

Vgg

笔记本：深度学习-图像分类篇

创建时间：2023/4/4 22:15

更新时间：2023/4/5 17:39

作者：Kaiyuecui

URL：about:blank

1、创新之处：

使用多个3*3的卷积核来代替5*5或者7*7卷积，可以减少参数。

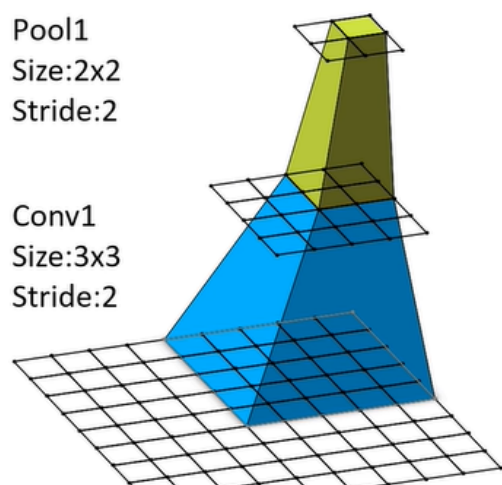
使用7x7卷积核所需参数，与堆叠三个3x3卷积核所需参数(假设输入输出channel为C)

$$7 \times 7 \times C \times C = 49C^2$$

$$3 \times 3 \times C \times C + 3 \times 3 \times C \times C + 3 \times 3 \times C \times C = 27C^2$$

2、感受野的概念：

如本图中输出特征中的一个单元，就对应输入特征中的一个5*5的区域的大小。



感受野计算公式：

$$F(i) = (F(i+1) - 1) \times \text{Stride} + \text{Ksize}$$

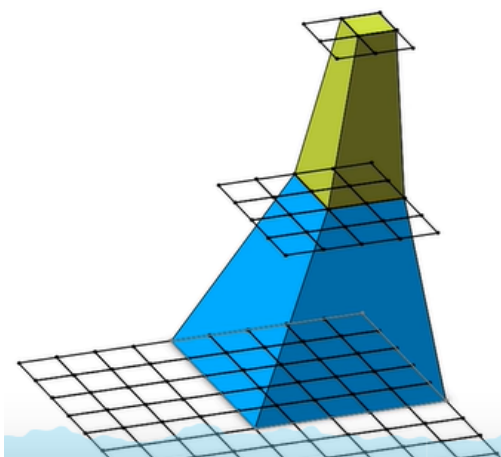
F(i)为第i层感受野，
Stride为第i层的步距，
Ksize为卷积核或池化核尺寸

Feature map: F = 1

$$\text{Pool1: } F = (1 - 1) \times 2 + 2 = 2$$

$$\text{Conv1: } F = (2 - 1) \times 2 + 3 = 5$$

三个3*3的卷积核与1个7*7的卷积核拥有相同的感受野。（vgg论文中s=2，k=3）



感受野计算公式：

$$F(i) = (F(i+1) - 1) \times \text{Stride} + \text{Ksize}$$

F(i)为第i层感受野，
Stride为第i层的步距，
Ksize为卷积核或采样核尺寸

Feature map: F = 1

$$\text{Conv3x3(3): } F = (1 - 1) \times 1 + 3 = 3$$

$$\text{Conv3x3(2): } F = (3 - 1) \times 1 + 3 = 5$$

$$\text{Conv3x3(1): } F = (5 - 1) \times 1 + 3 = 7$$

感受野就是输出特征中的一个元素对应输入层上的区域的大小。

3、代码：

①模型代码:

```
import torch.nn as nn
import torch

# official pretrain weights
model_urls = {
    'vgg11': 'https://download.pytorch.org/models/vgg11-bbd30ac9.pth',
    'vgg13': 'https://download.pytorch.org/models/vgg13-c768596a.pth',
    'vgg16': 'https://download.pytorch.org/models/vgg16-397923af.pth',
    'vgg19': 'https://download.pytorch.org/models/vgg19-dcbb9e9d.pth'
}

class VGG(nn.Module):
    def __init__(self, features, num_classes=1000, init_weights=False):
        super(VGG, self).__init__()
        self.features = features
        self.classifier = nn.Sequential(
            nn.Linear(512*7*7, 4096),
            nn.ReLU(True),
            nn.Dropout(p=0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(p=0.5),
            nn.Linear(4096, num_classes)
        )
        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        # N x 3 x 224 x 224
        x = self.features(x)
        # N x 512 x 7 x 7
        x = torch.flatten(x, start_dim=1)
        # N x 512*7*7
        x = self.classifier(x)
        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')
                nn.init.xavier_uniform_(m.weight)
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)
                # nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)

#cfg 是一个列表类型
def make_features(cfg: list):
```

```

layers = []
in_channels = 3
for v in cfg:
    if v == "M":
        layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
    else:
        conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
        layers += [conv2d, nn.ReLU(True)]
        in_channels = v
return nn.Sequential(*layers) #*可理解为解包操作，把layers列表中的元素拆分成一个个元素
#nn.Sequential 可以接受两种类型
①比如model = nn.Sequential(
    nn.Conv2d(1,20,5),
    nn.ReLU(),
    nn.Conv2d(20,64,5),
    nn.ReLU()
)

②model = nn.Sequential(OrderedDict([
    ('conv1', nn.Conv2d(1,20,5)),
    ('relu1', nn.ReLU()),
    ('conv2', nn.Conv2d(20,64,5)),
    ('relu2', nn.ReLU())
]))

cfgs = {
    'vgg11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'vgg13': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'vgg16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 'M'],
    'vgg19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 'M'],
}

def vgg(model_name="vgg16", **kwargs):
    assert model_name in cfgs, "Warning: model number {} not in cfgs dict!".format(model_name)
    cfg = cfgs[model_name]
    model = VGG(make_features(cfg), **kwargs)
    return model

```

②训练:

```

import os
import sys
import json
import torch, gc

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

```

```

import torch.optim as optim
from tqdm import tqdm

from model import vgg

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

    gc.collect()
    torch.cuda.empty_cache()

    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 = 4
    nw = 0 # number of workers
    print('Using {} dataloader workers every process'.format(nw))

    train_loader = torch.utils.data.DataLoader(train_dataset,

```

```

shuffle=True,
batch_size=batch_size,
num_workers=nw)

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=nw)

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()

model_name = "vgg11"
net = vgg(model_name=model_name, num_classes=5, init_weights=True)
net.to(device)
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.0001)

epochs = 10
best_acc = 0.0
save_path = './{}Net.pth'.format(model_name)
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()
        outputs = net(images.to(device))
        loss = loss_function(outputs, labels.to(device))
        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))
        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:

```

# 时间: 2023/4/5 17:38
# cky
import os
import json
import torch
from PIL import Image
from torchvision import transforms
import matplotlib.pyplot as plt

from model import vgg

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 = "../tulip.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 = vgg(model_name="vgg16", num_classes=5).to(device)
# load model weights
weights_path = "./vgg16Net.pth"
assert os.path.exists(weights_path), "file: '{} ' dose not
exist.".format(weights_path)
model.load_state_dict(torch.load(weights_path, map_location=device))

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_res = "class: {}    prob:
{: .3}".format(class_indict[str(predict_cla)],
                predict[predict_cla].numpy())

plt.title(print_res)
for i in range(len(predict)):
    print("class: {:10}    prob: {: .3}".format(class_indict[str(i)],
        predict[i].numpy()))

plt.show()

if __name__ == '__main__':
    main()

```

