Numerical Methods and Machine Learning for Image Processing

Week 3, Class 2: Basic Color Segmentation 2

October 12, 2021

Damon M. Chandler and Yi Zhang

Last Time: Basic Color Segmentation, Part 2

1. Grayscale thresholding
2. How to automatically find the threshold(s)
   - Otsu's method
   - Gaussian mixture model (GMM)

Today: Basic Color Segmentation, Part 2

1. EM Algorithm and GMM in 2D
2. Using a GMM for…
    1. Color segmentation
    2. Color-based detection

Today: Basic Color Segmentation, Part 2

1. EM Algorithm and GMM in 2D
2. Using a GMM for…
    1. Color segmentation
    2. Color-based detection

Steps of the EM algorithm

Given a Gaussian mixture model, the goal is to maximize the likelihood function with respect to the parameters (comprising the means and covariances of the components and the mixing coefficients).

1. Initialize the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$ and mixing coefficients π_k, and evaluate the initial value of the log likelihood.
2. **E step**. Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}.$$

3. **M step**. Re-estimate the parameters using the current responsibilities

4. Evaluate the log likelihood

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

3. **M step**. Re-estimate the parameters using the current responsibilities

$$\begin{aligned}
\boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})\mathbf{x}_n \\
\boldsymbol{\Sigma}_k^{\text{new}} &= \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})\left(\mathbf{x}_n-\boldsymbol{\mu}_k^{\text{new}}\right)\left(\mathbf{x}_n-\boldsymbol{\mu}_k^{\text{new}}\right)^{\mathrm{T}} \\
\pi_k^{\text{new}} &= \frac{N_k}{N}
\end{aligned}$$

where

$$N_k = \sum_{n=1}^{N}\gamma(z_{nk}).$$

4. Evaluate the log likelihood

$$\ln p(\mathbf{X}|\boldsymbol{\mu},\boldsymbol{\Sigma},\boldsymbol{\pi}) = \sum_{n=1}^{N}\ln\left\{\sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right\}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

EM Algorithm demo of GMM clustering of 1D data

```
import numpy as np
import matplotlib.pyplot as plt
import ipcv_utils.utils as ipcv_plt
import cv2 as cv
from sklearn import datasets
from scipy.stats import norm
from sklearn.cluster import Kmeans


#%% CREATE THE RANDOM DATA

X, y = datasets.make_blobs(n_samples=1000,
  n_features=1, centers=[(30,), (100,), (200,)],
  cluster_std=[5, 15, 20])
X[X<0] = 0
X[X>255] = 255

xs = X

fig, ax1 = plt.subplots()
ax1.plot(xs, np.zeros(len(xs)), "o", markersize=7,
  markeredgecolor=(0,0,0), alpha=0.2)
ax1.set_ylim([-0.001, None])


#%% GMM FITTING

def loglikelihood(xs, pis, mus, covs):
  res = 0
  for n in range(N):
    sum_k = 0.0
    for k in range(K):
      sum_k += pis[k]*norm.pdf(xs[n], mus[k],
        np.sqrt(covs[k]))

    if sum_k > 0:
      res += np.log(sum_k)

  return res
```

```
# number of data points and dimensionality of data
N, dims = xs.shape # 1000, 1
# number of clusters (Gaussians)
K = 3


#------------------------------------------------------------
# INITIAL GUESSES OF THE GMM GAUSSIAN PARAMETERS
#------------------------------------------------------------

# pis_k are the prior probabilities of the two classes
# pis_k are p(z_k=1)
pis = np.random.random(K)
pis /= pis.sum()

# initial guess for the means will be based on k-means
kmeans = KMeans(n_clusters=K)
kmeans.fit(xs)
mus = kmeans.cluster_centers_
# mus = 255*np.random.random((K, dims))

# random initial guess for the covariances
covs = 50*np.ones(3)

print("pis = \n", pis, "\n")
print("mus = \n", mus, "\n")
print("covs = \n", covs, "\n")


#------------------------------------------------------------
# MAIN ITERATION LOOP
#------------------------------------------------------------
tol = 0.001
max_iter = 20

ll_old = loglikelihood(xs, pis, mus, covs)

for i in range(max_iter):

  fig, ax1 = plt.subplots()
  # ax1.hist(X, bins=100)
  ax1.plot(xs, np.zeros(len(xs)), "o", markersize=7,
    markeredgecolor=(0,0,0), alpha=0.2)
  ax1.set_ylim([-0.001, None])

  print("~~~~~ Iteration:", i, "~~~~~")
  print("ll_old =", ll_old)
```

EM Algorithm demo of GMM clustering of 1D data (cont.)

```
#-------------------------------------------------------------
# E-step
#-------------------------------------------------------------
# gammas_k are the posterior probs of z_k=1 given xs
# gammas_k are p(z_k=1 | xs) [cond prob of z given x]
gammas = np.zeros((K, N))

# for each class (k)
for k in range(K):
  # compute p(x_i | C_j)
  for n in range(N):
    gammas[k, n] = \
      pis[k]*norm.pdf(xs[n], mus[k], np.sqrt(covs[k]))
gammas /= gammas.sum(0)

#-------------------------------------------------------------
# M-step
#-------------------------------------------------------------
for k in range(K):
  # total weight of class k
  N_k = gammas[k, :].sum()

  # new weighted mean (mus[k]) for each class (k)
  mus[k] = 0
  for n in range(N):
    mus[k] += gammas[k, n] * xs[n]
  mus[k] /= N_k

  # new weighted std (covs[k]) for each class (k)
  covs[k] = 0
  for n in range(N):
    xs_ms = xs[n] - mus[k]
    covs[k] += gammas[k, n] * xs_ms*xs_ms
  covs[k] /= N_k

  # new prior probability (pis[k]) for each class (k)
  pis[k] = N_k / N
```

```
  # plot the current fits
  x = np.arange(0, 256, 0.001)
  g1 = norm.pdf(x, mus[0], np.sqrt(covs[0]))
  g2 = norm.pdf(x, mus[1], np.sqrt(covs[1]))
  g3 = norm.pdf(x, mus[2], np.sqrt(covs[2]))
  g1 = g1.transpose()
  g2 = g2.transpose()
  g3 = g3.transpose()

  p = pis[0]*g1 + pis[1]*g2 + pis[2]*g3

  ax2 = ax1.twinx()
  ax2.plot(x, p, color=(0,0,0,0.35), linewidth=10)

  ax2.set_ylim([-0.001, None])
  ax2.plot(x, pis[0]*g1, color=(1,0,0,0.8))
  ax2.plot(x, pis[1]*g2, color=(0,1,0,0.8))
  ax2.plot(x, pis[2]*g3, color=(0,0,1,0.8))


  # update complete log likelihoood
  ll_new = loglikelihood(xs, pis, mus, covs)
  print("ll_new =", ll_new, "\n")

  if np.abs(ll_new - ll_old) < tol:
    break

  ll_old = ll_new

plt.show()
print("DONE")

# show the final best Gaussian parameters
print("pis = \n", pis, "\n")
print("mus = \n", mus, "\n")
print("covs = \n", covs, "\n")
```

```
pis =
 [0.12963267 0.79549856 0.07486877]
mus =
 [[157.19890199]
 [ 64.58191658]
 [121.58542168]]
covs =
 [50. 50. 50.]
```

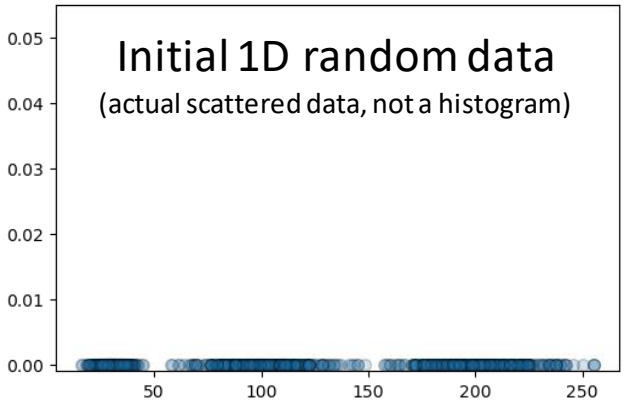
Initial guesses of parameters

```
~~~~~ Iteration: 0 ~~~~~
ll_old = [-16814.07394188]
ll_new = [[-5342.67989296]]

~~~~~ Iteration: 1 ~~~~~
ll_old = [[-5342.67989296]]
ll_new = [[-5309.00226606]]

~~~~~ Iteration: 2 ~~~~~
ll_old = [[-5309.00226606]]
ll_new = [[-5264.49050992]]

...

~~~~~ Iteration: 6 ~~~~~
ll_old = [[-4953.37083972]]
ll_new = [[-4953.37059425]]
```

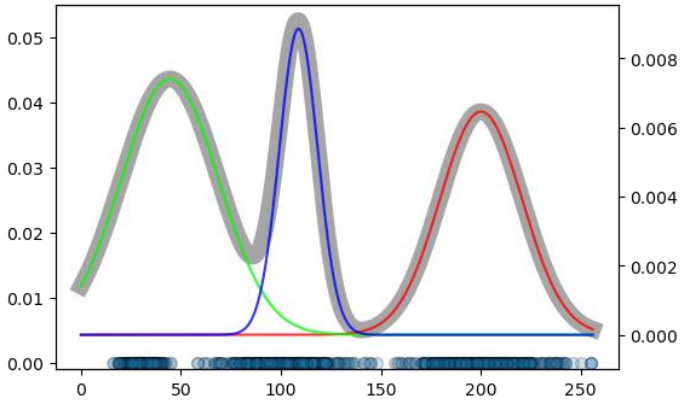
Goodness-of-fit log-likelihood is gradually decreasing, which is good

```
pis =
 [0.32957826 0.33387805 0.33654368]
mus =
 [[200.74319207]
 [ 30.35399544]
 [100.38186005]]
covs =
 [392.39639861 23.98945439
  247.38116502]
```

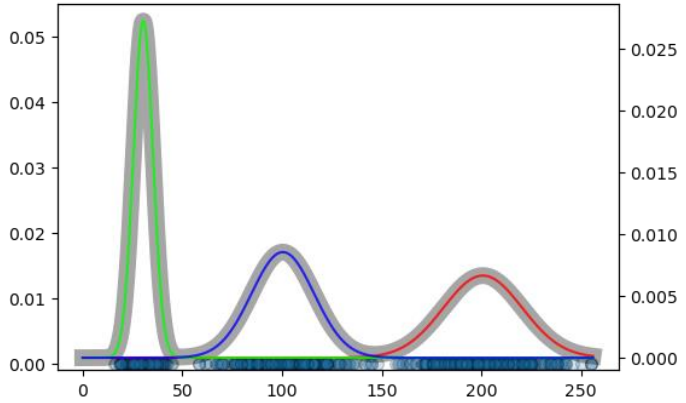
Final fitted parameters



After first iteration



After last iteration



EM Algorithm demo of GMM clustering of 2D data

```
import numpy as np
import matplotlib.pyplot as plt
import ipcv_utils.utils as ipcv_plt
import cv2 as cv
from scipy.stats import multivariate_normal as mvn
from sklearn.cluster import Kmeans


#%% CREATE THE RANDOM DATA

n = 1000
_mus = np.array([[0,4], [-2,0]])
_sigmas = np.array([[[3, 0], [0, 0.5]], [[1,0],[0,2]]])
_pis = np.array([0.6, 0.4])
xs = np.concatenate(
  [np.random.multivariate_normal(mu, sigma, int(pi*n))
    for pi, mu, sigma in zip(_pis, _mus, _sigmas)])

ax = plt.subplot(111)
plt.scatter(xs[:,0], xs[:,1], alpha=0.2)
plt.axis([-8,6,-6,8])
ax.axes.set_aspect('equal')
plt.tight_layout()
plt.show()


#%% GMM FITTING

def loglikelihood(xs, pis, mus, covs):
  res = 0
  for n in range(N):
    sum_k = 0.0
    for k in range(K):
      sum_k += pis[k]*norm.pdf(xs[n], mus[k],
        np.sqrt(covs[k]))

    if sum_k > 0:
      res += np.log(sum_k)

  return res
```

```
# number of data points and dimensionality of data
N, dims = xs.shape # 1000, 2
# number of clusters (Gaussians)
K = 2


#------------------------------------------------------------
# INITIAL GUESSES OF THE GMM GAUSSIAN PARAMETERS
#------------------------------------------------------------

# pis_k are the prior probabilities of the two classes
# pis_k are p(z_k=1)
pis = np.random.random(K)
pis /= pis.sum()

# initial guess for the means will be based on k-means
kmeans = KMeans(n_clusters=K)
kmeans.fit(xs)
mus = kmeans.cluster_centers_
# mus = 255*np.random.random((K, dims))

# random initial guess for the covariances
covs = 5*np.array([np.eye(dims)] * K)

print("pis = \n", pis, "\n")
print("mus = \n", mus, "\n")
print("covs = \n", covs, "\n")


#------------------------------------------------------------
# MAIN ITERATION LOOP
#------------------------------------------------------------
tol = 0.0001
max_iter = 10

ll_old = loglikelihood(xs, pis, mus, covs)

for i in range(max_iter):

  print("~~~~~ Iteration:", i, "~~~~~")
  print("ll_old =", ll_old)
```

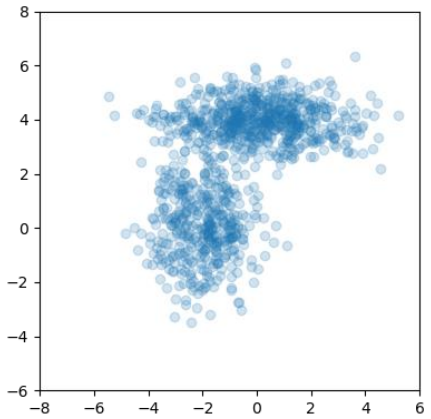
EM Algorithm demo of GMM clustering of 2D data (cont.)

```
#-------------------------------------------------------------
# E-step
#-------------------------------------------------------------
gammas = np.zeros((K, N))

# for each class (k)
for k in range(K):
  # compute p(x_i | C_j)
  for n in range(N):
    gammas[k, n] = pis[k]*mvn(mus[k], covs[k]).pdf(xs[n])
gammas /= gammas.sum(0)

#-------------------------------------------------------------
# M-step
#-------------------------------------------------------------
for k in range(K):
  # total weight of class k
  N_k = gammas[k, :].sum()

  # new weighted mean (mus[k]) for each class (k)
  mus[k] = 0
  for n in range(N):
    mus[k] += gammas[k, n] * xs[n]
  mus[k] /= N_k

  # new weighted std (covs[k]) for each class (k)
  covs[k] = 0
  for n in range(N):
    xs_ms = np.reshape(xs[n] - mus[k], (dims, 1))
    covs[k] += gammas[k, n] * np.dot(xs_ms, xs_ms.T)
  covs[k] /= N_k

  # new prior probability (pis[k]) for each class (k)
  pis[k] = N_k / N
```

```
  intervals = 101
  ys = np.linspace(-8, 8, intervals)
  Xm, Ym = np.meshgrid(ys, ys)
  _ys = np.vstack([Xm.ravel(), Ym.ravel()]).T

  z = np.zeros(len(_ys))
  for pi, mu, sigma in zip(pis, mus, covs):
    z += pi*mvn(mu, sigma).pdf(_ys)
  z = z.reshape((intervals, intervals))

  plt.figure()
  ax = plt.subplot(111)
  plt.scatter(xs[:,0], xs[:,1], alpha=0.2)
  plt.contour(Xm, Ym, z)
  plt.axis([-8,6,-6,8])
  ax.axes.set_aspect('equal')
  plt.tight_layout()


  # update complete log likelihoood
  ll_new = loglikelihood(xs, pis, mus, covs)
  print("ll_new =", ll_new, "\n")

  if np.abs(ll_new - ll_old) < tol:
    break

  ll_old = ll_new

plt.show()
print("DONE")

# show the final best Gaussian parameters
print("pis = \n", pis, "\n")
print("mus = \n", mus, "\n")
print("covs = \n", covs, "\n")
```

```
pis =
 [0.49295727 0.50704273]
mus =
 [[-0.05405294  4.01550299]
 [-2.03013382 -0.11026381]]
covs =
 [[[5. 0.]
  [0. 5.]]
 [[5. 0.]
  [0. 5.]]]
```

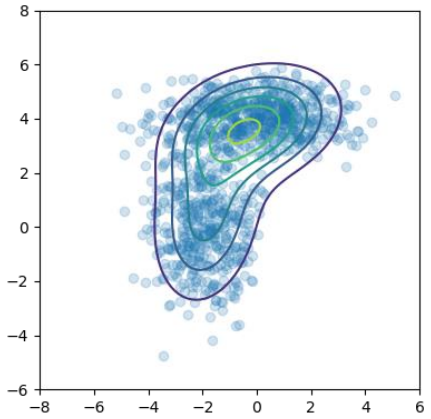
Initial guesses of parameters



Initial 1D random data
(actual scattered data, not a histogram)

```
~~~~~ Iteration: 0 ~~~~~
ll_old = -4300.265053608878
ll_new = -3881.8527387392564

~~~~~ Iteration: 1 ~~~~~
ll_old = -3881.8527387392564
ll_new = -3777.121885961512

...

~~~~~ Iteration: 9 ~~~~~
ll_old = -3723.100719063559
ll_new = -3723.094782708666
```

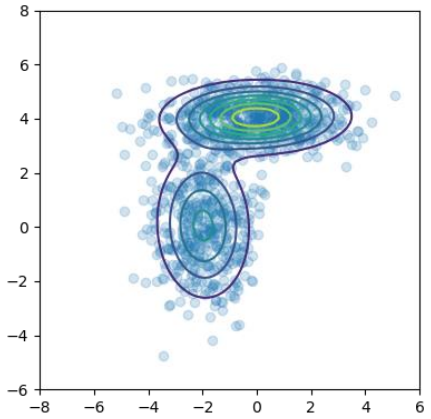
Goodness-of-fit log-likelihood is gradually decreasing, which is good



After first iteration

```
pis =
 [0.58502691 0.41497309]
mus =
 [[-0.02008143  4.0511843 ]
 [-1.97815387  0.04795264]]
covs =
 [[[ 3.00618401  0.03862379]
  [ 0.03862379  0.46424529]]
 [[ 0.98222215 -0.07911116]
  [-0.07911116  2.43409986]]]
```

Final fitted parameters



After last iteration

Today: Basic Color Segmentation, Part 2

1. EM Algorithm and GMM in 2D
2. Using a GMM for…
    1. Color segmentation
    2. Color-based detection

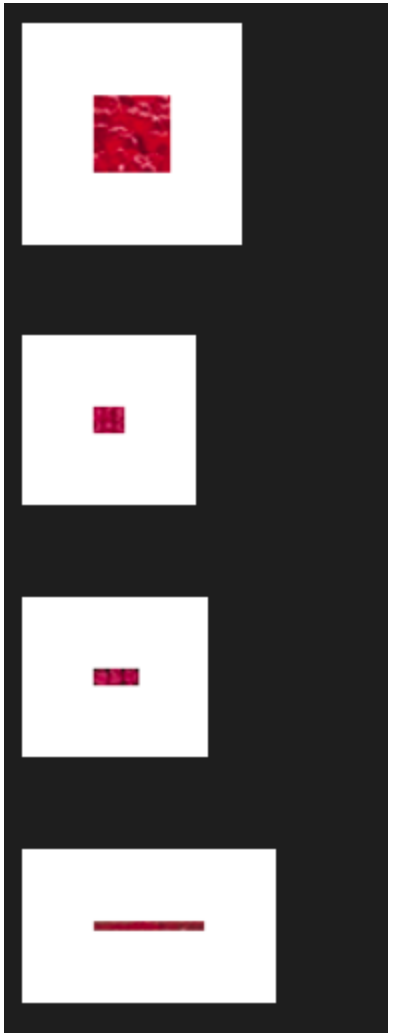
GMM color segmentation

```
import numpy as np
import matplotlib.pyplot as plt
import ipcv_utils.utils as ipcv_plt
import cv2 as cv
from sklearn.mixture import GaussianMixture


img_bgr = cv.imread("imgs/1600.png")
img_bgr = img_bgr.astype(np.float32) / 255

img_rgb = cv.cvtColor(img_bgr, cv.COLOR_BGR2RGB)
ipcv_plt.imshow(img_rgb)

img_lab = cv.cvtColor(img_rgb, cv.COLOR_RGB2Lab)
lab_vals = img_lab.reshape(-1, 3)

gmm = GaussianMixture(n_components=3, n_init=10,
  covariance_type="diag")
gmm.fit(lab_vals)

labels = gmm.predict(lab_vals)
centers = gmm.means_
centers = centers.astype(np.float32)

seg_vals = centers[labels]
centers_lab = centers[:, 0:3]
seg_lab_vals = seg_vals[:, 0:3]

seg_img_lab = seg_lab_vals.reshape(img_lab.shape)
seg_img_rgb = cv.cvtColor(seg_img_lab, cv.COLOR_Lab2RGB)
ipcv_plt.imshow(seg_img_rgb, zoom=1)
```

GMM color detection (detecting just one object—here, red flowers)

```python
import numpy as np
import matplotlib.pyplot as plt
import ipcv_utils.utils as ipcv_plt
import cv2 as cv
from sklearn.mixture import GaussianMixture


#%% BUILD THE GMM MODEL FROM THE TRAINING PATCHES

img_bgr_1 = cv.imread("imgs/1600.crop.png")
img_bgr_2 = cv.imread("imgs/1600.crop.2.png")
img_bgr_3 = cv.imread("imgs/1600.crop.3.png")
img_bgr_4 = cv.imread("imgs/1600.crop.4.png")

img_bgr_1 = img_bgr_1.astype(np.float32) / 255
img_bgr_2 = img_bgr_2.astype(np.float32) / 255
img_bgr_3 = img_bgr_3.astype(np.float32) / 255
img_bgr_4 = img_bgr_4.astype(np.float32) / 255

img_rgb_1 = cv.cvtColor(img_bgr_1, cv.COLOR_BGR2RGB)
img_rgb_2 = cv.cvtColor(img_bgr_2, cv.COLOR_BGR2RGB)
img_rgb_3 = cv.cvtColor(img_bgr_3, cv.COLOR_BGR2RGB)
img_rgb_4 = cv.cvtColor(img_bgr_4, cv.COLOR_BGR2RGB)

ipcv_plt.imshow(img_rgb_1)
ipcv_plt.imshow(img_rgb_2)
ipcv_plt.imshow(img_rgb_3)
ipcv_plt.imshow(img_rgb_4)

img_lab_1 = cv.cvtColor(img_rgb_1, cv.COLOR_RGB2Lab)
img_lab_2 = cv.cvtColor(img_rgb_2, cv.COLOR_RGB2Lab)
img_lab_3 = cv.cvtColor(img_rgb_3, cv.COLOR_RGB2Lab)
img_lab_4 = cv.cvtColor(img_rgb_4, cv.COLOR_RGB2Lab)
```

First, we will train a GMM model on the following training patches.



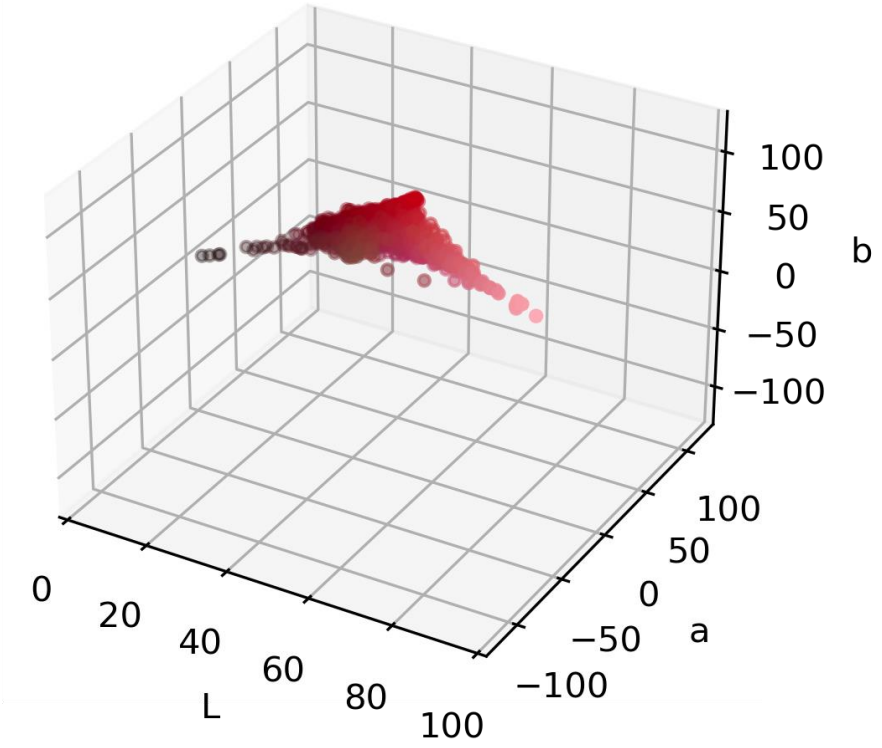
GMM color detection (detecting just one object—here, red flowers)

```
rgb_vals_1 = img_rgb_1.reshape(-1, 3)
rgb_vals_2 = img_rgb_2.reshape(-1, 3)
rgb_vals_3 = img_rgb_3.reshape(-1, 3)
rgb_vals_4 = img_rgb_4.reshape(-1, 3)

lab_vals_1 = img_lab_1.reshape(-1, 3)
lab_vals_2 = img_lab_2.reshape(-1, 3)
lab_vals_3 = img_lab_3.reshape(-1, 3)
lab_vals_4 = img_lab_4.reshape(-1, 3)

len_1 = lab_vals_1.shape[0]
len_2 = lab_vals_2.shape[0]
len_3 = lab_vals_3.shape[0]
len_4 = lab_vals_4.shape[0]

rgb_vals = np.zeros((len_1+len_2+len_3+len_4, 3), np.float32)
rgb_vals[:len_1, 0:3] = rgb_vals_1
rgb_vals[len_1:len_1+len_2, 0:3] = rgb_vals_2
rgb_vals[len_1+len_2:len_1+len_2+len_3, 0:3] = rgb_vals_3
rgb_vals[len_1+len_2+len_3:, 0:3] = rgb_vals_4

lab_vals = np.zeros((len_1+len_2+len_3+len_4, 3), np.float32)
lab_vals[:len_1, 0:3] = lab_vals_1
lab_vals[len_1:len_1+len_2, 0:3] = lab_vals_2
lab_vals[len_1+len_2:len_1+len_2+len_3, 0:3] = lab_vals_3
lab_vals[len_1+len_2+len_3:, 0:3] = lab_vals_4

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.scatter(lab_vals[:, 0], lab_vals[:, 1],
  lab_vals[:, 2], c=rgb_vals, s=10)
ax.set_xlabel("L")
ax.set_ylabel("a")
ax.set_zlabel("b")
ax.set_xlim([0, 100])
ax.set_ylim([-128, 128])
ax.set_zlim([-128, 128])
plt.show()
```

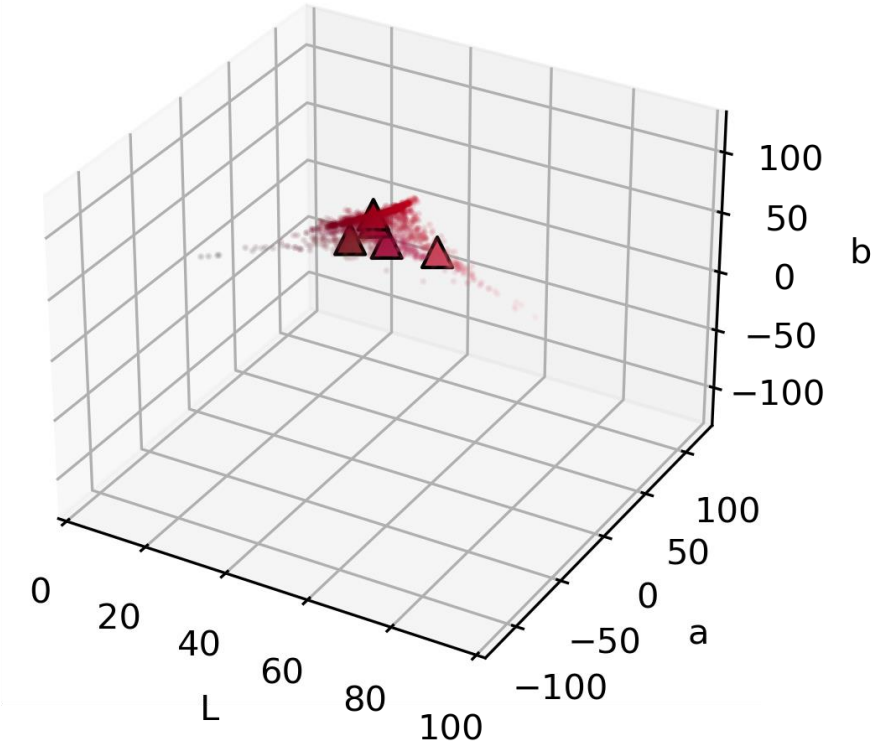
```
gmm = GaussianMixture(n_components=5, n_init=10)
gmm.fit(lab_vals)

labels = gmm.predict(lab_vals)
centers = gmm.means_
centers = centers.astype(np.float32)

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.scatter(lab_vals[:, 0], lab_vals[:, 1],
  lab_vals[:, 2], c=rgb_vals, s=10)
ax.set_xlabel("L"); ax.set_ylabel("a"); ax.set_zlabel("b")
ax.set_xlim([0, 100])
ax.set_ylim([-128, 128])
ax.set_zlim([-128, 128])
plt.show()

seg_vals = centers[labels]

centers_lab = centers[:, 0:3]
seg_lab_vals = seg_vals[:, 0:3]

# convert Lab centers and Lab segmented list to RGB to
# color each point in the 3D graphs
centers_as_rgb = cv.cvtColor(centers_lab.reshape(
  (-1, 1, 3)), cv.COLOR_Lab2RGB)
seg_lab_vals_as_rgb = cv.cvtColor(seg_lab_vals.reshape(
  (-1, 1, 3)), cv.COLOR_Lab2RGB)

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.scatter(lab_vals[::1, 0], lab_vals[::1, 1],
  lab_vals[::1, 2], c=rgb_vals[::1], s=1, alpha=0.1)
ax.scatter(centers[:, 0], centers[:, 1],
  centers[:, 2], c=centers_as_rgb, s=75, marker="^",
  edgecolors=[(0, 0, 0)], alpha=1)
ax.set_xlabel("L"); ax.set_ylabel("a"); ax.set_zlabel("b")
ax.set_xlim([0, 100])
ax.set_ylim([-128, 128])
ax.set_zlim([-128, 128])
plt.show()
```

GMM color detection (detecting just one object—here, red flowers)

Lab data of the training patches



Fitted GMM centers (5 Gaussians)



GMM color detection (detecting just one object—here, red flowers)

Now, we use the trained GMM model to detect the red flowers.

```
#%% DO DETECTION ON FULL IMAGE USING THE TRAINED MODEL

img_bgr = cv.imread("imgs/1600.png")
img_bgr = img_bgr.astype(np.float32) / 255

img_rgb = cv.cvtColor(img_bgr, cv.COLOR_BGR2RGB)
ipcv_plt.imshow(img_rgb)

img_lab = cv.cvtColor(img_rgb, cv.COLOR_RGB2Lab)
lab_vals = img_lab.reshape(-1, 3)

res = gmm.score_samples(lab_vals)
density_threshold = np.percentile(res, 85)
res[res >= density_threshold] = 255
res[res < density_threshold] = 0

res = res.reshape((img_lab.shape[0], img_lab.shape[1]))
ipcv_plt.imshow(res, cmap="gray", vmin=res.min(),
  vmax=res.max())
```