Lesson 12

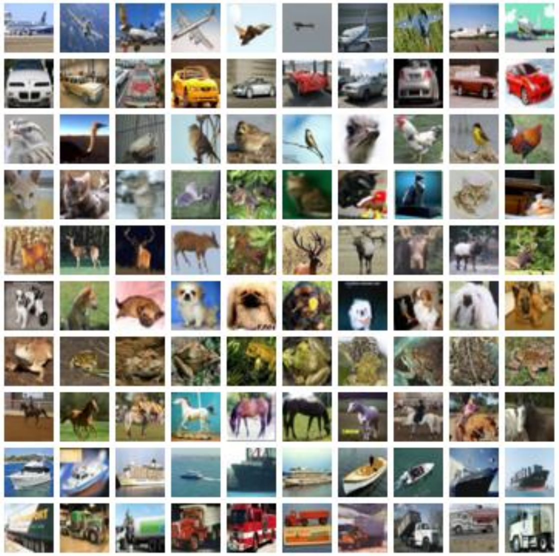
2018年8月

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

CIFAR10

- CIFAR-10 是一个包含60000张图片的数据集。其中每张照片为32*32的彩色照片，每个像素点包括RGB三个数值，数值范围 0～255。
- 与ImageNet不同，您可以相对快速地训练它
- 相对少量的数据
- 实际上很难识别图像，因为32乘32太小，不容易看到发生了什么



CIFAR10

- 需要提供训练集的均值与标准差来使标准化输入数据
- 若使用训练过的模型则可使用tfms_from_model，由于我们是从头开始训练，因此这里用tfms_from_stats

In [2]:
```
from fastai.conv_learner import *
PATH = Path("data/cifar10/")
os.makedirs(PATH,exist_ok=True)
torch.cuda.set_device(1)
```

In [4]:
```
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
stats = (np.array([ 0.4914 ,  0.48216,  0.44653]), np.array([ 0.24703,  0.24349,  0.26159]))

num_workers = num_cpus()//2
bs=256
sz=32
```

均值

标准差

In [6]:
```
tfms = tfms_from_stats(stats, sz, aug_tfms=[RandomFlip()], pad=sz//8)
data = ImageClassifierData.from_paths(PATH, val_name='test', tfms=tfms, bs=bs)
```

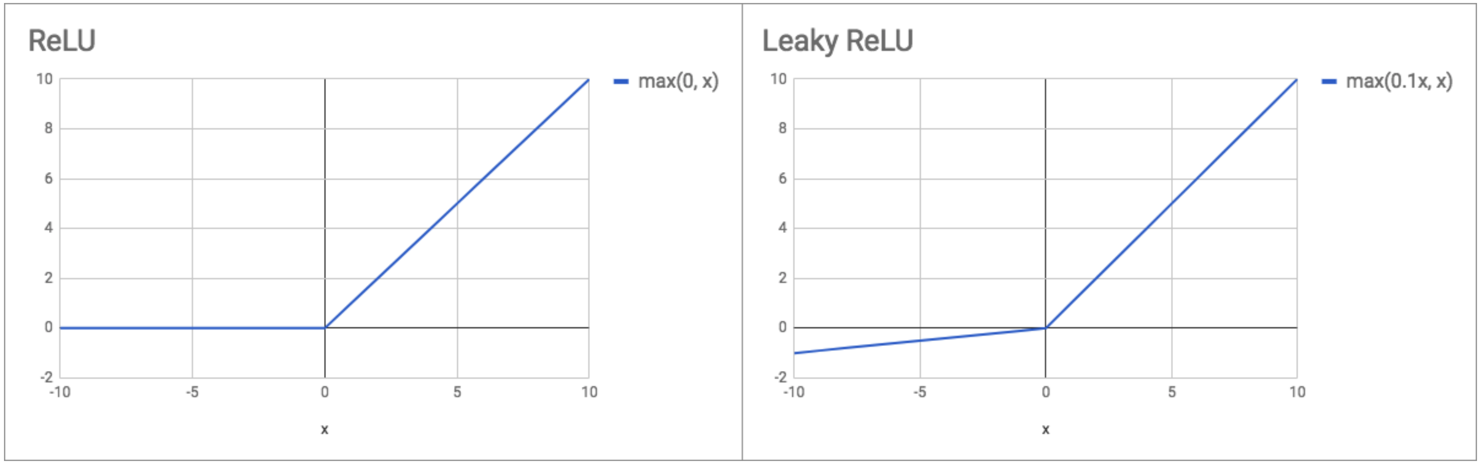
（1）RandomFlipXY()：通过翻转图片实现基本的数据增强功能；

（2）pad = sz//8：将在边缘添加填充，并允许我们正确捕捉图像的边缘

CIFAR10

- 使用BatchNorm相当于在神经网络的训练过程中对每个batch的输入数据加一个标准化处理（使其输出数据的均值接近0，其标准差接近1）
- Leaky ReLU 在$x<0$时的梯度变化但是大约0.1或0.01的常数，适合应用于较小的数据集

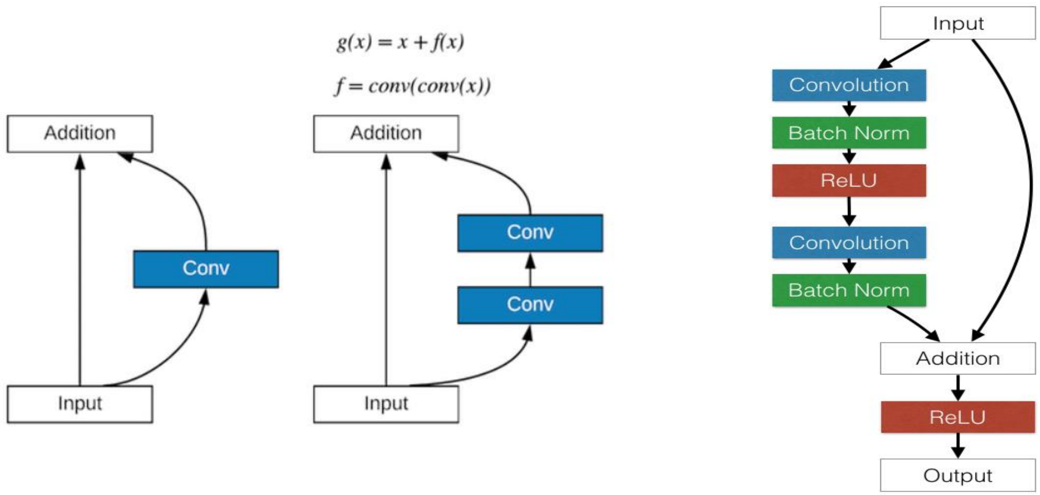
In [7]:
```
def conv_layer(ni, nf, ks=3, stride=1):
    return nn.Sequential(
        nn.Conv2d(ni, nf, kernel_size=ks, bias=False, stride=stride, padding=ks//2),
        nn.BatchNorm2d(nf, momentum=0.01),
        nn.LeakyReLU(negative_slope=0.1, inplace=True))
```



CIFAR10

- 第一个conv层将通道数量减半，然后第二个conv层将它再次加倍。通过这种方法，假设有64个channel输入，第一个转换为32个channel，然后再次恢复到64个channel。

In [8]:
```
class ResLayer(nn.Module):
    def __init__(self, ni):
        super().__init__()
        self.conv1=conv_layer(ni, ni//2, ks=1)
        self.conv2=conv_layer(ni//2, ni, ks=3)

    def forward(self, x): return x.add (self.conv2(self.conv1(x)))
```



Tips

In [7]:
```
def conv_layer(ni, nf, ks=3, stride=1):
    return nn.Sequential(
        nn.Conv2d(ni, nf, kernel_size=ks, bias=False, stride=stride, padding=ks//2),
        nn.BatchNorm2d(nf, momentum=0.01),
        nn.LeakyReLU(negative_slope=0.1, inplace=True))
```

In [8]:
```
class ResLayer(nn.Module):
    def __init__(self, ni):
        super().__init__()
        self.conv1=conv_layer(ni, ni//2, ks=1)
        self.conv2=conv_layer(ni//2, ni, ks=3)

    def forward(self, x): return x.add (self.conv2(self.conv1(x)))
```

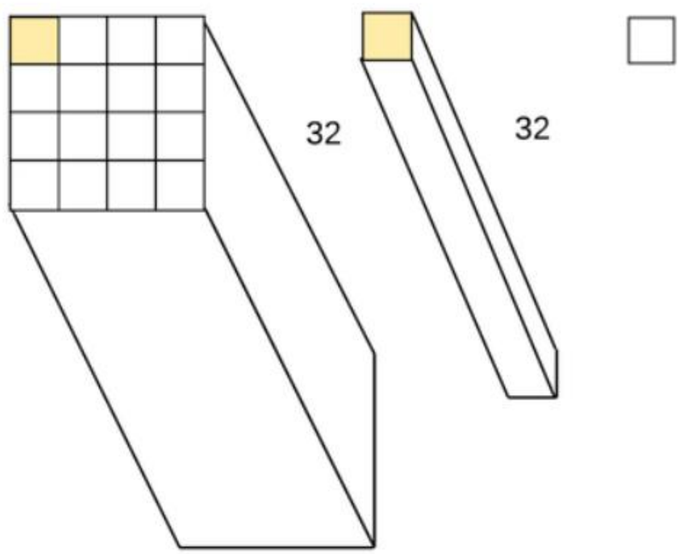
- inplace=True：可减少内存的分配
- bias=false：BatchNorm每次激活都有2个可学习的参数，故conv层不设偏置，可以减小存储及运算量
- Padding=ks//2
- add()：不用“+”减少内存使用量

CIFAR10

In [8]:
```
class ResLayer(nn.Module):
    def __init__(self, ni):
        super().__init__()
        self.conv1=conv_layer(ni, ni//2, ks=1)
        self.conv2=conv_layer(ni//2, ni, ks=3)

    def forward(self, x): return x.add (self.conv2(self.conv1(x)))
```

可更改

ks=1



Darknet

In [8]:
```
class Darknet(nn.Module):
    def make_group_layer(self, ch_in, num_blocks, stride=1):
        return [conv_layer(ch_in, ch_in*2,stride=stride)
               ] + [(ResLayer(ch_in*2)) for i in range(num_blocks)]

    def __init__(self, num_blocks, num_classes, nf=32):
        super().__init__()
        layers = [conv_layer(3, nf, ks=3, stride=1)]
        for i,nb in enumerate(num_blocks):
            layers += self.make_group_layer(nf, nb, stride=2-(i==1))
            nf *= 2
        layers += [nn.AdaptiveAvgPool2d(1), Flatten(), nn.Linear(nf, num_classes)]
        self.layers = nn.Sequential(*layers)

    def forward(self, x): return self.layers(x)
```

输入

输出

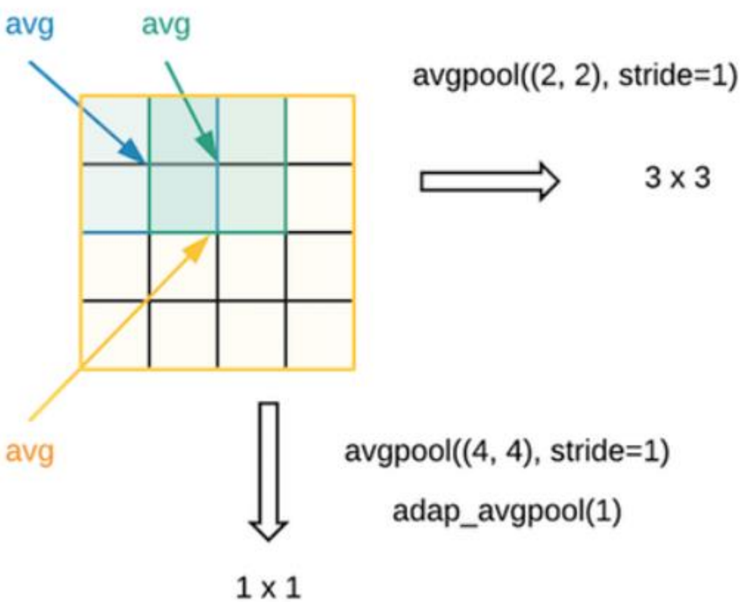
In [9]:
```
m = Darknet([1, 2, 4, 6, 3], num_classes=10, nf=32)
m = nn.DataParallel(m,[0,1])
```

- 每个group_layer包含1个conv_layer和若干ResLayer。
- channel数量加倍（64、128、256…）
- 5个group_layer,分别包含1,2,4,6,3个ResLayer

Darknet

```python
def __init__(self, num_blocks, num_classes, nf=32):
    super().__init__()
    layers = [conv_layer(3, nf, ks=3, stride=1)]
    for i,nb in enumerate(num_blocks):
        layers += self.make_group_layer(nf, nb, stride=2-(i==1))
        nf *= 2
    layers += [nn.AdaptiveAvgPool2d(1), Flatten(), nn.Linear(nf, num_classes)]
```

- 自适应平均池化：
- 输入的参数不是在几位中取平均值，而是输出的大小，函数自动计算应该采用几位



Darknet

- 学习结果

In [10]:
```
lr = 1.3
```

In [11]:
```
learn = ConvLearner.from_model_data(m, data)
learn.crit = nn.CrossEntropyLoss()
learn.metrics = [accuracy]
wd=1e-4
```

In [12]:
```
%time learn.fit(lr, 1, wds=wd, cycle_len=20, use_clr_beta=(20, 20, 0.95, 0.85))
```

Epoch 100% 20/20 [24:29<00:00, 73.49s/it]

epoch	trn_loss	val_loss	accuracy
0	2.17753	8.03552	0.0914
1	1.911743	2.512932	0.1483
2	1.731414	2.008981	0.2408
3	1.532451	2.023831	0.2801
4	1.346972	1.590166	0.434
5	1.176846	1.249545	0.547
6	1.058698	1.231115	0.5605
7	0.951039	1.232515	0.5835
8	0.849414	0.965388	0.659
9	0.772798	1.035023	0.6317
10	0.711469	0.840711	0.7109
11	0.656326	0.79456	0.7339
12	0.603825	0.914912	0.6917
13	0.542468	0.871336	0.7059
14	0.482631	0.590993	0.8007
15	0.395154	0.49495	0.8296
16	0.335288	0.418388	0.8562
17	0.296967	0.389469	0.8666
18	0.275888	0.350594	0.8849
19	0.244197	0.335619	0.8866

```
CPU times: user 59min 35s, sys: 7min 7s, total: 1h 6min 43s
Wall time: 24min 29s
```

Out[12]: [0.3356193162918091, 0.8866]

In [14]:
```
%time learn.fit(lr, 1, wds=wd, cycle_len=30, use_clr_beta=(20, 20, 0.95, 0.85))
```

Epoch 100% 30/30 [34:35<00:00, 69.17s/it]

epoch	trn_loss	val_loss	accuracy
0	0.372571	0.610885	0.8022
1	0.437792	0.867244	0.7353
2	0.473754	0.812246	0.7562
3	0.492188	0.884789	0.7195
4	0.48333	1.590372	0.5707
5	0.486836	0.698641	0.7698
6	0.484747	0.878449	0.7257
7	0.478203	1.625334	0.5654
8	0.477593	1.06544	0.6986
9	0.47769	1.105098	0.676
10	0.458395	1.072295	0.6947
11	0.46237	0.736145	0.7695
12	0.453079	0.674007	0.7735
13	0.440993	0.621825	0.7899
14	0.43662	0.730272	0.7648
15	0.419402	0.609936	0.7918
16	0.40889	0.677556	0.7751
17	0.386015	0.657854	0.7771
18	0.365494	0.630309	0.7935
19	0.349784	0.668576	0.785
20	0.326889	0.462847	0.8402
21	0.296274	0.469789	0.8408
22	0.248786	0.445246	0.849
23	0.187022	0.288453	0.9031
24	0.147661	0.291171	0.9041
25	0.124157	0.293738	0.9041
26	0.114031	0.276364	0.9124
27	0.095924	0.254817	0.9187
28	0.079033	0.251929	0.9199
29	0.070981	0.245937	0.9239

```
CPU times: user 1h 27min 57s, sys: 10min 22s, total: 1h 38min 20s
Wall time: 34min 35s
```

Out[14]: [0.2459366262435913, 0.9239]

目录 Contents

WGAN

- 论文推荐：
  - Wasserstein GAN （WGAN）
  - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- 生成式对抗网络（GAN）：一种深度学习模型，包含两个模块：生成模型和判别模型，通过互相博弈学习产生相当好的输出。

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$

  end for
  - Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

WGAN

- GAN最开始提出后存在着训练困难、生成器和判别器的loss无法指示训练进程、生成样本缺乏多样性等问题
- WGAN基于GAN算法流程进行了四点的改进：
  - 判别器最后一层去掉
  - 生成器和判别器的loss不取log
  - 每次更新判别器参数之后把他们的绝对值截到不超过一个固定常数c
  - 不要用基于动量的优化算法（Adam等），推荐RMSProp

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, \dots, n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

- 参考网址：https://zhuanlan.zhihu.com/p/25071913

WGAN

- 数据预处理

```
In [3]:  PATH = Path('data/lsun/')
         IMG_PATH = PATH/'bedroom'
         CSV_PATH = PATH/'files.csv'
         TMP_PATH = PATH/'tmp'
         TMP_PATH.mkdir(exist_ok=True)
```

```
In [69]: files = PATH.glob('bedroom/**/*.jpg')

         with CSV_PATH.open('w') as fo:
             for f in files: fo.write(f'{f.relative_to(IMG_PATH)},0\n')
```

```
In [5]:  # Optional - sampling a subset of files
         CSV_PATH = PATH/'files_sample.csv'
```

```
In [79]: files = PATH.glob('bedroom/**/*.jpg')

         with CSV_PATH.open('w') as fo:
             for f in files:
                 if random.random()<0.1: fo.write(f'{f.relative_to(IMG_PATH)},0\n')
```

A1 | f(x) Σ = | 7/7/7/777fb65649376

	A	B
1	7/7/7/777fb656493762afc6420a66308711d62d51d8d9.jpg	0
2	7/7/7/777f871ff52c5058bdf36d0d9be823bff1250156.jpg	0
3	7/7/7/777789da29a32326434b915d19f5e2445dcae967.jpg	0
4	7/7/7/77753606d0e85f8cc122797b16dd673377626f9f.jpg	0
5	7/7/7/7771e6c13301065a10a20b169f7ec7dc338e4fcb.jpg	0
6	7/7/7/7776e11f6d208e55263f64524537d24cefc7286e.jpg	0
7	7/7/2/77230cc540d361d0b204ea79ee9cab42ad0200c4.jpg	0
8	7/7/2/772ad851321c978ee7fb9c4ecad1ad992f2204a4.jpg	0
9	7/7/2/77207462235cd3175caf94ada59b00d79616aa98.jpg	0
10	7/7/2/7720305f8fc82aeb111142bc3debfb0a779edc17.jpg	0
11	7/7/5/775efaf64b94cc684da2a6ce26f2b941825627ee.jpg	0
12	7/7/5/7754d1099a941bdabdff07feafca437c2a37d692.jpg	0
13	7/7/5/775b1b1ad915502ee247e6b99ab52fbd4d8d124d.jpg	0
14	7/7/5/77510f645fedbd9a4b5bce7d1ef074cef19ba455.jpg	0
15	7/7/5/77547266c20ce58dd2b29bd9ed0801f408429205.jpg	0
16	7/7/5/7755a86fcc677a1e47bbe36985b6ed5ea3a40d69.jpg	0
17	7/7/b/77b5faac76ed3b8961e1b765ffd5659be9658c8f.jpg	0
18	7/7/b/77b7b35184b58825633b1c69ceaeb75028db4e37.jpg	0
19	7/7/b/77be1c2ce9bafbe4b158adc8f9e706d2b6a3a234.jpg	0
20	7/7/b/77b0daa64c616647d557073d477e0fe64e954300.jpg	0
21	7/7/b/77bfd2a9eebffbdf363cac313f662d498cd3cb50.jpg	0
22	7/7/b/77b020bca702fdaf63800eaa94b29a8481876ff9.jpg	0
23	7/7/b/77b1b4550e77ef7ef863512ea5e8704c4df3e008.jpg	0
24	7/7/b/77b2b0f4f8a82a67ad726a35e20dc3c7a25a0d86.jpg	0
25	7/7/b/77bafb49132b6159d47066e93308b0a2f3d7fa28.jpg	0
26	7/7/4/774ae1d30a0635ddda5dc8b1041a183989229903.jpg	0
27	7/7/4/7745f07176f6bf14b3760d2708ca3ce2fd7a8fb8.jpg	0
28	7/7/4/774d4382bcfa10a38456d3e413e81e6030537589.jpg	0
29	7/7/4/77418566401487848283b1039040bc842bedbc4f.jpg	0
30	7/7/4/77464c561ee32992824cab061ab8da2ecd21b626.jpg	0
31	7/7/4/7744b20e4499adb1e776d1d855380401705c92f4.jpg	0
32	7/7/4/774f1939fb8b57f97d6e90254138d839e0122e5a.jpg	0
33	7/7/4/774f4672fccfa39352c4e433139b271cda56ad48.jpg	0
34	7/7/4/774b9942a73b18cdaa571aa73b7acb7b285c26a9.jpg	0
35	7/7/c/77cf3cf009a44b00c8bb205b1bad0d20e8bce335.jpg	0
36	7/7/c/77c45d062f4551845e842be18a811522e4b58573.jpg	0

files_sample

WGAN

- 定义convblock

In [4]:
```python
class ConvBlock(nn.Module):
    def __init__(self, ni, no, ks, stride, bn=True, pad=None):
        super().__init__()
        if pad is None: pad = ks//2//stride
        self.conv = nn.Conv2d(ni, no, ks, stride, padding=pad, bias=False)
        self.bn = nn.BatchNorm2d(no) if bn else None
        self.relu = nn.LeakyReLU(0.2, inplace=True)

    def forward(self, x):
        x = self.relu(self.conv(x))
        return self.bn(x) if self.bn else x
```

- 定义判别模型：图像越真实，输出结果值越低

In [5]:
```python
class DCGAN_D(nn.Module):
    def __init__(self, isize, nc, ndf, n_extra_layers=0):
        super().__init__()
        assert isize % 16 == 0, "isize has to be a multiple of 16"

        self.initial = ConvBlock(nc, ndf, 4, 2, bn=False)
        csize,cndf = isize/2,ndf
        self.extra = nn.Sequential(*[ConvBlock(cndf, cndf, 3, 1)
                                    for t in range(n_extra_layers)])

        pyr_layers = []
        while csize > 4:
            pyr_layers.append(ConvBlock(cndf, cndf*2, 4, 2))
            cndf *= 2; csize /= 2
        self.pyramid = nn.Sequential(*pyr_layers)

        self.final = nn.Conv2d(cndf, 1, 4, padding=0, bias=False)

    def forward(self, input):
        x = self.initial(input)
        x = self.extra(x)
        x = self.pyramid(x)
        return self.final(x).mean(0).view(1)
```

输入图片尺寸

额外的convblock

步长为2，通过循环使网格大小不大于4*4

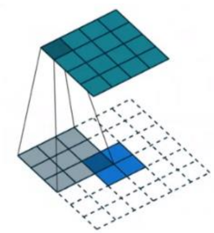
Channel=1
输出4*4*1的tensor

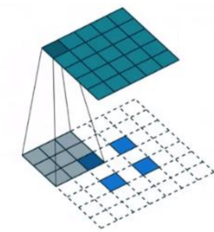
输出结果取均值

WGAN

- 定义生成模型：反卷积方法

In [6]:
```python
class DeconvBlock(nn.Module):
    def __init__(self, ni, no, ks, stride, pad, bn=True):
        super().__init__()
        self.conv = nn.ConvTranspose2d(ni, no, ks, stride, padding=pad, bias=False)
        self.bn = nn.BatchNorm2d(no)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.relu(self.conv(x))
        return self.bn(x) if self.bn else x
```





	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1				**Data**					**Filter**				**Result**								
2				1	2	3	4		0.1	0.2	0.3		13.1	15.1							
3				6	7	8	9		0.2	0.5	0.4		23.1	25.1							
4				11	12	13	14		-0.1	0.3	0.1										
5				16	17	18	19														
6																					
7		**Padded result**							**Deconv filter**				**Result**					**Error**			
8		0	0	0	0	0	0		0.7687	2E-04	0.678		2.73	2.89	3.57	4.452		-1.73	-0.89	-0.57	-0.45
9		0	0	0	0	0	0		0.0953	0.029	0.092		6.01	6.53	8	8.839		-0.01	0.467	0.002	0.161
10		0	0	13.1	15.1	0	0		0.2948	-0.02	0.208		11	13.2	13	14		9E-05	-1.2	0.009	-0
11		0	0	23.1	25.1	0	0						15.7	17	17.8	19.3		0.348	-0.01	0.238	-0.3
12		0	0	0	0	0	0														
13		0	0	0	0	0	0													**Total**	6.373

- 参考网站：http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

WGAN

- 定义生成模型：输入随机向量，输出为图片（wide*height*channel）

In [7]:
```python
class DCGAN_G(nn.Module):
    def __init__(self, isize, nz, nc, ngf, n_extra_layers=0):
        super().__init__()
        assert isize % 16 == 0, "isize has to be a multiple of 16"

        cngf, tisize = ngf//2, 4
        while tisize!=isize: cngf*=2; tisize*=2
        layers = [DeconvBlock(nz, cngf, 4, 1, 0)]

        csize, cndf = 4, cngf
        while csize < isize//2:
            layers.append(DeconvBlock(cngf, cngf//2, 4, 2, 1))
            cngf //= 2; csize *= 2

        layers += [DeconvBlock(cngf, cngf, 3, 1, 1) for t in range(n_extra_layers)]
        layers.append(nn.ConvTranspose2d(cngf, nc, 4, 2, 1, bias=False))
        self.features = nn.Sequential(*layers)

    def forward(self, input): return F.tanh(self.features(input))
```

步长为2，size倍增，channel减半

利用tanh使输出值稳定在-1,1之间

WGAN

In [8]:
```
bs,sz,nz = 64,64,100
```
nz：size of noise vector

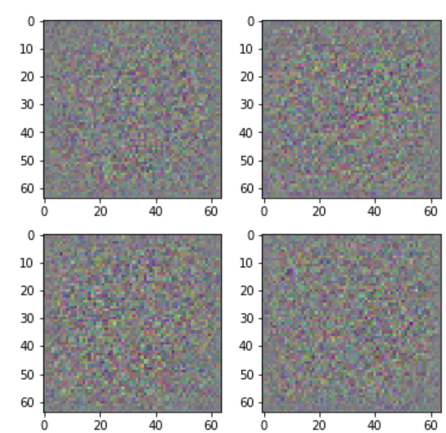
In [9]:
```
tfms = tfms_from_stats(inception_stats, sz)
md = ImageClassifierData.from_csv(PATH, 'bedroom', CSV_PATH, tfms=tfms, bs=128,
                                  skip_header=False, continuous=True)
```

In [10]:
```
md = md.resize(128)
```

In [15]:
```
netG = DCGAN_G(sz, nz, 3, 64, 1).cuda()
netD = DCGAN_D(sz, 3, 64, 1).cuda()
```
建立生成模型和判别模型

In [16]:
```
def create_noise(b): return V(torch.zeros(b, nz, 1, 1).normal_(0, 1))
```

In [17]:
```
preds = netG(create_noise(4))
pred_ims = md.trn_ds.denorm(preds)

fig, axes = plt.subplots(2, 2, figsize=(6, 6))
for i,ax in enumerate(axes.flat): ax.imshow(pred_ims[i])
```
随机生成噪声图



In [18]:
```
def gallery(x, nc=3):
    n,h,w,c = x.shape
    nr = n//nc
    assert n == nr*nc
    return (x.reshape(nr, nc, h, w, c)
              .swapaxes(1,2)
              .reshape(h*nr, w*nc, c))
```

In [19]:
```
optimizerD = optim.RMSprop(netD.parameters(), lr = 1e-4)
optimizerG = optim.RMSprop(netG.parameters(), lr = 1e-4)
```
采用RMSprop优化方法

WGAN

- Train函数

```
In [20]: def train(niter, first=True):
             gen_iterations = 0
             for epoch in trange(niter):
                 netD.train(); netG.train()
                 data_iter = iter(md.trn_dl)
                 i,n = 0,len(md.trn_dl)
                 with tqdm(total=n) as pbar:
                     while i < n:
                         set_trainable(netD, True)
                         set_trainable(netG, False)
                         d_iters = 100 if (first and (gen_iterations < 25) or (gen_iterations % 500 == 0)) else 5
                         j = 0
                         while (j < d_iters) and (i < n):
                             j += 1; i += 1
                             for p in netD.parameters(): p.data.clamp_(-0.01, 0.01)
                             real = V(next(data_iter)[0])
                             real_loss = netD(real)
                             fake = netG(create_noise(real.size(0)))
                             fake_loss = netD(V(fake.data))
                             netD.zero_grad()
                             lossD = real_loss-fake_loss
                             lossD.backward()
                             optimizerD.step()
                             pbar.update()

                         set_trainable(netD, False)
                         set_trainable(netG, True)
                         netG.zero_grad()
                         lossG = netD(netG(create_noise(bs)))
                         lossG.backward()
                         optimizerG.step()
                         gen_iterations += 1

                 print(f'Loss_D {to_np(lossD)}; Loss_G {to_np(lossG)}; '
                       f'D_real {to_np(real_loss)}; Loss_D_fake {to_np(fake_loss)}')
```

number of epochs

制作进度条

判别网络损失

生成网络损失

Tips

- 在进行训练时，必须将对应模块设置为train model；在评估时，设置为evaluation model：因为在train model下，batch norm 更新参数，并且可以dropout；但在evaluation model下却是不可以的；

- set_trainable(netG,Flase)；set_trainable(netD,True) ：保持netG的参数不动，只更新netD的参数；

- 模型中的参数始终保持在-0.01和0.01的小范围内。

WGAN

- 学习结果

```
In [21]: torch.backends.cudnn.benchmark=True

In [22]: train(1, False)

     0%|          | 0/1 [00:00<?, ?it/s]
   100%|██████████| 18957/18957 [19:48<00:00, 10.74it/s]
   Loss_D [-0.67574]; Loss_G [0.08612]; D_real [-0.1782]; Loss_D_fake [0.49754]
   100%|██████████| 1/1 [19:49<00:00, 1189.02s/it]

In [23]: fixed_noise = create_noise(bs)

In [24]: set_trainable(netD, True)
         set_trainable(netG, True)
         optimizerD = optim.RMSprop(netD.parameters(), lr = 1e-5)
         optimizerG = optim.RMSprop(netG.parameters(), lr = 1e-5)

In [25]: train(1, False)

     0%|          | 0/1 [00:00<?, ?it/s]
   100%|██████████| 18957/18957 [23:31<00:00, 13.43it/s]
   Loss_D [-1.01657]; Loss_G [0.51333]; D_real [-0.50913]; Loss_D_fake [0.50744]
   100%|██████████| 1/1 [23:31<00:00, 1411.84s/it]

In [26]: netD.eval(); netG.eval();
         fake = netG(fixed_noise).data.cpu()
         faked = np.clip(md.trn_ds.denorm(fake),0,1)

         plt.figure(figsize=(9,9))
         plt.imshow(gallery(faked, 8));
```

目录 Contents

cycle-GAN

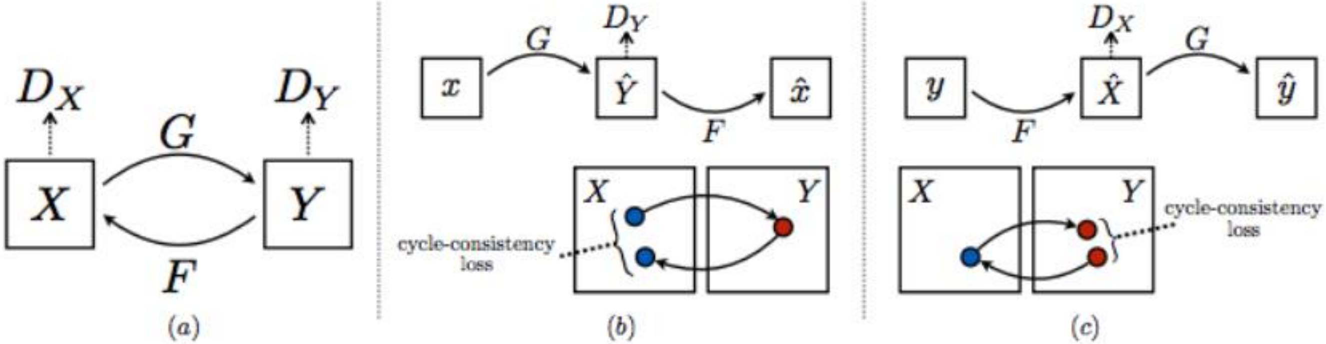


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

cycle-GAN

- GAN损失

Adversarial Loss

We apply adversarial losses [15] to both mapping functions. For the mapping function $G : X \to Y$ and its discriminator D_Y, we express the objective as:

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))], \tag{1}$$

- 循环一致损失

We can incentivize this behavior using a *cycle consistency loss*:

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]. \tag{2}$$

cycle-GAN

- 最终目标损失：
  - 识别马匹的GAN损失
  - 识别斑马的GAN损失
  - 循环一致性损失
- 最大化鉴别器区分的能力，同时最小化发生器的能力

Our full objective is:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) =& \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ &+ \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ &+ \lambda \mathcal{L}_{\text{cyc}}(G, F), \end{aligned} \tag{3}$$

where λ controls the relative importance of the two objectives. We aim to solve:

$$G^*, F^* = \arg \min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y). \tag{4}$$

cycle-GAN

- 代码部分

```
In [2]: from fastai.conv_learner import *
        from fastai.dataset import *

In [3]: from cgan.options.train_options import *

In [4]: opt = TrainOptions().parse(['--dataroot', '/data0/datasets/cyclegan/horse2zebra', '--nThreads', '8', '--no_dropout',
                                    '--niter', '100', '--niter_decay', '100', '--name', 'nodrop', '--gpu_ids', '2'])

        ------------ Options -------------
        batchSize: 1
        beta1: 0.5
        checkpoints_dir: ./checkpoints
        continue_train: False
        dataroot: /data0/datasets/cyclegan/horse2zebra
        dataset_mode: unaligned
        display_freq: 100
        display_id: 1
        display_port: 8097
        display_single_pane_ncols: 0
        display_winsize: 256
        epoch_count: 1
        fineSize: 256
        gpu_ids: [2]
        init_type: normal
        input_nc: 3
        isTrain: True
        lambda_A: 10.0

In [5]: from cgan.options.train_options import TrainOptions
        from cgan.data.data_loader import CreateDataLoader
        from cgan.models.models import create_model
```

参数输入

```
In [6]: data_loader = CreateDataLoader(opt)
        dataset = data_loader.load_data()
        dataset_size = len(data_loader)
        dataset_size

        CustomDatasetDataLoader
        dataset [UnalignedDataset] was created

Out[6]: 1334
```

数据导入

```
In [7]: model = create_model(opt)

        initialization method [normal]
        initialization method [normal]
        ---------- Networks initialized -------------
        ResnetGenerator(
          (model): Sequential(
            (0): ReflectionPad2d((3, 3, 3, 3))
            (1): Conv2d(3, 64, kernel_size=(7, 7), stride=(1, 1))
            (2): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=False)
            (3): ReLU(inplace)
            (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
            (5): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False)
            (6): ReLU(inplace)
            (7): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
            (8): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False)
            (9): ReLU(inplace)
            (10): ResnetBlock(
              (conv_block): Sequential(
                (0): ReflectionPad2d((1, 1, 1, 1))
                (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
```

建立模型

cycle-GAN

In [9]:

```python
total_steps = 0

for epoch in range(opt.epoch_count, opt.niter + opt.niter_decay + 1):
    epoch_start_time = time.time()
    iter_data_time = time.time()
    epoch_iter = 0

    for i, data in tqdm(enumerate(dataset)):
        iter_start_time = time.time()
        if total_steps % opt.print_freq == 0: t_data = iter_start_time - iter_data_time
        total_steps += opt.batchSize
        epoch_iter += opt.batchSize
        model.set_input(data)
        model.optimize_parameters()

        if total_steps % opt.display_freq == 0:
            save_result = total_steps % opt.update_html_freq == 0

        if total_steps % opt.print_freq == 0:
            errors = model.get_current_errors()
            t = (time.time() - iter_start_time) / opt.batchSize

        if total_steps % opt.save_latest_freq == 0:
            print('saving the latest model (epoch %d, total_steps %d)' % (epoch, total_steps))
            model.save('latest')

        iter_data_time = time.time()
    if epoch % opt.save_epoch_freq == 0:
        print('saving the model at the end of epoch %d, iters %d' % (epoch, total_steps))
        model.save('latest')
        model.save(epoch)

    print('End of epoch %d / %d \t Time Taken: %d sec' %
          (epoch, opt.niter + opt.niter_decay, time.time() - epoch_start_time))
    model.update_learning_rate()
```

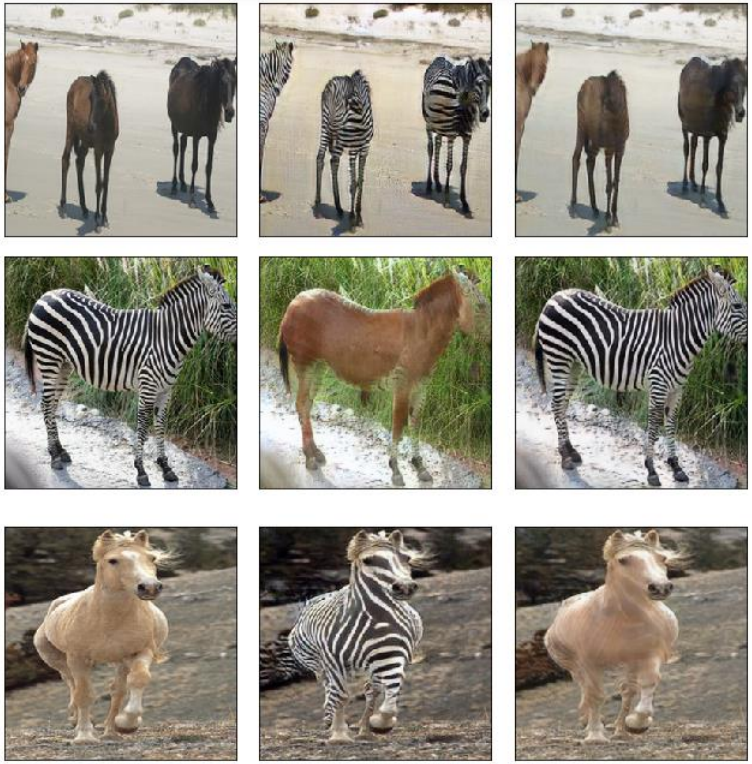
训练更新参数

```
saving the model at the end of epoch 200, iters 266800
End of epoch 200 / 200 	 Time Taken: 548 sec
learning rate = 0.0000000
```

cycle-GAN

- 学习结果

```
In [10]: def show_img(im, ax=None, figsize=None):
             if not ax: fig,ax = plt.subplots(figsize=figsize)
             ax.imshow(im)
             ax.get_xaxis().set_visible(False)
             ax.get_yaxis().set_visible(False)
             return ax

In [11]: def get_one(data):
             model.set_input(data)
             model.test()
             return list(model.get_current_visuals().values())

In [12]: model.save(201)

In [16]: test_ims = []
         for i,o in enumerate(dataset):
             if i>10: break
             test_ims.append(get_one(o))

In [17]: def show_grid(ims):
             fig,axes = plt.subplots(2,3,figsize=(9,6))
             for i,ax in enumerate(axes.flat): show_img(ims[i], ax);
             fig.tight_layout()

In [18]: for i in range(8): show_grid(test_ims[i])
```



谢谢！