dm-1-asst-4

December 16, 2023

[61]:
```
import os
import numpy as np, pandas as pd
import keras

import torch
from torch import nn, optim
from torchvision import datasets, models, transforms

import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
data_directory = '/content/drive/MyDrive/Dogs_Dataset/images_cropped'
```

[3]:
```
true_labels = []
k = 0
for i in os.listdir(data_directory):
  for j in os.listdir(os.path.join(data_directory,i)):
    true_labels.append(k)
  k+=1
true_labels = np.array(true_labels)
```

Resize

[4]:
```
transforms = transforms.Compose([
                                 transforms.Resize(224),
                                 transforms.ToTensor(),
                                ])
```

[5]:
```
image_datasets = datasets.ImageFolder(data_directory, transform=transforms)
```

[6]:
```
dataloaders  = torch.utils.data.DataLoader(image_datasets, batch_size=1,␣
 ↪shuffle = False)
```

Normalize

[7]:
```
norm_data = []
for i in dataloaders:
  mean,std,var = torch.mean(i[0]),torch.std(i[0]),torch.var(i[0])
```

```
  t = (i[0]-mean)/std
  norm_data.append(t)
```

Extract Features

[8]:
```
resnet18 = models.resnet18(pretrained=True)
resnet18 = torch.nn.Sequential(*(list(resnet18.children())[:-1]))

resnet18.eval()
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to
/root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|      | 44.7M/44.7M [00:00<00:00, 75.4MB/s]
```

[8]:
```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (5): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (6): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (7): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
```

```
  )
  (8): AdaptiveAvgPool2d(output_size=(1, 1))
)
```

[13]:
```
features_extracted = []
for i in norm_data:
  with torch.no_grad():
    features = resnet18(i)
  features_extracted.append(features)
```

Dimension Reduction

[14]:
```
features_array = []
for i in features_extracted:
  output_tensor = torch.nn.functional.adaptive_avg_pool2d(i, (1, 1))
  output_tensor = output_tensor.view(512)
  output_tensor = np.array(output_tensor)
  features_array.append(output_tensor)
```

[15]:
```
features_array = np.array(features_array)
features_array.shape
```

[15]:
```
(679, 512)
```

Clustering

KMeans

[47]:
```
from sklearn.cluster import KMeans
kmeans_1 = KMeans(n_clusters=4, random_state=0, init="random", n_init = 'auto').
 ↪fit(features_array)
kmeans_2 = KMeans(n_clusters=4, random_state=0, init="k-means++", n_init =␣
 ↪'auto').fit(features_array)
```

[48]:
```
from sklearn.cluster import BisectingKMeans
kmeans_3 = BisectingKMeans(n_clusters=4,init='random').fit(features_array)
```

[49]:
```
from sklearn.cluster import SpectralClustering
spectral = SpectralClustering(n_clusters=4).fit(features_array)
```

DBSCAN

[50]:
```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=7, min_samples=2).fit(features_array)
labels = db.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print("Number of clusters:",n_clusters_)
```

```
Number of clusters: 4
```

For eps = 7 and min_samples = 2, n_clusters = 4

Agglomerative Clustering

[51]:
```
from sklearn.cluster import AgglomerativeClustering
ag_single = AgglomerativeClustering(n_clusters=4, linkage = 'single').
 ↪fit(features_array)
ag_complete = AgglomerativeClustering(n_clusters=4, linkage = 'complete').
 ↪fit(features_array)
ag_group = AgglomerativeClustering(n_clusters=4, linkage = 'average').
 ↪fit(features_array)
ag_ward = AgglomerativeClustering(n_clusters=4, linkage = 'ward').
 ↪fit(features_array)
```

Clustering Evaluation

[62]:
```
cluster_models =␣
 ↪[kmeans_1,kmeans_2,kmeans_3,spectral,db,ag_single,ag_complete,ag_group,ag_ward]
fms = {'KMeans-Random':0,
       'KMeans-kmeans++':0,
       'Bisecting KMeans':0,
       'Spectral Clustering':0,
       'DBSCAN':0,
       'Agglomerative Clustering - single':0,
       'Agglomerative Clustering - complete':0,
       'Agglomerative Clustering - average':0,
       'Agglomerative Clustering - ward':0}
ss = {'KMeans-Random':0,
       'KMeans-kmeans++':0,
       'Bisecting KMeans':0,
       'Spectral Clustering':0,
       'DBSCAN':0,
       'Agglomerative Clustering - single':0,
       'Agglomerative Clustering - complete':0,
       'Agglomerative Clustering - average':0,
       'Agglomerative Clustering - ward':0}
```

[63]:
```
from sklearn.metrics import fowlkes_mallows_score
from sklearn.metrics import silhouette_score
for i,j,k in zip(cluster_models,fms,ss):
  predicts = i.fit_predict(features_array)
  fm_score = fowlkes_mallows_score(true_labels,predicts)
  s_score = silhouette_score(np.array(true_labels).reshape(-1,1),predicts)
  fms[j] = fm_score
  ss[k] = s_score
```

Ranking best to worst on fowlkes-mallows_score

[64]:
```
sorted(fms.items(), key=lambda item: item[1])[::-1]
```

[64]:
```
[('KMeans-Random', 0.8408168530682044),
 ('Bisecting KMeans', 0.8095209072219244),
 ('Agglomerative Clustering - ward', 0.7940864447509993),
 ('Agglomerative Clustering - complete', 0.7748626407369926),
 ('KMeans-kmeans++', 0.677359269306421),
 ('Agglomerative Clustering - average', 0.6698996277210605),
 ('Agglomerative Clustering - single', 0.49711258309478007),
 ('Spectral Clustering', 0.49711258309478007),
 ('DBSCAN', 0.4936848010419327)]
```

Ranking best to worst on Silhouette score

[65]:
```
sorted(ss.items(), key=lambda item: item[1])[::-1]
```

[65]:
```
[('KMeans-Random', 0.7441528286165865),
 ('Bisecting KMeans', 0.6890862668884007),
 ('Agglomerative Clustering - ward', 0.6600248710194033),
 ('Agglomerative Clustering - complete', 0.6244148818913582),
 ('Agglomerative Clustering - average', 0.08755115006712988),
 ('KMeans-kmeans++', -0.12242462372178381),
 ('Agglomerative Clustering - single', -0.4641194424878871),
 ('Spectral Clustering', -0.4641194424878871),
 ('DBSCAN', -0.7414110878570999)]
```