基于语义分割的钢铁表面缺陷检测

俸朗

/01

Phase1

Phase1

NG数据与OK数据**相互独立**

NG数据：7378个
OK数据：24812个
共：32190个

训练数据：90%，28971个
验证数据：10%，3219个

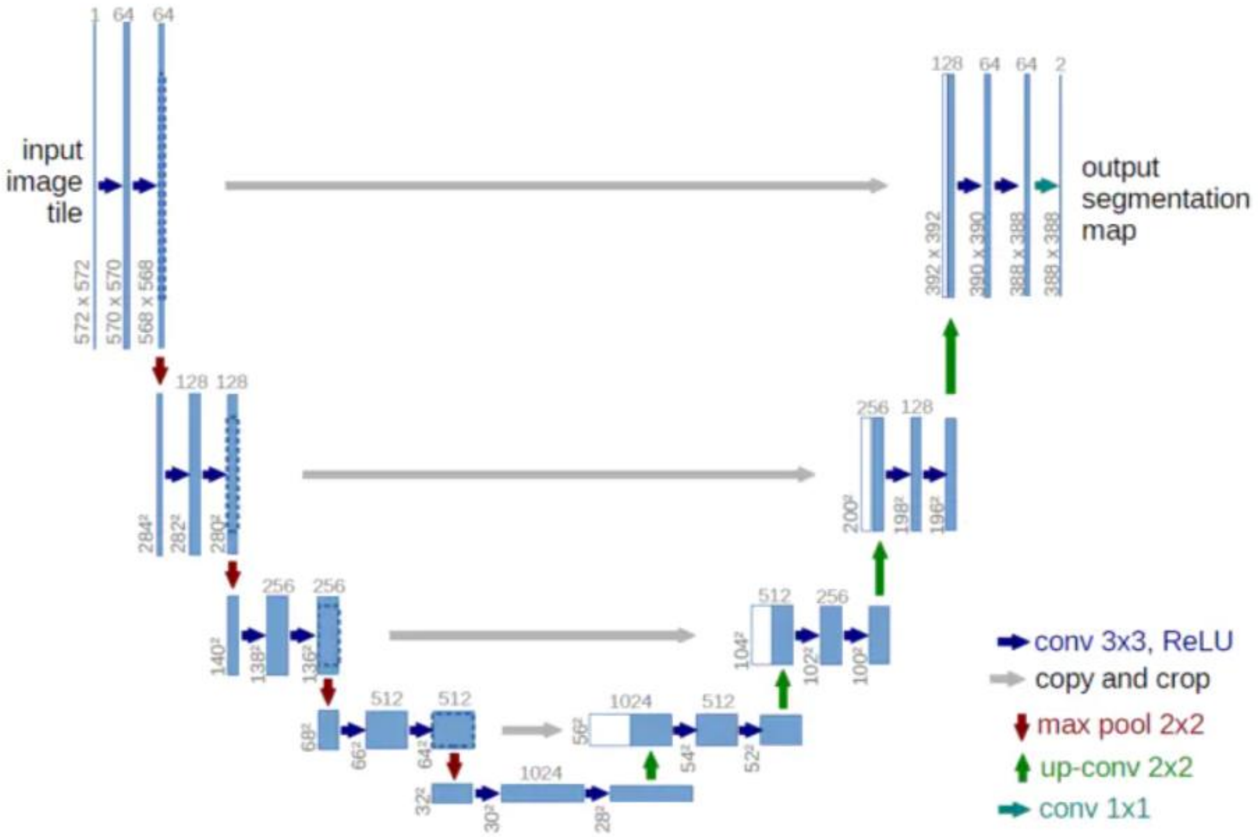
网络输入为单个NG图片或者单个OK图片

评价指标

把背景算成一类

共两类的MIoU作为精度指标

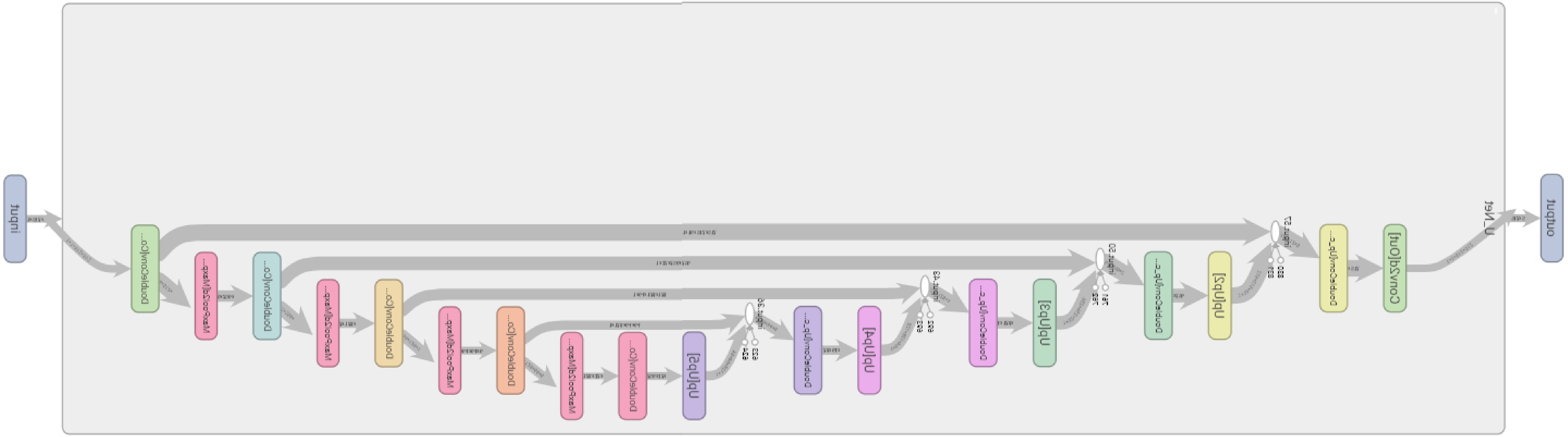
1.标准Unet



input image tile
output segmentation map
1 64 64
128 64 64 2
572 x 572
570 x 570
568 x 568
392 x 392
390 x 390
388 x 388
388 x 388
128 128
256 128
256 256
512 256
512 512
1024 512
1024
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Phase1

1.标准Unet



2.分类分割

Motivation

发现数据严重不均衡（ NG：7378，OK：24812)

这会导致Unet架构难以focus on学习缺陷特征。（Iou：0)

分类（所有数据）+分割（缺陷特征)

2.分类分割

分类框架

- 数据：所有数据
- 损失函数：CrossEntropy
- 数据增强：水平翻转，旋转，亮度，模糊
- 主干网络：VGG16

- 分类精度：92%

分割框架

- 数据：缺陷数据
- 损失函数：BCE+dice （0.8，0.2）权重
- 数据增强：水平翻转，旋转，亮度，模糊
- 网络：Unet架构，Backbone：VGG10

- 分割IoU：56%

2.分类分割

```
class VGGnet(nn.Module):
    @property
    def init_ch(self):
        return 32

    @property
    def channels(self):
        return [self.init_ch, self.init_ch * 2, self.init_ch * 4, self.init_ch * 8]

    def __init__(self, in_channel=1, num_class=2):
        super(VGGnet, self).__init__()
        self.feature = nn.Sequential(
            Conv(in_channel, self.channels[0]),
            Conv(self.channels[0], self.channels[0]),
            nn.MaxPool2d(kernel_size=2, stride=2),
            Conv(self.channels[0], self.channels[1]),
            Conv(self.channels[1], self.channels[1]),
            nn.MaxPool2d(kernel_size=2, stride=2),
            Conv(self.channels[1], self.channels[2]),
            Conv(self.channels[2], self.channels[2]),
            Conv(self.channels[2], self.channels[2]),
            nn.MaxPool2d(kernel_size=2, stride=2),
            Conv(self.channels[2], self.channels[3]),
            Conv(self.channels[3], self.channels[3]),
            Conv(self.channels[3], self.channels[3]),
            nn.MaxPool2d(kernel_size=2, stride=2),
            Conv(self.channels[3], self.channels[3]),
            Conv(self.channels[3], self.channels[3]),
            Conv(self.channels[3], self.channels[3]),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Linear(self.channels[3]* 16 * 16, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),

            nn.Linear(1024, 512),
```

决策：

```
output_vgg = model_vgg((data - 0.823) / 0.253)
output = model((data - 0.518) / 0.361)

vgg_result = output_vgg.argmax(dim=1, keepdim=True)
#
output[1 - vgg_result] = -50

total_loss += (data.shape[0] * calculate_loss(output, mask).item())

correct_num, num = calcuate_acc(output, mask)
```

Phase2

NG数据与OK数据**不独立**

↓

一个NG图片对应一个OK图片

↓

网络同时输入一个NG图片和对应一个OK图片

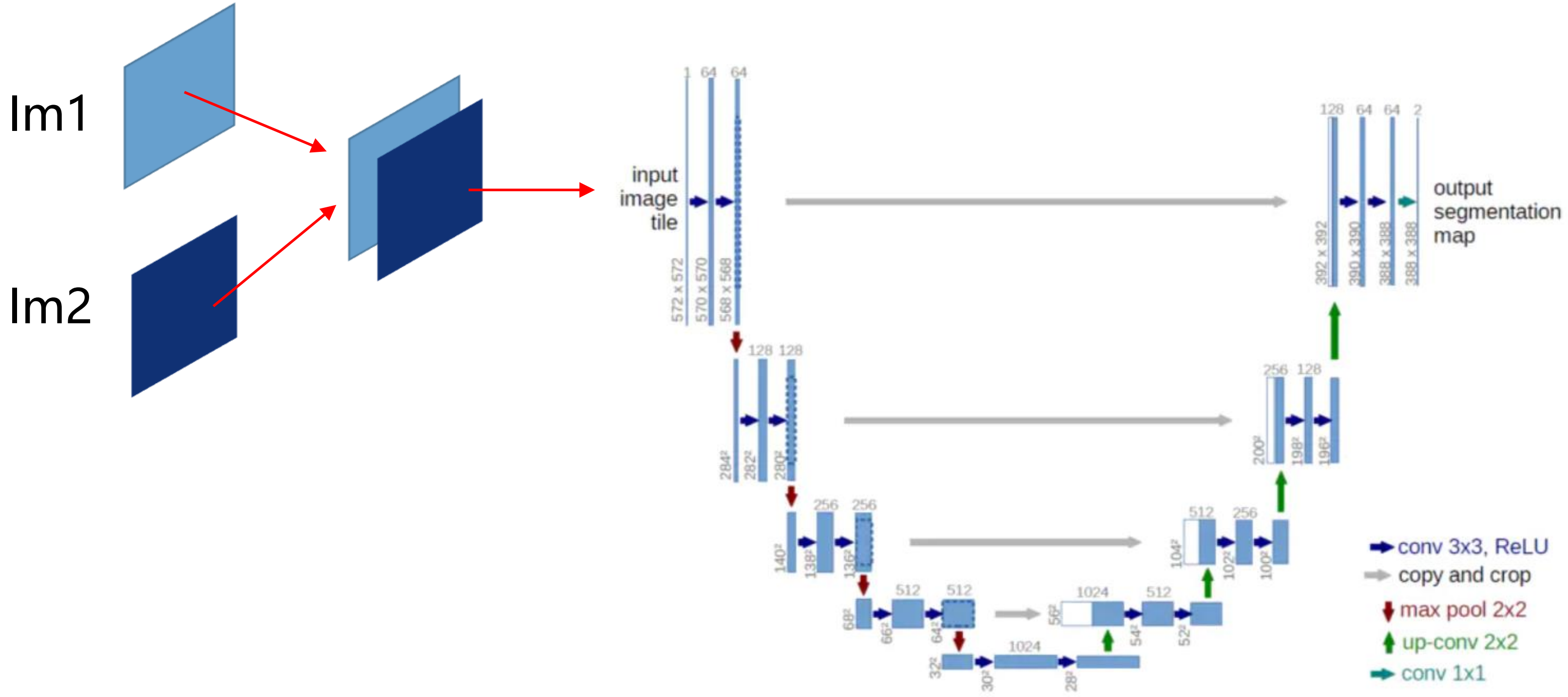
↓

评价指标：只看缺陷部分

3.前背景cat

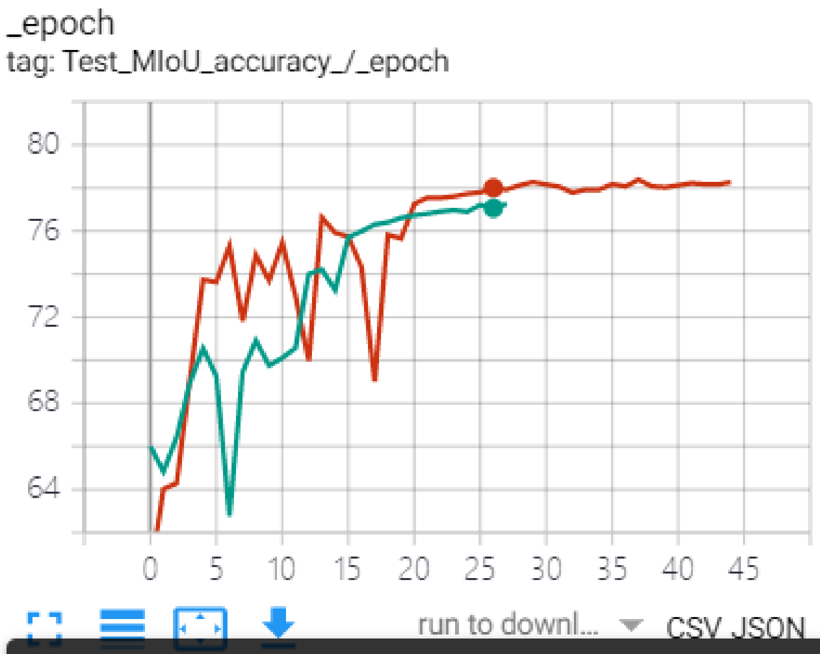
Motivation

网络输入：Im1和Im2（如何利用好背景Im1？）



考虑在通道维度做torch.cat

3.前背景cat



Name	Smoothed	Value	Step	Time	Relative
unet_deoublec_9e4_05	77.06	77.06	26	Wed Dec 22, 10:25:00	9h 18m 21s
unet_onlytraindata	78	78	26	Tue Dec 21, 07:07:52	6h 25m 22s

4.迁移学习

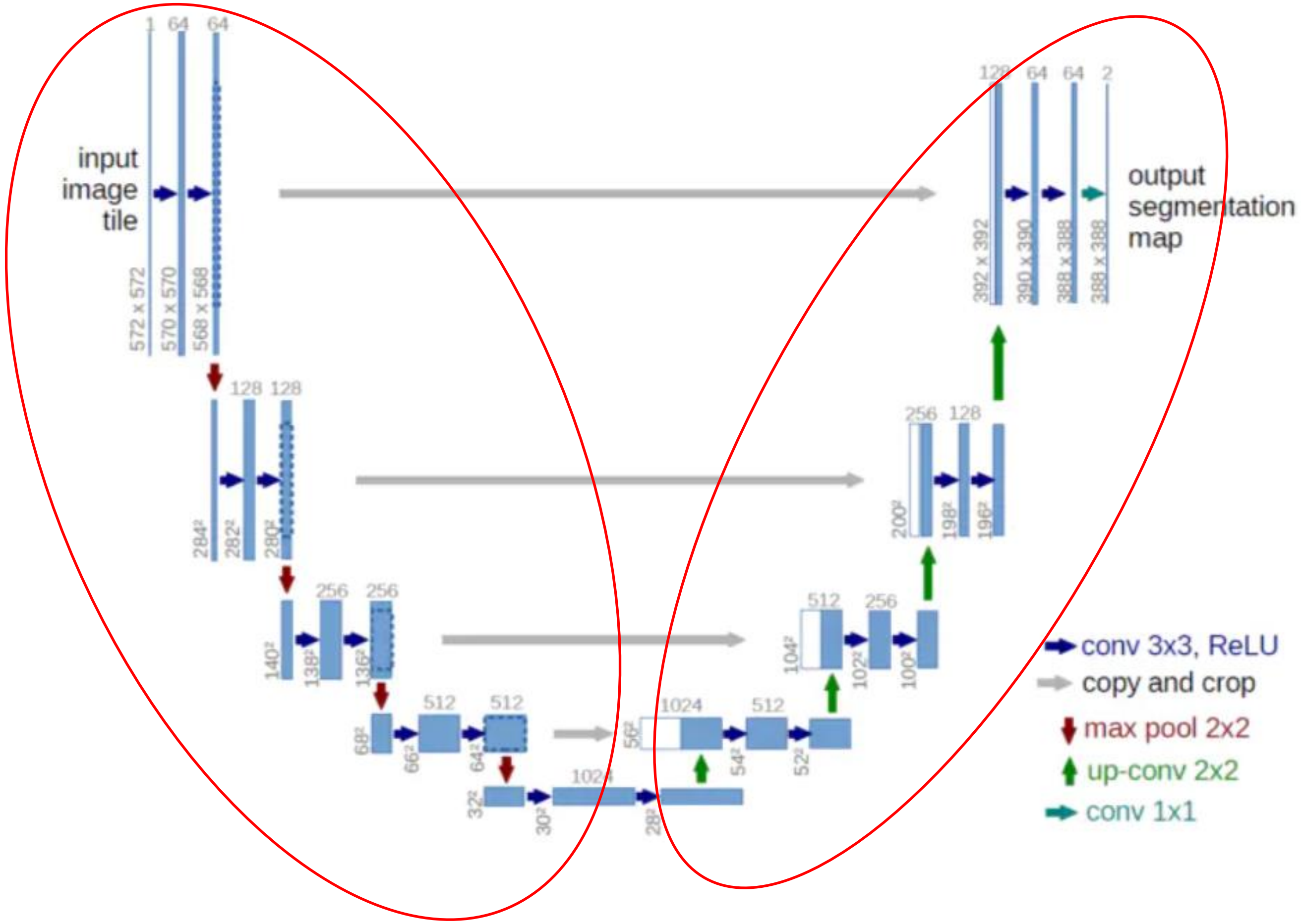
Motivation

- 数据集不够大
- 快速收敛深度模型

Fine-tuning

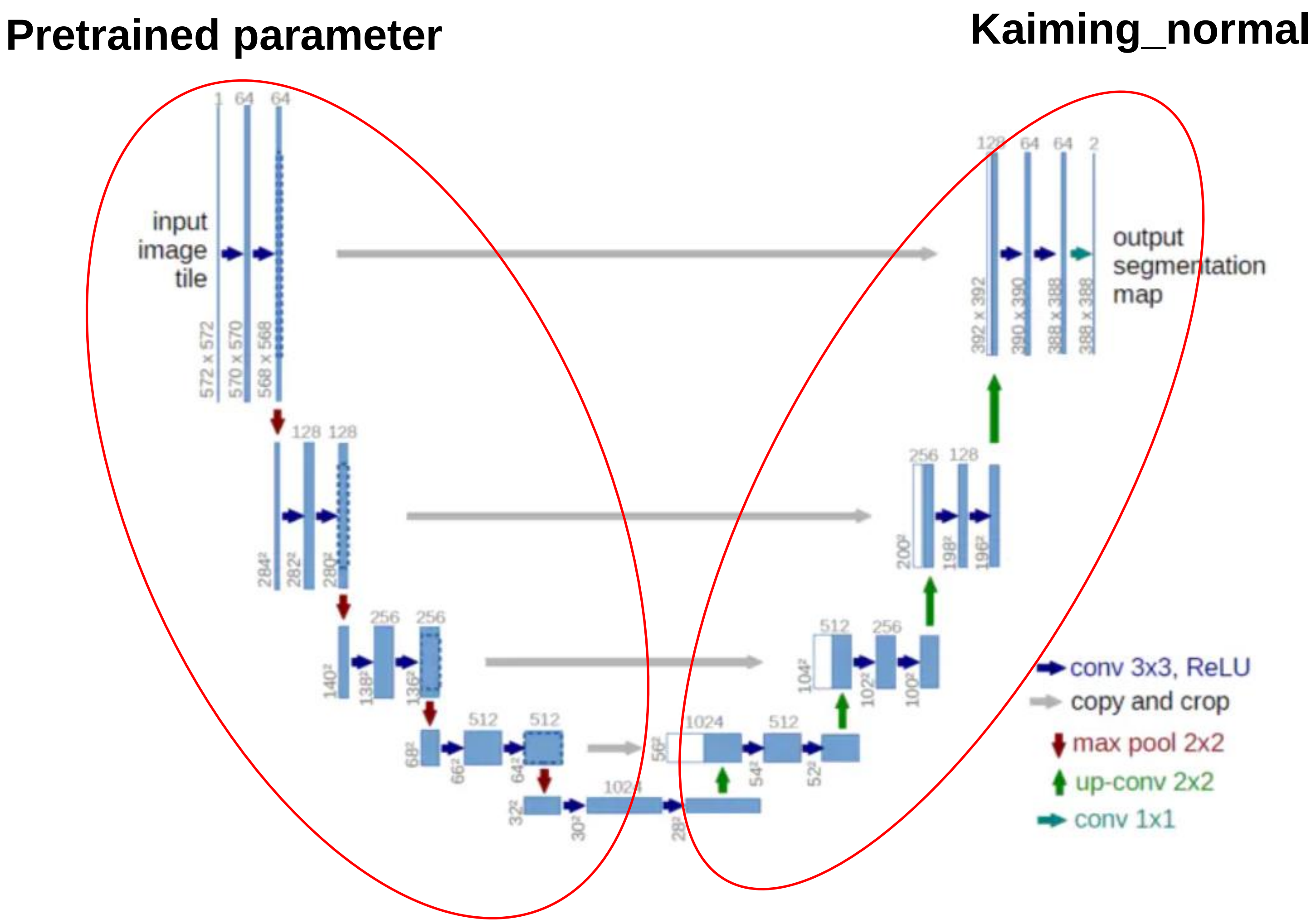
Pretrained parameter

Kaiming_normal



4.迁移学习

- 前5个epoch冻结Pretrained parameter，只训练后半部分参数
- 5个epoch之后，开始对Pretrained parameter进行微调



4.迁移学习

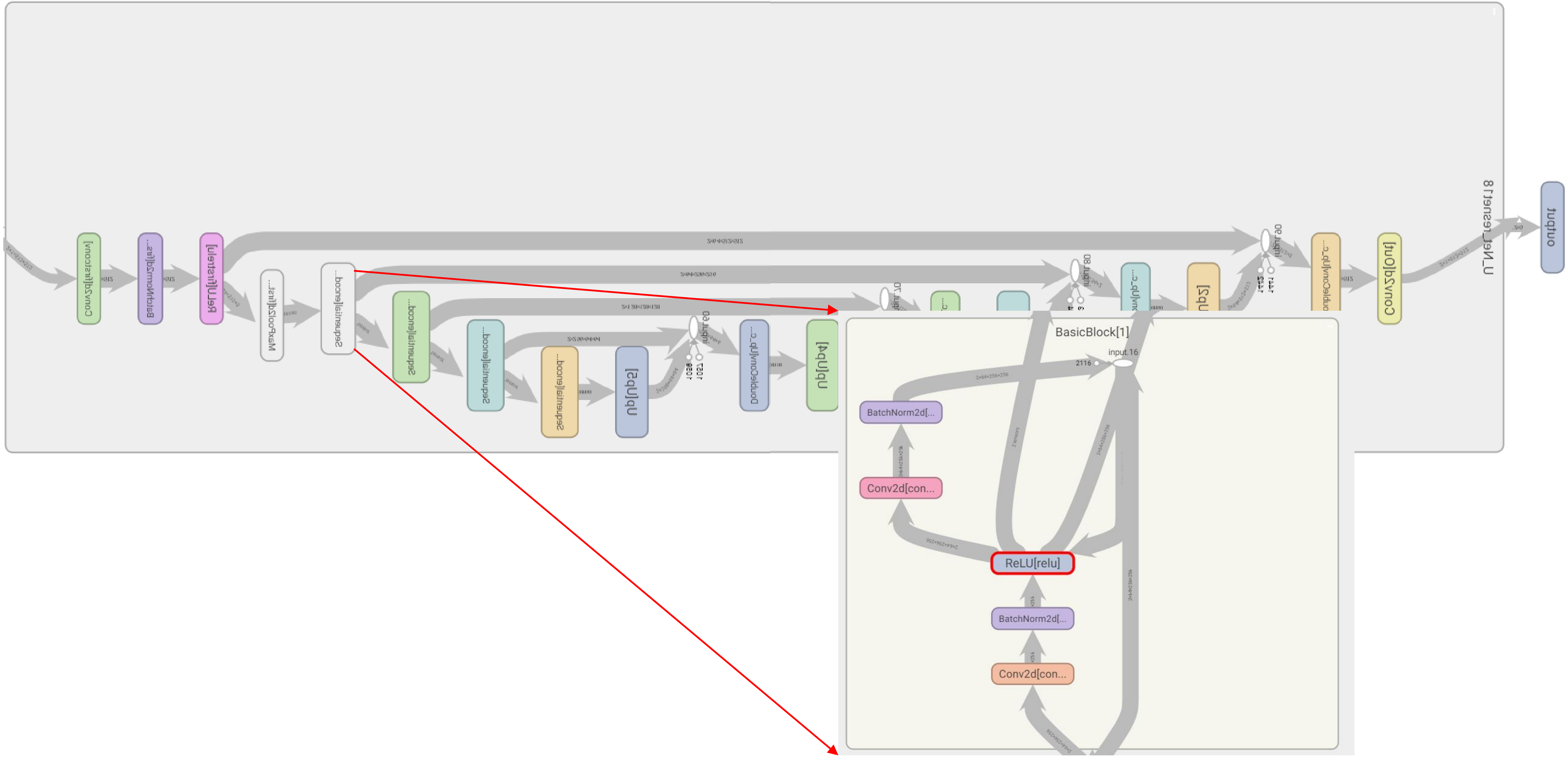
Backbone: ResNet18

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Firstconv重定义

四次下采样

4.迁移学习



BasicBlock[1]
input.16
2116
BatchNorm2d[...
Conv2d[con...
ReLU[relu]
BatchNorm2d[...
Conv2d[con...

4.迁移学习

Backbone: VGG16

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					

Firstconv重定义

四次下采样

4.迁移学习

细节：

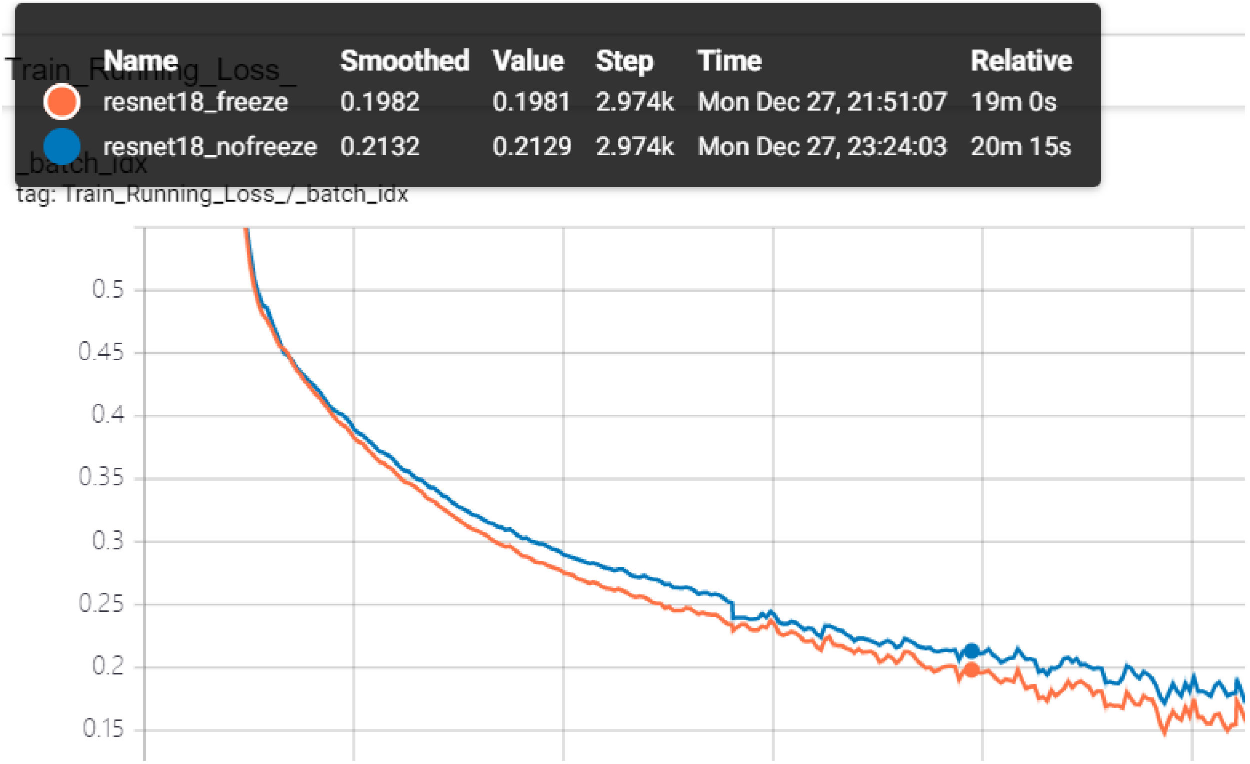
- **pytorch预训练参数**
- **第一层conv自定义**
- **前5个epoch冻结backbone参数**
- **5个后微调**

```python
def is_freezing_parameter(self, is_freezing):
    blocks = [
                self.encoder1,
                self.encoder2,
                self.encoder3,
                self.encoder4]

    for block in blocks:
        for p in block.parameters():
            p.requires_grad = not is_freezing
```

```python
ef train(args, model, optimizer, train_loader, device, wı
    model.train()
    if epoch < 5:
        model.is_freezing_parameter(True)
    else:
        model.is_freezing_parameter(False)
```

4.迁移学习

结果


Name Smoothed Value Step Time Relative
resnet18_freeze 0.1982 0.1981 2.974k Mon Dec 27, 21:51:07 19m 0s
resnet18_nofreeze 0.2132 0.2129 2.974k Mon Dec 27, 23:24:03 20m 15s
tag: Train_Running_Loss_/_batch_idx
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15

5.轻量Backbone

MobileNet

Depthwise separable convolution

深度可分离卷积

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

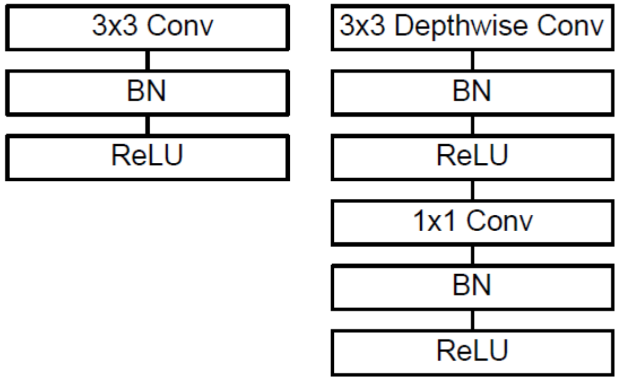
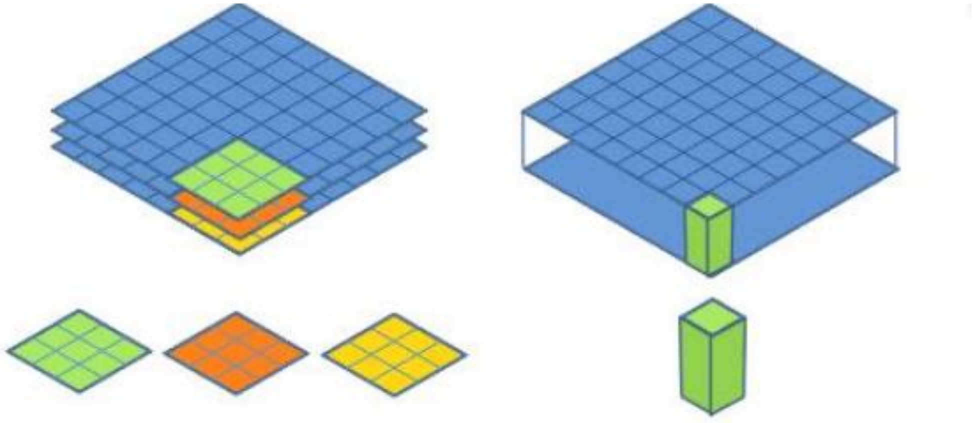


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.



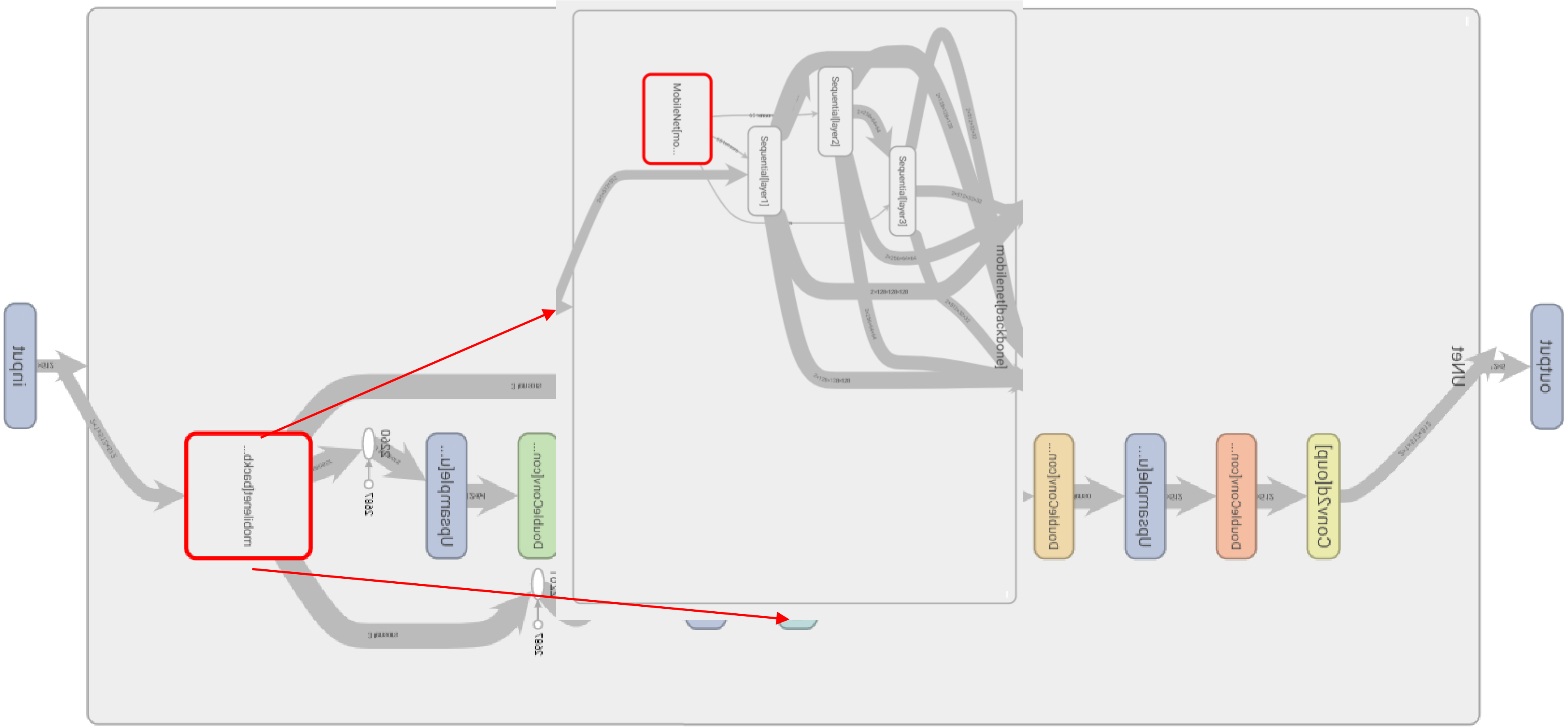
Depthwise Convolutional Filters Pointwise Convolutional Filters

5.轻量Backbone

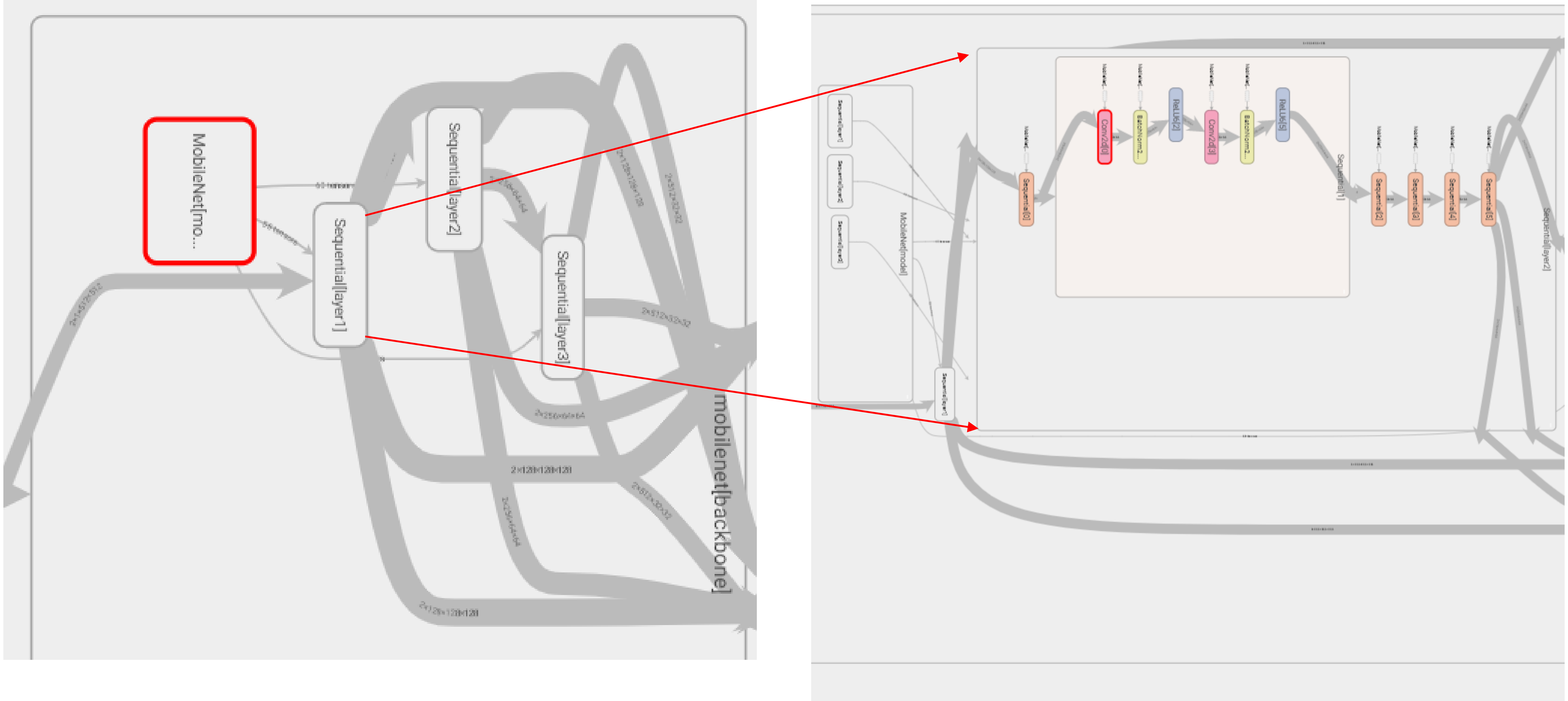
```python
class MobileNet(nn.Module):

    def __init__(self, n_channels, channels):
        super(MobileNet, self).__init__()
        self.layer1 = nn.Sequential(
            conv_bn(n_channels, channels[0], 1),

            conv_dw(channels[0], channels[1], 1),

            conv_dw(channels[1], channels[2], 2),
            conv_dw(channels[2], channels[2], 1),

            conv_dw(channels[2], channels[3], 2),
            conv_dw(channels[3], channels[3], 1),
        )
        self.layer2 = nn.Sequential(
            conv_dw(channels[3], channels[4], 2),
            conv_dw(channels[4], channels[4], 1),
            conv_dw(channels[4], channels[4], 1),
            conv_dw(channels[4], channels[4], 1),
            conv_dw(channels[4], channels[4], 1),
            conv_dw(channels[4], channels[4], 1),
        )
        self.layer3 = nn.Sequential(
            conv_dw(channels[4], channels[5], 2),
            conv_dw(channels[5], channels[5], 1),
        )
```

```python
class UNet(nn.Module):
    @property
    def init_ch(self):
        return 16

    @property
    def channels(self):
        return [self.init_ch, self.init_ch * 2, self.init_ch * 4, self.init_ch * 8, self.init_ch * 16, self.init_ch * 32]

    def __init__(self, in_channel, out_channel):
        super(UNet, self).__init__()
        self.n_channels = in_channel
        self.num_classes = out_channel

        self.backbone = mobilenet(in_channel, channels=self.channels)

        self.up1 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.conv1 = DoubleConv(self.channels[5], self.channels[4])

        self.up2 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.conv2 = DoubleConv(self.channels[5], self.channels[3])

        self.up3 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.conv3 = DoubleConv(self.channels[4], self.channels[2])

        self.up4 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        #nn.Upsample(scale_factor=2, mode='bilinear')
        self.conv4 = DoubleConv(self.channels[2], self.channels[1])

        self.oup = nn.Conv2d(self.channels[1], out_channel, kernel_size=1)
```

5.轻量Backbone



5.轻量Backbone



5.轻量Backbone

测试条件：
Backbone：VGG10
推理Batchsize：16
显存占用：约8G
测试数据量：752
推理时间：约42s

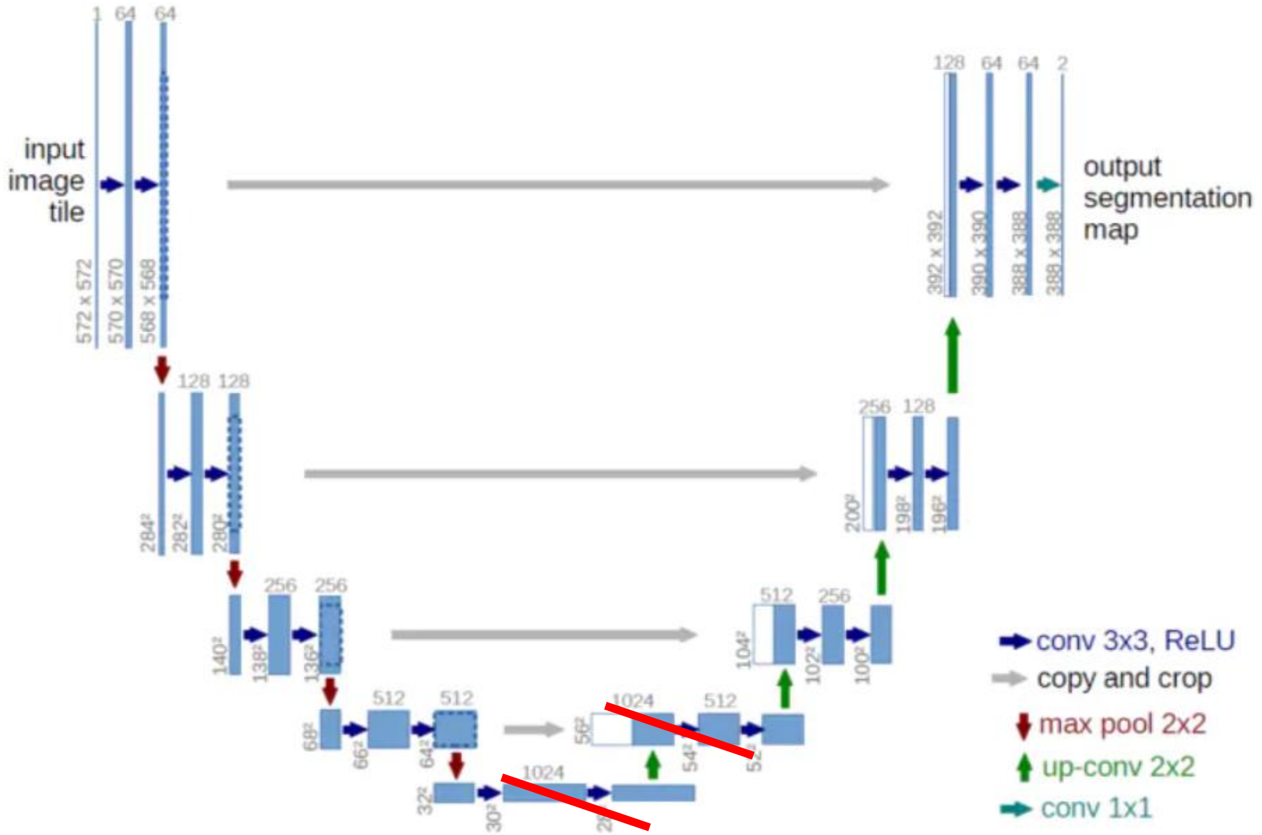
测试条件：
Backbone：MobileNet通道16
推理Batchsize：16
显存占用：约8G
测试数据量：752
推理时间：约56s

why?

6.减少下采样

Motivation

- 缺陷目标小
- 避免缺陷目标信息损失过多



6.减少下采样

Backbone: VGG16

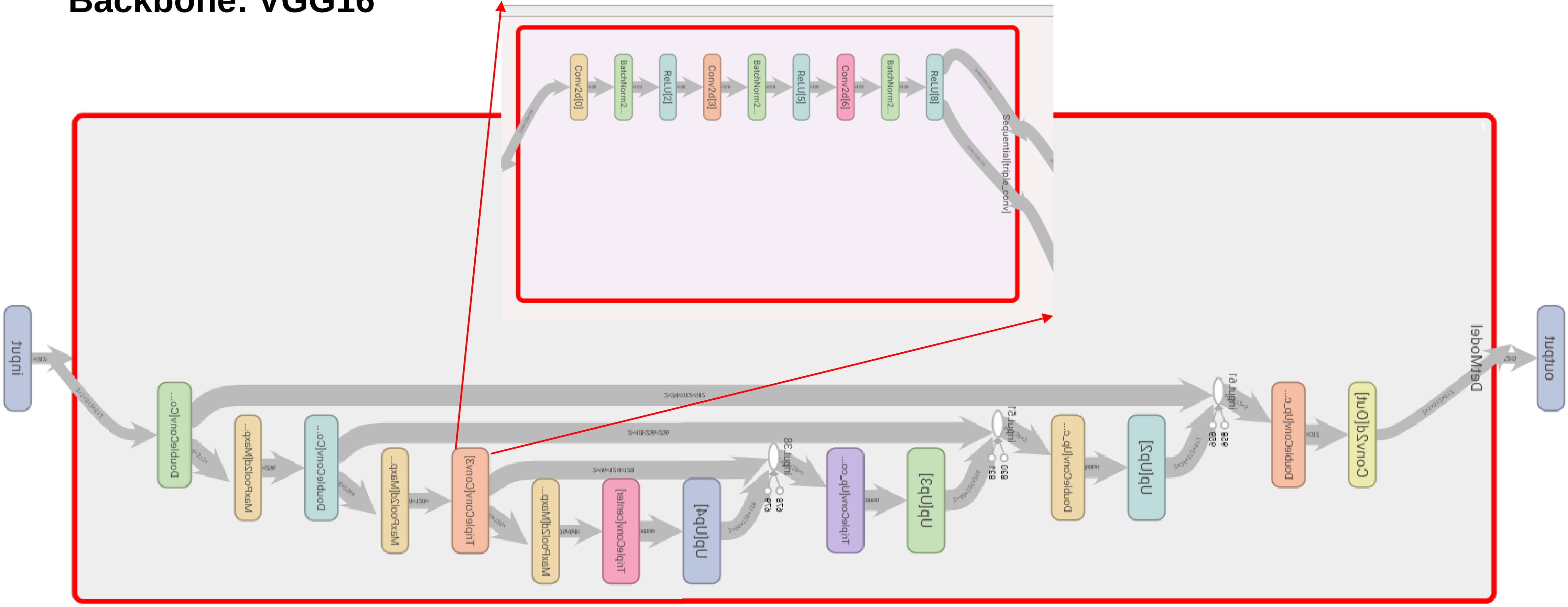
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					

Backbone: ResNet18

18-layer	34-layer	50-layer	101-layer	152-layer
7×7, 64, stride 2				
3×3 max pool, stride 2				
$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
average pool, 1000-d fc, softmax				
1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

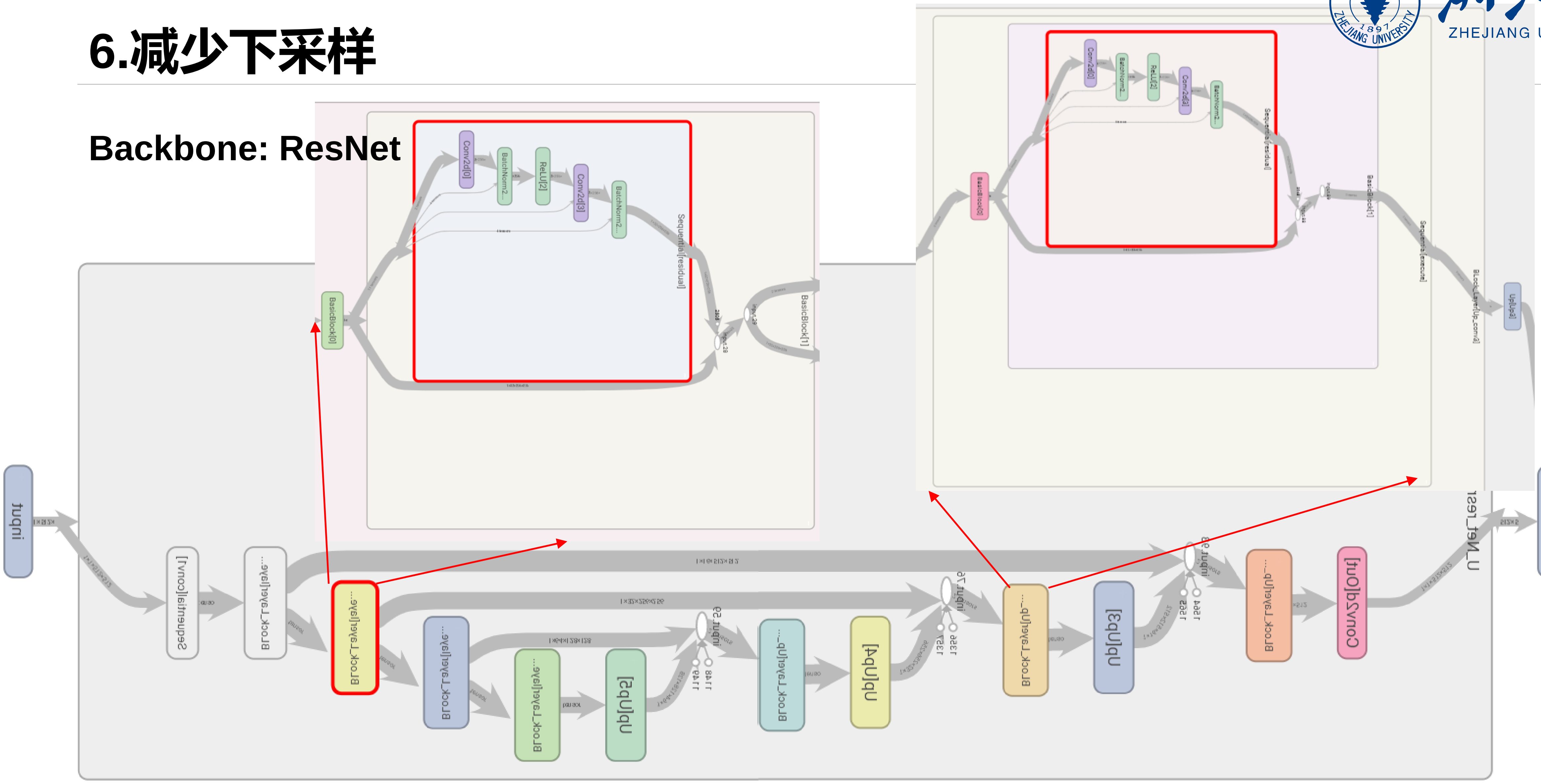
6.减少下采样

Backbone: VGG16



6.减少下采样

Backbone: ResNet



ZHEJIANG UNIVERSITY

最终模型

模型

- Unet
- Bac
- 通道
- 上采
- 输入

数据

- 数据
  小于
- 归一
  std=
- 水平翻转，旋转，亮度，模糊

```
d
is
t
t
t
t
f
f
```

[redacted]

```
]
+ 1) * k]
]
i + 1) * k]
test_data_path_list.append(data_path_list[i * k + index])
test_label_path_list.append(label_path_list[i * k + index])
], p=0.3),
```

结果

- **网络可训练参数量很小，方便部署**
- **综合显存占用率和速度和精度**

显存占用/M	训练集IOU	精度（测试集IOU）	精度排名	计算时间
5007	0.5648	0.40244		16.5821
3805	0.5542	0.35608		11.10519
3655	0.5255	0.317207		10.9593

最终模型

```
    exp_avg_sq.mul_(beta2).addcmul_(grad, grad, value=1 - beta2)
KeyboardInterrupt
(fl_pytorch) fenglang@s245:~/project/cv_homework/final$ cd summaries/
(fl_pytorch) fenglang@s245:~/project/cv_homework/final/summaries$ ls
final_down2                                      unet_olytrain_08_vgg13
final_resnet_d3_olytrainremove_08_bilinear       unet_olytrain_08_vgg13pre_sgd
final_vgg16ch24_d3_olytrainremove_08_bilinear    unet_olytrain_08_vgg13pre_sgd_0epo
final_vgg16ch24_d3_olytrainremove_08_bilinearIII unet_olytrain_08_vgg13pretrain
resnet18_freeze                                  unet_olytrain_08_vggpretrain
resnet18_nofreeze                                unet_olytrain_09d_bilinear_sgd
resnetmaxpool_d3_olytrainremove_08_bilinear      unet_olytrain_falossbce_allzq
resnetnew_d3_olytrain_08_bilinear                unet_olytrain_falossdice_allzq
resnetzz_d3_olytrainremove_08_bilinear           unet_onlytraindata
unet12_d5_olytrain_09_bilinear_sgd               unet_onlytraindata_05
unet_16ch                                        unetvgg16ch24_d3_olytrainremove_08_bilinear
unet_16ch_0_9                                    unetvgg16ch24_d3_olytrainremove_08_bilinear_zq3
unet_deoublec_9e4_05                             unetvgg16vh24_d3_olytrainremove_08_bilinear2
unet_olytrain_06_allzengqiang                    unetvgg_d3_olytrain_08_bilinear
unet_olytrain_08_resnet18_sgd                    vgg
unet_olytrain_08_resnet_sgd
(fl_pytorch) fenglang@s245:~/project/cv_homework/final/summaries$
```

End!