**1.1 导入库**

import os

import shutil

import numpy as np

from paddle.io import Dataset,DataLoader

from paddle.vision import transforms as T

from paddle.nn import functional as F

import cv2

import paddle

import matplotlib.pyplot as plt

import paddle.nn as nn

from tqdm import tqdm

**1.2 定义超参**

#验证集的数量

eval\_num=1000

#所有图像的大小

image\_size=(224,224)

#训练图片路径

train\_images\_path="data/train\_image"

#标签图像路径

label\_images\_path="data/train\_50k\_mask"

#测试图片路径

test\_images\_path="data/val\_image"

#批量大小

batch\_size=4

**2.2 读取图像路径**

* input：

train\_images\_path="data/train\_image"

label\_images\_path="data/train\_50k\_mask"

* output：

shape = (49000, 2)、(1000, 2)、(11878, 1)

train\_data[0] = ['data/train\_image/n02910353/n02910353\_3750.png', 'data/train\_50k\_mask/n02910353/n02910353\_3750.png']

eval\_data[0] = ['data/train\_image/n03785016/n03785016\_285.png','data/train\_50k\_mask/n03785016/n03785016\_285.png']

test\_data = ['data/val\_image/ILSVRC2012\_val\_00011639.JPEG']

def get\_path(image\_path):

    files=[]

    for dir\_name in os.listdir(image\_path):

        for image\_name in os.listdir(os.path.join(image\_path,dir\_name)):

            if image\_name.endswith('.png') and not image\_name.startswith('.'):

                files.append(os.path.join(image\_path,dir\_name,image\_name))

    return sorted(files)

def get\_test\_data(test\_images\_path):

    test\_data=[]

    for name in os.listdir(test\_images\_path):

        img\_path=os.path.join(test\_images\_path,name)

        test\_data.append(img\_path)

    test\_data=np.expand\_dims(np.array(test\_data),axis=1)

    return test\_data

#扩展一维，以便将images与labels合并

images=np.expand\_dims(np.array(get\_path(train\_images\_path)),axis=1)

labels=np.expand\_dims(np.array(get\_path(label\_images\_path)),axis=1)

data=np.array(np.concatenate((images,labels),axis=1))

#打乱数据，同时也不会影响images与labels的对应关系

np.random.shuffle(data)

#分割数据集

train\_data=data[:-eval\_num,:]

eval\_data=data[-eval\_num:,:]

test\_data=get\_test\_data(test\_images\_path)

print(train\_data.shape, train\_data[0])

print(eval\_data.shape, eval\_data[0])

print(test\_data.shape, test\_data[0])

**2.3 数据增强**

封装数据预处理函数，其中训练集与验证集、测试集的函数不同，训练集可以加入图像色彩的调整，但如果要加入水平翻转，缩放等方法，注意要同时对label也做同样处理验证集和测试集可以将图像色彩的调整去掉，因为加入数据预处理的原因是扩大特征空间，使得模型有更好的拟合能力，但验证集就没必要扩大数据的特征空间了。

train\_transform=T.Compose([

            T.Resize(image\_size),  #裁剪

            T.ColorJitter(0.1,0.1,0.1,0.1), #亮度，对比度，饱和度和色调

            T.Transpose(), #CHW

            T.Normalize(mean=0.,std=255.) #归一化

        ])

eval\_transform=T.Compose([

            T.Resize(image\_size),

            T.Transpose(),

            T.Normalize(mean=0.,std=255.)

        ])

**2.4 定义数据读取器**

* 继承Paddle.io.Dataset类
* 调用Paddle.io.DataLoader类

class ImageDataset(Dataset):

    def \_\_init\_\_(self,path,transform):

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

        self.path=path

        self.transform=transform

    def \_load\_image(self,path):

        '''

        该方法作用为通过路径获取图像

        '''

        img=cv2.imread(path)

        img=cv2.resize(img,image\_size)

        return img

    def \_\_getitem\_\_(self,index):

        '''

        这里之所以不对label使用transform，因为观察数据集发现label的图像矩阵主要为0或1

        但偶尔也有0-255的值，所以要对label分情况处理

        而对data都进行transform是因为data都是彩色图片，图像矩阵皆为0-255，所以可以统一处理

        '''

        path=self.path[index]

        if len(path)==2:

            data\_path,label\_path=path

            data,label=self.\_load\_image(data\_path),self.\_load\_image(label\_path)

            data,label=self.transform(data),label

            label = label.transpose((2, 0, 1))

            label = label[0, :, :]

            label = np.expand\_dims(label, axis=0)

            if True in (label>1):

                label=label/255.

            label = label.astype("int64")

            return data,label

        if len(path)==1:

            data=self.\_load\_image(path[0])

            data=self.transform(data)

            return data

    def \_\_len\_\_(self):

        return len(self.path)

#获取数据读取器

train\_dataset=ImageDataset(train\_data,train\_transform)

eval\_dataset=ImageDataset(eval\_data,eval\_transform)

test\_dataset=ImageDataset(test\_data,eval\_transform)

train\_dataloader=DataLoader(train\_dataset,batch\_size=batch\_size,shuffle=True,drop\_last=True)

eval\_dataloader=DataLoader(eval\_dataset,batch\_size=batch\_size,shuffle=True,drop\_last=True)

**2.5 观察读取的图像**

def show\_images(imgs):

    #imgs是一个列表，列表里是多个tensor对象

    #定义总的方框的大小

    plt.figure(figsize=(3\*len(imgs),3), dpi=80)

    for i in range(len(imgs)):

        #定义小方框

        plt.subplot(1, len(imgs), i + 1)

        #matplotlib库只能识别numpy类型的数据，tensor无法识别

        imgs[i]=imgs[i].numpy()

        #展示取出的数据

        plt.imshow(imgs[i][0],cmap="gray",aspect="auto")

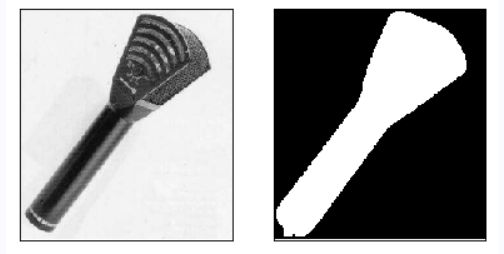
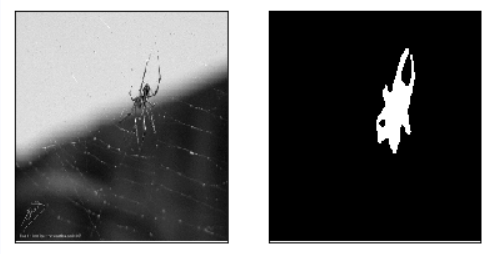
        #设置坐标轴

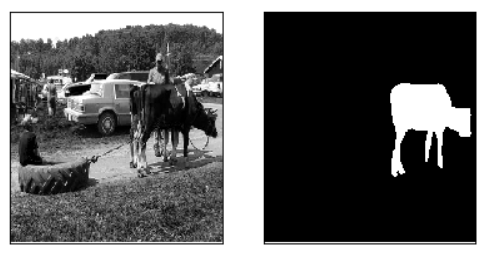
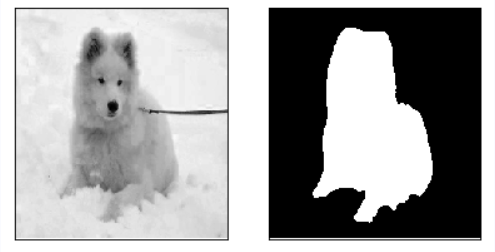
        plt.xticks([])

        plt.yticks([])

data,label=next(train\_dataloader())

show\_images([data[0],label[0]])

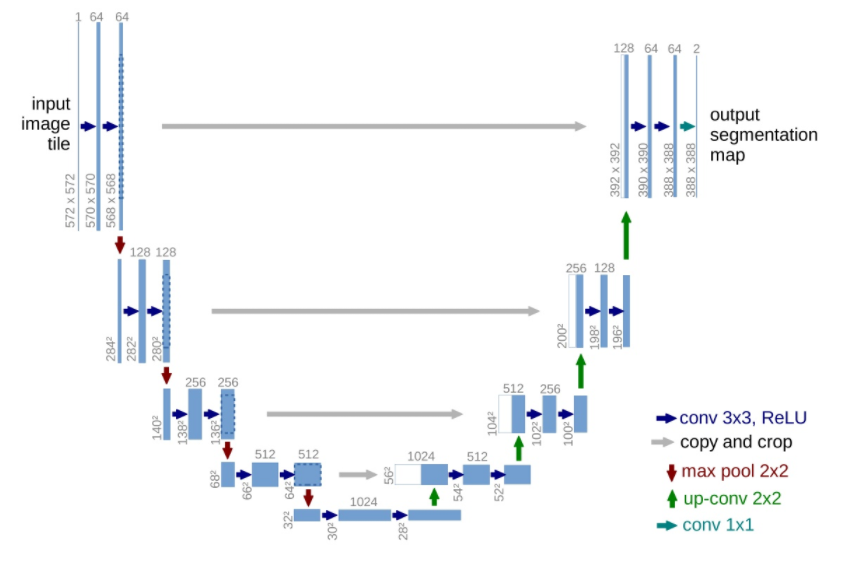




# 三、网络模型搭建

## 3.1 UNet

U-Net网络结构因为形似字母“U”而得名，最早是在医学影像的细胞分割任务中提出，结构简单适合处理小数量级的数据集。比较于FCN网络的像素相加，U-Net是对通道进行concat操作，保留上下文信息的同时，加强了它们之间的语义联系。整体是一个Encode-Decode的结构，如下图所示。



* **知识点1**:下采样Encode包括conv和max pool，上采样Decode包括up-conv和conv。
* **知识点2**:U-Net特点在于灰色箭头，利用通道融合使上下文信息紧密联系起来。

"""

paddlepaddle-gpu==2.2.1

time:2021.07.16 9:00

author:CP

backbone：U-net

"""

import paddle

from paddle import nn

class Encoder(nn.Layer):#下采样：两层卷积，两层归一化，最后池化。

    def \_\_init\_\_(self, num\_channels, num\_filters):

        super(Encoder,self).\_\_init\_\_()#继承父类的初始化

        self.conv1 = nn.Conv2D(in\_channels=num\_channels,

                              out\_channels=num\_filters,

                              kernel\_size=3,#3x3卷积核，步长为1，填充为1，不改变图片尺寸[H W]

                              stride=1,

                              padding=1)

        self.bn1   = nn.BatchNorm(num\_filters,act="relu")#归一化，并使用了激活函数

        self.conv2 = nn.Conv2D(in\_channels=num\_filters,

                              out\_channels=num\_filters,

                              kernel\_size=3,

                              stride=1,

                              padding=1)

        self.bn2   = nn.BatchNorm(num\_filters,act="relu")

        self.pool  = nn.MaxPool2D(kernel\_size=2,stride=2,padding="SAME")#池化层，图片尺寸减半[H/2 W/2]

    def forward(self,inputs):

        x = self.conv1(inputs)

        x = self.bn1(x)

        x = self.conv2(x)

        x = self.bn2(x)

        x\_conv = x           #两个输出，灰色 ->

        x\_pool = self.pool(x)#两个输出，红色 |

        return x\_conv, x\_pool

class Decoder(nn.Layer):#上采样：一层反卷积，两层卷积层，两层归一化

    def \_\_init\_\_(self, num\_channels, num\_filters):

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

        self.up = nn.Conv2DTranspose(in\_channels=num\_channels,

                                    out\_channels=num\_filters,

                                    kernel\_size=2,

                                    stride=2,

                                    padding=0)#图片尺寸变大一倍[2\*H 2\*W]

        self.conv1 = nn.Conv2D(in\_channels=num\_filters\*2,

                              out\_channels=num\_filters,

                              kernel\_size=3,

                              stride=1,

                              padding=1)

        self.bn1   = nn.BatchNorm(num\_filters,act="relu")

        self.conv2 = nn.Conv2D(in\_channels=num\_filters,

                              out\_channels=num\_filters,

                              kernel\_size=3,

                              stride=1,

                              padding=1)

        self.bn2   = nn.BatchNorm(num\_filters,act="relu")

    def forward(self,input\_conv,input\_pool):

        x = self.up(input\_pool)

        h\_diff = (input\_conv.shape[2]-x.shape[2])

        w\_diff = (input\_conv.shape[3]-x.shape[3])

        pad = nn.Pad2D(padding=[h\_diff//2, h\_diff-h\_diff//2, w\_diff//2, w\_diff-w\_diff//2])

        x = pad(x)                                #以下采样保存的feature map为基准，填充上采样的feature map尺寸

        x = paddle.concat(x=[input\_conv,x],axis=1)#考虑上下文信息，in\_channels扩大两倍

        x = self.conv1(x)

        x = self.bn1(x)

        x = self.conv2(x)

        x = self.bn2(x)

        return x

class UNet(nn.Layer):

    def \_\_init\_\_(self,num\_classes=59):

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

        self.down1 = Encoder(num\_channels=  3, num\_filters=64) #下采样

        self.down2 = Encoder(num\_channels= 64, num\_filters=128)

        self.down3 = Encoder(num\_channels=128, num\_filters=256)

        self.down4 = Encoder(num\_channels=256, num\_filters=512)

        self.mid\_conv1 = nn.Conv2D(512,1024,1)                 #中间层

        self.mid\_bn1   = nn.BatchNorm(1024,act="relu")

        self.mid\_conv2 = nn.Conv2D(1024,1024,1)

        self.mid\_bn2   = nn.BatchNorm(1024,act="relu")

        self.up4 = Decoder(1024,512)                           #上采样

        self.up3 = Decoder(512,256)

        self.up2 = Decoder(256,128)

        self.up1 = Decoder(128,64)

        self.last\_conv = nn.Conv2D(64,num\_classes,1)           #1x1卷积，softmax做分类

    def forward(self,inputs):

        x1, x = self.down1(inputs)

        x2, x = self.down2(x)

        x3, x = self.down3(x)

        x4, x = self.down4(x)

        x = self.mid\_conv1(x)

        x = self.mid\_bn1(x)

        x = self.mid\_conv2(x)

        x = self.mid\_bn2(x)

        x = self.up4(x4, x)

        x = self.up3(x3, x)

        x = self.up2(x2, x)

        x = self.up1(x1, x)

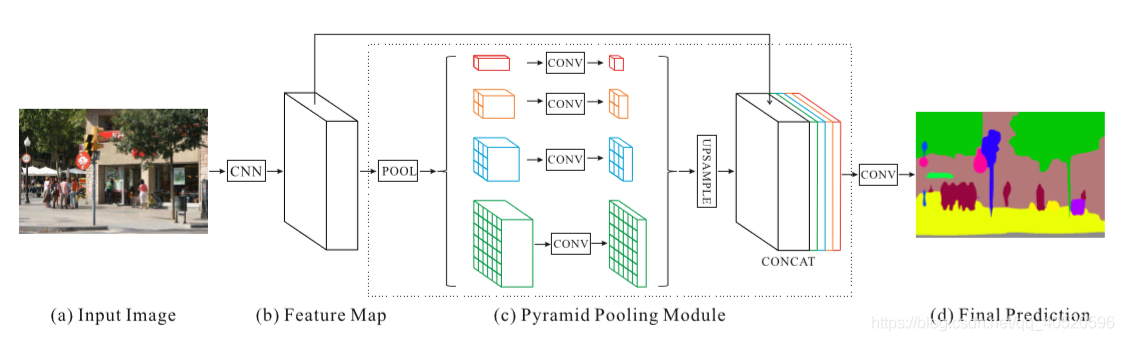
        x = self.last\_conv(x)

        return x

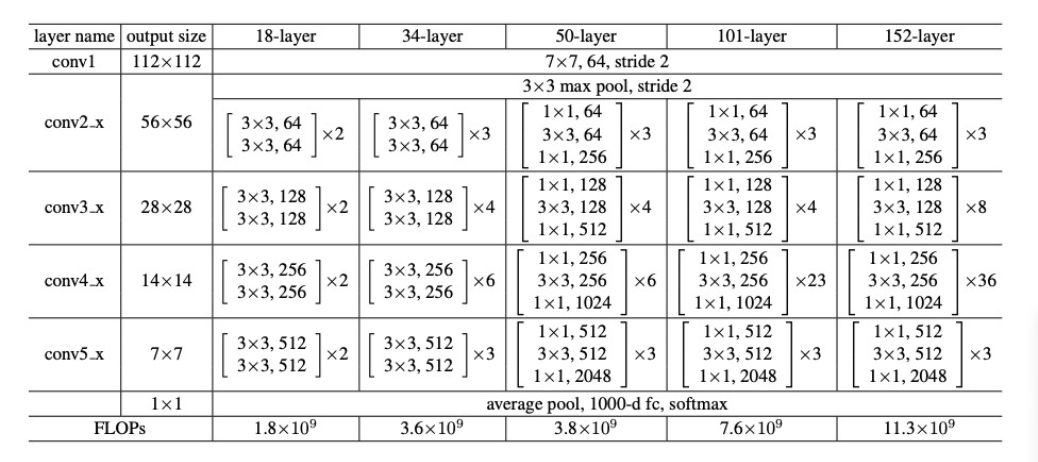
paddle.summary(UNet(), (1, 3, 600, 600))

**3.2 PSPNet**

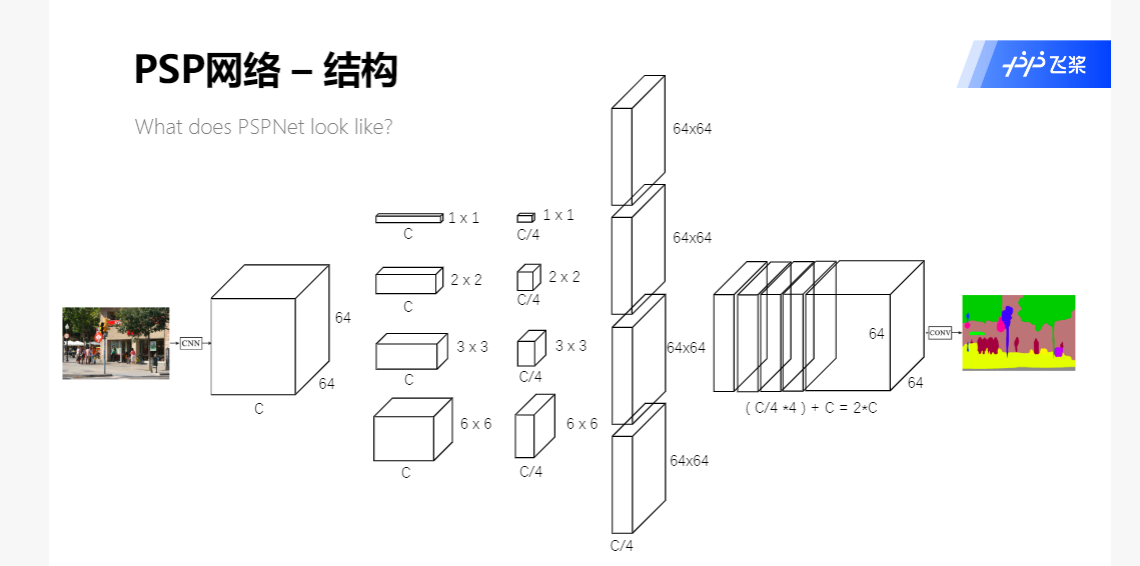
Pyramid Scene Parsing Network（PSPNet）网络结构形似金字塔而被命名，能够聚合不同尺度下的上下文信息，在场景解析上有很好的效果。PSPNet的精髓在于pyramid parsing module的构建，能够增大深层区域的感受野。



* **知识点1**:多尺度特征融合可以提高模型性能，深层网络中包含更多的语义信息和较小的位置信息。
* **知识点2**:input image需要通过CNN网路提取特征，这里使用的是飞桨预训练的resnet50网络。



* **知识点3**:PSPmodule将CNN的输出划成四个通道，然后进行上采样，全局特征和局部特征进行融合得到2C通道。



"""

paddlepaddle-gpu==2.2.1

time:2021.07.16 9:00

author:CP

backbone：PSPnet

"""

import paddle

import paddle.nn as nn

class PSPModule(nn.Layer):

    """

    num\_channels：输入通道数为C

    num\_filters ：输出通道数为C/4

    bin\_size\_list=[1,2,3,6]

    get1:

        nn.LayerList创建一个空列表的层

        .append拼接“层”的列表

    get2:

        paddle.nn.AdaptiveMaxPool2D输出固定尺寸的image\_size[H,W]

        paddle.nn.functional.interpolate卷积操作后，还原图片尺寸大小

        paddle.concat [H,W]同尺寸的图片，合并通道[C]

    """

    def \_\_init\_\_(self, num\_channels, bin\_size\_list):

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

        num\_filters = num\_channels // len(bin\_size\_list) #C/4

        self.features = nn.LayerList()         #一个层的空列表

        for i in range(len(bin\_size\_list)):

            self.features.append(

                paddle.nn.Sequential(

                    paddle.nn.AdaptiveMaxPool2D(output\_size=bin\_size\_list[i]),

                    paddle.nn.Conv2D(in\_channels=num\_channels, out\_channels=num\_filters, kernel\_size=1),

                    paddle.nn.BatchNorm2D(num\_features=num\_filters)

                )

            )

    def forward(self, inputs):

        out = [inputs] #list

        for idx, layerlist in enumerate(self.features):

            x = layerlist(inputs)

            x = paddle.nn.functional.interpolate(x=x, size=inputs.shape[2::], mode='bilinear', align\_corners=True)

            out.append(x)

        out = paddle.concat(x=out, axis=1) #NCHW

        return out

from paddle.vision.models import resnet50

class PSPnet(nn.Layer):

    def \_\_init\_\_(self, num\_classes=59, backbone='resnet50'):

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

        """

        https://github.com/PaddlePaddle/Paddle/blob/release/2.1/python/paddle/vision/models/resnet.py

        重复利用resnet50网络模型：

            1.初始化函数关键词——backbone

            2.神经网络模型实例化

            3.源代码查找层的变量名

        """

        # resnet50 3->2048

        res = resnet50()

        self.layer0 = nn.Sequential(res.conv1, res.bn1, res.relu, res.maxpool)

        self.layer1 = res.layer1

        self.layer2 = res.layer2

        self.layer3 = res.layer3

        self.layer4 = res.layer4#输出通道为2048

        #pspmodule 2048->4096

        num\_channels = 2048

        self.pspmodule = PSPModule(num\_channels, [1,2,3,6])

        num\_channels \*= 2

        #cls 4096->num\_classes

        self.classifier = nn.Sequential(

                             nn.Conv2D(num\_channels,512,3,1,1),

                             nn.BatchNorm(512,act='relu'),

                             nn.Dropout(),

                             nn.Conv2D(512,num\_classes,1))

        #aux:1024->256->num\_classes

        #单独分离出一层来计算函数损失

    def forward(self,inputs):

        x = self.layer0(inputs)

        x = self.layer1(x)

        x = self.layer2(x)

        x = self.layer3(x)

        x = self.layer4(x)

        x = self.pspmodule(x)

        x = self.classifier(x)

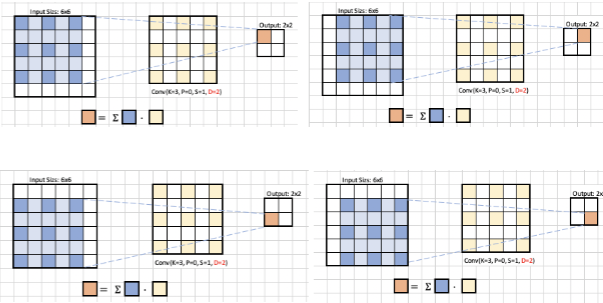
        x = paddle.nn.functional.interpolate(x=x, size=inputs.shape[2::], mode='bilinear', align\_corners=True)

        return x

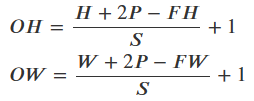
paddle.summary(PSPnet(), (1, 3, 600, 600))

**3.3 Deeplabv3**

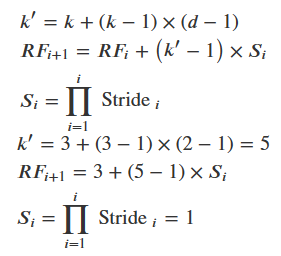
空洞卷积（Dilatee/Atrous Convolution）是一种特殊的卷积算子，针对卷积神经网络在下采样时图像分辨率降低、部分信息丢失而提出的卷积思路，通过在卷积核中添加空洞以获得更大的感受野。

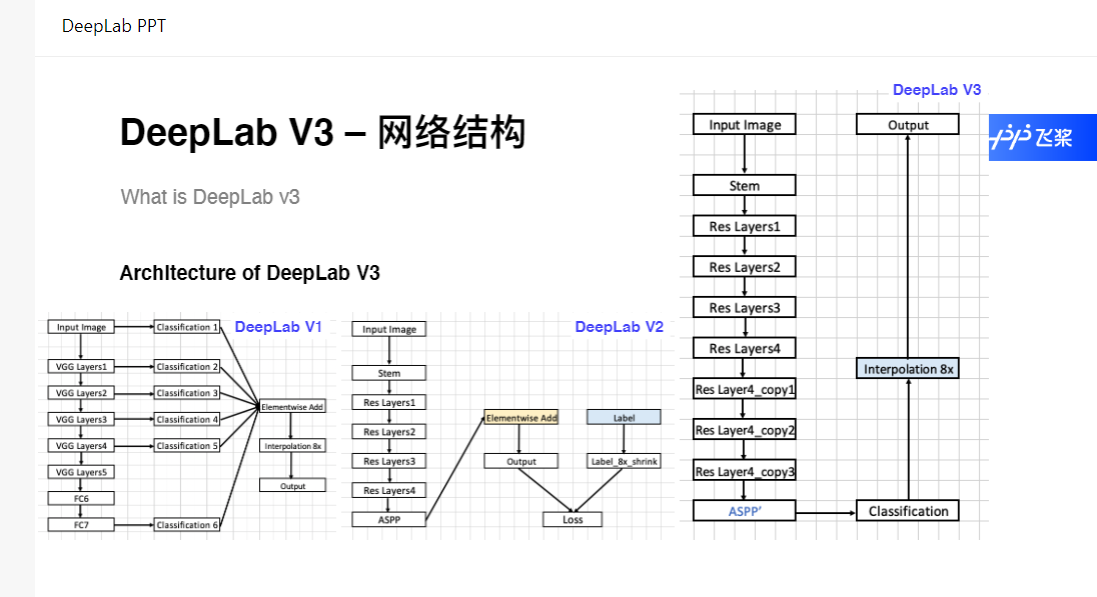


* 3x3卷积核，dilation rate 分别为1， 2， 4，空洞部分填充零。
* 输入大小为[H，W]，卷积核大小为{FH，FW]，填充为P，步幅为S，计算输出大小



* 3x3卷积核可以等效为5x5，假设卷积核大小k=3，空洞数d=2，则等效卷积核k‘，感受野RF，第一层感受野为3，Si为之前所有层步长的乘积。RFa=3，RFb=5，RFc=8。





"""

paddlepaddle-gpu==2.2.1

time:2021.07.20 9:00

author:CP

Deeplabv3

"""

import paddle

import paddle.nn as nn

class ASPPPooling(nn.Layer):

    def \_\_init\_\_(self,num\_channels,num\_filters):

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

        self.adaptive\_pool = nn.AdaptiveMaxPool2D(output\_size=3)

        self.features = nn.Sequential(

                                        nn.Conv2D(num\_channels, num\_filters,1),

                                        nn.BatchNorm(num\_filters, act="relu")

                                          )

    def forward(self, inputs):

        n1, c1, h1, w1 = inputs.shape

        x = self.adaptive\_pool(inputs)

        x = self.features(x)

        x = nn.functional.interpolate(x, (h1, w1), align\_corners=False)

        return x

class ASPPConv(nn.Layer):

    def \_\_init\_\_(self,num\_channels,num\_filters,dilations):

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

        self.asppconv = nn.Sequential(

                            nn.Conv2D(num\_channels,num\_filters,3,padding=dilations,dilation=dilations),

                            nn.BatchNorm(num\_filters, act="relu")

                            )

    def forward(self,inputs):

        x = self.asppconv(inputs)

        return x

#ASPP模块最大的特点是使用了空洞卷积来增大感受野

class ASPPModule(nn.Layer):

    def \_\_init\_\_(self, num\_channels, num\_filters, rates):

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

        self.features = nn.LayerList()

        #层一

        self.features.append(nn.Sequential(

                                        nn.Conv2D(num\_channels, num\_filters,1),

                                        nn.BatchNorm(num\_filters, act="relu")

                                          )

                            )

        #层二

        for r in rates:

            self.features.append(ASPPConv(num\_channels, num\_filters, r))

        #层三

        self.features.append(ASPPPooling(num\_channels, num\_filters))

        #层四

        self.project  = nn.Sequential(

                                    nn.Conv2D(num\_filters\*(2+len(rates)), num\_filters, 1),#TODO

                                    nn.BatchNorm(num\_filters, act="relu")

                                     )

    def forward(self, inputs):

        out = []

        for op in self.features:

            out.append(op(inputs))

        x = paddle.concat(x=out,axis=1)

        x = self.project(x)

        return x

class DeeplabHead(nn.Layer):

    def \_\_init\_\_(self, num\_channels, num\_classes):

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

        self.head = nn.Sequential(

                            ASPPModule(num\_channels, 256, [12, 24, 36]),

                            nn.Conv2D(256, 256, 3, padding=1),

                            nn.BatchNorm(256, act="relu"),

                            nn.Conv2D(256, num\_classes, 1)

                            )

    def forward(self, inputs):

        x = self.head(inputs)

        return x

from paddle.vision.models import resnet50

from paddle.vision.models.resnet import BottleneckBlock

class Deeplabv3(nn.Layer):

    def \_\_init\_\_(self, num\_classes=59, backbone='resnet50'):

        super(Deeplabv3, self).\_\_init\_\_()

        # resnet50 3->2048

        # resnet50 四层layers = [3 4 6 3]

        # 调用resnet.py模块，空洞卷积[2 4 8 16]

        res = resnet50()

        res.inplanes = 64        #初始化输入层

        self.layer0 = nn.Sequential(res.conv1, res.bn1, res.relu, res.maxpool)

        self.layer1 = res.\_make\_layer(BottleneckBlock, 64,3)

        self.layer2 = res.\_make\_layer(BottleneckBlock,128,4)

        self.layer3 = res.\_make\_layer(BottleneckBlock,256,6)

        self.layer4 = res.\_make\_layer(BottleneckBlock,512,3,stride=2,dilate= 2) #dilation=2

        self.layer5 = res.\_make\_layer(BottleneckBlock,512,3,stride=2,dilate= 4) #dilation=4

        self.layer6 = res.\_make\_layer(BottleneckBlock,512,3,stride=2,dilate= 8) #dilation=8

        self.layer7 = res.\_make\_layer(BottleneckBlock,512,3,stride=2,dilate=16) #dilation=16

        feature\_dim = 2048      #输出层通道2048

        self.deeplabhead = DeeplabHead(feature\_dim, num\_classes)

    def forward(self, inputs):

        x = self.layer0(inputs)

        x = self.layer1(x)

        x = self.layer2(x)

        x = self.layer3(x)

        x = self.layer4(x)

        x = self.layer5(x)

        x = self.layer6(x)

        x = self.layer7(x)

        x = self.deeplabhead(x)#ASPP模块进行分类

        # 恢复原图尺寸

        x = paddle.nn.functional.interpolate(x=x, size=inputs.shape[2::], mode='bilinear', align\_corners=True)

        return x

paddle.summary(Deeplabv3(), (1, 3, 600, 600))

# 四、高层API训练

CodeMarkdown

## 4.1 模型训练

#模型参数保存路径

work\_path = "/home/aistudio/net\_params/"

if os.path.exists(work\_path):

    shutil.rmtree(work\_path)

os.mkdir(work\_path)

#实例化，网络三选一，默认U-Net

model = paddle.Model(UNet(2))       #U-Net

# model = paddle.Model(PSPnet(2))    #PSPNet

#model = paddle.Model(Deeplabv3(2)) #Deeplabv3

"""

beta1 = paddle.to\_tensor([0.9], dtype="float32")

beta2 = paddle.to\_tensor([0.99], dtype="float32")

opt=paddle.optimizer.Adam(learning\_rate=1e-3

                            , beta1=0.9

                            , beta2=0.999

                            , epsilon=1e-08

                            , parameters=model.parameters()

                            , weight\_decay=1e-2

                            , grad\_clip=None

                            , name=None

                            )

"""

#定义优化器和损失函数

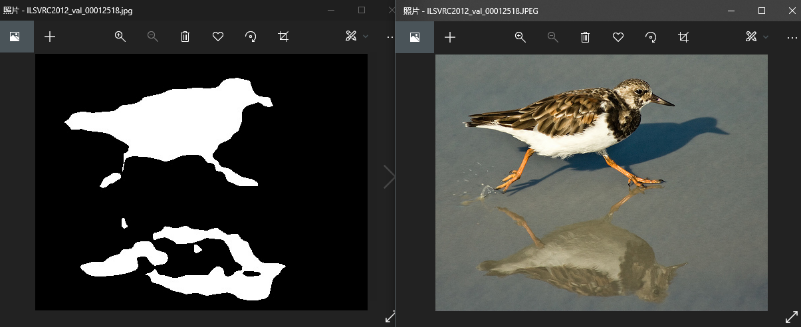
opt=paddle.optimizer.Momentum(learning\_rate=1e-3,parameters=model.parameters(),weight\_decay=1e-2)

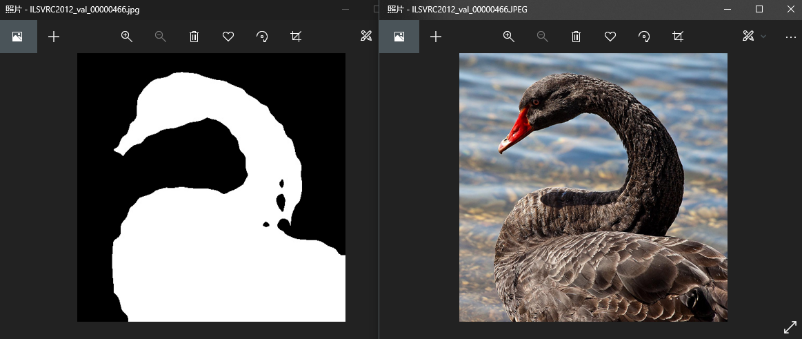
model.prepare(opt, paddle.nn.CrossEntropyLoss(axis=1))

#启动模型训练

model.fit(train\_dataloader, eval\_dataloader, epochs=10,verbose=2,save\_dir="./net\_params",log\_freq=200)

**4.2 模型预测**





#预测图像保存路径

work\_path = "/home/aistudio/data/val\_label"

if os.path.exists(work\_path):

    shutil.rmtree(work\_path)

os.mkdir(work\_path)

#读取模型参数文件路径

save\_dir=work\_path

checkpoint\_path="./net\_params/final"

#实例化，网络三选一，默认U-Net

model = paddle.Model(UNet(2))       #U-Net

#model = paddle.Model(PSPnet(2))    #PSPNet

#model = paddle.Model(Deeplabv3(2)) #Deeplabv3

model.load(checkpoint\_path)

for i,img in tqdm(enumerate(test\_dataset)):

    img=paddle.to\_tensor(img).unsqueeze(0)

    predict=np.array(model.predict\_batch(img)).squeeze(0).squeeze(0)

    predict=predict.argmax(axis=0)

    image\_path=test\_dataset.path[i]

    path\_lst=image\_path[0].split("/")

    save\_path=os.path.join(save\_dir,path\_lst[-1][:-5])+".jpg"

    cv2.imwrite(save\_path,predict\*255)