

Assignment 4

Part I: K-means Segmentation

NOTE: First several "theoretical" problems are independent of the programming part (problem 4). If you prefer, you can start working on Problem 4.

Problem 0 (*probability simplex* and distribution entropy)

The general concept of "entropy" is very important in science (statistics, physics, computer vision, ML, AI, information theory, data analysis, etc). The general formula for the entropy is

$$H(\mathbf{S}) = - \sum_k S^k \ln S^k$$

where

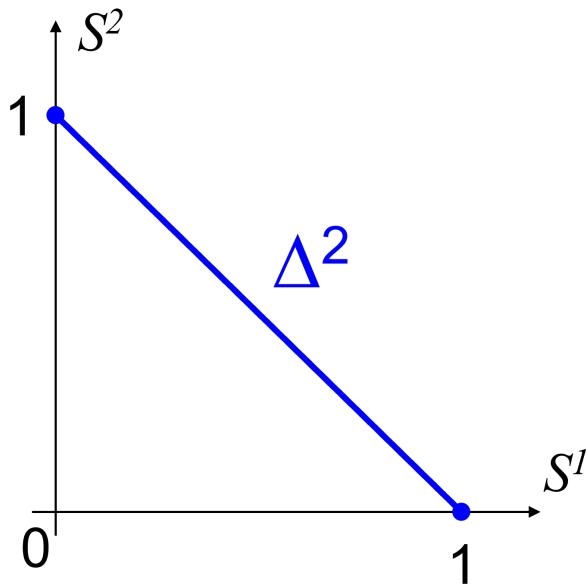
$$\mathbf{S} := (S^1, \dots, S^K) \in \Delta^K$$

is any distribution over K values (e.g. classes, categories, decisions, etc) and

$\Delta^K := \{p \in \mathcal{R}^K \mid p^k \geq 0, \sum_{k=1}^K p^k = 1\} \subset \mathcal{R}^K$ is a so-called *probability simplex*. Probably the most basic property of entropy one should know is that it measures "randomness" of a distribution.

(part a)

In this exercise you should visualize the entropy function $H : \Delta^K \rightarrow \mathcal{R}^1$ for $K = 2$. In this simple case the distribution $\mathbf{S} = (S^1, S^2)$ may correspond to some binary random variable X so that $S^1 = \Pr(X = 1)$ and $S^2 = \Pr(X = 0)$. For example, X could represent a binary decision about the category of an object observed in an image (person or not-a-person). Since $S^1 + S^2 = 1$, probability simplex Δ^2 has only one degree of freedom - one scalar is enough to represent an arbitrary binary distribution. It is easy to visualize the entropy function over all possible binary distributions $\mathbf{S} = (S^1, S^2) \in \Delta^2$ as probability simplex $\Delta^2 = \{(x, 1 - x) \mid 0 \leq x \leq 1\}$ is a line interval inside \mathcal{R}^2 .



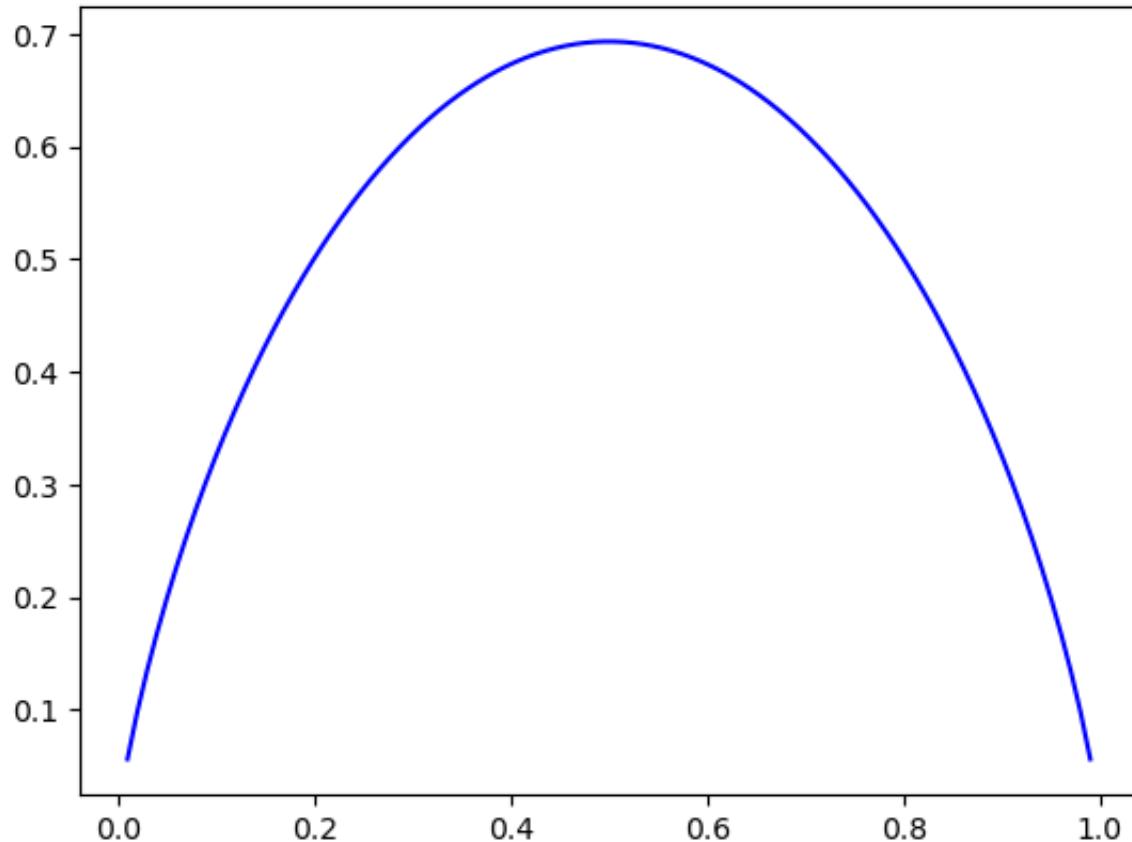
Visualize the entropy function $H(\mathbf{S})$ for $K = 2$ as follows. Derive the expression for function $H(x) := H(\mathbf{S})$ for $\mathbf{S} = (x, 1 - x)$ and use *matplotlib* to plot $H(x)$ for $x \in [0, 1]$ in the code cell below. State which binary distribution(s) $\mathbf{S} = (S^1, S^2)$ have the lowest and the largest entropy values. Informally relate your observations to "randomness" of these distributions.

Solution:

$$H(x) = -x \ln(x) - (1-x) \ln(1-x)$$

```
In [1]: # Write code generating 2D plot of the entropy H(x) for x in [0,1]
# HINT: you might need to be carefull with potential numerical issues at x=0
# but often it works fine without doing anything special.
%matplotlib notebook
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0.01,0.99,100)
y = -x*np.log(x) - (1-x)*np.log(1-x)
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
plt.plot(x,y, 'b')
plt.show()
```

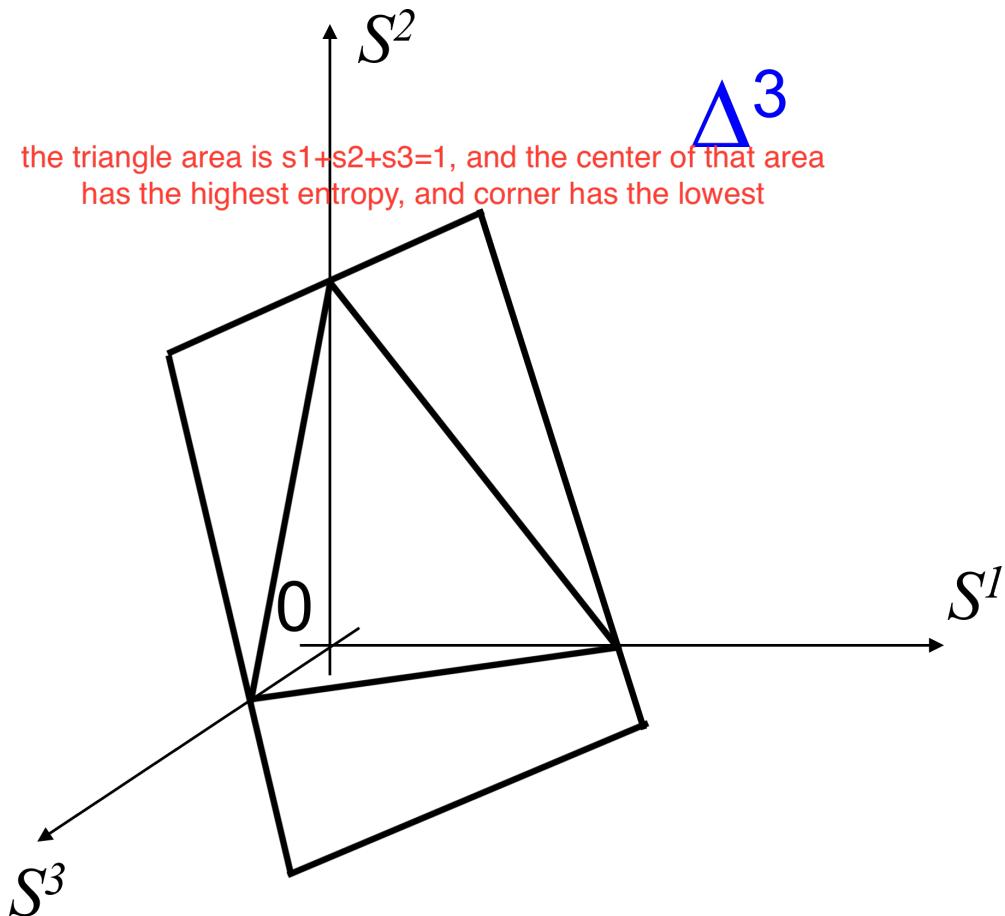


The lowest entropy value is 0 and it corresponds to $x = 0$ and $x = 1$

The largest entropy value is 0.693 and it corresponds to $x = 0.5$

(part b)

In the previous part you saw a visualization of the probability simplex Δ^2 as a subset of \mathcal{R}^2 . Using any picture editor (e.g. "paint"), edit file *images/simplex3.png* (shown below) to draw your best impression of the probability simplex $\Delta^3 \subset \mathcal{R}^3$ for distributions $S = (S^1, S^2, S^3)$ over $K = 3$ possible values/classes/categories. Your drawing should visualize simplex Δ^3 as a subset of \mathcal{R}^3 .



In [2]:

```
# BONUS: one can also visualize the entropy function over the probability si
# NOTE: There are a number of different ways to do this visualizatin, but it
#       Do this bonus excercise only if you have time.
```

Problem 1 (soft-max)

Following the material on slide 54 in topic 9, derive the optimal "soft" clustering (distribution over clusters) at point p

$$\mathbf{S}_p = \{S_p^k \mid 1 \leq k \leq K, S_p^k \geq 0, \sum_k S_p^k = 1\}$$

that Lloyd's algorithm would obtain when re-estimating segmentation for fixed cluster models parameters μ_k . Note that the total K-means objective $E(\mathbf{S}, \mu)$ as a function of segmentation \mathbf{S} (when μ is fixed) is a sum of independent terms for every pixel. When computing optimal distribution \mathbf{S}_p it is enough to focus on the terms dependent only on its components S_p^k . For the K-means formulation on slide 54, these terms are

$$-\sum_{k=1}^K S_p^k a_p^k - T H(\mathbf{S}_p) \quad (*)$$

where constant $a_p^k := \log P(f_p | \mu_k)$ (assuming fixed μ) corresponds to k -th cluster's log-likelihood at the observed feature point f_p , constant T represents a so-called "temperature" parameter, and $H(\mathbf{S}_p) := -\sum_k S_p^k \log S_p^k$ is the entropy of \mathbf{S}_p .

Use your solution to show what happens with the optimal distribution \mathbf{S}_p when the temperature parameter reduces to zero $T \rightarrow 0$.

HINT 1: Optimization of $(*)$ should be done over variable \mathbf{S}_p representing a probability distribution. Thus, constraint $\sum_k S_p^k = 1$ should be respected. You should use the standard general **Lagrangian approach** that converts constrained optimization into unconstrained one. In particular, you can combine objective function $(*)$ with the constraint $\sum_k S_p^k = 1$ into the *Lagrangian*:

$$L(\mathbf{S}_p, \lambda) = - \sum_{k=1}^K S_p^k a_p^k - T H(\mathbf{S}_p) + \lambda \left(\sum_{k=1}^K S_p^k - 1 \right)$$

that includes one extra optimization variable λ , the so called *Lagrange multiplier*. The solution \mathbf{S}_p for the original constrained optimization problem follows directly from the solution $\{\mathbf{S}_p, \lambda\}$ that minimizes the Lagrangian.

HINT 2: Similarly to optimization of single-variate functions, you can find extrema points for the multi-variate Lagrangian $(**)$ by finding values of variables $(S_p^1, \dots, S_p^K, \lambda)$ where its derivative (gradient) equals zero. That is, the whole problem boils down to solving the system of $K + 1$ equations $\nabla L = \mathbf{0}$ for the Lagrangian in $(**)$.

HINT 3: The goal of this exercise is to see how adding the entropy affects a linear loss. Optimization of $(*)$ over distributions \mathbf{S}_p should result in the, so-called, **soft-max operator** applicable to arbitrary K potentials $\{a_p^k \mid 1 \leq k \leq K\}$.

Problem 2 (Mahalanobis distance, decorrelation, etc.)

Let $X \in R^N$ be a Gaussian random vector with given mean μ and covariance matrix Σ . Find $N \times N$ matrix A such that

linear transformation $Y = AX$ gives a random vector Y with covariance $\Sigma_Y = \mathbf{I}$. That is, the components of the transformed random vector Y should be i.i.d. You should derive an equation for matrix A assuming as given eigen-decomposition of the covariance matrix $\Sigma = U\Lambda U^T$ where $\Lambda = \text{diag}(s_1, \dots, s_n)$ is a diagonal matrix of (non-negative!) eigen-values and U is an orthogonal $N \times N$ matrix (its columns are unit eigen-vectors of Σ).

HINT: you should solve the following (equivalent) simple geometric problem on "linear warps" (linear domain transforms): find a linear transformation A of points in R^N such that Mahalanobis distances (see slide 57) between any two given vectors $X, \mu \in R^N$ are equivalent to Euclidean distances between the corresponding vectors $Y = AX$ and $m = A\mu$ in the transformed space, that is,

$$\|X - \mu\|_{\Sigma}^2 = \|Y - m\|^2.$$

The proof should be simple (just a couple of lines) if you use linear algebraic expressions for two squared metrics above and the given eigen decomposition of matrix Σ .

$$\begin{aligned}\|X - \mu\|_{\Sigma}^2 &:= (X - \mu)^T \Sigma^{-1} (X - \mu) = (X - \mu)^T (U^T)^{-1} \Lambda^{-1} U^{-1} (X - \mu) \\ \|Y - m\|^2 &= \|AX - A\mu\|^2\end{aligned}$$

INTERPRETATION 1: reading the geometric result in reverse shows that linear transformation "distort" Euclidean distances into Mahalanobis distances.

INTERPRETATION 2 (Euclidean embedding): a space with Mahalanobis metric can be isometrically embedded in a Euclidean space. This is a trivial spacial case of the **Nash theorem** on existence of Euclidean embeddings of more general (Riemannian) metric spaces.

Problem 3

Show algebraic equivalence between two non-parametric formulations for K-means (objectives $E(S)$ at the bottom of slide 72, Topic 9):

$$\sum_{k=1}^K \frac{\sum_{pq \in S^k} \|f_p - f_q\|^2}{2 |S^k|} = const - \sum_{k=1}^K \frac{\sum_{pq \in S^k} \langle f_p, f_q \rangle}{|S^k|}$$

Solution:

Focus on fixed k , then,

$$\sum_{pq \in S^k} \|f_p - f_q\|^2 = \sum_{pq \in S^k} \|f_p\|^2 + \|f_q\|^2 - 2\langle f_p, f_q \rangle = \sum_{pq \in S^k} \|f_p\|^2 + \|f_q\|^2 - 2 \sum_{pq \in S^k} \langle f_p,$$

Substitute into the LHS

$$LHS = \sum_{k=1}^K \frac{\sum_{pq \in S^k} \|f_p - f_q\|^2}{2 |S^k|} = \sum_{k=1}^K \frac{const - 2 \sum_{pq \in S^k} \langle f_p, f_q \rangle}{2 |S^k|} = const - \sum_{k=1}^K \frac{2 \sum_l}{|S^k|}$$

Problem 4 - OPTIONAL BONUS (a simple finite-dimensional version of Mercer theorem)

Let A be an $n \times n$ positive semi-definite matrix defining pairwise affinities between n points. Find a closed-form expression for n vectors ϕ_i (a so-called "Euclidean embedding") such that their Euclidean dot products agree with the given affinities, i.e. $\langle \phi_i, \phi_j \rangle = A_{ij}$ for all $1 \leq i, j \leq n$. You can assume known eigen-decomposition $A = Q \Lambda Q^T$ where $\Lambda = diag(\lambda_1, \dots, \lambda_n)$ is a diagonal matrix of (non-negative!) eigen-values and Q is an orthogonal $n \times n$ matrix whose columns Q_i are unit eigen-vectors of A .

Solution:

Problem 5 - OPTIONAL BONUS (approximate low-dimensional Euclidean embedding)

Assume that \tilde{A} is a low-rank approximation of matrix A in problem 4 of given rank $m < n$. That is, $\tilde{A} = Q\Lambda_m Q^T$ where $\Lambda_m = \text{diag}(\lambda_1, \dots, \lambda_m, 0, \dots, 0)$ is a diagonal matrix of the largest m eigen values of A (a la Eckart–Young–Mirsky theorem, Topic 8). Using your solution for problem 4, specify a formula for "Euclidean embedding" $\{\tilde{\phi}_i\}$ such that $\langle \tilde{\phi}_i, \tilde{\phi}_j \rangle = \tilde{A}_{ij}$ and show that $\tilde{\phi}_i \in \mathcal{R}^m$.

Comment: basic K-means (Lloyd's algorithm) over such points $\{\tilde{\phi}_i\}$ can be used as an approximate algorithm for kernel clustering (e.g. for average association criteria). This approach is an example of "spectral clustering", which uses eigen decomposition of the affinity matrix A .

Solution:

Problem 4 (K-means).

Subproblem 4.1

Implement K-means (Lloyd's algorithm) for clustering pixel features. Most of the work is already done for you, but you do get a chance to play with numpy and to evaluate empirical properties of K-means. Note that your implementation will be slow if you use double-loops to traverse the pixels. There will be deductions for such double-loops. You should learn how to use functions like `np.where`, `np.minimum`, `np.square`, `np.ogrid` or others similar general functions that allow to avoid multi-loops over matrix (image) elements (pixels).

The provided code below only computes random pixel segments. You need to write code producing correct clusters and correct "means". To achieve this you only need to complete implementation of functions

compute_means and *compute_labels* inside "MyKmeansApp" corresponding to the two iterative steps in Lloyd's algorithm (as in "compute_k_means_clusters").

Your implementation of the main two steps of K-means algorithm should use RGBXY features. Relative contribution of "squared errors" from XY features must be set by parameter "weightXY" (or self.w inside MyKmeansApp), so that the squared error between RGBXY feature $F_p = [R_p, G_p, B_p, X_p, Y_p]$ at any pixel p and any given cluster mean $m = [R_m, G_m, B_m, X_m, Y_m]$ is

$$\|F_p - m\|^2 = (R_p - R_m)^2 + (G_p - G_m)^2 + (B_p - B_m)^2 + w \cdot (X_p - X_m)^2 + w \cdot (Y_p - Y_m)^2$$

Fully implemented "KmeansPresenter" visualizes the segmentation results (cluster labels mask) where each cluster is highlighted either by some random color (press r-key) or by the "mean" segment color (press m-key). All keys that "KmeansPresenter" responds to are as follows:

1. press 'i'-key for each (i)teration of K-means
2. press 'c'-key to run K-means to (c)onvergence (when energy improvement is less than given threshold)
3. press 'v'-key to run K-means to convergence with (v)isualization of each iteration
4. press 'r'-key to start over from (r)andom means
5. press 's'-key to change to a random (s)olid color-palette for displaying clusters
6. press 't'-key to change to a random (t)ransparent palette for displaying clusters
7. press 'm'-key to change to the (m)eans-color palette for displaying clusters

```
In [3]: %matplotlib notebook
# loading standard modules
import numpy as np
import math
import matplotlib.pyplot as plt
from skimage import img_as_ubyte
from skimage.color import rgb2gray

# loading custom module (requires file asg1.py in the same directory as the
from asg1_error_handling import Figure, KmeansPresenter
```

```
In [4]: class MyKmeansApp:

    def __init__(self, img, num_clusters=2, weightXY=1.0):
        self.k = num_clusters
        self.w = weightXY
        self.iteration = 0 # iteration counter
        self.energy = np.infty # energy - "sum of squared errors" (SSE)
```

```

num_rows = self.num_rows = img.shape[0]
num_cols = self.num_cols = img.shape[1]

self.im = img

self.means = np.zeros((self.k,5), 'd') # creates a zero-valued (double)
self.init_means()

self.no_label = num_clusters # special label value indicating pixel
# mask "labels" where pixels of each "region" will have a unique index
# the default mask value is "no-label" (K) implying pixels that do not
self.labels = np.full((num_rows, num_cols), fill_value=self.no_label)

self.fig = Figure()
self.pres = KmeansPresenter(img, self)
self.pres.connect_figure(self.fig)

def run(self):
    self.fig.show()

def init_means(self):
    self.iteration = 0 # resets iteration counter
    self.energy = np.infty # and the energy

    poolX = range(self.num_cols)
    poolY = range(self.num_rows)

    # generate K random pixels (Kx2 array with X,Y coordinates in each row)
    random_pixels = np.array([np.random.choice(poolX, self.k), np.random.choice(poolY, self.k)]).T

    for label in range(self.k):
        self.means[label,:3] = self.im[random_pixels[label,1],random_pixels[label,0],:3]
        self.means[label,3] = random_pixels[label,0]
        self.means[label,4] = random_pixels[label,1]

    # This function compute average values for R, G, B, X, Y channel (feature)
    # represented by labels in given mask "self.labels" storing indeces in rows
    # saved in (Kx5) matrix "self.means". The return value should be the number
def compute_means(self):
    labels = self.labels
    non_empty_clusters = 0

    # Your code below should compute average values for R,G,B,X,Y features
    # and save them in (Kx5) matrix "self.means". For empty clusters set
    # to infinity (np.infty). Report the correct number of non-empty clusters

    count = np.zeros(self.k) # used to count the number of every "group"
    tot = np.zeros((self.k,5), 'd') # a zero-valued (double) matrix of size Kx5
    for k in range(self.k):
        count[k] = np.sum(np.where(labels == k,1,0))
        tot[k][0] = np.sum(np.where(labels == k, self.im[:,:,:0],0)) # get R
        tot[k][1] = np.sum(np.where(labels == k, self.im[:,:,:1],0)) # get G

```

```

        tot[k][2] = np.sum(np.where(labels == k, self.im[:, :, 2], 0)) # get
    for i in range(self.num_rows):
        for k in range(self.k):
            tot[k][3] = tot[k][3] + np.sum(np.where(labels[i] == k, i, 0))
    for j in range(self.num_cols):
        for k in range(self.k):
            tot[k][4] = tot[k][4] + np.sum(np.where(labels[:, i] == k, j, 0))

    for k in range(self.k):
        if count[k] != 0:
            non_empty_clusters = non_empty_clusters + 1
            self.means[k][0] = tot[k][0] / count[k]
            self.means[k][1] = tot[k][1] / count[k]
            self.means[k][2] = tot[k][2] / count[k]
            self.means[k][3] = tot[k][3] / count[k]
            self.means[k][4] = tot[k][4] / count[k]
        else:
            for ind in range(5):
                self.means[k][ind] = np.inf

    return non_empty_clusters

# The segmentation mask is used by KmeanPresenter to paint segments in c
# NOTE: valid region labels are in [0,K), but the color map in KmeansPre
#       accepts labels in range [0,K] where pixels with no_label=K are n
def get_region_mask(self):
    return self.labels

# This function computes optimal (cluster) index/label in range 0,1,...,
# given current cluster means (self.means). The functions should save th
# The return value should be the corresponding optimal SSE.
def compute_labels(self):
    shape = (self.num_rows, self.num_cols)
    opt_labels = np.full(shape, fill_value=self.no_label, dtype=np.int)

    min_dist = np.full(shape, fill_value=np.inf) # HINT: you can use th
                                                # the (squared) distan
                                                # use 'self.w' as a re

    # Replace the code below by your code that computes "opt_labels" arr
    # each pixel's label is an index 'i' such that self.mean[i] is the c
    # Your code should also update min_dist so that it contains the optn

    all_shape = (self.k, self.num_rows, self.num_cols)
    dist_all = np.full(all_shape, fill_value=np.inf)
    for k in range(self.k):
        # sum the RGB distance for all pixels
        dist_all[k] = np.add(np.add(np.square(self.im[:, :, 0] - self.mean

```

```

    for i in range(self.num_rows):
        for k in range(self.k):
            temp_col = np.where(self.labels[i] == k, i, self.means[k][3])
                # set to mean value since
            temp_col = self.w * (np.square(temp_col - self.means[k][3]))
            dist_all[k][i] = np.add(dist_all[k][i], temp_col)      # add i

    for j in range(self.num_cols):
        for k in range(self.k):
            temp_row = np.where(self.labels[:,j] == k, j, self.means[k][4])
                # set to mean value
            temp_row = self.w * (np.square(temp_row - self.means[k][4]))
            dist_all[k,:,j] = np.add(dist_all[k,:,j], temp_row)

    for k in range(self.k):
        min_dist = np.minimum(min_dist, dist_all[k])

    for k in range(self.k):
        opt_labels = np.where(min_dist == dist_all[k], k, opt_labels)

    # update the labels based on opt_labels computed above
    self.labels = opt_labels

    # returns the optimal SSE (corresponding to optimal clusters/labels
    return min_dist.sum()

# The function below is called by "on_key_down" in KmeansPresenter".
# It's goal is to run an iteration of K-means procedure
# updating the means and the (segment) labels
def compute_k_means_clusters(self):
    self.iteration += 1

    # the main two steps of K-means algorithm
    energy = self.compute_labels()
    num_clusters = self.compute_means()

    # computing improvement and printing some information
    num_pixels = self.num_rows*self.num_cols
    improve_per_pixel = (self.energy - energy)/num_pixels
    energy_per_pixel = energy/num_pixels
    self.energy = energy

    self.fig.ax.text(0, -8,   # text location
                    'iteration = {:_>2d},  clusters = {:_>2d},  SSE/p'
                    self.iteration,      num_clusters,       energy_
                    bbox={'facecolor':'white', 'edgecolor':'none' })

    return improve_per_pixel

```

Subproblem 4.2

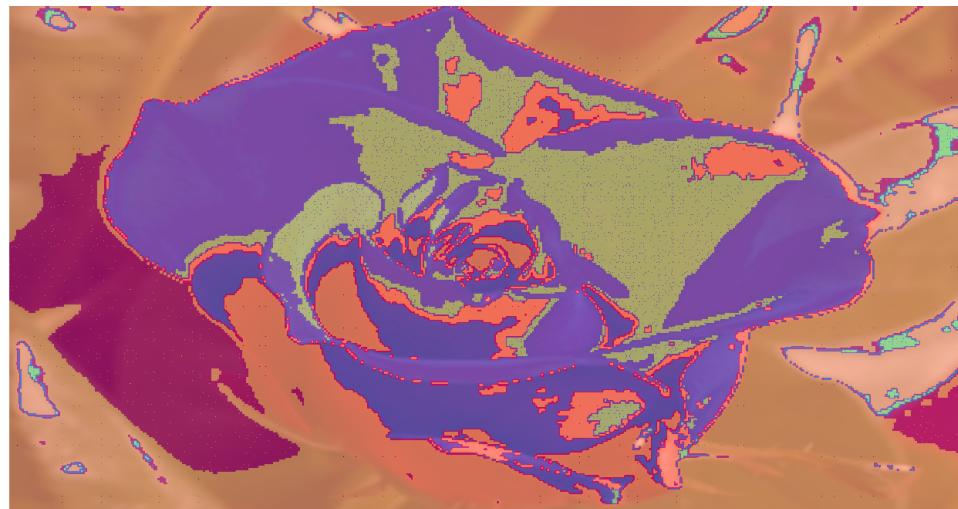
Use K-means to generate 3-4 representative results (you can use your own images) with color quantization and superpixels. Experiment with different values of parameter K (in the range 2-80). Compare representative values of optimal SSE for smaller and larger K and explain the observed differences. Add more cells (code and/or text) as necessary.

```
In [5]: img = plt.imread('images/rose.bmp')
app = MyKmeansApp(img, num_clusters=4, weightXY=2)
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfhg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/doc/1.20.0-notes.html#deprecations
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

```
iteration = _2, clusters = _4, SSE/p = _8278.4, improve/p = -1761.214
```

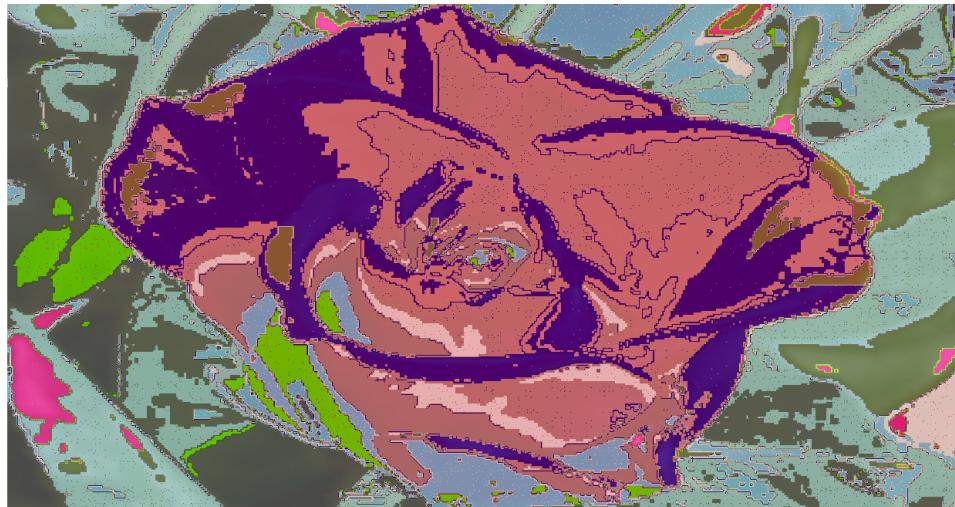


```
In [6]: img = plt.imread('images/rose.bmp')
app = MyKmeansApp(img, num_clusters=10, weightXY=2)
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfhg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

```
iteration = _2, clusters = 10, SSE/p = _3585.8, improve/p = -1230.300
```



```
In [7]: img = plt.imread('images/rose.bmp')
app = MyKmeansApp(img, num_clusters=40, weightXY=2)
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfhg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

```
iteration = _2, clusters = 40, SSE/p = _590.8, improve/p = -226.984
```

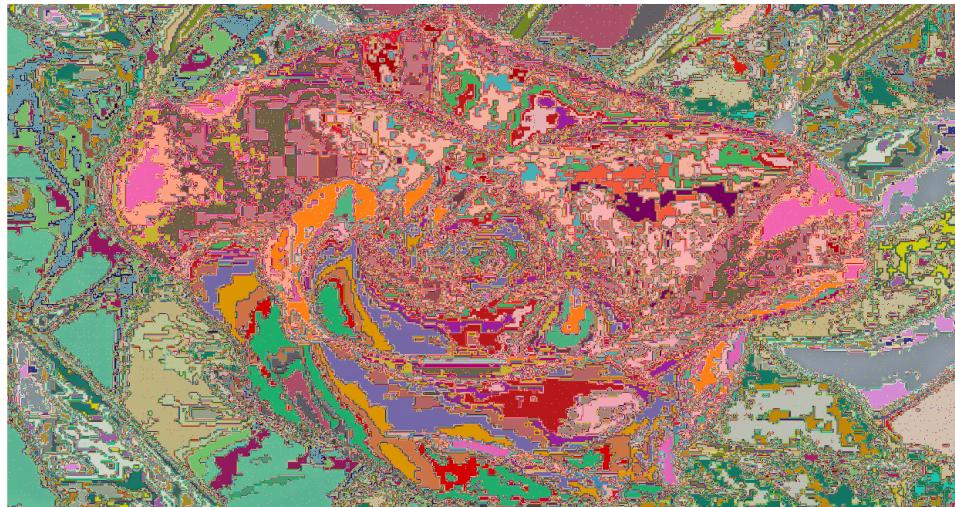


```
In [10]: img = plt.imread('images/rose.bmp')
app = MyKmeansApp(img, num_clusters=80, weightXY=2)
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfhg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

```
iteration = _4, clusters = 80, SSE/p = __218.1, improve/p = -45.155
```



Subprobelm 4.3

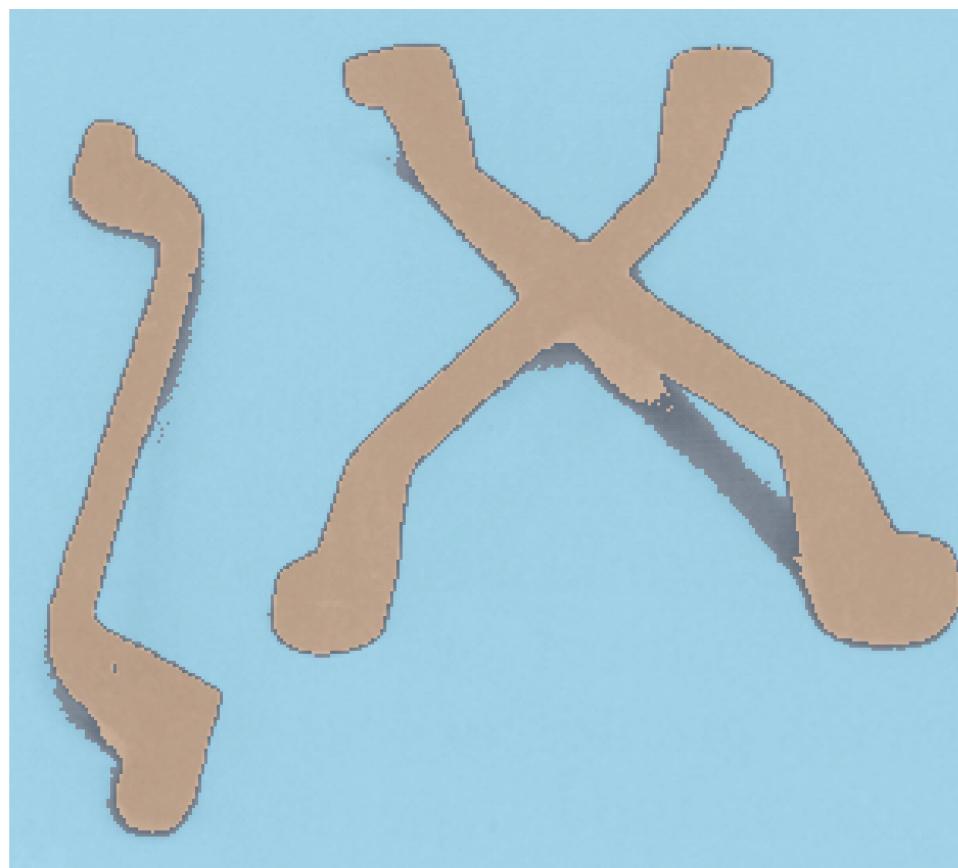
Demonstrate sensitivity of K-means to local minima (you can use your own images). Show 2-3 different solutions for different random initial means and display the corresponding values of the K-means energy. Add more cells (code and/or text) as necessary. Play wth different weights w and different number of clusters, different images.

```
In [11]: img = plt.imread('images/tools.bmp')
app = MyKmeansApp(img, num_clusters=3, weightXY=0.0)
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfsg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/doc/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = _6, clusters = _3, SSE/p = _290.4, improve/p = _0.000

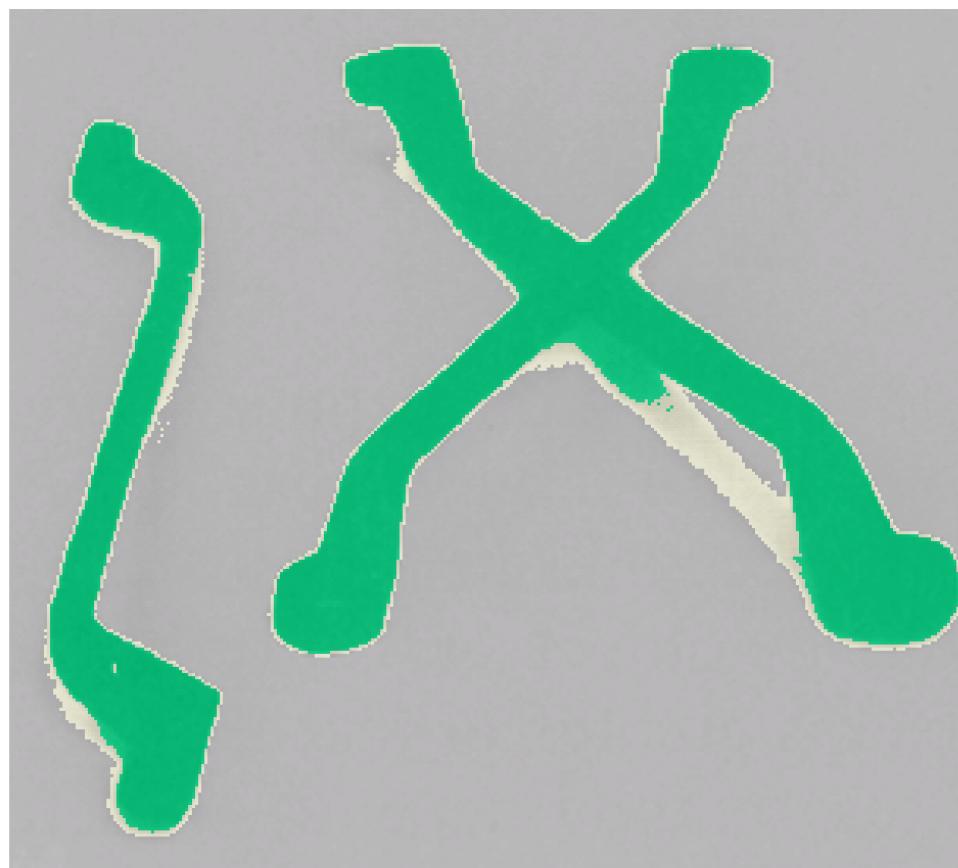


```
In [12]: img = plt.imread('images/tools.bmp')  
app = MyKmeansApp(img, num_clusters=3, weightXY=0.0)  
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfsg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/doc/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = 17, clusters = _3, SSE/p = __290.4, improve/p = __0.000

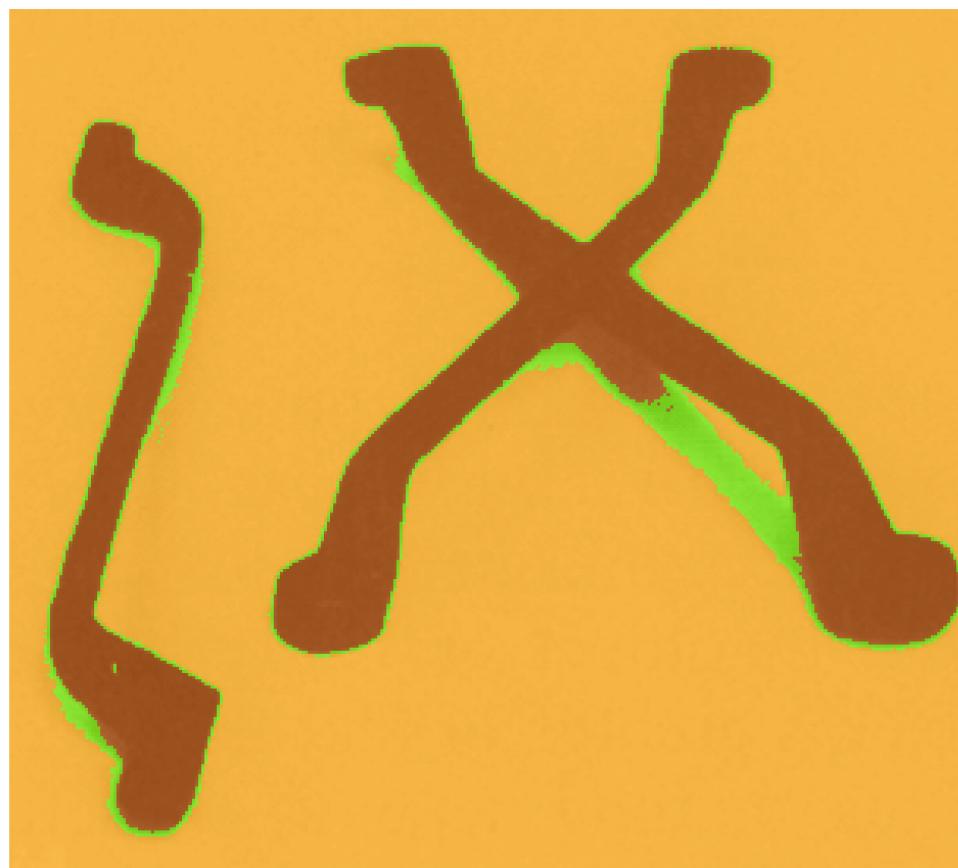


```
In [13]: img = plt.imread('images/tools.bmp')  
app = MyKmeansApp(img, num_clusters=3, weightXY=0.0)  
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfsg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/doc/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = 20, clusters = _3, SSE/p = __290.4, improve/p = __0.000

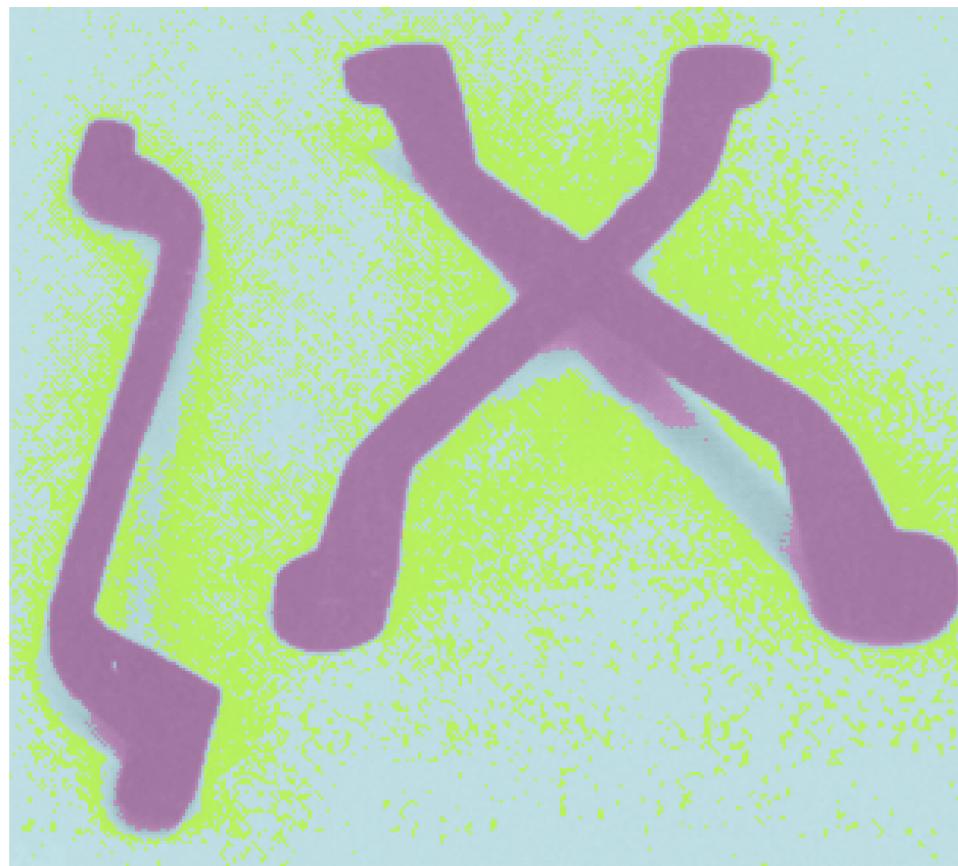


```
In [14]: img = plt.imread('images/tools.bmp')  
app = MyKmeansApp(img, num_clusters=3, weightXY=1)  
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfhg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/doc/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = _5, clusters = _3, SSE/p = _2924.4, improve/p = -12.142

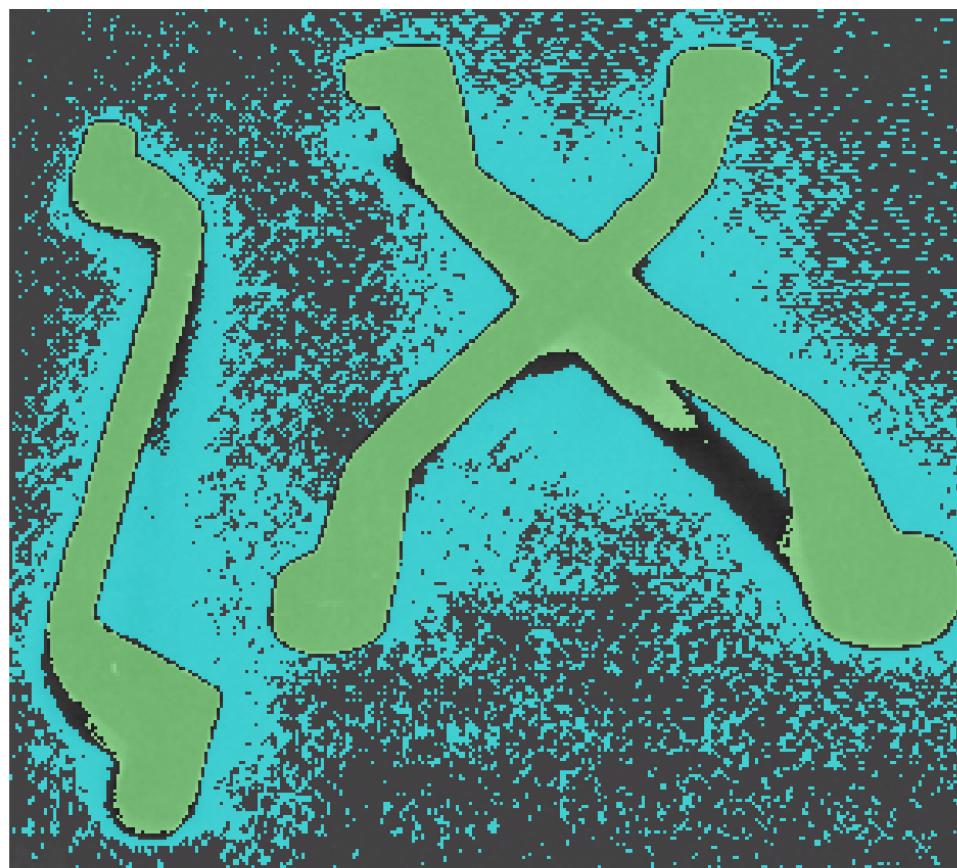


```
In [15]: img = plt.imread('images/tools.bmp')  
app = MyKmeansApp(img, num_clusters=3, weightXY=1.0)  
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfsg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/doc/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = _5, clusters = _3, SSE/p = _2930.8, improve/p = _-5.250



```
In [16]: img = plt.imread('images/bunny.bmp')  
app = MyKmeansApp(img, num_clusters=4, weightXY=0.0)  
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfsg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = 14, clusters = _4, SSE/p = _1551.2, improve/p = __0.000



```
In [17]: img = plt.imread('images/bunny.bmp')  
app = MyKmeansApp(img, num_clusters=4, weightXY=0.0)  
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfsg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/doc/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = 18, clusters = _4, SSE/p = _1551.2, improve/p = _0.000



```
In [18]: img = plt.imread('images/bunny.bmp')  
app = MyKmeansApp(img, num_clusters=4, weightXY=1.0)  
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfsg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/doc/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = _2, clusters = _4, SSE/p = _9004.0, improve/p = -6275.423

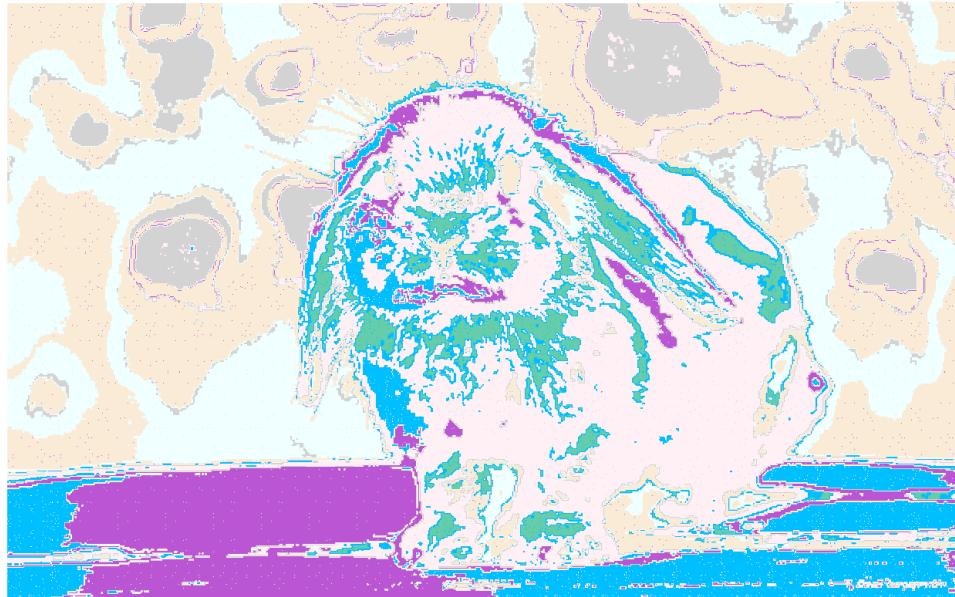


```
In [19]: img = plt.imread('images/bunny.bmp')  
app = MyKmeansApp(img, num_clusters=8, weightXY=1.0)  
app.run()
```

```
/var/folders/78/63ljq_56k5fcy3sfhg5zp10000gn/T/ipykernel_90750/4282981988.py:21: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations  
    self.labels = np.full((num_rows, num_cols), fill_value=self.no_label, dtype=np.int)
```

K-means

iteration = _2, clusters = _8, SSE/p = _3744.3, improve/p = -2616.804



The sensitivity is quite good if the image has only few colors and $w = 0$. As we add w , the local min is not as stable as the previous case, some noise appears. Similar situation happens when the image is not that simple.

In []: