TypeB_AntsBees_PyTorch_ModelClass_02

January 7, 2022

Binary classification model : AntsBees

Coding Stye: TypeB

Section: Model Class

[1]:
```
%pwd
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[2]:
```
import numpy as np
from PIL import Image
from torchvision import transforms
import matplotlib.pyplot as plt
import os
import torch
import torch.utils.data as data
from pathlib import Path
```

[3]:
```
data_dir = '/content/gdrive/My Drive/Colab Notebooks/AntsBees/data'
root_dir= '/content/gdrive/My Drive/Colab Notebooks/AntsBees/data/
 ↪hymenoptera_data'
```

[4]:
```
os.chdir('/content/gdrive/My Drive/Colab Notebooks/AntsBees')
```

1 *DataSet Class*

[5]:
```
from util.ImageTransform import ImageTransform
from dsets.dsets import make_path_list
from dsets.dsets import MakeBalancedDataset
from torch.utils.data import DataLoader


train_list = make_path_list(phase='train',root_dir=root_dir)
val_list = make_path_list(phase='val',root_dir=root_dir)
```

```
file_list={'train':train_list,'val':val_list}

SIZE = 224
# RGB
MEAN = (0.485, 0.456, 0.406) # ImageNet
# RGB
STD = (0.229, 0.224, 0.225)  # ImageNet


#
size, mean, std = SIZE, MEAN, STD

# MakeDataset
train_dataset = MakeBalancedDataset(
    file_list=file_list, #
    ratio_int=True,
    transform=ImageTransform(size, mean, std), #
    phase='train',records=300)
# MakeDataset
val_dataset = MakeBalancedDataset(
    file_list=file_list, #
    ratio_int=True,
    transform=ImageTransform(size, mean, std), #
    phase='val',records=200)


#
batch_size = 32

# :(, 3, 224, 224)
train_dl = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# :(, 3, 224, 224)
val_dl = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
```

2 *Model Class*

[6]:
```
# (CPUGPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

2.0.1 network function

[7]:
```
from models.networks import network_vgg16
```

[8]:
```
'''
7. VGG16
'''
from torchvision import models
import torch.nn as nn

class MakeModel():

  def␣
 ↪__init__(self,isTrain=True,continue_train=False,which_epoch=0,use_cuda=True,):
 ↪

    self.save_dir = 'network'
    self.log_dir = 'logmetrics'
    self.isTrain = isTrain
    self.continue_train = continue_train
    self.use_cuda = use_cuda
    self.which_epoch = which_epoch

    if self.isTrain:
      self.model = network_vgg16()
      self.optimizer = torch.optim.SGD(self.model.parameters(), lr=0.
 ↪001,momentum=0.99)

      if not self.continue_train:
        print('first training')

    if not self.isTrain or self.continue_train:
      self.load_network(self.model,self.which_epoch)
      print('continued train')

    if self.use_cuda:
      self.model.cuda()
      print('cuda')

  def load_network(self,network,which_epoch):
      save_filename = 'net_%s.pth' % (which_epoch)
      save_path = os.path.join(self.save_dir, save_filename)
      network.load_state_dict(torch.load(save_path))
      print('load network:',save_path)

  def train(self):
    self.model.train()
```

```
  def eval(self):
    self.model.eval()
```

[ ]:
```
model=MakeModel(isTrain=True,continue_train=False,which_epoch=0,use_cuda=True)
```

```
classifier.6.weight
classifier.6.bias
first training
cuda
```

[ ]:
```
print(model.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=2, bias=True)
  )
)
```

[ ]:
```
batch_iterator = iter(train_dl)
batch_tup = next(batch_iterator)

input_t, label_t = batch_tup

input_g=input_t.to(device)
label_g=label_t.to(device)
model = model.model.to(device)

outputs = model(input_g)

pred_label= outputs.argmax(dim=-1)
print(pred_label)
pred_label_list = pred_label.tolist()

print(pred_label_list)

print(outputs)
print(outputs.argmax(dim=-1).tolist())
print(label_t.tolist())
from sklearn.metrics import accuracy_score
print(accuracy_score(label_t.tolist(),pred_label.tolist()))
```

```
tensor([0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
        0, 0, 1, 1, 1, 1, 1, 1], device='cuda:0')
[0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
1, 1, 1, 1, 1, 1]
tensor([[ 0.5449, -0.0446],
        [ 0.5561, -0.3948],
        [ 0.3755, -0.4448],
        [ 0.0841, -0.5765],
```

```
        [-0.2710, -0.2093],
        [ 0.7786, -0.1600],
        [-0.2764, -0.0134],
        [ 1.7875, -1.0424],
        [ 0.3777, -0.1978],
        [-0.3500, -0.8331],
        [-0.0028, -0.6390],
        [ 1.0162,  0.0248],
        [ 0.0667, -0.2581],
        [ 0.3133, -0.0347],
        [-0.1829, -0.1037],
        [ 1.2638, -0.2564],
        [-0.4332,  0.7801],
        [-0.8082, -0.3675],
        [-0.5021, -0.5937],
        [ 0.9682, -0.7932],
        [-0.3518, -0.0567],
        [-0.7471, -0.2317],
        [ 0.1478, -0.4633],
        [-0.4366,  0.6627],
        [-0.1802, -0.2809],
        [ 0.3550,  0.1060],
        [-0.4606, -0.4347],
        [-0.2133,  0.2191],
        [-0.0816,  0.4309],
        [-0.1263,  0.4392],
        [-0.0159,  0.5338],
        [-0.8969, -0.6592]], device='cuda:0', grad_fn=<AddmmBackward0>)
[0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
1, 1, 1, 1, 1, 1]
[0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
1, 0, 1, 0, 1, 0]
0.5
```

[ ]:
```
softmax = nn.Softmax(dim=1)
prob_g=softmax(outputs)
print(prob_g[:,1])
```

```
tensor([0.3567, 0.2787, 0.3057, 0.3406, 0.5154, 0.2812, 0.5654, 0.0557, 0.3600,
        0.3815, 0.3461, 0.2706, 0.4195, 0.4139, 0.5198, 0.1794, 0.7709, 0.6084,
        0.4771, 0.1466, 0.5732, 0.6261, 0.3518, 0.7501, 0.4748, 0.4381, 0.5065,
        0.6065, 0.6254, 0.6377, 0.6341, 0.5592], device='cuda:0',
       grad_fn=<SelectBackward0>)
```

[ ]:
```
loss_func = nn.CrossEntropyLoss(reduction='none')
loss_g = loss_func(outputs,label_g).detach()
print(loss_g)
```

```
tensor([0.4412, 1.2776, 0.3648, 1.0770, 0.7245, 0.3302, 0.8332, 2.8872, 1.0218,
        0.9636, 1.0610, 1.3070, 0.8687, 0.5342, 0.6543, 1.7179, 0.2602, 0.4969,
        0.6484, 0.1585, 0.5565, 0.4683, 0.4336, 0.2875, 0.7448, 0.8254, 0.6803,
        0.9326, 0.4694, 1.0153, 0.4556, 0.8191], device='cuda:0')
```

3 END