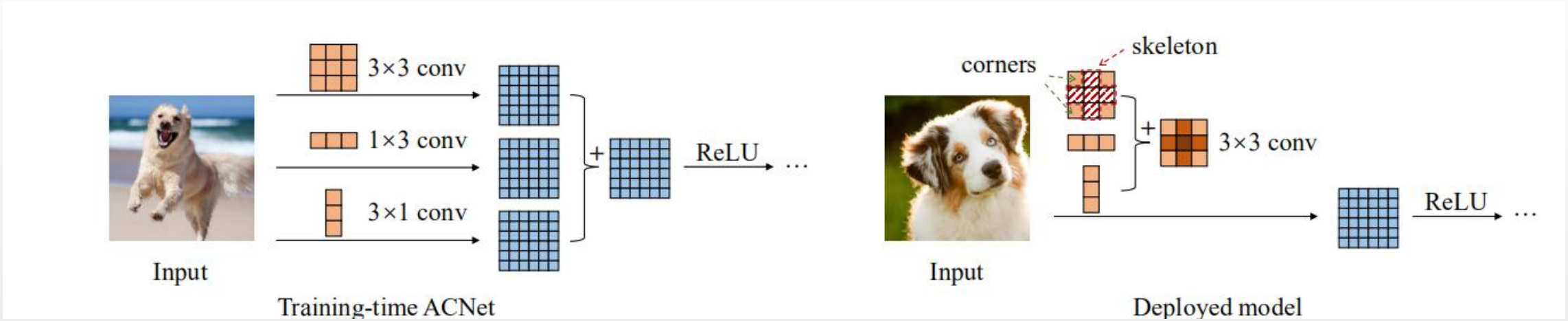
非对称卷积(Asymmetric Convolution)

异构卷积(Heterogeneous Convolution)

八度卷积(Octave Convolution)

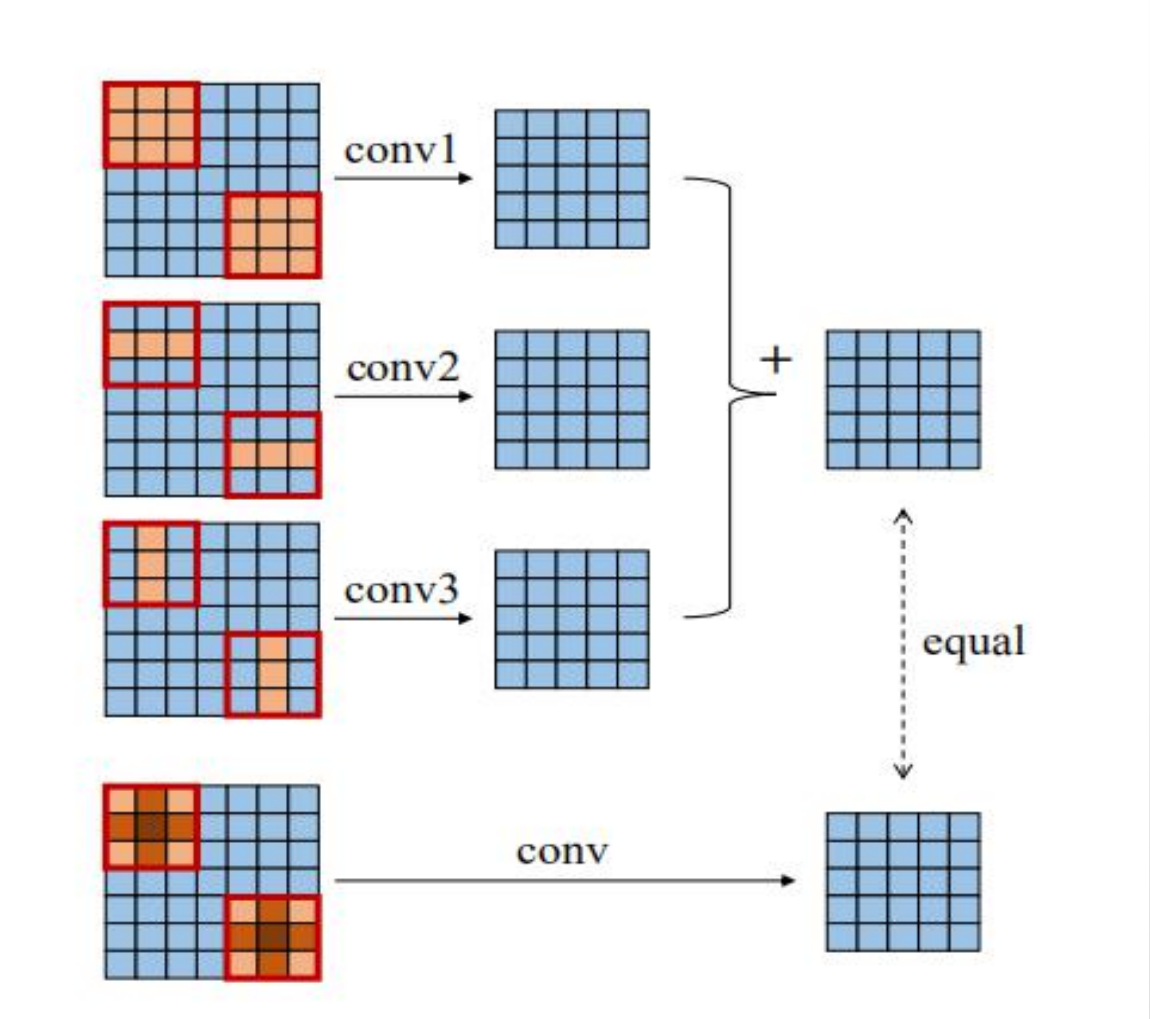
非对称卷积(Asymmetric Convolution)

李玉光



非对称卷积(Asymmetric Convolution)

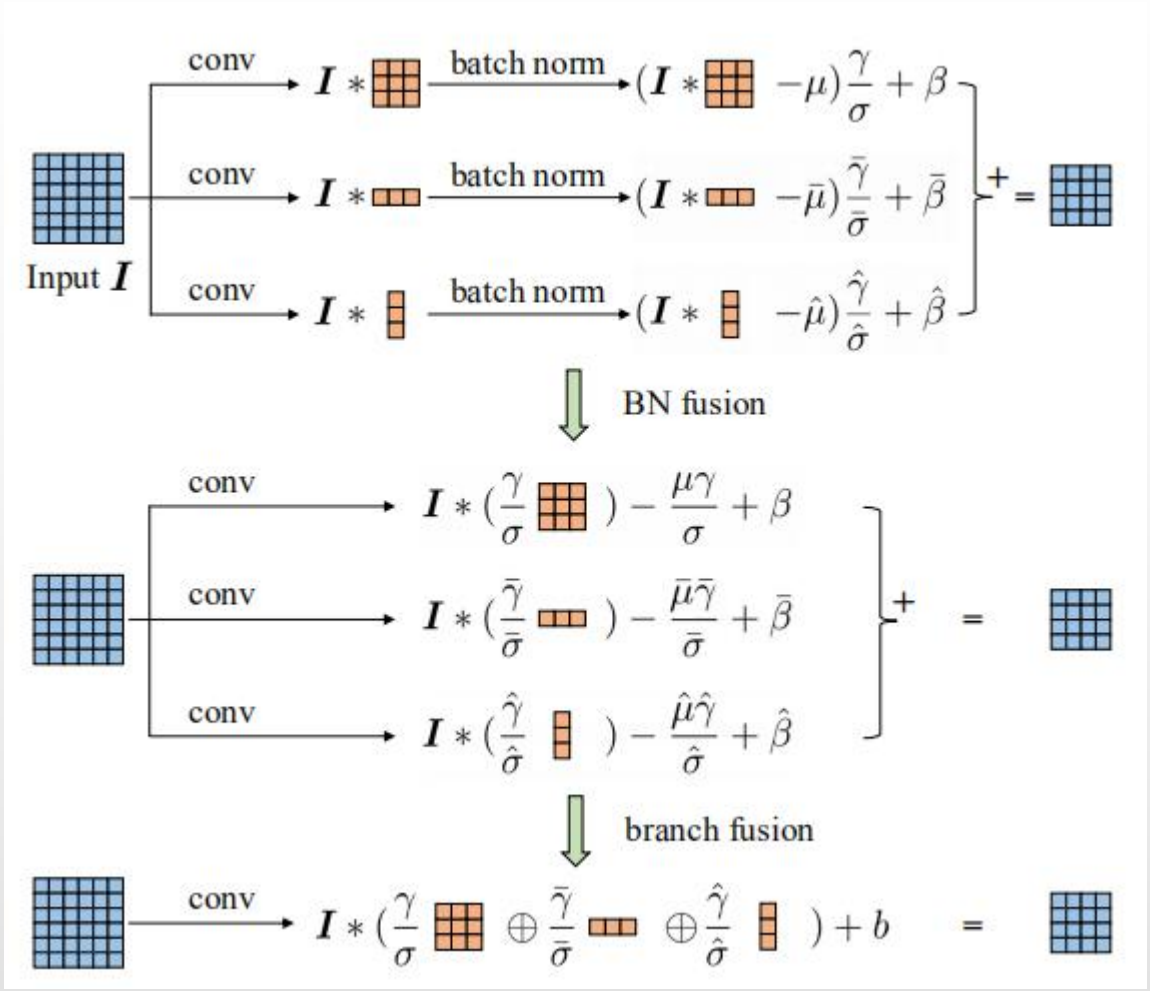
李玉光



利用卷积的可加性

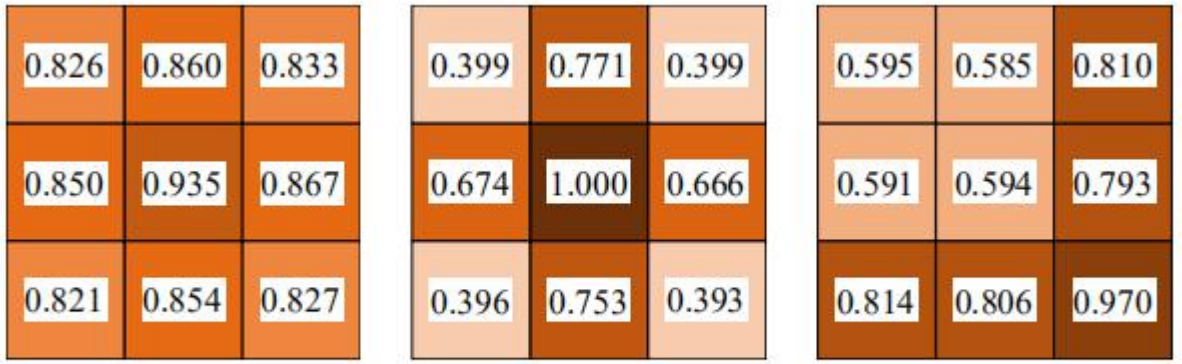
非对称卷积(Asymmetric Convolution)

李玉光

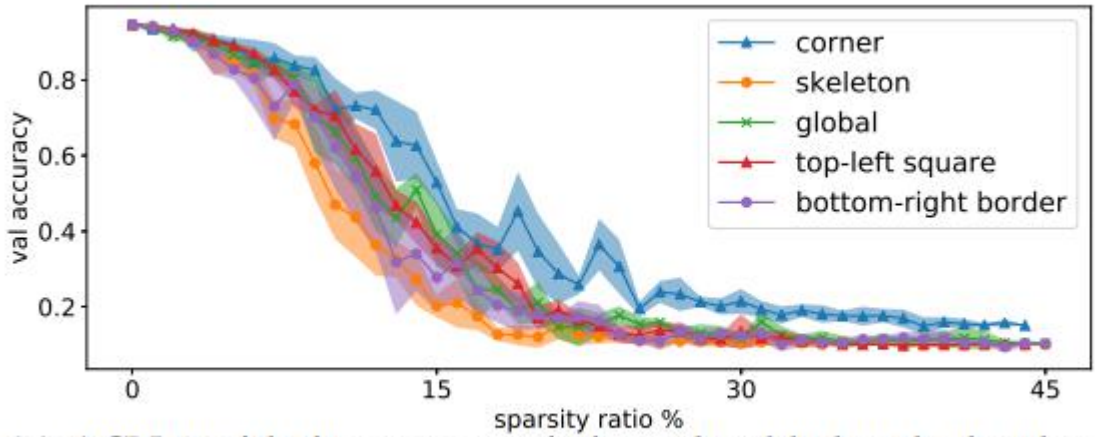


非对称卷积(Asymmetric Convolution)

李玉光



(a) Normal. (b) ACNet, skeleton. (c) ACNet, border.



(c) ACNet with the asymmetric kernels added to the border.

非对称卷积(Asymmetric Convolution)

李玉光

```python
import torch.nn as nn
import torch.nn.init as init
from custom_layers.crop_layer import CropLayer
# source: https://github.com/DingXiaoH/ACNet/blob/master/acnet/acb.py

class ACBlock(nn.Module):

    def __init__(self, in_channels, out_channels, kernel_size, stride=1, padding =0):
        super(ACBlock, self).__init__()
        self.square_conv = nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                                     kernel_size=(kernel_size, kernel_size), stride=stride,
                                     padding=padding)
        self.square_bn = nn.BatchNorm2d(num_features=out_channels, affine=use_affine)
        self.ver_conv = nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=(kernel_size, 1)
                                  stride=stride,padding=padding)

        self.hor_conv = nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=(1, kernel_size)
                                  stride=stride,padding=padding)
        self.ver_bn = nn.BatchNorm2d(num_features=out_channels, affine=use_affine)
        self.hor_bn = nn.BatchNorm2d(num_features=out_channels, affine=use_affine)
```

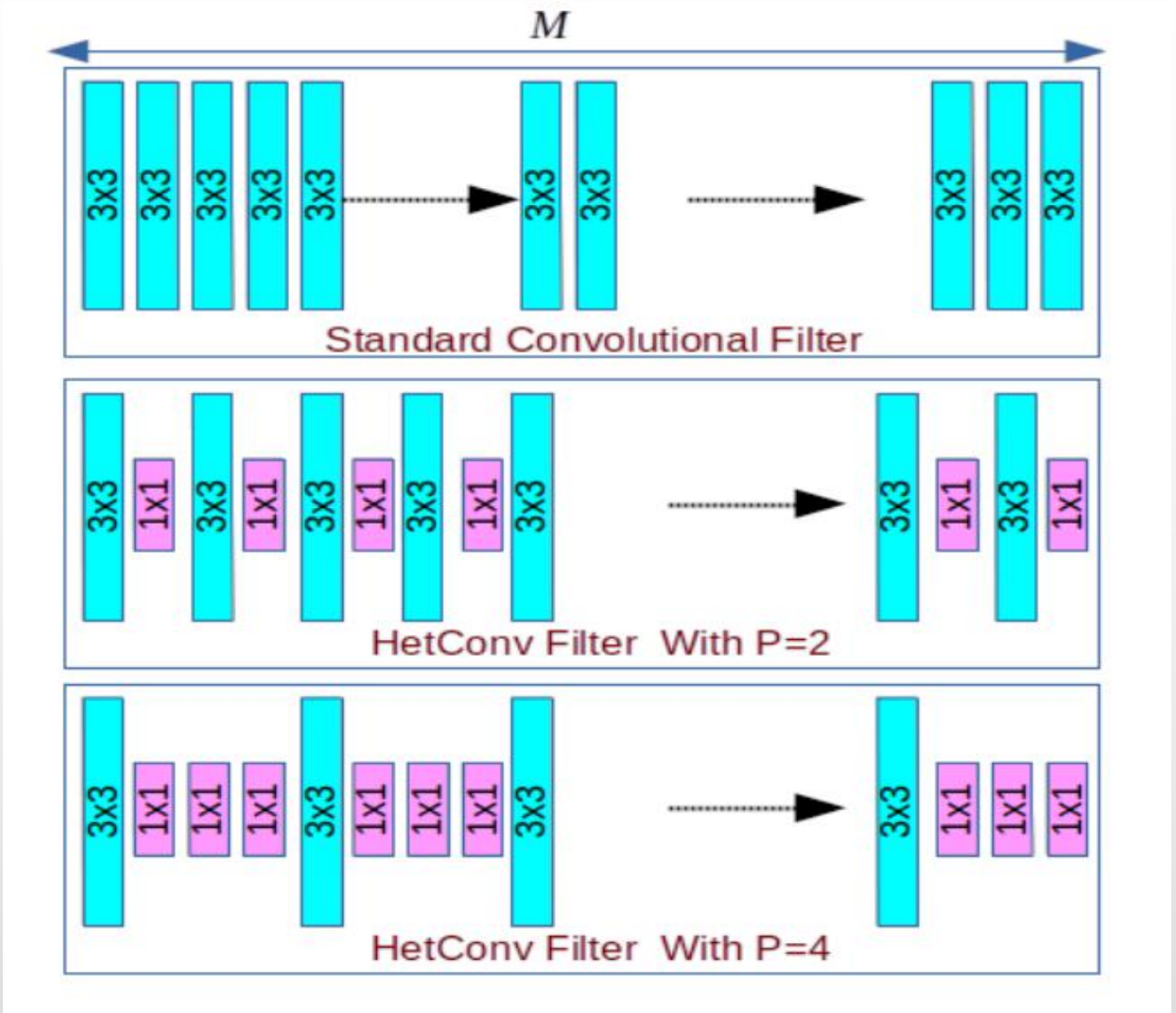
非对称卷积(Asymmetric Convolution)

李玉光

```
def forward(self, input):
    if self.deploy:
        return self.fused_conv(input)
    else:
        square_outputs = self.square_conv(input)
        square_outputs = self.square_bn(square_outputs)
        vertical_outputs = self.ver_conv(vertical_outputs)
        vertical_outputs = self.ver_bn(vertical_outputs)
        horizontal_outputs = self.hor_conv(horizontal_outputs)
        horizontal_outputs = self.hor_bn(horizontal_outputs)
        result = square_outputs + vertical_outputs + horizontal_outputs
        return result
```

异构卷积(Heterogeneous Convolution)

李玉光



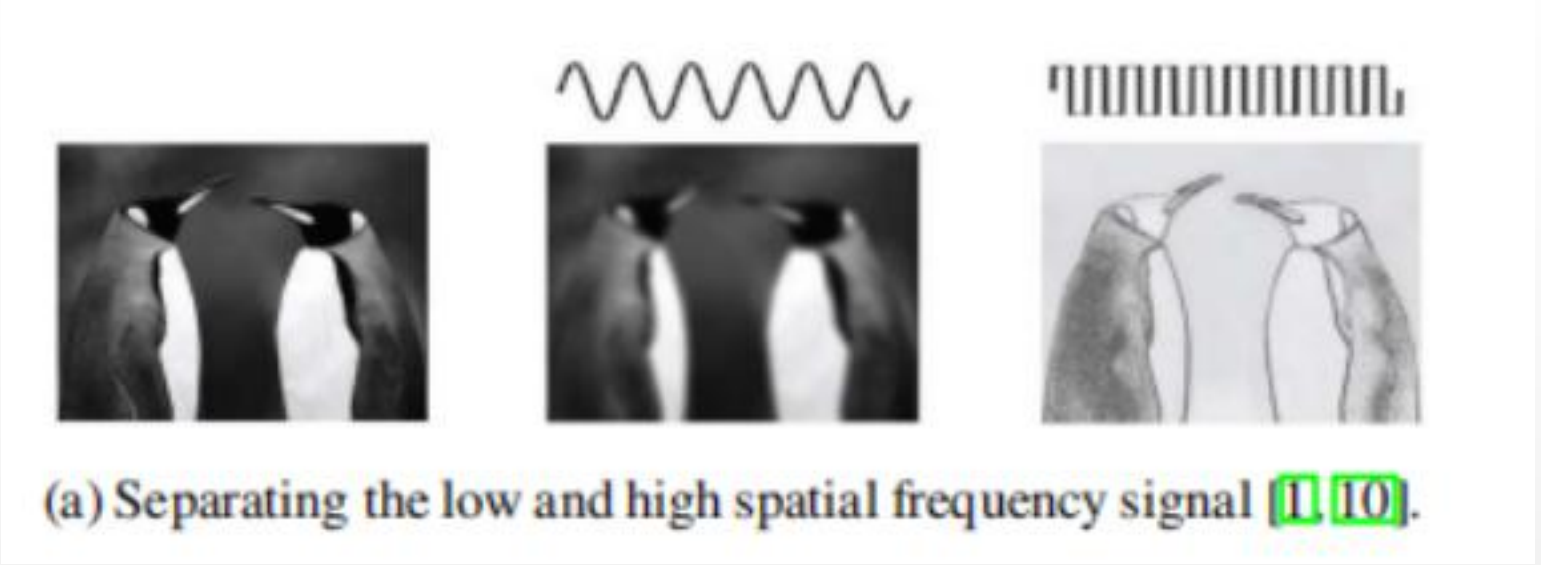
异构卷积(Heterogeneous Convolution)

李玉光

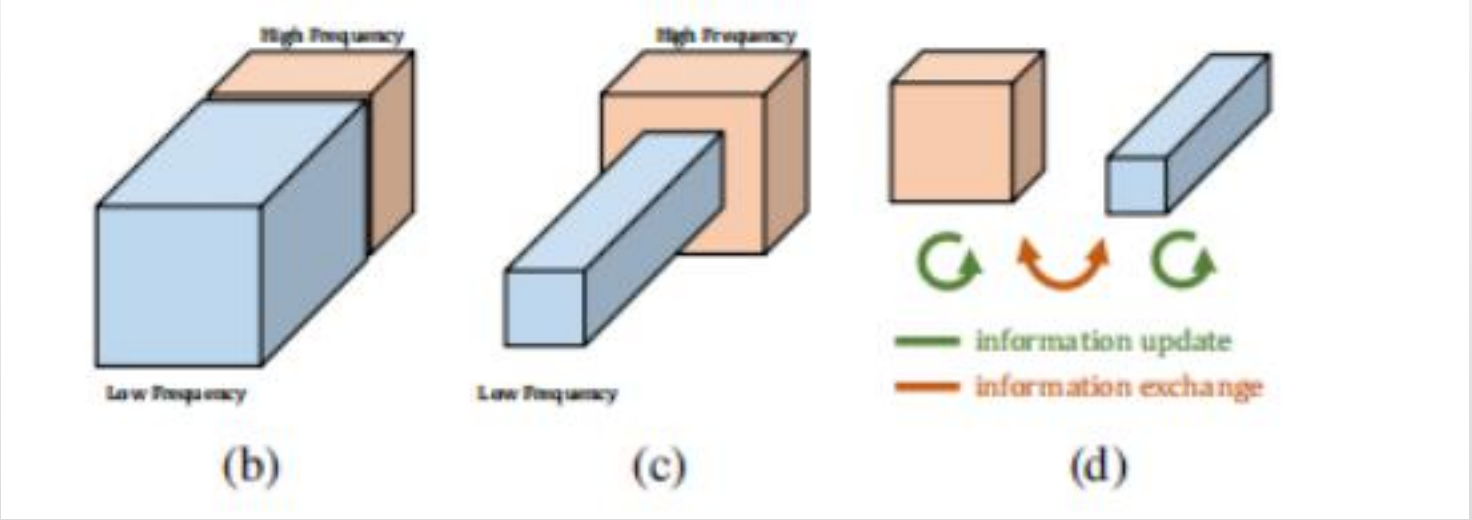
```python
class HetConv(nn.Module):
    def __init__(self, in_channels, out_channels, p):
        super(HetConv, self).__init__()
        # Groupwise Convolution
        self.gwc = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1, groups=p, bias=False)
        # Pointwise Convolution
        self.pwc = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False)

    def forward(self, x):
        return self.gwc(x) + self.pwc(x)
```

八度卷积(Octave Convolution)

李玉光

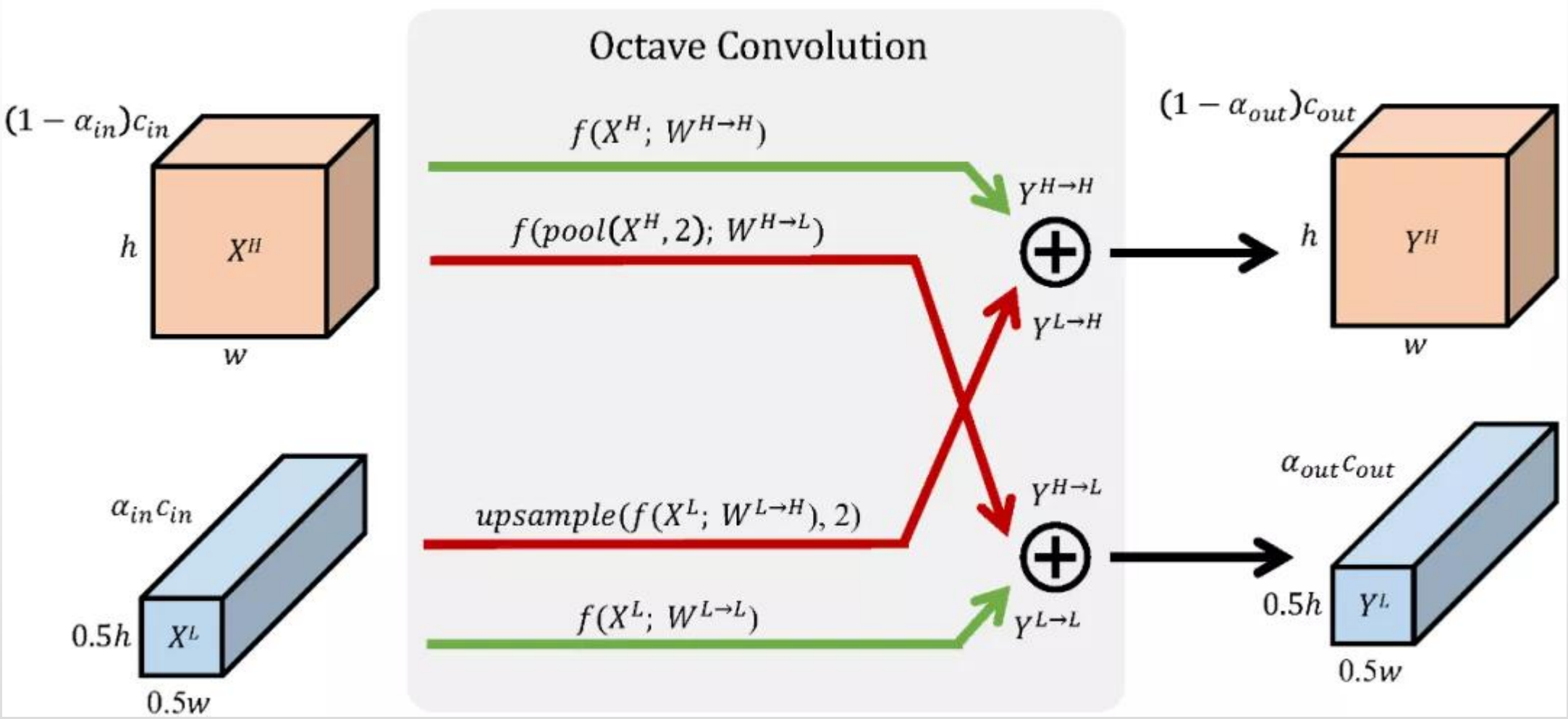


(a) Separating the low and high spatial frequency signal [1, 10].



八度卷积(Octave Convolution)

李玉光



八度卷积(Octave Convolution)

李玉光

```
class OctaveConv(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, alpha=0.5, stride=1, padding=1, dilation=1,
                 groups=1, bias=False):
        super(OctaveConv, self).__init__()
        kernel_size = kernel_size[0]
        self.h2g_pool = nn.AvgPool2d(kernel_size=(2, 2), stride=2)
        self.upsample = torch.nn.Upsample(scale_factor=2, mode='nearest')
        self.stride = stride
        self.l2l = torch.nn.Conv2d(int(alpha * in_channels), int(alpha * out_channels),
                                   kernel_size, 1, padding, dilation, groups, bias)
        self.l2h = torch.nn.Conv2d(int(alpha * in_channels), out_channels - int(alpha * out_channels),
                                   kernel_size, 1, padding, dilation, groups, bias)
        self.h2l = torch.nn.Conv2d(in_channels - int(alpha * in_channels), int(alpha * out_channels),
                                   kernel_size, 1, padding, dilation, groups, bias)
        self.h2h = torch.nn.Conv2d(in_channels - int(alpha * in_channels),
                                   out_channels - int(alpha * out_channels),
                                   kernel_size, 1, padding, dilation, groups, bias)
```

八度卷积(Octave Convolution)

李玉光

```
def forward(self, x):
    X_h, X_l = x
    if self.stride ==2:
        X_h, X_l = self.h2g_pool(X_h), self.h2g_pool(X_l)
    X_h2l = self.h2g_pool(X_h)
    X_h2h = self.h2h(X_h)
    X_l2h = self.l2h(X_l)
    X_l2l = self.l2l(X_l)
    X_h2l = self.h2l(X_h2l)

    X_l2h = self.upsample(X_l2h)
    X_h = X_l2h + X_h2h
    X_l = X_h2l + X_l2l
    return X_h, X_l
```

感谢大家！