**代码实现总结**

报告撰写人：王自全

# 【模型结构搭建】

# 【搭建卷积神经网络Backbone】

要求根据网络的结构，在PyTorch中把模型搭建起来。

代码地址：https://github.com/Ares-debugger/PaperReproduction

## 【AlexNet】

特征提取器：**卷积+Relu+池化；分类器Dropout+Linear+Relu**

class AlexNet(nn.Module):

def \_\_init\_\_(self,classes=1000):

super(AlexNet, self).\_\_init\_\_()

self.features = nn.Sequential(

nn.Conv2d(3,64,kernel\_size=11,stride=4,padding=2),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=3,stride=2),

nn.Conv2d(64,192,kernel\_size=3,padding=2),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=3,stride=2),

nn.Conv2d(192,384,kernel\_size=3,stride=2,padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=3, stride=2),

nn.Conv2d(384,256,kernel\_size=3,stride=2,padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=3, stride=2),

nn.Conv2d(256,256,kernel\_size=3,stride=2,padding=1),

nn.ReLU(inplace=True)，

nn.MaxPool2d(kernel\_size=3,stride=2)

)

self.avgpool = nn.AdaptiveAvgPool2d((6,6)),

self.classifier = nn.Sequential(

nn.Dropout(),

nn.Linear(6\*6\*256,4096),

nn.Relu(inplace=True),

nn.Dropout(),

nn.Linear(4096,4096),

nn.Relu(inplace=True),

nn.Linear(4096,classes),

)

def forward(self,x):

x = self.features(x)

x = self.avgpool(x)

x = torch.flatten(x,1),

x = self.classifier(x)

return x

## 【VGG】

加深了网络结构，可以照葫芦画瓢的写出来：见VGG16\_origin\_arch.py

但这个是相当傻的，并且VGG开始，模型不断堆叠，可以考虑使用cfg和循环创建模型了。代码见VGG\_archs.py

典型的自顶向下编程，\_\_init\_\_函数中只保留线性层和avgpool层，features层从传入构建：

class VGG(nn.Module):

**# 只保留通用的部分，features需要根据cfg构造，自顶向下编程思想**

def \_\_init\_\_(self,features,classes=1000):

super(VGG, self).\_\_init\_\_()

self.features = features

self.avgpool = nn.AdaptiveAvgPool2d((6,6))

self.classifier = nn.Sequential(

nn.Dropout(p=0.2),

nn.Linear(6\*6\*512,4096),

nn.ReLU(inplace=True),

nn.Dropout(p=0.2),

nn.Linear(4096,4096),

nn.ReLU(inplace=True),

nn.Linear(4096,classes)

)

def forward(self,x): pass

0、配置cfgs：

cfgs = [

‘16\_layers’:[…’M’….’M’…..]

‘19\_layers’:[…’M’….’M’…..]

]

**1、顶部：构建4个函数，使用\_vgg做为通用的函数**

def vgg16():

return \_vgg('16\_layer',batch\_norm=False)

def vgg19():

return \_vgg('16\_layer',batch\_norm=False)

def vgg16\_bn():

return \_vgg('19\_layer',batch\_norm=True)

def vgg19\_bn():

return \_vgg('19\_layer',batch\_norm=True)

**2、构建VGG模型的通用函数：写\_make\_layers的通用函数**

def \_vgg(cfgs[cfg],batch\_norm):

rreturn VGG(\_make\_layers(cfgs[cfg],batch\_norm)) –>生成不同的函数

**3、写\_make\_layers函数**：

def \_make\_layers(cfg,batch\_norm):

layers = [] # nn.squential 允许以一个\*列表的方式传入，这是很重要的

in\_channels = 3 # 初始化！！初始化！！

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,stride=1,padding=1)

if batch\_norm:

layers += [conv2d,nn.BatchNorm2d(v),nn.Relu(inplace=True)]

else:

layers += [conv2d,nn.Relu(inplace=True)]

in\_channels = v # 更新输入的通道

## 【GoogleNetv1】

## 【GoogleNetv2】

## 【GoogleNetv3】

## 【GoogleNetv4】

## 【ResNet】

## 【denseNet】

## 【ResNeXt】

# 【搭建目标检测部件】

## 【backbone调用】

## 【SPP】

## 【FPN】·

# 【搭建GAN部件】

## 【DCGAN】