Fastai Lesson 13

2018年8月

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

Stochastic weight averaging 随机加权平均

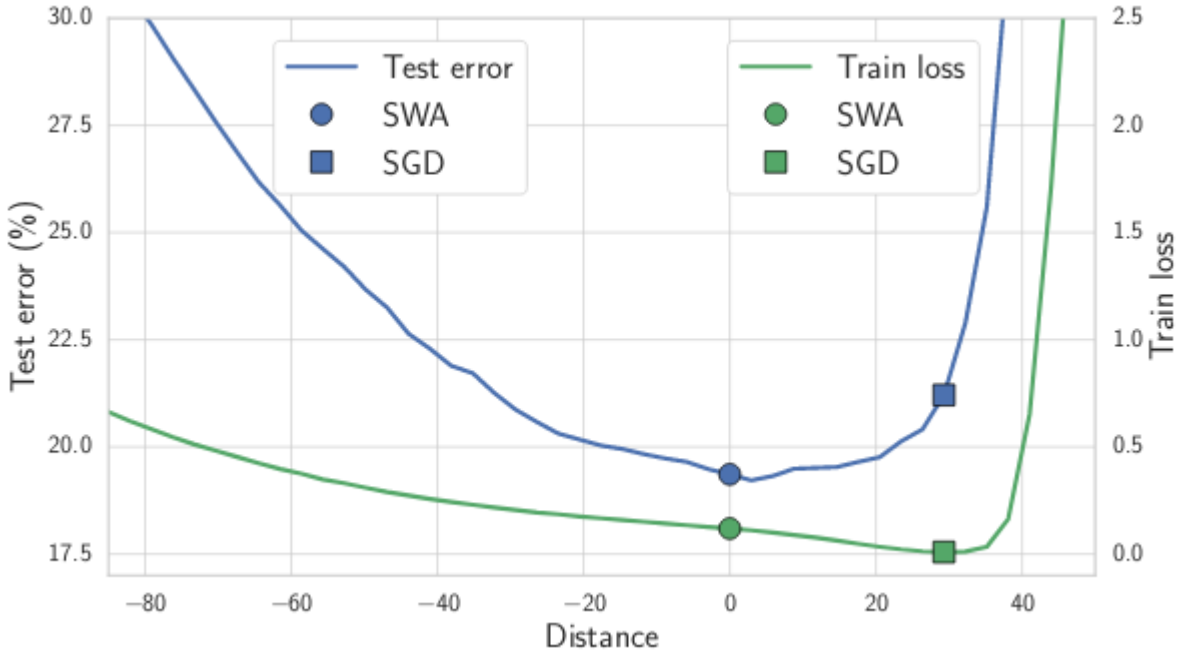
William Horton 贡献，只需在fit函数里加入use_swa，swa_start两个参数即可运行

Algorithm 1 Stochastic Weight Averaging

Require:
weights $\hat{w}$, LR bounds α_1, α_2,
cycle length c (for constant learning rate $c = 1$), number of iterations n

Ensure: w_{SWA}
$w \leftarrow \hat{w}$ {Initialize weights with $\hat{w}$}
$w_{\text{SWA}} \leftarrow w$
for $i \leftarrow 1, 2, \ldots, n$ **do**
  $\alpha \leftarrow \alpha(i)$ {Calculate LR for the iteration}
  $w \leftarrow w - \alpha \nabla \mathcal{L}_i(w)$ {Stochastic gradient update}
  if $\text{mod}(i, c) = 0$ **then**
    $n_{\text{models}} \leftarrow i/c$ {Number of models}
    $w_{\text{SWA}} \leftarrow \frac{w_{\text{SWA}} \cdot n_{\text{models}} + w}{n_{\text{models}} + 1}$ {Update average}
  end if
end for

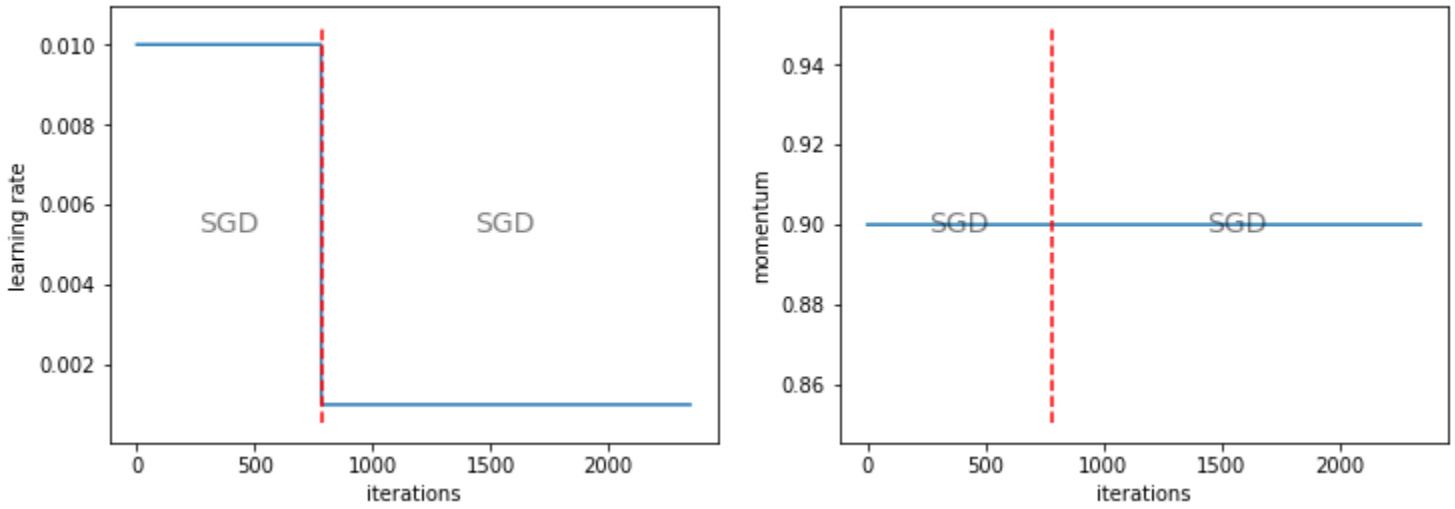
SWA的效果


Test error
SWA
SGD
Train loss
SWA
SGD
Test error (%)
Train loss
Distance
30.0
27.5
25.0
22.5
20.0
17.5
2.5
2.0
1.5
1.0
0.5
0.0
−80
−60
−40
−20
0
20
40

TrainPhase

```
phases = [TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-2), TrainingPhase(epochs=2, opt_fn=optim.SGD, lr = 1e-3)]
```

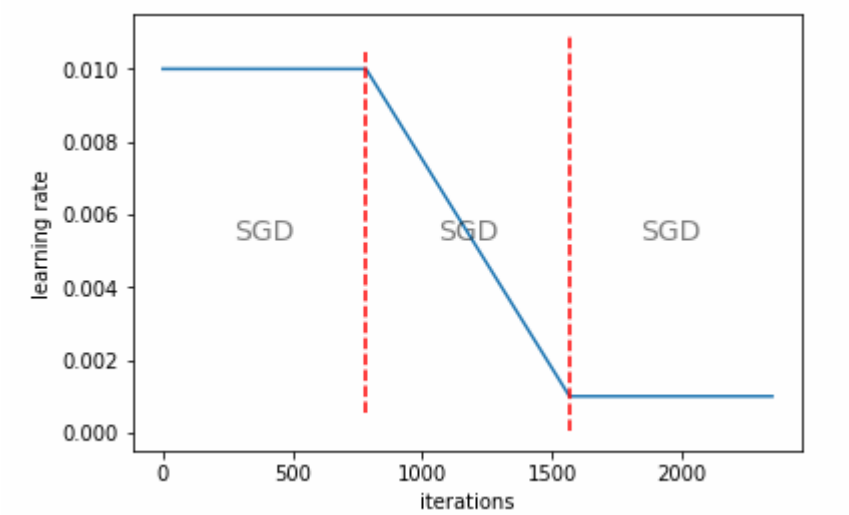
```
learn.fit_opt_sched(phases)
```



将各个阶段的学习率放入一个array中，在使用fit_opt_sched()函数进行训练

实现学习率的线性衰减

```
phases = [TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-2),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = (1e-2,1e-3), lr_decay=DecayType.LINEAR),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-3)]
```



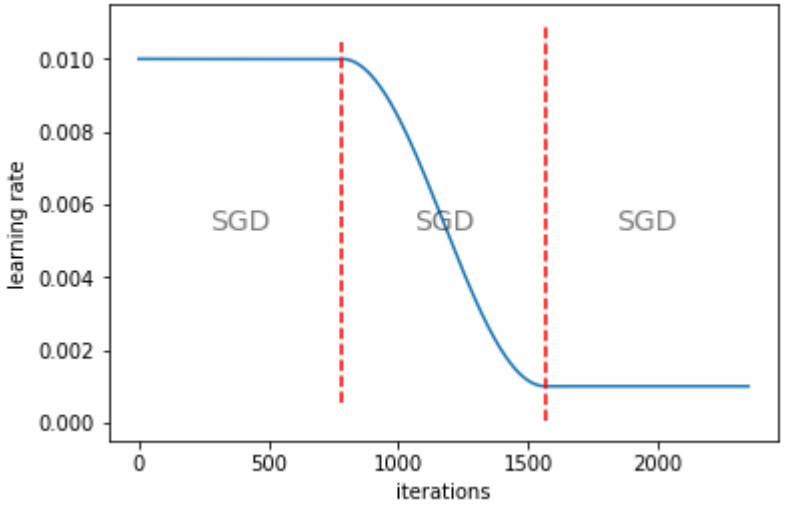
余弦衰减与指数衰减

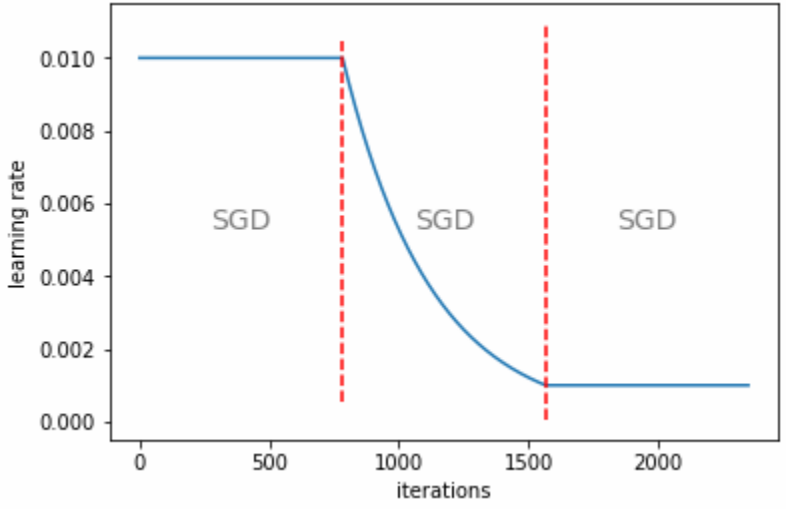
```
phases = [TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-2),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = (1e-2,1e-3), lr_decay=DecayType.COSINE),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-3)]
```

$$lr_i = l + \frac{l+h}{2}(1 + \frac{cos\pi i}{n})$$

$$lr_i = l + \left(\frac{h}{l}\right)^{\frac{i}{n}}$$

```
phases = [TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-2),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = (1e-2,1e-3), lr_decay=DecayType.EXPONENTIAL),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-3)]
```



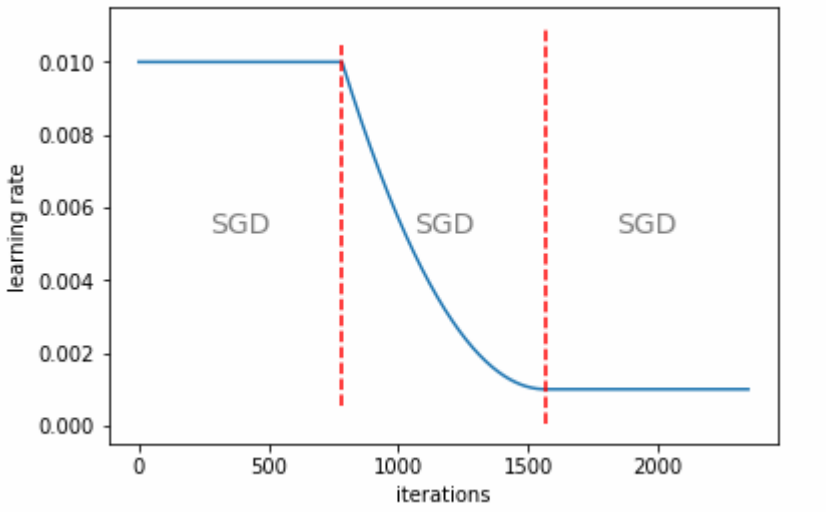


多项式衰减

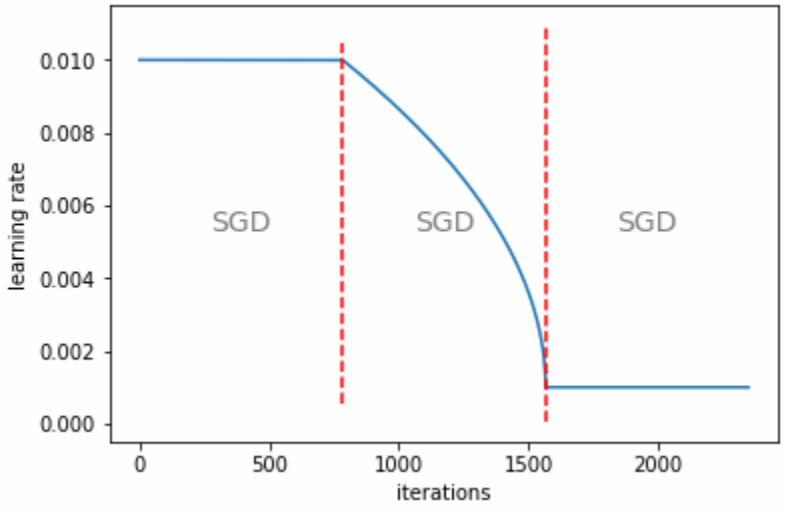
```
phases = [TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-2),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = (1e-2,1e-3), lr_decay=(DecayType.POLYNOMIAL, 2)),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr = 1e-3)]
```

$$lr_i = l + (h - l) \cdot \left(1 - \frac{i}{n}\right)^p$$

$p = 2$



$p = 0.5$



SGDR的实现

```
def phases_sgdr(lr, opt_fn, num_cycle,cycle_len,cycle_mult):
    phases = [TrainingPhase(epochs = cycle_len/ 20, opt_fn=opt_fn, lr=lr/100),
              TrainingPhase(epochs = cycle_len * 19/20, opt_fn=opt_fn, lr=lr, lr_decay=DecayType.COSINE)]
    for i in range(1,num_cycle):
        phases.append(TrainingPhase(epochs = cycle_len * (cycle_mult**i), opt_fn=opt_fn, lr=lr, lr_decay=DecayType.COSINE))
    return phases
```

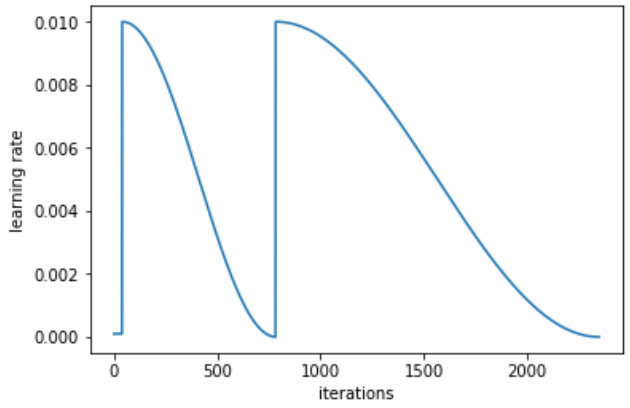
```
learn.fit_opt_sched(phases_sgdr(1e-2, optim.Adam, 2, 1, 2))
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

epoch	trn_loss	val_loss	accuracy
0	3.576923	4.471715	0.1359
1	3.773803	4.577302	0.1383
2	3.521146	4.507113	0.1402

```
[array([4.50711]), 0.1402]
```

```
learn.sched.plot_lr(show_text=False,show_moms=False)
```



学习率与优化方法都可自定义

```
phases = [TrainingPhase(epochs=1, opt_fn=optim.SGD, lr=(1e-3,1e-2), lr_decay=DecayType.LINEAR,
                        momentum=(0.95,0.85), momentum_decay=DecayType.LINEAR),
          TrainingPhase(epochs=1, opt_fn=optim.SGD, lr=(1e-2,1e-3), lr_decay=DecayType.LINEAR,
                        momentum=(0.85,0.95), momentum_decay=DecayType.LINEAR),
          TrainingPhase(epochs=1, opt_fn=optim.Adam, lr=1e-3, lr_decay=DecayType.COSINE, momentum=0.9)]
```

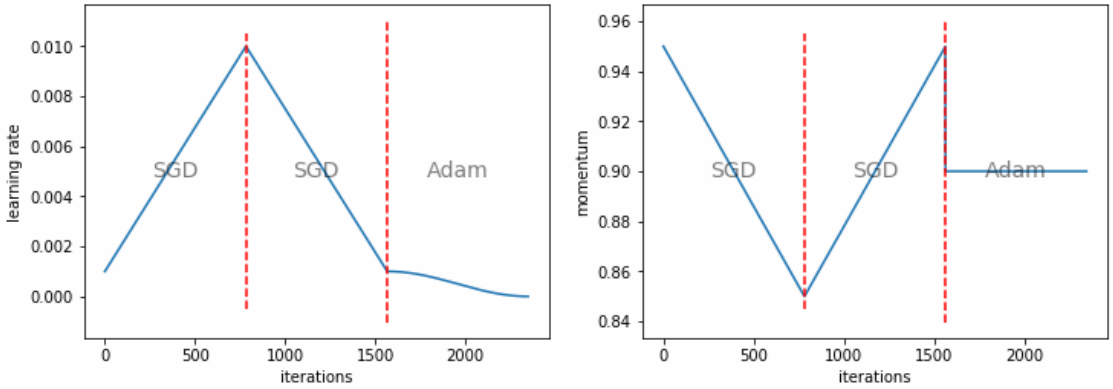
```
learn.fit_opt_sched(phases)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

epoch	trn_loss	val_loss	accuracy
0	3.59037	4.5196	0.14
1	3.607808	4.601357	0.133
2	3.543177	4.520047	0.1417

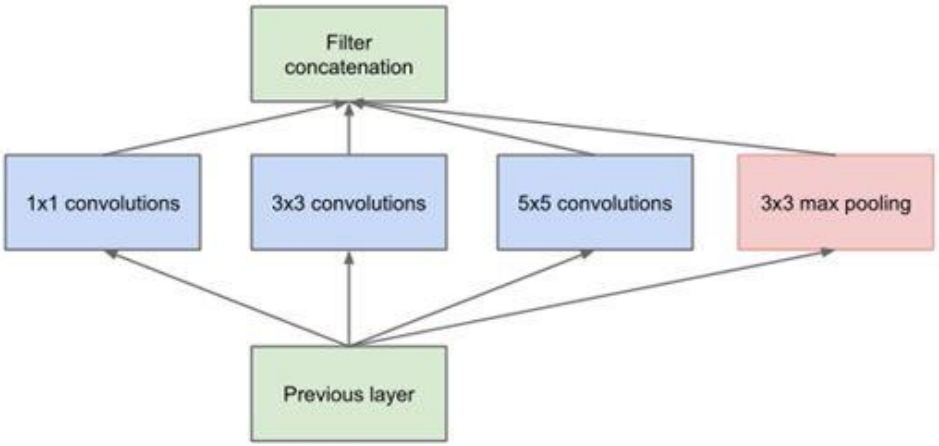
```
[array([4.52005]), 0.1417]
```

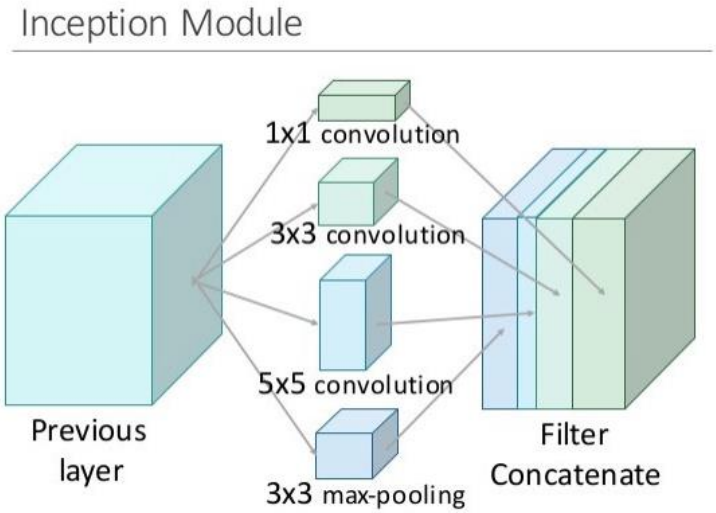
```
learn.sched.plot_lr()
```



Inception network

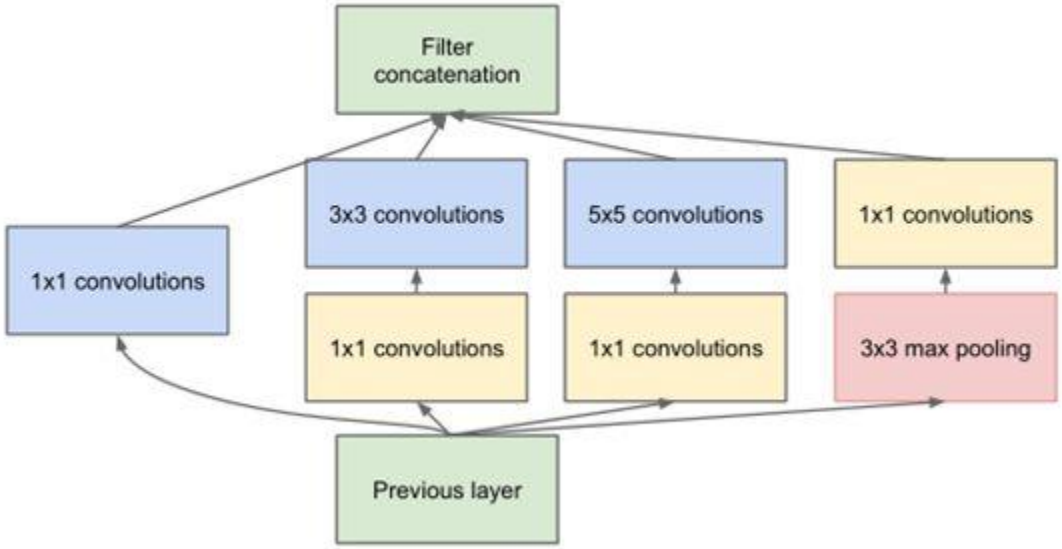
参考https://zhuanlan.zhihu.com/p/30756181





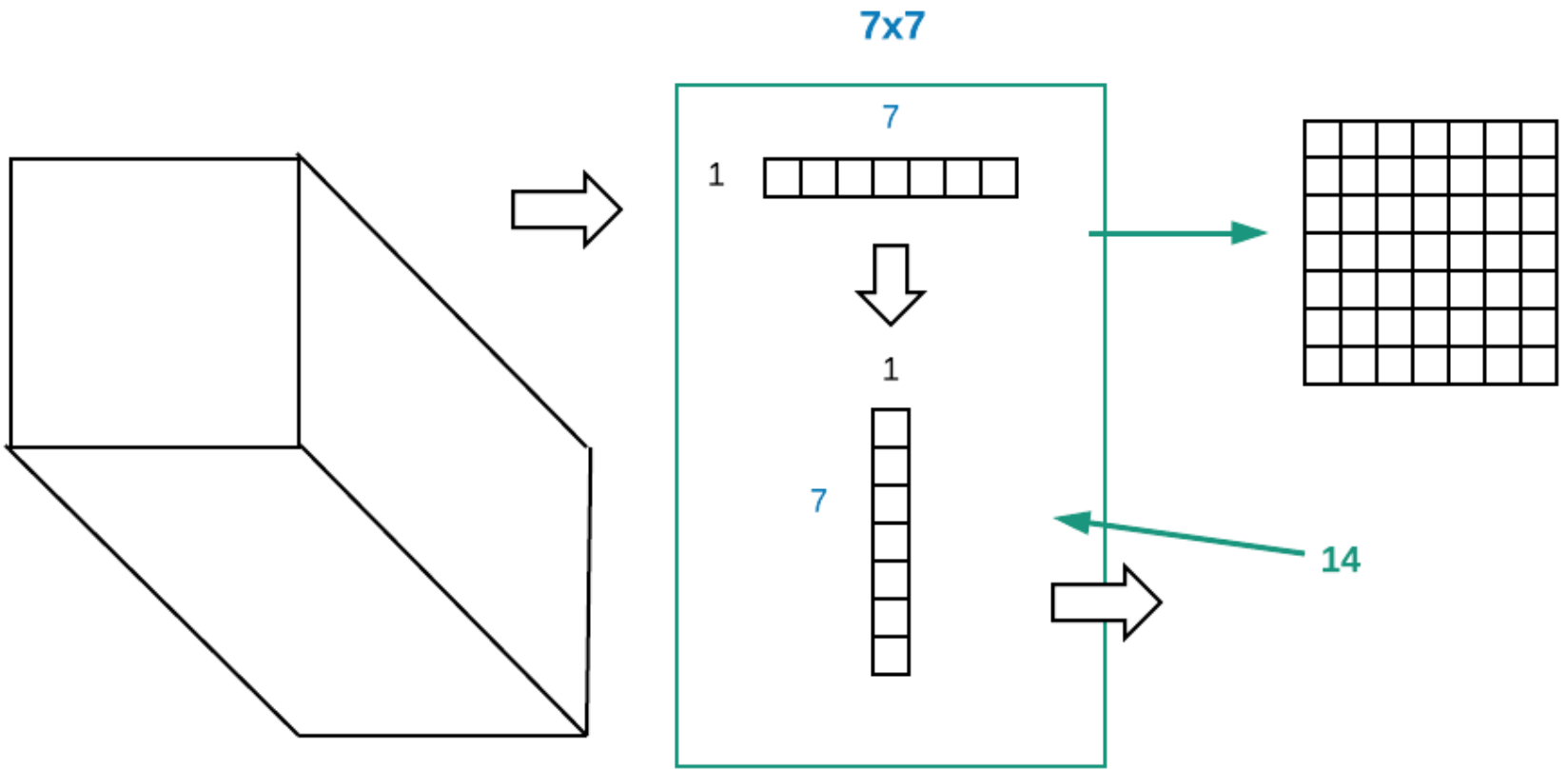
1. 一层block就包含1x1卷积，3x3卷积，5x5卷积，3x3池化(使用这样的尺寸不是必需的，可以根据实际需要进行调整)。这样，网络中每一层都能学习到“稀疏”（3x3、5x5）或“不稀疏”（1x1）的特征，既增加了网络的宽度，也增加了网络对尺度的适应性；
2. 通过deep concat在每个block后合成特征，获得非线性属性。

Inception network V1



加入了1*1卷积层，其作用是将多通道的信息进行整合，同时也对卷积核通道数进行降维，使得网络的参数减少。

7*7=1*7 * 7*1



利用两个1*n与n*1卷积层替代一个n*n卷积层，大大减少了计算成本，同时效果也不会大幅下降

Inception-ResNet-v2

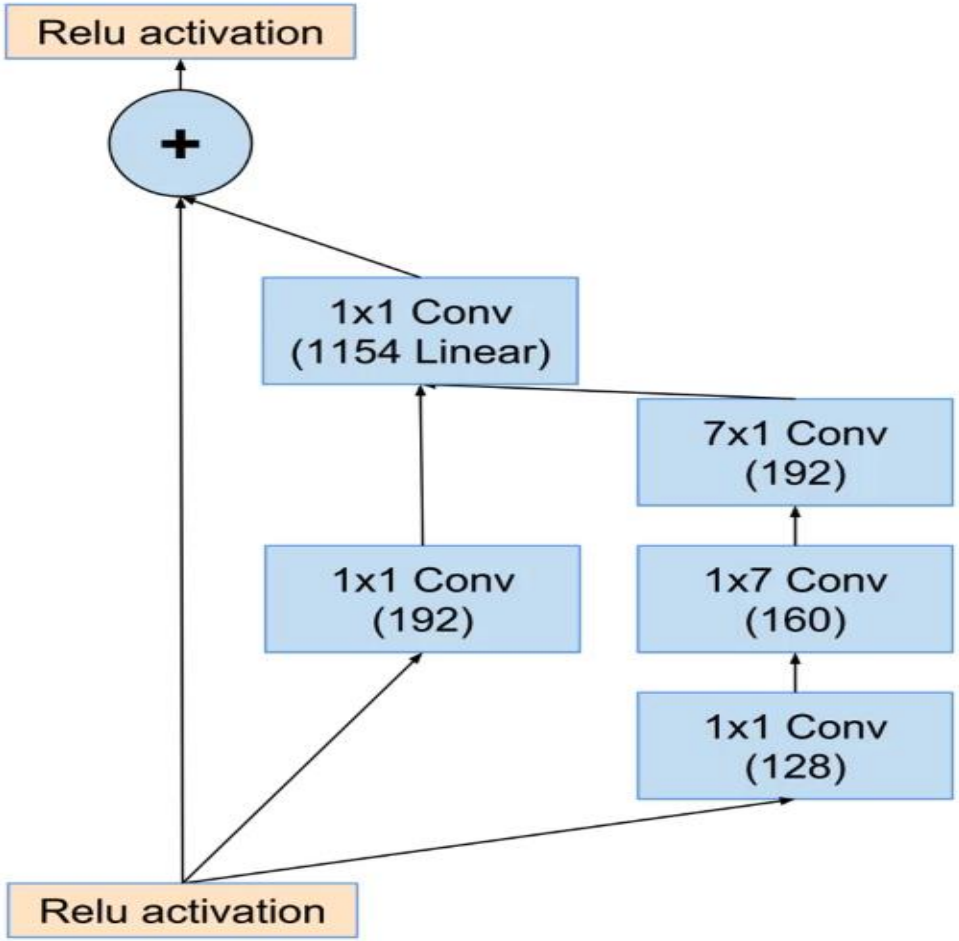


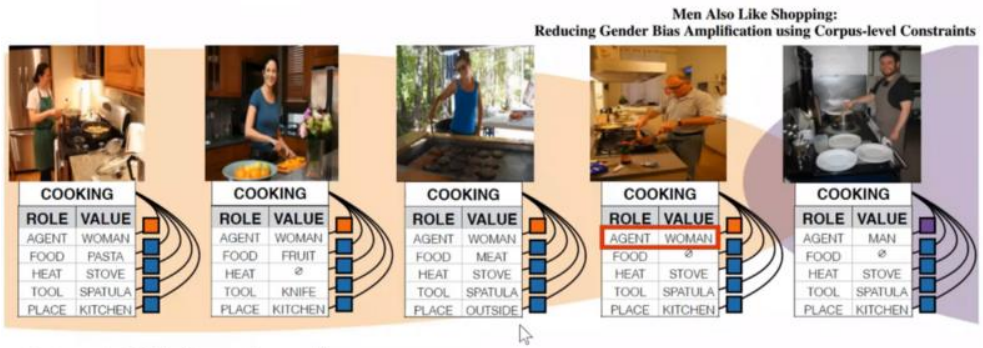
Figure 17. The schema for 17 × 17 grid (Inception-ResNet-B) module of the Inception-ResNet-v2 network.

ResNet的结构既可以加速训练，还可以提升性能（防止梯度弥散），Inception模块可以在同一层上获得稀疏或非稀疏的特征。将两者结合起来，使得模型对图像识别的错误率进一步得到了降低。

AI的伦理

AI程序的运行会带入偏见，并且将此类偏见放大

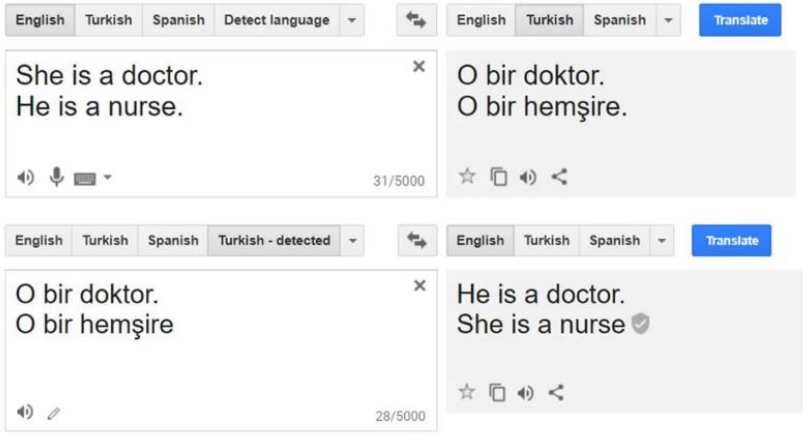
Machine Learning can amplify bias.



Data set: 67% of people cooking are women

Algorithm predicts: 84% of people cooking are women

Language



AI的伦理

技术本身没有错，但是不能用技术来作恶。

In the concentration camps, IBM's code for Jews was 8. Its code for Gypsies was 12. General executions were coded as 4, death in the gas chambers as 6. Only Jews and Gypsies were systematically murdered in gas chambers.

The Swiss judge ruled "It does not thus see unreasonable to deduce that IBM's technic facilitated the tasks of the Nazis in the com their crimes against humanity, acts also involving accountancy and classification by IBM machines and utilized in the concentration camps themselves."

"To the blind technocrat, the means were more important than the ends. The destruction of the Jewish people became even less important because the invigorating nature of IBM's technical achievement was only heightened by the fantastical profits to be made at a time when bread lines stretched across the world."

Style transfer图像的风格转换

参考论文：https://arxiv.org/pdf/1508.06576.pdf

将现有的一张图片，保持内容不变，转换为特定的（艺术）风格。

ImageNet

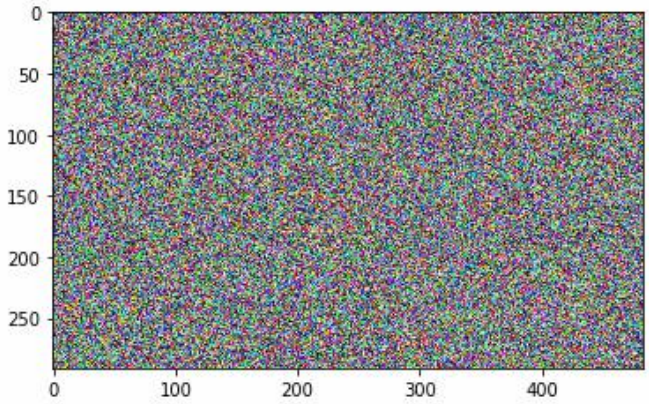
下载地址http://files.fast.ai/data/imagenet-sample-train.tar.gz

约2.1G的部分图集，也可在https://www.kaggle.com/c/imagenet-object-localization-challenge/data下载全部数据

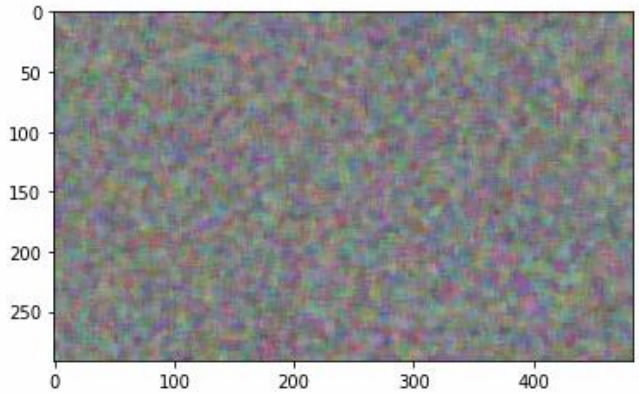
选取一张鸟的照片（如上图）进行尝试。

建立随机图

```
opt_img = np.random.uniform(0, 1, size=img.shape).astype(np.float32)
plt.imshow(opt_img);
```



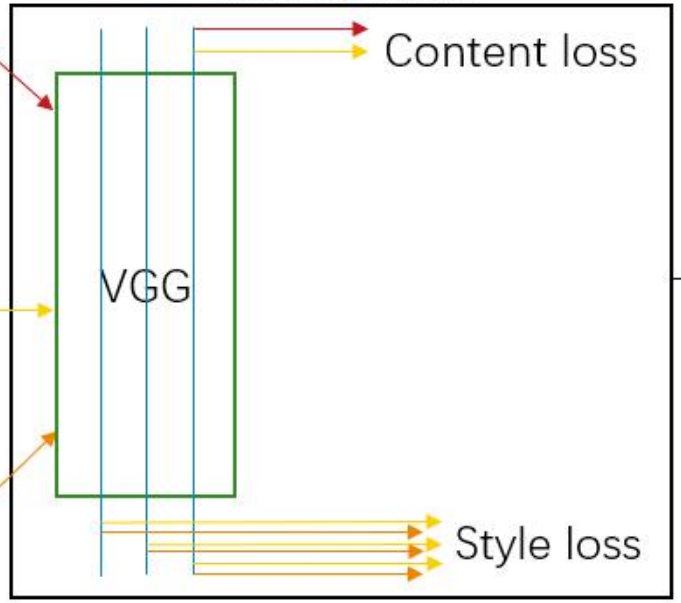

```
opt_img = scipy.ndimage.filters.median_filter(opt_img, [8,8,1])
plt.imshow(opt_img);
```



我们利用（0,1）之间的随机数建立一个图像，但是随机建立的图像并不具有像一般图像一样的平滑性，在训练过程中会效果很差。因此，我们利用一个中值滤波函数将初始生成的随机图形变得模糊，也使得两个像素之间有一定的关联渐变，正如自然拍摄的图像那样。

风格转换问题的loss

$$TotalLoss = ContentLoss + \lambda \cdot StyleLoss$$



CL越小，新图就和原图越相似，而SL越小，则新图的风格就越像风格图。

Content Loss

我们不能用每个像素点与原图像素点的均方差作为内容误差的值——这只会使得我们得出来的图像和之前的图像整体一样，而不是形状类似风格不同。
总体来说，我们只需要把握这个图像的特征，鸟的羽毛的层次，鸟的眼镜的形状，鸟喙的形状等等。我们希望鸟的色彩风格和原图不一样，但是我们可以看出来它就是我们原图中的鸟。

在这个任务中，我们将原图像输入VGG网络，并且截取通过某几个卷积层之后的输出作为Ground Truth，并且将生成图也通过同样的网络，在同样的层截取输出，计算其与Ground Truth的均方误差，以此来构成Content Loss，我们也将其称为perceptual loss（感知误差）。

```
m_vgg = to_gpu(vgg16(True)).eval()
set_trainable(m_vgg, False)
```

图像经过VGG网络

```
sz=288
```

```
trn_tfms,val_tfms = tfms_from_model(vgg16, sz)
img_tfm = val_tfms(img)
img_tfm.shape
```

```
(3, 288, 288)
```

```
opt_img = val_tfms(opt_img)/2
opt_img_v = V(opt_img[None], requires_grad=True)
opt_img_v.shape
```

```
torch.Size([1, 3, 288, 288])
```

将两个图像经过统一尺寸等操作转化成shape为3*288*288的图片，此时img_tfm为转化之后用于输入网络的原图，opt_img_v为需要优化的新图。

Content Loss

```
m_vgg = nn.Sequential(*children(m_vgg)[:37])
```

选取差不多在VGG网络中间偏后的前37层网络作为一个新的网络。

```
targ_t = m_vgg(VV(img_tfm[None]))
targ_v = V(targ_t)
targ_t.shape
```

```
torch.Size([1, 512, 18, 18])
```

```
targ_v.shape
```

```
torch.Size([1, 512, 18, 18])
```

将原图输入该网络之后，得到的一个值可以理解为一个“像原来图中的鸟”的图。要将随机图输入后与其进行对比，其误差采用均方差。

```
def actn_loss(x): return F.mse_loss(m_vgg(x), targ_v)*1000
```

训练过程

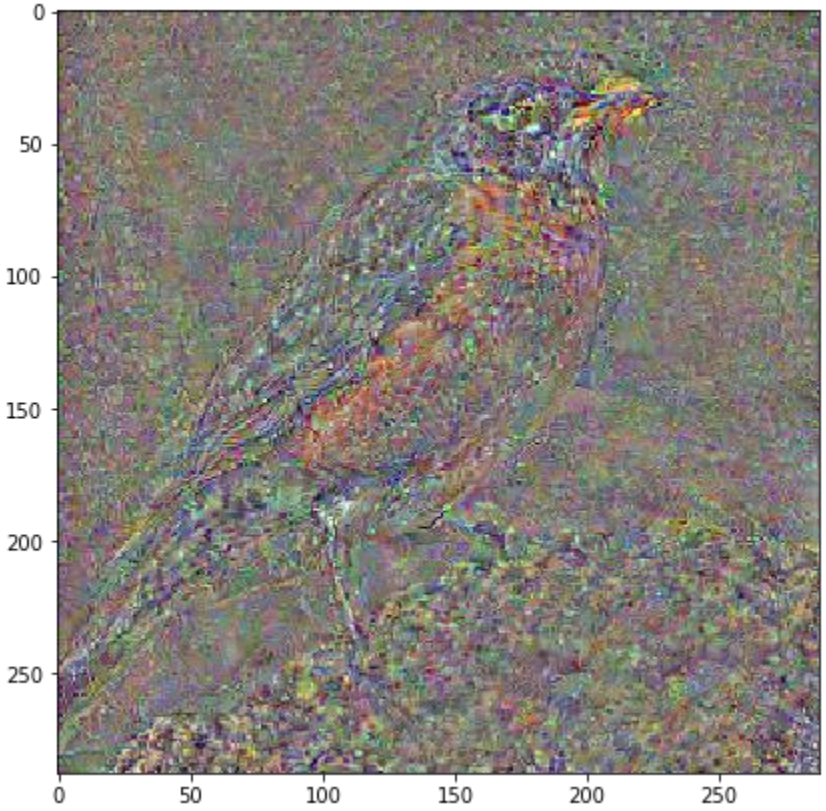
```
max_iter = 1000
show_iter = 100
optimizer = optim.LBFGS([opt_img_v], lr=0.5)
```

```
def step(loss_fn):
    global n_iter
    optimizer.zero_grad()
    loss = loss_fn(opt_img_v)
    loss.backward()
    n_iter+=1
    if n_iter%show_iter==0: print(f'Iteration: {n_iter}, loss: {loss.data[0]}')
    return loss
```

```
n_iter=0
while n_iter <= max_iter: optimizer.step(partial(step,actn_loss))
```

```
Iteration: 100, loss: 0.7847330570220947
Iteration: 200, loss: 0.332084059715271
Iteration: 300, loss: 0.21638627350330353
Iteration: 400, loss: 0.16194190084934235
Iteration: 500, loss: 0.13296256959438324
Iteration: 600, loss: 0.11422532796859741
Iteration: 700, loss: 0.10144167393445969
Iteration: 800, loss: 0.09302172064781189
Iteration: 900, loss: 0.0865076407790184
Iteration: 1000, loss: 0.08136843144893646
```

训练结果



optim.LBFGS

```
max_iter = 1000
show_iter = 100
optimizer = optim.LBFGS([opt_img_v], lr=0.5)
```

Broyden-Fletcher-Goldfarb- Shanno（BFGS）with Limited memory

在梯度下降过程中，如果损失函数二阶可微，则可用牛顿法实现更加精确的梯度下降，但这会需要运算Hessian矩阵，导致计算量巨大。BFGS就是一种拟牛顿法，是比较折中的适宜的选择，它可计算下降的梯度。

BFGSMultiply$(\mathbf{H}_0^{-1}, \{s_k\}, \{y_k\}, d)$:

$r \leftarrow d$

// Compute right product

for $i = n, \ldots, 1$:

$\alpha_i \leftarrow \rho_i s_i^T r$

$r \leftarrow r - \alpha_i y_i$

// Compute center

$r \leftarrow \mathbf{H}_0^{-1} r$

// Compute left product

for $i = 1, \ldots, n$:

$\beta \leftarrow \rho_i y_i^T r$

$r \leftarrow r + (\alpha_{n-i+1} - \beta) s_i$

return r

而在实际运算过程中，大量的值储存会占用大量的内存，因此我们用limited memory，保留之前10-20个梯度进行之后的运算。

前钩

forward hook

```
class SaveFeatures():
    features=None
    def __init__(self, m): self.hook = m.register_forward_hook(self.hook_fn)
    def hook_fn(self, module, input, output): self.features = output
    def close(self): self.hook.remove()
```

在此是用来在数据前向传播到某几层的时候将输出储存起来，用来进行后续计算

```
def __call__(self, *input, **kwargs):
    result = self.forward(*input, **kwargs)
    for hook in self._forward_hooks.values():
        #将注册的hook拿出来用
        hook_result = hook(self, input, result)
    ...
    return result
```

前钩的作用

```
block_ends = [i-1 for i,o in enumerate(children(m_vgg))
              if isinstance(o,nn.MaxPool2d)]
block_ends
```

任意选取一些层，此处是以MaxPool作为筛选

```
[5, 12, 22, 32, 42]
```

```
sf = SaveFeatures(children(m_vgg)[block_ends[3]])
```

之前内容的封装

```
def get_opt():
    opt_img = np.random.uniform(0, 1, size=img.shape).astype(np.float32)
    opt_img = scipy.ndimage.filters.median_filter(opt_img, [8,8,1])
    opt_img_v = V(val_tfms(opt_img/2)[None], requires_grad=True)
    return opt_img_v, optim.LBFGS([opt_img_v])
```

```
opt_img_v, optimizer = get_opt()
```

实际获取的是第32层的值

```
m_vgg(VV(img_tfm[None]))
targ_v = V(sf.features.clone())
targ_v.shape
```

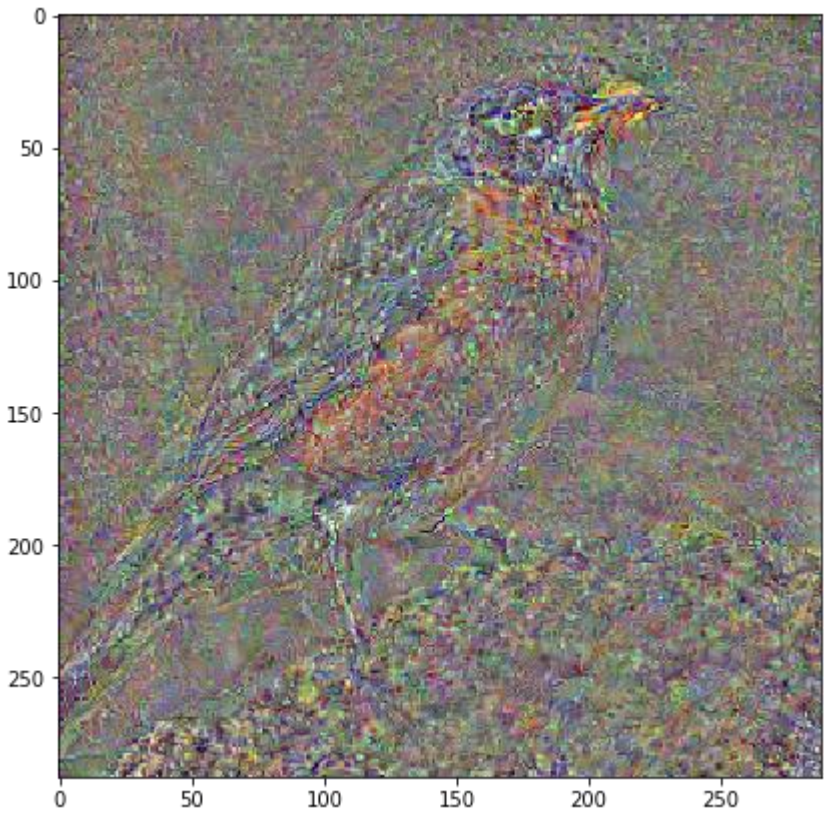
```
torch.Size([1, 512, 36, 36])
```

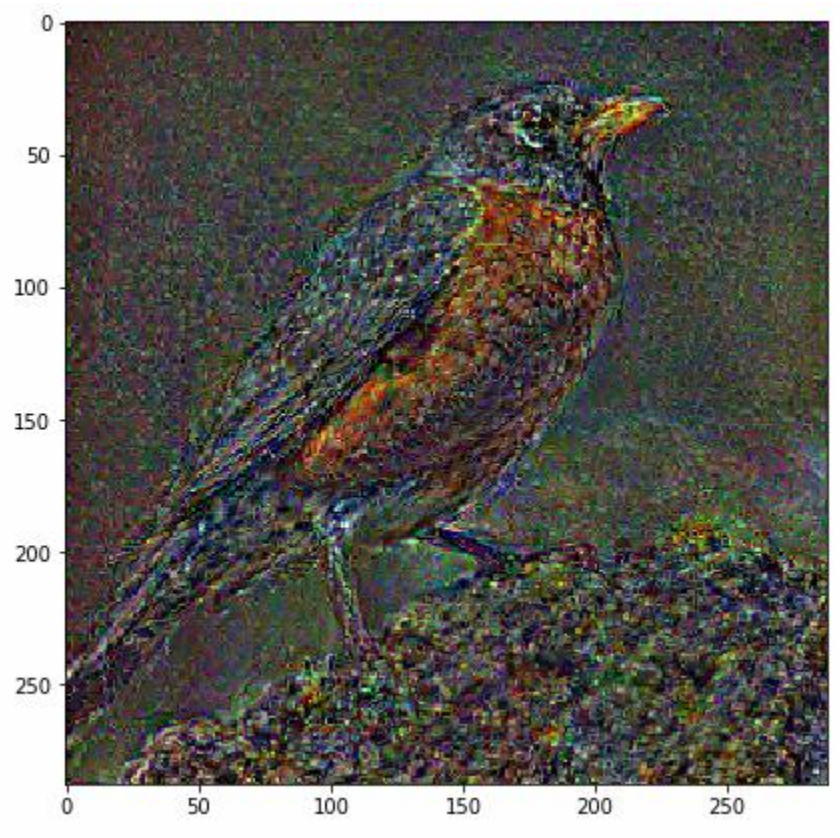
计算误差的时候用的也是第32层的值

```
def actn_loss2(x):
    m_vgg(x)
    out = V(sf.features)
    return F.mse_loss(out, targ_v)*1000
```

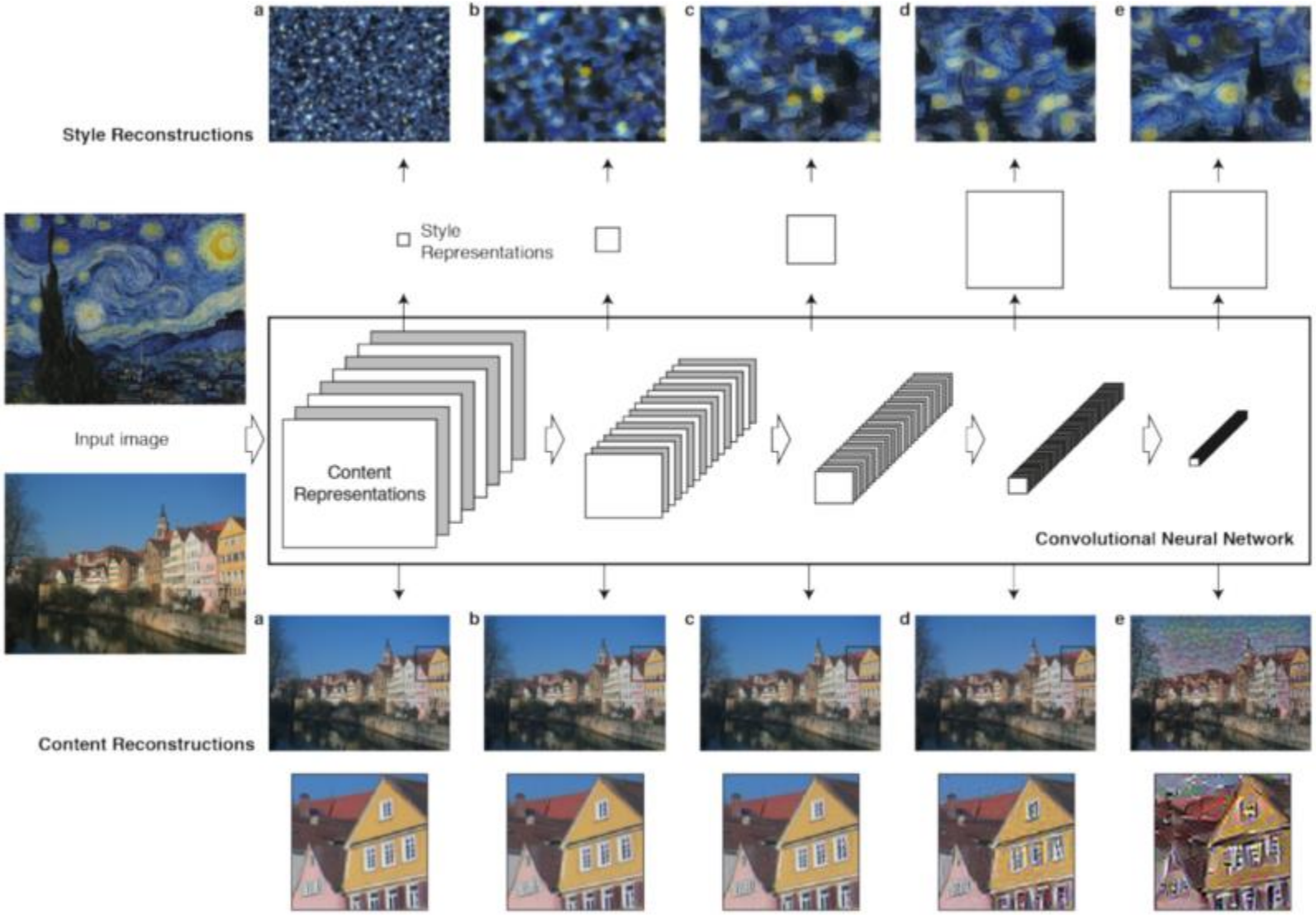
这样，也就不用再次重新生成一个m_vgg模型，要进行改动选取的层也就可以直接从前钩进行改动。

37层VS 32层





越前面的层，越像是有原图鸟的特征。


a
b
c
d
e
Style Reconstructions
Style Representations
Input image
Content Representations
Convolutional Neural Network
a
b
c
d
e
Content Reconstructions

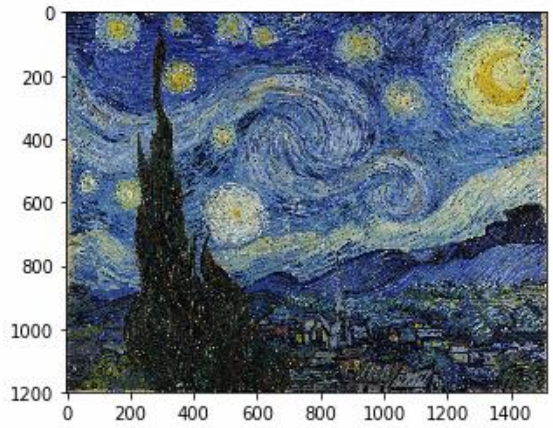
Style Loss

```
style_fn = PATH/'style'/'starry_night.jpg'
```

```
style_img = open_image(style_fn)
style_img.shape, img.shape
```

```
((1198, 1513, 3), (291, 483, 3))
```

```
plt.imshow(style_img);
```




```
def scale_match(src, targ):
    h,w,_ = src.shape
    sh,sw,_ = targ.shape
    rat = max(h/sh,w/sw); rat
    res = cv2.resize(targ, (int(sw*rat), int(sh*rat)))
    return res[:h,:w]
```

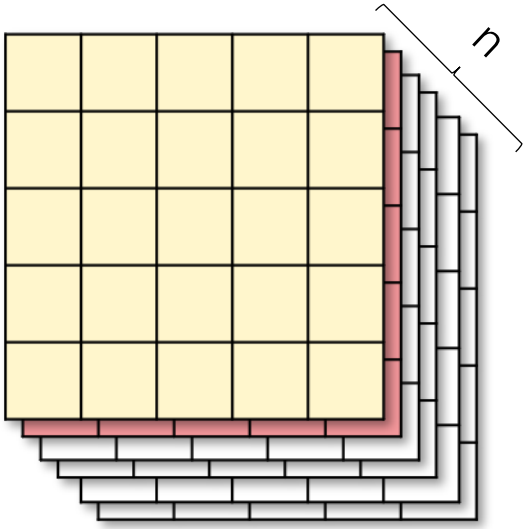
```
style = scale_match(img, style_img)
```

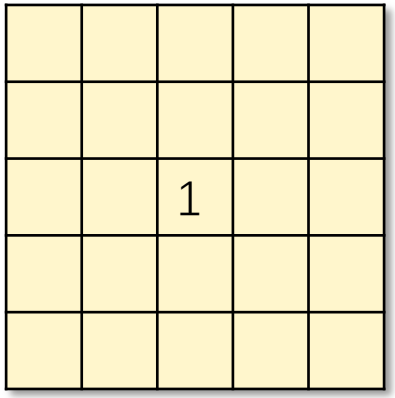
```
plt.imshow(style)
style.shape, img.shape
```

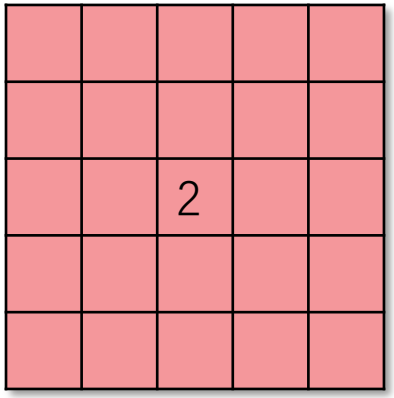
```
((291, 483, 3), (291, 483, 3))
```

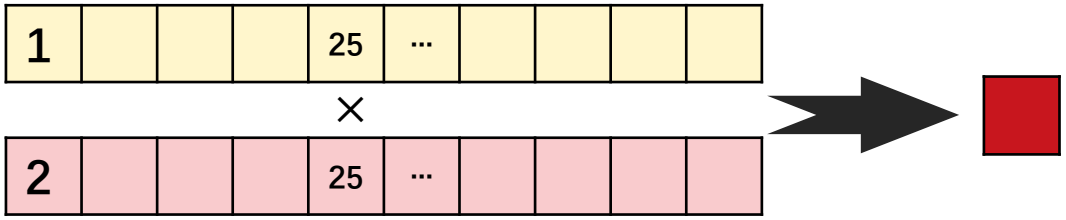
选取梵高的画作为目标风格，并将其大小转变为与原图相同。

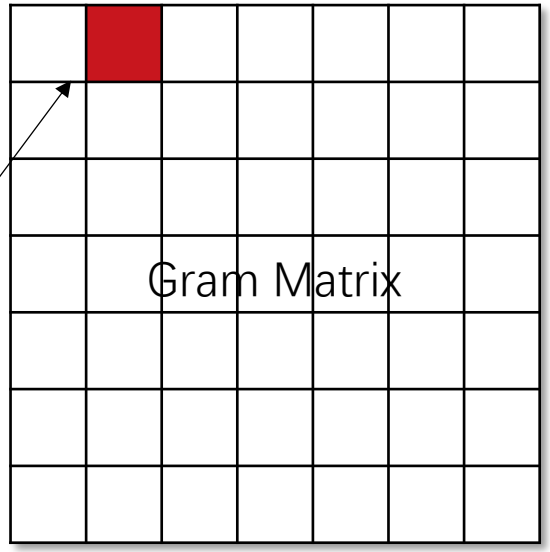
Style Loss



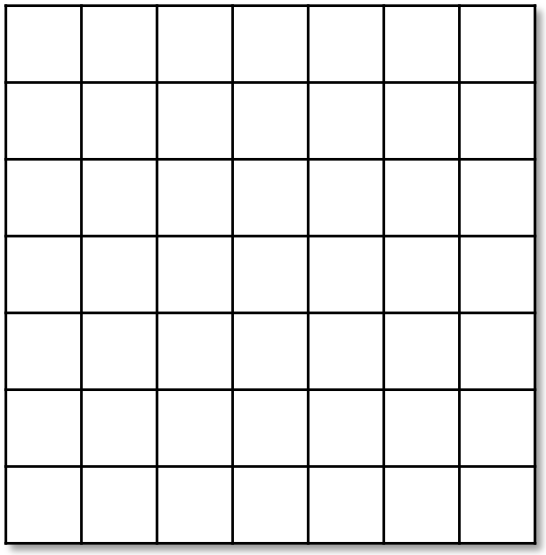








Gram Matrix



第n行第m列的值表示第m层和第n层的关系，其数值越大，则说明两者关系越强，而第n行第n列数值越大，则说明这一层的特征最明显。

Gram Matrix体现的是风格的特征。

Style Loss实现

```
def gram(input):
        b,c,h,w = input.size()
        x = input.view(b*c, -1)
        return torch.mm(x, x.t())/input.numel()*1e6

def gram_mse_loss(input, target): return F.mse_loss(gram(input), gram(target))
```

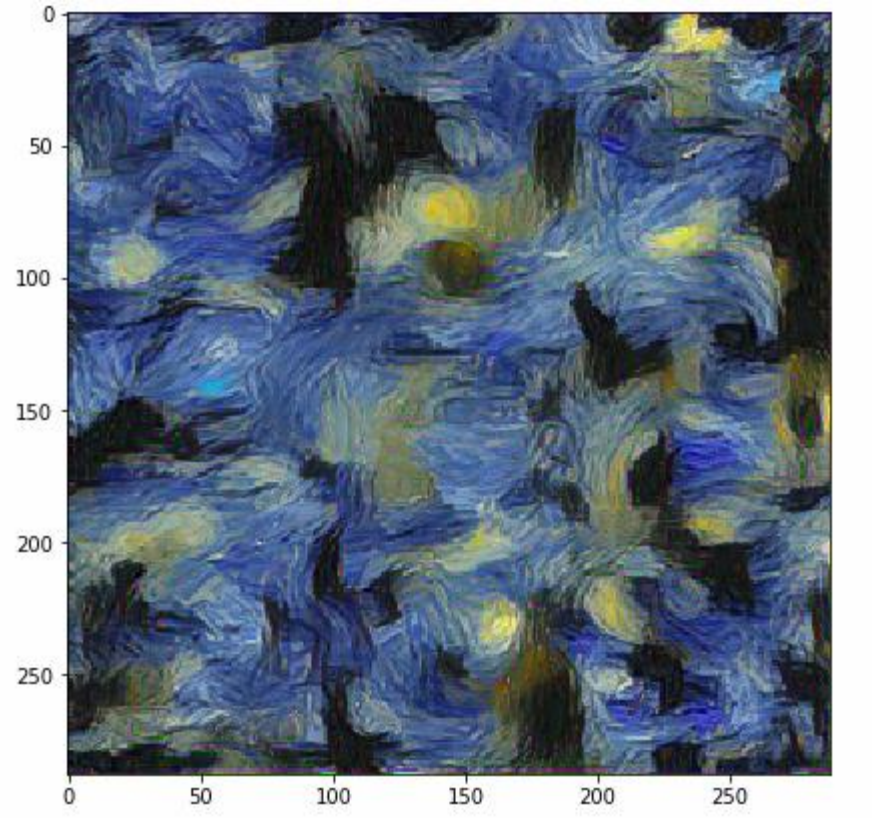
```
def style_loss(x):
    m_vgg(opt_img_v)
    outs = [V(o.features) for o in sfs]
    losses = [gram_mse_loss(o, s) for o,s in zip(outs, targ_styles)]
    return sum(losses)
```

```
n_iter=0
while n_iter <= max_iter: optimizer.step(partial(step,style_loss))
```

```
Iteration: n_iter, loss: 230718.453125
Iteration: n_iter, loss: 219493.21875
Iteration: n_iter, loss: 202618.109375
Iteration: n_iter, loss: 481.5616760253906
Iteration: n_iter, loss: 147.41177368164062
Iteration: n_iter, loss: 80.62625122070312
Iteration: n_iter, loss: 49.52326965332031
Iteration: n_iter, loss: 32.36254119873047
Iteration: n_iter, loss: 21.831811904907227
Iteration: n_iter, loss: 15.61091423034668
```

```
x = val_tfms.denorm(np.rollaxis(to_np(opt_img_v.data),1,4))[0]
plt.figure(figsize=(7,7))
plt.imshow(x);
```

结果



风格转换—Content Loss+Style Loss

Style transfer

```
opt_img_v, optimizer = get_opt()
```

```
def comb_loss(x):
    m_vgg(opt_img_v)
    outs = [V(o.features) for o in sfs]
    losses = [gram_mse_loss(o, s) for o,s in zip(outs, targ_styles)]
    cnt_loss   = F.mse_loss(outs[3], targ_vs[3])*1000000
    style_loss = sum(losses)
    return cnt_loss + style_loss
```

```
n_iter=0
while n_iter <= max_iter: optimizer.step(partial(step,comb_loss))
```

```
Iteration: n_iter, loss: 1802.36767578125
Iteration: n_iter, loss: 1163.05908203125
Iteration: n_iter, loss: 961.6024169921875
Iteration: n_iter, loss: 853.079833984375
Iteration: n_iter, loss: 784.970458984375
Iteration: n_iter, loss: 739.18994140625
Iteration: n_iter, loss: 706.310791015625
Iteration: n_iter, loss: 681.6689453125
Iteration: n_iter, loss: 662.4088134765625
Iteration: n_iter, loss: 646.329833984375
```

```
x = val_tfms.denorm(np.rollaxis(to_np(opt_img_v.data),1,4))[0]
plt.figure(figsize=(9,9))
plt.imshow(x, interpolation='lanczos')
plt.axis('off');
```

结果

谢 谢！