Kaggle狗猫识别

# coding=utf-8  
import os  
import numpy as np  
import  torch  
import torch.nn as nn//torch.nn包含了用来搭建各个层的模块，一系列有用的loss函数，import<库名>as<库别名>  
import torch.nn.functional as F//torch.nn包定义网络有两种常用的方法1.继承nn.Module类的方式2.使用torch.nn.Sequential进行定义  
import torch.optim as optim//torch.optim包主要包含了用来更新参数的优化算法。  
from torch.autograd import Variable// torch.autograd 提供了实现任意标量值功能的自动化区分的类和功能。  
from torch.utils.data import Dataset//  
from torchvision import transforms, datasets, models  
  
# 随机种子设置  
random\_state = 42  
np.random.seed(random\_state)  
  
# kaggle原始数据集地址  
original\_dataset\_dir = &apos;E:\python ese\c\_d\data\\train\\train&apos;  
total\_num = int(len(os.listdir(original\_dataset\_dir)) / 2)  
random\_idx = np.array(range(total\_num))  
np.random.shuffle(random\_idx)  
  
# 待处理的数据集地址  
base\_dir = &apos;E:\python ese\c\_d\data2&apos;  
if not os.path.exists(base\_dir):  
    os.mkdir(base\_dir)  
  
# 训练集、测试集的划分  
sub\_dirs = [&apos;train&apos;, &apos;test&apos;]  
animals = [&apos;cats&apos;, &apos;dogs&apos;]  
train\_idx = random\_idx[:int(total\_num \* 0.9)]  
test\_idx = random\_idx[int(total\_num \* 0.9):]  
numbers = [train\_idx, test\_idx]  
for idx, sub\_dir in enumerate(sub\_dirs):  
    dir = os.path.join(base\_dir, sub\_dir)  
    if not os.path.exists(dir):  
        os.mkdir(dir)  
    for animal in animals:  
        animal\_dir = os.path.join(dir, animal)  #  
        if not os.path.exists(animal\_dir):  
            os.mkdir(animal\_dir)  
        fnames = [animal[:-1] + &apos;.{}.jpg&apos;.format(i) for i in numbers[idx]]  
        for fname in fnames:  
            src = os.path.join(original\_dataset\_dir, fname)  
            dst = os.path.join(animal\_dir, fname)  
            shutil.copyfile(src, dst)  
  
        # 验证训练集、验证集、测试集的划分的照片数目  
        print(animal\_dir + &apos; total images : %d&apos; % (len(os.listdir(animal\_dir))))  
    # coding=utf-8  
  
# 配置参数  
random\_state = 1  
torch.manual\_seed(random\_state)  # 设置随机数种子，确保结果可重复  
torch.cuda.manual\_seed(random\_state)  
torch.cuda.manual\_seed\_all(random\_state)  
np.random.seed(random\_state)  
# random.seed(random\_state)  
  
epochs = 10 # 训练次数  
batch\_size = 4  # 批处理大小  
num\_workers = 0  # 多线程的数目  
use\_gpu = torch.cuda.is\_available()  
PATH=&apos;E:\python ese\c\_d\model.pt&apos;  
# 对加载的图像作归一化处理， 并裁剪为[224x224x3]大小的图像  
data\_transform = transforms.Compose([  
    transforms.Resize(256),  
    transforms.CenterCrop(224),  
    transforms.ToTensor(),  
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  
])  
  
train\_dataset = datasets.ImageFolder(root=&apos;./data2/train/&apos;,  
                                     transform=data\_transform)  
train\_loader = torch.utils.data.DataLoader(train\_dataset,  
                                           batch\_size=batch\_size,  
                                           shuffle=True,  
                                           num\_workers=num\_workers)  
  
test\_dataset = datasets.ImageFolder(root=&apos;./data2/test/&apos;, transform=data\_transform)  
test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=True, num\_workers=num\_workers)  
  
  
# 创建模型  
class Net(nn.Module):  
    def \_\_init\_\_(self):  
        super(Net, self).\_\_init\_\_()  
        self.conv1 = nn.Conv2d(3, 6, 5)  
        self.maxpool = nn.MaxPool2d(2, 2)  
        self.conv2 = nn.Conv2d(6, 16, 5)  
        self.fc1 = nn.Linear(16 \* 53 \* 53, 1024)  
        self.fc2 = nn.Linear(1024, 512)  
        self.fc3 = nn.Linear(512, 2)  
  
    def forward(self, x):  
        x = self.maxpool(F.relu(self.conv1(x)))  
        x = self.maxpool(F.relu(self.conv2(x)))  
        x = x.view(-1, 16 \* 53 \* 53)  
        x = F.relu(self.fc1(x))  
        x = F.relu(self.fc2(x))  
        x = self.fc3(x)  
  
        return x  
  
  
net = Net()  
if(os.path.exists(&apos;model.pt&apos;)):  
    net=torch.load(&apos;model.pt&apos;)  
  
if use\_gpu:  
    net = net.cuda()  
print(net)  
  
# 定义loss和optimizer  
cirterion = nn.CrossEntropyLoss()  
optimizer = optim.SGD(net.parameters(), lr=0.0001, momentum=0.9)  
  
def train():  
  
    for epoch in range(epochs):  
        running\_loss = 0.0  
        train\_correct = 0  
        train\_total = 0  
        for i, data in enumerate(train\_loader, 0):  
            inputs, train\_labels = data  
            if use\_gpu:  
                inputs, labels = Variable(inputs.cuda()), Variable(train\_labels.cuda())  
            else:  
                inputs, labels = Variable(inputs), Variable(train\_labels)  
            optimizer.zero\_grad()  
            outputs = net(inputs)  
            \_, train\_predicted = torch.max(outputs.data, 1)  
            train\_correct += (train\_predicted == labels.data).sum()  
            loss = cirterion(outputs, labels)  
            loss.backward()  
            optimizer.step()  
            running\_loss += loss.item()  
            train\_total += train\_labels.size(0)  
  
        print(&apos;train %d epoch loss: %.3f  acc: %.3f &apos; % (  
            epoch + 1, running\_loss / train\_total, 100 \* train\_correct / train\_total))  
        # 模型测试  
        correct = 0  
        test\_loss = 0.0  
        test\_total = 0  
        test\_total = 0  
        net.eval()  
        for data in test\_loader:  
            images, labels = data  
            if use\_gpu:  
                images, labels = Variable(images.cuda()), Variable(labels.cuda())  
            else:  
                images, labels = Variable(images), Variable(labels)  
            outputs = net(images)  
            \_, predicted = torch.max(outputs.data, 1)  
            loss = cirterion(outputs, labels)  
            test\_loss += loss.item()  
            test\_total += labels.size(0)  
            correct += (predicted == labels.data).sum()  
  
        print(&apos;test  %d epoch loss: %.3f  acc: %.3f &apos; % (epoch + 1, test\_loss / test\_total, 100 \* correct / test\_total))  
  
    torch.save(net, &apos;model.pt&apos;)  
  
  
train()

验证码识别

import numpy as np //numpy是一个扩充程序库，支持高级大量的维度数组与矩阵运算，也针对数组运算提供大量的数学函数库。

|  |  |
| --- | --- |
|  | from skimage import io //io模块是用来操作图片输入输出的，skimage包对scipy.ndimage进行了扩展，提供了更多的图片处理功能。 |
|  | import os //os模块是python标准库中的一个用于访问操作系统功能的模块，os模块提供了一种可移植的方法是用操作系统的功能。可以实现跨平台访问。 |
|  | from keras.models import load\_model //keras.model实现神经网络模型的保存与打开。Load\_model使用该类将3DS max模型转换成SSM格式，然后用该类载入模型。 |
|  |  |
|  | def predict\_img(model\_path,img\_data): |
|  | model = load\_model(model\_path) |
|  | pred = model.predict(img\_data,batch\_size = 50) |
|  | outdict = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','0','1','2','3','4','5','6','7','8','9'] |
|  | f = open('mappings.txt','w') |
|  | for i in range(pred.shape[0]): |
|  | c0 = outdict[np.argmax(pred[i][:36])] |
|  | c1 = outdict[np.argmax(pred[i][36:36\*2])] |
|  | c2 = outdict[np.argmax(pred[i][36\*2:36\*3])] |
|  | c3 = outdict[np.argmax(pred[i][36\*3:36\*4])] |
|  | c4 = outdict[np.argmax(pred[i][36\*4:])] |
|  | c = c0+c1+c2+c3+c4 |
|  | n = np.str("{:0>4d}".format(i)) |
|  | f.write(n+','+c+'\r') |
|  | f.close() |
|  |  |
|  | def read\_data\_img(img\_path): |
|  | img\_file = os.listdir(img\_path) |
|  | img\_data = [] |
|  | for img in img\_file: |
|  | path = os.path.join(img\_path,img) |
|  | image = io.imread(path,0) |
|  | img\_data.append(image) |
|  | img\_data = np.array(img\_data) |
|  | img\_data = img\_data.reshape(img\_data.shape[0], 60, 200, 1) |
|  | img\_data = img\_data.astype('float32') |
|  | img\_data /= 255 |
|  | return img\_data |
|  |  |
|  | def main(): |
|  | path = os.getcwd() |
|  | model\_path = os.path.join(path,'my\_model\_data\_2.h5') |
|  | img\_path = os.path.join(path,'train\_change') |
|  | img\_data = read\_data\_img(img\_path) |
|  | predict\_img(model\_path,img\_data) |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |

**import** **spacy //spacy是一个自然语言处理工具包。**

**from** **sklearn.datasets** **import** fetch\_20newsgroups *# import packages which help us download dataset and load intp python //sklearn.datasets 生成数据。*

**from** **sklearn.pipeline** **import** Pipeline //跟踪记录个步骤的操作，对个步骤进行一个封装。

**import** **numpy** **as** **np** *# numpy package is for fast numerical computation in Python*

*# Load the filenames and data from the 20 newsgroups dataset*

*# This will download the datasets if you don't have them already*

twenty\_train = fetch\_20newsgroups(subset='train', shuffle=**True**, download\_if\_missing=**True**)

twenty\_test = fetch\_20newsgroups(subset='test', shuffle=**True**, download\_if\_missing=**True**)

*# What is this dataset about?*

print(twenty\_train.description)

twenty\_train.data[0]

twenty\_train.target\_names

*# Extracting features from text files*

**from** **sklearn.feature\_extraction.text** **import** CountVectorizer

count\_vect = CountVectorizer()//文本特征，库引用，from<库名>import<函数名>

X\_train\_counts = count\_vect.fit\_transform(twenty\_train.data)

print(f'Shape of Term Frequency Matrix: **{X\_train\_counts.shape}**')

**from** **sklearn.feature\_extraction.text** **import** TfidfTransformer

tfidf\_transformer = TfidfTransformer()

X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts)

print(f'Shape of TFIDF Matrix: **{X\_train\_tfidf.shape}**')

*# Machine Learning*

*# Training Naive Bayes (NB) classifier on training data.*

**from** **sklearn.naive\_bayes** **import** MultinomialNB

clf = MultinomialNB().fit(X\_train\_tfidf, twenty\_train.target)

*# Building a pipeline: We can write less code and do all of the above, by building a pipeline as follows:*

text\_nb\_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('clf', MultinomialNB())])

text\_nb\_clf = text\_nb\_clf.fit(twenty\_train.data, twenty\_train.target)

predicted = text\_nb\_clf.predict(twenty\_test.data)

naivebayes\_clf\_accuracy = np.mean(predicted == twenty\_test.target) \* 100.

print(f'Test Accuracy is **{naivebayes\_clf\_accuracy}** %')

**from** **sklearn.linear\_model** **import** LogisticRegression **as** LR

%%time

text\_lr\_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('clf',LR())])

text\_lr\_clf = text\_lr\_clf.fit(twenty\_train.data, twenty\_train.target)

lr\_predicted = text\_lr\_clf.predict(twenty\_test.data)

lr\_clf\_accuracy = np.mean(lr\_predicted == twenty\_test.target) \* 100.

print(f'Test Accuracy is **{lr\_clf\_accuracy}**')

15

**线性回归**

**import torch**

import torch.optim as optim //优化算法

import matplotlib.pyplot as plt //绘图

def get\_fake\_data(batch\_size=32)://batch\_size批处理参数

''' y=x\*2+3 '''

x = torch.randn(batch\_size, 1) \* 20

y = x \* 2 + 3 + torch.randn(batch\_size, 1)

return x, y

x, y = get\_fake\_data()//调用函数

class LinerRegress(torch.nn.Module)://定义一个类

def \_\_init\_\_(self)://初始化

super(LinerRegress, self).\_\_init\_\_()//使用父类的构造方法的功能，调用超类

self.fc1 = torch.nn.Linear(1, 1)

def forward(self, x)://向前传播函数

return self.fc1(x)

net = LinerRegress()

loss\_func = torch.nn.MSELoss()//损失函数

optimzer = optim.SGD(net.parameters())//优化函数

for i in range(40000):

optimzer.zero\_grad()

out = net(x)//输出层

loss = loss\_func(out, y)

loss.backward()//反向求导函数

optimzer.step()

w, b = [param.item() for param in net.parameters()]//获得网络参数

print w, b # 2.01146, 3.184525

# 显示原始点与拟合直线

plt.scatter(x.squeeze().numpy(), y.squeeze().numpy())//squeeze()移除维度为一的维度，plt.scatter画散点图，pd.scatter\_matrix画矩阵散点图。

plt.plot(x.squeeze().numpy(), (x\*w + b).squeeze().numpy())//绘图，对图形进行一些更改

plt.show()//显示图像