人工智能原理与技术

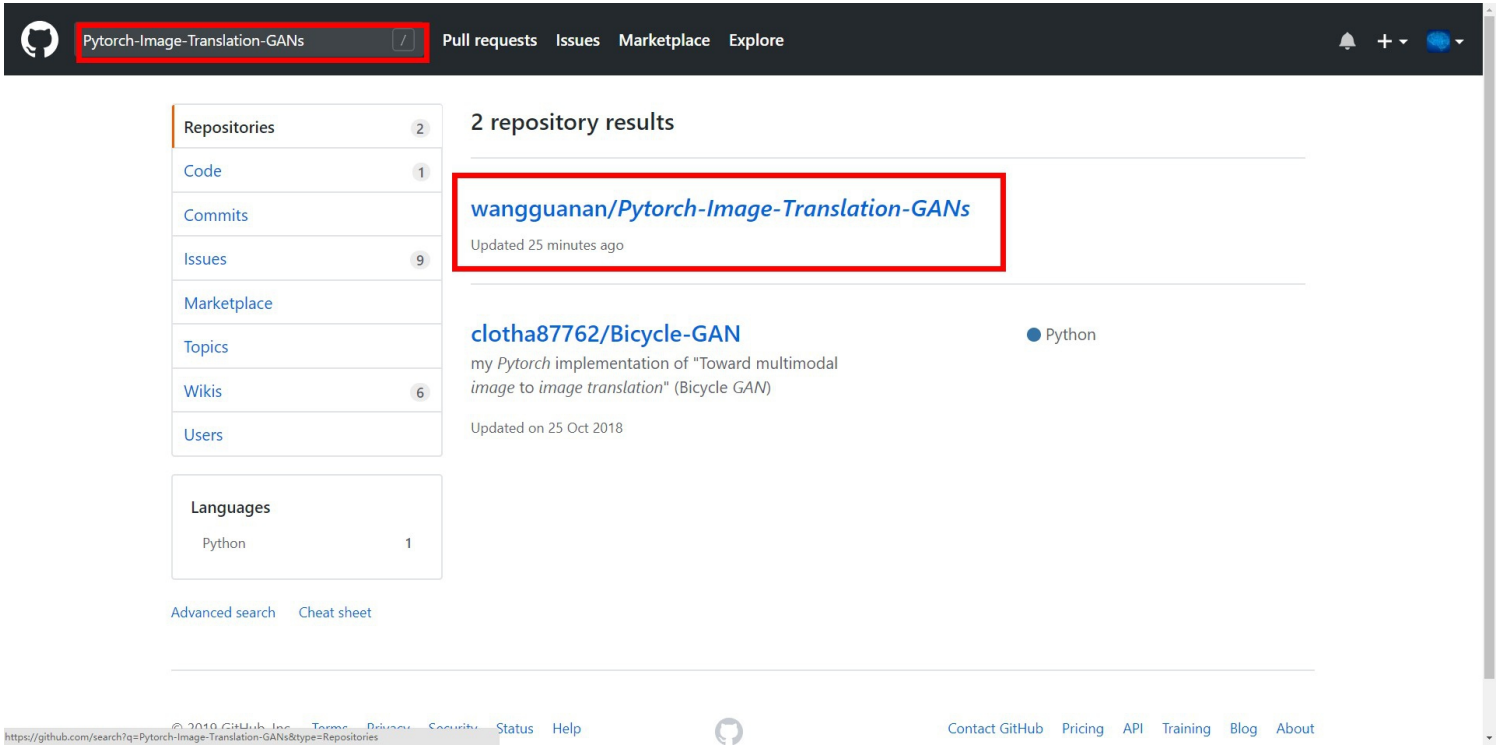
生成式对抗网络前沿

代码

代码

所有代码（Pixel2Pixel，CycleGAN，StarGAN）已开源到Github：
https://github.com/wangguanan/Pytorch-Image-Translation-GANs

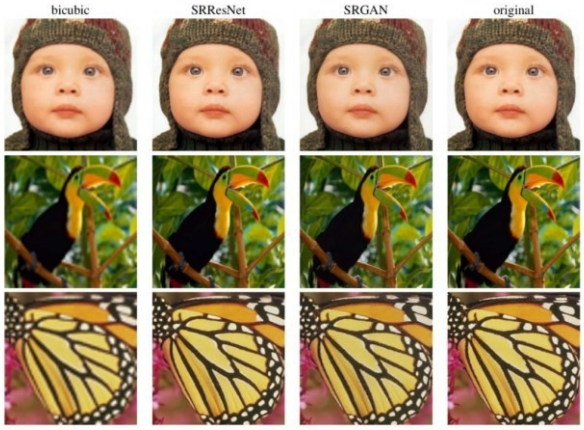
或 Github 搜索 Pytorch-Image-Translation-GANs

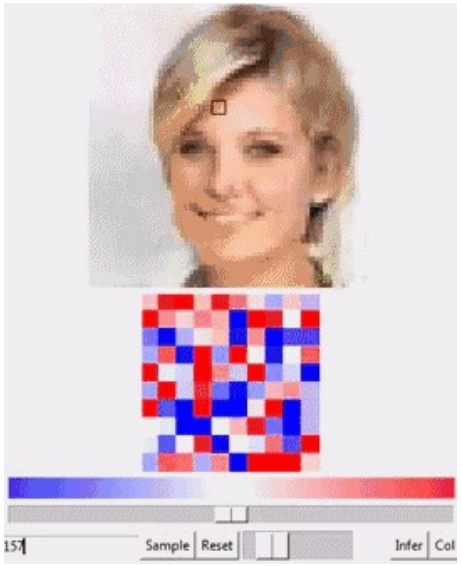


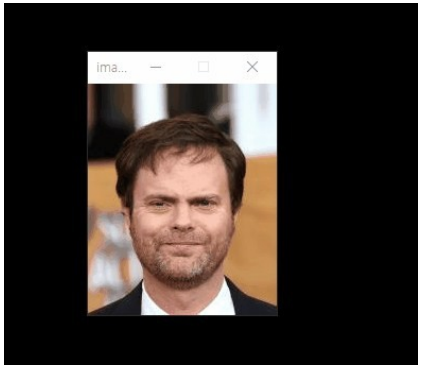
CONTENTS
目录

图像翻译

广泛存在于计算机视觉任务中







图像翻译

黑白视频还原成彩色视频

图像翻译

黑白视频还原成彩色视频

图像翻译

图像翻译定义

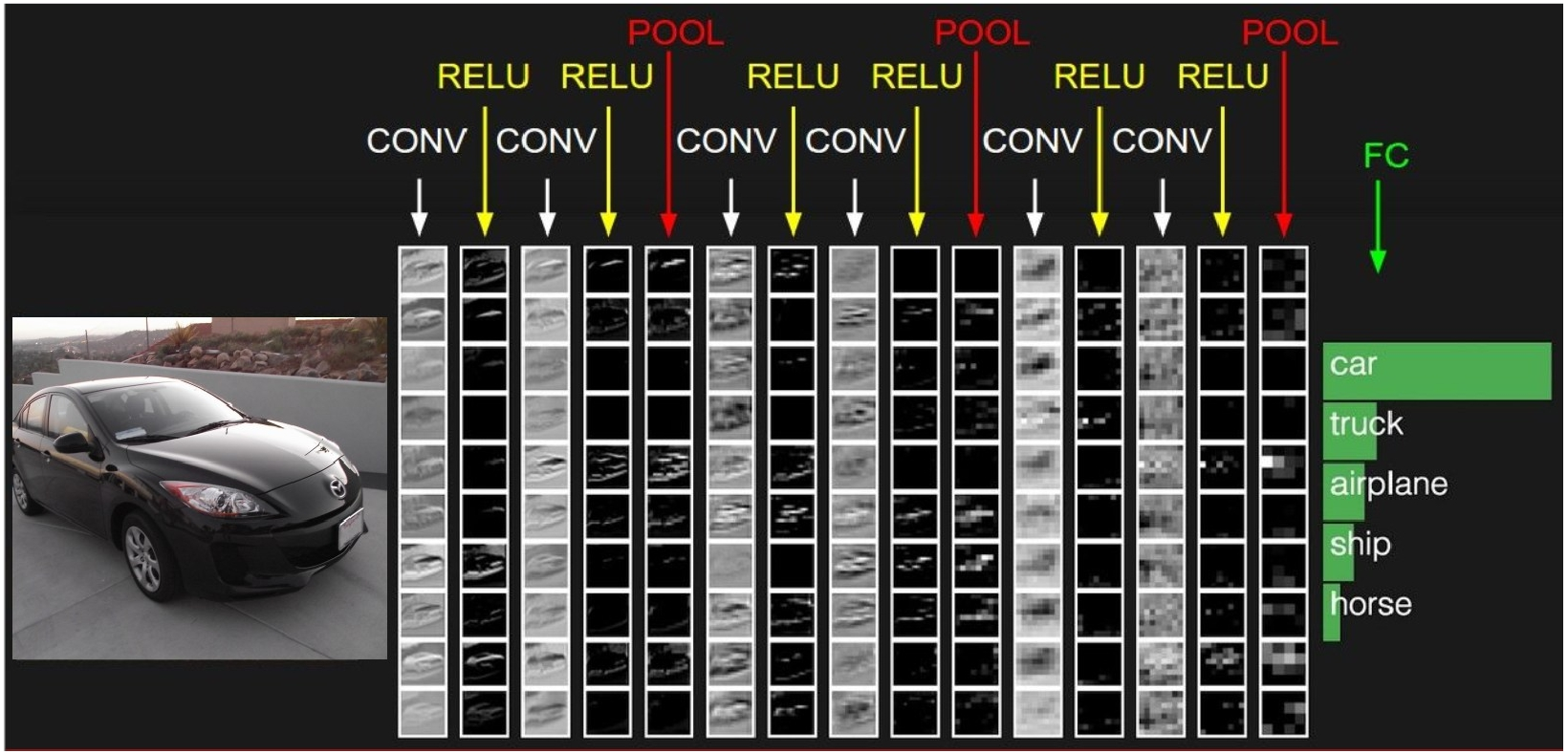
f(x|y)

学习图像到图像的映射
例如：学习彩色图像的条件分布

CONTENTS
目录

预备知识

U-NET



神经网络中，**浅层**卷积核提取**Low-Level特征**，**深层**卷积核提取**High-Level特征**

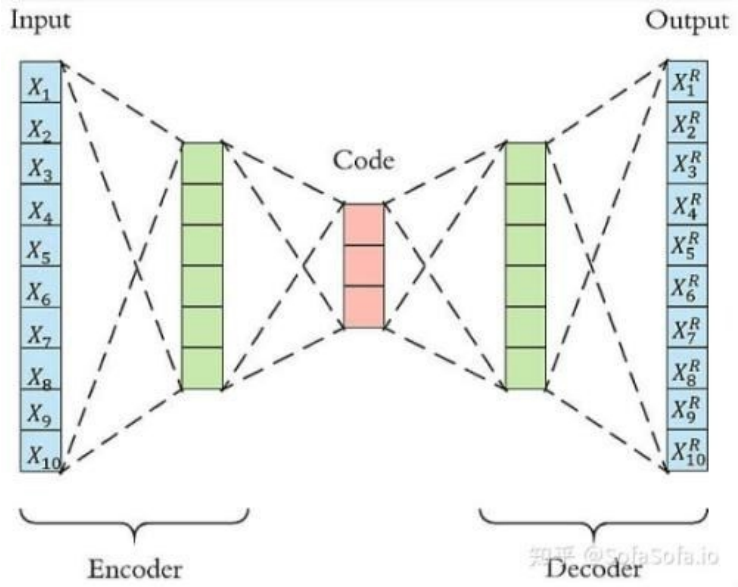
High-Level特征：分类、检测需要对图像深层理解等任务

Low-Level特征：保留更多图像细节。

图像翻译任务：需要Low-Level和High-Level 特征

预备知识

Encoder-Decoder V.S. U-Net



Encoder-Decoder：
只能提供High-Level Feature 不能提供 Low-Level Feature

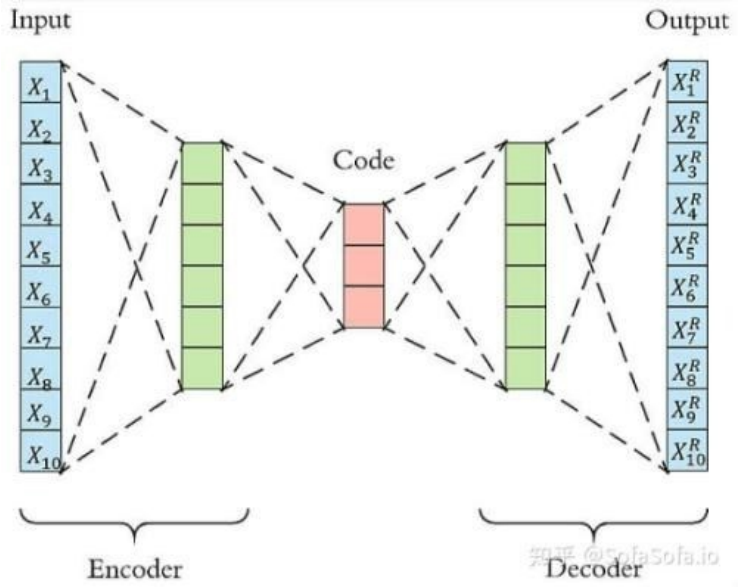
High-Level Feature

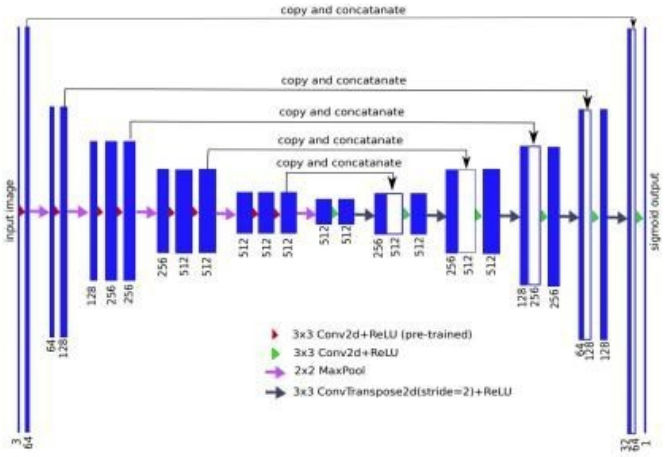
➡ 奥迪 ➡

只有奥迪（High-Level Feature）
我生成的汽车应该是什么颜色、角度、位置呢？

预备知识

Encoder-Decoder V.S. U-Net





Encoder-Decoder：
只能提供High-Level Feature 不能提供 Low-Level Feature

High-Level Feature → 奥迪 →

只有奥迪（High-Level Feature）
我生成的汽车应该是什么颜色、角度、位置呢？

U-Net：同时提供High- Low- Level feature

Low-Level Feature 位置（中间）
角度（正面）
颜色（红色）

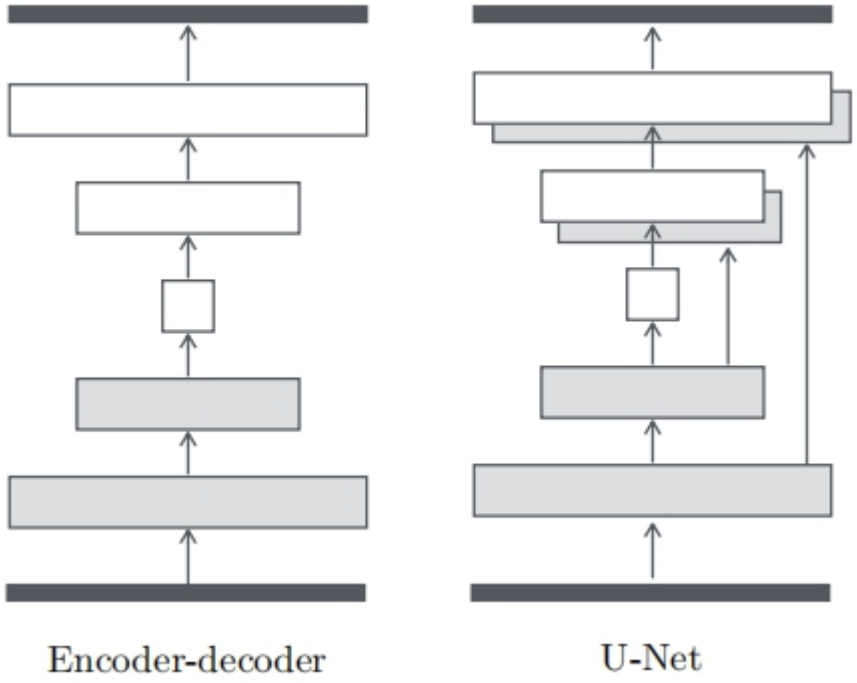
High-Level Feature → 奥迪 →

我应该生成一个红色奥迪正面的图片，汽车在图片中心。

预备知识

Encoder-Decoder V.S. U-Net

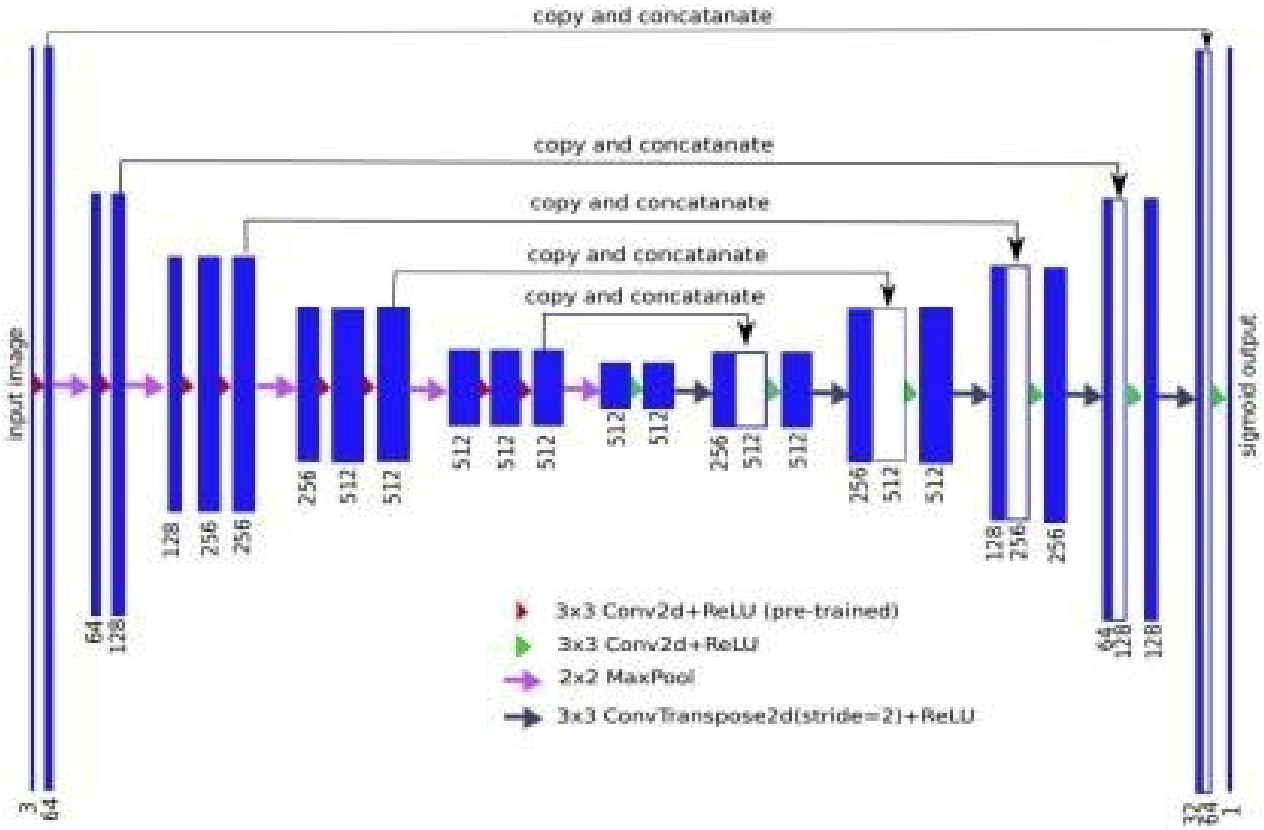
- 深度神经网络：
  - 浅层学习低级特征(如边缘)
  - 深层学习高级特征(如类别)
- 图像翻译：
  - 输入图像和输出图片共享大量的低级特征
- 自编码器结构：
  - 只学习到高级特征
  - 丢失低级特征
- U-Net：
  - 同时学习高级、低级特征



Encoder-decoder　　U-Net

预备知识

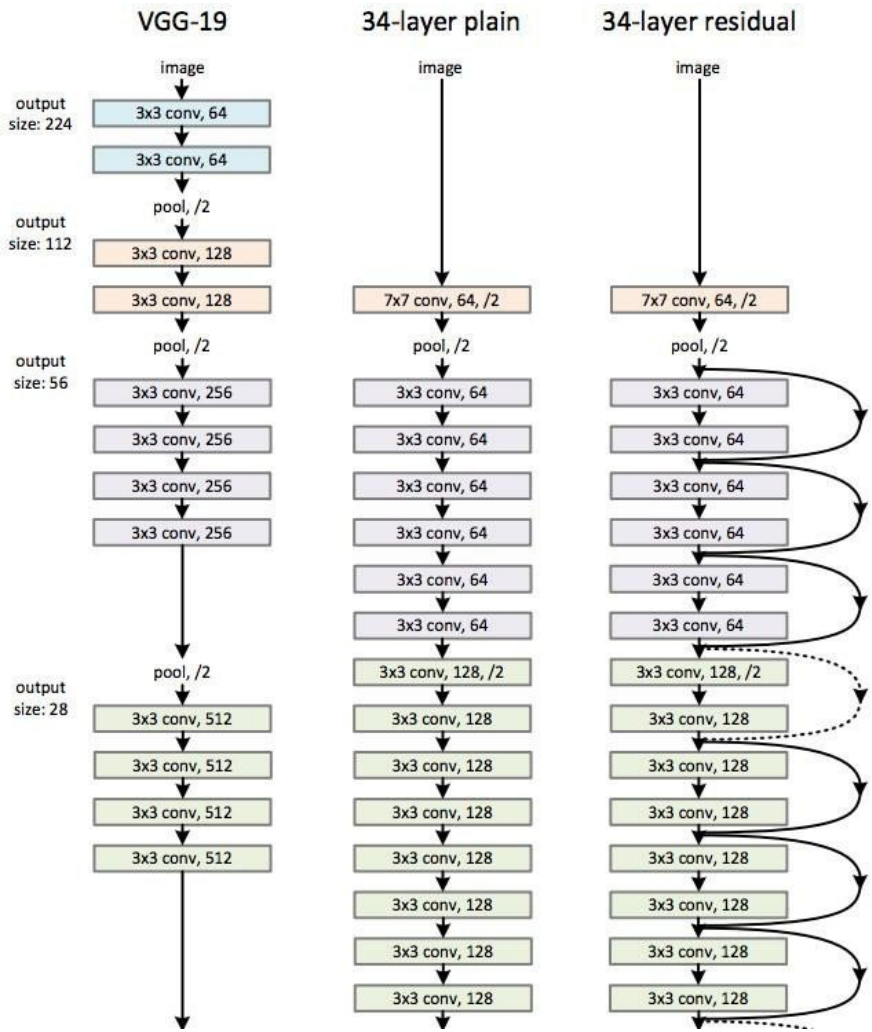
U-Net



把浅层特征沿着**channel**维度**concat**到深层特征

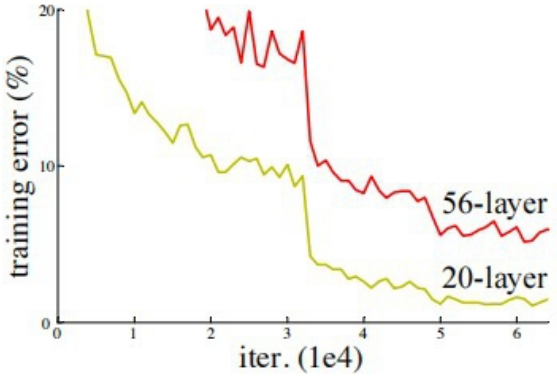
预备知识

U-Net --> ResGenerator

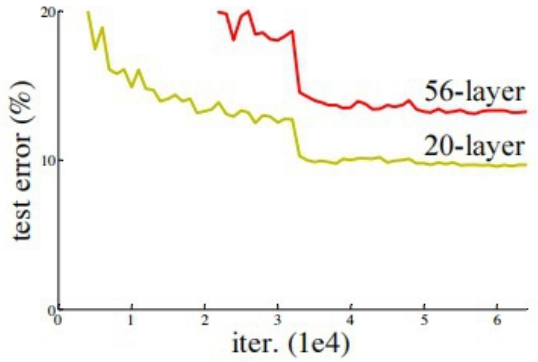


ResNet 通过残差链接，解决了退化问题

退化

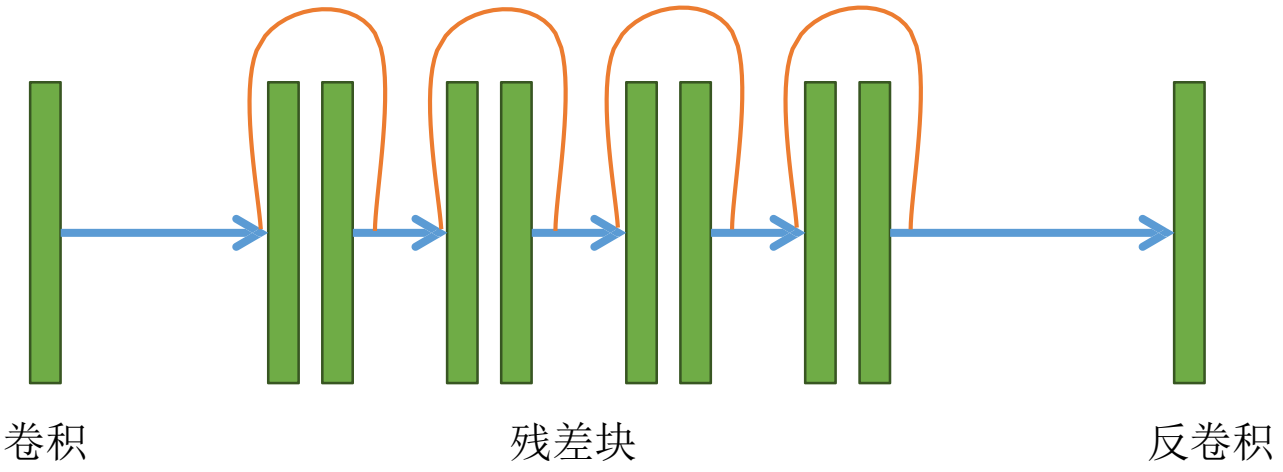


测试误差



预备知识

U-Net --> ResGenerator



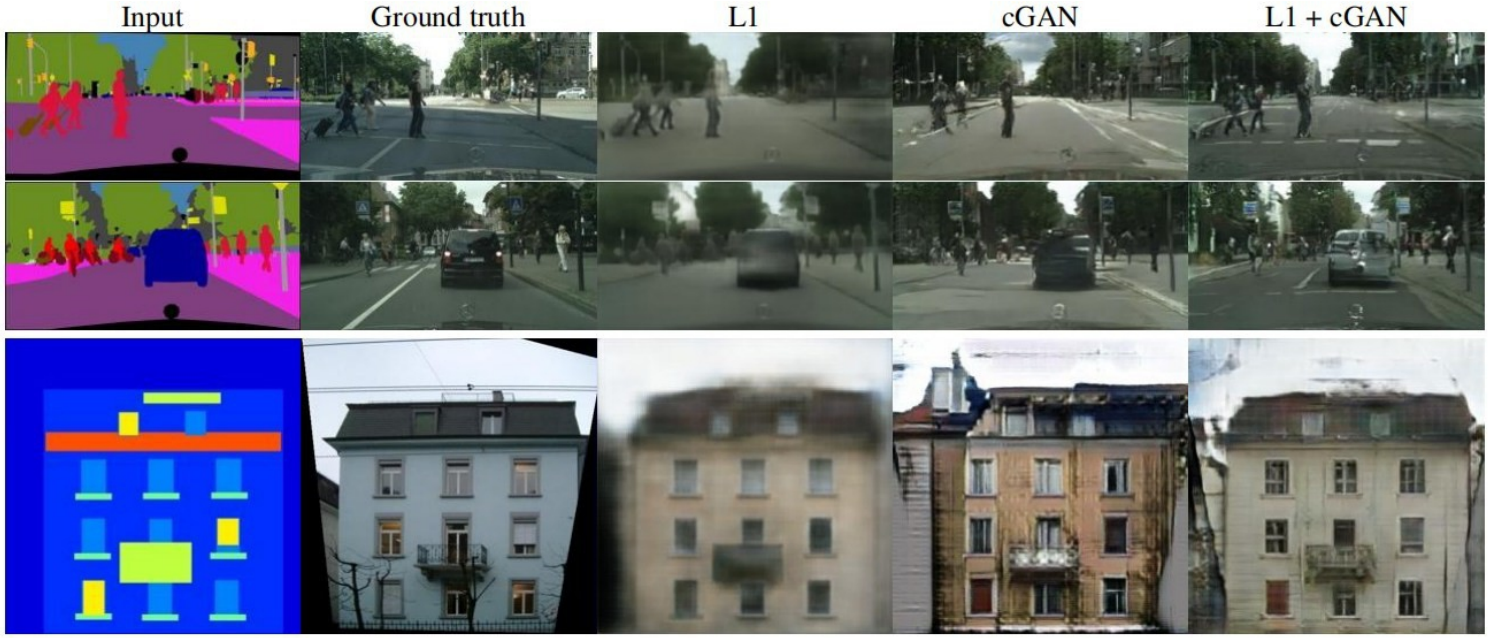
ResGenerator：

把残差链接的思想引入到生成器中

既可以传递High/Low Level Feature，又可以避免退化问题（使得网络更容易训练）

预备知识

PatchGAN



- 为什么要PatchGAN
  - L1、L2损失可以学习到低频特征：局部模糊，但 整体准确
  - GAN Loss可以学习到高频和低频特征
- 结论
  - 不需要 GAN Loss 去学习低频特征（整体特征）

- PatchGAN优势
  - 判别器参数更少，训练速度更快
  - 适用于任意大小图片

预备知识

优势：
1. 减少网络参数
2. 关注局部细节
3. 可以用于任意大小的图像

PatchGAN

整幅图像

大网络

真假

损失

原始GAN 判别器：精心设计一个大网络
比如对于256x256的图像，精心设计一个CNN，刚好卷积成一个1x1的点

Patch Patch Patch Patch

小网络

PatchGAN 判别器

x4

损失

平均

平均损失

PatchGAN 判别器：精心设计一个小网络
比如对于64x64的图像，精心设计一个小CNN，刚好卷积成一个1x1的点

预备知识

优势：
1. 减少网络参数
2. 关注局部细节
3. 可以用于任意大小的图像

PatchGAN

整幅图像

大网络

真假

损失

原始GAN 判别器：精心设计一个大网络
比如对于256x256的图像，精心设计一个CNN，刚好卷积成一个1x1的点

Patch Patch Patch Patch

小网络

PatchGAN 判别器

平均

损失

x4

平均损失

PatchGAN 判别器：精心设计一个小网络
比如对于64x64的图像，精心设计一个小CNN，刚好卷积成一个1x1的点

实际上这样用

整幅图像

小网络

PatchGAN 判别器

2x2

如果图像大小不同
这里的大小也不同

真假

x4

损失

x4

平均

平均损失

PatchGAN 判别器（实际中）：随便设计一个小网络
对于256x256的图像，随便设计一个小CNN，输出是一个任意大小（?x?）的feature map。对于不同的输入，输出不同大小的feature map

预备知识

实例归一化（Instance Normalization）

- Batch Normalizalization：对一个Batch里的所有图片进行归一化。

$$y_{tijk} = \frac{x_{tijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}, \quad \mu_i = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_i^2 = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_i)^2. \tag{2}$$

- Instance Normalization：只对一张图片进行归一化。

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_{ti})^2.$$

分类任务中：学习分类面需要考虑不同数据之间的区别。
比如，分类老虎和猫可以通过判断皮毛颜色（是否黄色）；
但是判断老虎和狗就不能通过判断比跑颜色（都是黄色）

图像翻译任务中：我只需要考虑输入图片，不需要考虑其他图片

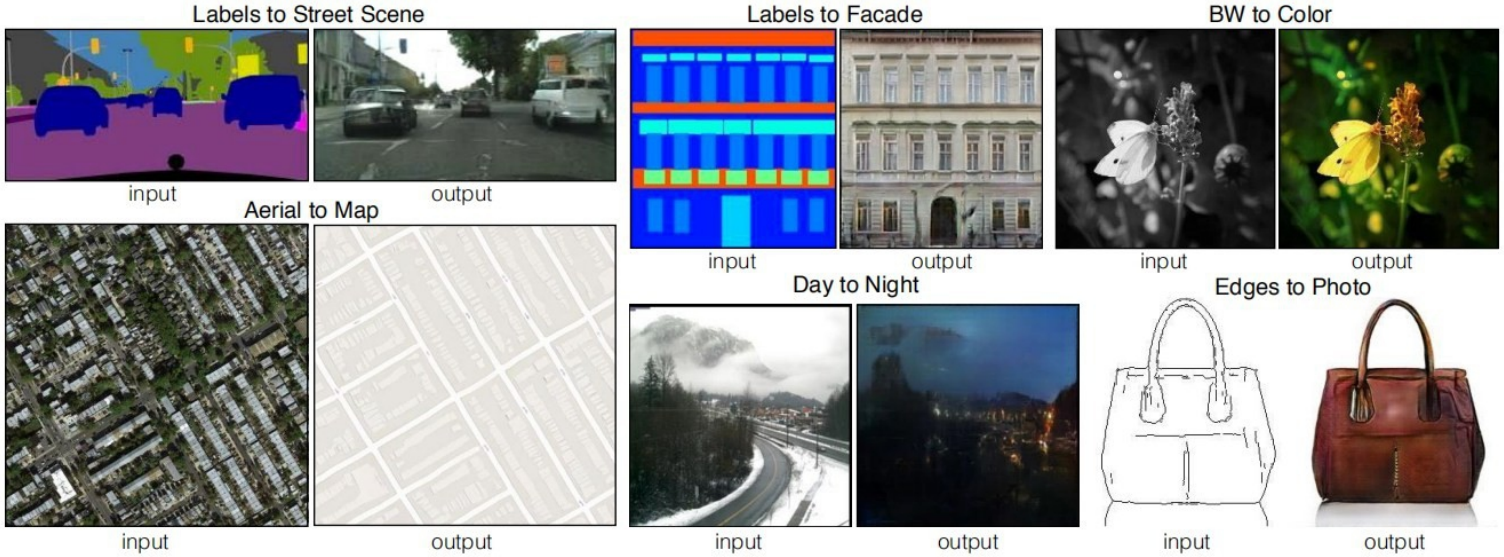
CONTENTS
目录

Pixel2Pixel

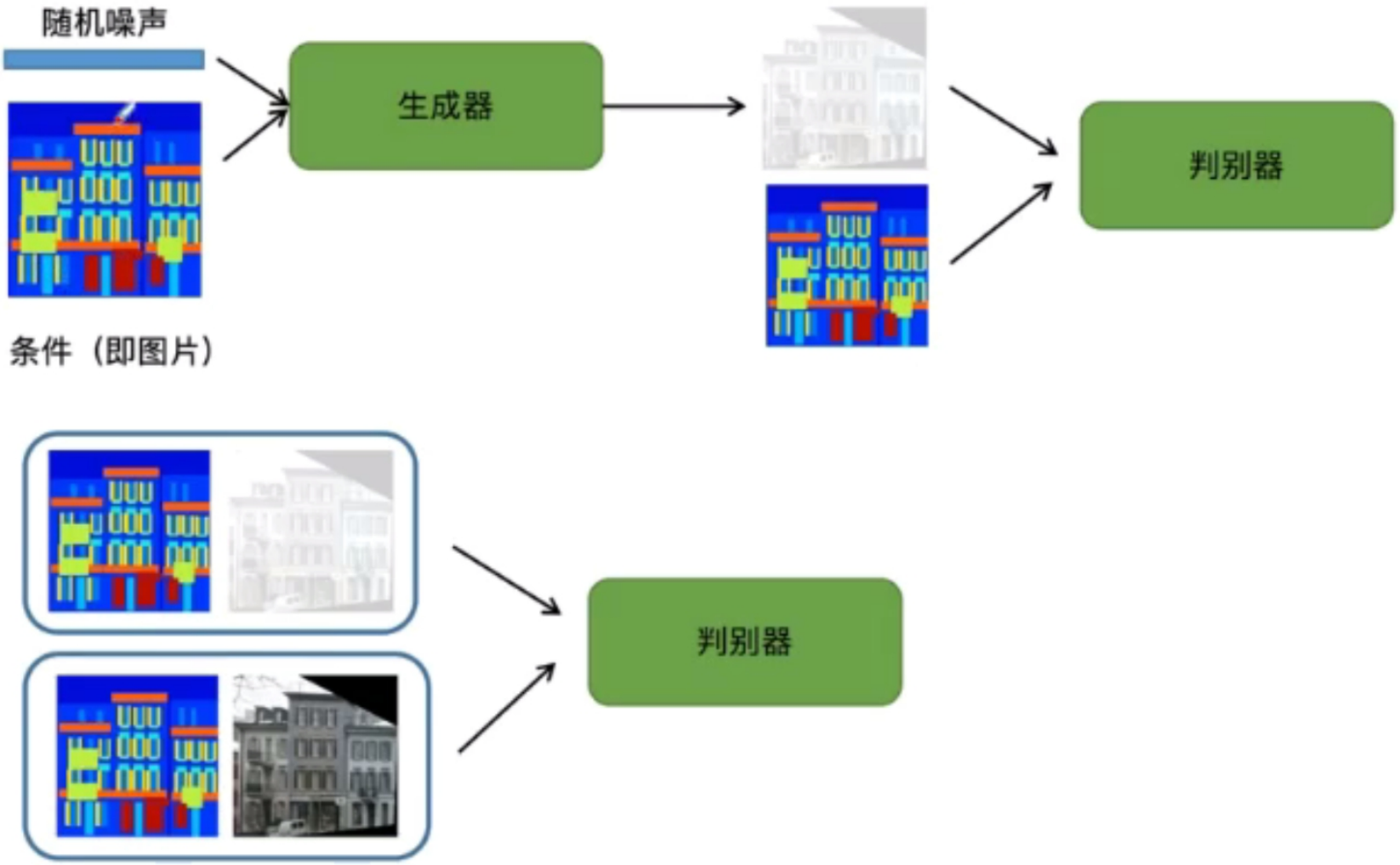
图像翻译

- 成对的数据：
  - (分割结果、街景)
  - (外观、建筑)
  - (黑白、彩色)
  - (航拍、道路)
  - (白天、黑夜)
  - (边、实物)

- 学习两两映射：
  - 分割结果->街景,分割结果<-街景
  - 外观->建筑,外观<-建筑
  - 黑白->彩色,黑白->彩色
  - 航拍->道路,航拍<-道路
  - 白天->黑夜,白天<-黑夜
  - 边->实物, 边<-实物

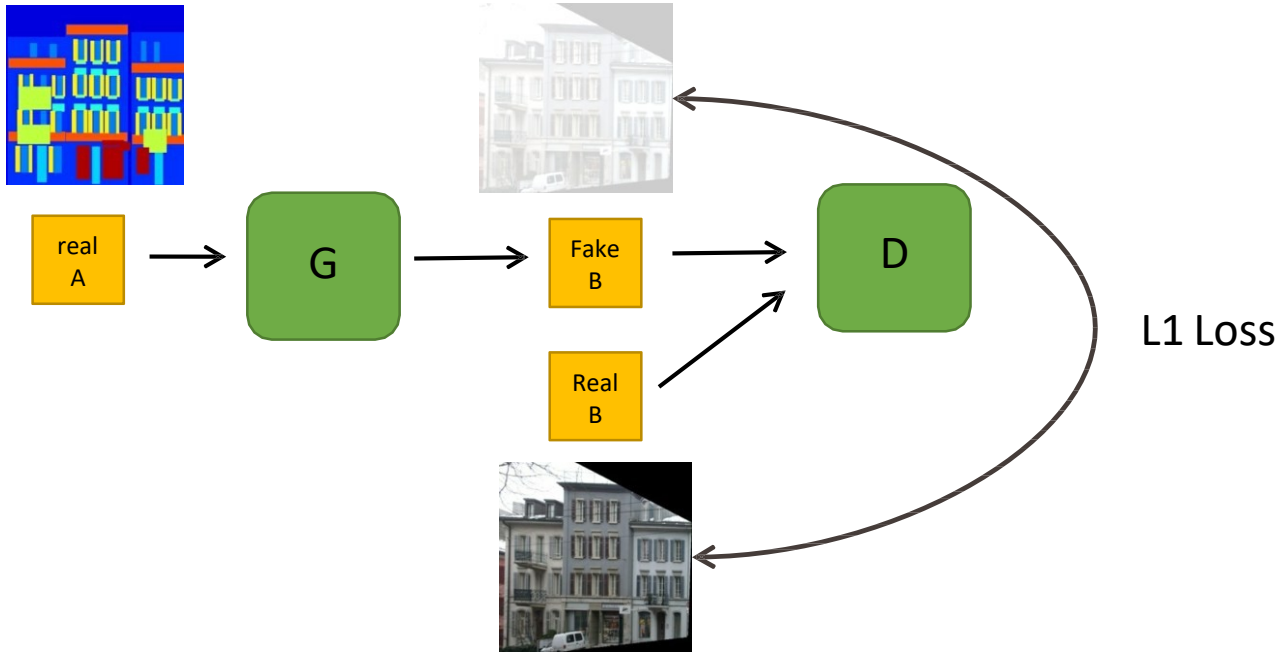


Pixel2Pixel



Pixel2Pixel

模型

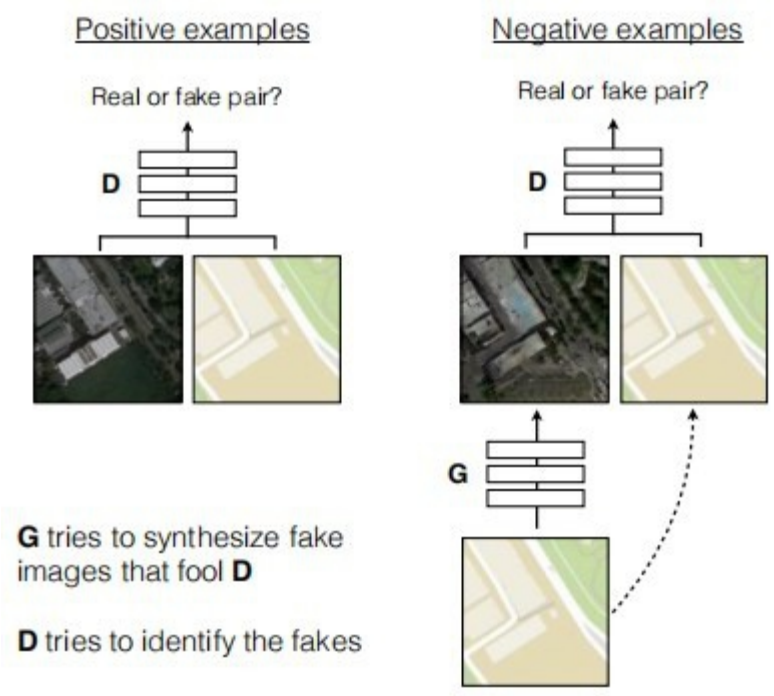


解决方案：Conditional GAN（为了方便，判别器的输入条件没有画出来） + L1 Loss

Pixel2Pixel

Pixel2Pixel

- 网络结构



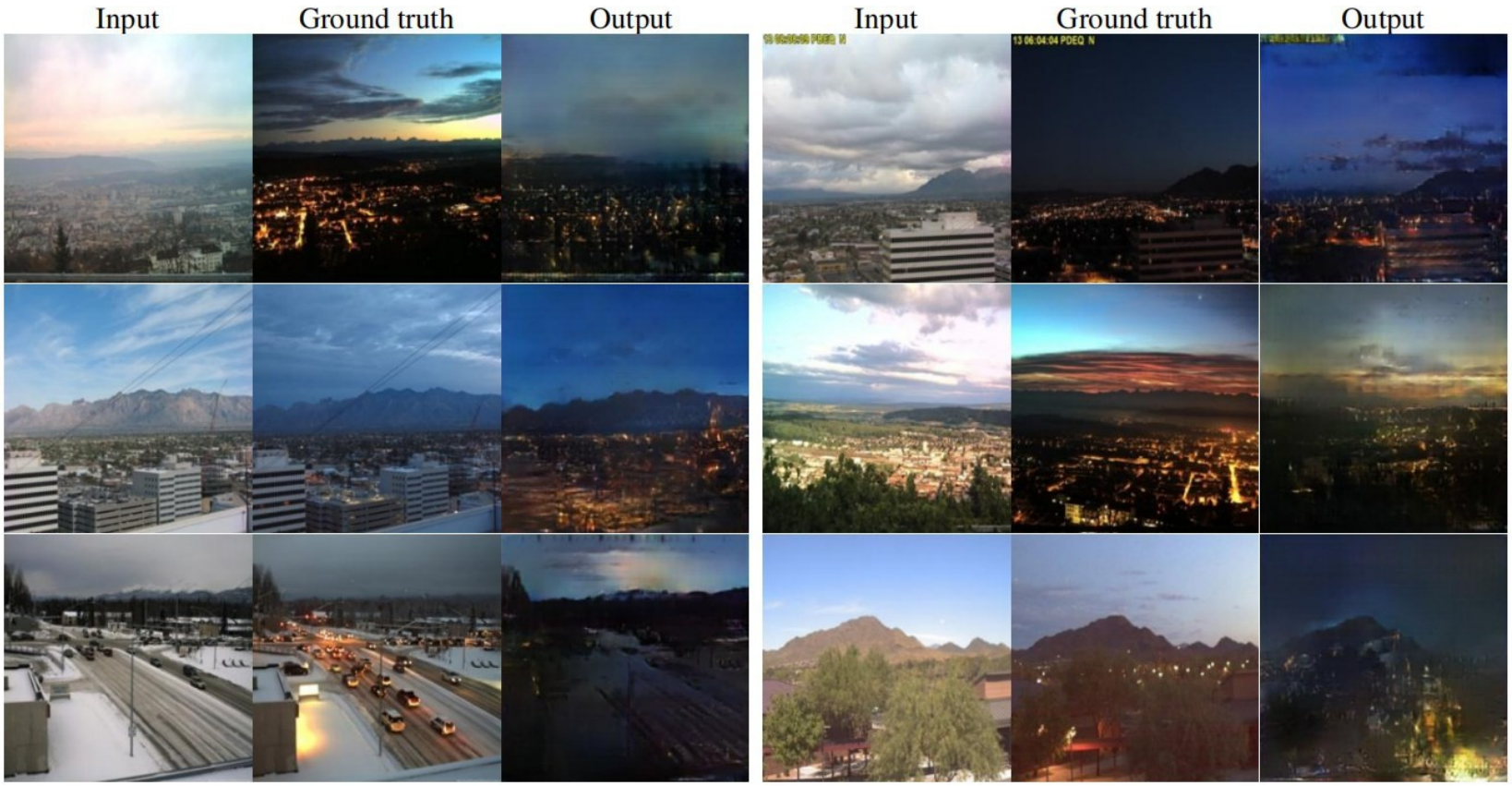
- 目标函数

$$\mathcal{L}_{GAN}(G,D) = \mathbb{E}_{y\sim p_{data}(y)}[\log D(y)] + \mathbb{E}_{x\sim p_{data}(x),z\sim p_z(z)}[\log(1-D(G(x,z))].$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y\sim p_{data}(x,y),z\sim p_z(z)}[\|y-G(x,z)\|_1]$$

Pixel2Pixel

实验结果



Pixel2Pixel

实验结果

Pixel2Pixel

Pytorch 实现

Github：
https://github.com/wangguanan/Pytorch-Image-Translation-GANs

深度学习三要素：

MAIN

数据 → **模型** → **目标函数和优化**

数据下载
数据预处理
数据加载

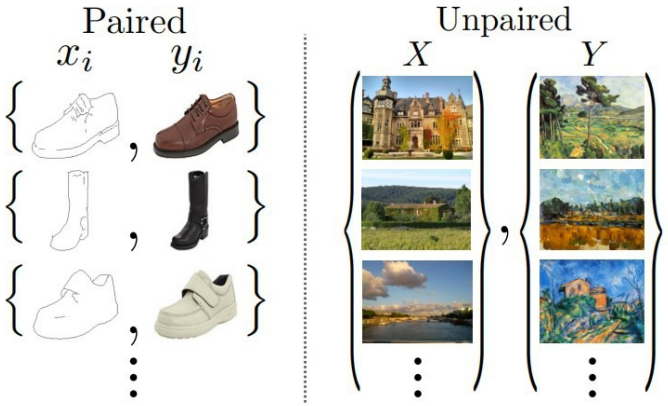
定义模型结构
定于权重初始化方式
生成器
判别器

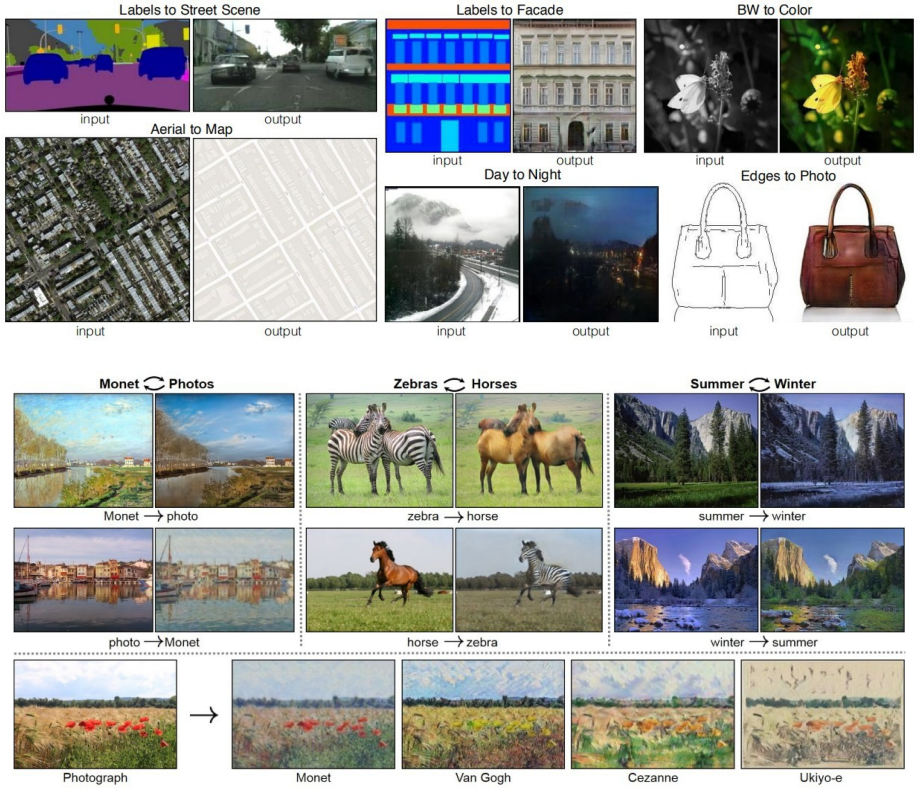
加载数据 加载模型
前向传播并计算loss
反向传播

CONTENTS
目录

CycleGAN

Paired v.s. Unpaired Data

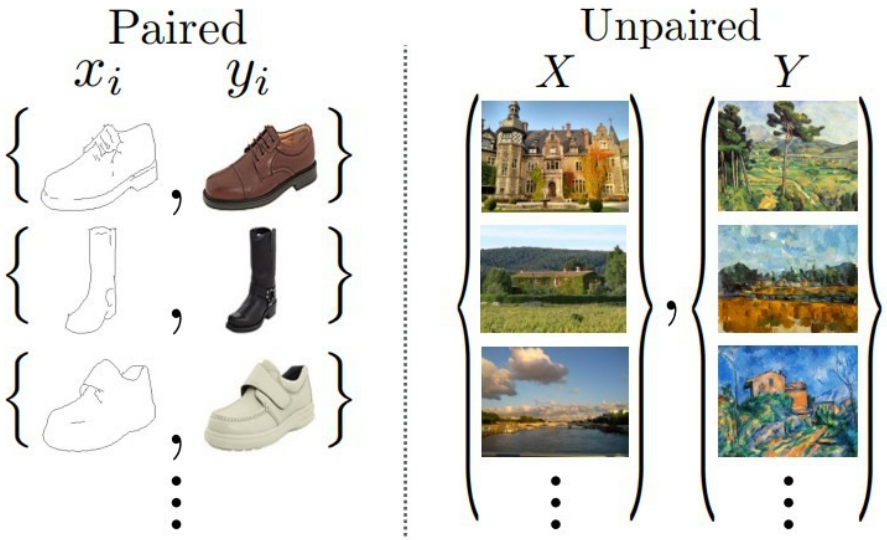




- 成对的数据难以收集
  - （分割，街景）
  - （黑白，彩色）
  - （边缘、彩色）
- 有些情况下甚至无法收集
  - （马，斑马）？
  - （照片，油画）？
- 无法收集成对数据，我们是否还能够学习映射？

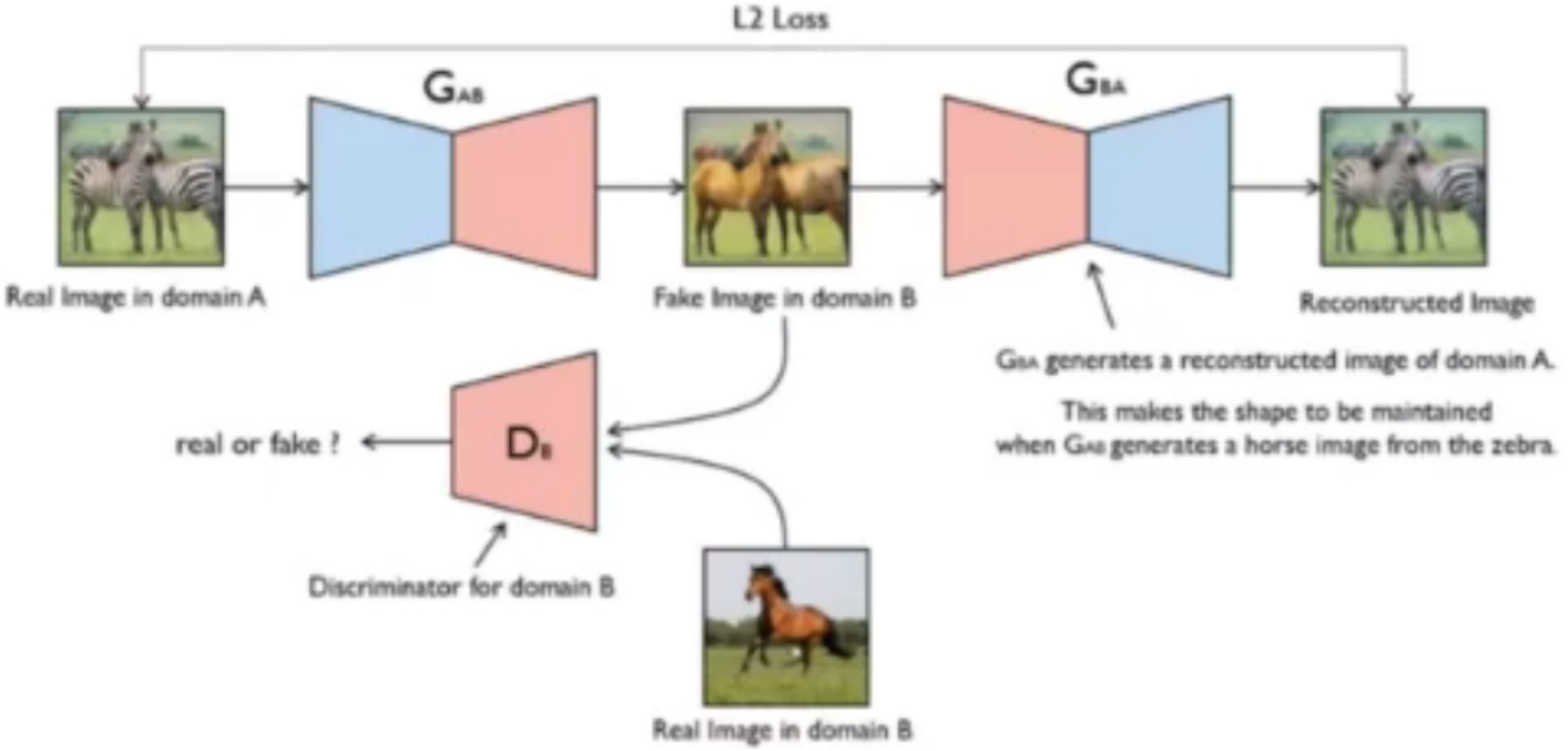
CycleGAN

Paired v.s. Unpaired Data



- 虽然我们没有办法获得成对的数 据，但是有大量的不成对的数据 供我们使用
- 这种数据能不能用来学习映射?
- 如果可以，应该怎么使用?

CycleGAN

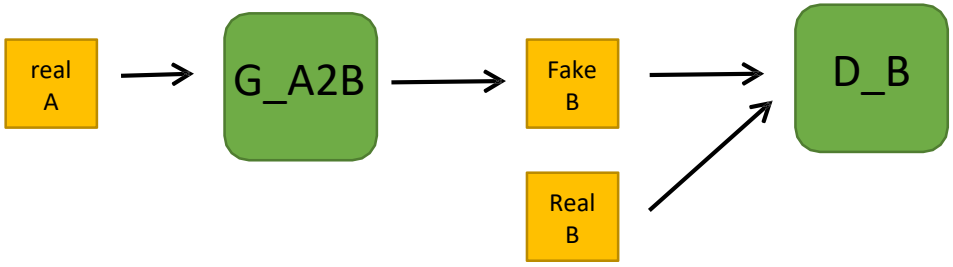

L2 Loss
G_{AB}
G_{BA}
Real Image in domain A
Fake Image in domain B
Reconstructed Image
G_{BA} generates a reconstructed image of domain A.
This makes the shape to be maintained
when G_{AB} generates a horse image from the zebra.
real or fake ?
D_B
Discriminator for domain B
Real Image in domain B

CycleGAN

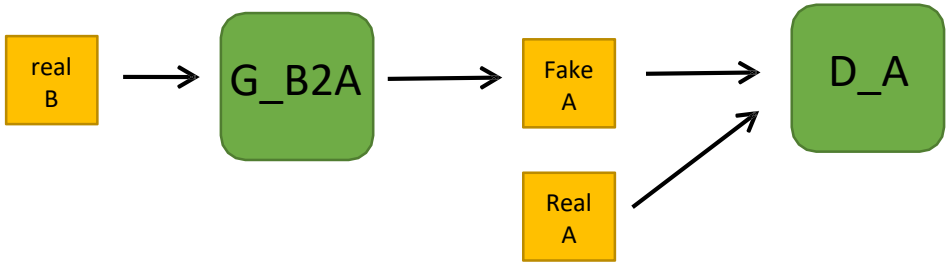
Cycle-Consistent Loss

因为要学习两个映射，所以需要两套GAN（G+D），参数不共享

学习映射A2B，把图片A（比如马）变成图片B（比如斑马）的风格



学习映射B2A，把图片B（比如斑马）变成图片A（比如马）的风格

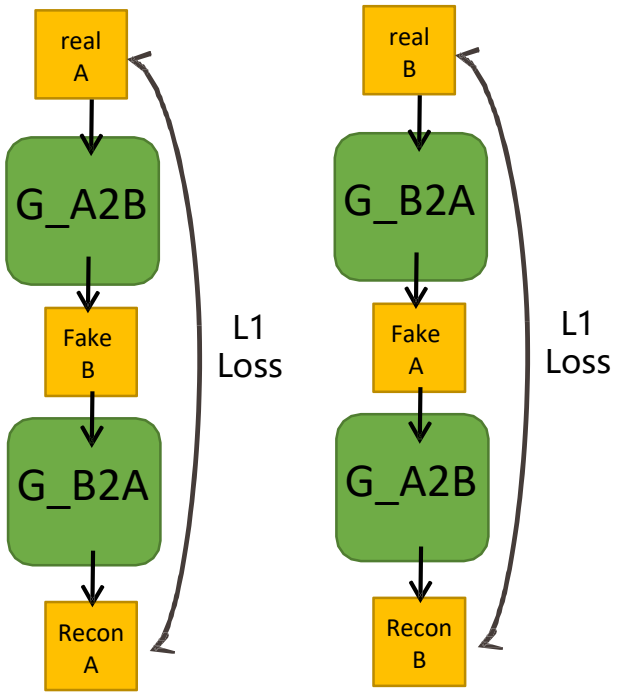


但是，**怎么保证Fake_B具有B的风格，同时还有A的结构**

比如：一直奔跑的马应该变成一个奔跑的斑马，而不是一个喝水的斑马

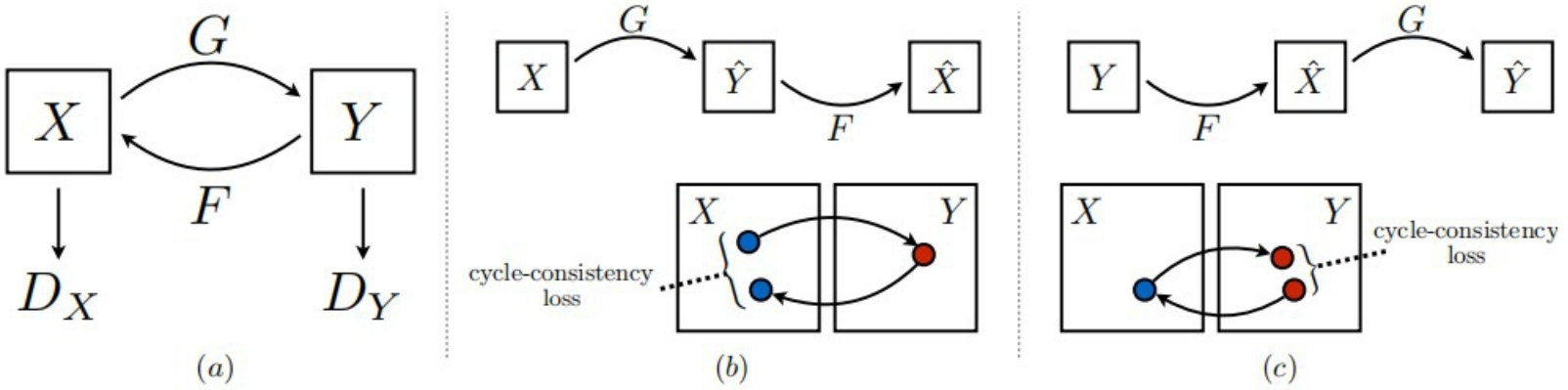
解决方案：**循环一致性损失（Cycle-Consistent Loss）**

解释：奔跑的马变成斑马，再变回马，它仍然应该是奔跑的，间接地跑本斑马也是奔跑的



CycleGAN

循环一致性损失



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

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

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F), \quad (3)$$

保证输入图片原有信息不会丢失！

CycleGAN

实验结果

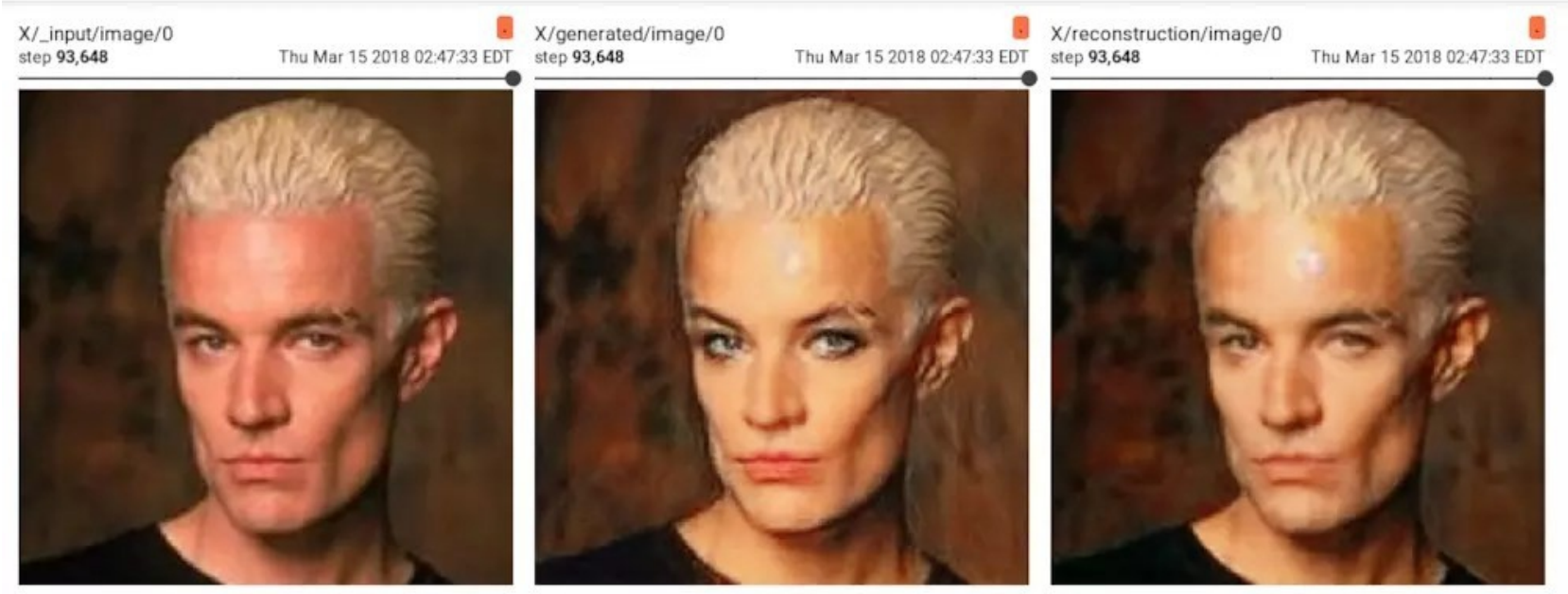
一堆男女头像，但不是成对的

CycleGAN

实验结果

https://github.com/wangguanan/Pytorch-Image-Translation-GANs

X



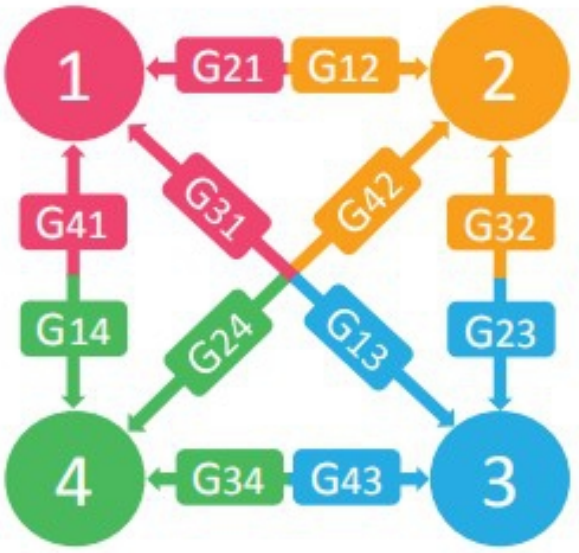
效果不错~

CONTENTS
目录

StarGAN

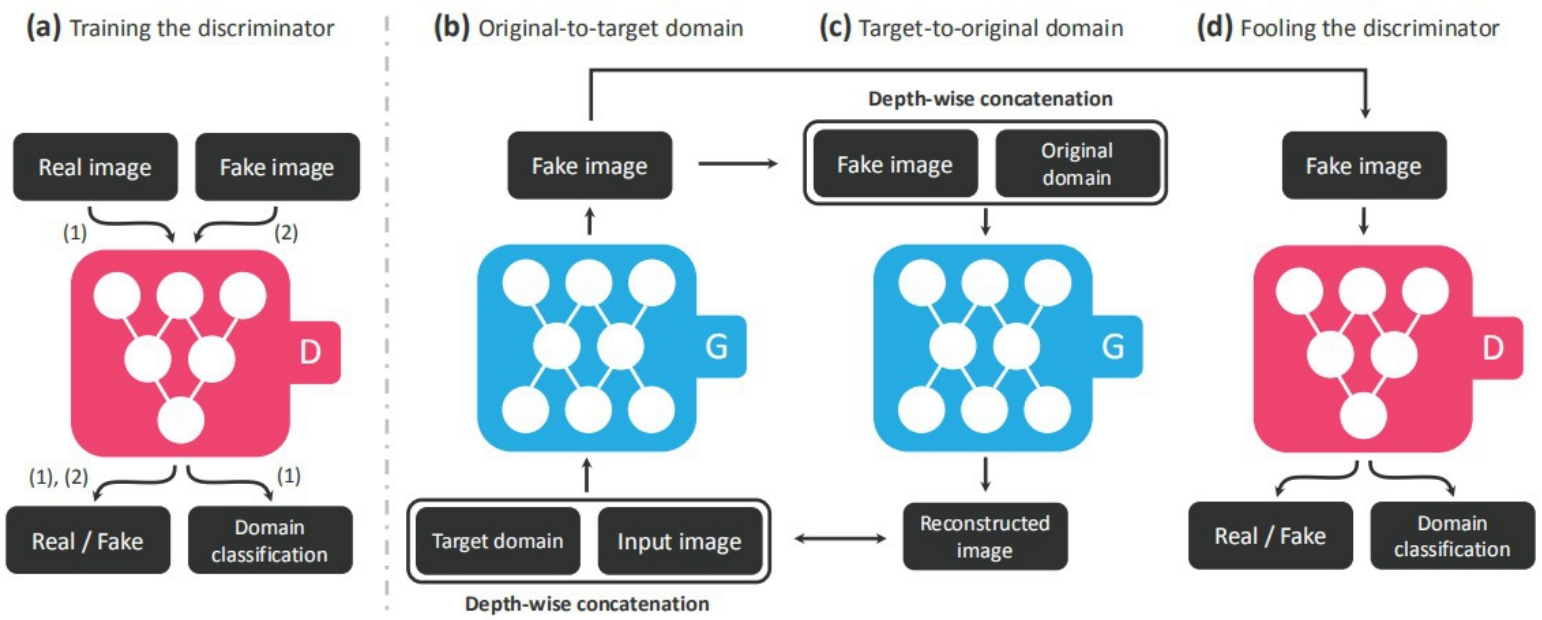
Pytorch实现

- Pixel2Pixel一次只能学习一种映射
  - 白天->黑夜
  - 黑夜->白天
  - 边缘->图片
  - 图片->边缘
- CycleGAN一次只能学习两种映射
  - 白天->黑夜，黑夜->白天
  - 马->斑马，斑马->马
- 如果我想一次学习K个映射该怎么办？
  - 害怕->生气
  - 害怕->开心
  - 生气->害怕
  - 生气->开心
  - 开心->害怕
  - 开心->生气
- 训练K个Pixel2Pixel？
- 训练K/2个CycleGAN？
- 计算资源不够！时间不够！



StarGAN

StarGAN



$$\mathcal{L}_{adv} = \mathbb{E}_x [\log D_{src}(x)] + \mathbb{E}_{x,c}[\log (1 - D_{src}(G(x,c))]$$

$$\mathcal{L}_{cls}^r = \mathbb{E}_{x,c'}[-\log D_{cls}(c'|x)]$$

$$\mathcal{L}_{cls}^f = \mathbb{E}_{x,c}[-\log D_{cls}(c|G(x,c))]$$

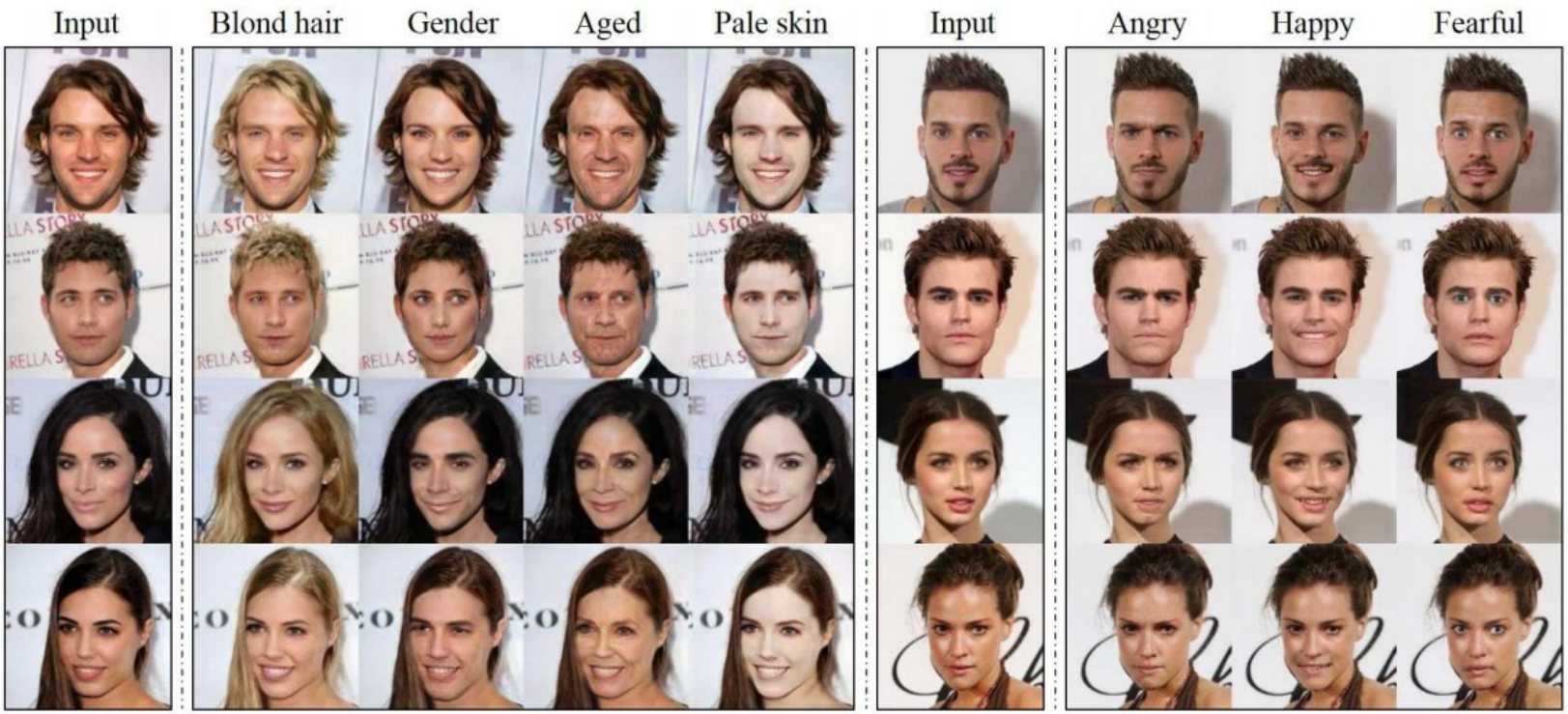
$$\mathcal{L}_{rec} = \mathbb{E}_{x,c,c'}[||x - G(G(x,c),c')||_1]$$

$$\mathcal{L}_D = -\mathcal{L}_{adv} + \lambda_{cls}\,\mathcal{L}_{cls}^r,$$

$$\mathcal{L}_G = \mathcal{L}_{adv} + \lambda_{cls}\,\mathcal{L}_{cls}^f + \lambda_{rec}\,\mathcal{L}_{rec},$$

StarGAN

实验结果



- 一个模型学习多种映射!
- 省时又省力!

StarGAN

Pytorch实现

Github：
https://github.com/wangguanan/Pytorch-Image-Translation-GANs

THANKS