python3.8/site-packages/skimage/_shared/
utils.py:348: RuntimeWarning: Images with dimensions (M, N, 3) are
interpreted as 2D+RGB by default. Use `multichannel=False` to inte
rpret as 3D image with last dimension of length 3.
  return func(*args, **kwargs)
```





Tried with both manual and built-in unsharpened mask as well as Laplace sharpen method. The built-in skimage unsharpened mask gave the best results.

The point of the unsharpened mask is to take the image, unsharpen it, and subtract the original image from this unsharp image. This will yield a sharper image.

The point of Laplace sharpening is to get the edges and add these to the image, this will create the effect of a sharper image.

b)

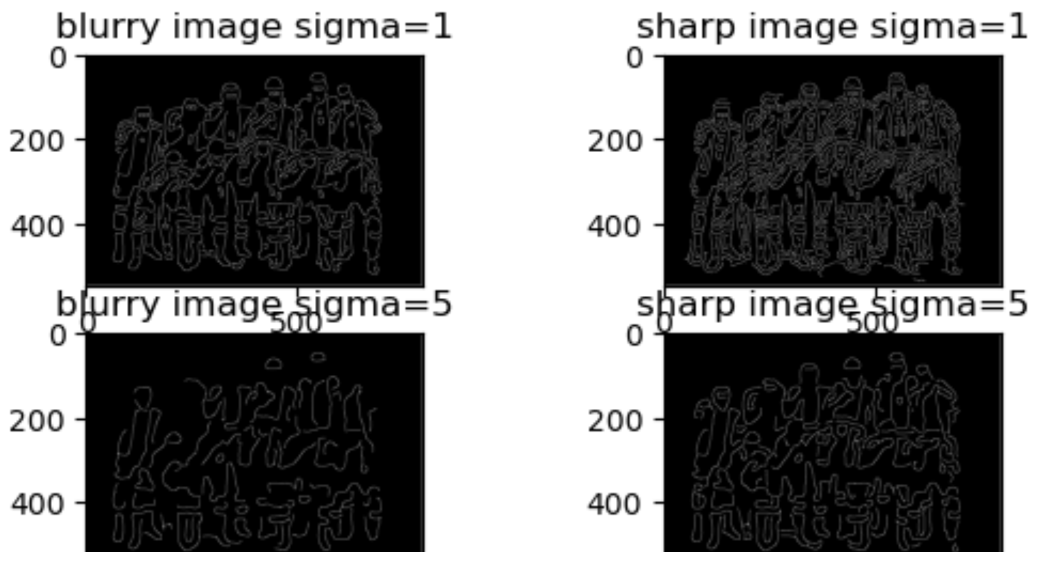
Canny filter is a type of multi-stage edge detector. That can detect a wide range of edges in images. Some of the stages include:

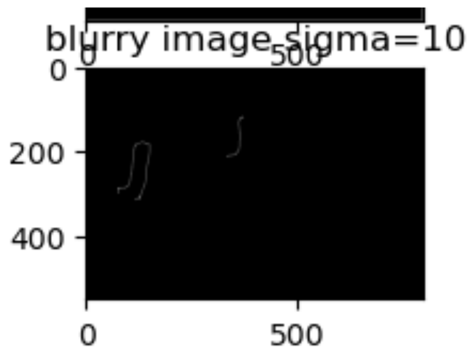
- Gaussian filter, to remove noise. This makes edges more accurate.
- Sobel filter, which is used to find edge gradient and direction for each pixel
- Non-maximum Suppression, used to remove unwanted pixels which may not be part of an edge.
- Hysteresis Thresholding, is used to double-check that an edge is an edge. Create two threshold values, a min and a max value. Edges that have an intensity gradient of more than the max value are edges and if any edges are below the min value we remove them because we know that they are not real edges.

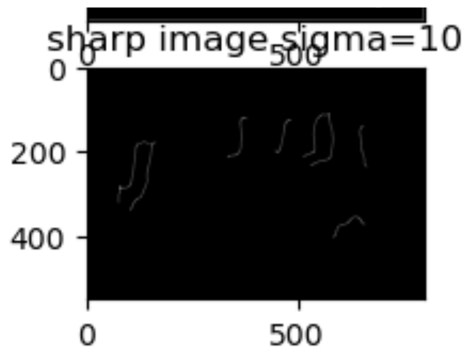
The image on the left, with the fewest edges, is the original one.

In [2]:

```python
#Read image
soccerteam_image = skimage.color.rgb2gray(soccerteam_image_color)
soccerteam_unsharpened_built_inn = skimage.color.rgb2gray(soccertea

#Blurred image with different sigma values
blurry_image_1 = skimage.feature.canny(soccerteam_image, sigma=1)
blurry_image_5 = skimage.feature.canny(soccerteam_image, sigma=5)
blurry_image_10 = skimage.feature.canny(soccerteam_image, sigma=10)

#Sharpend image with different sigma values
sharp_image_1 = skimage.feature.canny(soccerteam_unsharpened_built_
sharp_image_5 = skimage.feature.canny(soccerteam_unsharpened_built_
sharp_image_10 = skimage.feature.canny(soccerteam_unsharpened_built

#Plot results
fig = plt.figure()
fig.add_subplot(3, 2, 1)
plt.imshow(blurry_image_1, cmap='gray')
plt.title('blurry image sigma=1')

fig.add_subplot(3, 2, 3)
plt.imshow(blurry_image_5, cmap='gray')
plt.title('blurry image sigma=5')

fig.add_subplot(3, 2, 5)
plt.imshow(blurry_image_10, cmap='gray')
plt.title('blurry image sigma=10')

fig.add_subplot(3, 2, 2)
plt.imshow(sharp_image_1, cmap='gray')
plt.title('sharp image sigma=1')

fig.add_subplot(3, 2, 4)
plt.imshow(sharp_image_5, cmap='gray')
plt.title('sharp image sigma=5')

fig.add_subplot(3, 2, 6)
plt.imshow(sharp_image_10, cmap='gray')
plt.title('sharp image sigma=10')

plt.show()
```







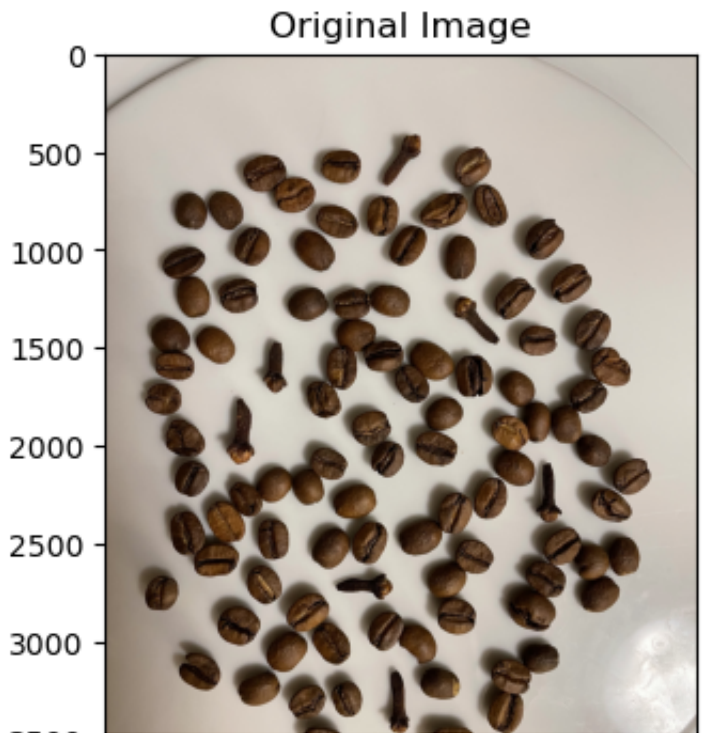
Since sigma in Canny filter is the standard deviation of the Gaussian filter, the higher the sigma the more the picture will be smoothened. Therefore we lose a lot of the edges when the sigma is higher.

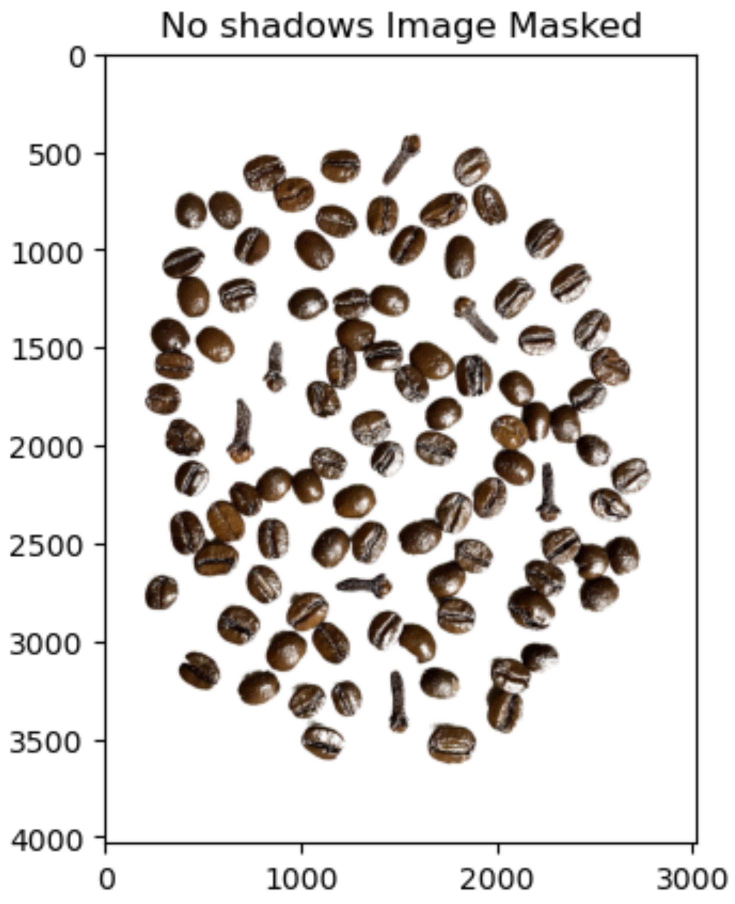
Exercise 2

a)

In [3]:

```
#Import librarys
import skimage
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
import scipy

#Get image
coffeebeans_image = skimage.io.imread('coffeebeans.jpg')
plt.imshow(coffeebeans_image)
plt.title('Original Image')
plt.show()


#Brown has a red ratio much higher then shadows/gray therefor i am t
def remove_based_on_red(image, fraction=0.45):
    shape = np.shape(image)
    temp_image = image
    for i in range(shape[0]):
        for j in range(shape[1]):
            #Need this because we cant devid by zero.
            if (int(image[i][j][0])+int(image[i][j][1])+int(image[i]
                continue
                #Here we take the red value devided by all the value
            elif (int(image[i][j][0]) / (int(image[i][j][0])+int(ima
                temp_image[i][j][0] = 255
                temp_image[i][j][1] = 255
                temp_image[i][j][2] = 255
    return temp_image


no_background_coffee = remove_based_on_red(coffeebeans_image)

plt.imshow(no_background_coffee)
plt.title('No shadows Image Masked')
plt.show()
```





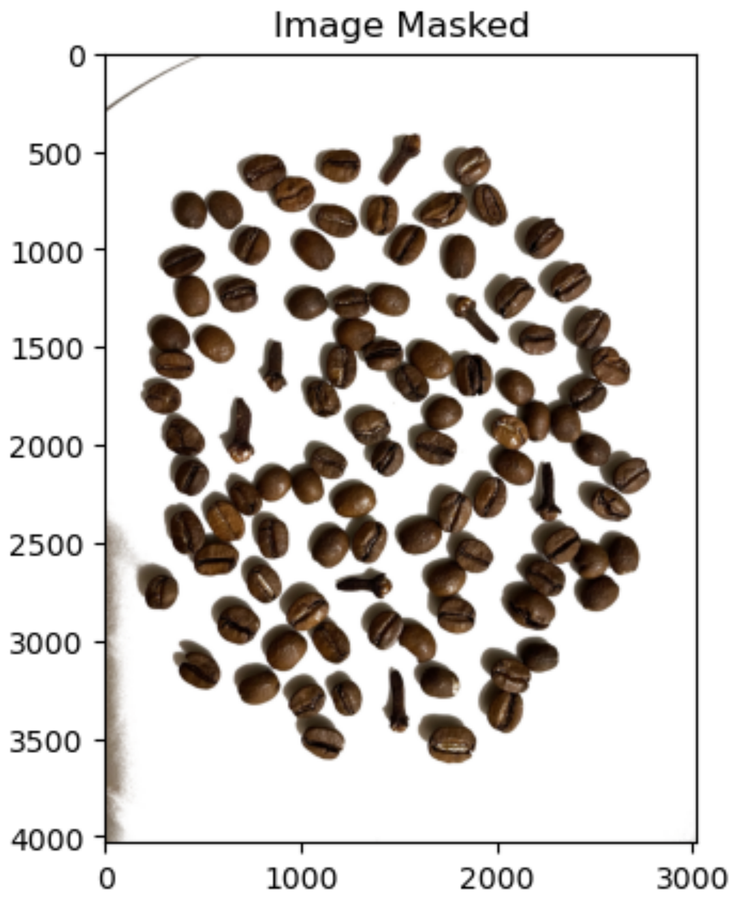
Since both coffee beans and shadows are dark it's hard to remove them by thresholding. Therefore I thresholded with the fraction of red in each pixel. Brown(coffee beans and cloves) have a much higher fraction of red than grey (shadow). This way I get to remove everything that isn't red enough.

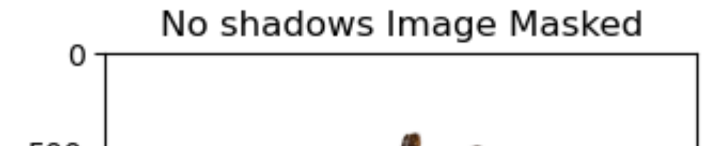
b)

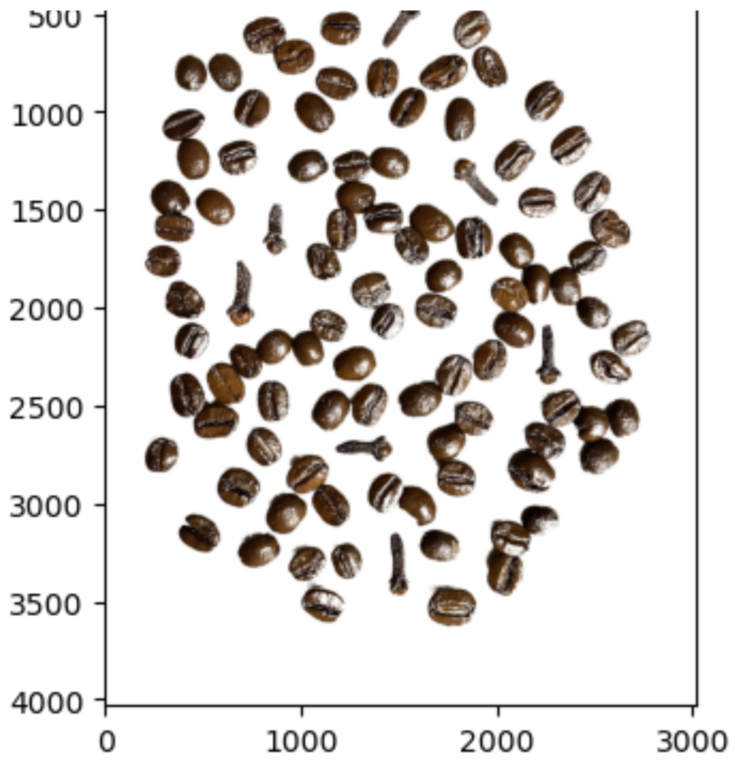
In [4]:

```
```

```
#Function to thershold and mask image
def masked_image(image):
    shape = np.shape(image)
    temp_image = skimage.color.rgb2gray(image)
    thres_image = skimage.filters.threshold_otsu(temp_image)
    binary_image = temp_image > thres_image
    for i in range(shape[0]):
        for j in range(shape[1]):
            if binary_image[i][j] == 1:
                image[i][j][0] = 255
                image[i][j][1] = 255
                image[i][j][2] = 255
    return image

coffeebeans_image_1 = skimage.io.imread('coffeebeans.jpg')
masked_coffee = masked_image(coffeebeans_image_1)


#Plott results
plt.imshow(masked_coffee)
plt.title('Image Masked')
plt.show()

plt.imshow(no_background_coffee)
plt.title('No shadows Image Masked')
plt.show()
```







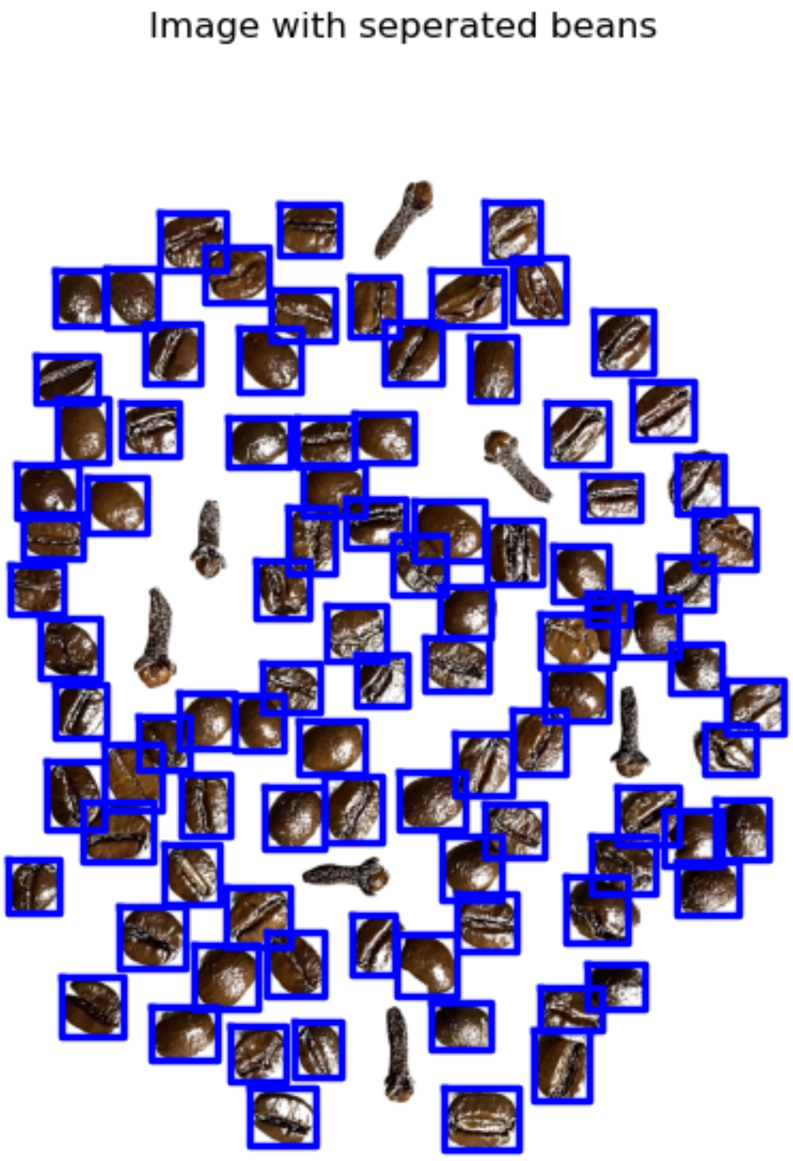
I already have the image without shadows masked. So the only thing I had to do was to threshold the original image with otsu thresholding. This way I get the background removed, but keep the shadows and beans and cloves.

c)

```
In [5]: #Need math to calculate size for later task.
import math

#Threshold agin on the image without shadows
grayscale_image = skimage.color.rgb2gray(no_background_coffee)
thresh = skimage.filters.threshold_yen(grayscale_image)
binary_image = grayscale_image > thresh

#Fill holes if there are any
binary_invert_image = np.invert(binary_image)
img_fill_holes = scipy.ndimage.binary_fill_holes(binary_invert_imag


#Remove noise
image_erode = skimage.morphology.erosion(img_fill_holes, skimage.mo
image_dilate = skimage.morphology.dilation(image_erode, skimage.mor

#Separate Non stop that are close to each other
distance = scipy.ndimage.distance_transform_edt(image_dilate)
local_maxi = skimage.feature.peak_local_max(distance, min_distance=
                                            footprint=np.ones((3, 3
markers = scipy.ndimage.label(local_maxi)[0]
labels = skimage.segmentation.watershed(-distance, markers, mask=im

#Create objects from the Non stop
properties = skimage.measure.regionprops(labels)

#Create and plot squares that separate Non stop, broken Non stop an
```

```python
#create and plot squares that separate Non stop, broken Non stop and
fig, ax = plt.subplots(figsize=(10, 6))
ax.imshow(coffeebeans_image)

handle = {}
number_of_beans= 0
list_of_centers = []
for prop in properties:
    printing = False
    #Formula for circularity and roundnes
    circularity = 4 * np.pi * (prop.area / prop.perimeter ** 2)
    roundnes = 4 * prop.area / (np.pi * prop.axis_major_length**2)
    #Check to see if properti is bean or clove.
    if prop.area > 13000:
        minr, minc, maxr, maxc = prop.bbox
        bx = (minc, maxc, maxc, minc, minc)
        by = (minr, minr, maxr, maxr, minr)
        ax.plot(bx, by, '-b', linewidth=2.5)

        number_of_beans += 1
        y0, x0 = prop.centroid
        area = prop.area
        tuple1 = (y0,x0,area)
        list_of_centers.append(tuple1)

    #Not all beans was in the first criteria
    elif 8000 < prop.area < 10000 and roundnes < 0.23:
        minr, minc, maxr, maxc = prop.bbox
        bx = (minc, maxc, maxc, minc, minc)
        by = (minr, minr, maxr, maxr, minr)
        ax.plot(bx, by, '-b', linewidth=2.5)

        number_of_beans += 1
        y0, x0 = prop.centroid
        area = prop.area
        tuple1 = (y0,x0,area)
        list_of_centers.append(tuple1)

    elif 10000 < prop.area < 11000 and circularity > 0.6:
        minr, minc, maxr, maxc = prop.bbox
        bx = (minc, maxc, maxc, minc, minc)
        by = (minr, minr, maxr, maxr, minr)
        ax.plot(bx, by, '-b', linewidth=2.5)

        number_of_beans += 1
        y0, x0 = prop.centroid
        area = prop.area
        tuple1 = (y0,x0,area)
        list_of_centers.append(tuple1)


#Plott results
ax.set_axis_off()
plt.tight_layout()
plt.title('Image with seperated beans')
plt.show()
print('Number of beans :', number_of_beans, '\n')
```

```
/var/folders/7x/qyhpfc0518313f53n8x1h4m40000gn/T/ipykernel_7654/14
17883950.py:20: FutureWarning: indices argument is deprecated and
will be removed in version 0.20. To avoid this warning, please do
not use the indices argument. Please see peak_local_max documentat
ion for more details.
  local_maxi = skimage.feature.peak_local_max(distance, min_distan
ce=100, indices=False,
```



```
Number of beans : 93
```

Here I use the skimage regionprops function. This mask out all the objects that have a different value than the background. Then I need some criteria to only get the beans, and so that I get every bean. When that is done I plot to see if all beans have been accounted for and count up all the beans. It's 93 beans in total.

d)

At the same time as I count the beans, I also add their x and y coordinates for their centre in a tuple and add that tuple to a list for all the beans.

In [6]:
```
print('List of all beans with center coords and size, (x,y,size) :'
```

```
List of all beans with center coords and size, (x,y,size) : [(580.
5980463891125, 1202.9599034169898, 27334), (583.06421231978, 1869
.264197158684, 26537), (615.943712772998, 813.4159166115155, 30220
), (725.8834925058627, 966.2424633789892, 29423), (802.7641472679459
59, 436.4760397389695, 20534), (777.2003289096264, 1960.0366390025
624, 26147), (805.7282913165266, 1728.9324229691877, 31416), (803.
2035298384172, 616.4601684329291, 24817), (832.3134151921904, 1415
.3205841976815, 26224), (861.4182731978096, 1179.830057760108, 266
62), (939.1962268834553, 2235.6611642050393, 24171), (983.78734507
50163, 1544.6478944698122, 27594), (987.480986481829, 751.32658063
03374, 26113), (1027.2186059005237, 1812.7357747235528, 22337), (1
009.297272511124, 1067.9959580177303, 29441), (1073.3610743254428,
396.5420046349942, 24164), (1163.173946922125, 2362.7623412934868,
21817), (1246.831993712714, 450.93977065695, 27993), (1251.9998189
193104, 2081.9900405620747, 27612), (1264.5361028285922, 1449.4769
522121042, 24818), (1240.9661080775747, 666.638156655997, 21244),
(1282.7850217843545, 1033.7675943007418, 25477), (1281.10165389269
88, 1256.8016225180404, 22311), (1418.733206590621, 2507.294254330
376, 9468), (1457.9099221683994, 2195.868992543406, 18373), (1441.
1067781559864, 1278.628755364807, 27028), (1444.9170751942077, 332
.08560223244586, 26518), (1492.2145157666093, 563.1180810426185, 2
6702), (1550.432126785365, 1422.573625513598, 25555), (1576.098236
3040425, 1664.2727953800531, 32035), (1596.4064278683613, 353.8907
861514101, 23367), (1604.7831247449326, 2571.1143041763025, 29404)
, (1641.601020228672, 1874.3162005277045, 28425), (1606.6555826609
12, 1205.6765246762996, 26645), (1692.2234756474124, 1564.10457408
70352, 24289), (1715.0785523339503, 2100.021961752005, 24315), (17
43.107574314502, 2446.5005263421726, 19949), (1769.6017491993102,
1111.5889792231255, 24354), (1769.0444247395255, 299.327732210153,
22555), (1831.5510561666504, 2189.6516110191765, 10273), (1838.346
4776723863, 1710.7898859639774, 18766), (1896.4773316062176, 2335.
0388260158165, 29336), (1905.9238751307987, 1347.058423439135, 229
36), (1934.1352014927816, 2081.018266932923, 30547), (1971.1954311
13622, 405.4799096109032, 28322), (2019.4309095301364, 2475.227865
6842614, 16558), (2018.7392305512258, 1682.9196542440131, 28228),
(2057.3834075462855, 1426.2110382001406, 17068), (2097.41418602596
92, 1137.9452786856284, 22642), (2115.986083430692, 2085.143495292
4196, 29102), (2147.9418581517953, 2666.572565403362, 20106), (217
4.3815100922, 442.35426530442726, 24078), (2213.9513879665296, 103
5.4618143179705, 22587), (2204.9834056037357, 865.3560707138092, 2
3984), (2270.9574774256216, 1960.1843724142893, 27797), (2278.2413
53902832, 719.6637347455676, 21715), (2292.316932417106, 1273.4683
434518647, 29978), (2303.638730628988, 2593.7604261622605, 17552),
(2341.134135100733, 1763.7534565246017, 22783), (2394.635506223534
, 617.2485243166944, 31172), (2452.367842257134, 419.9433792882334
, 31190), (2471.3008940631635, 1609.023895285264, 31094), (2481.03
7442956535, 857.36844467719, 26734), (2498.9440076010746, 1346.724
0678854596, 30522), (2515.5695110258866, 2323.5942615674157, 28161
), (2526.681463214139, 1153.7120838471023, 29196), (2557.469038674
033, 2633.0472486187846, 22625), (2566.198366256922, 1874.62918744

84546, 25463), (2582.092854137869, 564.8691027104994, 33278), (259
6.176877311465, 2479.7731548480465, 24876), (2678.4508401084013, 2
242.7193495934957, 27675), (2694.3472525970474, 1740.3403157463094
, 29264), (2714.470683792973, 811.2327162825841, 26470), (2756.597
323647486, 2515.0846109730455, 26155), (2757.544481548078, 289.793
64671632845, 23358), (2819.4346671723874, 2142.1217902522285, 2636
5), (2853.8256860592755, 1028.9779520150541, 31885), (2875.8619596
64217, 1787.3483576578774, 28709), (2942.0378971255673, 1396.41406
9591528, 13220), (2923.027858263914, 680.6254980079681, 33132), (3
010.044092175616, 1597.170984260863, 26558), (3013.8375996935188,
1153.1417476404415, 28713), (3053.2559361860003, 923.11541553302,
32344), (3077.3369416590704, 2202.259799453054, 17552), (3162.9158
788515406, 2060.9161414565824, 22848), (3157.6290545203587, 485.38
25396825397, 28980), (3217.5791327685984, 1701.5163549442254, 2384
6), (3241.8258934928094, 784.6833262138687, 28092), (3291.82059568
05086, 1226.3220229439328, 18567), (3306.5118791426357, 1025.85155
37574244, 23234), (3346.48342806054, 2029.554696979373, 31318), (3
516.342903239151, 1098.0009928432548, 24173), (3528.992839787003,
1766.6485550328284, 38686)]
```

e)

It's difficult because not all beans have the same shape. As well as spaces or that skimage regionprops split a bean in two so I get two smaller sizes.

I tried using the formula for ellipsis as this is the closest to a bean I could get. I ended up getting an answer close to the built-in .area function. So I used this instead. This value is added to the tuple with the x and y centre coordinates.

Exercise 3

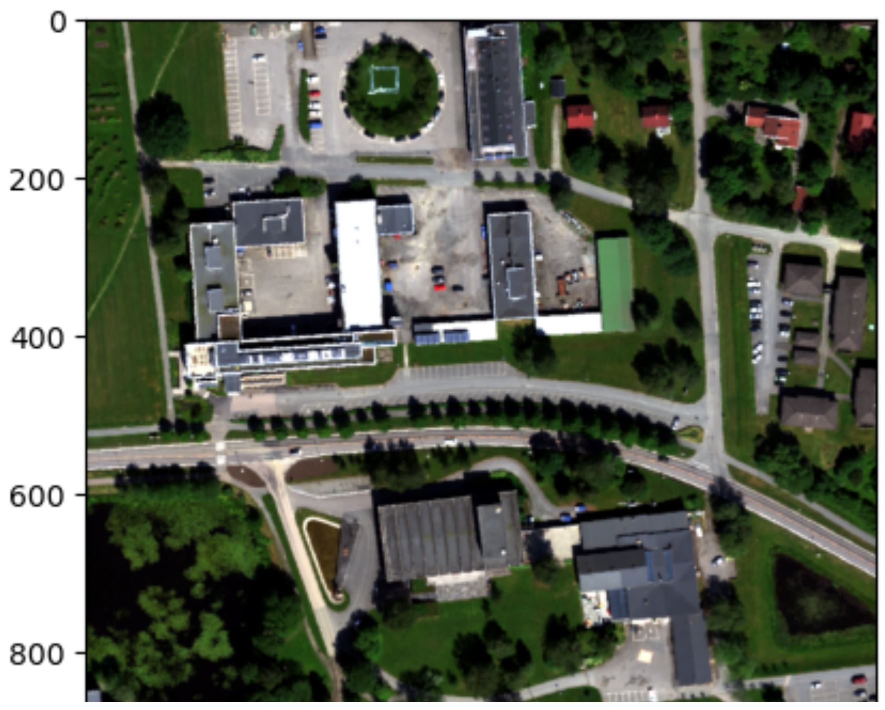
a)

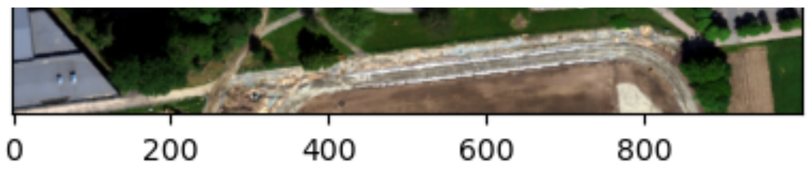
In [7]:

```

```

```python
#Import librarys
from spectral import *
import numpy as np
import matplotlib.pyplot as plt

#Load image with spectral
hyperim = np.load("nmbu.npy")
wavelength = envi.read_envi_header('nmbu.hdr')['wavelength']
ww = [float(i) for i in wavelength]


#Function to get image band to corresponding wavelengths
def color_band(image_band, wavelength_in_function):
    diff_band = []
    for i, value in enumerate(image_band):
        index_tuple = (i, abs(value - wavelength_in_function))
        diff_band.append(index_tuple)
    closest = sorted(diff_band, key=lambda t: t[1])[0]

    return closest[0]

#create dictonary with colors and teir corosponding bands
bands = {
    'blue': color_band(ww, 440),
    'green': color_band(ww, 535),
    'red': color_band(ww, 645),
    'NIR': color_band(ww, 800)}

print("bands = ", bands)

#Plott image with red, green and blue bands.
imshow(hyperim, (bands['red'], bands['green'], bands['blue']),
       stretch=((0.02, 0.98), (0.02, 0.98), (0.02, 0.98)))

plt.show()
```

```
bands =  {'blue': 11, 'green': 41, 'red': 75, 'NIR': 124}
```

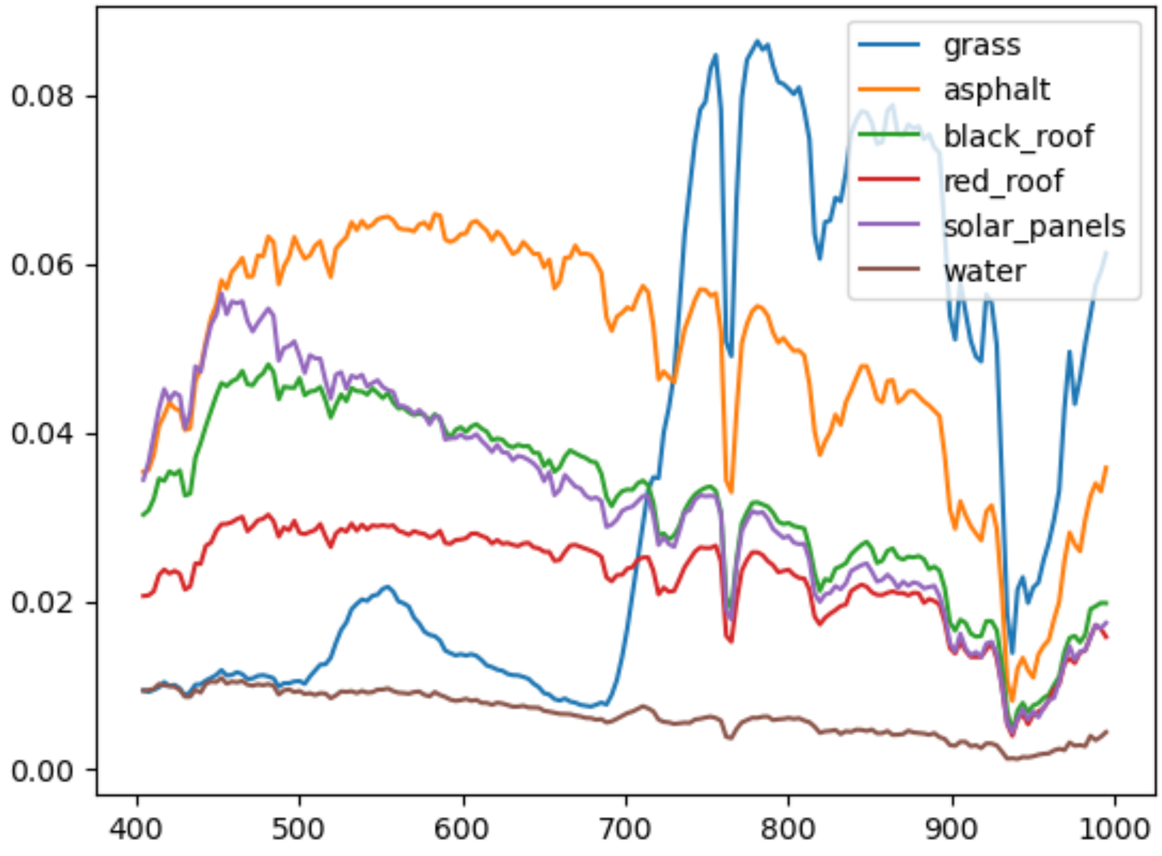




I displayed the RGB image, with the wavebands, blue: 11, green: 41 and red: 75. Their wavelength corresponds with red: 654nm, blue: 440nm and green: 535nm

b)

In [8]:
```
#Find pixles with the materials as varibal name
grass = np.array(hyperim[200, 10, :].reshape(-1, 1))
asphalt = np.array(hyperim[573, 790, :].reshape(-1, 1))
black_roof = np.array(hyperim[700, 700, :].reshape(-1, 1))
red_roof = np.array(hyperim[138, 877, :].reshape(-1, 1))
solar_panels = np.array(hyperim[410, 429, :].reshape(-1, 1))
water = np.array(hyperim[735, 886, :].reshape(-1, 1))

#Plott the materials with the wavebands to se the spectrum as a fun
plt.figure()
plt.plot(ww, grass)
plt.plot(ww, asphalt)
plt.plot(ww, black_roof)
plt.plot(ww, red_roof)
plt.plot(ww, solar_panels)
plt.plot(ww, water)
plt.legend(['grass', 'asphalt', 'black_roof', 'red_roof', 'solar_pa
plt.show()
```

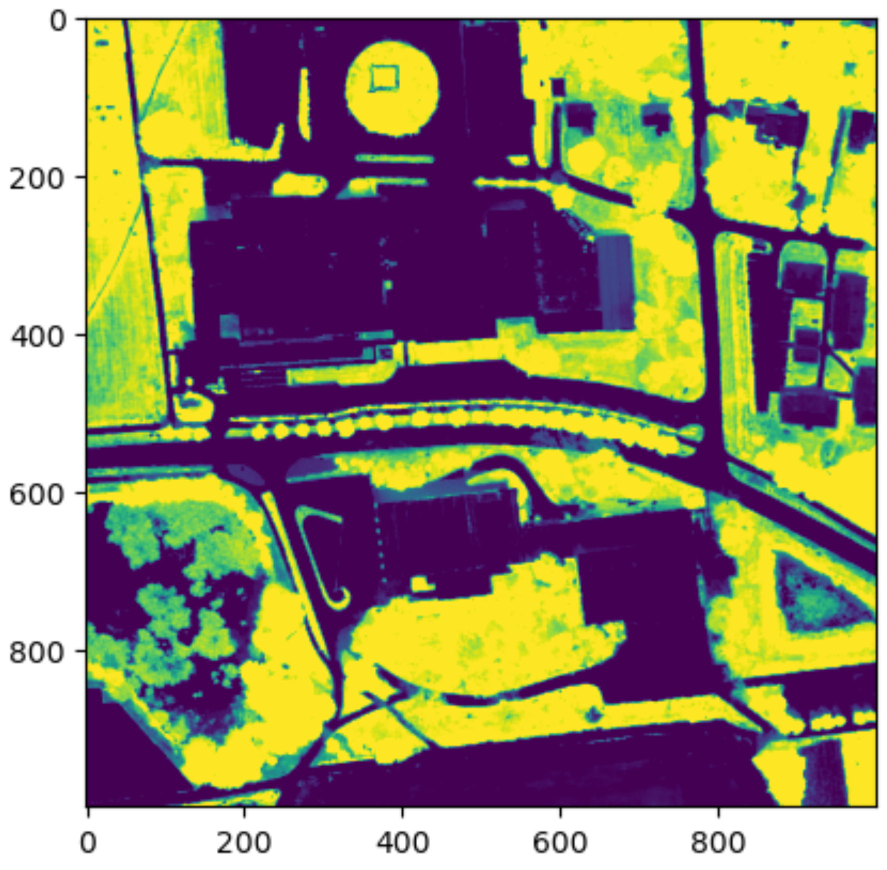


The grass is recognizable by the peaks at the green wavelength 500-600 nm. Vegetation reflects more infrared radiation compared to other region types. This is the way the grass also has a high value of around 800 nm.

We also see that the values are higher at the read wavelength for the red roofs and the blue wavelength has a higher value for water. Asphalt which is grey absorbs most of the wavelengths and that means it doesn't have any characteristics about them in the plot.

c)

In [9]:
```
#Define image with just the one bands.
band_nir = hyperim[:, :, bands['NIR']]
band_red = hyperim[:, :, bands['red']]

#Function to calculate the NDVI from red and nir band
def calculate_NDVI(NIR, Red):
    NDVI = (NIR - Red) / (NIR + Red)
    return NDVI

ndvi_ima = calculate_NDVI(band_nir, band_red)

#Plott the results
plt.imshow(ndvi_ima, vmin=0, vmax=0.7)
plt.show()
```

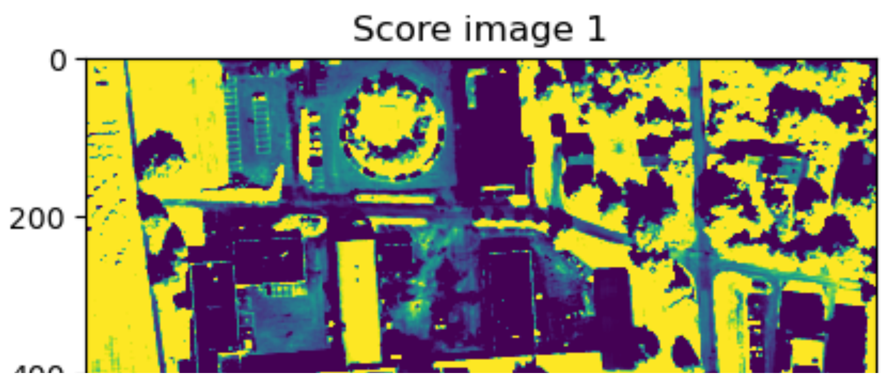


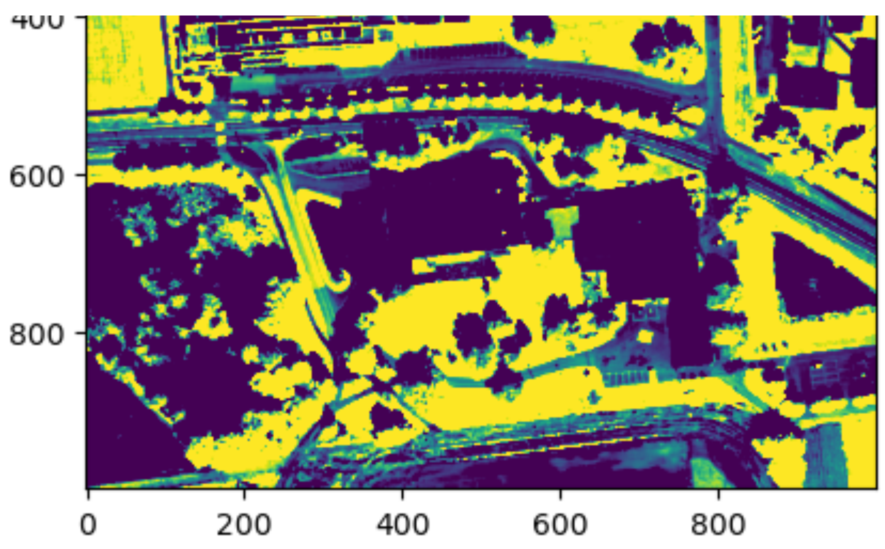
The green regions are mostly vegetation, the brighter or closer to yellow the green colour is, the healthier the vegetation is supposed to be. Healthie vegetation reflects more infrared-radiation.

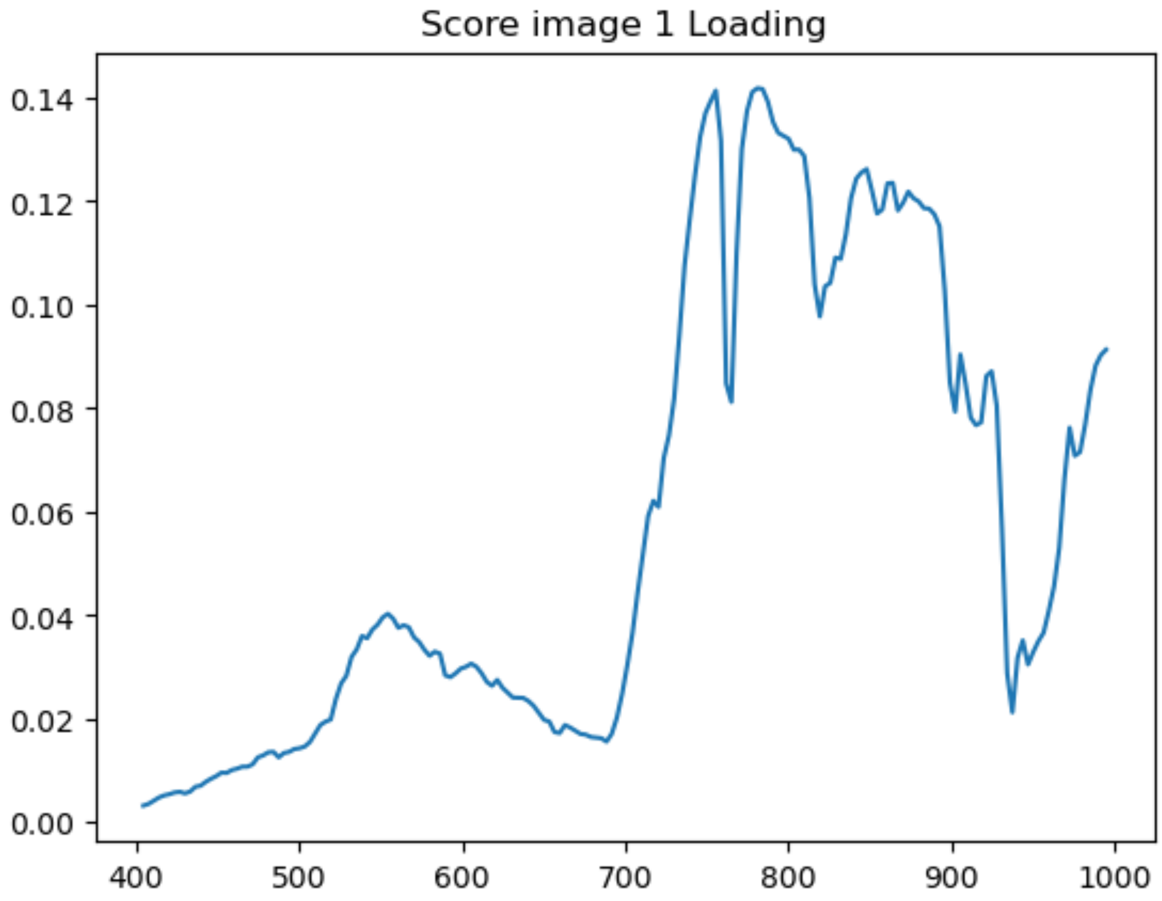
d)

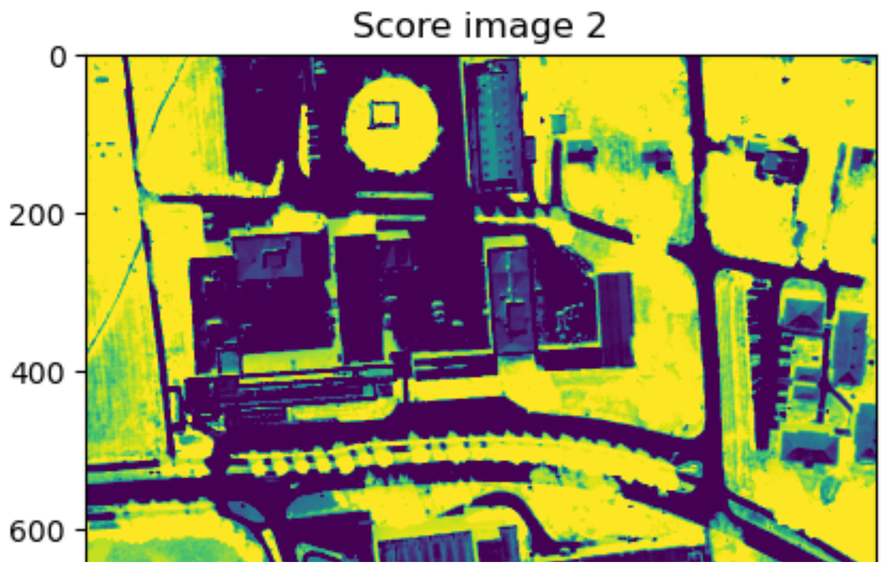
In [10]:

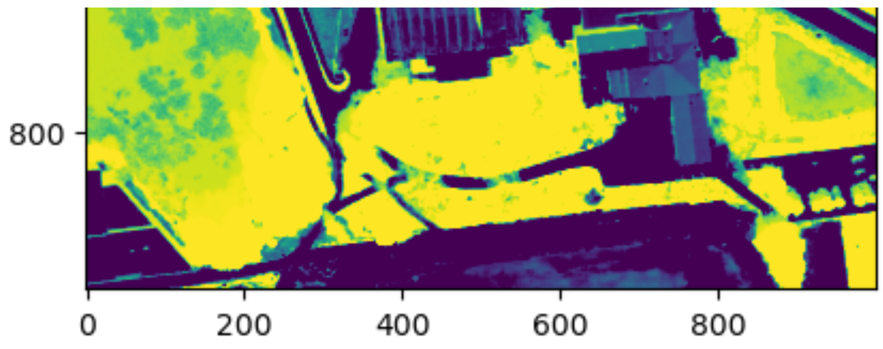
```python
#Create a principal components Analysis of the hole image.
pc = principal_components(hyperim)
img_pc = pc.transform(hyperim)

loading = pc.eigenvectors

#Plotting results
plt.imshow(img_pc[:, :, 0], vmin=-0.1, vmax=0.15)
plt.title('Score image 1')
plt.show()

plt.plot(ww, loading[:, 0])
plt.title('Score image 1 Loading')
plt.show()

plt.imshow(img_pc[:, :, 1], vmin=-0.1, vmax = 0.15)
plt.title('Score image 2')
plt.show()

plt.plot(ww, loading[:, 1])
plt.title('Score image 2 Loading')
plt.show()

plt.imshow(img_pc[:, :, 2], vmin=-0.1, vmax = 0.15)
plt.title('Score image 3')
plt.show()

plt.plot(ww, loading[:, 2])
plt.title('Score image 3 Loading')
plt.show()

plt.imshow(img_pc[:, :, 3], vmin=-0.1, vmax = 0.15)
plt.title('Score image 4')
plt.show()

plt.plot(ww, loading[:, 3])
plt.title('Score image 4 Loading')
plt.show()

plt.imshow(img_pc[:, :, 4], vmin=-0.1, vmax = 0.15)
plt.title('Score image 5')
plt.show()

plt.plot(ww, loading[:, 4])
plt.title('Score image 5 Loading')
plt.show()
```



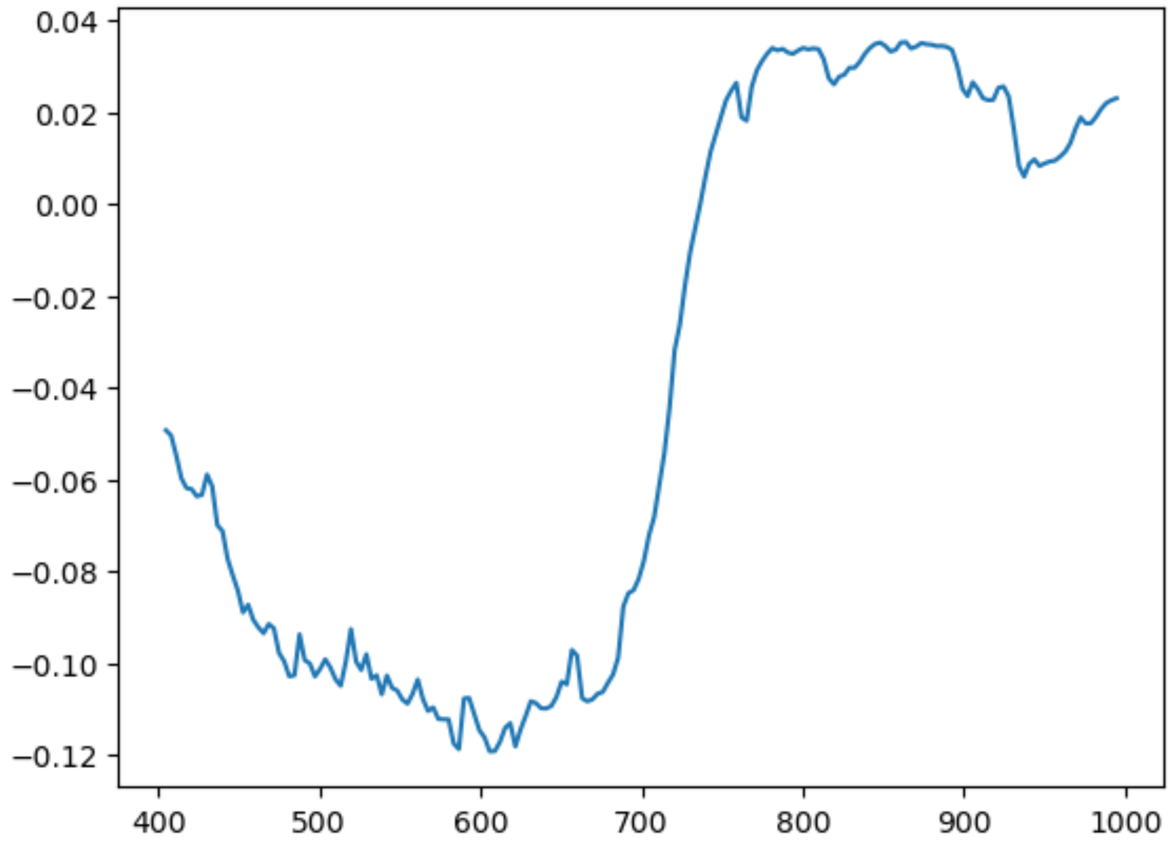



Score image 1 Loading

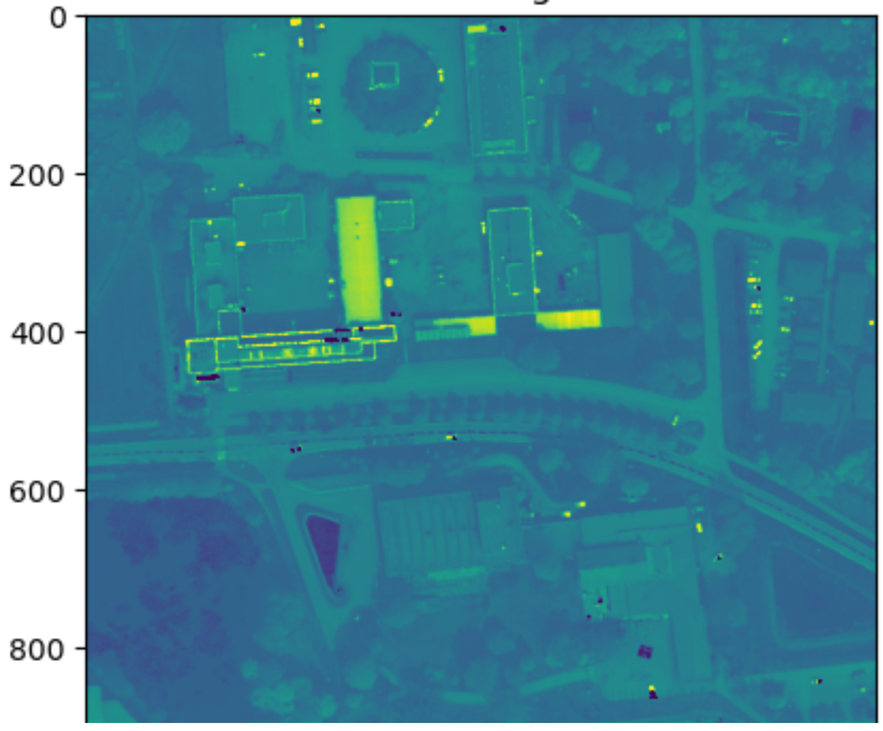

Score image 2

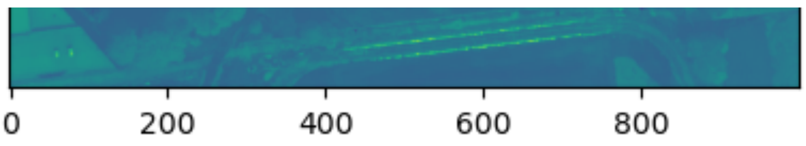


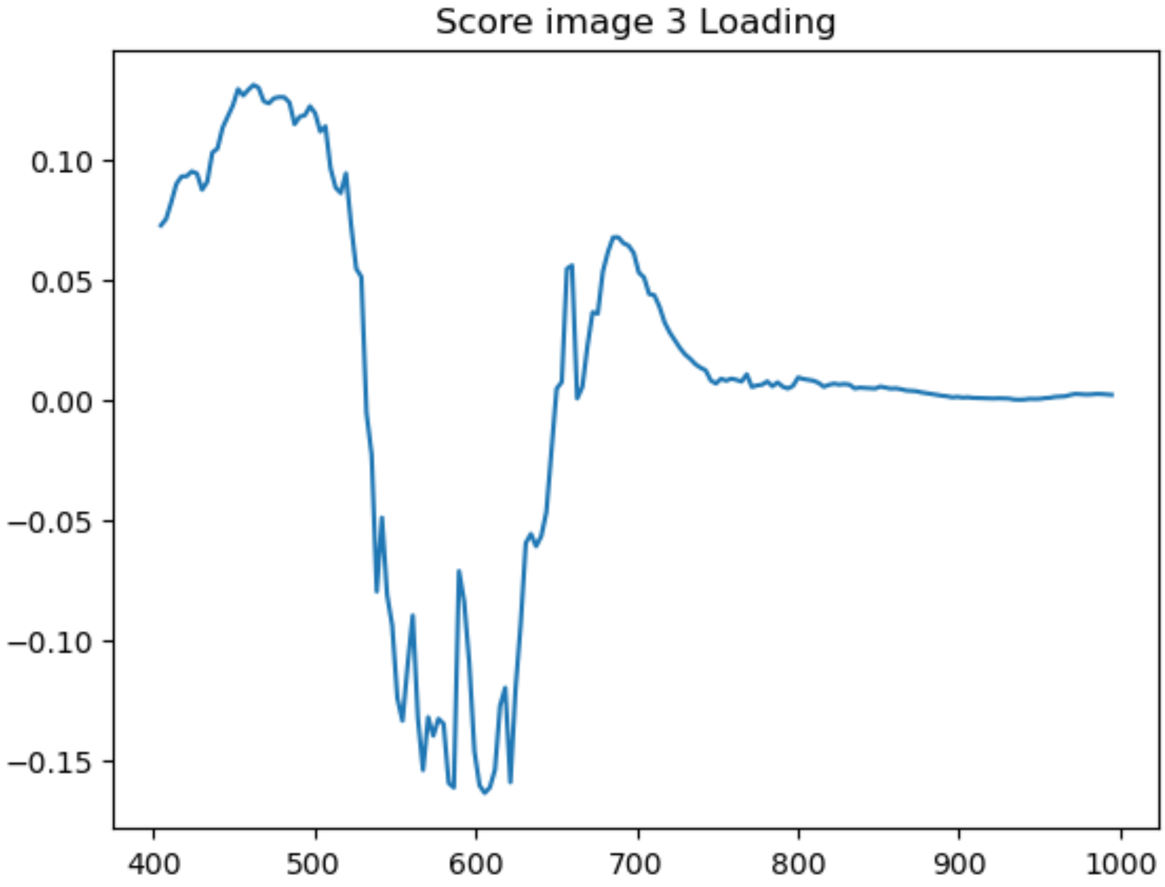
Score image 2 Loading

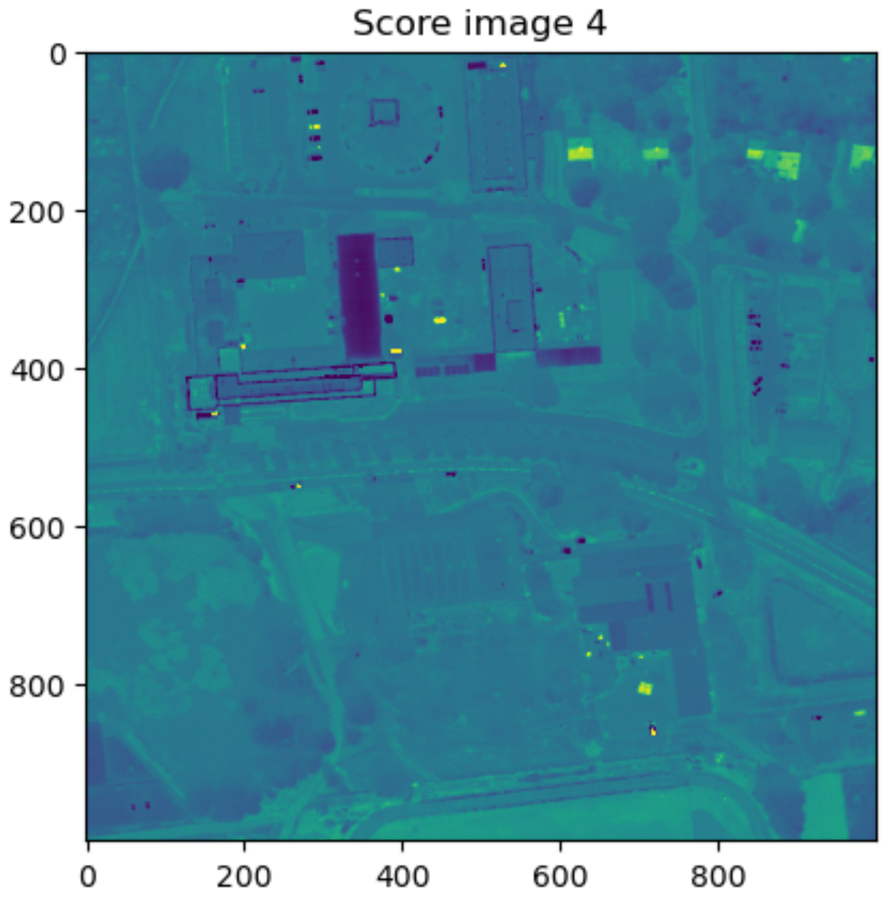


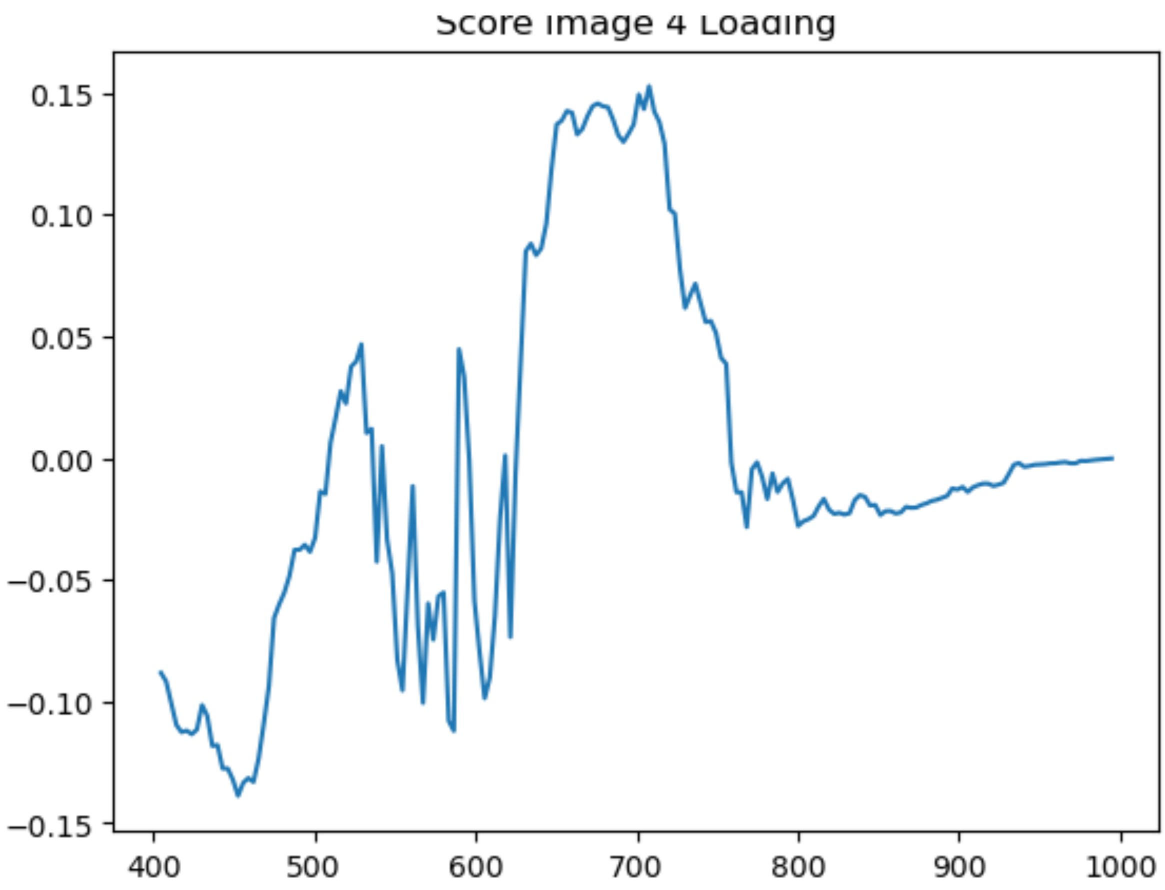
Score image 3



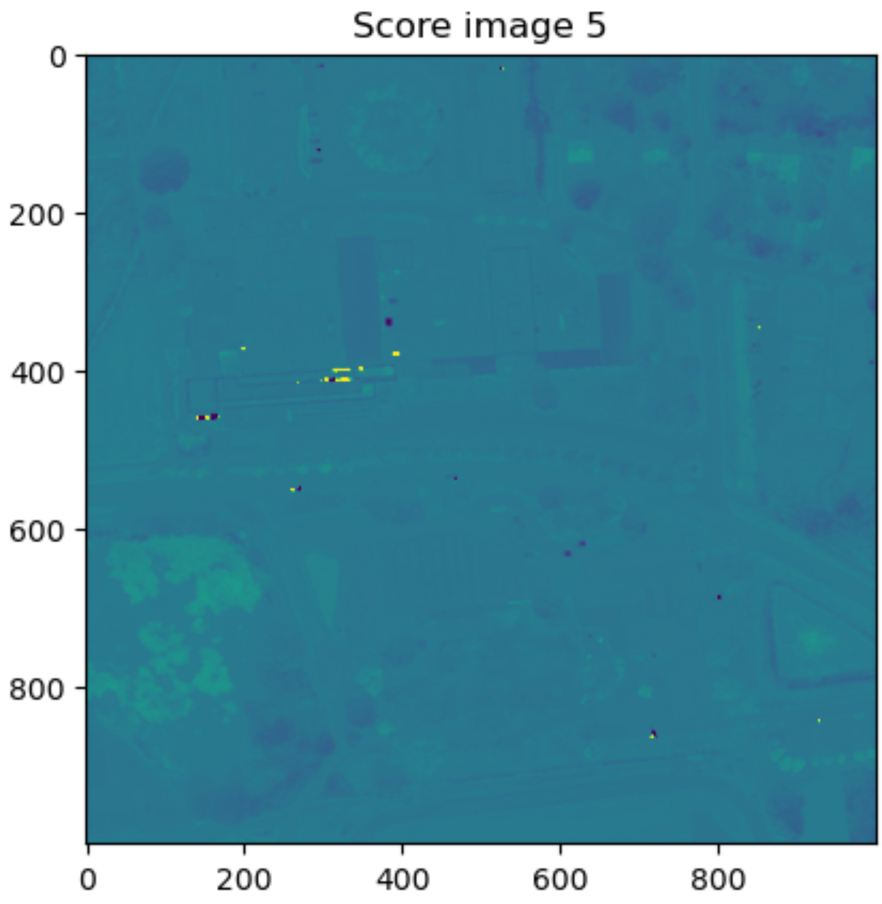

0
200
400
600
800


Score image 3 Loading
0.10
0.05
0.00
−0.05
−0.10
−0.15
400
500
600
700
800
900
1000

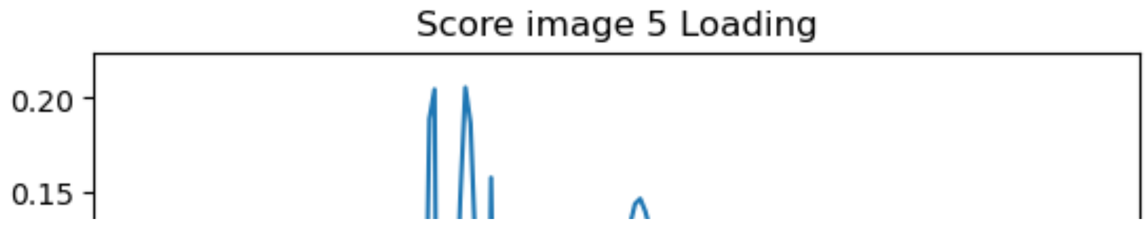

Score image 4
0
200
400
600
800
0
200
400
600
800



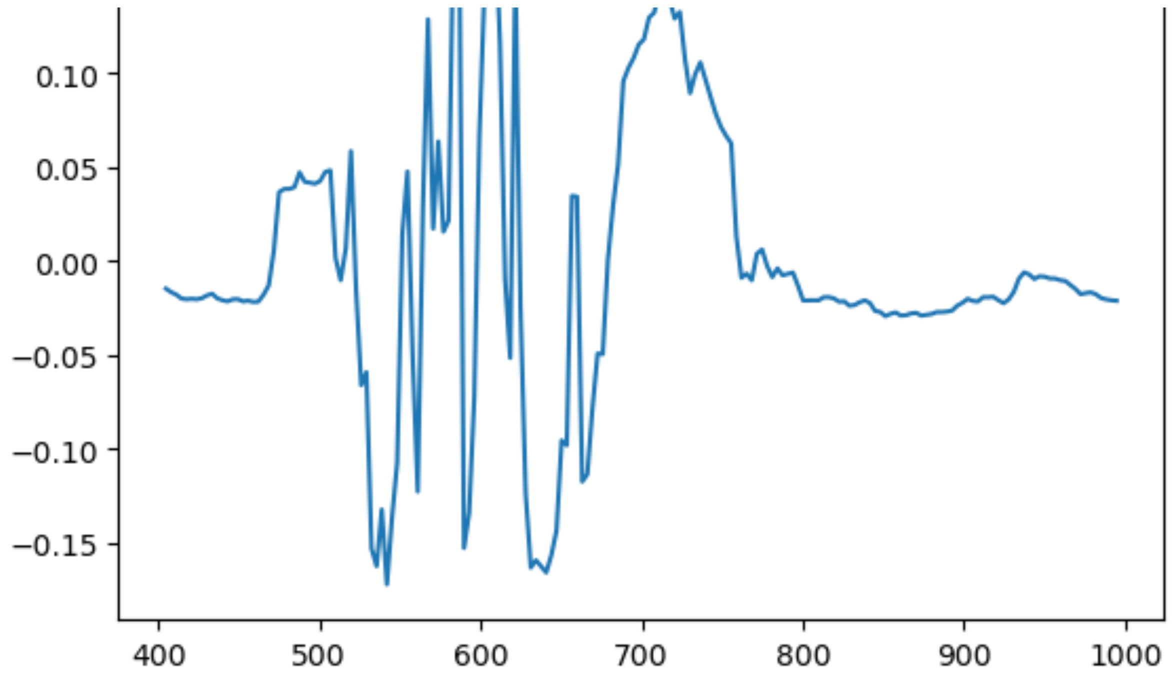
Score image 4 Loading
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
400
500
600
700
800
900
1000



Score image 5
0
200
400
600
800
0
200
400
600
800



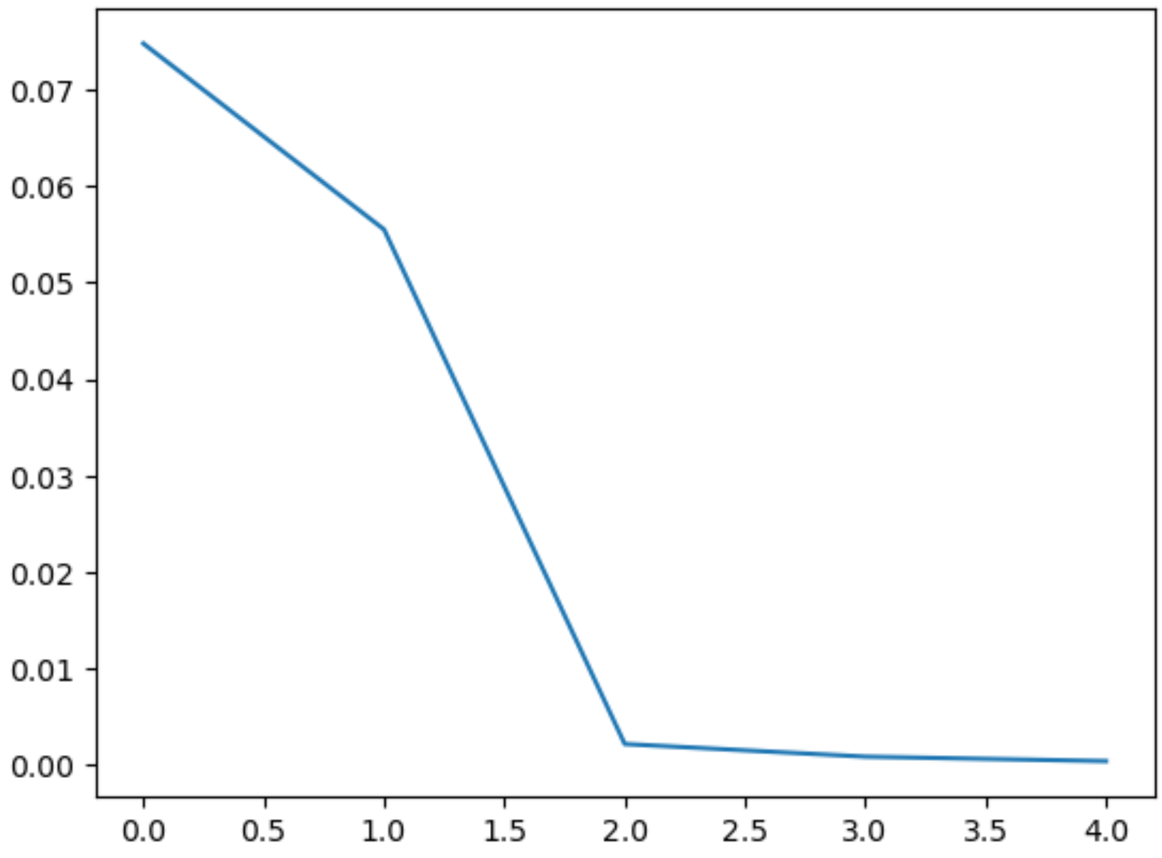
Score image 5 Loading
0.20
0.15



We can see that the first 2 images contain the most variance and the other images don't add that much new.

We can also see that the first score image most likely represents vegetation, it has a noticeable peak at the green wavelength as well as the infrared wavelength. In the second one, we see a low value in green and a high in infrared, this high in infrared could also categorise vegetation. In the third one, we see a high value in the blue colour region, and a low in red, which could represent water.

In [11]:
```
#Plotting the eigenvales of the first 5 score images.
plt.figure()
plt.plot(pc.eigenvalues[0:5])
plt.show()
```



There's a break in the curve at component no 2 for the same reason the first two images have the most colour. There is a lot of different information in these two, and in the 3 and 4, we do capture not a lot of variation.

In [12]:
```
#Test to check wich presentage the 2 score image applay and when th
pc_01 = pc.reduce(fraction=0.96693)
img_pc = pc_01.transform(hyperim)
shape = np.shape(img_pc)

pc_02 = pc.reduce(fraction=0.96694)
img_pc2 = pc_02.transform(hyperim)
shape_2 = np.shape(img_pc2)

#Print to check the number of components
print(shape)
print(shape_2)
```

```
(1000, 1000, 2)
(1000, 1000, 3)
```

We see that 2 componets cover 96.693% of the explained variance

In [13]:
```python
#Reduse to only take with 99.9% of the explained variace
pc_0999 = pc.reduce(fraction=0.999)
img_pc = pc_0999.transform(hyperim)
shape = np.shape(img_pc)

#The last element in the print statement diffines number of compone
print(shape)
```

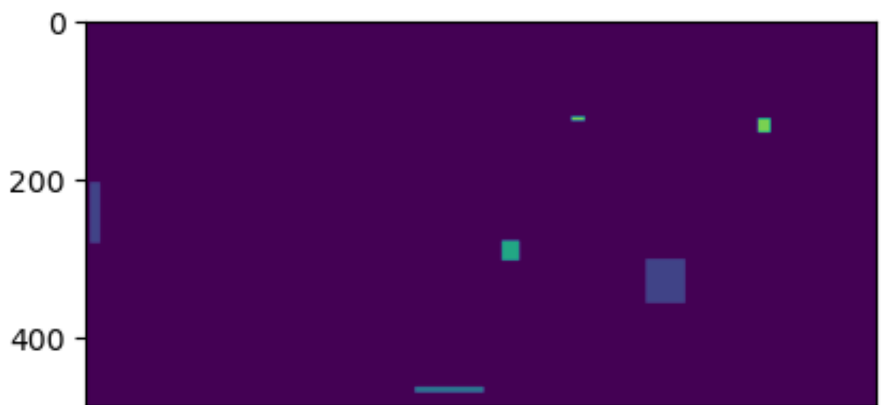
```
(1000, 1000, 31)
```

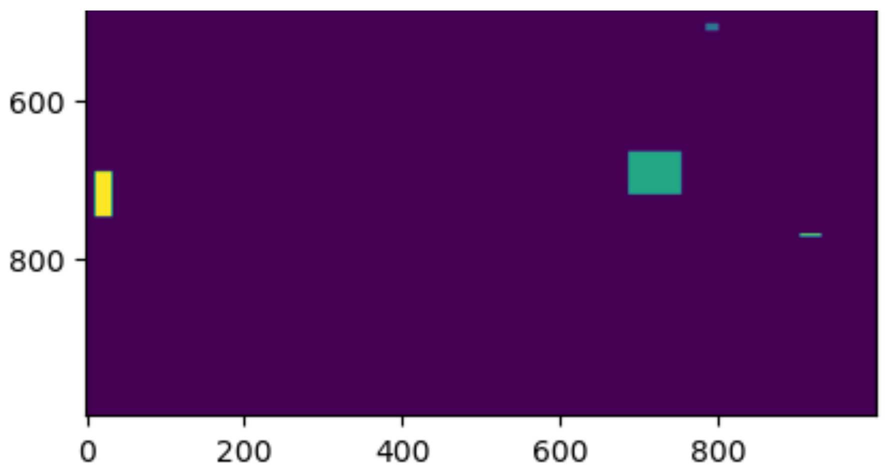
Here we can see that 31 components cover 99.9% of the explained variance

e)

In [14]:

```
#Create an empty image called groundtruth.
shape = hyperim.shape
groundtruth = np.zeros([shape[0],shape[1]])

#Create pixel areas with the diffined material so we can train the
groundtruth[206:283, 6:19] = 1.0   #grass
groundtruth[303:359, 709:759] = 1.0   #grass

groundtruth[465:472, 417:504] = 2.0 # asphalt
groundtruth[505:514, 785:801] = 2.0 # asphalt

groundtruth[667:721, 687:754] = 3.0 # black roof
groundtruth[280:305, 527:549] = 3.0 # black roof

groundtruth[125:143, 851:867] = 4.0 # red roof
groundtruth[123:128, 615:632] = 4.0 # red roof


groundtruth[692:749, 12:34] = 5.0 # water
groundtruth[771:774, 904:931] = 5.0 # water

#Plott the sample areas.
plt.figure()
plt.imshow(groundtruth)
plt.show()

# Gaussian Maximum Likelihood classification
classes = create_training_classes(hyperim, groundtruth)
gmlc = GaussianClassifier(classes)
clmap = gmlc.classify_image(hyperim)

#Plott with all score images
plt.figure()
imshow(classes=clmap)
plt.show()

#Create need training with only the first 8 score images
classes = create_training_classes(img_pc[:, :, 0:8], groundtruth)
gmlc = GaussianClassifier(classes)
clmap = gmlc.classify_image(img_pc[:, :, 0:8])

#Plott with 8 score images
plt.figure()
imshow(classes=clmap)
plt.show()
```

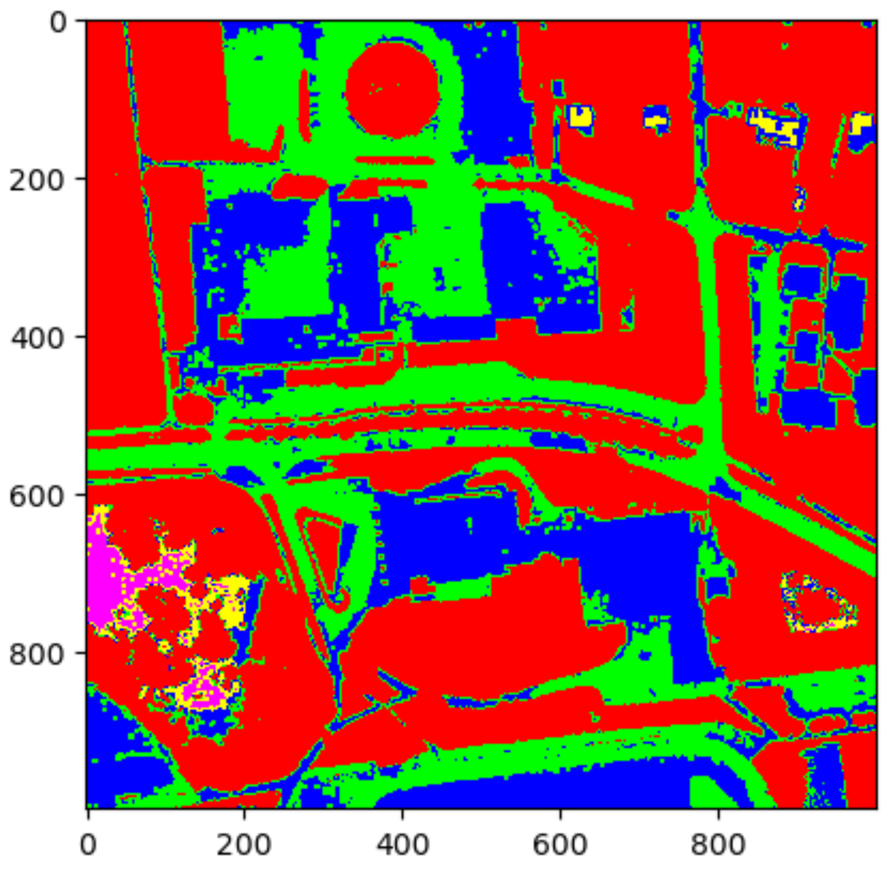




```
spectral:INFO: Setting min samples to 186
```

```
Processing...  0.0Processing... 20.0Processing... 40.0Processing..
. 60.0Processing... 80.0Processing... 100.Processing...done
```

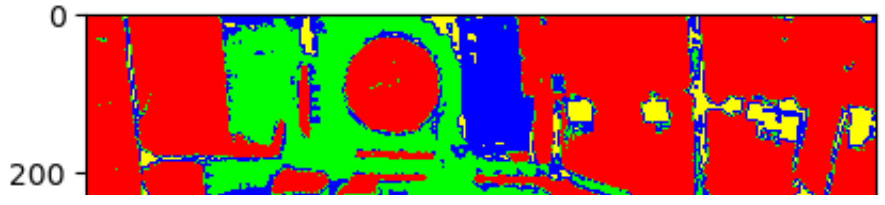
```
<Figure size 640x480 with 0 Axes>
```

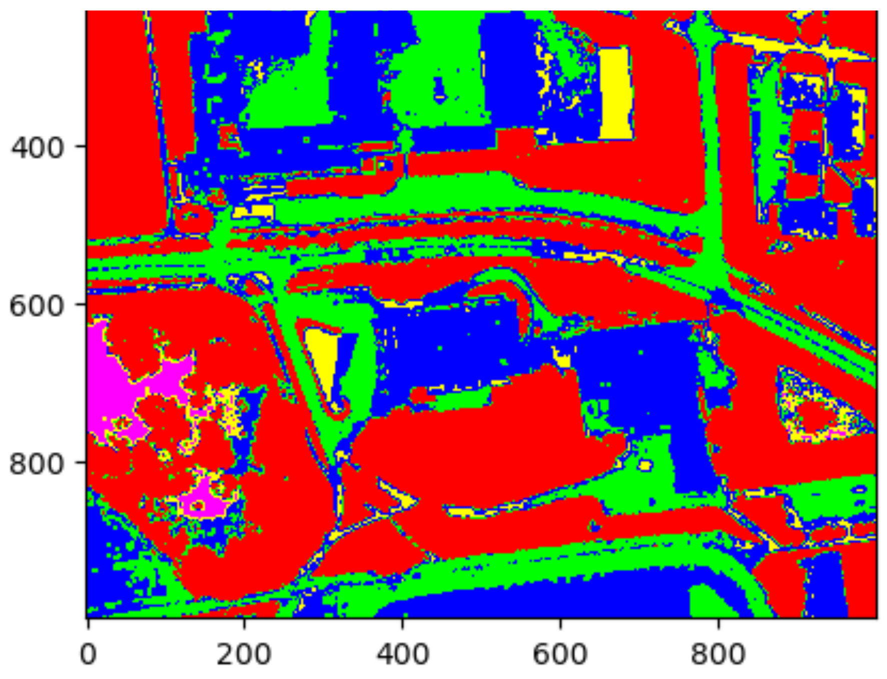


```
spectral:INFO: Setting min samples to 8
```

```
Processing...  0.0Processing... 20.0Processing... 40.0Processing..
. 60.0Processing... 80.0Processing... 100.Processing...done
```

```
<Figure size 640x480 with 0 Axes>
```





In the training image we see that:

- Yellow = water
- Light green = red roofs
- Dark blue = grass/vegetation
- Blue = asphalt
- Light blue = dark roofs

We can see that water isn't very well represented in these images. It managed to get the large water on the left-hand side, but not the small pond on the right-hand side. We also see that asphalt and grass are quite accurate, with some disturbance.

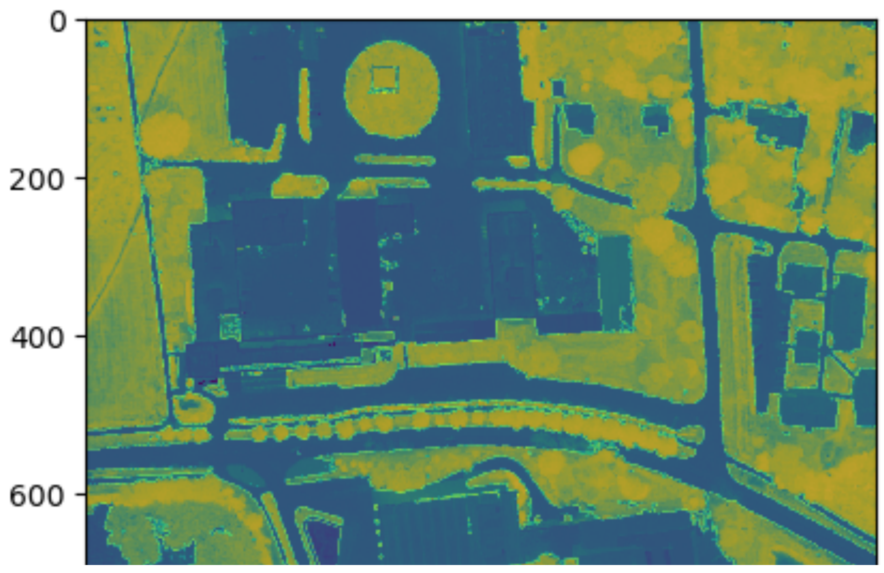
The differences between using the whole image and only the first 8 score images are noticeable. We get a lot more false true. So we got a lot of places that have red roofs when in reality it doesn't.

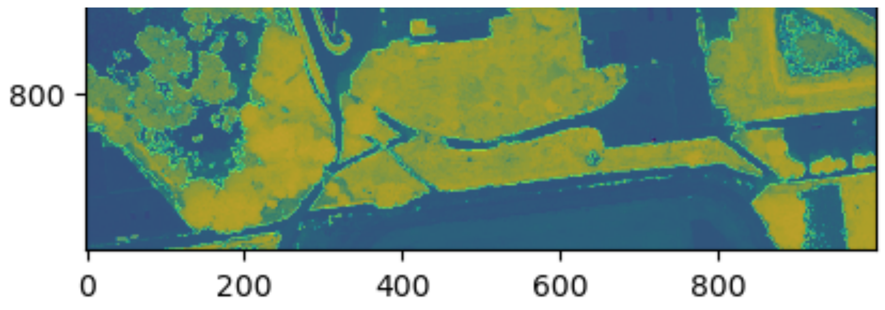
f)

In [15]:

```
#Create an empty image called groundtruth.
shape = hyperim.shape
groundtruth = np.zeros([shape[0],shape[1]])

groundtruth[206:283, 6:19] = 1.0   #grass
groundtruth[303:359, 709:759] = 1.0   #grass
groundtruth[465:472, 417:504] = 2.0 # asphalt
groundtruth[505:514, 785:801] = 2.0 # asphalt
groundtruth[667:721, 687:754] = 3.0 # black roof
groundtruth[280:305, 527:549] = 3.0 # black roof
groundtruth[125:143, 851:867] = 4.0 # red roof
groundtruth[123:128, 615:632] = 4.0 # red roof
groundtruth[692:749, 12:34] = 5.0 # water
groundtruth[771:774, 904:931] = 5.0 # water


# Gaussian Maximum Likelihood classification
classes = create_training_classes(hyperim, groundtruth)
gmlc = GaussianClassifier(classes)
clmap = gmlc.classify_image(hyperim)

#We remove all the other values than grass
shape = np.shape(clmap)
for i in range(shape[0]):
    for j in range(shape[1]):
        if int(clmap[i][j]) != 1:
            clmap[i][j] = 0

#Create edges on the binary image with grass as 1
class_edge = skimage.filters.sobel(clmap)

#Plott the NDVI image and the edge image ontop with a alpha paramet
#The alpha parameter is how visible the top image should be
plt.imshow(ndvi_ima, interpolation='none')
plt.imshow(class_edge, alpha=0.3, interpolation='none')
plt.show()
```

```
spectral:INFO: Setting min samples to 186

Processing...  0.0Processing... 20.0Processing... 40.0Processing..
. 60.0Processing... 80.0Processing... 100.Processing...done
```





Found that matplotlib build inn function alpha was the best to layer the two images. Tried first adding all the pixels, one by one, and this image looks exactly like just the NDVI image. But with the built-in function, we can much clearer see the edges and where the vegetation is.