基于Gan网络生成蝴蝶图像

运行

更新于 2020-10-14 13:40:07 3 1

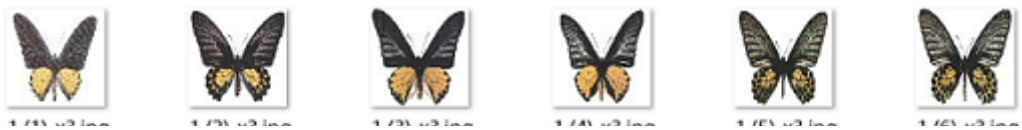
随着互联网技术的发展，人们可以获取大量图像信息，丰富了日常生活。但是在医疗症断、卫星遥感等领域，由于现实场景和图像采集工具的限制，现有的图像不足以满足要求。因此，图像生成技术也越加受到重视。

本案例使用生成对抗网络（GAN），训练包含蝴蝶图像的数据集，取得了较好的图像生成效果。

目录

1 数据集简介

本案例采用的数据集大小为$21.5MB$，包含3447张蝴蝶图像，像素大小一致，都为96×96。数据集来源于《中国蝶类志》，图像中的蝴蝶品种不同，颜色、形态各异，图像背景色皆为白色。

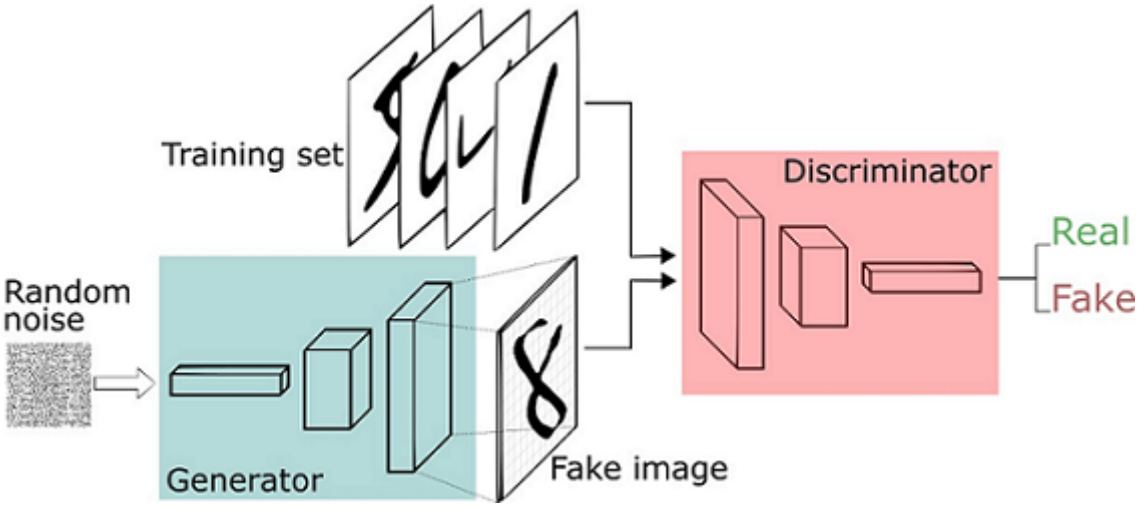


2 模型介绍

本案例采用生成对抗网络（GAN）生成蝴蝶图像。GAN是无监督学习网络，故不需要向训练数据集添加标签。

GAN网络由生成网络（Generator）和判别网络（Discriminator）两部分组成。生成网络尽量生成与训练数据相似的图像；判别网络用以区分生成图像和真实图像，使生成图像得分较低，真实图像得分较高。在训练过程中，GAN迭代优化生成网络和判别网络。

下图展示了GAN网络生成图像的思路。



3 数据处理

接下来加载训练模型需要的数据集，并处理图像数据。首先加载需要使用的库。

```
!pip install jovian --upgrade --quiet
```

```
import os
```

```
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
import torchvision.transforms as tt
import torch
import torch.nn as nn
import cv2
import jovian
from tqdm.notebook import tqdm
import torch.nn.functional as F
from torchvision.utils import save_image
from torchvision.utils import make_grid
from IPython.display import Image
import matplotlib.pyplot as plt
%matplotlib inline
```

读取数据集 butterfly.zip ，使用 ZipFile 函数解压数据集。

```
import zipfile
# 解压zip格式的数据集
zipName = "/content/butterfly.zip"
fileZip = zipfile.ZipFile(zipName)
fileZip.extractall()
fileZip.close()
DATA_DIR = '/content'
```

之后我们给定部分参数的值。因为模型要求图像大小一致，设置变量 image_size ，确保像素大小都为64×64。 batch_size 表示一个批次（batch）训练的图像数量； stats 是归一化函数 tt.Normalize 的参数，将像素值映射至$(-1,1)$的范围内，便于训练判别网络。

```
# 图像大小
image_size = 64
# 一个batch的图像数量
batch_size = 128
# 归一化函数的参数
stats = (0.5, 0.5, 0.5), (0.5, 0.5, 0.5)
```

再使用 ImageFolder 函数加载数据集，并处理图像。设置 DataLoader 函数的参数，在训练模型的时候，可以使数据集分批次输入进模型。

```
# 处理图像数据
train_ds = ImageFolder(DATA_DIR, transform=tt.Compose([
    tt.Resize(image_size),
    tt.CenterCrop(image_size),
    tt.ToTensor(),
```

```
    tt.Normalize(*stats)]))

# 将数据集分批次
train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=3, pin_m
emory=True)
```

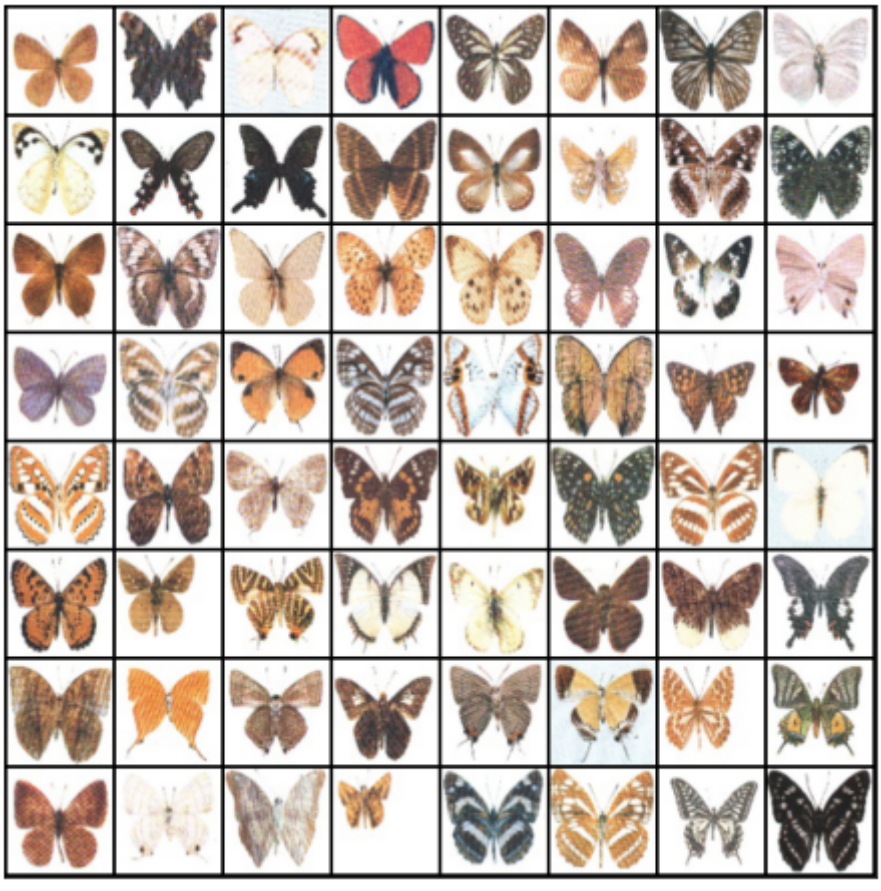
定义函数，从数据集中抽取图像，可以展示数据集中蝴蝶的图像。

```
# 改变图像大小
def denorm(img_tensors):
    return img_tensors * stats[1][0] + stats[0][0]

# 设置图像展示的形式
def show_images(images, nmax=64):
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.imshow(make_grid(denorm(images.detach()[:nmax]), nrow=8).permute(1, 2,
0))

# 抽取图像进行展示
def show_batch(dl, nmax=64):
    for images, _ in dl:
        show_images(images, nmax)
        break
```

```
# 展示图像
show_batch(train_dl)
```



4 构建GAN网络

接下来我们构建GAN网络，分别构建生成网络和判别网络。生成网络生成图像，并传输给判别网络，判别网络检测给定的图像，判断是否是真实图像。

4.1 构建生成网络

首先构建生成网络（Generator）。它能根据随机生成的隐变量，生成图像数据。设定参数 `latent_size` 的值为128。生成网络够将维度为$128 \times 1 \times 1$的数据，转换为$3 \times 64 \times 64$的图像数据。网络结构如下：

```
latent_size = 128
generator = nn.Sequential(
    # 输入数据：latent_size x 1 x 1
    # 反卷积
    nn.ConvTranspose2d(latent_size, 512, kernel_size=4, stride=1, padding=0, b
ias=False),
    # BN层
    nn.BatchNorm2d(512),
    # 激活层 ReLU函数
    nn.ReLU(True),
    # 输出数据：512 x 4 x 4

    nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1, bias=Fals
e),
    nn.BatchNorm2d(256),
    nn.ReLU(True),
    # 输出数据：256 x 8 x 8

    nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1, bias=Fals
e),
    nn.BatchNorm2d(128),
    nn.ReLU(True),
    # 输出数据：128 x 16 x 16

    nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1, bias=Fals
e),
    nn.BatchNorm2d(64),
    nn.ReLU(True),
    # 输出数据：64 x 32 x 32

    nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1, bias=False
),
    nn.Tanh()
    # 输出数据：3 x 64 x 64
)
```

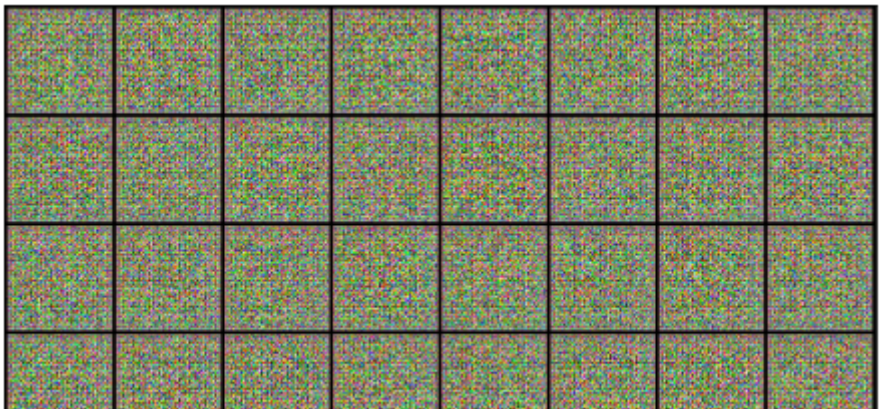
```
# 展示网络结构
generator
```

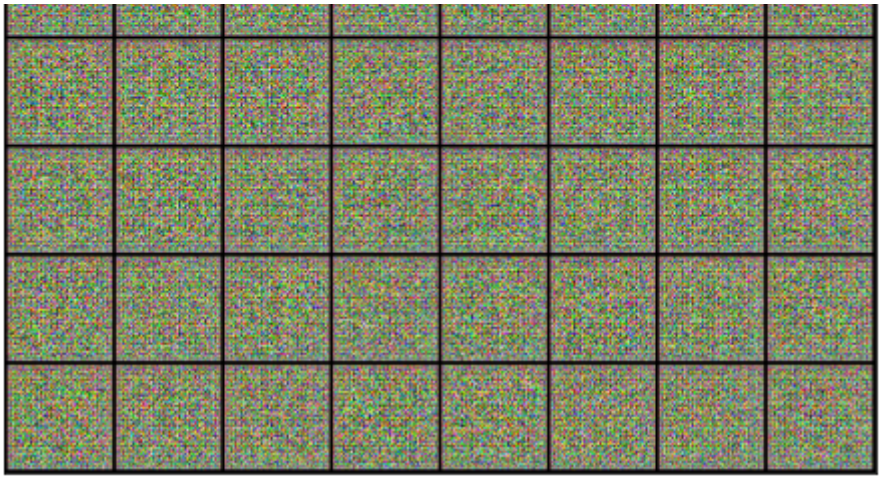
```
Sequential(
  (0): ConvTranspose2d(128, 512, kernel_size=(4, 4), stride=(1, 1), bias=Fals
e)
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
  (2): ReLU(inplace=True)
  (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=
(1, 1), bias=False)
  (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
  (5): ReLU(inplace=True)
  (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=
(1, 1), bias=False)
  (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
  (8): ReLU(inplace=True)
  (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1), bias=False)
  (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
  (11): ReLU(inplace=True)
  (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1), bias=False)
  (13): Tanh()
)
```

随机选取隐变量数据，通过图像展示生成效果。如下图所示，可以看到随机生成的图像没有任何轮廓，表明生成网络需要进行优化。

```
# 随机生成隐变量
xb = torch.randn(batch_size, latent_size, 1, 1)
# 生成图像
fake_images = generator(xb)
# 生成图像的数量和维度
print(fake_images.shape)
show_images(fake_images)
```

```
torch.Size([128, 3, 64, 64])
```





4.2 构建判别网络

下面定义判别网络（Discriminator）。判别网络的输入数据是$3 \times 64 \times 64$的图像数据，对给定的图像进行评分，将其分类为“真实图像”和“生成图像”。网络结构如下：

```
discriminator = nn.Sequential(
    # 输入数据：3 x 64 x 64

    # 卷积层
    nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1, bias=False),
    # BN层
    nn.BatchNorm2d(64),
    # 激活层 LeakyReLU函数
    nn.LeakyReLU(0.2, inplace=True),
    # 输出数据：64 x 32 x 32

    nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(0.2, inplace=True),
    # 输出数据：128 x 16 x 16

    nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.LeakyReLU(0.2, inplace=True),
    # 输出数据：256 x 8 x 8

    nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(512),
    nn.LeakyReLU(0.2, inplace=True),
    # 输出数据：512 x 4 x 4

    nn.Conv2d(512, 1, kernel_size=4, stride=1, padding=0, bias=False),
    # 输出数据：1 x 1 x 1

    # 降至1维数据
    nn.Flatten(),
    # Sigmoid函数对图像评分
    nn.Sigmoid())
```

```
    nn.Sigmoid())
```

```
# 展示网络结构
discriminator
```

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=F
alse)
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
  (2): LeakyReLU(negative_slope=0.2, inplace=True)
  (3): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias
=False)
  (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
  (5): LeakyReLU(negative_slope=0.2, inplace=True)
  (6): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bia
s=False)
  (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
  (8): LeakyReLU(negative_slope=0.2, inplace=True)
  (9): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bia
s=False)
  (10): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
  (11): LeakyReLU(negative_slope=0.2, inplace=True)
  (12): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (13): Flatten()
  (14): Sigmoid()
)
```

5 构建训练模型

在训练过程中，需要迭代优化生成网络和判别网络，所以需要定义优化这二者的函数。首先定义优化判别网络的函数 `train_discriminator` 。

```
def train_discriminator(real_images, opt_d):
    # 梯度值设置为0
    opt_d.zero_grad()

    # 将真实图像输入判别网络
    real_preds = discriminator(real_images)
    real_targets = torch.ones(real_images.size(0), 1, device=device)
    real_loss = F.binary_cross_entropy(real_preds, real_targets)
    real_score = torch.mean(real_preds).item()

    # 通过生成网络得到生成图像
    latent = torch.randn(batch_size, latent_size, 1, 1, device=device)
    fake_images = generator(latent)

    # 将生成图像输入判别网络
    fake_targets = torch.zeros(fake_images.size(0), 1, device=device)
```

```
    fake_targets = torch.zeros(fake_images.size(0), 1, device=device)
    fake_preds = discriminator(fake_images)
    fake_loss = F.binary_cross_entropy(fake_preds, fake_targets)
    fake_score = torch.mean(fake_preds).item()

    # 优化判别网络
    loss = real_loss + fake_loss
    loss.backward()
    opt_d.step()
    return loss.item(), real_score, fake_score
```

接下来定义优化生成网络的函数 `train_generator` 。

```
def train_generator(opt_g):
    # 梯度值设置为0
    opt_g.zero_grad()

    # 生成图像
    latent = torch.randn(batch_size, latent_size, 1, 1, device=device)
    fake_images = generator(latent)

    # 生成图像输入判别网络,得到评分等返回值
    preds = discriminator(fake_images)
    targets = torch.ones(batch_size, 1, device=device)
    loss = F.binary_cross_entropy(preds, targets)

    # 优化生成网络
    loss.backward()
    opt_g.step()

    return loss.item()
```

保存每个epoch的图像生成结果（一个epoch就是将所有训练样本训练一次的过程），可以方便后续查看比较。我们定义函数 `save_samples` ，将图像保存在新建 `generated` 文件夹中。

```
# 为文件夹命名
sample_dir = 'generated'
os.makedirs(sample_dir, exist_ok=True)
# 保存图像生成结果
def save_samples(index, latent_tensors, show=True):
    fake_images = generator(latent_tensors)
    fake_fname = 'generated-images-{0:0=4d}.png'.format(index)
    save_image(denorm(fake_images), os.path.join(sample_dir, fake_fname), nrow
=8)
    print('Saving', fake_fname)
    if show:
        fig, ax = plt.subplots(figsize=(8, 8))
        ax.set_xticks([]); ax.set_yticks([])
```

```
        ax.imshow(make_grid(fake_images.cpu().detach(), nrow=8).permute(1, 2,
0))
```

最后，我们定义训练GAN网络的函数 `fit` ，实现图像生成、保存生成结果，并输出损失值、真实图像和生成图像的得分。

```
def fit(epochs, lr, start_idx=1):
    torch.cuda.empty_cache()

    # 损失和图像得分
    losses_g = []
    losses_d = []
    real_scores = []
    fake_scores = []

    # 构造优化器
    opt_d = torch.optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.
999))
    opt_g = torch.optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999
))

    for epoch in range(epochs):
        for real_images, _ in tqdm(train_dl):
            # 训练判别网络
            loss_d, real_score, fake_score = train_discriminator(real_images,
opt_d)
            # 训练生成网络
            loss_g = train_generator(opt_g)

        # 记录每个epoch的损失和图像得分
        losses_g.append(loss_g)
        losses_d.append(loss_d)
        real_scores.append(real_score)
        fake_scores.append(fake_score)

        # 每训练10个epoch，输出一次损失和图像得分
        if (epoch+1)%10 == 0:
            print("Epoch [{}/{}], loss_g: {:.4f}, loss_d: {:.4f}, real_scor
e: {:.4f}, fake_score: {:.4f}".format(
                    epoch+1, epochs, loss_g, loss_d, real_score, fake_score))

        # 保存生成的图像
        save_samples(epoch+start_idx, fixed_latent, show=False)

    return losses_g, losses_d, real_scores, fake_scores
```

6 模型训练

我们已经定义了所需的函数，接下来可以开始训练模型，生成蝴蝶图像。首先选择合适的设

备，本案例采用CPU训练模型。

```
def get_default_device():
    # 查看可用设备,选择GPU或CPU训练模型
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

# 在合适的设备上加载数据
def to_device(data, device):
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        # 将数据集分为多批次数据
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        # 生成的批次数
        return len(self.dl)
```

```
device = get_default_device()
print(device)
train_dl = DeviceDataLoader(train_dl, device)
```

cuda

```
# 在合适的设备上运行生成网络和判别网络
generator = to_device(generator, device)
discriminator = to_device(discriminator, device)
```

给定学习率 `lr` 和训练所有样本的次数 `epochs` 。

```
lr = 0.0002
epochs = 100
```

现在我们可以调用 `fit` 函数，开始训练模型。

```
# 给定初始的随机生成的隐变量
fixed_latent = torch.randn(64, latent_size, 1, 1, device=device)
# 训练模型
```

```
# 训练模型
history = fit(epochs, lr)
```

```
Epoch [5/100], loss_g: 6.6280, loss_d: 0.0323, real_score: 0.9855, fake_score:
0.0160
Saving generated-images-0005.png
Epoch [10/100], loss_g: 5.8907, loss_d: 0.1141, real_score: 0.9454, fake_scor
e: 0.0510
Saving generated-images-0010.png
Epoch [15/100], loss_g: 0.7017, loss_d: 0.9484, real_score: 0.4817, fake_scor
e: 0.0675
Saving generated-images-0015.png
Epoch [20/100], loss_g: 3.3085, loss_d: 0.4127, real_score: 0.9340, fake_scor
e: 0.2671
Saving generated-images-0020.png
Epoch [25/100], loss_g: 4.2024, loss_d: 0.6067, real_score: 0.8982, fake_scor
e: 0.3656
Saving generated-images-0025.png
Epoch [30/100], loss_g: 2.3837, loss_d: 0.4532, real_score: 0.9059, fake_scor
e: 0.2762
Saving generated-images-0030.png
Epoch [35/100], loss_g: 2.3174, loss_d: 0.8016, real_score: 0.5203, fake_scor
e: 0.0646
Saving generated-images-0035.png
Epoch [40/100], loss_g: 5.8409, loss_d: 1.0619, real_score: 0.9706, fake_scor
e: 0.6147
Saving generated-images-0040.png
Epoch [45/100], loss_g: 5.1178, loss_d: 0.8472, real_score: 0.9219, fake_scor
e: 0.5040
Saving generated-images-0045.png
Epoch [50/100], loss_g: 3.1330, loss_d: 0.4079, real_score: 0.9012, fake_scor
e: 0.2459
Saving generated-images-0050.png
Epoch [55/100], loss_g: 4.0297, loss_d: 0.6759, real_score: 0.9377, fake_scor
e: 0.4278
Saving generated-images-0055.png
Epoch [60/100], loss_g: 2.8585, loss_d: 0.4275, real_score: 0.7514, fake_scor
e: 0.1045
Saving generated-images-0060.png
Epoch [65/100], loss_g: 0.9700, loss_d: 0.8861, real_score: 0.5197, fake_scor
e: 0.0981
Saving generated-images-0065.png
Epoch [70/100], loss_g: 1.6326, loss_d: 0.5004, real_score: 0.6564, fake_scor
e: 0.0302
Saving generated-images-0070.png
Epoch [75/100], loss_g: 2.4796, loss_d: 0.4049, real_score: 0.9590, fake_scor
e: 0.2787
Saving generated-images-0075.png
Epoch [80/100], loss_g: 1.5469, loss_d: 0.5539, real_score: 0.6389, fake_scor
e: 0.0523
Saving generated-images-0080.png
Epoch [85/100], loss_g: 2.1258, loss_d: 0.5414, real_score: 0.6329, fake_scor
e: 0.0165
Saving generated-images-0085.png
Epoch [90/100], loss_g: 3.0037, loss_d: 0.2394, real_score: 0.9507, fake_scor
e: 0.1617
```

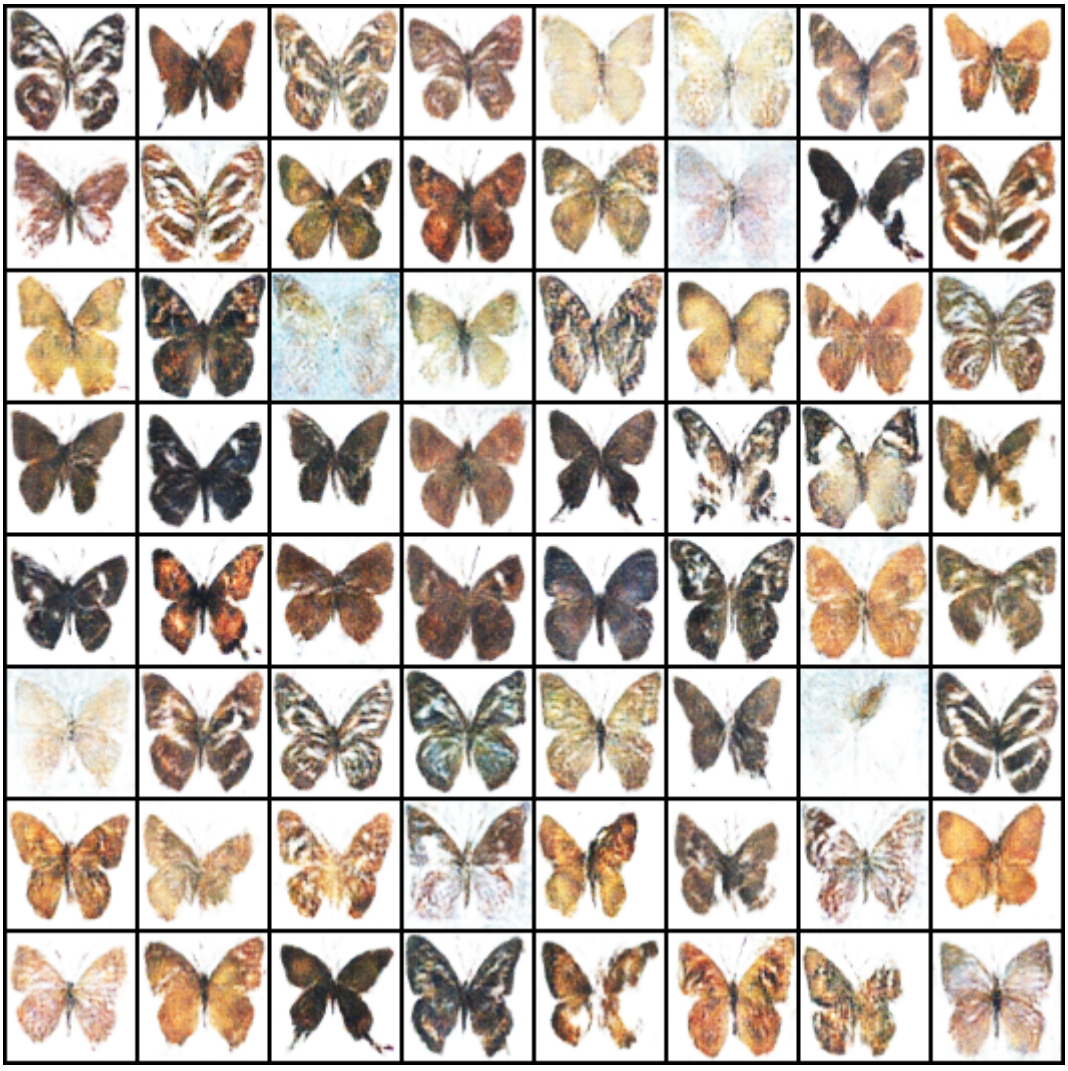
```
Saving generated-images-0090.png
Epoch [95/100], loss_g: 4.6037, loss_d: 0.6921, real_score: 0.9819, fake_scor
e: 0.4330
Saving generated-images-0095.png
Epoch [100/100], loss_g: 3.0478, loss_d: 0.3417, real_score: 0.9158, fake_scor
e: 0.2059
Saving generated-images-0100.png
```

7 模型效果评估

用图像方式，可以更直观地展示蝴蝶图像生成效果。

下图为经过100次epoch后，生成的蝴蝶图像。可以看到，除了少数图像仍需要进一步的优化，大部分生成的图像已经与真实图像十分接近。

```
Image('/content/generated/generated-images-0100.png')
```

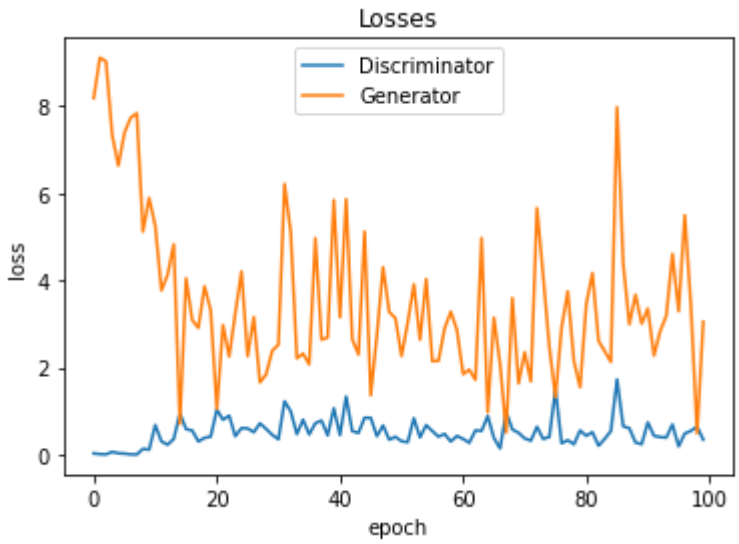


接下来，可以读取损失值、图像得分的数据，以分析训练过程。

```
losses_g, losses_d, real_scores, fake_scores = history
```

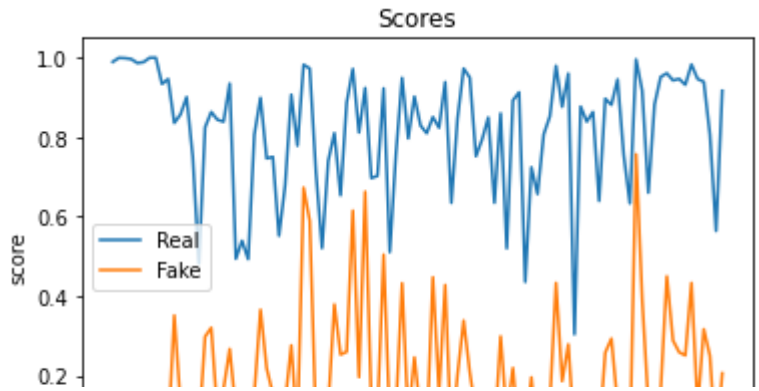
下图表明随着训练次数的增加，生成网络和判别网络损失值的变化。可以发现，在前20epoch，生成网络的损失值下降较快，后续损失值的变化波动较大。

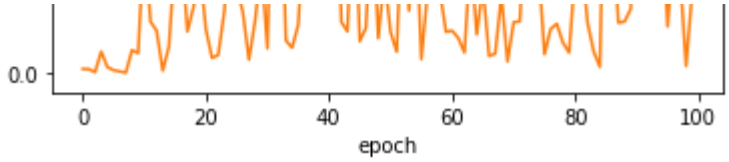
```
plt.plot(losses_d, '-')
plt.plot(losses_g, '-')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['Discriminator', 'Generator'])
plt.title('Losses');
```



下图表明在判别网络中，生成图像和真实图像的得分情况。同样，在前20epoch，生成图像的得分值增加较快，后续分值波动情况较大，但模型向着生成图像得分增加的方向，不断优化。

```
plt.plot(real_scores, '-')
plt.plot(fake_scores, '-')
plt.xlabel('epoch')
plt.ylabel('score')
plt.legend(['Real', 'Fake'])
plt.title('Scores');
```





8 总结

本案例介绍了如何使用GAN网络进行图像生成，通过生成的蝴蝶图像与原数据集对比，可证明在经过多次优化训练后，模型的图像生成效果较好。

发表您的讨论