goolenet

笔记本:	深度学习-图像分类篇		
创建时间:	2023/4/5 21:19	**更新时间:**	2023/4/5 21:45
作者:	Kaiyuecui		
URL:	https://mp.weixin.qq.com/s?__biz=MzU1NjEwMTY0Mw==&mid=2247515761&idx=1...		

1、创新点

①使用了Inception模块（融合了不同尺度的特征信息）
注意Inception模块不同分支输出的长和高必相同，只是通道数不同而已
②使用了1*1卷积进行降维处理，减少了参数
③使用了两个辅助分类器
在GoogLeNet的结构中，我们可以发现共有3个softmax层，这是为了避免梯度消失，作者们额外增加了2个辅助的softmax（softmax0、softmax1）用于向前传导梯度。辅助分类器是将中间某一层的输出用作分类，并按一个较小的权重（比如0.3）加到最终分类结果中，这样相当于做了模型融合，同时给网络增加了反向传播的梯度信号，也提供了额外的正则化，对于整个网络的训练很有好处。而在实际测试的时候，这两个额外的softmax会被去掉。

2、有关dropout的坑

注意:
在model.eval()的情况下，只会不启动继承自nn.Modeul模块下的类。比如nn.Dropout()
但是对于nn.functional.dropout()方法，其并不继承自nn.Modeul，在eval()情况下，也不会使dropout失效。所以我们可以这样写nn.functional.dropout(x,p=,training=self.training),如果是在训练模式中则为TRUE，启动，反之失效。

3、
对于每一个不同的类别，我们其实都可以使用一个新的类来进行封装
比如这里的Inception 模块以及Inceptinaux 辅助分类器

4、代码:
①模型

```
import torch.nn as nn
import torch
import torch.nn.functional as F

class GoogLeNet(nn.Module):
    def __init__(self, num_classes=1000, aux_logits=True,
init_weights=False):
        super(GoogLeNet, self).__init__()
        self.aux_logits = aux_logits


        self.conv1 = BasicConv2d(3, 64, kernel_size=7, stride=2,
padding=3)
        self.maxpool1 = nn.MaxPool2d(3, stride=2, ceil_mode=True)
```

```
        self.conv2 = BasicConv2d(64, 64, kernel_size=1)
        self.conv3 = BasicConv2d(64, 192, kernel_size=3, padding=1)
        self.maxpool2 = nn.MaxPool2d(3, stride=2, ceil_mode=True)


        self.inception3a = Inception(192, 64, 96, 128, 16, 32, 32)
        self.inception3b = Inception(256, 128, 128, 192, 32, 96, 64)
        self.maxpool3 = nn.MaxPool2d(3, stride=2, ceil_mode=True)


        self.inception4a = Inception(480, 192, 96, 208, 16, 48, 64)
        self.inception4b = Inception(512, 160, 112, 224, 24, 64, 64)
        self.inception4c = Inception(512, 128, 128, 256, 24, 64, 64)
        self.inception4d = Inception(512, 112, 144, 288, 32, 64, 64)
        self.inception4e = Inception(528, 256, 160, 320, 32, 128, 128)
        self.maxpool4 = nn.MaxPool2d(3, stride=2, ceil_mode=True)


        self.inception5a = Inception(832, 256, 160, 320, 32, 128, 128)
        self.inception5b = Inception(832, 384, 192, 384, 48, 128, 128)


        if self.aux_logits:
            self.aux1 = InceptionAux(512, num_classes)
            self.aux2 = InceptionAux(528, num_classes)


        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.dropout = nn.Dropout(0.4)
        self.fc = nn.Linear(1024, num_classes)
        if init_weights:
            self._initialize_weights()


    def forward(self, x):
        # N x 3 x 224 x 224
        x = self.conv1(x)
        # N x 64 x 112 x 112
        x = self.maxpool1(x)
        # N x 64 x 56 x 56
        x = self.conv2(x)
        # N x 64 x 56 x 56
        x = self.conv3(x)
        # N x 192 x 56 x 56
        x = self.maxpool2(x)


        # N x 192 x 28 x 28
        x = self.inception3a(x)
        # N x 256 x 28 x 28
        x = self.inception3b(x)
        # N x 480 x 28 x 28
        x = self.maxpool3(x)
        # N x 480 x 14 x 14
        x = self.inception4a(x)
        # N x 512 x 14 x 14
        if self.training and self.aux_logits:    # eval model lose
this layer
```

```
            aux1 = self.aux1(x)


        x = self.inception4b(x)
        # N x 512 x 14 x 14
        x = self.inception4c(x)
        # N x 512 x 14 x 14
        x = self.inception4d(x)
        # N x 528 x 14 x 14
        if self.training and self.aux_logits:    # eval model lose
this layer
            aux2 = self.aux2(x)


        x = self.inception4e(x)
        # N x 832 x 14 x 14
        x = self.maxpool4(x)
        # N x 832 x 7 x 7
        x = self.inception5a(x)
        # N x 832 x 7 x 7
        x = self.inception5b(x)
        # N x 1024 x 7 x 7


        x = self.avgpool(x)
        # N x 1024 x 1 x 1
        x = torch.flatten(x, 1)
        # N x 1024
        x = self.dropout(x)
        x = self.fc(x)
        # N x 1000 (num_classes)
        if self.training and self.aux_logits:   # eval model lose this
layer
            return x, aux2, aux1
        return x


    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)




class Inception(nn.Module):
    def __init__(self, in_channels, ch1x1, ch3x3red, ch3x3, ch5x5red,
ch5x5, pool_proj):
        super(Inception, self).__init__()


        self.branch1 = BasicConv2d(in_channels, ch1x1, kernel_size=1)
```

```
        self.branch2 = nn.Sequential(
            BasicConv2d(in_channels, ch3x3red, kernel_size=1),
            BasicConv2d(ch3x3red, ch3x3, kernel_size=3, padding=1)   #
保证输出大小等于输入大小
        )


        self.branch3 = nn.Sequential(
            BasicConv2d(in_channels, ch5x5red, kernel_size=1),
            # 在官方的实现中，其实是3x3的kernel并不是5x5，这里我也懒得改
了，具体可以参考下面的issue
            # Please see https://github.com/pytorch/vision/issues/906
for details.
            BasicConv2d(ch5x5red, ch5x5, kernel_size=5, padding=2)   #
保证输出大小等于输入大小
        )


        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1),
            BasicConv2d(in_channels, pool_proj, kernel_size=1)
        )


    def forward(self, x):
        branch1 = self.branch1(x)
        branch2 = self.branch2(x)
        branch3 = self.branch3(x)
        branch4 = self.branch4(x)


        outputs = [branch1, branch2, branch3, branch4]
        return torch.cat(outputs, 1)




class InceptionAux(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(InceptionAux, self).__init__()
        self.averagePool = nn.AvgPool2d(kernel_size=5, stride=3)
        self.conv = BasicConv2d(in_channels, 128, kernel_size=1)  #
output[batch, 128, 4, 4]
        self.fc1 = nn.Linear(2048, 1024)
        self.fc2 = nn.Linear(1024, num_classes)


    def forward(self, x):
        # aux1: N x 512 x 14 x 14, aux2: N x 528 x 14 x 14
        x = self.averagePool(x)
        # aux1: N x 512 x 4 x 4, aux2: N x 528 x 4 x 4
        x = self.conv(x)
        # N x 128 x 4 x 4
        x = torch.flatten(x, 1)
        x = F.dropout(x, 0.5, training=self.training)
        # N x 2048
        x = F.relu(self.fc1(x), inplace=True)
        x = F.dropout(x, 0.5, training=self.training)
        # N x 1024
        x = self.fc2(x)
```

```
        # N x num_classes
        return x
class BasicConv2d(nn.Module):
    def __init__(self, in_channels, out_channels, **kwargs):
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, **kwargs)
        self.relu = nn.ReLU(inplace=True)


    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        return x
```

②训练

```
import os
import sys
import json


import torch
import torch.nn as nn
from torchvision import transforms, datasets
import torch.optim as optim
from tqdm import tqdm


from model import GoogLeNet




def main():
    device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
    print("using {} device.".format(device))


    data_transform = {
        "train":
transforms.Compose([transforms.RandomResizedCrop(224),
                                     transforms.RandomHorizontalFlip(),
                                     transforms.ToTensor(),
                                     transforms.Normalize((0.5, 0.5,
0.5), (0.5, 0.5, 0.5))]),
        "val": transforms.Compose([transforms.Resize((224, 224)),
                                   transforms.ToTensor(),
                                   transforms.Normalize((0.5, 0.5,
0.5), (0.5, 0.5, 0.5))])}


    data_root = os.path.abspath(os.path.join(os.getcwd(), "../.."))  #
get data root path
    image_path = os.path.join(data_root, "data_set", "flower_data")  #
flower data set path
    assert os.path.exists(image_path), "{} path does not
exist.".format(image_path)
    train_dataset = datasets.ImageFolder(root=os.path.join(image_path,
"train"),
                                         transform=data_transform["train"])
```

```
    train_num = len(train_dataset)


    # {'daisy':0, 'dandelion':1, 'roses':2, 'sunflower':3, 'tulips':4}
    flower_list = train_dataset.class_to_idx
    cla_dict = dict((val, key) for key, val in flower_list.items())
    # write dict into json file
    json_str = json.dumps(cla_dict, indent=4)
    with open('class_indices.json', 'w') as json_file:
        json_file.write(json_str)


    batch_size = 16
    nw = min([os.cpu_count(), batch_size if batch_size > 1 else 0,
8])  # number of workers
    print('Using {} dataloader workers every process'.format(nw))


    train_loader = torch.utils.data.DataLoader(train_dataset,
                                               batch_size=batch_size,
shuffle=True,
                                               num_workers=0)


    validate_dataset =
datasets.ImageFolder(root=os.path.join(image_path, "val"),
                                            transform=data_transform["val"])
    val_num = len(validate_dataset)
    validate_loader = torch.utils.data.DataLoader(validate_dataset,
                                                  batch_size=batch_size,
shuffle=False,
                                                  num_workers=0)


    print("using {} images for training, {} images for
validation.".format(train_num,
                                                                         val_num))


    # test_data_iter = iter(validate_loader)
    # test_image, test_label = test_data_iter.next()


    net = GoogLeNet(num_classes=5, aux_logits=True, init_weights=True)
    # 如果要使用官方的预训练权重，注意是将权重载入官方的模型，不是我们自己实
现的模型
    # 官方的模型中使用了bn层以及改了一些参数，不能混用
    # import torchvision
    # net = torchvision.models.googlenet(num_classes=5)
    # model_dict = net.state_dict()
    # # 预训练权重下载地址:
https://download.pytorch.org/models/googlenet-1378be20.pth
    # pretrain_model = torch.load("googlenet.pth")
    # del_list = ["aux1.fc2.weight", "aux1.fc2.bias",
    #             "aux2.fc2.weight", "aux2.fc2.bias",
    #             "fc.weight", "fc.bias"]
    # pretrain_dict = {k: v for k, v in pretrain_model.items() if k
not in del_list}
    # model_dict.update(pretrain_dict)
    # net.load_state_dict(model_dict)
```

```
    net.to(device)
    loss_function = nn.CrossEntropyLoss()
    optimizer = optim.Adam(net.parameters(), lr=0.0003)


    epochs = 30
    best_acc = 0.0
    save_path = './googleNet.pth'
    train_steps = len(train_loader)
    for epoch in range(epochs):
        # train
        net.train()
        running_loss = 0.0
        train_bar = tqdm(train_loader, file=sys.stdout)
        for step, data in enumerate(train_bar):
            images, labels = data
            optimizer.zero_grad()
            logits, aux_logits2, aux_logits1 = net(images.to(device))
            loss0 = loss_function(logits, labels.to(device))
            loss1 = loss_function(aux_logits1, labels.to(device))
            loss2 = loss_function(aux_logits2, labels.to(device))
            loss = loss0 + loss1 * 0.3 + loss2 * 0.3
            loss.backward()
            optimizer.step()


            # print statistics
            running_loss += loss.item()


            train_bar.desc = "train epoch[{}/{}] loss:
{:.3f}".format(epoch + 1,
                                                                     epochs,
                                                                     loss)


        # validate
        net.eval()
        acc = 0.0  # accumulate accurate number / epoch
        with torch.no_grad():
            val_bar = tqdm(validate_loader, file=sys.stdout)
            for val_data in val_bar:
                val_images, val_labels = val_data
                outputs = net(val_images.to(device))  # eval model
only have last output layer
                predict_y = torch.max(outputs, dim=1)[1]
                acc += torch.eq(predict_y,
val_labels.to(device)).sum().item()


        val_accurate = acc / val_num
        print('[epoch %d] train_loss: %.3f  val_accuracy: %.3f' %
              (epoch + 1, running_loss / train_steps, val_accurate))


        if val_accurate > best_acc:
            best_acc = val_accurate
            torch.save(net.state_dict(), save_path)
```

```
    print('Finished Training')



if __name__ == '__main__':
    main()
```

③预测

```
import os
import json


import torch
from PIL import Image
from torchvision import transforms
import matplotlib.pyplot as plt


from model import GoogLeNet




def main():
    device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")


    data_transform = transforms.Compose(
        [transforms.Resize((224, 224)),
         transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


    # load image
    img_path = "../tulips.jpg"
    assert os.path.exists(img_path), "file: '{}' dose not
exist.".format(img_path)
    img = Image.open(img_path)
    plt.imshow(img)
    # [N, C, H, W]
    img = data_transform(img)
    # expand batch dimension
    img = torch.unsqueeze(img, dim=0)


    # read class_indict
    json_path = './class_indices.json'
    assert os.path.exists(json_path), "file: '{}' dose not
exist.".format(json_path)


    with open(json_path, "r") as f:
        class_indict = json.load(f)


    # create model
    model = GoogLeNet(num_classes=5, aux_logits=False).to(device)
```

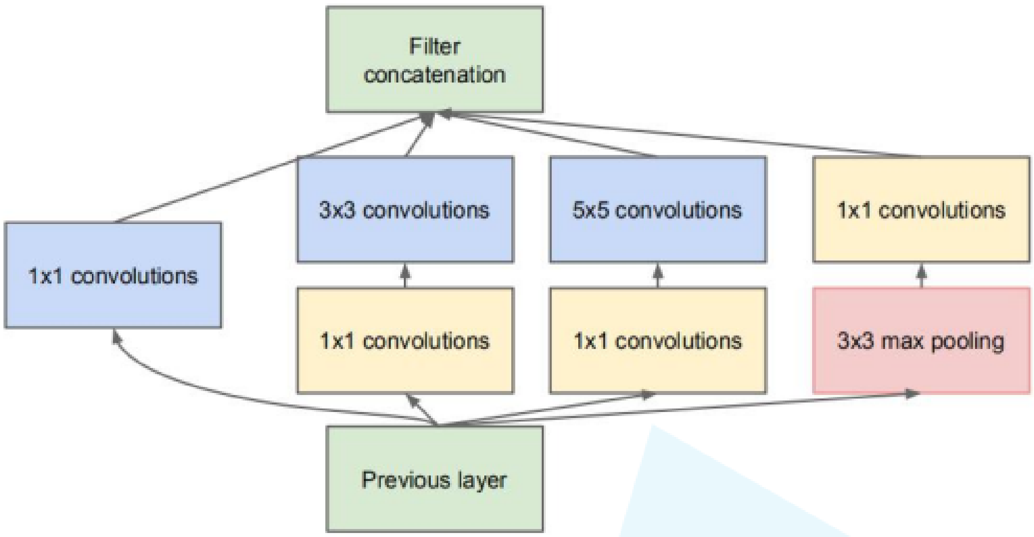
```
    # load model weights
    weights_path = "./googleNet.pth"
    assert os.path.exists(weights_path), "file: '{}' dose not
exist.".format(weights_path)
    missing_keys, unexpected_keys =
model.load_state_dict(torch.load(weights_path, map_location=device),
                                                          strict=False)
    print(unexpected_keys)

    model.eval()
    with torch.no_grad():
        # predict class
        output = torch.squeeze(model(img.to(device))).cpu()
        predict = torch.softmax(output, dim=0)
        predict_cla = torch.argmax(predict).numpy()
    print(predict_cla)



if __name__ == '__main__':
    main()
```

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								



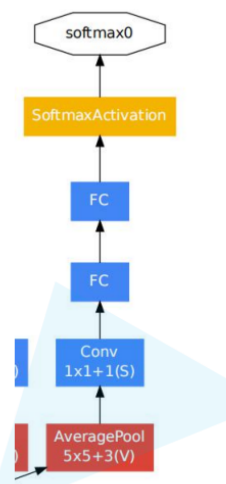
(b) Inception module with dimension reductions

注意：每个分支所得的特征矩阵高和宽必须相同

辅助分类器（Auxiliary Classifier）

The exact structure of the extra network on the side, including the auxiliary classifier, is as follows:

- An average pooling layer with 5×5 filter size and stride 3, resulting in an $4\times4\times512$ output for the (4a), and $4\times4\times528$ for the (4d) stage.
- A 1×1 convolution with 128 filters for dimension reduction and rectified linear activation.
- A fully connected layer with 1024 units and rectified linear activation.
- A dropout layer with 70% ratio of dropped outputs.
- A linear layer with softmax loss as the classifier (predicting the same 1000 classes as the main classifier, but removed at inference time).



$$out_{size} = (in_{size} - F_{size} + 2P)/S + 1$$