# FeatureExtraction

April 4, 2022

# 1 Exercise of feature extraction from CNNs with PyTorch

### 1.0.1 1. First of all, you must load the CIFAR10 dataset, which is already available for download in the PyTorch library, from *torchvision.datasets*

[ ]:
```
%matplotlib inline
from matplotlib import pyplot as plt
import collections

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import models, datasets, transforms

torch.set_printoptions(edgeitems=2)
torch.manual_seed(123)

from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
import numpy as np
```

[ ]:
```
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']

data_path = '../cifar10_dataset/'
```

[ ]:
```
cifar10 = datasets.CIFAR10(
    data_path, train=True, download=True,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
../cifar10_dataset/cifar-10-python.tar.gz

  0%|          | 0/170498071 [00:00<?, ?it/s]

Extracting ../cifar10_dataset/cifar-10-python.tar.gz to ../cifar10_dataset/
```

[ ]:
```
cifar10_val = datasets.CIFAR10(
    data_path, train=False, download=True,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))
```

```
Files already downloaded and verified
```

### 1.0.2 2. You must select three (3) classes from the CIFAR10 dataset, and then you must extract all the training and validation samples corresponding to those three classes which exist in CIFAR10.

[ ]:
```
device = (torch.device('cuda') if torch.cuda.is_available()
          else torch.device('cpu'))
print(f"Training on device {device}.")
```

```
Training on device cuda.
```

[ ]:
```
label_map = {1:0, 8:1, 9:2}
class_names = ['automobile', 'ship', 'truck']
cifar3 = [(img.to(device=device), label_map[label])
          for img, label in cifar10
          if label in [1, 8, 9]]
cifar3_val = [(img.to(device=device), label_map[label])
              for img, label in cifar10_val
              if label in [1, 8, 9]]
```

### 1.0.3 3. You must load the pretrained VGG16 model and define a restricted model which contains all the layers of the VGG16 up to the last convolutional layer. Then you must find out and print the number of features that the restricted model extracts.

[ ]:
```
original_model = models.vgg16(pretrained=True).to(device=device)

print(original_model)

class VGG16LastConv(nn.Module):
```

```python
        def __init__(self):
            super(VGG16LastConv, self).__init__()
            self.features = nn.Sequential(
                # stop at last conv
                *list(original_model.features.children())[:-1]
            )
        def forward(self, x):
            x = self.features(x)
            return x

model = VGG16LastConv().to(device=device)

print(model)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
VGG16LastConv(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
        (25): ReLU(inplace=True)
        (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (27): ReLU(inplace=True)
        (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (29): ReLU(inplace=True)
      )
    )
```

[ ]:
```
img, _ = cifar3[0]
output = model(img.unsqueeze(0))
num_features = list(output.flatten().shape)[0]
num_features
```

[ ]: 2048

### 1.0.4 4. You must extract the features from the training and validation sets, and train four classifiers with the extracted features from the training set. The four classifiers are: Gaussian naïve Bayes, K-nearest neighbors, decision tree, and quadratic discriminant analysis.

[ ]:
```
train_loader = torch.utils.data.DataLoader(cifar3, batch_size=64,
                                           shuffle=True)  # <1>
val_loader = torch.utils.data.DataLoader(cifar3_val, batch_size=64,
                                         shuffle=False)
```

[ ]:
```
X_train = np.zeros((0,num_features))
y_train = np.zeros((0))
for imgs, labels in train_loader:  # <3>
            outputs = model(imgs)  # <4>

            original_output = outputs.detach().cpu().numpy().squeeze()
            batch_size = original_output.shape[0]
            reshaped_output = np.reshape(original_output, (batch_size,2048))

            X_train = np.concatenate((X_train,reshaped_output))
            y_train = np.concatenate((y_train, labels))
```

[ ]:
```
X_validation = np.zeros((0,num_features))
y_validation = np.zeros((0))
for imgs, labels in val_loader:  # <3>
            outputs = model(imgs)  # <4>

            original_output = outputs.detach().cpu().numpy().squeeze()
            batch_size = original_output.shape[0]
            reshaped_output = np.reshape(original_output, (batch_size,2048))
```

```
            X_validation = np.concatenate((X_validation,reshaped_output))
            y_validation = np.concatenate((y_validation, labels))
```

[ ]:
```
X_validation.shape
```

[ ]:
```
(3000, 2048)
```

[ ]:
```
y_train.shape
```

[ ]:
```
(15000,)
```

[ ]:
```
clf1 = GaussianNB()
clf1.fit(X_train, y_train)

clf2 = KNeighborsClassifier()
clf2.fit(X_train, y_train)

clf3 = DecisionTreeClassifier()
clf3.fit(X_train, y_train)

clf4 = QuadraticDiscriminantAnalysis()
clf4.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/discriminant_analysis.py:878:
UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
```

[ ]:
```
QuadraticDiscriminantAnalysis()
```

#### 1.0.5 5. You must measure the performance of the trained classifiers on the training and validation sets.

[ ]:
```
print(clf1.score(X_train,y_train))
print(clf1.score(X_validation,y_validation))
```

```
0.6412666666666667
0.6243333333333333
```

[ ]:
```
print(clf2.score(X_train,y_train))
print(clf2.score(X_validation,y_validation))
```

```
0.8528666666666667
0.77
```

[ ]:
```
print(clf3.score(X_train,y_train))
print(clf3.score(X_validation,y_validation))
```

```
1.0
0.683
```

[ ]:
```
print(clf4.score(X_train,y_train))
print(clf4.score(X_validation,y_validation))
```

```
0.7340666666666666
0.6236666666666667
```

### 1.0.6 In order to generate a good quality PDF, you may put the following code as the last cell of your notebook:

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py

from colab_pdf import colab_pdf

colab_pdf('FeatureExtraction.ipynb')
```

```
File 'colab_pdf.py' already there; not retrieving.

Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

[ ]:
```

```