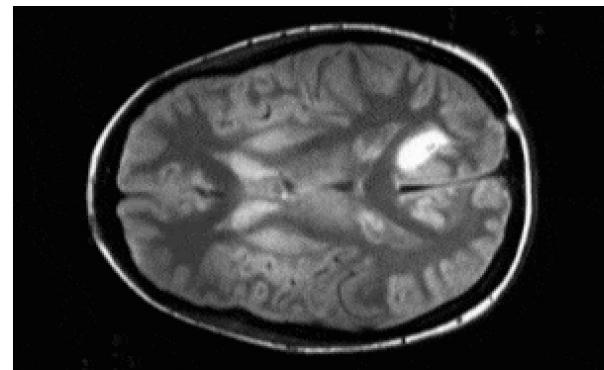
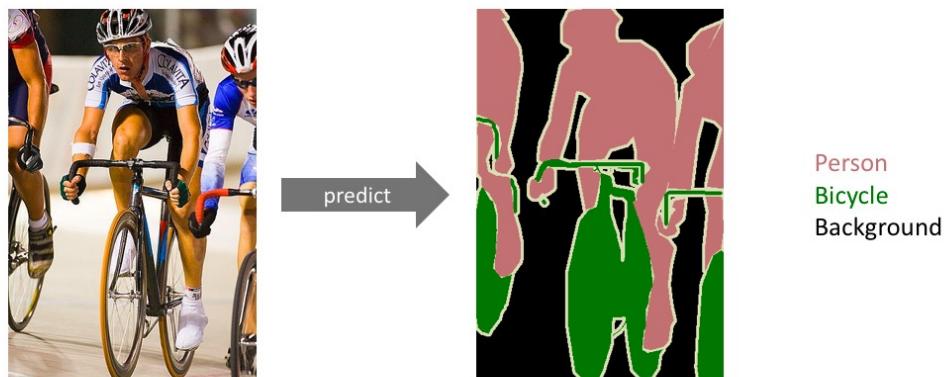


图像分割, 合成, 风格迁移

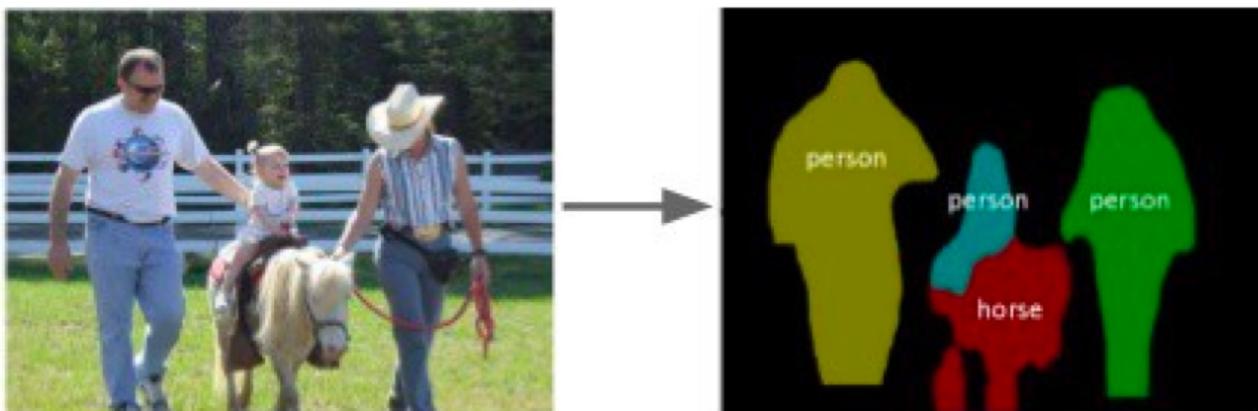
图像分割之语义分割(semantic segmentation)

语义图像分割的目标是标记图像每个像素的类别。因为我们需要预测图像中的每个像素，所以此任务通常被称为密集预测。



图像分割之实例分割(instance segmentation)

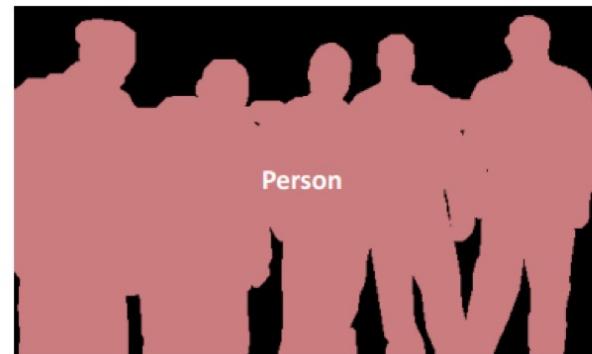
实例分割比语义分割更进一步，除了像素级分类，还需要分别对类的每个实例进行分类。例如，在图中有3个人，技术上是“人”类的3个实例。所有3个都被分类（以不同的颜色）。但是语义分割不区分特定类的实例。



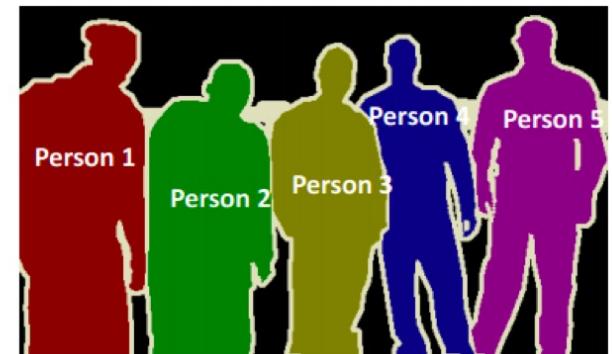
目标检测, 语义分割, 实例分割比较



Object Detection



Semantic Segmentation



Instance Segmentation

语义分割的应用

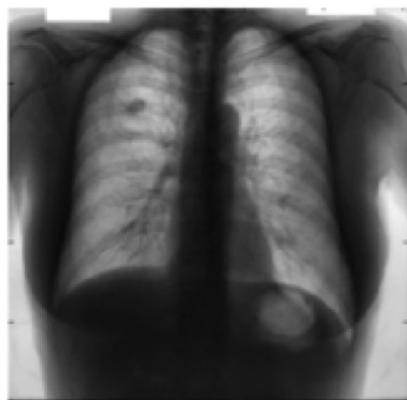
- 1. 自动驾驶



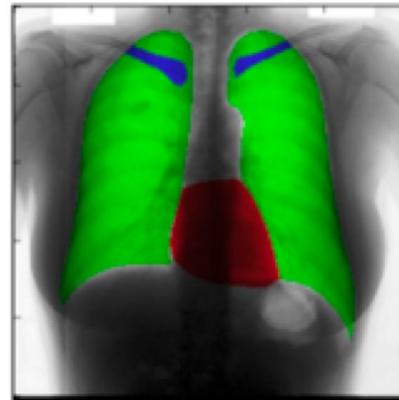
自动驾驶是一项复杂的任务，需要在不断变化的环境中进行感知，规划和执行。由于安全性至关重要，因此还需要以最高精度执行此任务。语义分割提供有关道路上空间的信息，以及检测车道标记和交通标志。

语义分割的应用

- 2. 医学图像处理



Input Image



Segmented Image

可以辅助放射科医师进行图像分析，大大减少了诊断所需的时间。

语义分割的应用

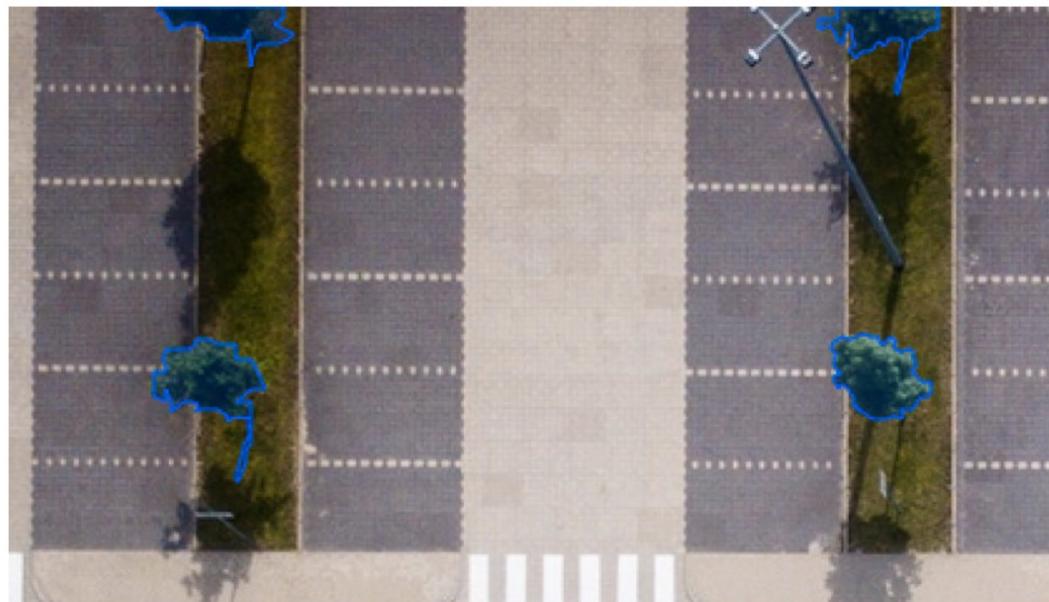
- 3. 地理感应 (开发数据集 SpaceNet)



卫星图像用于检测土地覆盖信息，对于应用：例如监测毁林的区域。

语义分割的应用

- 4. 精准农业

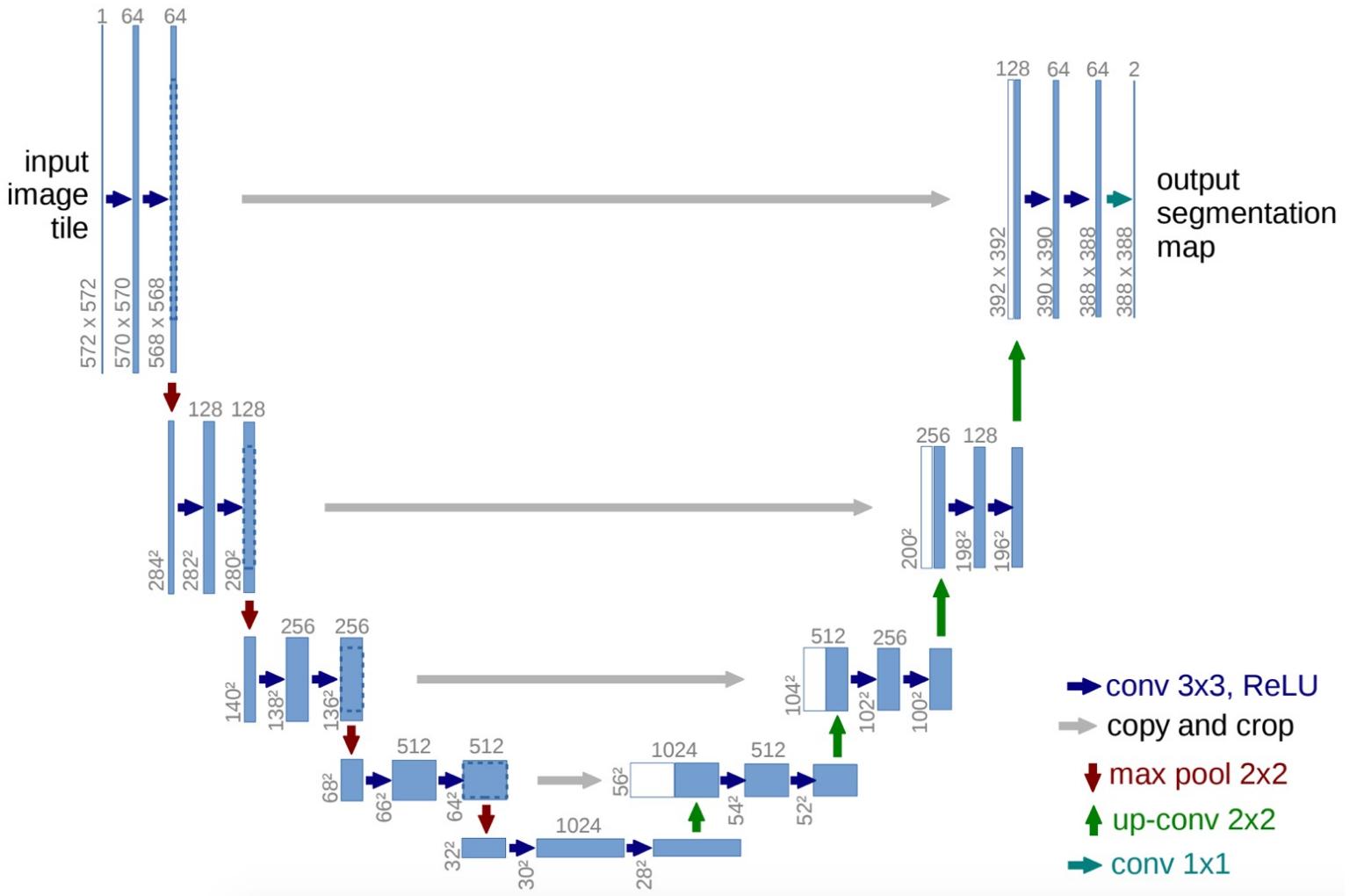


精准农业机器人可以减少需要在田间喷洒的除草剂的剂量，并且作物和杂草的语义分割可以帮助它们实时触发除草行为。

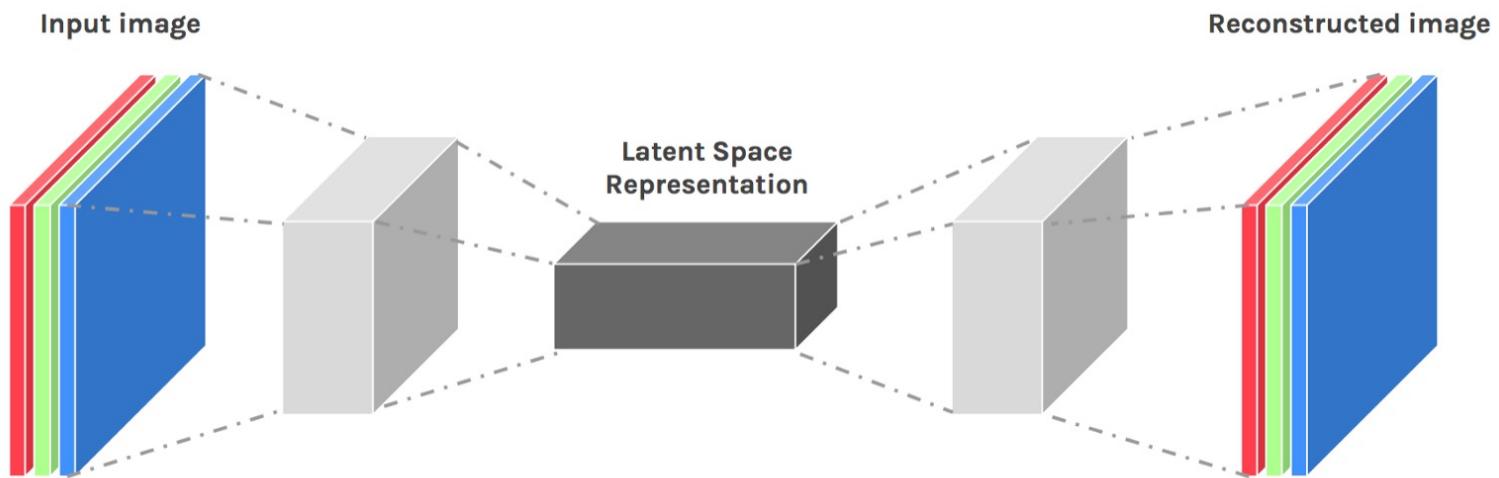
U-Net: Convolutional Networks for Biomedical Image Segmentation

- <https://arxiv.org/pdf/1505.04597.pdf>

网络结构



Transposed Convolutions/ Deconvolution/*up convolutions*



<https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694>

Transposed Convolutions/ Deconvolution/*up convolutions*

- 卷积和池化的输出相对于输入尺寸缩小
- 池化通过增加视场, 帮助我们理解图像中的物体是什么, 但是这个操作也丢失了物体的位置信息
- 在语义分割中, 我们不但需要知道图像中的物体是什么, 还需要知道物体在哪儿. 我们需要一种将图像放大同时保留位置信息的操作
- Transposed Convolution是图像过采样(up sampling)的最佳选择, 它通过误差向后传递学习最佳的权值使得低分辨率图像转换为高分辨率图像的效果最好.

Conv2DTranspose V.S. Image Resizing



Ground Truth



$\frac{1}{4}$ Sized
Input



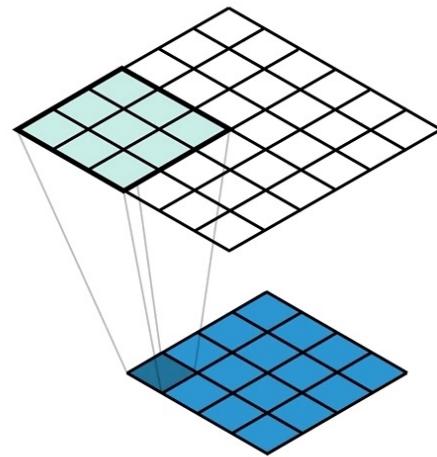
Bicubic



Super Resolution
Network

http://openaccess.thecvf.com/content_ECCV_2018/html/Seong-Jin_Park_SRFeat_Single_Image_ECCV_2018_paper.html

Conv2DTranspose :
3*3卷积核, 输入4*4, 输出6*6, 填充 0



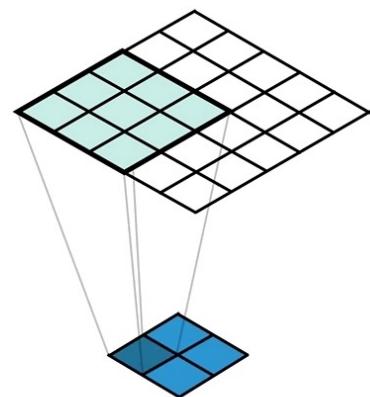
Conv2DTranspose演示

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1																						
2	Input				Kernel				Output													
3																						
4																						
5	1	1	1	1					1	1	1				1	2	3	3	2	1		
6	1	1	1	1					1	1	1				2	4	6	6	4	2		
7	1	1	1	1					1	1	1				3	6	9	9	6	3		
8	1	1	1	1											3	6	9	9	6	3		
9															2	4	6	6	4	2		
10															1	2	3	3	2	1		

你可能已经发现输出的边缘积累比中心位置少。通常这不是问题，因为内核权重会为此进行调整，也可能是负的。

Deconvolution:

具有 3×3 卷积核和 2×2 步长的Conv2DTranspose应
用于 2×2 输入，产生 5×5 输出



具有 3×3 内核和 2×2 步长的Conv2DTranspose
应用于 2×2 输入，提供 5×5 输出

Conv2DTranspose的问题

- 1. 与插值法(双三次插值bicubic)或者最近邻插值相比, Conv2DTranspose是监督式学习算法, 是需要训练的
- 2. 会产生棋盘效应, 其中一个解决方案是先插值, 再使用Conv2DTranspose



Radford, et al., 2015 [1]

Salimans et al., 2016 [2]

Donahue, et al., 2016 [3]

Dumoulin, et al., 2016 [4]

<https://distill.pub/2016/deconv-checkerboard/>

代码演示

- Unet 模型: unet_semantic_segmentation.py
- 练习: <https://www.kaggle.com/c/tgs-salt-identification-challenge/data>

图像生成 (GAN)

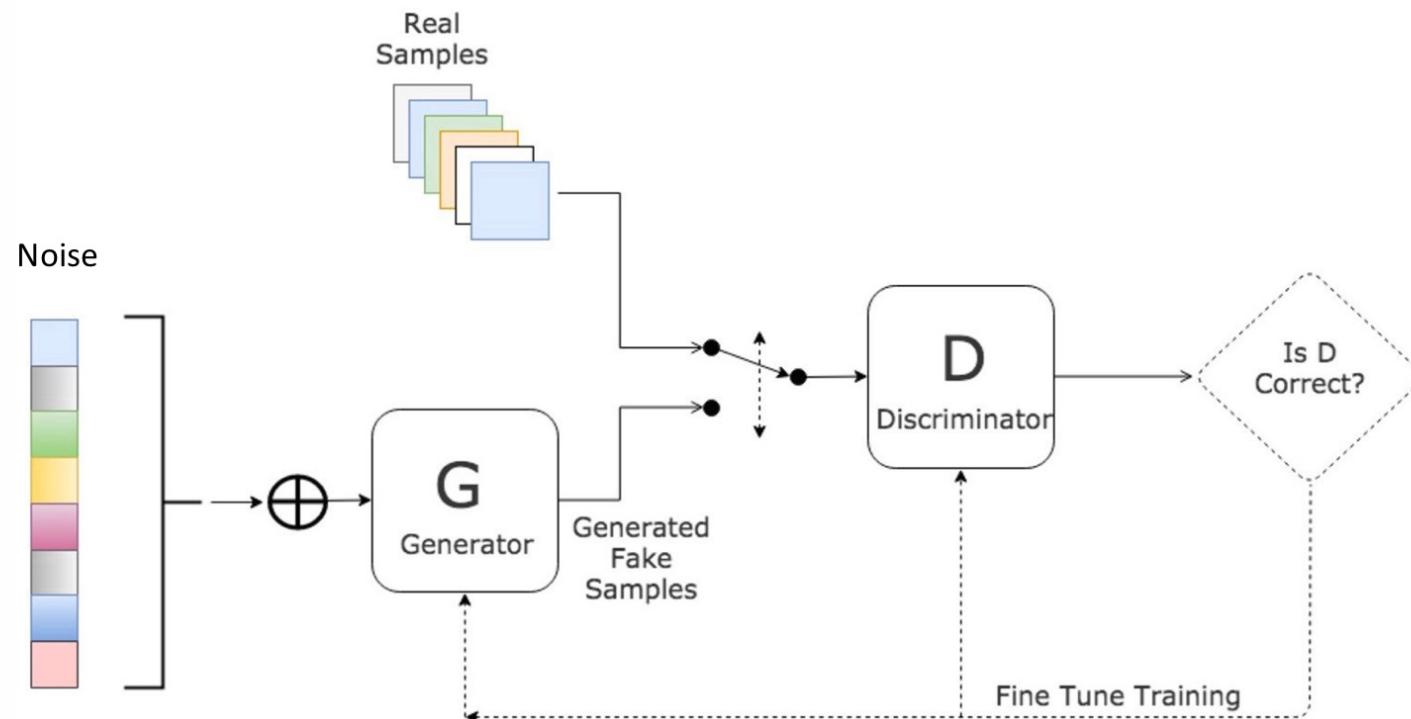
Generative Adversarial Networks (GAN)

- Ian Goodfellow 2014年提出
- 非监督式学习任务
- 使用两个深度神经网络: Generator (生成器), Discriminator(判别器)

原理

- 考虑制造假钞的例子
- 生成器: 制造假钞的人
- 判别器: 警察
- 训练过程
 - 1. 制造假钞的人生产假钞
 - 2. 警察判断是否是假钞, 如果认为是假钞, 说明假钞与真钞的区别
 - 3. 制造假钞的人按照警察给出的反馈改进假钞制造工艺
 - 重复以上3个步骤, 直到警察无法区分假钞和真钞为止

Generative Adversarial Network



<https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html>

训练GAN的基本步骤

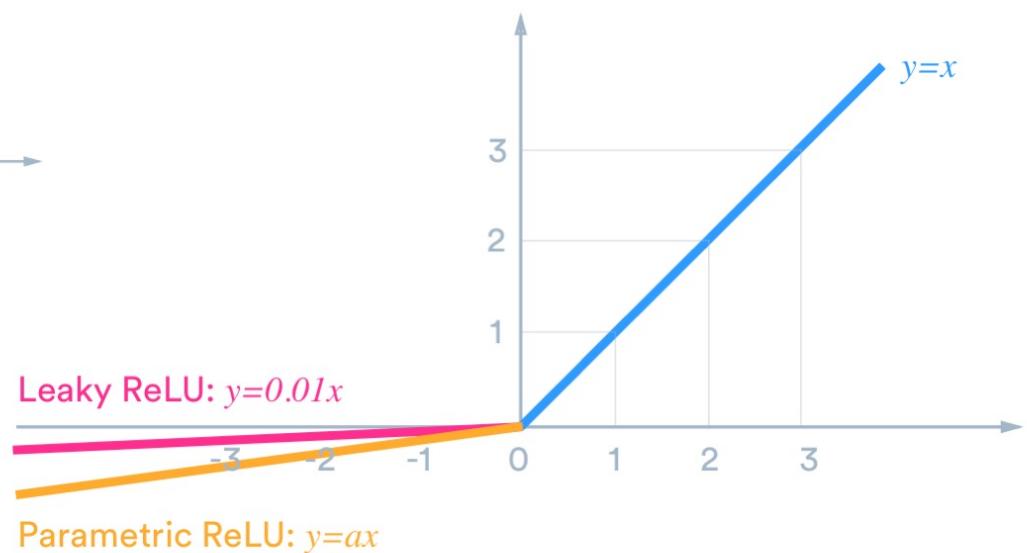
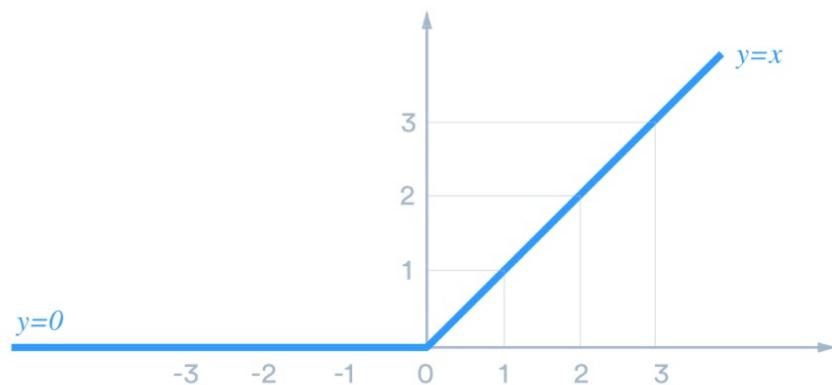
1. 对**噪声集**和**实际数据集**进行采样，选择 m 个。
2. 使用这些数据训练**判别器**。
3. 采样大小为 m 的不同的**噪声集**。
4. 在此数据上训练**生成器**。
 - 从步骤1开始重复。

案例：自动生成数字0-9

使用MNIST数据集: <http://yann.lecun.com/exdb/mnist/>

代码: fake.mnist.gan.py

LeakyRelu: 并非总是更好



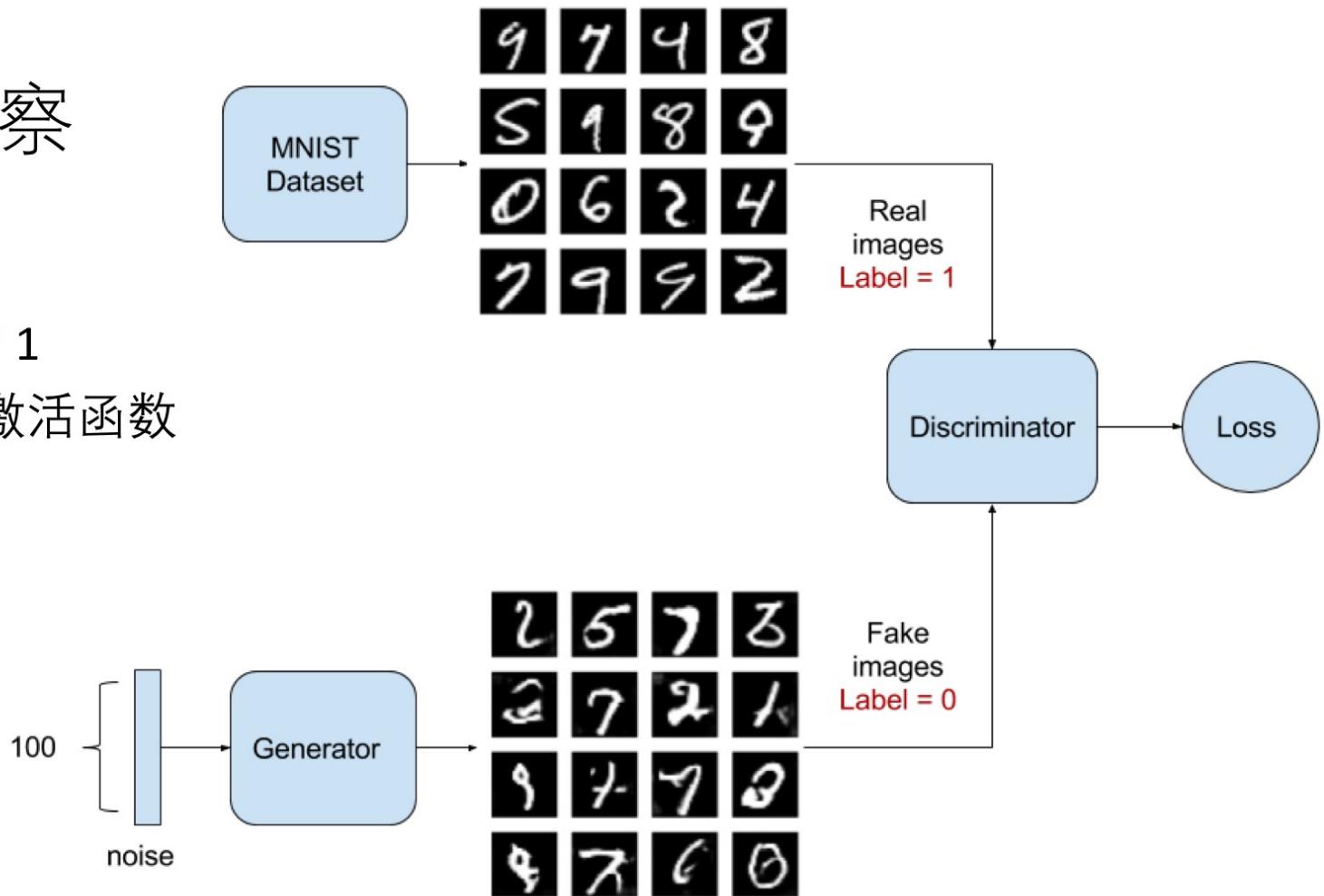
生成器: 假钞生产者

- 输入: 长度为100的向量(-1.0到1.0之间的随机数)
- 输出: $28 \times 28 \times 1$ 激活函数tanh



判别器: 警察

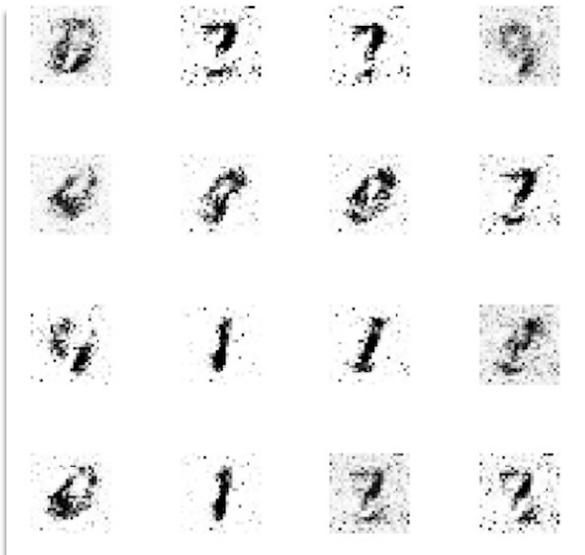
- 输入: $28 * 28 * 1$
- 输出: sigmoid 激活函数



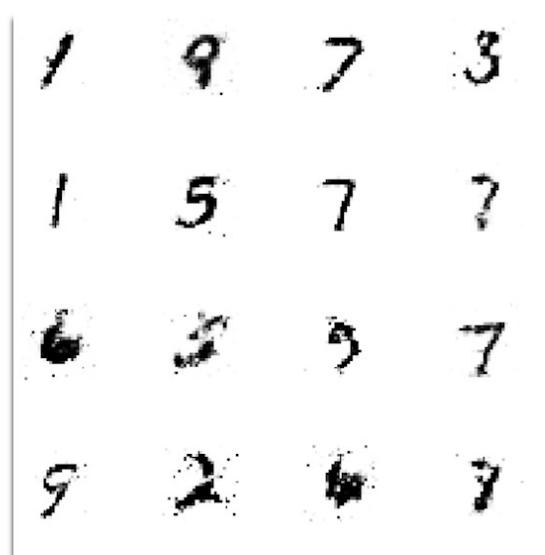
技巧

- <https://github.com/soumith/ganhacks>

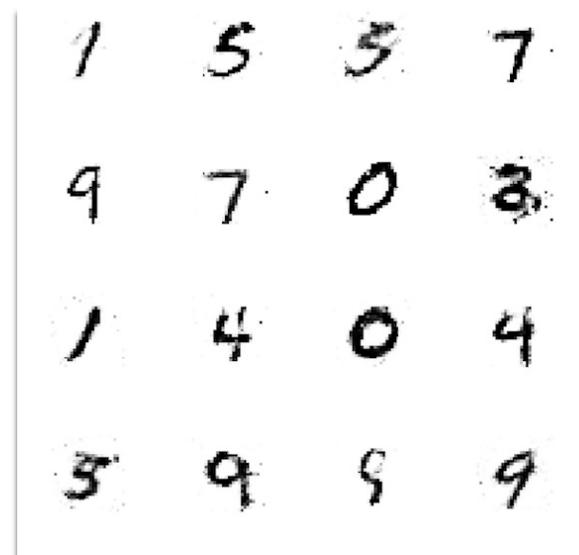
训练的结果



迭代1次



迭代20次



迭代40次

风格转移 Style Transfer

Image Style Transfer Using Convolutional Neural Networks: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf

转换为梵高风格



C



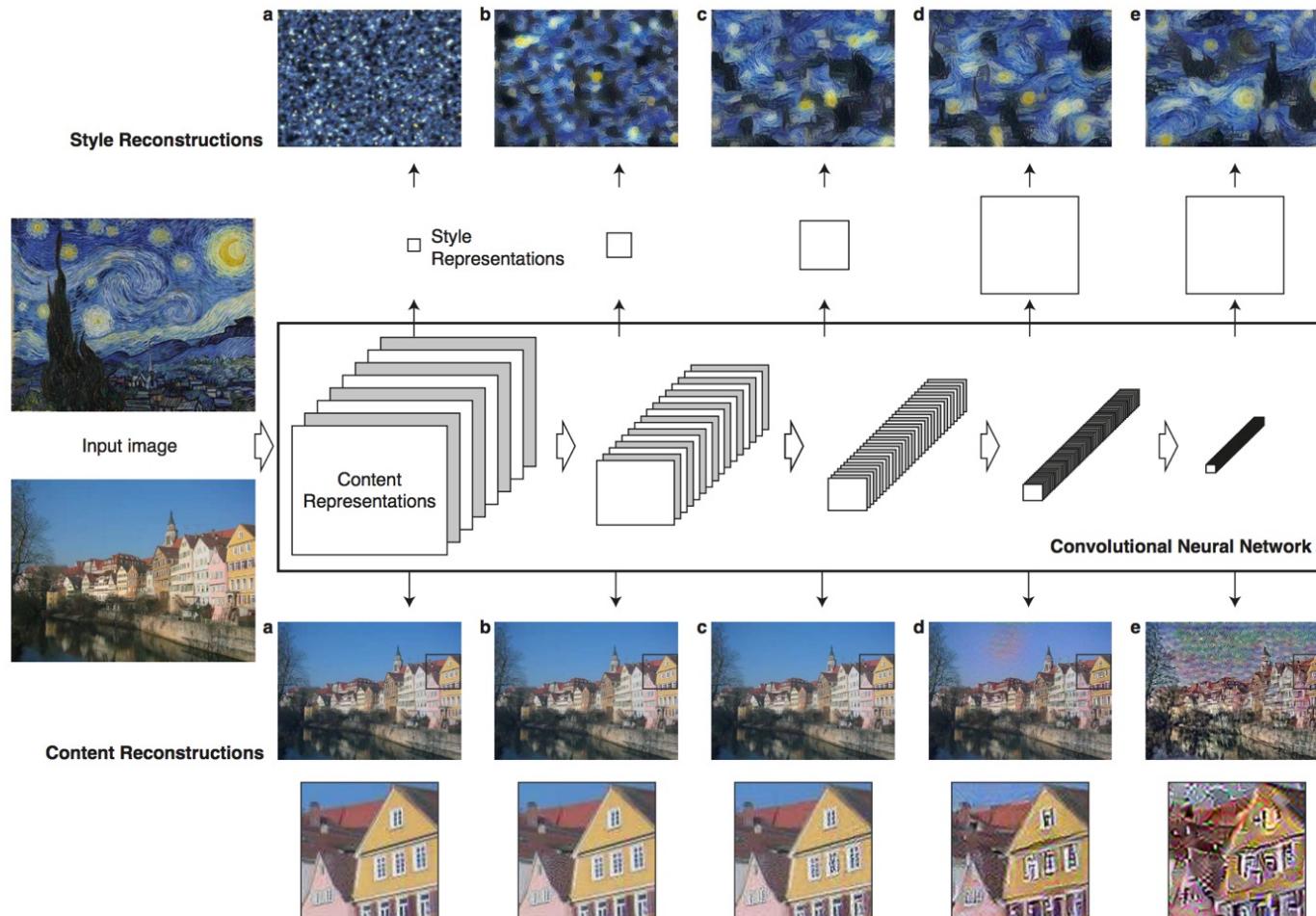
转换为毕加索风格

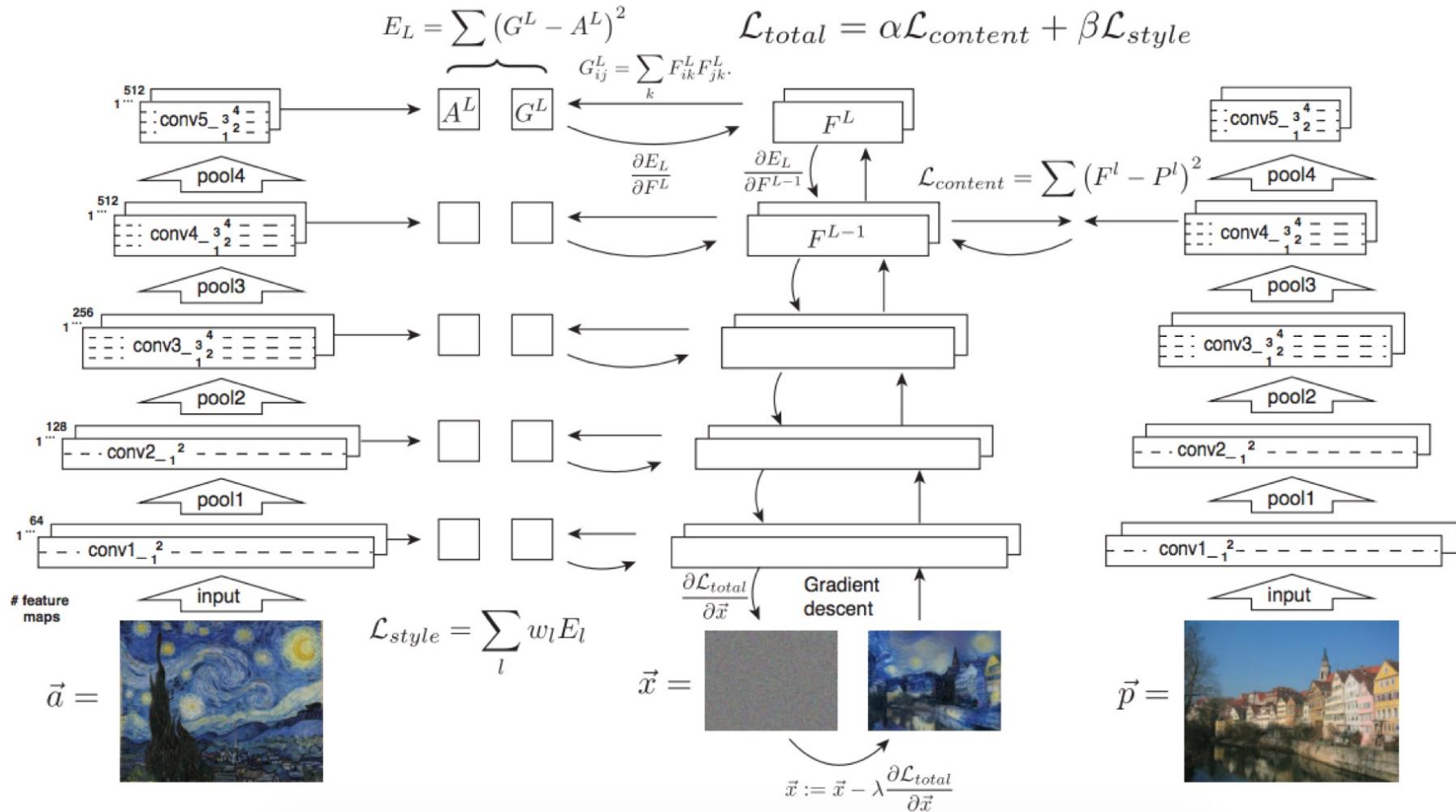


E



风格/内容表达





内容损失函数

- 第 l 层输出的特征图有 N_l 个通道, 每一个通道具有 M_l 个像素, M_l 为第 l 层特征图的高度乘以宽度
- 第 l 层的特征图可以保存在这样一个矩阵中: $F^l \in \mathcal{R}^{N_l \times M_l}$, 其中 F_{ij}^l 为第 l 层的第 i 个通道的第 j 个值
- 令 \vec{p} 与 \vec{x} 分别为内容原始图和生成的图, P^l 和 F^l 分别为它们在第 l 层的特征表达. 内容的损失函数为:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

风格损失函数

- 第 l 层输出的特征图的风格特征使用Gram matrix 来表示, $G^l \in \mathcal{R}^{N_l \times N_l}$, 其中 G_{ij}^l 为第 l 层的第 i 个通道(展开为向量)与第 j 个通道(展开为向量)的点积

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

- 令 \vec{a} 与 \vec{x} 分别为风格原始图和生成的图, A^l 和 G^l 分别为它们在第 l 层的风格表达. 第 l 层的风格损失函数为:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

- 总的风格损失函数为:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l$$

局部变化损失

- 使得生成的图像相邻像素之间的变化比较小, 较少噪声, 相当于 regularization. β 取值为1到2之间

$$\mathcal{L}_{variation} = \sum_i |y_{i+1} - y_i|^\beta$$

总的损失函数

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) + \gamma \mathcal{L}_{\text{variation}}(\vec{x})$$

‘conv1_1’, ‘conv2_1’, ‘conv3_1’, ‘conv4_1’ and ‘conv5_1’ ($w_l = 1/5$ in those layers, $w_l = 0$ in all other layers) . The ratio α/β was either 1×10^{-3} (Fig 3 B), 8×10^{-4} (Fig 3 C), 5×10^{-3} (Fig 3 D), or 5×10^{-4} (Fig 3 E, F).

示例

A



B



C



代码演示

- https://github.com/keras-team/keras/blob/master/examples/neural_style_transfer.py



+



=



The End