



Введение в искусственные нейронные сети

GAN

На этом уроке

1. Познакомимся с генеративно-состязательными сетями
2. Изучим их архитектуру
3. Попрактикуемся в обучении сети

Оглавление

[На этом уроке](#)

[Оглавление](#)

[Что такое GAN](#)

[Общие сведения об архитектуре GAN](#)

[Практика](#)

[Подключение необходимых библиотек](#)

[Предварительная обработка данных](#)

[Импортирование Pix2Pix модели](#)

[Loss functions](#)

[Checkpoints](#)

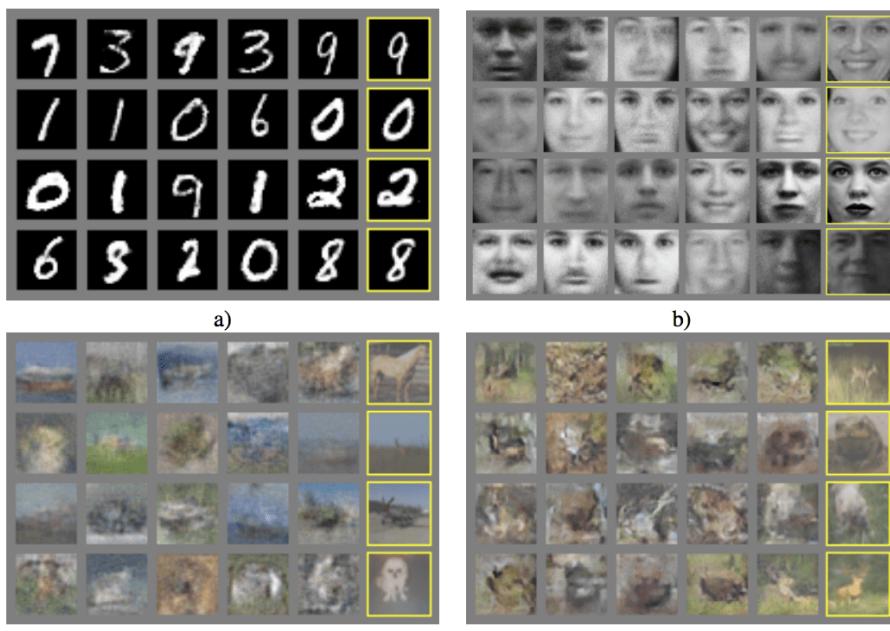
[Training](#)

[Generate using test dataset](#)

[Используемые источники](#)

Что такое GAN

GAN, Generative adversarial network (Генеративно-состязательная сеть), появились в 2014 г. Их автор — Ian Goodfellow. Обычно они применяются для генерации и изменения стилей изображений.



Такие нейронные сети примечательны сразу несколькими фактами:

- Они появились уже после революции нейронных сетей в 2012 г.
- Они не делают заключение об уже имеющихся данных, а генерируют новые
- В ряде популярных архитектур GAN используется обучение без учителя

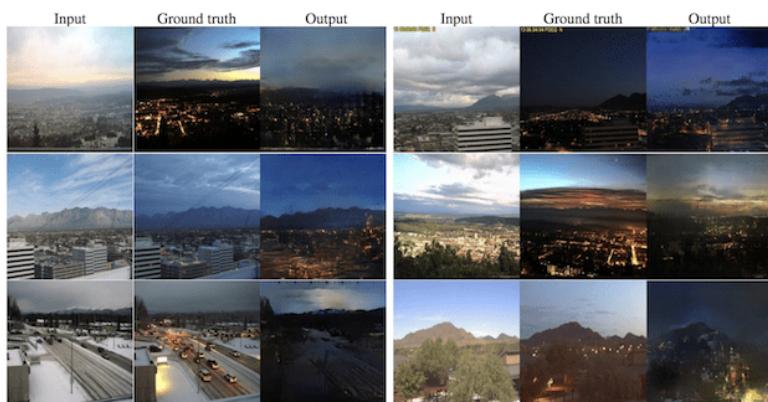
После своего появления эти нейронные сети стремительно прогрессируют. Если первые могли генерировать небольшие черно-белые изображения цифр и лиц, то сейчас они могут создавать реалистичные фотографии несуществующих людей, а также иные изображения в хорошем качестве.



Среди разновидностей GAN есть следующие:

- Генерируют новые изображения
- Переносят стиль изображения
- Создают изображения по текстовому описанию

- Рисуют изображения по эскизу
- Улучшают качество изображения
- И др.



this small bird has a pink breast and crown, and black primaries and secondaries.



the flower has petals that are bright pinkish purple with white stigma



this magnificent fellow is almost all black with a red crest, and white cheek patch.



this white and yellow flower have thin white petals and a round yellow stamen



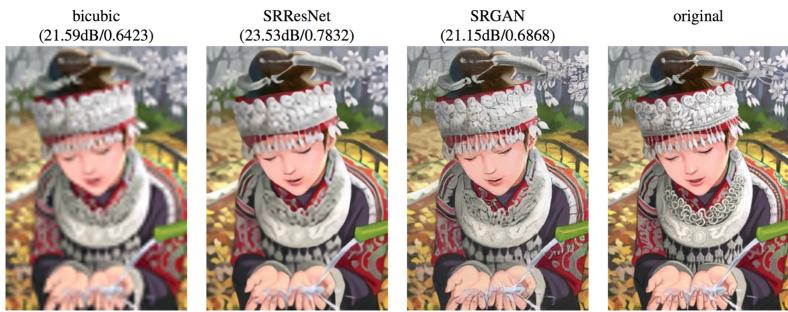
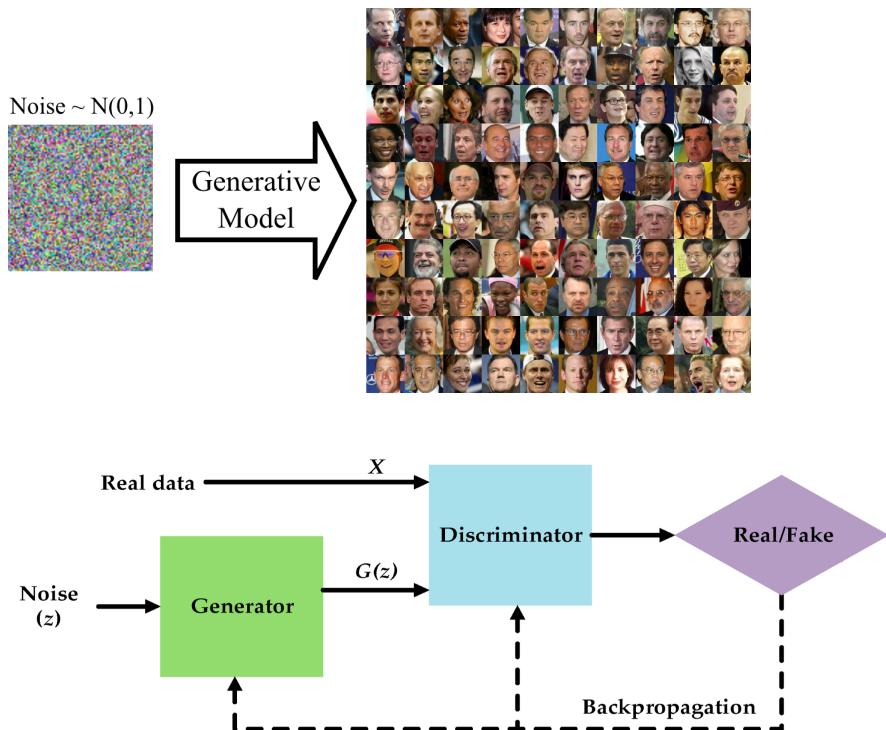


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

Генеративно-состязательные сети генерируют не только изображения, но и другие объекты, например молекулы (в медицинских целях).

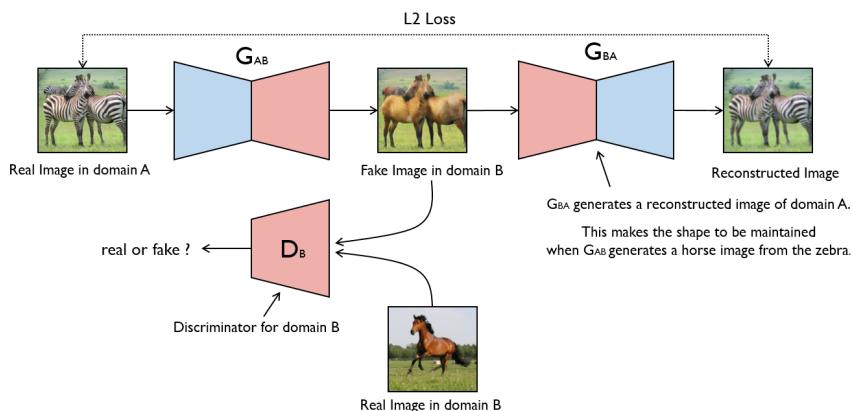
Общие сведения об архитектуре GAN

Архитектура GAN состоит из двух нейронных сетей. Одна является генератором, другая — дискриминатором. В задачу генератора входит генерация изображения из подающегося ему на вход шума. Задача дискриминатора заключается в сравнении сгенерированного изображения с настоящим. Одна нейронная сеть учится заставлять путать настоящее изображение и сгенерированное, а другая — находить настоящее. В качестве генератора может выступать полно связная нейронная сеть, в качестве дискриминатора обычно выступает свёрточная.



Существуют разные архитектуры GAN, такие, как CGAN, DCGAN, ProgressiveGAN, BigGAN, StyleBaseGan. В этом уроке мы рассмотрим архитектуру CycleGAN.

Она появилась в 2017 г. и была призвана генерировать объекты с определённым стилем при условии, что в датасете нет изображения, с которым можно было бы сравнить результат. Данная нейронная сеть сначала генерирует изображение с определённым стилем, а затем возвращает его к обратному состоянию. Таким образом, мы получаем возможность контролировать результат обучения. В этой архитектуре используются сразу 2 GAN: одна проверяет результат переноса стиля, а другая — возврат к исходному изображению.



Практика

В качестве практики попробуем обучить нейронную сеть превращать лошадей в зебр. Для этого нам понадобится архитектура CycleGAN и датасет horse2zebra. Для работы данного исходного кода необходим tensorflow 2.1.0.

Copyright 2019 The TensorFlow Authors.

```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
#
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#
# See the License for the specific language governing permissions and
# limitations under the License.
```

Подключение необходимых библиотек

```
!pip install -q git+https://github.com/tensorflow/examples.git

import tensorflow as tf

tf.config.experimental.set_visible_devices([], 'GPU')

import tensorflow_datasets as tfds

from tensorflow_examples.models.pix2pix import pix2pix

import os

import time

import matplotlib.pyplot as plt

from IPython.display import clear_output

tfds.disable_progress_bar()

AUTOTUNE = tf.data.experimental.AUTOTUNE
```

Предварительная обработка данных

Ссылка на датасет [здесь](#). Необходимо конвертировать изображения в 286x286 и случайно выбранные из них обрезать до 256x256. Помимо этого, перевернём изображения горизонтально, т.е. слева направо. Таким образом, мы проведём процедуру, похожую на image augmentation.

```
dataset, metadata = tfds.load('cycle_gan/horse2zebra',
                             with_info=True, as_supervised=True)

train_horses, train_zebras = dataset['trainA'], dataset['trainB']

test_horses, test_zebras = dataset['testA'], dataset['testB']

BUFFER_SIZE = 1000

BATCH_SIZE = 1

IMG_WIDTH = 256

IMG_HEIGHT = 256

def random_crop(image):

    cropped_image = tf.image.random_crop(
        image, size=[IMG_HEIGHT, IMG_WIDTH, 3])

    return cropped_image

# normalizing the images to [-1, 1]

def normalize(image):

    image = tf.cast(image, tf.float32)
```

```

image = (image / 127.5) - 1

return image

def random_jitter(image):

    # resizing to 286 x 286 x 3

    image = tf.image.resize(image, [286, 286],
                           method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)

    # randomly cropping to 256 x 256 x 3

    image = random_crop(image)

    # random mirroring

    image = tf.image.random_flip_left_right(image)

    return image

def preprocess_image_train(image, label):

    image = random_jitter(image)

    image = normalize(image)

    return image

def preprocess_image_test(image, label):

    image = normalize(image)

    return image

train_horses = train_horses.map(
    preprocess_image_train, num_parallel_calls=AUTOTUNE).cache().shuffle(
    BUFFER_SIZE).batch(1)

train_zebras = train_zebras.map(
    preprocess_image_train, num_parallel_calls=AUTOTUNE).cache().shuffle(
    BUFFER_SIZE).batch(1)

test_horses = test_horses.map(
    preprocess_image_test, num_parallel_calls=AUTOTUNE).cache().shuffle(
    BUFFER_SIZE).batch(1)

test_zebras = test_zebras.map(
    preprocess_image_test, num_parallel_calls=AUTOTUNE).cache().shuffle(
    BUFFER_SIZE).batch(1)

sample_horse = next(iter(train_horses))

sample_zebra = next(iter(train_zebras))

```

```

plt.subplot(121)
plt.title('Horse')
plt.imshow(sample_horse[0] * 0.5 + 0.5)

plt.subplot(122)
plt.title('Horse with random jitter')
plt.imshow(random_jitter(sample_horse[0]) * 0.5 + 0.5)

plt.subplot(121)
plt.title('Zebra')
plt.imshow(sample_zebra[0] * 0.5 + 0.5)

plt.subplot(122)
plt.title('Zebra with random jitter')
plt.imshow(random_jitter(sample_zebra[0]) * 0.5 + 0.5)

```

Импортирование Pix2Pix модели

Генератор и дискриминатор возьмём из Pix2Pix модели, генерация будет осуществляться с применением Unet.

```

OUTPUT_CHANNELS = 3

generator_g = pix2pix.unet_generator(OUTPUT_CHANNELS, norm_type='instancenorm')
generator_f = pix2pix.unet_generator(OUTPUT_CHANNELS, norm_type='instancenorm')

discriminator_x = pix2pix.discriminator(norm_type='instancenorm', target=False)
discriminator_y = pix2pix.discriminator(norm_type='instancenorm', target=False)

to_zebra = generator_g(sample_horse)
to_horse = generator_f(sample_zebra)

plt.figure(figsize=(8, 8))
contrast = 8

imgs = [sample_horse, to_zebra, sample_zebra, to_horse]

title = ['Horse', 'To Zebra', 'Zebra', 'To Horse']

for i in range(len(imgs)):
    plt.subplot(2, 2, i+1)
    plt.title(title[i])

    if i % 2 == 0:

```

```

plt.imshow(imgs[i][0] * 0.5 + 0.5)

else:

    plt.imshow(imgs[i][0] * 0.5 * contrast + 0.5)

plt.show()

plt.figure(figsize=(8, 8))

plt.subplot(121)

plt.title('Is a real zebra?')

plt.imshow(discriminator_y(sample_zebra)[0, ..., -1], cmap='RdBu_r')

plt.subplot(122)

plt.title('Is a real horse?')

plt.imshow(discriminator_x(sample_horse)[0, ..., -1], cmap='RdBu_r')

plt.show()

```

Loss functions

Loss функции для генератора и дискриминатора можно также взять из [pix2pix](#).

```

LAMBDA = 10

loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real, generated):

    real_loss = loss_obj(tf.ones_like(real), real)

    generated_loss = loss_obj(tf.zeros_like(generated), generated)

    total_disc_loss = real_loss + generated_loss

    return total_disc_loss * 0.5

def generator_loss(generated):

    return loss_obj(tf.ones_like(generated), generated)

def calc_cycle_loss(real_image, cycled_image):

    loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))

    return LAMBDA * loss1

def identity_loss(real_image, same_image):

    loss = tf.reduce_mean(tf.abs(real_image - same_image))

    return LAMBDA * 0.5 * loss

```

Инициализация оптимайзеров для всех генераторов и дискриминаторов.

```
generator_g_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
generator_f_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

discriminator_x_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_y_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
```

Checkpoints

Сохранение промежуточных результатов для того, чтобы при необходимости можно было продолжить обучение, а не начинать сначала.

```
checkpoint_path = "./checkpoints/train"

ckpt = tf.train.Checkpoint(generator_g=generator_g,
                           generator_f=generator_f,
                           discriminator_x=discriminator_x,
                           discriminator_y=discriminator_y,
                           generator_g_optimizer=generator_g_optimizer,
                           generator_f_optimizer=generator_f_optimizer,
                           discriminator_x_optimizer=discriminator_x_optimizer,
                           discriminator_y_optimizer=discriminator_y_optimizer)

ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)

# if a checkpoint exists, restore the latest checkpoint.
if ckpt_manager.latest_checkpoint:
    ckpt.restore(ckpt_manager.latest_checkpoint)
    print ('Latest checkpoint restored!!!')
```

Training

По умолчанию количество эпох выставлено 1, хотя для корректных результатов понадобится от нескольких десятков до нескольких сотен.

```
EPOCHS = 1
```

```

def generate_images(model, test_input):

    prediction = model(test_input)

    plt.figure(figsize=(12, 12))

    display_list = [test_input[0], prediction[0]]

    title = ['Input Image', 'Predicted Image']

    for i in range(2):
        plt.subplot(1, 2, i+1)
        plt.title(title[i])
        # getting the pixel values between [0, 1] to plot it.
        plt.imshow(display_list[i] * 0.5 + 0.5)
        plt.axis('off')
    plt.show()

```

Несмотря на то, что тренировочный процесс у GAN более сложный, он состоит из тех же этапов, что обычно:

- Получить предсказание
- Вычислить ошибку
- Посчитать градиенты, используя обратное распространение ошибки
- Применить градиенты для оптимайзера

```

@tf.function

def train_step(real_x, real_y):
    # persistent is set to True because the tape is used more than
    # once to calculate the gradients.

    with tf.GradientTape(persistent=True) as tape:
        # Generator G translates X -> Y
        # Generator F translates Y -> X.

        fake_y = generator_g(real_x, training=True)
        cycled_x = generator_f(fake_y, training=True)

        fake_x = generator_f(real_y, training=True)
        cycled_y = generator_g(fake_x, training=True)

        # same_x and same_y are used for identity loss.

```

```

same_x = generator_f(real_x, training=True)

same_y = generator_g(real_y, training=True)

disc_real_x = discriminator_x(real_x, training=True)
disc_real_y = discriminator_y(real_y, training=True)

disc_fake_x = discriminator_x(fake_x, training=True)
disc_fake_y = discriminator_y(fake_y, training=True)

# calculate the loss

gen_g_loss = generator_loss(disc_fake_y)
gen_f_loss = generator_loss(disc_fake_x)

total_cycle_loss = calc_cycle_loss(real_x, cycled_x) + calc_cycle_loss(real_y, cycled_y)

# Total generator loss = adversarial loss + cycle loss

total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_y, same_y)
total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_x, same_x)

disc_x_loss = discriminator_loss(disc_real_x, disc_fake_x)
disc_y_loss = discriminator_loss(disc_real_y, disc_fake_y)

# Calculate the gradients for generator and discriminator

generator_g_gradients = tape.gradient(total_gen_g_loss,
                                       generator_g.trainable_variables)

generator_f_gradients = tape.gradient(total_gen_f_loss,
                                       generator_f.trainable_variables)

discriminator_x_gradients = tape.gradient(disc_x_loss,
                                           discriminator_x.trainable_variables)

discriminator_y_gradients = tape.gradient(disc_y_loss,
                                           discriminator_y.trainable_variables)

# Apply the gradients to the optimizer

generator_g_optimizer.apply_gradients(zip(generator_g_gradients,
                                          generator_g.trainable_variables))

generator_f_optimizer.apply_gradients(zip(generator_f_gradients,
                                          generator_f.trainable_variables))

```

```

        generator_f.trainable_variables))

discriminator_x_optimizer.apply_gradients(zip(discriminator_x_gradients,
                                             discriminator_x.trainable_variables))

discriminator_y_optimizer.apply_gradients(zip(discriminator_y_gradients,
                                             discriminator_y.trainable_variables))

for epoch in range(EPOCHS):

    start = time.time()

    n = 0

    for image_x, image_y in tf.data.Dataset.zip((train_horses, train_zebras)):

        train_step(image_x, image_y)

        if n % 10 == 0:

            print ('.', end='')

        n+=1

    clear_output(wait=True)

# Using a consistent image (sample_horse) so that the progress of the model
# is clearly visible.

generate_images(generator_g, sample_horse)

if (epoch + 1) % 5 == 0:

    ckpt_save_path = ckpt_manager.save()

    print ('Saving checkpoint for epoch {} at {}'.format(epoch+1,
                                                          ckpt_save_path))

print ('Time taken for epoch {} is {} sec\n'.format(epoch + 1,
                                                    time.time()-start))

```

Generate using test dataset

```

# Run the trained model on the test dataset

for inp in test_horses.take(5):

    generate_images(generator_g, inp)

```

Используемые источники

1. <https://www.tensorflow.org/tutorials/generative/cyclegan>
2. Generative Adversarial Networks: An Overview. Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, Anil A Bharath. 2017
3. A Survey and Taxonomy of Adversarial Neural Networks for Text-to-Image Synthesis. Jorge Agnese, Jonathan Herrera, Haicheng Tao, Xingquan Zhu. 2019