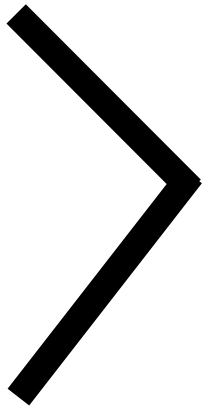
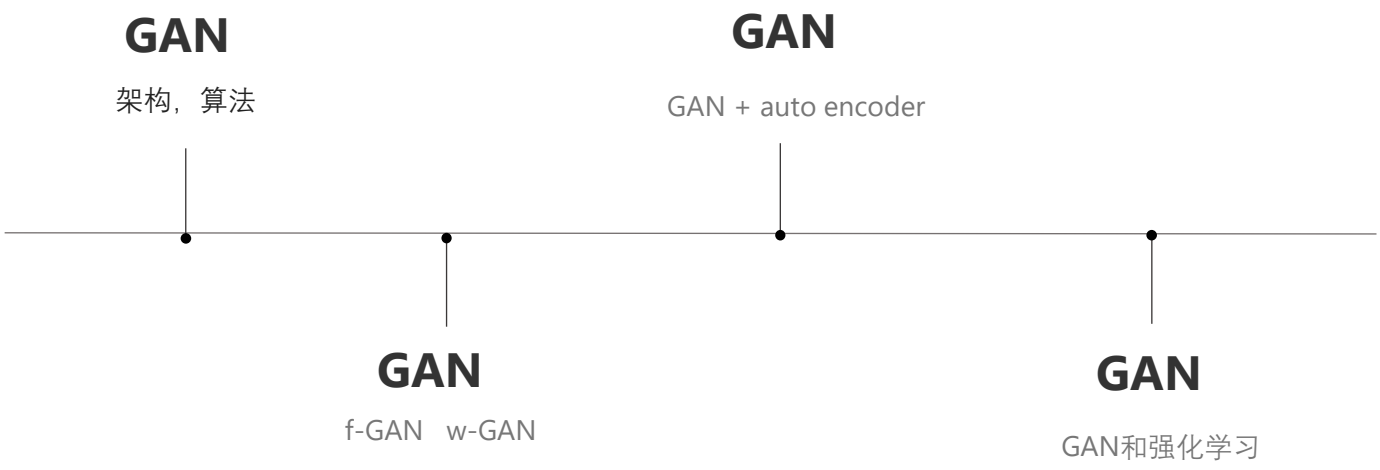
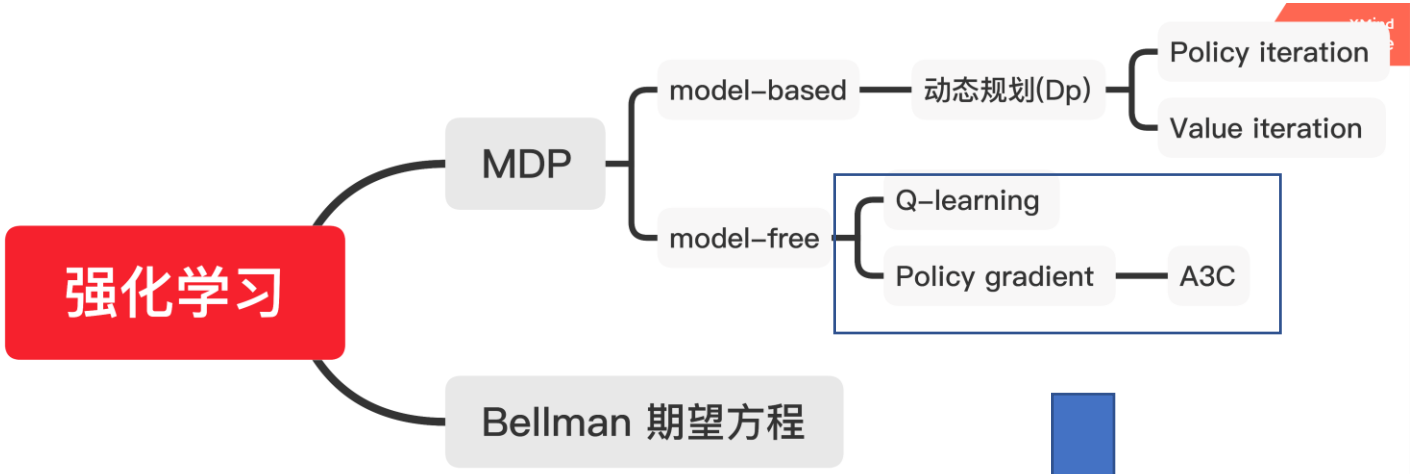
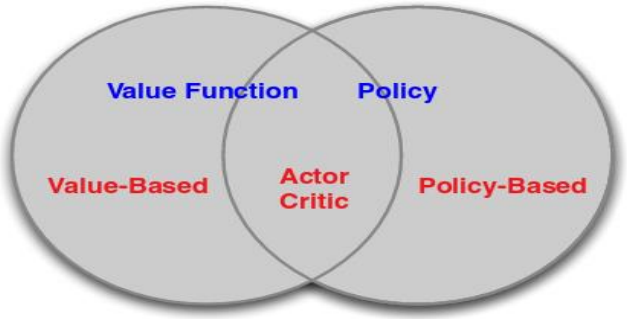
每周总结

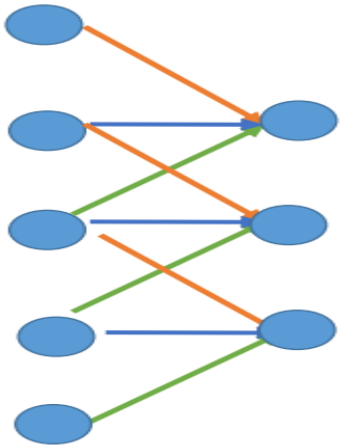



GAN
架构，算法
GAN
f-GAN w-GAN
GAN
GAN + auto encoder
GAN
GAN和强化学习

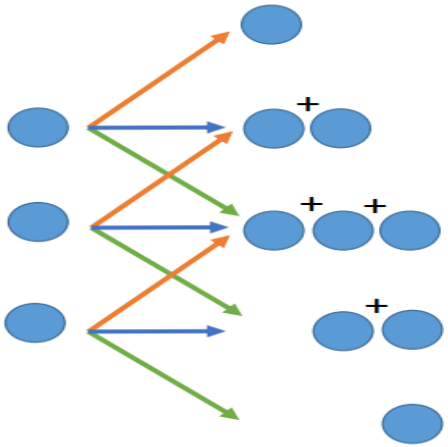




深度强化学习（DRL）是强化学习结合了深度学习而延伸出的概念

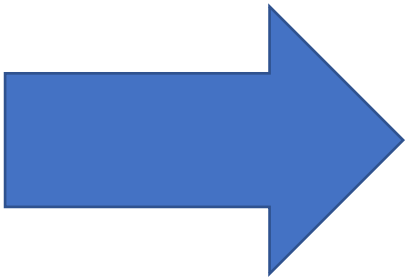


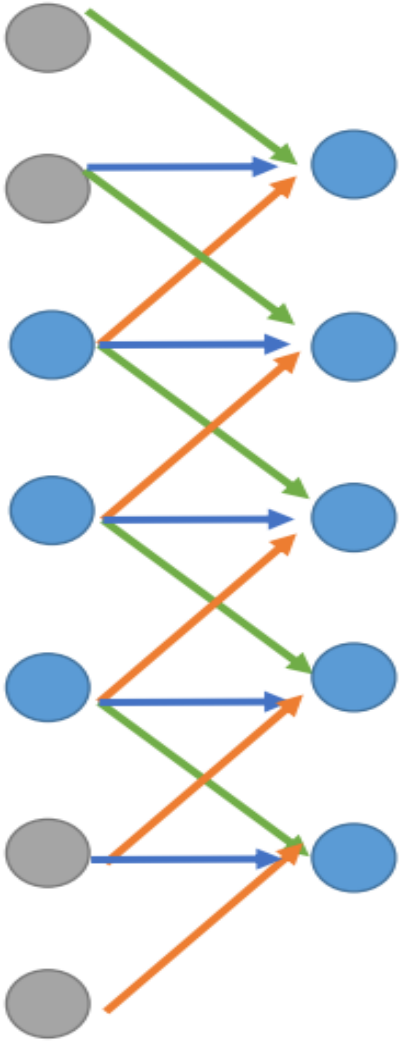
卷积



反卷积

反卷积是一种卷积





Pytorch中的反卷积

```
class torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
                               output_padding=0, groups = 1, bias=True, dilation=1)
```

- in_channels(int) – 输入信号的通道数
- out_channels(int) – 卷积产生的通道数
- kerner_size(int or tuple) - 卷积核的大小
- stride(int or tuple,optional) - 卷积步长，即要将输入扩大的倍数。
- padding(int or tuple, optional) - 输入的每一条边补充0的层数，高宽都增加2*padding
- output_padding(int or tuple, optional) - 输出边补充0的层数，高宽都增加padding
- groups(int, optional) – 从输入通道到输出通道的阻塞连接数
- bias(bool, optional) - 如果bias=True，添加偏置
- dilation(int or tuple, optional) – 卷积核元素之间的间距

$$output = (input - 1) * stride + outputpadding - 2 * padding + kernelsize$$

```
def dconv_bn_relu(in_dim, out_dim):
    return nn.Sequential(
        nn.ConvTranspose2d(in_dim, out_dim, 5, 2,
                           padding=2, output_padding=1, bias=False),
        nn.BatchNorm2d(out_dim),
        nn.ReLU())
```

Output=(Input-1)*2+1-2*2+5=2*Input

Discriminator

```
z = Variable(torch.randn(bs, z_dim)).cuda()
r_imgs = Variable(imgs).cuda()
f_imgs = G(z)

# 手动将生成图设为0，训练集的图片设为1
r_label = torch.ones((bs)).cuda()
f_label = torch.zeros((bs)).cuda()

# 用discriminator来分辨
r_logit = D(r_imgs.detach())
f_logit = D(f_imgs.detach())

# 计算loss
r_loss = criterion(r_logit, r_label)
f_loss = criterion(f_logit, f_label)
loss_D = (r_loss + f_loss) / 2

D.zero_grad()
loss_D.backward()
opt_D.step()
```

```
        self.ls = nn.Sequential(
            nn.Conv2d(in_dim, dim, 5, 2, 2), nn.LeakyReLU(0.2),# (64-5+2*2)/2+1=32
            conv_bn_lrelu(dim, dim * 2),#(32-5+2*2)/2+1=16
            conv_bn_lrelu(dim * 2, dim * 4),#(16-5+2*2)/2+1=8
            conv_bn_lrelu(dim * 4, dim * 8),#(16-5+2*2)/2+1=4
            nn.Conv2d(dim * 8, 1, 4),#(4-4)/1+1=1
            nn.Sigmoid())
        self.apply(weights_init)
    def forward(self, x):
        y = self.ls(x)
        y = y.view(-1)
        return y
```

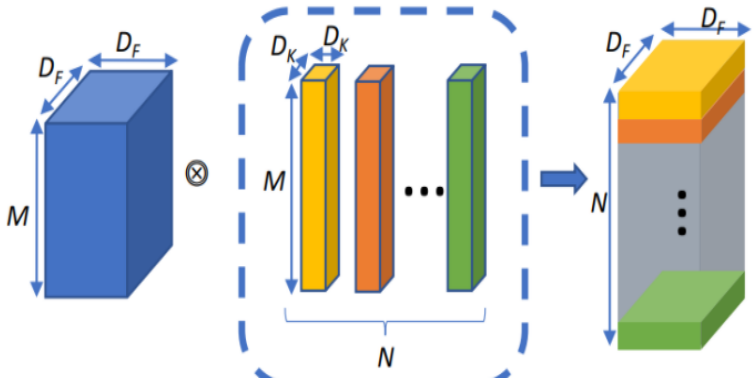
Generator

```
z = Variable(torch.randn(bs, z_dim)).cuda()
f_imgs = G(z)
f_logit = D(f_imgs)
loss_G = criterion(f_logit, r_label)
G.zero_grad()
loss_G.backward()
opt_G.step()
```

```
        self.l1 = nn.Sequential(
            nn.Linear(in_dim, dim * 8 * 4 * 4, bias=False),
            nn.BatchNorm1d(dim * 8 * 4 * 4),
            nn.ReLU())
        self.l2_5 = nn.Sequential(
            dconv_bn_relu(dim * 8, dim * 4),# (4-1)*2+1-2*2+5=8
            dconv_bn_relu(dim * 4, dim * 2),# (8-1)*2+1-2*2+5=16
            dconv_bn_relu(dim * 2, dim), #(16-1)*2+1-2*2+5=32
            nn.ConvTranspose2d(dim, 3, 5, 2, padding=2, output_padding=1),
            #(32-1)*2+1-2*2+5=64 加上bias
            nn.Tanh())
        self.apply(weights_init)
    def forward(self, x):
        y = self.l1(x)
        y = y.view(y.size(0), -1, 4, 4)
        y = self.l2_5(y)
        return y
```

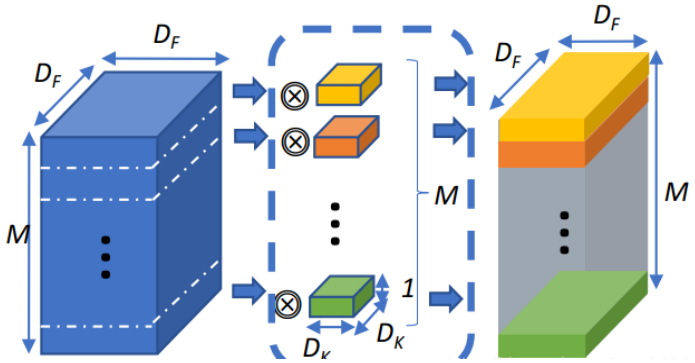
Epoch [10/10] 1115/1115 Loss_D: 0.0454 Loss_G: 4.4398 | Save some samples to ./logs/Epoch_010.jpg.

Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions



传统卷积

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} F_{k+\hat{i},l+\hat{j},m}$$



深度可分离卷积

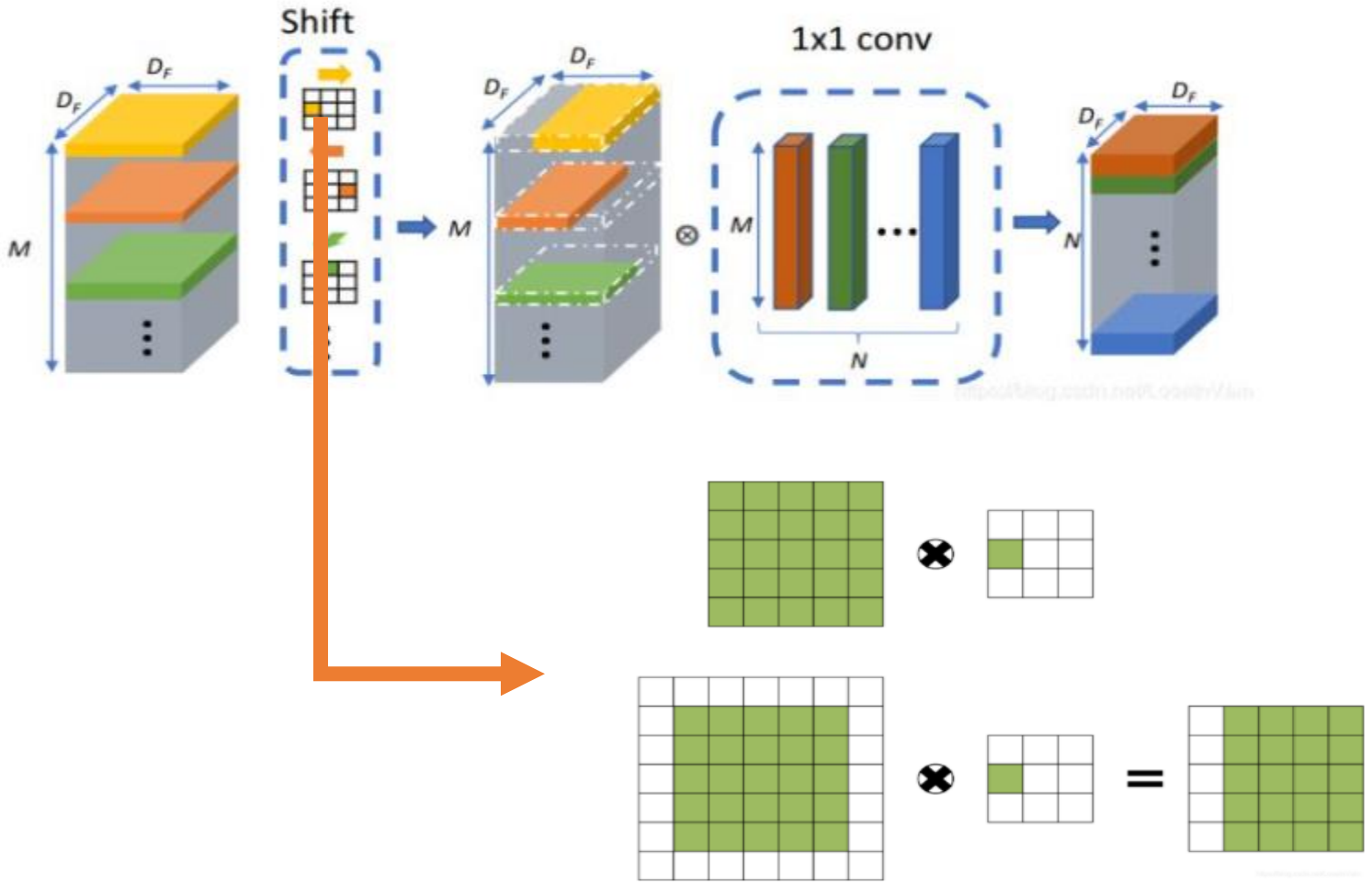
$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} F_{k+\hat{i},l+\hat{j},m}$$

本文提出的移位卷积:

$$\tilde{G}_{k,l,m} = \sum_{i,j} \tilde{K}_{i,j,m} F_{k+\hat{i},l+\hat{j},m}$$

$$\tilde{K}_{i,j,m} = \begin{cases} 1, & \text{当} i = i_m, \ j = j_m \\ 0, & \text{其他} \end{cases}$$

Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions



Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions

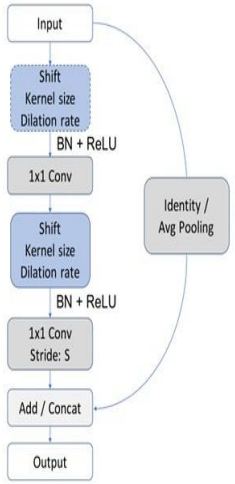


Figure 3: Illustration of the *Conv-Shift-Conv CSC* module and the *Shift-Conv-Shift-Conv* (SC^2) module.

Table 5: SHIFT OPERATION ANALYSIS USING CIFAR10 AND CIFAR100

Model	$\mathcal{E}$	ShiftResNet CIFAR10 / 100 Accuracy	ResNet (Module) CIFAR10 / 100 Accuracy	ResNet (Net) CIFAR100 Accuracy	ShiftResNet Params / FLOPs ($\times 10^6$)	Reduction Rate Params / FLOPs
20	1	86.66% / 55.62%	85.54% / 52.40%	49.58%	0.03 / 6	7.8 / 6.7
20	3	90.08% / 62.32%	88.33% / 60.61%	58.16%	0.10 / 17	2.9 / 2.5
20	6	90.59% / 68.64%	90.09% / 64.27%	63.22%	0.19 / 32	1.5 / 1.3
20	9	**91.69% / 69.82%**	91.35% / 66.25%	66.25%	0.28 / 48	0.98 / 0.85
56	1	89.71% / 65.21%	87.46% / 56.78%	56.62%	0.10 / 16	8.4 / 8.1
56	3	92.11% / 69.77%	89.40% / 62.53%	64.49%	0.29 / 45	2.9 / 2.8
56	6	92.69% / 72.13%	89.89% / 61.99%	67.45%	0.58 / 89	1.5 / 1.4
56	9	**92.74% / 73.64%**	92.01% / 69.27%	69.27%	0.87 / 133	0.98 / 0.95
110	1	90.34% / 67.84%	76.82% / 39.90%	60.44%	0.20 / 29	8.5 / 8.5
110	3	91.98% / 71.83%	74.30% / 40.52%	66.61%	0.59 / 87	2.9 / 2.9
110	6	93.17% / 72.56%	79.02% / 40.23%	68.87%	1.18 / 174	1.5 / 1.5
110	9	**92.79% / 74.10%**	92.46% / 72.11%	72.11%	1.76 / 260	0.98 / 0.97

Note that the ResNet-9 results are replaced with accuracy of the original model. The number of parameters holds for both ResNet and ShiftResNet across CIFAR10, CIFAR100. FLOPs are computed for ShiftResNet. All accuracies are Top 1. "Reduction Rate" is the original ResNet's parameters/flops over the new ShiftResNet's parameters/flops.

证实其在分类任务上精度和计算量/参数量的一个比较：
shift算子的确在计算量和参数量上有着比较大的优势