The Chinese University of Hong Kong,
Shenzhen

CIE6004 Image Processing and Computer Vision

Homework 3 Report

PortraitNet: Real-time Portrait Segmentation Network for Mobile Device

Name: Xiang Fei
Student ID: 120090414
Email: xiangfei@link.cuhk.edu.cn
Date: 2022.12

Table of Contents

1 Introduction

In homework 3, we are required to write a program to reproduce PortraitNet, which is an Real-time Portrait Segmentation Network for Mobile Device, based on the given paper.

Semantic segmentation is a classification at the pixel level, and pixels belonging to the same category must be classified into one category, so semantic segmentation understands images from the pixel level. For example, in the following photos, the pixels belonging to people should be divided into one category, the pixels belonging to motorcycles should also be divided into one category, and the background pixels should also be divided into one category. Note that semantic segmentation is different from instance segmentation. For example, if there are multiple people in a photo, for semantic segmentation, it is only necessary to classify the pixels of all the people into one category, but instance segmentation also divides the pixels of different people. categorized into different categories. That is to say, instance segmentation is a step further than semantic segmentation. The following is an example.

Figure 1: original graph

Figure 2: segmentation result

High-accuracy and efficient semantic image segmentation using convolutional neural networks has been a hot research topic in the field of computer vision. With the rapid development of mobile technology, automatic portrait segmentation has received increasing attention as a dedicated segmentation problem, since it benefits many mobile applications that require background editing (e.g., blurring, replacement, etc.) on portrait images. Portrait fine segmentation is a subtask of semantic segmentation. However, the general semantic segmentation network is not very good for fine segmentation of portraits. The author of the paper gives the following reasons:

- The face occupies at least 10% of the image area.
- Under strong lighting, the boundary will be blurred, which is not suitable for semantic segmentation, but this situation is common in selfies.
- The segmentation network is very large, which is not suitable for real-time and fast portrait segmentation on the mobile terminal.

Real-time portrait segmentation plays an important role in many applications on mobile devices, such as background replacement in video chats or conference calls. In the given paper, the authors propose a real-time portrait segmentation model called PortraitNet that can run effectively and efficiently on mobile devices. PortraitNet is based on a lightweight U-shaped architecture with two auxiliary losses in the training phase, and does not require additional cost for portrait inference in the testing phase. The two auxiliary losses are boundary loss and consistency constraint loss. The former improves the accuracy of boundary pixels, and the latter enhances the robustness in complex lighting environments. The authors evaluate PortraitNet on the portrait segmentation datasets EG1800 and Supervise-Portrait. Compared with the state-of-the-art methods, our method achieves remarkable performance in terms of both accuracy and efficiency, especially when generating results with sharper boundaries and harsh lighting conditions. Meanwhile, PortraitNet is able to process 224 $\times$ 224 RGB images at 30 FPS on iPhone 7.

2 Design and Method

In this part, I will introduce the details of PortraitNet, and also my code implementation. First, I need to stress that my implementation based on paddle. And in the following, I will focus on the network construction, which is the core part of this program.

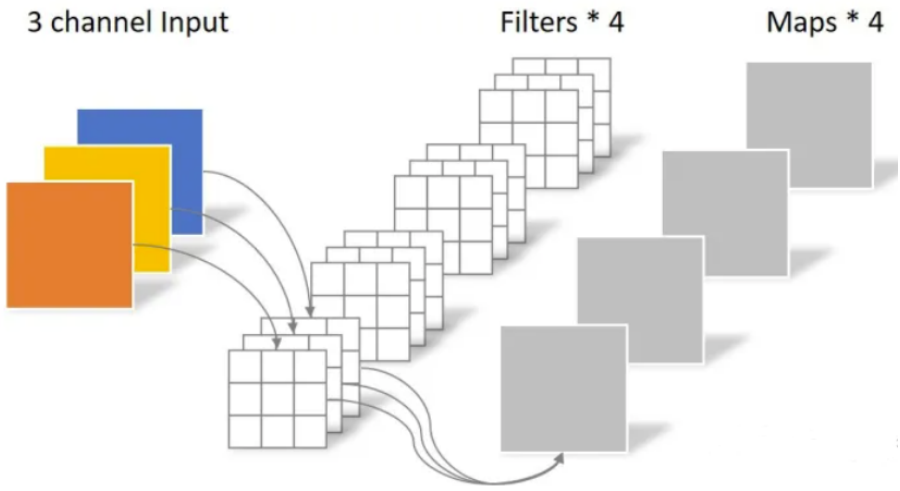
2.1 Depthwise convolution and Pointwise convolution

Depthwise (DW) convolution and Pointwise (PW) convolution are collectively called Depthwise Separable Convolution (see Google's Xception). This structure is similar to conventional convolution operations and can be used to extract features, but compared to conventional

convolution operation, its parameter quantity and operation cost are relatively low. So this structure will be encountered in some lightweight networks such as MobileNet.

Regular convolution operation:

For a 5×5 pixel, three-channel color input image (shape 5×5×3). After the convolution layer of 3×3 convolution kernel (assuming that the number of output channels is 4, the shape of the convolution kernel is 3×3×3×4), and finally output 4 Feature Maps. If there is same padding, the size is the same as that of the input layer Same (5×5), if not the size becomes 3×3.



At this time, the convolutional layer has a total of 4 Filters, each Filter contains 3 Kernels, and the size of each Kernel is 3×3. Therefore, the number of parameters of the convolutional layer can be calculated by the following formula:

$$N_{std} = 4 \times 3 \times 3 \times 3 = 108$$

Depthwise Separable Convolution:

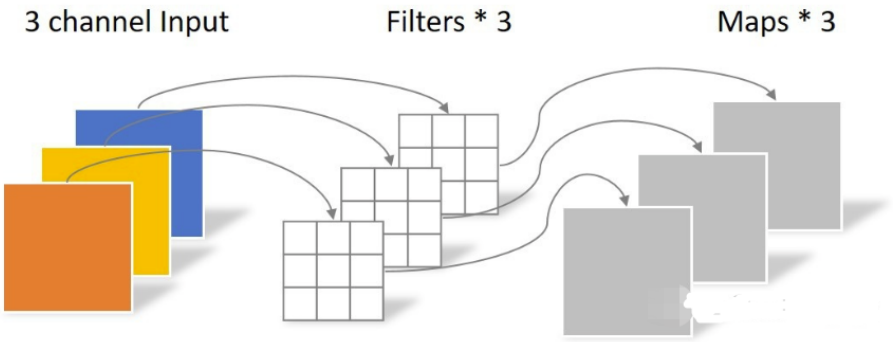
Depthwise Separable Convolution decomposes a complete convolution operation into two steps, namely Depthwise Convolution and Pointwise Convolution.

Depthwise Convolution:

Unlike conventional convolution operations, one convolution kernel of Depthwise Convolution is responsible for one channel, and one channel is convolved by only one convolution kernel. Each convolution kernel of the conventional convolution mentioned above operates each channel of the input image at the same time.

Also for a 5×5 pixel, three-channel color input image (shape is 5×5×3), Depthwise Convolution first undergoes the first convolution operation. Unlike the conventional convolution above, DW is completely in two dimensions. within the plane. The number of convolution kernels is the same as the number of channels in the previous layer (one-to-one correspondence between channels and convolution kernels). Therefore, a three-channel image is processed to

generate three Feature maps (if there is same padding, the size is the same as the input layer, which is 5×5), as shown in the figure below.



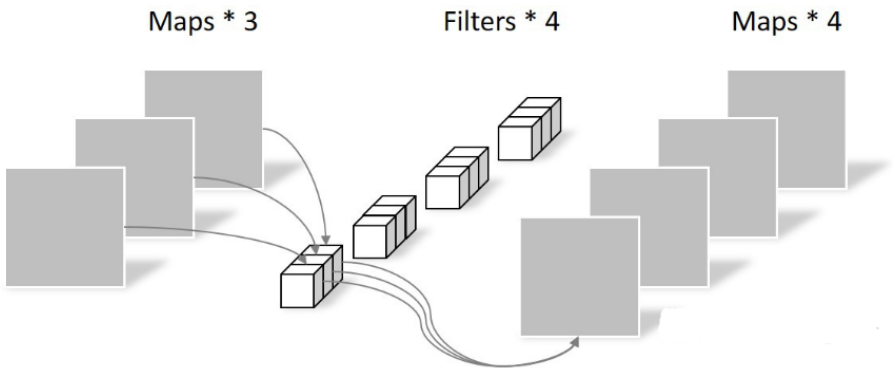
One of the Filters only contains a Kernel with a size of 3×3, and the number of parameters in the convolution part is calculated as follows:

$$N_{depthwise} = 3 \times 3 \times 3 = 27$$

The number of Feature maps after Depthwise Convolution is the same as the number of channels in the input layer, and the Feature map cannot be expanded. Moreover, this operation independently performs convolution operations on each channel of the input layer, and does not effectively use the feature information of different channels at the same spatial position. Therefore, Pointwise Convolution is needed to combine these Feature maps to generate a new Feature map.

Pointwise Convolution:

The operation of Pointwise Convolution is very similar to the conventional convolution operation. The size of its convolution kernel is 1×1×M, and M is the number of channels in the previous layer. Therefore, the convolution operation here will weight and combine the maps in the previous step in the depth direction to generate a new Feature map. There are several convolution kernels and several output Feature maps. As shown below.



Since the 1×1 convolution method is used, the number of parameters involved in the convo-

lution in this step can be calculated as:

$$N_{pointwise} = 1 \times 1 \times 3 \times 4 = 12$$

After Pointwise Convolution, 4 Feature maps are also output, which are the same as the output dimensions of conventional convolution.

Parameter comparison:

To recap, the number of parameters for a regular convolution is:

$$N_{std} = 4 \times 3 \times 3 \times 3 = 108$$

The parameters of Separable Convolution are obtained by adding two parts:

$$N_{depthwise} = 3 \times 3 \times 3 = 27$$

$$N_{pointwise} = 1 \times 1 \times 3 \times 4 = 12$$

$$N_{separable} = N_{depthwise} + N_{pointwise} = 39$$

With the same input, 4 Feature maps are also obtained, and the number of parameters of Separable Convolution is about 1/3 of that of conventional convolution. Therefore, under the premise of the same amount of parameters, the number of neural network layers using Separable Convolution can be made deeper.

2.2 MobileNet-v2

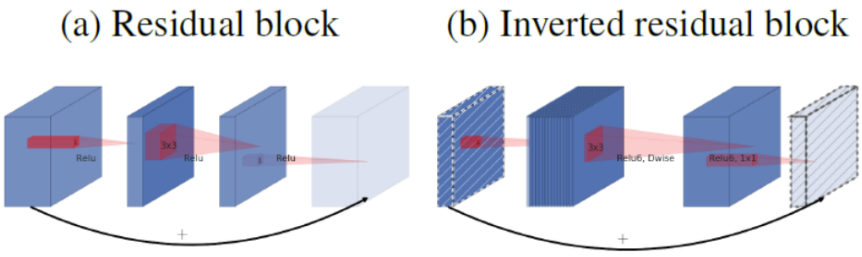
The main idea of MobileNet-v2 is to introduce a linear bottleneck (Linear Bottleneck) and an inverse residual (Inverted Residual) on the basis of v1 to improve the representation ability of the network. It is also a lightweight convolutional neural network. The main idea of MobileNet-v1 is depthwise separable convolution.

2.2.1 Problems with MobileNet-v1

- Structural issues: The structure of MobileNet-v1 is very simple, similar to VGGNet, which is a very retro straight structure. The cost performance of this structure is actually not high. A series of subsequent structures such as ResNet and DenseNet have proved that the cost performance of the network can be greatly improved by multiplexing image features and using operations such as Concat/Elwise+ for fusion.
- Problems with Depthwise convolution: Depthwise convolution does greatly reduce the amount of calculation, and the structure of Depthwise+Pointwise can also be close to ordinary convolution in terms of performance. However, in actual application, we found that the kernels in Depthwise are relatively easy to train and discard. After training, we found that many of the kernels trained by Depthwise were empty. Because each

kernel_dim of depthwise is much smaller than ordinary convolution, under the influence of too small kernel_dim plus the activation of ReLU, the output neuron can easily become 0, so it is useless to learn. The output gradient of ReLU for 0 is 0, so once it falls into 0 output, it cannot be recovered.

2.2.2 Inverted Residual

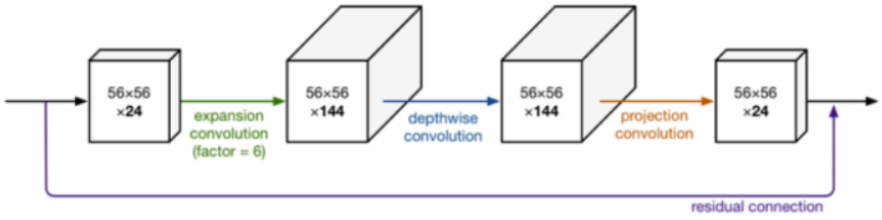


Original residual block: reduce – transfer – expand (narrow in the middle and wide at both ends)

The Residual block first uses the 1x1 convolution down channel to pass the ReLU, then 3x3 convolution to the ReLU, and finally uses the 1x1 convolution to pass the ReLU recovery channel, and adds it to the input. The reason for the 1*1 convolution drop channel is to reduce the amount of calculation, otherwise the calculation amount of the 3x3 convolution in the middle is too large. So the Residual block is narrow in the middle and wide at both ends.

Inverted residual block: expand – transfer – reduce (wide in the middle and narrow at both ends)

In the Inverted Residual block, the 3x3 convolution becomes Depthwise, and the amount of calculation is very small, so the number of channels can be a little more, and the effect is better, so first increase the number of channels through 1x1 convolution, then Depthwise3x3 convolution, and finally use 1x1 convolution product to reduce the number of channels. The number of channels at both ends is very small, so the calculation amount of the 1x1 convolution ascending channel and descending channel is not large, and although the number of channels in the middle is large, the convolution calculation amount of Depthwise is not large. The following figure is a specific example:



Note: Why use Inverted residual?

Some thinking, may not be right. The bottleneck structure of skip connection has been proved to be very effective, so I want to use it. But if you compress the channels first as before, the number of channels will be small, and then the number of channels will disappear, so it is better to increase and then decrease.

2.2.3 Linear Bottleneck

Original bottlenecks: Elwise + with ReLU at end of a bottleneck

Linear bottlenecks: Elwise + with NO ReLU at the end of a bottleneck

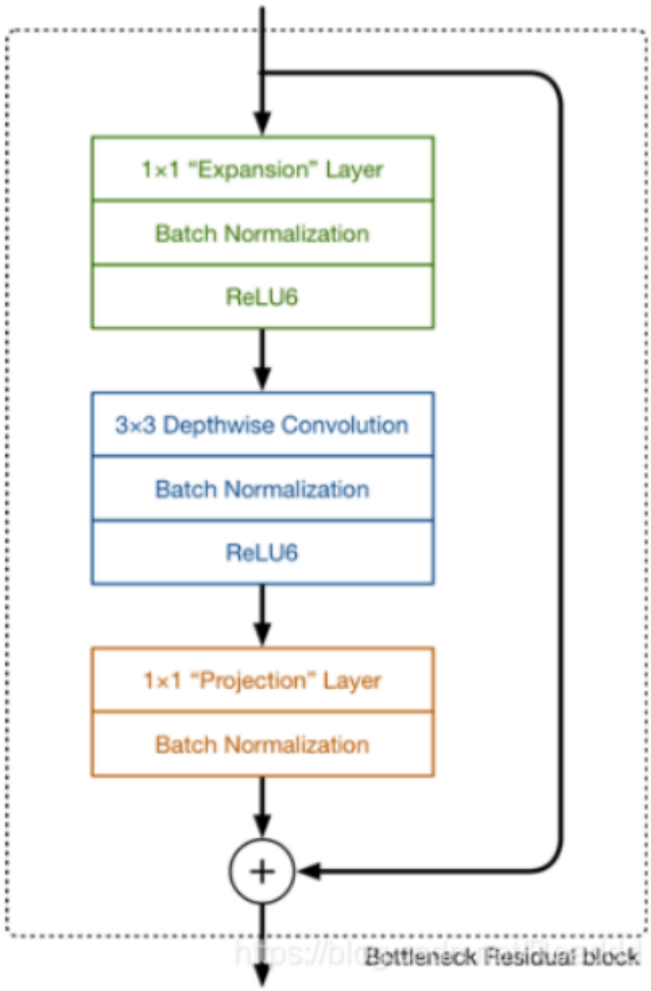
The description of this part in the paper is relatively obscure, but the general meaning is: When we design the network structure, if we want to reduce the amount of computation, we need to design the network dimension as low as possible, but if the dimension is low, ReLU Activation functions may filter out a lot of useful information. And ReLU acts as a linear classifier for the part that is not filtered out, that is, the non-zero part. Since using ReLU for activation transformation in low-dimensional space will lose a lot of information, the paper uses linear bottleneck at the end of Bottleneck (that is, does not use ReLU activation, but does linear transformation) to replace the original nonlinear activation transformation.

Note: Why Use Linear Bottleneck

Some thinking: ReLU makes the negative semi-axis 0. There are not many parameters in the first place, and the learning ability is limited. Now let some parameters be 0, and there is nothing to learn. Simply don't use ReLU after elwise.

2.2.4 Inverted Residual + Linear Bottleneck

The structure is shown in the figure below. In the network design of MobileNet-v2, in addition to continuing to use the Depthwise Conv (convolutional layer in the middle of Bottleneck) structure, Expansion layer and Projection layer are also used. The Expansion layer uses 1x1 convolution to map the low-dimensional space to the high-dimensional space (expand the number of channels). Here Expansion has a hyperparameter that expands the dimension several times, which can be adjusted according to the actual situation. The default value is 6, that is, Expand 6 times. The Projection layer also uses 1x1 convolution. Its purpose is to map high-dimensional features to low-dimensional spaces (reduce the number of channels). It should be noted that the residual connection is connected in the input and output parts. In addition, as mentioned earlier, because the conversion from high-dimensional to low-dimensional, using the ReLU activation function may cause information loss or destruction, so in the Projection convolution part, we no longer use the ReLU activation function but use the linear activation function.



The following is my code implementation of Inverted Residual block.

```
class InvertedResidual(nn.Layer):
    def __init__(self, input_c, output_c, stride, bn_scale):
        super(InvertedResidual, self).__init__()
        self.stride = stride
        self.input_c = input_c
        self.output_c = output_c

        self.conv = nn.Sequential(
            # pointwise convolution
            nn.Conv2D(input_c,input_c*bn_scale,kernel_size=1,stride=1,padding=0,dilation=1,
            nn.BatchNorm(input_c*bn_scale,param_attr=ParamAttr(),bias_attr=ParamAttr()),
            nn.ReLU6(),
            # depthwise convolution
            nn.Conv2D(input_c*bn_scale,input_c*bn_scale,kernel_size=3,stride=stride,padding
            nn.BatchNorm(input_c*bn_scale,param_attr=ParamAttr(),bias_attr=ParamAttr()),
            nn.ReLU6(),
            # pointwise-linear
            nn.Conv2D(input_c*bn_scale,output_c,kernel_size=1,stride=1,padding=0,dilation=1
            nn.BatchNorm(output_c,param_attr=ParamAttr(),bias_attr=ParamAttr())
        )

    def forward(self, inputs):
        y = self.conv(inputs)
        if ((self.stride==1) and (self.input_c == self.output_c)):
            y = paddle.add(inputs, y)
        return y
```

2.2.5 Overall network structure of MobileNet-v2

The model of MobileNetV2 is shown in the figure below, where t is the multiple of the internal dimension of the bottleneck, c is the number of channels, n is the number of times the bottleneck is repeated, and s is the sride.

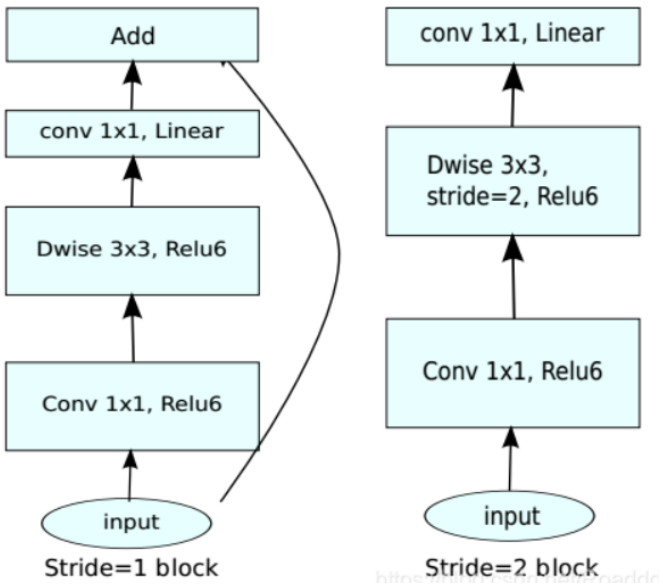
Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k		

My code of this structure:

```
self.e_stage0 = nn.Sequential(
    nn.Conv2D(3,32,kernel_size=3,stride=2,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False),
    nn.BatchNorm(32,param_attr=ParamAttr(),bias_attr=ParamAttr()),
    nn.ReLU()
)
# 1/2
self.e_stage1 = InvertedResidual(32,16,1,1)
# 1/4
self.e_stage2 = nn.Sequential(
    InvertedResidual(16,24,2,6),
    InvertedResidual(24,24,1,6)
)
# 1/8
self.e_stage3 = nn.Sequential(
    InvertedResidual(24,32,2,6),
    InvertedResidual(32,32,1,6),
    InvertedResidual(32,32,1,6)
)
```

```
# 1/16
self.e_stage4 = nn.Sequential(
    InvertedResidual(32,64,2,6),
    InvertedResidual(64,64,1,6),
    InvertedResidual(64,64,1,6),
    InvertedResidual(64,64,1,6)
)
# 1/16
self.e_stage5 = nn.Sequential(
    InvertedResidual(64,96,1,6),
    InvertedResidual(96,96,1,6),
    InvertedResidual(96,96,1,6)
)
# 1/32
self.e_stage6 = nn.Sequential(
    InvertedResidual(96,160,2,6),
    InvertedResidual(160,160,1,6),
    InvertedResidual(160,160,1,6)
)
self.e_stage7 = InvertedResidual(160,320,1,6)
```

Among them, when stride=1, the elementwise sum will be used to connect the input and output features (the left side of the figure below); when stride=2, there is no short cut to connect the input and output features (the right side of the figure below).



Have to be aware of is:

1) When $n > 1$ (that is, the number of repetitions of the bottleneck layer> 1), only the first

bottleneck layer stride is the corresponding s, and the other repeated bottleneck layer strides are all 1;

2) Only when stride=1, the output feature size is consistent with the input feature size, the elementwise sum will be used to add the output to the input;

3) When $n > 1$, the dimension is only increased in the first bottleneck layer, and the channel remains unchanged at other times. (for the dimension of the entire bottleneck layer)

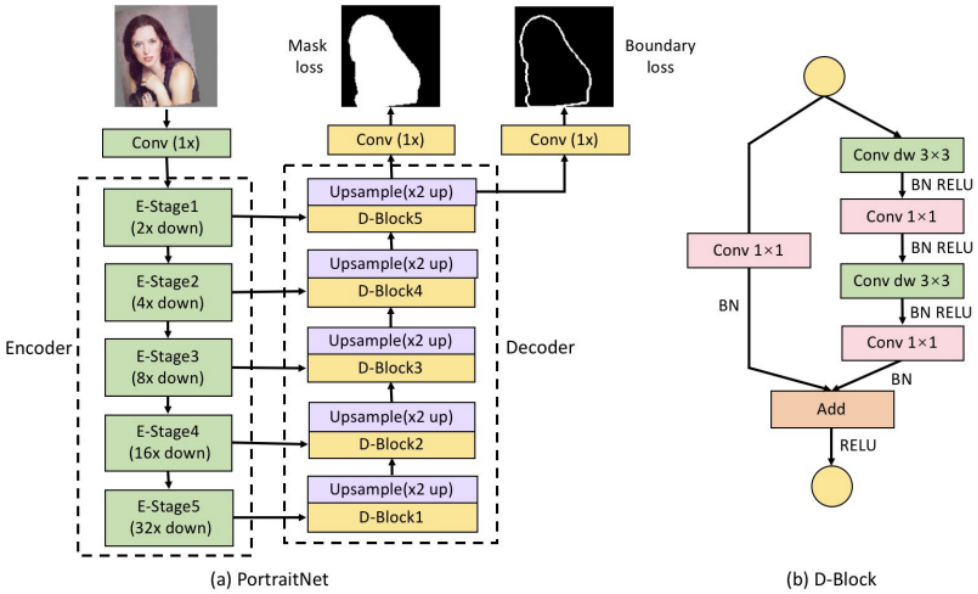
(For example, for the layer of 56x56x24 in the figure, there are 3 bottleneck layers in total, only use stride=2 in the first bottleneck layer, stride=1 in the last two bottleneck layers; the first bottleneck layer is due to the input and output size Inconsistent, so there is no short cut connection. Since the latter two have stride=1, the input and output feature sizes are the same, and the short cut will be used to sum the input and output features elementwise; in addition, only in the last 1x1conv pair feature of the first bottleneck layer For dimension enhancement, the output dimensions of the last two bottleneck layers remain unchanged (not to be confused with the dimension enhancement inside the bottleneck layer). The input feature of this layer is 56x56x24, and the output of the first bottleneck layer is 28x28x32 (the feature size is reduced, and the feature dimension is increased. , no short cut), the input and output of the second and third bottleneck layers are both 28x28x32 (at this time c=32, s=1, with short cut).) In addition, there is a k in the table. The width scaling factor is proposed in mobileNet-v1, and its function is to slim down the dimension (number of features) of each layer of the network as a whole. In MobileNetV2, when k¡1, the last 1*1 conv does not perform width scaling, otherwise it performs width scaling.

About ReLU6:

Convolution is usually followed by a ReLU non-linear activation, ReLU6 is used in MobileNet. ReLU6 is an ordinary ReLU but the maximum output is limited to 6, which is to have a good numerical resolution when the float16/int8 precision of the mobile device is low. If the activation range of ReLU is not limited, the output range is from 0 to positive infinity. If the activation value is very large and distributed in a large range, the low-precision float16/int8 cannot accurately describe such a large range. value, resulting in a loss of precision.

2.3 PortraitNet

2.3.1 Network structure



Because of the focus on speed, networking is actually quite simple. The backbone is mobile-NetV2, and the u-net structure is used as a decoder. The decoder is transposed convolution + residual block. Conv is replaced by depthwise conv. As shown in the figure above, the designed D-block is the residual block, and the 3*3 conv is replaced by the depthwise conv. The overall structure is still easy to understand.

My code implementation of D-block:

```
class D_block(nn.Layer):
    def __init__(self,input_c,output_c):
        super(D_block, self).__init__()
        self.input_c = input_c
        self.output_c = output_c

        self.block = nn.Sequential(
            nn.Conv2D(input_c,input_c,kernel_size=3,stride=1,padding=1,groups=input_c,weight_attr=ParamAttr(),bias_attr=False),
            nn.BatchNorm(input_c,param_attr=ParamAttr(),bias_attr=ParamAttr()),
            nn.ReLU(),
            nn.Conv2D(input_c,output_c,kernel_size=1,stride=1,padding=0,groups=1,weight_attr=ParamAttr(),bias_attr=False),
            nn.BatchNorm(output_c,param_attr=ParamAttr(),bias_attr=ParamAttr()),
            nn.ReLU(),
            nn.Conv2D(output_c,output_c,kernel_size=3,stride=1,padding=1,groups=output_c,weight_attr=ParamAttr(),bias_attr=False),
            nn.BatchNorm(output_c,param_attr=ParamAttr(),bias_attr=ParamAttr()),
            nn.ReLU(),
            nn.Conv2D(output_c,output_c,kernel_size=1,stride=1,padding=0,groups=1,weight_attr=ParamAttr(),bias_attr=False),
            nn.BatchNorm(output_c,param_attr=ParamAttr(),bias_attr=ParamAttr())
        )

        self.residual = nn.Sequential(
            nn.Conv2D(input_c,output_c,kernel_size=1,stride=1,padding=0,groups=1,weight_attr=ParamAttr(),bias_attr=False),
            nn.BatchNorm(output_c,param_attr=ParamAttr(),bias_attr=ParamAttr())
        )

        self.relu = nn.ReLU()

    def forward(self, inputs):
        residual = inputs
        output = self.block(inputs)
        if self.input_c != self.output_c:
            residual = self.residual(inputs)

        output += residual
        output = self.relu(output)
        return output
```

My code implementation of PortraitNet:

```
class PortraitNet(nn.Layer):
    def __init__(self,n_class=2):
        super(PortraitNet, self).__init__()

        self.e_stage0 = nn.Sequential(
            nn.Conv2D(3,32,kernel_size=3,stride=2,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False),
            nn.BatchNorm(32,param_attr=ParamAttr(),bias_attr=ParamAttr()),
            nn.ReLU()
        )
        # 1/2
        self.e_stage1 = InvertedResidual(32,16,1,1)
        # 1/4
        self.e_stage2 = nn.Sequential(
            InvertedResidual(16,24,2,6),
            InvertedResidual(24,24,1,6)
        )
        # 1/8
        self.e_stage3 = nn.Sequential(
            InvertedResidual(24,32,2,6),
            InvertedResidual(32,32,1,6),
            InvertedResidual(32,32,1,6)
        )
        # 1/16
        self.e_stage4 = nn.Sequential(
            InvertedResidual(32,64,2,6),
            InvertedResidual(64,64,1,6),
            InvertedResidual(64,64,1,6),
            InvertedResidual(64,64,1,6)
        )
        # 1/16
        self.e_stage5 = nn.Sequential(
            InvertedResidual(64,96,1,6),
            InvertedResidual(96,96,1,6),
            InvertedResidual(96,96,1,6)
        )
```

```
# 1/32
self.e_stage6 = nn.Sequential(
    InvertedResidual(96,160,2,6),
    InvertedResidual(160,160,1,6),
    InvertedResidual(160,160,1,6)
)
self.e_stage7 = InvertedResidual(160,320,1,6)

self.deconv1 = nn.Conv2DTranspose(96,96,kernel_size=4,stride=2,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False)
self.deconv2 = nn.Conv2DTranspose(32,32,kernel_size=4,stride=2,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False)
self.deconv3 = nn.Conv2DTranspose(24,24,kernel_size=4,stride=2,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False)
self.deconv4 = nn.Conv2DTranspose(16,16,kernel_size=4,stride=2,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False)
self.deconv5 = nn.Conv2DTranspose(8,8,kernel_size=4,stride=2,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False)

self.d_block1 = D_block(320,96)
self.d_block2 = D_block(96,32)
self.d_block3 = D_block(32,24)
self.d_block4 = D_block(24,16)
self.d_block5 = D_block(16,8)

self.pred = nn.Conv2D(8,n_class,kernel_size=3,stride=1,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False)
self.edge = nn.Conv2D(8,n_class,kernel_size=3,stride=1,padding=1,groups=1,weight_attr=ParamAttr(),bias_attr=False)
```

```
def forward(self, inputs):
    feature2  = self.e_stage0(inputs)
    feature2  = self.e_stage1(feature2)
    feature4  = self.e_stage2(feature2)
    feature8  = self.e_stage3(feature4)
    feature16 = self.e_stage4(feature8)
    feature16 = self.e_stage5(feature16)
    feature32 = self.e_stage6(feature16)
    feature32 = self.e_stage7(feature32)

    db1 = self.d_block1(feature32)
    up16 = self.deconv1(db1)
    db2 = self.d_block2(feature16+up16)
    up8  = self.deconv2(db2)
    db3 = self.d_block3(feature8+up8)
    up4  = self.deconv3(db3)
    db4 = self.d_block4(feature4+up4)
    up2  = self.deconv4(db4)
    db5 = self.d_block5(up2)
    up1  = self.deconv5(db5)

    return self.pred(up1),self.edge(up1)
```

2.3.2 Two extra loss

Notice the mask loss and boundary loss in the picture above. Among them, the mask loss is the binary cross entropy loss used to calculate the pixel classification. In addition to boundary loss, there is also a consistency constraint loss.

boundary loss:

First of all, in order to ensure the petiteness of the network, we cannot add additional branches just because we want to optimize the boundary, so the author adds a conv layer to the last layer. used to predict boundaries. The label of the boundary comes from the output of the canny operator for the segmentation gt. Set the line width to 4.

Because the border occupies a small part of the image, in order to avoid the appearance of

extreme sample imbalance. So the focal loss is used. lambda is set to a very small value.

$$L_m = -\sum_{i=1}^{n}(y_i log(p_i) + (1 - y_i) log(1 - p_i))$$

$$L_e = -\sum_{i=1}^{n}((1 - p_i)^{\gamma} y_i log(p_i) + p_i^{\gamma}(1 - y_i) log(1 - p_i))$$

$$L = L_m + \lambda \times L_e$$

My code implementation of Focal loss:

```
class FocalLoss(nn.Layer):
    def __init__(self, alpha=[1.]*2, gamma=2.0, ignore_index=255,smooth_rate=0,online_reweighting=False):
        super(FocalLoss, self).__init__()
        self.ignore_index = ignore_index
        self.EPS = 1e-8
        if isinstance(alpha,list):
            self.alpha=paddle.to_tensor(alpha)
        self.alpha=paddle.reshape(self.alpha,(1,1,1,2))
        self.gamma=gamma
        self.smooth_rate=smooth_rate

    def forward(self, logit, label):
        label_one_hot=nn.functional.one_hot(label,2)
        p=nn.functional.softmax(logit)*(1-2*self.EPS)+self.EPS
        loss=-paddle.pow(1-p,self.gamma)*paddle.log(p)
        loss=paddle.max(loss*label_one_hot,axis=-1)
        return paddle.mean(loss)
```

consistency constraint loss:

Selfies sometimes have different brightness under different lighting conditions, but photos with the same content, the labels of these images are the same. But the network may get different segmentation predictions because of the different pixel values of these images. In order to avoid this problem, the author proposes a consistency limit loss. The purpose is to make the impact of these lighting effects not affect the final segmentation result. resulting in a more stable effect.

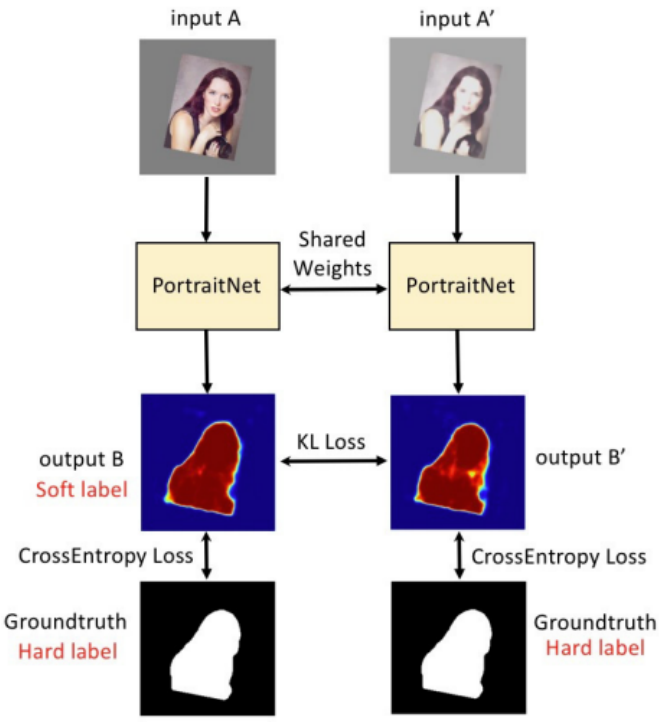


Fig. 4. Illustration of consistency constraint loss.

The schematic diagram of this auxiliary loss can be seen from the above figure. Assume that A is obtained from the original image through data enhancement. A' is obtained by changing the contrast of A, etc. The content of this picture is the same, and B and B' are obtained respectively through the network, which is the heatmap, which is the final output picture of the network. Both B and B' go to participate in the calculation of loss with GT.

Theoretically speaking, the quality of A' is worse than that of A, so the quality of B' is also worse than that of B. But A' is only obtained by simulating light changes, and the content of A' is consistent with A, so we naturally hope that B' and B are the same. To this end, the author proposes to use B as the soft label of B', and calculate the KL loss between B and B'.

$$L'_m = -\sum_{i=1}^{n}(y_i log(p_i) + (1-y_i) log(1-p_i)) - \sum_{i=1}^{n}(y_i log(p'_i) + (1-y_i) log(1-p'_i))$$

$$L_c = \frac{1}{n}\sum_{i=1}^{n} q_i \times log\frac{q_i}{q'_i} \times T^2$$

$$L = L'_m + \alpha \times L_c$$

L'_m is the loss obtained by using GT and B, B', which is the ordinary BCE.

L_c is KL loss. T is the soft label coefficient, which is used to soften the label.

In addition, B and B' are logits that have not passed softmax. The p in the loss formula will pass through softmax.

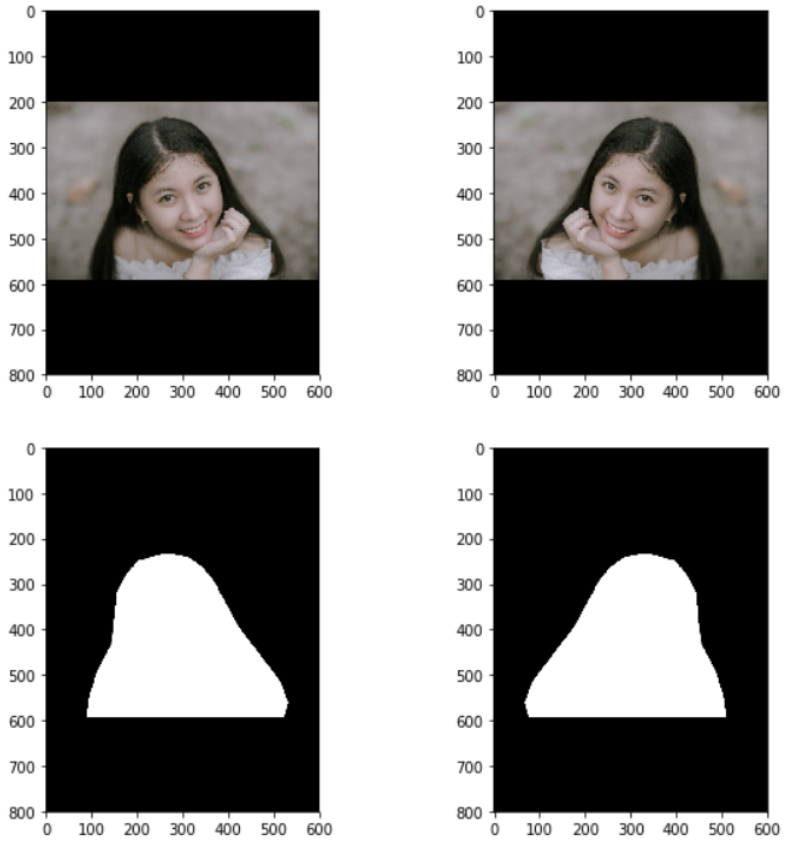
And q is obtained by softmax with a softening coefficient.

$$q_i(z_j) = \frac{e^{\frac{z_j}{T}}}{\sum_{k=1}^{K} e^{\frac{z_k}{T}}} \qquad q_i'(z_j') = \frac{e^{\frac{z_j'}{T}}}{\sum_{k=1}^{K} e^{\frac{z_k'}{T}}}$$

2.4 Data augmentation

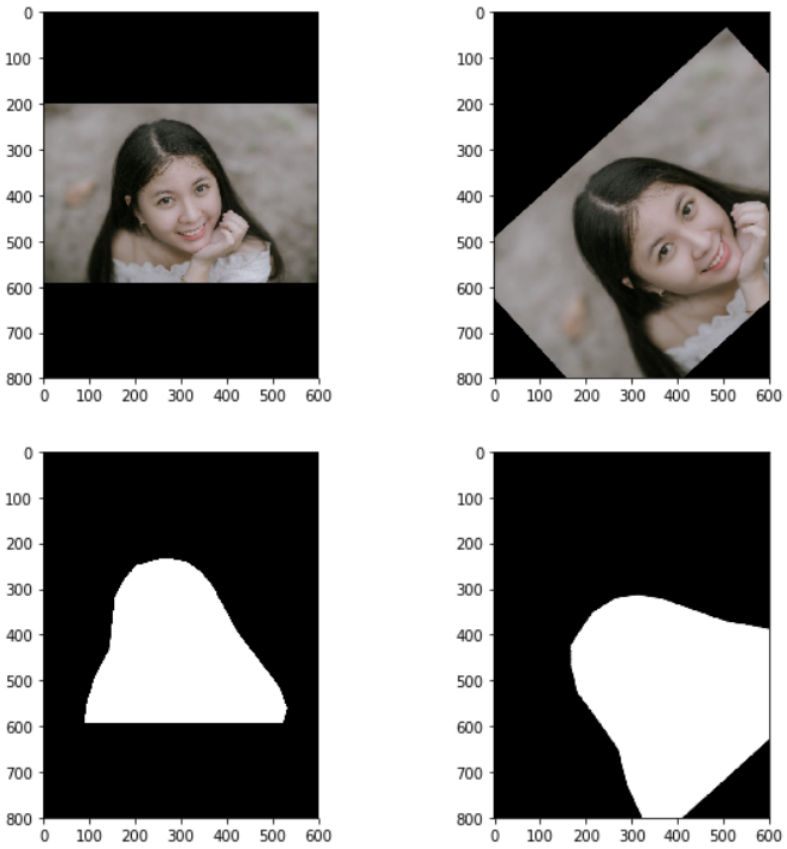
2.4.1 random flip

```
class RandomHorizontalFlip():
    def __init__(self,prob=0.5):
        self.prob=prob

    def __call__(self,img,mask):
        if np.random.uniform(0,1)<self.prob:
            img=cv2.flip(img,1)
            mask=cv2.flip(mask,1)
        return img,mask
```



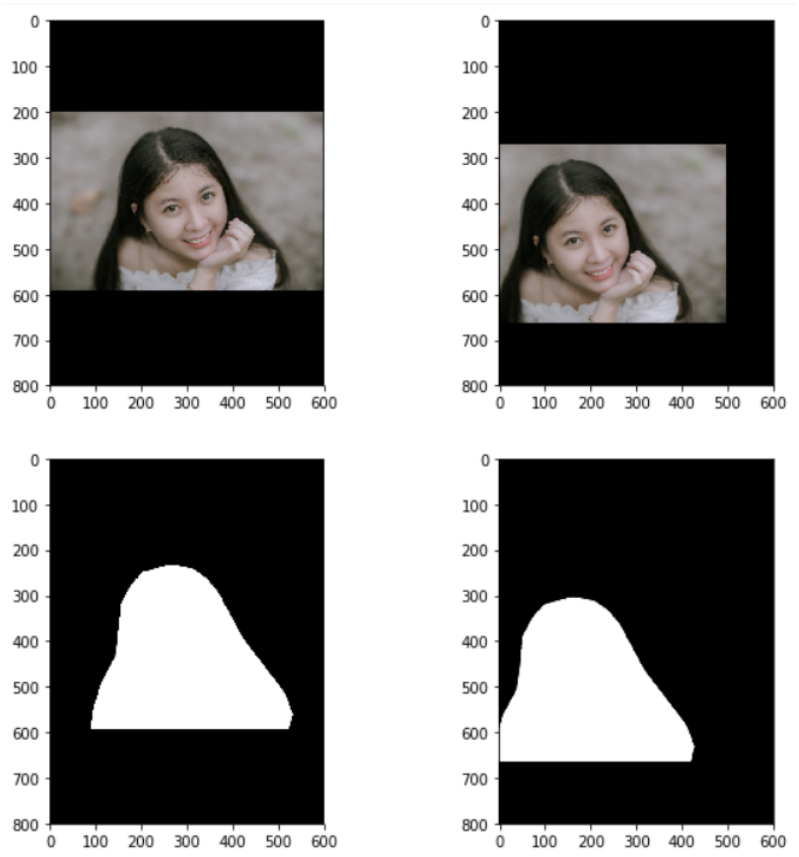
2.4.2 random rotation

```python
class RandomRotation():
    def __init__(self,max_rotation=45,resize_range=(0.5,1.5)):
        self.max_rotation=max_rotation
        self.resize_range=resize_range

    def __call__(self, img, mask):
        h,w=img.shape[:2]
        rot=np.random.uniform(-self.max_rotation,self.max_rotation)
        resize=np.random.uniform(*self.resize_range)
        M=cv2.getRotationMatrix2D((h//2,w//2), rot, resize)
        img=cv2.warpAffine(img,M,(w,h))
        mask=cv2.warpAffine(mask,M,(w,h))
        return img,mask
```



2.4.3 random translation

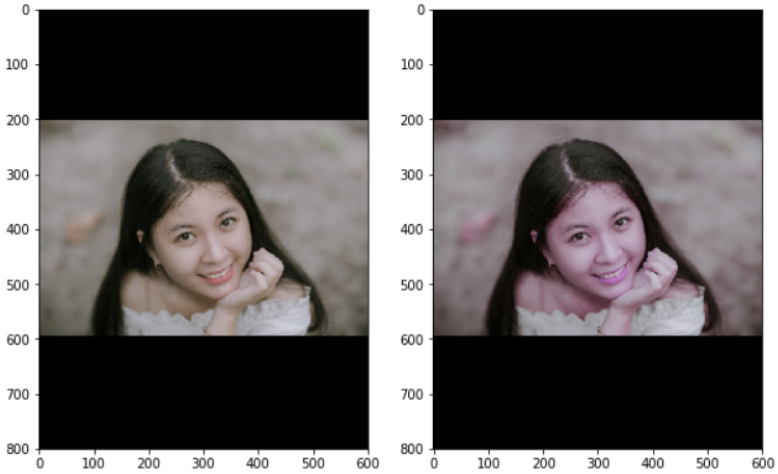
```python
class RandomTranslate():
    def __init__(self,randrange=0.25):
        self.range=randrange

    def __call__(self,img,label):
        c1=np.random.uniform(-self.range,self.range)
        c2=np.random.uniform(-self.range,self.range)
        M = np.array([[1,0,c1*img.shape[1]],[0,1,c2*img.shape[0]]])
        img = cv2.warpAffine(img,M,(img.shape[1],img.shape[0]))
        label = cv2.warpAffine(label,M,(label.shape[1],label.shape[0]))
        return img,label
```



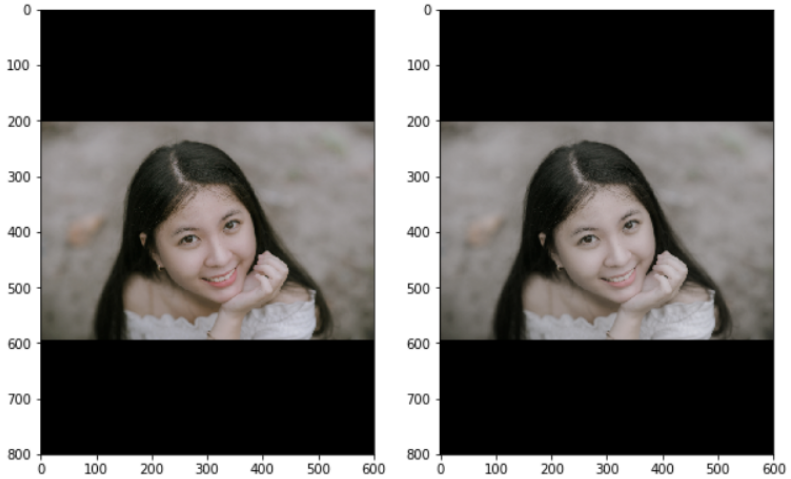
2.4.4 Randomly change hue, saturation, lightness

```python
class RandomHSV():
    def __init__(self,
                 hue_prob=0.5,hue_range=(0.4,1.7),
                 saturation_prob=0.5,saturation_range=(0.4,1.7),
                 value_prob=0.5,value_range=(0.4,1.7)
                 ):
        self.hue_prob=hue_prob
        self.hue_range=hue_range
        self.saturation_prob=saturation_prob
        self.saturation_range=saturation_range
        self.value_prob=value_prob
        self.value_range=value_range

    def __call__(self, img):
        img=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
        if np.random.uniform()<self.hue_prob:
            h=np.random.uniform(self.hue_range[0],self.hue_range[1])
            img[:,:,0]=img[:,:,0]*h
            img[:,:,0]=np.clip(img[:,:,0],0,179)
        if np.random.uniform()<self.saturation_prob:
            s=np.random.uniform(self.saturation_range[0],self.saturation_range[1])
            img[:,:,1]=img[:,:,1]*s
        if np.random.uniform()<self.value_prob:
            v=np.random.uniform(self.value_range[0],self.value_range[1])
            img[:,:,2]=img[:,:,2]*v
        img=np.clip(img,0,255)
        img=cv2.cvtColor(img,cv2.COLOR_HSV2BGR)
        return img
```

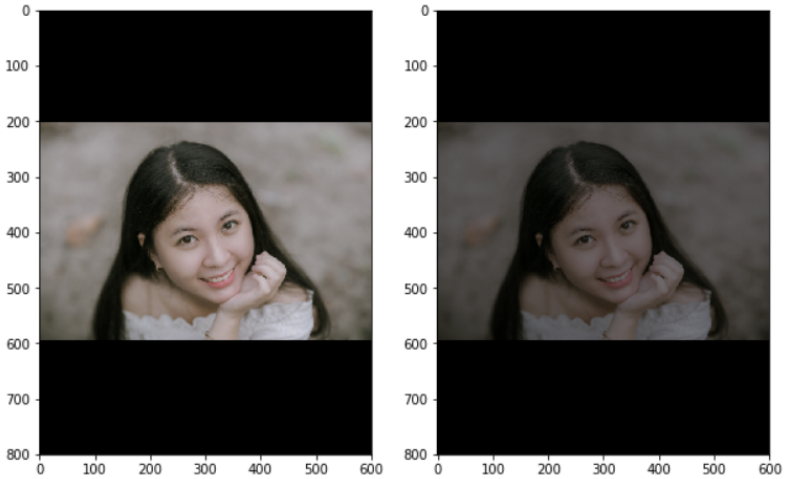
```
transform=RandomHSV(hue_prob=1, saturation_prob=0, value_prob=0)
show(transform, example)
```



```
transform=RandomHSV(hue_prob=0, saturation_prob=1, value_prob=0)
show(transform, example)
```

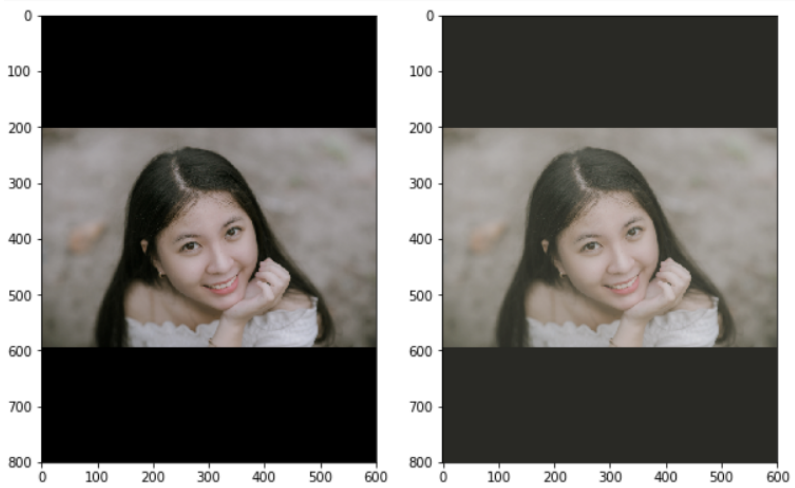


```
transform=RandomHSV(hue_prob=0, saturation_prob=0, value_prob=1)
show(transform, example)
```



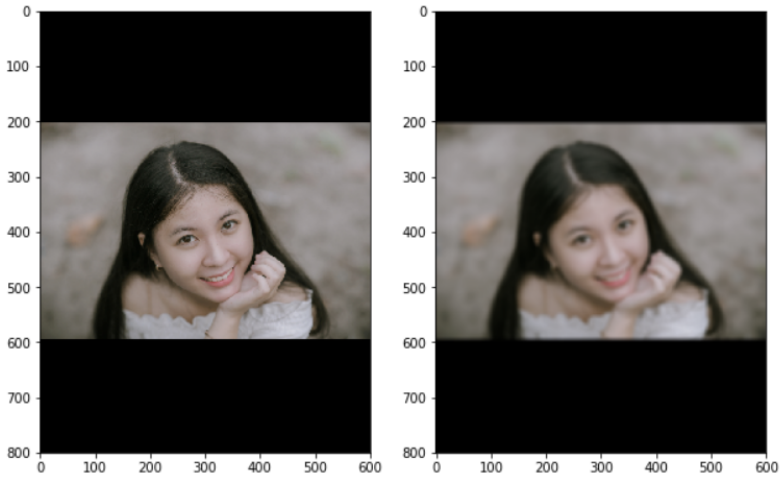
2.4.5 Randomly change the contrast

```python
class RandomContrast():
    def __init__(self,prob=0.5,randrange=(0.6,1.5)):
        self.prob=prob
        self.range=randrange

    def __call__(self,img):
        if np.random.uniform()<self.prob:
            c=np.random.uniform(*self.range)
            for i in range(3):
                maxc,minc=np.max(img[:,:,i]),np.min(img[:,:,i])
                img[:,:,i]=(img[:,:,i]-minc)*c+(128-(maxc-minc)*c/2)
            img=np.clip(img,0,255)
        return img
```



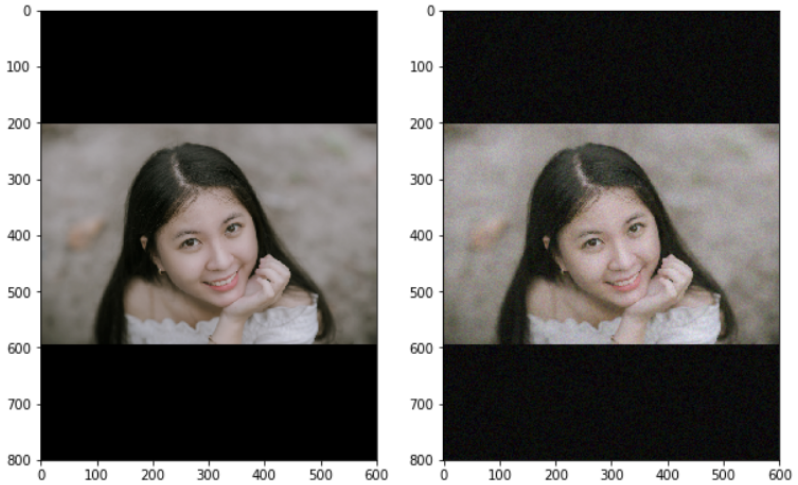
2.4.6 random blur

```python
class RandomBlur():
    def __init__(self,prob=0.5):
        self.prob=prob

    def __call__(self,img):
        if np.random.uniform()<self.prob:
            return img
        ks=np.random.randint(3, 10)
        img = cv2.blur(img, (ks, ks))
        return img
```



2.4.7 Add Gaussian noise

```python
class RandomNoise():
    def __init__(self,prob=0.5):
        self.prob=prob

    def __call__(self,img):
        if np.random.uniform()<self.prob:
            return img
        img = cv2.GaussianBlur(img, (0,0), 10)
        return img
```

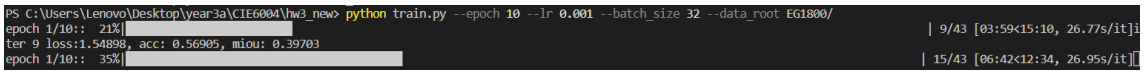


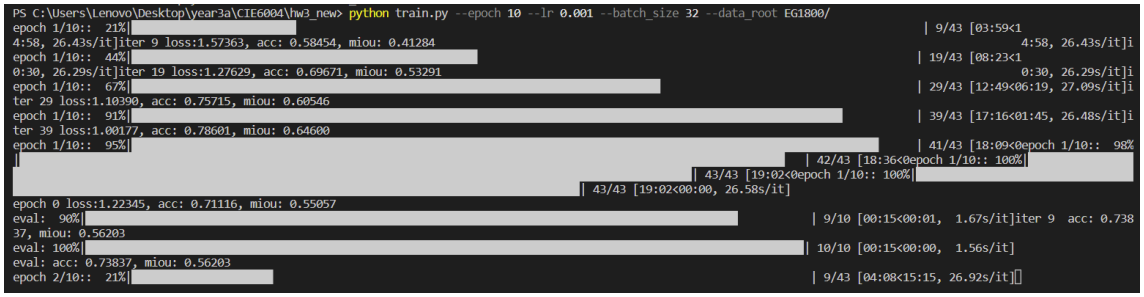
3 Execution

3.0.1 Train

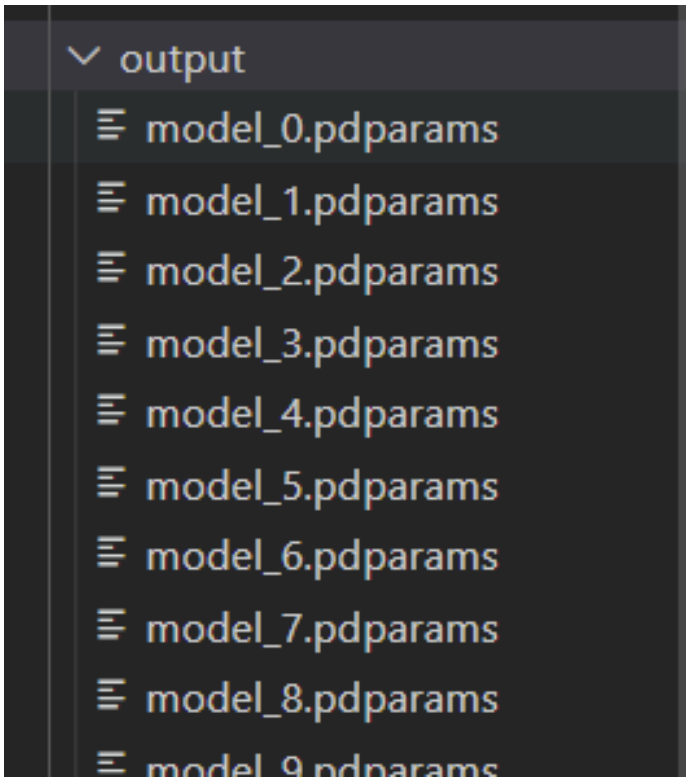
Use the following commend, you can train the model. In fact, the hyper-parameters can be adjusted by yourself.

```
python train.py --epoch 60 --lr 0.001 --batch_size 32 --data_root EG1800/
```





And finally, you will get output/model_epoch_num.pdparams files, which is the trained model.



3.0.2 Evaluation

Using the following commend, you can test the trained model. If you trained for 60 epoch, you can use the following one.

```
python evaluate.py --model_path output/model_59.pdparams --batch_size 32 --log_iter 10
--data_root EG1800/
```

The miou of my model can reach 0.9497

And if you do some image processing when you get the mask and with the original images, you can get the visual result.

4 Result

4.1 test 1

original image:

output result:

4.2 test 2

original image:

output result:

4.3 test 3

original image:

output result:

4.4 test 4

original image:

output result:

4.5 test 5

original image:

output result:

5 My feeling

As an undergraduate, this was my first exposure to the field of semantic segmentation, and I thought PortraitNet was really a very elegant approach. I feel that I have gained a lot from the whole process this time.

First of all, after my study, I completely understand the whole network of PortraitNet, including depthwise convolution, pointwise convolution, mobilenet-v2, and the construction of PortraitNet, which I think is very important for my understanding. And I also read the given paper very seriously, I think this is a very good exercise for me, and it has cultivated my ability to study independently through the paper. This time I also implemented PortraitNet from a very low level, I construct the network by myself. I think it is very helpful for me to clearly understand the problem and my ability to code.

However, I also encountered many difficulties in this study. First of all, reading a paper is not an easy process. It is difficult for me to understand this problem very well by relying on the full English paper, so I also found a lot of relevant information on the Internet, which

is very helpful for me to understand this problem. Secondly, to understand PortraitNet, I need to learn a lot of extra knowledge, such as depthwise convolution, pointwise convolution, mobilenet-v2, and the construction of PortraitNet using U-net. But by looking up some github source code, some implementations and some online blogs, I understood the details. This has improved my ability to learn quickly and problem-solve, to deeply understand a neural network. I have benefited a lot

All in all, I learned a lot through this homework!!!