**models.vgg19(pretrained)**

**VGG19网络结构**

vgg\_model = models.vgg19(pretrained=True)

print(vgg\_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): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(17): ReLU(inplace=True)

(18): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(19): Conv2d(256, 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): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(24): ReLU(inplace=True)

(25): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(26): ReLU(inplace=True)

(27): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(28): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(29): ReLU(inplace=True)

(30): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(31): ReLU(inplace=True)

(32): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(33): ReLU(inplace=True)

(34): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(35): ReLU(inplace=True)

(36): 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)

)

)

获取各个部分名称

vgg\_model.\_modules.keys()*#vgg\_model.\_modules返回一个以部分名称和部分结构为键值对的字典*

>>>

odict\_keys(['features', 'avgpool', 'classifier'])

models.vgg19(pretrained).features表示只选取VGG19的feature部分

vgg\_features = vgg\_model.features

print(type(vgg\_model))

print(type(vgg\_features))

lst = list(vgg\_features)*#生成由VGG19features部分各层网络类实例组成的列表*

print(lst[0])

print(lst[0].weight.size())

print(type(lst[0]))

>>>  
<class 'torchvision.models.vgg.VGG'>

<class 'torch.nn.modules.container.Sequential'>

Conv2d(3, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

torch.Size([64, 3, 3, 3])

<class 'torch.nn.modules.conv.Conv2d'>

**VGG19网络的输入输出**

输入图像宽高不一定是224×224

print(vgg\_model(style\_img).size())*#接下面代码，style\_img形状是1×3×128×128*

>>>

torch.Size([1, 3, 128, 128])

print(vgg\_features(style\_img).size())

>>>

torch.Size([1, 512, 4, 4])