Lecture 01: Clustering class demo

Contents

Let's cluster images!!

For this demo, I'm going to use two image datasets:

1. A small subset of 200 Bird Species with 11,788 Images dataset (available here)
2. A tiny subset of Food-101 (available here)

To run the code below, you need to install pytorch and torchvision in the course conda environment.

```
conda install pytorch torchvision -c pytorch
```

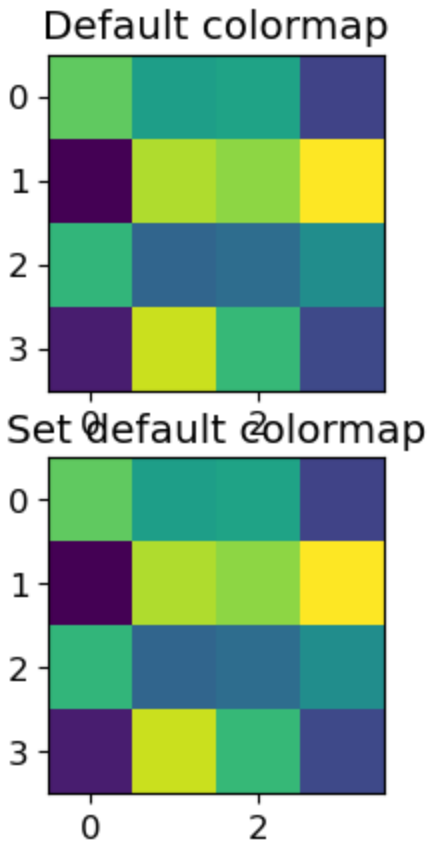
```
import os
import random
import sys
import time

import numpy as np
import pandas as pd

sys.path.append(os.path.join(os.path.abspath(".."), "code"))
from plotting_functions import *

DATA_DIR = os.path.join(os.path.abspath(".."), "data/")

import torch
import torchvision
from torchvision import datasets, models, transforms, utils
from PIL import Image
import matplotlib.pyplot as plt
import random
```



Let's start with small subset of birds dataset. You can experiment with a bigger dataset if you like.

```
#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device = torch.device('mps' if torch.backends.mps.is_available() else 'cpu')
device
```

```
device(type='mps')
```

```
def set_seed(seed=42):
    torch.manual_seed(seed)
    np.random.seed(seed)
    random.seed(seed)
```

```
set_seed(seed=42)
```

```
import glob
IMAGE_SIZE = 224
def read_img_dataset(data_dir):
    data_transforms = transforms.Compose(
        [
            transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
            transforms.ToTensor(),
            transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5]),
        ])

    image_dataset = datasets.ImageFolder(root=data_dir, transform=data_transfo
    dataloader = torch.utils.data.DataLoader(
         image_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=0
    )
    dataset_size = len(image_dataset)
    class_names = image_dataset.classes
    inputs, classes = next(iter(dataloader))
    return inputs, classes
```

```
def plot_sample_imgs(inputs):
    plt.figure(figsize=(10, 70)); plt.axis("off"); plt.title("Sample Training
    plt.imshow(np.transpose(utils.make_grid(inputs, padding=1, normalize=True)
```

```
data_dir = DATA_DIR + "/birds"
file_names = [image_file for image_file in glob.glob(data_dir + "/*/*.jpg")]
n_images = len(file_names)
BATCH_SIZE = n_images  # because our dataset is quite small
birds_inputs, birds_classes = read_img_dataset(data_dir)
```

```
X_birds = birds_inputs.numpy()
```

```
plot_sample_imgs(birds_inputs[0:24,:,:,:])
plt.show()
```

For clustering we need to calculate distances between points. So we need a vector representation for each data point. A simplest way to create a vector representation of an image is by flattening the image.

```
flatten_transforms = transforms.Compose([
                    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
                    transforms.ToTensor(),
                    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5]),
                    transforms.Lambda(torch.flatten)])
flatten_images = datasets.ImageFolder(root='../data/birds', transform=flatten_
```

```
flatten_dataloader = torch.utils.data.DataLoader(
        flatten_images, batch_size=BATCH_SIZE, shuffle=True, num_workers=0
    )
```

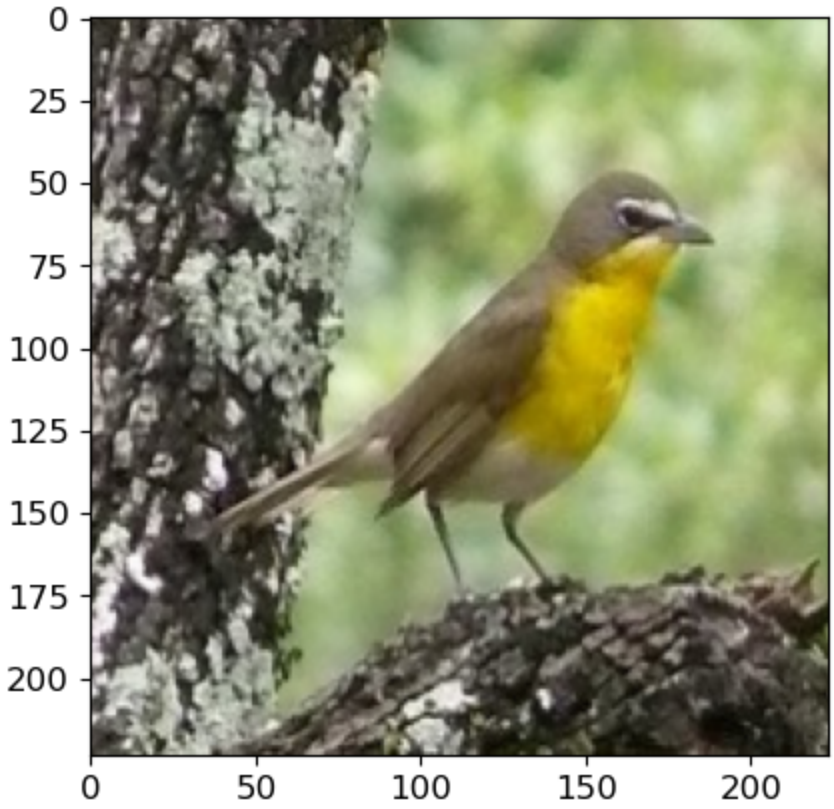
```
flatten_train, y_train = next(iter(flatten_dataloader))
```

```
flatten_images = flatten_train.numpy()
```

```

```

```
image_shape=[3,224,224]
img = flatten_images[20].reshape(image_shape)
plt.imshow(np.transpose(img / 2 + 0.5, (1, 2, 0)));
```




```
flatten_images.shape # 224 by 224 images with 3 color channels
```

```
(176, 150528)
```

```
from sklearn.cluster import KMeans
k = 3
km_flatten = KMeans(k, n_init='auto', random_state=123)
km_flatten.fit(flatten_images)
```

KMeans

```
KMeans(n_clusters=3, random_state=123)
```

```
km_flatten.cluster_centers_.shape
```

```
(3, 150528)
```

```
flatten_images.shape
```

```
(176, 150528)
```

```
unflatten_inputs = np.array([img.reshape(image_shape) for img in flatten_image
```

```
for cluster in range(k):
    # user-defined functions defined in ../code/plotting_functions.py
    get_cluster_images(km_flatten, flatten_images, unflatten_inputs, cluster,
```

```
158
Image indices:  [158  65  48 125  95]
```

Cluster center 0

```
165
Image indices:  [165  94  77 152 108]
```

Cluster center 1

```
156
Image indices:  [156  89 100  25 133]
```

Cluster center 2

Let's try clustering with GMMs

```
from sklearn.mixture import GaussianMixture

gmm_flatten = GaussianMixture(n_components=k,covariance_type='diag', random_st
gmm_flatten.fit(flatten_images)
```

GaussianMixture

```
GaussianMixture(covariance_type='diag', n_components=3, random_state=123)
```

```
for cluster in range(k):
    # user-defined functions defined in ../code/plotting_functions.py
    get_cluster_images(gmm_flatten, flatten_images, unflatten_inputs, cluster=
```

```
Image indices:  [ 48 106 104 122  87]
```

Cluster 0

```
Image indices:  [ 56  55  54 113 175]
```

Cluster 1

```
Image indices:  [114  39 126  90  64]
```

We still see some mis-categorizations. It seems like when we flatten images, clustering doesn't seem that great.

Let's try out a different input representation. Let's use transfer learning as a feature extractor with a pre-trained vision model. For each image in our dataset we'll pass it through a pretrained network and get a representation from the last layer, before the classification layer given by the pre-trained network.

We see some mis-categorizations.

How about trying out a different input representation? Let's use transfer learning as a feature extractor with a pre-trained vision model. For each image in our dataset we'll pass it through a pretrained network and get a representation from the last layer, before the classification layer given by the pre-trained network.

```
def get_features(model, inputs):
    """Extract output of densenet model"""
    model.eval()
    with torch.no_grad():  # turn off computational graph stuff
        Z = model(inputs).detach().numpy()
    return Z
```

```
densenet = models.densenet121(weights="DenseNet121_Weights.IMAGENET1K_V1")
densenet.classifier = torch.nn.Identity()  # remove that last "classification"
Z_birds = get_features(densenet, birds_inputs)
```

```
Z_birds.shape
```

```
(176, 1024)
```

```
pd.DataFrame(Z_birds)
```

	0	1	2	3	4	5	6
0	0.000221	0.005660	0.002462	0.004169	0.097525	0.287082	0.000673
1	0.000184	0.006229	0.002463	0.001197	0.102545	0.178865	0.000516
2	0.000444	0.007750	0.002796	0.001045	0.126620	0.224383	0.000616
3	0.000131	0.005346	0.001581	0.001190	0.124439	0.318437	0.000681
4	0.000338	0.006431	0.004826	0.001502	0.127398	0.353062	0.000712
...	...	...	...	...	...	...	...
171	0.000093	0.002115	0.005041	0.002880	0.083908	0.650543	0.000448
172	0.000397	0.005551	0.003777	0.001320	0.113525	0.468127	0.001020
173	0.000258	0.002662	0.001886	0.000997	0.092180	0.201172	0.000416
174	0.000233	0.003904	0.005026	0.003179	0.115082	0.625989	0.000691
175	0.000150	0.005762	0.004460	0.002486	0.114589	0.769496	0.000627

176 rows × 1024 columns

Do we get better clustering with this representation?

```
from sklearn.cluster import KMeans

k = 3
km = KMeans(n_clusters=k, n_init='auto', random_state=123)
km.fit(Z_birds)
```

KMeans

```
KMeans(n_clusters=3, random_state=123)
```

```
km.cluster_centers_.shape
```

```
(3, 1024)
```

```
for cluster in range(k):
    # user-defined functions defined in ../code/plotting_functions.py
    get_cluster_images(km, Z_birds, X_birds, cluster, n_img=6)
```

```
103
Image indices:  [103  23  86 162 168 122]
```

Cluster center 0

```
55
Image indices:  [55 31 53 15 88 84]
```

Cluster center 1

```
120
Image indices:  [120   5  11  14  22  69]
```

Cluster center 2

KMeans seems to be doing a good job. But cluster centers are not interpretable at all now. Let's try GMMs.

```
gmm = GaussianMixture(n_components=k, random_state=123)
gmm.fit(Z_birds)
```

```
GaussianMixture(n_components=3, random_state=123)
```

```
gmm.weights_
```

```
array([0.34090909, 0.32386364, 0.33522727])
```

```
for cluster in range(k):
    # user-defined functions defined in ../code/plotting_functions.py
    get_cluster_images(gmm, Z_birds, X_birds, cluster, n_img=6)
```

```
Image indices:  [107 106  42 103 135  87]
```

Cluster 0

```
Image indices:  [ 84  79  78  77 100 175]
```

Cluster 1

```
Image indices:  [137  61  28 141  25 124]
```

Cluster 2

Cool! Both models are doing a great job with this representation!! This dataset seems easier, as the birds have very distinct colors. Let's try a bit more complicated dataset.

```
data_dir = DATA_DIR + "food"
file_names = [image_file for image_file in glob.glob(data_dir + "/*/*.jpg")]
n_images = len(file_names)
BATCH_SIZE = n_images  # because our dataset is quite small
food_inputs, food_classes = read_img_dataset(data_dir)
n_images
```

```
350
```

```

```

```
X_food = food_inputs.numpy()
```

```
plot_sample_imgs(food_inputs[0:24,:,:,:])
```


```
Z_food = get_features(
    densenet, food_inputs,
)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[36], line 1
----> 1 Z_food = get_features(
      2     densenet, food_inputs,
      3 )

Cell In[23], line 5, in get_features(model, inputs)
      3 model.eval()
      4 with torch.no_grad():  # turn off computational graph stuff
----> 5     Z = model(inputs).detach().numpy()
      6 return Z

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/modu
   1734     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[m
   1735 else:
-> 1736     return self._call_impl(*args, **kwargs)

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/modu
   1742 # If we don't have any hooks, we want to skip the rest of the logic in
   1743 # this function, and just call forward.
   1744 if not (self._backward_hooks or self._backward_pre_hooks or self._forwa
   1745         or _global_backward_pre_hooks or _global_backward_hooks
   1746         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1747     return forward_call(*args, **kwargs)
   1749 result = None
   1750 called_always_called_hooks = set()

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torchvision/models/de
    212 def forward(self, x: Tensor) -> Tensor:
--> 213     features = self.features(x)
    214     out = F.relu(features, inplace=True)
    215     out = F.adaptive_avg_pool2d(out, (1, 1))

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/modu
   1734     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[m
   1735 else:
-> 1736     return self._call_impl(*args, **kwargs)

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/modu
   1742 # If we don't have any hooks, we want to skip the rest of the logic in
   1743 # this function, and just call forward.
   1744 if not (self._backward_hooks or self._backward_pre_hooks or self._forwa
   1745         or _global_backward_pre_hooks or _global_backward_hooks
   1746         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1747     return forward_call(*args, **kwargs)
   1749 result = None
   1750 called_always_called_hooks = set()

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/cont
    248 def forward(self, input):
    249     for module in self:
--> 250         input = module(input)
    251     return input
```

```
File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/modu
   1734     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[m
   1735 else:
-> 1736     return self._call_impl(*args, **kwargs)

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/modu
   1742 # If we don't have any hooks, we want to skip the rest of the logic in
   1743 # this function, and just call forward.
   1744 if not (self._backward_hooks or self._backward_pre_hooks or self._forwa
   1745         or _global_backward_pre_hooks or _global_backward_hooks
   1746         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1747     return forward_call(*args, **kwargs)
   1749 result = None
   1750 called_always_called_hooks = set()

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/cont
    248 def forward(self, input):
    249     for module in self:
--> 250         input = module(input)
    251     return input

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/modu
   1734     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[m
   1735 else:
-> 1736     return self._call_impl(*args, **kwargs)

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/modu
   1742 # If we don't have any hooks, we want to skip the rest of the logic in
   1743 # this function, and just call forward.
   1744 if not (self._backward_hooks or self._backward_pre_hooks or self._forwa
   1745         or _global_backward_pre_hooks or _global_backward_hooks
   1746         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1747     return forward_call(*args, **kwargs)
   1749 result = None
   1750 called_always_called_hooks = set()

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/conv
    553 def forward(self, input: Tensor) -> Tensor:
--> 554     return self._conv_forward(input, self.weight, self.bias)

File ~/miniforge3/envs/jbook/lib/python3.12/site-packages/torch/nn/modules/conv
    537 if self.padding_mode != "zeros":
    538     return F.conv2d(
    539         F.pad(
    540             input, self._reversed_padding_repeated_twice, mode=self.pad
   (...)
    547         self.groups,
    548     )
--> 549 return F.conv2d(
    550     input, weight, bias, self.stride, self.padding, self.dilation, self
    551 )

KeyboardInterrupt:
```

```
Z_food.shape
```

```
(350, 1024)
```

```
from sklearn.cluster import KMeans

k = 5
km = KMeans(n_clusters=k, n_init='auto', random_state=123)
km.fit(Z_food)
```

KMeans

```
KMeans(n_clusters=5, random_state=123)
```

```
km.cluster_centers_.shape
```

```
(5, 1024)
```

```
for cluster in range(k):
    get_cluster_images(km, Z_food, X_food, cluster, n_img=6)
```

```
84
Image indices:  [ 84 169 328   0 143  12]
```

Cluster center 0

```
263
Image indices:  [263  80 257 301  44 326]
```

Cluster center 1

```
188
Image indices:  [188   1 339 273  55 238]
```

Cluster center 2

```
282
Image indices:  [282 150 177 138 116 123]
```

Cluster center 3

```
20
Image indices:  [ 20  39 332  15 226 322]
```

Cluster center 4

There are some mis-classifications but overall it seems pretty good! You can experiment with

- Different values for number of clusters
- Different pre-trained models
- Other possible representations
- Different image datasets

See an example of using K-Means clustering on customer segmentation in AppendixA.