1 Mumford-Shah Piecewise Constant

1.3
Yes because variation between colors in regions with low variation in the noised image are set to 0 which minimizes the potts energy term and the sharp edges between these regions helps minimize the squared error term by ensuring that colors are at least close.

2 Foreground-background graph-cut

2.2.c
Because these are superpixels. Close to the diagonal means likely close together, but it is not always the case like it is with pixels.

2.3.c
It is a hyperparameter that balances between two goals. The source to every weight makes it so that we will prioritize pixels that are close to the foreground pixel in terms of color and distance while the weights to adjacent pixels make it so that edges are captured well. By weighting the adjacent pixels less, this is basically saying we care more about similarity to the foreground pixel.

2.4.a
The boots are close to each other and visually similar in terms of color distribution. These are the two terms of the capacity so there is high capacity between the boots.

2.4.b
Since the source connects to all pixels, not just the foreground pixel, even if there is no adjacent pixel between the signs, the close visual similarity means that all signs will be segmented. If we changed the value of dnorm to something smaller this might not be the case anymore since the capacity to further away pixels would be smaller.

2.4.c
Like I said in b decreasing the value of dnorm would make pixels further away from the selected foreground have less capacity. We could also simply enforce that the source is only connected to the foreground node.

3 Segmentation with minimum spanning forest

3.3
The main difference is that whereas before we had capacity now we have cost. This means that we need to invert the sign. We then scale and add by constants to ensure that everything is positive to make a distance measure.

EECS_504_PS3_adempst_61259596

March 17, 2026

1 EECS 504 Problem Set 3

Please provide the following information (e.g. Jason Corso, jjcorso):

Aidan Dempster, adempst

Important: after you download the .ipynb file, please name it as EECS504_PS3_<your_uniquename>_<your_umid>.ipynb before you submit it to canvas. Example: EECS504_PS3_adam_01101100.ipynb.

2 Introduction

We'll provide you with starter code, like this, in a Jupyter notebook for most problem sets. Please fill in the code to complete the assignment, and submit your notebook to Canvas as a .ipynb file. You can, of course, initially write your code offline in an editor like Emacs or Vim – we'd just like the final output to be in a notebook format to make grading more consistent.

Please note that *we won't run your code*. The notebook you submit should already contain all of the results we ask for. In other words, outputs should be computed *before you submit*. Also, please do not include long, unnecessary outputs (a few print statements and visualizations are fine, but pages of debugging messages make grading difficult).

You will have to make three submissions:

To Gradescope:

1a. A pdf file as your write-up, including your answers to all the questions. You DON'T need to include the result images from the code in the writeup.

1b: Convert the .ipynb notebook to a pdf using the code in the last cell. Make sure it contains all the visualizations (i.e, the result images). Merge this notebook pdf to your write-up and upload your file to Gradescope as a single submission. While assigning pages for your Gradescope assign pages for the code as well.

To Canvas: 2. This .ipynb file containing all the visualizations should be submitted to canvas. ***

3 Starting

Run the following code to import the modules and download all the files you'll need.. After your finish the assignment, remember to run all cells and save the note book to your local machine as a

.ipynb file for Canvas submission.

[1]:
```
%matplotlib inline
import cv2
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal
import scipy
from google.colab.patches import cv2_imshow
from skimage import segmentation
from skimage import io, color
import skimage
```

[2]:
```
! wget -O img1.bmp "https://drive.google.com/uc?
 ↪export=download&id=1l6Isz73GERmQBLe2bXIOUdoHndJsmG20"
! wget -O img2.bmp "https://drive.google.com/uc?
 ↪export=download&id=1wqflaXvmMF_V9xlR3DK1lc9CpD7USgFx"
! wget -O fhat.png "https://drive.google.com/uc?
 ↪export=download&id=1TO-X-2wisOMUPHIqoHi9fRSd5oWVC6-3"
! wget -O f.png "https://drive.google.com/uc?
 ↪export=download&id=1qN_MOOK8-j5lqNJb19u8oaZ9CJqVycFF"
! wget -O flower.jpg "https://drive.google.com/uc?
 ↪export=download&id=1_blXIpePOJM51JHahPmuJUTMeD1L696P"
! wget -O veggies.jpg "https://drive.google.com/uc?
 ↪export=download&id=118AOfJ6dOzCHPY1zlWnQwfPw_sCvxnqF"
! wget -O porch.jpg "https://drive.google.com/uc?
 ↪export=download&id=1JTFnJkuhHHasNgVpexocbuNWYF-QpmKC"
```

```
--2026-03-17 18:51:31--
https://drive.google.com/uc?export=download&id=1l6Isz73GERmQBLe2bXIOUdoHndJsmG20
Resolving drive.google.com (drive.google.com)… 74.125.130.113, 74.125.130.101,
74.125.130.100, …
Connecting to drive.google.com (drive.google.com)|74.125.130.113|:443…
connected.
HTTP request sent, awaiting response… 303 See Other
Location: https://drive.usercontent.google.com/download?id=1l6Isz73GERmQBLe2bXIO
UdoHndJsmG20&export=download [following]
--2026-03-17 18:51:32--  https://drive.usercontent.google.com/download?id=1l6Isz
73GERmQBLe2bXIOUdoHndJsmG20&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)…
74.125.24.132, 2404:6800:4003:c03::84
Connecting to drive.usercontent.google.com
(drive.usercontent.google.com)|74.125.24.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 486 [application/octet-stream]
Saving to: ‘img1.bmp’
```

```
img1.bmp            100%[===================>]     486  --.-KB/s    in 0s

2026-03-17 18:51:33 (12.2 MB/s) - ‘img1.bmp’ saved [486/486]

--2026-03-17 18:51:33--
https://drive.google.com/uc?export=download&id=1wqflaXvmMF_V9xlR3DK1lc9CpD7USgFx
Resolving drive.google.com (drive.google.com)… 74.125.130.113, 74.125.130.101,
74.125.130.100, …
Connecting to drive.google.com (drive.google.com)|74.125.130.113|:443…
connected.
HTTP request sent, awaiting response… 303 See Other
Location: https://drive.usercontent.google.com/download?id=1wqflaXvmMF_V9xlR3DK1
lc9CpD7USgFx&export=download [following]
--2026-03-17 18:51:33--  https://drive.usercontent.google.com/download?id=1wqfla
XvmMF_V9xlR3DK1lc9CpD7USgFx&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)…
74.125.24.132, 2404:6800:4003:c03::84
Connecting to drive.usercontent.google.com
(drive.usercontent.google.com)|74.125.24.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 486 [application/octet-stream]
Saving to: ‘img2.bmp’

img2.bmp            100%[===================>]     486  --.-KB/s    in 0s

2026-03-17 18:51:34 (28.3 MB/s) - ‘img2.bmp’ saved [486/486]

--2026-03-17 18:51:35--
https://drive.google.com/uc?export=download&id=1TO-X-2wisOMUPHIqoHi9fRSd5oWVC6-3
Resolving drive.google.com (drive.google.com)… 74.125.130.113, 74.125.130.101,
74.125.130.100, …
Connecting to drive.google.com (drive.google.com)|74.125.130.113|:443…
connected.
HTTP request sent, awaiting response… 303 See Other
Location: https://drive.usercontent.google.com/download?id=1TO-X-
2wisOMUPHIqoHi9fRSd5oWVC6-3&export=download [following]
--2026-03-17 18:51:35--  https://drive.usercontent.google.com/download?id=1TO-X-
2wisOMUPHIqoHi9fRSd5oWVC6-3&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)…
74.125.24.132, 2404:6800:4003:c03::84
Connecting to drive.usercontent.google.com
(drive.usercontent.google.com)|74.125.24.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 543 [image/png]
Saving to: ‘fhat.png’

fhat.png            100%[===================>]     543  --.-KB/s    in 0s
```

```
2026-03-17 18:51:36 (20.0 MB/s) - ‘fhat.png’ saved [543/543]

--2026-03-17 18:51:36--
https://drive.google.com/uc?export=download&id=1qN_M0OK8-j5lqNJb19u8oaZ9CJqVycFF
Resolving drive.google.com (drive.google.com)… 74.125.130.113, 74.125.130.101,
74.125.130.100, …
Connecting to drive.google.com (drive.google.com)|74.125.130.113|:443…
connected.
HTTP request sent, awaiting response… 303 See Other
Location: https://drive.usercontent.google.com/download?id=1qN_M0OK8-
j5lqNJb19u8oaZ9CJqVycFF&export=download [following]
--2026-03-17 18:51:36--  https://drive.usercontent.google.com/download?id=1qN_M0
OK8-j5lqNJb19u8oaZ9CJqVycFF&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)…
74.125.24.132, 2404:6800:4003:c03::84
Connecting to drive.usercontent.google.com
(drive.usercontent.google.com)|74.125.24.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 5138 (5.0K) [image/png]
Saving to: ‘f.png’

f.png               100%[===================>]   5.02K  --.-KB/s    in 0s

2026-03-17 18:51:38 (37.7 MB/s) - ‘f.png’ saved [5138/5138]

--2026-03-17 18:51:38--
https://drive.google.com/uc?export=download&id=1_blXIpePOJM51JHahPmuJUTMeD1L696P
Resolving drive.google.com (drive.google.com)… 74.125.130.113, 74.125.130.101,
74.125.130.100, …
Connecting to drive.google.com (drive.google.com)|74.125.130.113|:443…
connected.
HTTP request sent, awaiting response… 303 See Other
Location: https://drive.usercontent.google.com/download?id=1_blXIpePOJM51JHahPmu
JUTMeD1L696P&export=download [following]
--2026-03-17 18:51:38--  https://drive.usercontent.google.com/download?id=1_blXI
pePOJM51JHahPmuJUTMeD1L696P&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)…
74.125.24.132, 2404:6800:4003:c03::84
Connecting to drive.usercontent.google.com
(drive.usercontent.google.com)|74.125.24.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 239429 (234K) [image/jpeg]
Saving to: ‘flower.jpg’

flower.jpg          100%[===================>] 233.82K  --.-KB/s    in 0.003s

2026-03-17 18:51:40 (70.7 MB/s) - ‘flower.jpg’ saved [239429/239429]
```

```
--2026-03-17 18:51:40--
https://drive.google.com/uc?export=download&id=118AOfJ6dOzCHPY1zlWnQwfPw_sCvxnqF
Resolving drive.google.com (drive.google.com)… 74.125.130.113, 74.125.130.101,
74.125.130.100, …
Connecting to drive.google.com (drive.google.com)|74.125.130.113|:443…
connected.
HTTP request sent, awaiting response… 303 See Other
Location: https://drive.usercontent.google.com/download?id=118AOfJ6dOzCHPY1zlWnQ
wfPw_sCvxnqF&export=download [following]
--2026-03-17 18:51:40--  https://drive.usercontent.google.com/download?id=118AOf
J6dOzCHPY1zlWnQwfPw_sCvxnqF&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)…
74.125.24.132, 2404:6800:4003:c03::84
Connecting to drive.usercontent.google.com
(drive.usercontent.google.com)|74.125.24.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 95134 (93K) [image/jpeg]
Saving to: ‘veggies.jpg’

veggies.jpg         100%[===================>]  92.90K  --.-KB/s    in 0.001s

2026-03-17 18:51:41 (65.7 MB/s) - ‘veggies.jpg’ saved [95134/95134]

--2026-03-17 18:51:42--
https://drive.google.com/uc?export=download&id=1JTFnJkuhHHasNgVpexocbuNWYF-QpmKC
Resolving drive.google.com (drive.google.com)… 74.125.130.113, 74.125.130.101,
74.125.130.100, …
Connecting to drive.google.com (drive.google.com)|74.125.130.113|:443…
connected.
HTTP request sent, awaiting response… 303 See Other
Location:
https://drive.usercontent.google.com/download?id=1JTFnJkuhHHasNgVpexocbuNWYF-
QpmKC&export=download [following]
--2026-03-17 18:51:42--
https://drive.usercontent.google.com/download?id=1JTFnJkuhHHasNgVpexocbuNWYF-
QpmKC&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)…
74.125.24.132, 2404:6800:4003:c03::84
Connecting to drive.usercontent.google.com
(drive.usercontent.google.com)|74.125.24.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 339081 (331K) [image/png]
Saving to: ‘porch.jpg’

porch.jpg           100%[===================>] 331.13K  --.-KB/s    in 0.004s

2026-03-17 18:51:43 (84.6 MB/s) - ‘porch.jpg’ saved [339081/339081]
```

#Problem 1: Mumford-Shah Piecewise Constant

In this problem, you will Implement a discrete piecewise constant variant of the Mumford-Shah model.

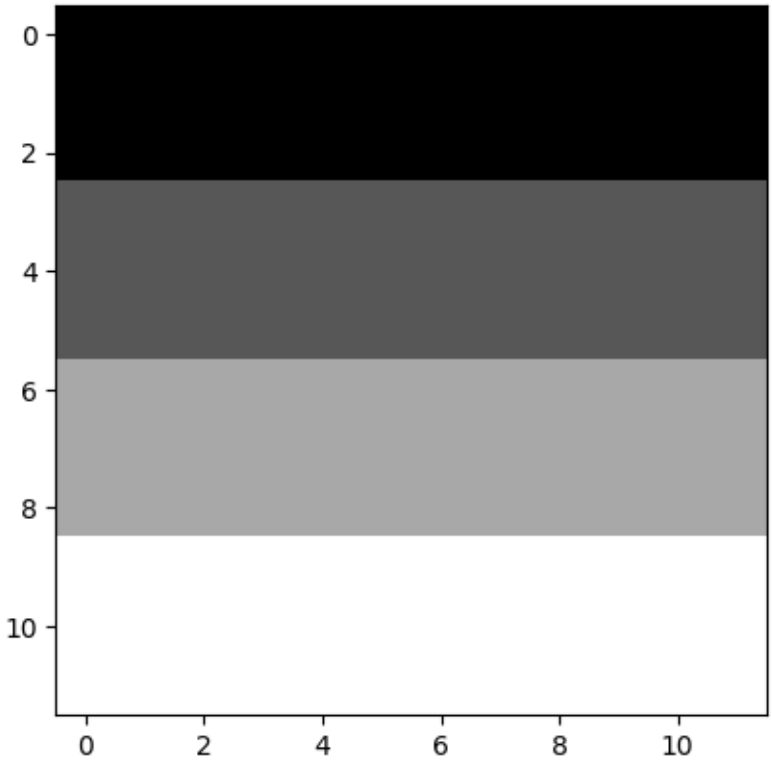
To do that you will first implement the discrete Potts model and test it on a pair of tiny images.

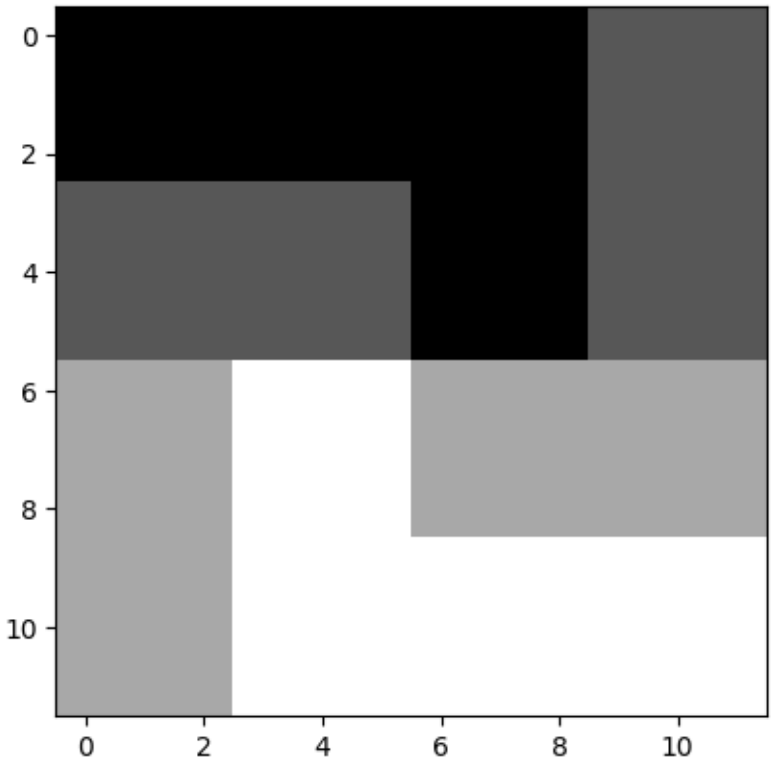
#Visualizing the images

You will use img1 and img2 to test your potts model implementation.

[3]:
```
img1 = plt.imread('img1.bmp')
plt.imshow(img1[:,:,0], cmap="gray")
plt.show()

img2 = plt.imread('img2.bmp')
plt.imshow(img2[:,:,0], cmap="gray")
plt.show()
```





#Implement Potts model

Implement the potts model function that we discussed when introducing the concept of image as functions. You could use loops for this. However, there is a very efficient way to do it too.

[4]:
```
def potts(img, beta=1):
  """function that takes an image and  the hyperparameter beta as inputs and
  return a scalar E representing the potts energy oif the image"""
  E = None
  # ================================================

  horiz_comp_1 = img[:, 1:]
  horiz_comp_2 = img[:, :-1]
  term_1 = np.sum(~np.isclose(horiz_comp_1, horiz_comp_2))

  vert_comp_1 = img[1:, :]
  vert_comp_2 = img[:-1, :]
  term_2 = np.sum(~np.isclose(vert_comp_1, vert_comp_2))

  E = beta * (term_1 + term_2)

  # ================================================
  return E
```

#Check your Potts model

You should get 0 error for both images.

[5]:
```
print("Error for energy of img1: ", 36 - potts(img1[:,:,0]))
print("Error for energy of img2: ",42 - potts(img2[:,:,0]))
```

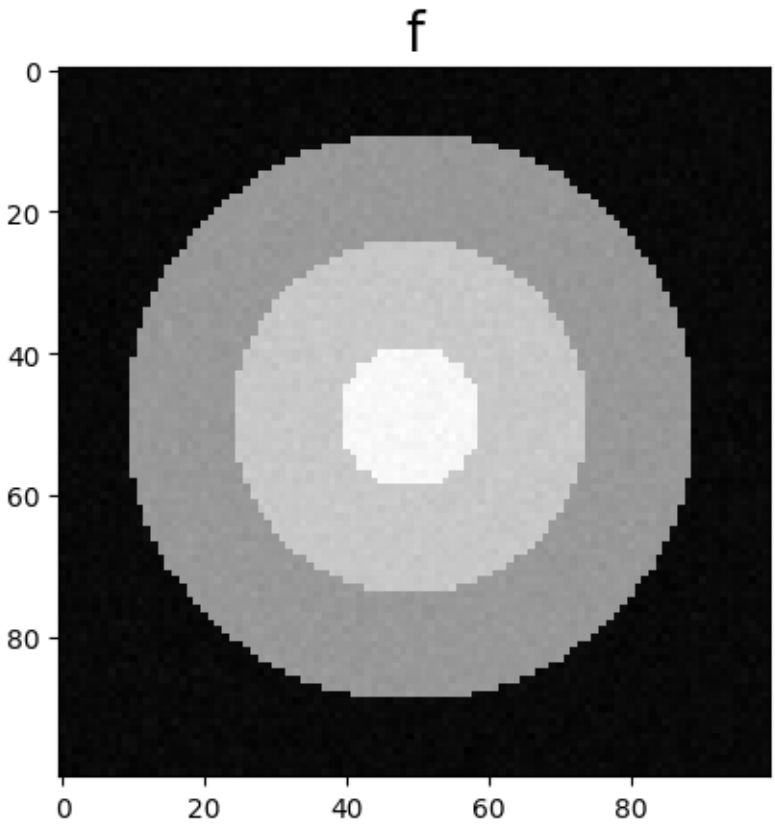
```
Error for energy of img1:  0
Error for energy of img2:  0
```

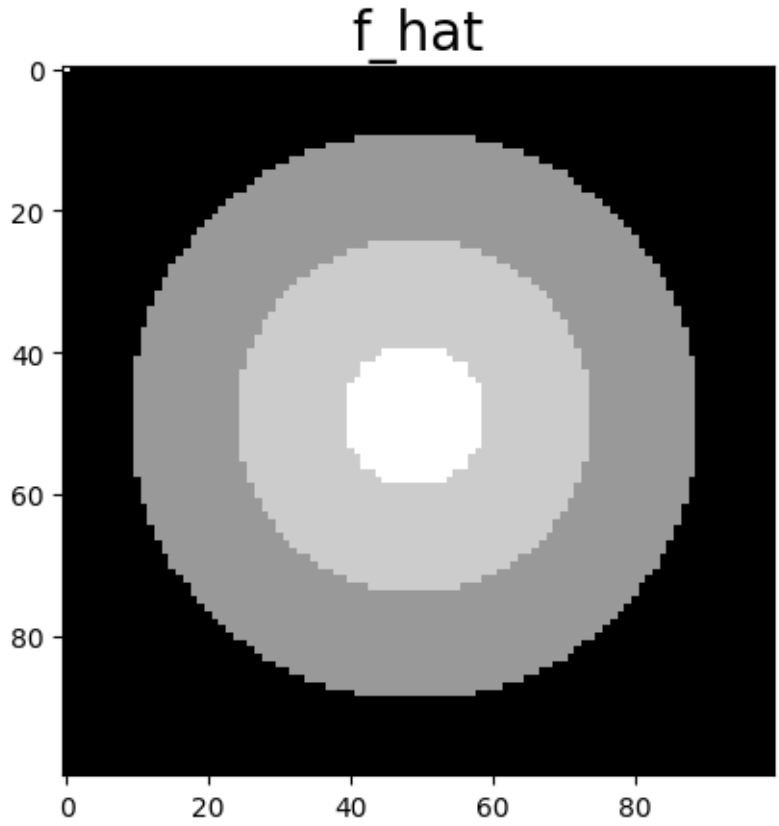
#Visualizing the images

You will use f and f_hat to test your implementation of the mumford_shah function. The image f is noisy and f_hat is a denoised version of f.

[6]:
```
f = plt.imread('f.png')
plt.title("f", fontsize=20)
plt.imshow(f, cmap="gray")
plt.show()

plt.title("f_hat", fontsize=20)
f_hat = plt.imread('fhat.png')
plt.imshow(f_hat,cmap="gray")
plt.show()
```





#Implement the Mumford Shah model

You will now implement the mumford shah function. We usually use this model in an optimization scenario where we try to find the image (f_hat) that minimizes the value of E_{PC}. In this case, we won't be actually minimizing anything. Instead, to test your implementation we are providing you with a denoised image (f_hat). This denoised image may or may not be the BEST denoised image (the image that minimizes the mumford_shah model).

[7]:
```
def mumford_shah(f_hat, f = f, alpha=1, gamma=0.1, beta=1):
  """ The function takes a denoise image (f_hat), the original image (f), an
  three hyperparameters (alpha, gamma, beta). The function returs a single value
  E_pc.

  Think about how this function uses the potts model that you implemented␣
↪before.
  Were is the beta parameter used?
  """
  E_pc = None
  # ============================================
```

```
    square_error = np.sum((f_hat - f)**2)
    E_pc = alpha * square_error + gamma * potts(f_hat, beta)

    # ============================================
    return E_pc
```

#Check your mumford shah model

This is a sanity check to help you debug your results.

[8]:
```
print("Error: ", 67.61031913757324 - mumford_shah(f_hat))
```

```
Error:  0.0
```

#Problem 2: Foreground-background graph-cut

In this problem you will implement a foreground-background segmentation method using the max-flow/min-cut method. You will implement the histogram feature representation and all aspects of taking the superpixel result and implementing the graph on top of it as well as reading out the results of the graph cut as a two-class segmentation.

#Visualizing the images

For this problem we will be working with an image of some vegetables, a porch and a flag.

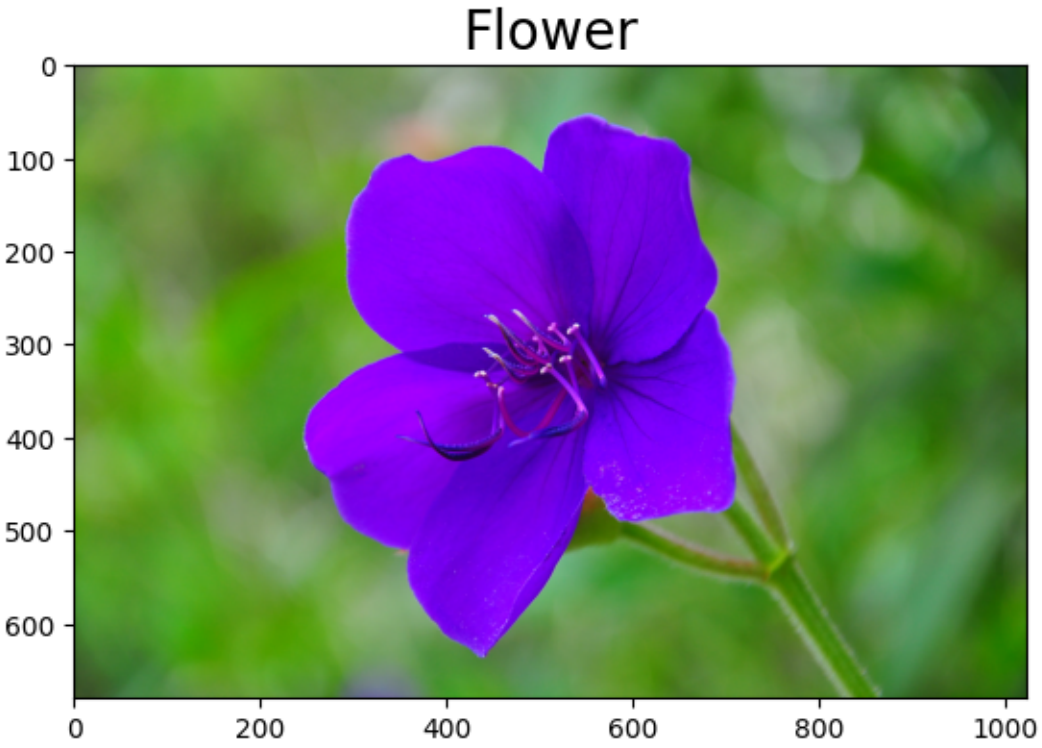
[9]:
```
veggies = plt.imread('veggies.jpg')
veggies = veggies[:,:,:3]
plt.title("Veggies", fontsize=20)
plt.imshow(veggies)
plt.show()

porch = plt.imread('porch.jpg')
porch = porch[:,:,:3]
plt.title("Porch", fontsize=20)
plt.imshow(porch)
plt.show()

flower = plt.imread('flower.jpg')
flower = flower[:,:,:3]
plt.title("Flower", fontsize=20)
plt.imshow(flower)
plt.show()
```


Veggies
SWEET YELLOW PEPPERS
$.99
SWEET RED PEPPERS
$.99
POUND
MIXED MINI POTATOES
$.99
0
50
100
150
200
250
300
0
100
200
300
400
500


Porch
0
50
100
150
200
250
300
350
400
0
100
200
300



4 Creating the superpixel maps

The code below will make superpixel maps of our three images. We use the skimage implementation of the SLIC superpixel algorithm. Here is a great medium article explaining how the algorithm works: https://medium.com/@darshita1405/superpixels-and-slic-6b2d8a6e4f08. This algorithm generates superpixels by clustering pixels based on their color similarity and proximity in the image plane. We provide a lot of functions for you. The output of the slic function is a label map (every segment has a differnt number assigned to it).

For visualization purposed we use the function label2rgb to convert this map to a 3-channel image where every label is assigned a color. We also use the function apply_supermap to assign the color of the cluster centers to every pixel of their corresponding segmentation groups.

[10]:
```
def label2rgb(labels):
  """
  Convert a labels image to an rgb image using a matplotlib colormap
  """
  label_range = np.linspace(0, 1, 256)
  lut = np.uint8(plt.cm.viridis(label_range)[:,2::-1]*256).reshape(256, 1, 3) #␣
↪replace viridis with a matplotlib colormap of your choice
  return cv2.LUT(cv2.merge((labels, labels, labels)), lut)
```

[11]:
```
def cluster_centers(superpixel_map):
  """
  This function takes a superpixel map and returns a list with the
  (row,col) positions of the cluster centers for that map
  """

  unique_labels = np.unique(superpixel_map)
  cluster_center_list = []

  for label_id, superpixel_label in enumerate(unique_labels):

      # Compute the coordinates where we have the superpixel_map = current label

      cluster_indices = np.where(superpixel_map == superpixel_label)

      # Compute the centroid of the coordinates to find the cluster centers

      centers = np.round(np.mean(cluster_indices, axis = 1)).astype('int')
      cluster_center_list.append(centers)

  return cluster_center_list
```

[12]:
```
def apply_supermap(img, superpixel_map):
  """ This function returns an image where we assign the color of the cluster␣
 ↪centers
  to every pixel of their corresponding segmentation groups."""
  centers = cluster_centers(superpixel_map)
  out = np.zeros_like(img)
  for i,(row, col) in enumerate(centers):
    out[superpixel_map == i] = img[row, col]
  return out
```

#Visualizing SLIC superpixel maps

[13]:
```
super_veggies = segmentation.slic(veggies, n_segments=144, compactness=20,␣
 ↪max_num_iter=20, start_label = 0) ###################### CHANGED
super_porch = segmentation.slic(porch, n_segments=144, compactness=20,␣
 ↪max_num_iter=20, start_label = 0) ###################### CHANGED
super_flower = skimage.segmentation.slic(flower, n_segments=144,␣
 ↪compactness=20, max_num_iter=20, start_label = 0) ######################␣
 ↪CHANGED

fig, ax = plt.subplots(3, 4, figsize=(25,25))
images = [veggies, porch, flower]
maps = [super_veggies, super_porch, super_flower]

for i,a in enumerate(ax):
```

```
  a[0].set_axis_off()
  a[0].set_title('Original', fontsize=20)
  a[0].imshow(images[i])
  a[1].set_axis_off()
  a[1].set_title('Superpixel map', fontsize=20)
  a[1].imshow(skimage.color.label2rgb(maps[i]))
  a[2].set_axis_off()
  a[2].set_title('Map overlayed on original', fontsize=20)
  a[2].imshow(skimage.color.label2rgb(maps[i], images[i]))
  a[3].set_axis_off()
  a[3].set_title('Color from cluster centers', fontsize=20)
  a[3].imshow(apply_supermap(images[i], maps[i]))
```

```
Output hidden; open in https://colab.research.google.com to view.
```

#Implement color histogram

You will implement the function color_histogram next. This function takes an image, a binary mask and the number of bins that you want to use for every channel. The function will compute the histogram over the 1 values in the mask only.

[14]:
```
def color_histogram(img, mask, num_bins):
  """For each channel in the image, compute a color histogram with the number␣
 ↪of bins
  given by num_bins of the pixels in
  image where the mask is true. Then, concatenate the vectors together into one␣
 ↪column vector (first
  channel at top).

  Mask is a matrix of booleans the same size as image.

  You MUST normalize the histogram of EACH CHANNEL so that it sums to 1.
  You CAN use the numpy.histogram function.
  You MAY loop over the channels.
  The output should be a 3*num_bins vector because we have a color image and
  you have a separate histogram per color channel.

  Hint: np.histogram(img[:,:,channel][mask], num_bins)"""

  rows, cols, channels = img.shape
  histogram = np.zeros(num_bins*3)

  # ============================================

  r_hist = np.histogram(img[:,:,0][mask], num_bins)[0]
  g_hist = np.histogram(img[:,:,1][mask], num_bins)[0]
  b_hist = np.histogram(img[:,:,2][mask], num_bins)[0]
```

```
  r_hist = r_hist / np.sum(r_hist)
  g_hist = g_hist / np.sum(g_hist)
  b_hist = b_hist / np.sum(b_hist)

  histogram[:num_bins] = r_hist
  histogram[num_bins:2*num_bins] = g_hist
  histogram[2*num_bins:] = b_hist

  # ================================================

  return histogram
```

5 Check histogram implementation

This will check if your implementation is correct. The error should be very close to 0 (~e-8)

[15]:
```
histogram = color_histogram(veggies, super_veggies==90, 10)
correct_histogram = np.array([
    0.01069855, 0.01510384, 0.00881057, 0.02580239, 0.16991819, 0.33480176,
    0.33920705, 0.07866583, 0.01258653, 0.00440529, 0.87413468, 0.07426054,
    0.0188798, 0.01195721, 0.0094399, 0.00566394, 0.00188798, 0.00251731,
    0.0, 0.00125865, 0.73127753, 0.20830711, 0.02139711, 0.01069855,
    0.01384519, 0.00566394, 0.00440529, 0.00314663, 0.0, 0.00125865
])

error = np.sum((correct_histogram - histogram)**2)
print("Error: ", error)
```

```
Error:  2.1084251864217788e-16
```

#Implement adjacency matrix

You need to implement the adjacency matrix function that takes a superpixel map as an input and outputs a binary adjacency matrix.

[16]:
```
def adjacencyMatrix(superpixel_map):
  """Implement the code to compute the adjacency matrix for the superpixel map
  The input is a superpixel map and the output is a binary adjacency matrix NxN
  (N being the number of superpixels in svMap).  Bmap has a 1 in cell i,j if
  superpixel i and j are neighbors. Otherwise, it has a 0.  Superpixels are␣
 ↪neighbors
  if any of their pixels are neighbors."""

  segmentList = np.unique(superpixel_map)
  segmentNum = len(segmentList)
  adjMatrix = np.zeros((segmentNum, segmentNum))

  # ================================================
```

```
  for i in range(segmentNum):
    for j in range(segmentNum):
        if i == j:
            continue

        mask_1 = superpixel_map == segmentList[i]
        mask_2 = superpixel_map == segmentList[j]

        overlap_1 = np.any(mask_1[:, :-1] & mask_2[:, 1:])
        overlap_2 = np.any(mask_1[:, 1:] & mask_2[:, :-1])
        overlap_3 = np.any(mask_1[:-1, :] & mask_2[1:, :])
        overlap_4 = np.any(mask_1[1:, :] & mask_2[:-1, :])
        has_overlap = overlap_1 or overlap_2 or overlap_3 or overlap_4

        if has_overlap:
            adjMatrix[i, j] = 1

  # ================================================

  return adjMatrix
```

#Visualize your implementation

The following code will show the adjacency matrices for the 4 images.

[17]:
```
fig, ax = plt.subplots(1, 3, figsize=(25,25))
images = [veggies, porch, flower]
maps = [super_veggies, super_porch, super_flower]

adjMatrix_veggies = adjacencyMatrix(super_veggies)
adjMatrix_porch = adjacencyMatrix(super_porch)
adjMatrix_flower = adjacencyMatrix(super_flower)



ax[0].set_axis_off()
ax[0].set_title('Veggies', fontsize=20)
ax[0].imshow(adjMatrix_veggies, cmap="gray")
ax[1].set_axis_off()
ax[1].set_title('Porch', fontsize=20)
ax[1].imshow(adjMatrix_porch, cmap="gray")
ax[2].set_axis_off()
ax[2].set_title('Flower', fontsize=20)
ax[2].imshow(adjMatrix_flower, cmap="gray")
```

[17]:
```
<matplotlib.image.AxesImage at 0x7d846997a780>
```

6 Calculate average node degree

You will implement a function that takes an adjacency matrix as input and outputs the average node degree. This is the average number of neighborghs that segments have.

[18]:
```
def average_node_degree(adjMatrix):
  """ This function takes an adjacency matrix and returns
  the average number of neighborghs that the segments have
  (average node degree)"""

  # ================================================

  average_node_degree = np.sum(adjMatrix) / adjMatrix.shape[0]

  # ================================================

  return average_node_degree
```

#Check your implementation

This will check that your adjacency maps and average_node_degree function works correctly. Errors should be very close to 0.

[19]:
```
print("Error on veggies: ", 5.221238938053097  -␣
  ↪average_node_degree(adjMatrix_veggies))
print("Error on porch: ", 5.196078431372549  -␣
  ↪average_node_degree(adjMatrix_porch))
print("Error on flower: ", 5.235294117647059 -␣
  ↪average_node_degree(adjMatrix_flower))
```

```
Error on veggies:  0.0
Error on porch:  0.0
Error on flower:  0.0
```

#Implement your graph-cut algorithm

It is time to build your foreground-background segmentation algorithm. For this we provide you with the implementation of the Ford-fulkerson algorithm which you will need to determine where your graph should be cut. You can learn more about the algorithm here: https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/. Please note that our version of the Ford Fulkerson algorithm doesn't return the max flow as a scalar. It returns the current flow through each edge of the graph when we reach the point of maximum flow.

We also implemented the reduce function which takes an image, its corresponding superpixel map, and a number of bins as input. The output is a list of feature vectors. Each feature vector is the resulting histogram from applying the color_histogram function you implemented to every segment on the superpixel map.

You will implement the graph_cut function

[20]:
```
# Python program for implementation of Ford Fulkerson algorithm
# The author of this code is Neelam Yadav

from collections import defaultdict

#This class represents a directed graph using adjacency matrix representation
class Graph:

    def __init__(self,graph):
        self.graph = graph # residual graph
        self. ROW = len(graph)
        # self.COL = len(gr[0])


    '''Returns true if there is a path from source 's' to sink 't' in
    residual graph. Also fills parent[] to store the path '''
    def BFS(self,s, t, parent):

        # Mark all the vertices as not visited
        visited =[False]*(self.ROW)

        # Create a queue for BFS
        queue=[]

        # Mark the source node as visited and enqueue it
        queue.append(s)
        visited[s] = True

         # Standard BFS Loop
        while queue:

            #Dequeue a vertex from queue and print it
            u = queue.pop(0)
```

```
            # Get all adjacent vertices of the dequeued vertex u
            # If a adjacent has not been visited, then mark it
            # visited and enqueue it
            for ind, val in enumerate(self.graph[u]):
                if visited[ind] == False and val > 0 :
                    queue.append(ind)
                    visited[ind] = True
                    parent[ind] = u

        # If we reached sink in BFS starting from source, then return
        # true, else false
        return True if visited[t] else False


    # Returns tne current flow from s to t in the given graph
    def FordFulkerson(self, source, sink):

        # This array is filled by BFS and to store path
        parent = [-1]*(self.ROW)

        max_flow = 0 # There is no flow initially
        current_flow = np.zeros_like(self.graph)

        # Augment the flow while there is path from source to sink
        while self.BFS(source, sink, parent) :
            # Find minimum residual capacity of the edges along the
            # path filled by BFS. Or we can say find the maximum flow
            # through the path found.
            path_flow = float("Inf")
            s = sink
            while(s !=  source):
                path_flow = min (path_flow, self.graph[parent[s]][s])
                s = parent[s]

            # Add path flow to overall flow
            max_flow +=  path_flow

            # update residual capacities of the edges and reverse edges
            # along the path
            v = sink
            while(v !=  source):
                u = parent[v]
                self.graph[u][v] -= path_flow
                self.graph[v][u] += path_flow
                current_flow[u][v] += path_flow
                current_flow[v][u] -= path_flow
                v = parent[v]
```

```
        return current_flow
```

[21]:
```
def reduce(img, superpixel_map, num_bins=10):
    """This function takes as input an image, its corresponding superpixel map,␣
    ↪and a
    number of bins as input. The output is a list of feature vectors.
    Each feature vector is the resulting histogram from applying the␣
    ↪color_histogram
    function you implemented to every segment on the superpixel map."""

    feature_vectors = []
    num_segments = len(np.unique(superpixel_map))
    for i in range(num_segments):
        mask = superpixel_map == i
        feature_vectors.append(color_histogram(img, mask, num_bins))
    return(feature_vectors)
```

[22]:
```
def graph_cut(superpixel_map, features, centers, keyindex):
    """Function to take a superpixel set and a keyindex and convert to a
    foreground/background segmentation.

    keyindex is the index to the superpixel segment we wish to use as foreground␣
    ↪and
    find its relevant neighbors.

    centers is a list of tuples (row, col) with the positions of the cluster␣
    ↪centers
    of the superpixel_map

    features is a list of histograms (obtained from the reduce function) for␣
    ↪every superpixel
    segment in an image.

    """

    #Compute basic adjacency information of superpixels
    #Note that adjacencyMatrix is code you need to implement

    # ================================================
    # TODO: this should be one line of code

    adjMatrix = adjacencyMatrix(superpixel_map)

    # ================================================
```

```
 # normalization for distance calculation based on the image size
 # for points (x1,y1) and (x2,y2), distance is
 # exp(-||(x1,y1)-(x2,y2)||^2/dnorm)
 dnorm = 2*(superpixel_map.shape[0]/2 *superpixel_map.shape[1] /2)**2
 k = len(features) #number of superpixels in image

 #Generate capacity matrix
 capacity = np.zeros((k+2,k+2))
 source = k
 sink = k+1

 # This is a single planar graph with an extra source and sink
 #  Capacity of a present edge in the graph is to be defined as the product of
 #  1:  the histogram similarity between the two color histogram feature␣
↪vectors.
 #  The similarity between histograms should be computed as the intersections␣
↪between
 #  the histograms. i.e: sum(min(histogram 1, histogram 2))
 #  2:  the spatial proximity between the two superpixels connected by the␣
↪edge.
 #      use exp(-||(x1,y1)-(x2,y2)||^2/dnorm)
 #
 #  Source gets connected to every node except sink
 #  Capacity is with respect to the keyindex superpixel
 #  Sink gets connected to every node except source and its capacity is␣
↪opposite
 # The weight between a pixel and the sink is going to be the max of all the␣
↪weights between
 # the source and the image pixels minus the weight between that specific␣
↪pixel and the source.
 # Other superpixels get connected to each other based on computed adjacency
 # matrix: the capacity is defined as above, EXCEPT THAT YOU ALSO NEED TO␣
↪MULTIPLY BY A SCALAR 0.25 for
 # adjacent superpixels.


 key_features = features[keyindex] # color histogram representation of␣
↪superpixel # keyindex
 key_x = centers[keyindex][1] # row of cluster center for superpixel # keyindex
 key_y =  centers[keyindex][0] # col of cluster center for superpixel #␣
↪keyindex

 # ================================================

 for i in range(k):
```

```
  for j in range(i+1, k):
      if adjMatrix[i, j] == 0:
          continue

      feature_1 = features[i]
      feature_2 = features[j]
      intersection = np.sum(np.minimum(feature_1, feature_2))

      center_1 = np.array([centers[i][1], centers[i][0]])
      center_2 = np.array([centers[j][1], centers[j][0]])
      distance = np.exp(-np.linalg.norm(center_1 - center_2)**2 / dnorm)

      capacity[i, j] = 0.25 * intersection * distance
      # multiplier = 0.25 if adjMatrix[i, j] == 1 else 1
      # capacity[i, j] = multiplier * intersection * distance
      capacity[j, i] = capacity[i, j]

# Compute the capacity between every node and the source/sink
for i in range(k):
  node_feature = features[i]
  node_x = centers[i][1]
  node_y = centers[i][0]

  node_center = np.array([node_x, node_y])
  source_center = np.array([key_x, key_y])

  intersection = np.sum(np.minimum(key_features, node_feature))
  distance = np.exp(-np.linalg.norm(node_center - source_center)**2 / dnorm)
  # capacity[source, i] = 0.25 * intersection * distance
  capacity[source, i] = 1 * intersection * distance
  # capacity[i, source] = capacity[source, i]

max_source_capacity = np.max(capacity[source])
for i in range(k):
  capacity[i, sink] = max_source_capacity - capacity[source, i]
  # capacity[sink, i] = capacity[i, sink]

# ================================================

# Obtaining the current flow of the graph when the flow is max
capacity = (1e6*capacity).astype('int') ###################### CHANGED
g = Graph(capacity.copy())
current_flow = g.FordFulkerson(source, sink)

# Extract the two-class segmentation.
# the cut will separate all nodes into those connected to the
# source and those connected to the sink.
```

```
  # The current_flow matrix contains the necessary information about
  # the max-flow through the graph.

  segment_map = np.zeros_like(superpixel_map)
  rem_capacity = capacity - current_flow

  # ================================================
  # TODO: Do the segmentation and fill segmentation map with 1s where the␣
 ↪foreground is.
  # Replace pass with your code

  seen_nodes = set()
  frontier = set()
  frontier.add(source)
  while len(frontier):
    cur_node = frontier.pop()
    if cur_node in seen_nodes:
      continue
    seen_nodes.add(cur_node)

    for i in range(k):
      if rem_capacity[cur_node, i] > 0:
        frontier.add(i)

  for segment_index in seen_nodes:
    if segment_index == source or segment_index == sink:
      continue
    mask = superpixel_map == segment_index
    segment_map[mask] = 1


  # ================================================

  return capacity, segment_map
```

#Visualize capacity matrix

[23]:
```
veggies_features = reduce(veggies, super_veggies)
veggies_centers = cluster_centers(super_veggies)
veggies_capacity, veggies_segment_map = graph_cut(super_veggies,␣
 ↪veggies_features, veggies_centers, 56) ###################### CHANGED

porch_features = reduce(porch, super_porch)
porch_centers = cluster_centers(super_porch)
porch_capacity,porch_segment_map = graph_cut(super_porch, porch_features,␣
 ↪porch_centers, 24)
```

```
flower_features = reduce(flower, super_flower)
flower_centers = cluster_centers(super_flower)
flower_capacity,flower_segment_map = graph_cut(super_flower, flower_features,␣
 ↪flower_centers, 80)


fig, ax = plt.subplots(1, 3, figsize=(25,25))
ax[0].set_axis_off()
ax[0].set_title('Veggies', fontsize=20)
ax[0].imshow(veggies_capacity, cmap="gray")
ax[1].set_axis_off()
ax[1].set_title('Porch', fontsize=20)
ax[1].imshow(porch_capacity, cmap="gray")
ax[2].set_axis_off()
ax[2].set_title('Flower', fontsize=20)
ax[2].imshow(flower_capacity, cmap="gray")
ax[2].set_axis_off()
```

#Visualizing graphcut segmentation

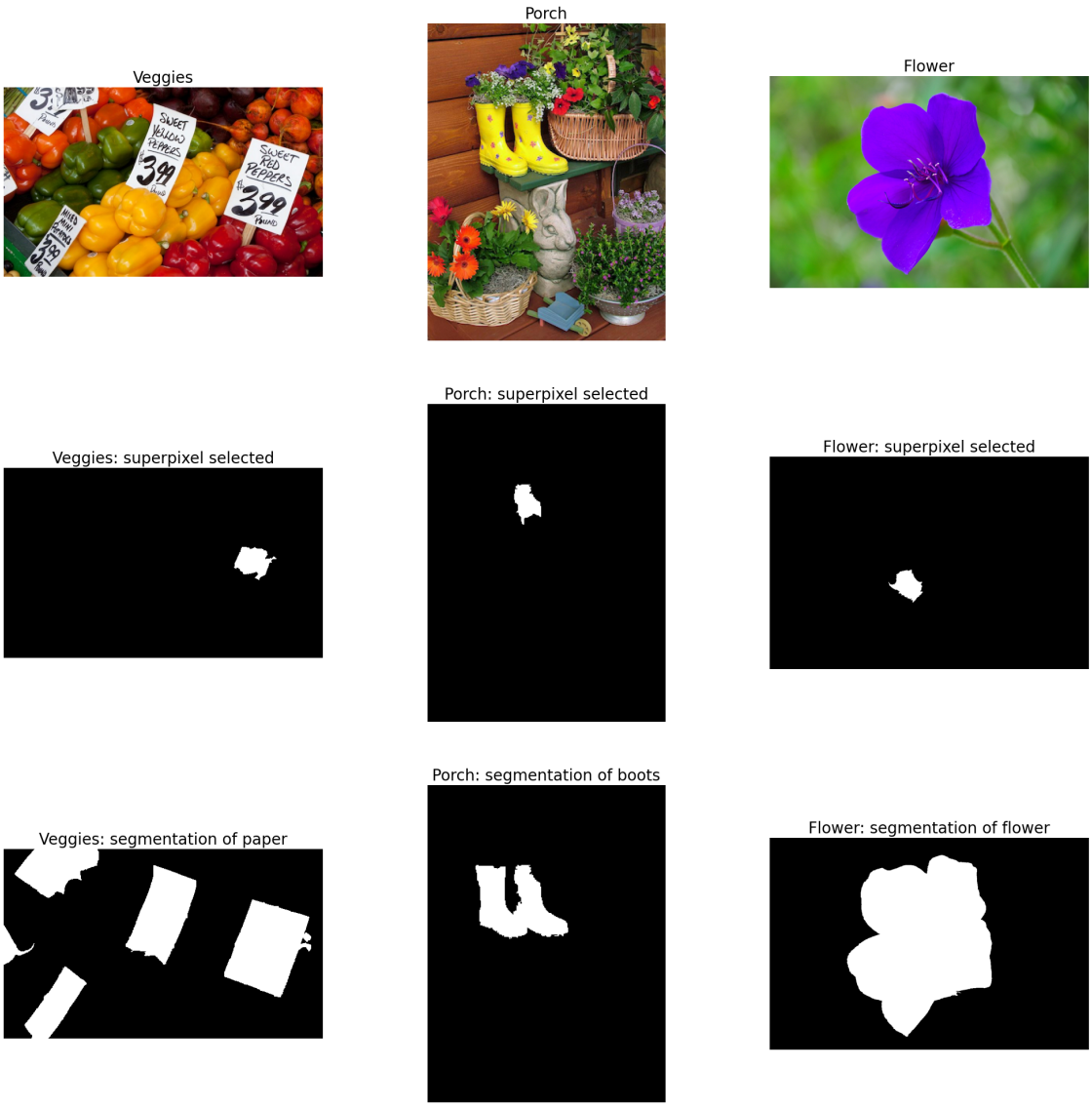
[24]:
```
fig, ax = plt.subplots(3, 3, figsize=(25,25))

ax[0][0].set_axis_off()
ax[0][0].set_title('Veggies', fontsize=20)
ax[0][0].imshow(veggies, cmap="gray")
ax[0][1].set_axis_off()
ax[0][1].set_title('Porch', fontsize=20)
ax[0][1].imshow(porch, cmap="gray")
ax[0][2].set_axis_off()
ax[0][2].set_title('Flower', fontsize=20)
ax[0][2].imshow(flower, cmap="gray")
ax[0][2].set_axis_off()

ax[1][0].set_axis_off()
```

```
ax[1][0].set_title('Veggies: superpixel selected', fontsize=20)
ax[1][0].imshow(super_veggies==56, cmap="gray")       ######################␣
 ↪CHANGED
ax[1][1].set_axis_off()
ax[1][1].set_title('Porch: superpixel selected', fontsize=20)
ax[1][1].imshow(super_porch==24, cmap="gray")
ax[1][2].set_axis_off()
ax[1][2].set_title('Flower: superpixel selected', fontsize=20)
ax[1][2].imshow(super_flower==80, cmap="gray")
ax[1][2].set_axis_off()

ax[2][0].set_axis_off()
ax[2][0].set_title('Veggies: segmentation of paper', fontsize=20)
ax[2][0].imshow(veggies_segment_map, cmap="gray")
ax[2][1].set_axis_off()
ax[2][1].set_title('Porch: segmentation of boots', fontsize=20)
ax[2][1].imshow(porch_segment_map, cmap="gray")
ax[2][2].set_axis_off()
ax[2][2].set_title('Flower: segmentation of flower', fontsize=20)
ax[2][2].imshow(flower_segment_map, cmap="gray")
ax[2][2].set_axis_off()
```



#Problem 3: Segmentation with minimum spanning forest

For this problem you are going to do segmentation using the Felzenszwalb-Huttenlocher algorithm (http://vision.stanford.edu/teaching/cs231b_spring1415/slides/ranjay_pres.pdf). This algorithm will segment into multiple groups as opposed to the graph-cut algorithm you implemented which will only did foreground-background segmentation.

The first step will be to implement the function compute_MSF_edges.

[25]:
```
def compute_MSF_edges(superpixel_map, adjMatrix, centers, features):
  """ For this function you will turn the adjacency matrix into a  mx3 matrix,
  of which each row is an edge (node1, node2, weights). You will return this␣
 ↪matrix.
```

```
    m is the number of edges. You should avoid duplicated edges. You will only use
    the edges where the adjacency matrix is 1 (between neighbors)

    The edge weight is defined as the product of (1) difference of histograms
    (or equivalently, 1-intersection of histograms/3), and (2) inverse
    proximity exp(D(a,b)) with D specified below.
    Notice the difference between the weights here and those in graph cuts.

    Normalization for distance calculation based on the image size
    For points (x1,y1) and (x2,y2), normalized distance D =
    ||(x1,y1)-(x2,y2)||^2/dnorm.

    hint: np.triu, np.where (you don't necessarily need to use these)"""


    rows, cols = superpixel_map.shape
    dnorm = np.sum(rows**2 + cols**2)
    n, _  = adjMatrix.shape

    G = None

    # ============================================
    # TODO: Replace pass with your code

    node1, node2 = np.where(np.triu(adjMatrix) == 1)

    edges = []
    for i, j in zip(node1, node2):
      feature_1 = features[i]
      feature_2 = features[j]
      intersection = np.sum(np.minimum(feature_1, feature_2))
      difference = 1 - intersection / 3

      center_1 = np.array([centers[i][1], centers[i][0]])
      center_2 = np.array([centers[j][1], centers[j][0]])
      D = np.exp(np.linalg.norm(center_1 - center_2)**2 / dnorm)

      edges.append([i, j, difference * D])

    G = np.array(edges)

    # ============================================

    return G
```

Now, you need to implement the minimum random forest segmentation (MSF) function below. We recommend that you implement MSF before filter_segments, even though you will have to run

filter_segments first so that MSF works.

YOU SHOULD IMPLEMENT FILTER_SEGMENTS AFTER YOU IMPLEMENT MSF

[26]:
```
def filter_segments(edges, seg_id, min_size):
  """A common postprocessing step used with F-H is to merge all of the small,␣
 ↪unnecessary components that result from uneven
  regions. This function should 1) iterate through the edges, in
  ascending order, and 2) for each edge, if it connects two distinct segments,␣
 ↪AND if at least one of those segments contains less
  than min_size pixels in it, merge the segments.

  You will change seg_id and return it. This function will be used in MSF.
  """

  num_edges = edges.shape[0]

  # ================================================
  # TODO: Replace pass with your code

  seg_sizes = np.zeros_like(seg_id)
  for i in range(len(seg_id)):
    seg = seg_id[i]
    seg_sizes[seg] = np.sum(seg_id == seg)

  for i in range(num_edges):
    node_1, node_2, _ = edges[i]
    node_1 = int(node_1)
    node_2 = int(node_2)
    segment_1 = seg_id[node_1]
    segment_2 = seg_id[node_2]
    if segment_1 != segment_2:
        if seg_sizes[segment_1] < min_size or seg_sizes[segment_2] < min_size:
            # Then merge by assigning all
            seg_sizes[segment_1] += seg_sizes[segment_2]
            seg_sizes[segment_2] = 0
            seg_id[seg_id == segment_2] = segment_1

  # ================================================

  return seg_id
```

[27]:
```
def MSF(edges, k, filter_min_size=None):
  """Felzenszwalb-Huttenlocher algorithm implementation, which is a modified
  version of the Kruskal algorithm.
  Input  edges: mx3 matrix, the adjacency list of a graph, of which each row is
            an edge (node1, node2, weights).
         m: is the number of edges.
```

```
        k: hyperparameter for F-H algorithm
        filter_size: if a filter size is passed, this will remove groups from␣
↪the
        segmentation that have less than filter_size superpixels
 Output seg: array of size (n,), the segment id assigned to each node in the␣
↪graph
             n is the number of nodes"""

 num_edges = edges.shape[0]
 num_nodes = int(np.max(edges[:,:2])) +1

 # Sort edges in ascending order
 edges = edges[np.argsort(edges[:,2]), :]

 # You will assign every pixel to the id corresponding to a segment id.␣
↪Initially,
 # every superpixel is in its own segment.
 seg_id = np.array(list(range(num_nodes)))


 # ================================================
 # TODO: Replace pass with your code

 # Initialize structures for F-H algorithm
 size = np.ones(num_nodes, dtype=int)        # Size of each segment
 internal_diff = np.zeros(num_nodes, dtype=float) # Internal difference (max␣
↪edge weight in segment)

 # Helper function: Find with Path Compression
 def find(i):
     root = i
     # Find the root
     while seg_id[root] != root:
         root = seg_id[root]
     # Path compression: make nodes point directly to the root
     curr = i
     while curr != root:
         nxt = seg_id[curr]
         seg_id[curr] = root
         curr = nxt
     return root

 # Loop through sorted edges
 for i in range(num_edges):
     u = int(edges[i, 0])
     v = int(edges[i, 1])
     weight = edges[i, 2]
```

```
    # Find roots of both nodes
    root_u = find(u)
    root_v = find(v)

    # If they are in different segments, check if we should merge
    if root_u != root_v:
        # Calculate thresholds for both segments
        tau_u = k / size[root_u]
        tau_v = k / size[root_v]

        # F-H Merge Condition
        if weight <= min(internal_diff[root_u] + tau_u, internal_diff[root_v] + tau_v):
            # Union: Attach root_v to root_u
            seg_id[root_v] = root_u

            # Update the size of the new merged segment
            size[root_u] += size[root_v]

            # Update the internal difference (since edges are sorted,
            # this edge is guaranteed to be the maximum weight in the new segment)
            internal_diff[root_u] = weight

# Final pass: Flatten the tree so every node in seg_id points directly to its final root
# This makes post-processing much easier
for i in range(num_nodes):
    seg_id[i] = find(i)

# ================================================

if filter:
  # Postprocessing to remove fragments
  seg_id = filter_segments(edges, seg_id, filter_min_size)


return seg_id
```

#Visualize MSF segmentations

This will show you your segmentations with and without filtering.

[28]:
```
def build_msf_map(segmentation, superpixel_map):

  num_nodes = len(segmentation)
  map = np.zeros_like(superpixel_map)
```

```
  rows, cols = map.shape
  for row in range(rows):
    for col in range(cols):
      map[row, col] = segmentation[superpixel_map[row, col]]
  return map
```

[29]:
```
edges_veggies = compute_MSF_edges(super_veggies, adjMatrix_veggies,␣
 ↪veggies_centers, veggies_features)
edges_porch = compute_MSF_edges(super_porch, adjMatrix_porch, porch_centers,␣
 ↪porch_features)
edges_flower = compute_MSF_edges(super_flower, adjMatrix_flower,␣
 ↪flower_centers, flower_features)

segmentation_veggies = MSF(edges_veggies,1, 0)
segmentation_porch = MSF(edges_porch,1, 0)
segmentation_flower = MSF(edges_flower,4, 0)

veggies_map = build_msf_map(segmentation_veggies, super_veggies)
porch_map = build_msf_map(segmentation_porch, super_porch)
flower_map = build_msf_map(segmentation_flower, super_flower)

segmentation_veggies = MSF(edges_veggies,1, 5)
segmentation_porch = MSF(edges_porch,1, 3)
segmentation_flower = MSF(edges_flower,4, 15)

veggies_map_filtered = build_msf_map(segmentation_veggies, super_veggies)
porch_map_filtered = build_msf_map(segmentation_porch, super_porch)
flower_map_filtered = build_msf_map(segmentation_flower, super_flower)


fig, ax = plt.subplots(3, 3, figsize=(25,25))

ax[0][0].set_axis_off()
ax[0][0].set_title('Veggies', fontsize=20)
ax[0][0].imshow(veggies)
ax[0][1].set_axis_off()
ax[0][1].set_title('Porch', fontsize=20)
ax[0][1].imshow(porch)
ax[0][2].set_axis_off()
ax[0][2].set_title('Flower', fontsize=20)
ax[0][2].imshow(flower)
ax[0][2].set_axis_off()

ax[1][0].set_axis_off()
ax[1][0].set_title('Veggies: segmentation', fontsize=20)
ax[1][0].imshow(veggies_map)
ax[1][1].set_axis_off()
```

```
ax[1][1].set_title('Porch: segmentation', fontsize=20)
ax[1][1].imshow(porch_map)
ax[1][2].set_axis_off()
ax[1][2].set_title('Flower: segmentation', fontsize=20)
ax[1][2].imshow(flower_map)
ax[1][2].set_axis_off()

ax[2][0].set_axis_off()
ax[2][0].set_title('Veggies: segmentation filtered', fontsize=20)
ax[2][0].imshow(veggies_map_filtered)
ax[2][1].set_axis_off()
ax[2][1].set_title('Porch: segmentation filtered', fontsize=20)
ax[2][1].imshow(porch_map_filtered)
ax[2][2].set_axis_off()
ax[2][2].set_title('Flower: segmentation filtered', fontsize=20)
ax[2][2].imshow(flower_map_filtered)
ax[2][2].set_axis_off()
```

[35]:
```
# %%capture

!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive is already the newest version (2021.20220204-1).
texlive-latex-extra is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 141 not upgraded.
```

```
Requirement already satisfied: pypandoc in /usr/local/lib/python3.12/dist-
packages (1.17)
```

[40]:
```
# generate pdf
# Please provide the full path of the notebook file below
# Important: make sure that your file name does not contain spaces!
import os
# Give the path to your notebook here.
# Find the location of your notebook using File->Locate in Drive
# Example: notebookpath = '/content/drive/My Drive/Colab Notebooks/
 ↪EECS_442_PS2_FA_2022_Starter_Code.ipynb'

notebookpath = "/content/drive/MyDrive/courses/EECS_504_PS3_adempst_61259596.
 ↪ipynb"
drive_mount_point = '/content/drive/'
from google.colab import drive
drive.mount(drive_mount_point)
file_name = notebookpath.split('/')[-1]
get_ipython().system("apt update && apt install texlive-xetex␣
 ↪texlive-fonts-recommended texlive-generic-recommended")
get_ipython().system("jupyter nbconvert --to PDF {}".format(notebookpath.
 ↪replace(' ', '\\ ')))
from google.colab import files
files.download(notebookpath.split('.')[0]+'.pdf')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call
drive.mount("/content/drive/", force_remount=True).
[NbConvertApp] Converting notebook
/content/drive/MyDrive/courses/EECS_504_PS3_adempst_61259596.ipynb to PDF
[NbConvertApp] Support files will be in EECS_504_PS3_adempst_61259596_files/
[NbConvertApp] Making directory ./EECS_504_PS3_adempst_61259596_files
[NbConvertApp] Writing 141344 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 3453888 bytes to
/content/drive/MyDrive/courses/EECS_504_PS3_adempst_61259596.pdf
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[ ]: