```
import tensorflow as tf

from __future__ import absolute_import, division, print_function, unicode_literals

import numpy as np

import tensorflow_datasets as tfds
tfds.disable_progress_bar()

from IPython.display import clear_output
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
tf.config.experimental.set_visible_devices([], 'GPU')
```

```
!pip install shap
```

```
Collecting shap
  Downloading shap-0.40.0-cp37-cp37m-manylinux2010_x86_64.whl (564 kB)
     |████████████████████████████████| 564 kB 13.2 MB/s
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Collecting slicer==0.0.7
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/d
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in /usr/local/lib/python3
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Installing collected packages: slicer, shap
Successfully installed shap-0.40.0 slicer-0.0.7
```

```
ds_names_seg = ('amazon_us_reviews','caltech_birds2010', 'oxford_iiit_pet:3.*.*', 'caltech

# читаем данные (можно не все сразу split? обязательно и информацией with_info=True,
# имя набора срисовываем из https://www.tensorflow.org/datasets/catalog/overview)
# объект датасет https://www.tensorflow.org/api_docs/python/tf/data/Dataset

dataset, info = tfds.load(ds_names_seg[2],split =['train[:10%]'] ,with_info=True)
```

```
Downloading and preparing dataset oxford_iiit_pet/3.2.0 (download: 773.52 MiB, genera
Shuffling and writing examples to /root/tensorflow_datasets/oxford_iiit_pet/3.2.0.inc
```

```
Shuffling and writing examples to /root/tensorflow_datasets/oxford_iiit_pet/3.2.0.inc
Dataset oxford_iiit_pet downloaded and prepared to /root/tensorflow_datasets/oxford_i
```

```
info
```

```
tfds.core.DatasetInfo(
    name='oxford_iiit_pet',
    version=3.2.0,
    description='The Oxford-IIIT pet dataset is a 37 category pet image dataset with
images for each class. The images have large variations in scale, pose and
lighting. All images have an associated ground truth annotation of breed.',
    homepage='http://www.robots.ox.ac.uk/~vgg/data/pets/',
    features=FeaturesDict({
        'file_name': Text(shape=(), dtype=tf.string),
        'image': Image(shape=(None, None, 3), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=37),
        'segmentation_mask': Image(shape=(None, None, 1), dtype=tf.uint8),
        'species': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),
    }),
    total_num_examples=7349,
    splits={
        'test': 3669,
        'train': 3680,
    },
    supervised_keys=('image', 'label'),
    citation="""@InProceedings{parkhi12a,
      author       = "Parkhi, O. M. and Vedaldi, A. and Zisserman, A. and Jawahar, C
      title        = "Cats and Dogs",
      booktitle    = "IEEE Conference on Computer Vision and Pattern Recognition",
      year         = "2012",
    }""",
    redistribution_info=,
)
```

Обработка для примеров датасета зависит от состава данных

```
def normalize(input_image, input_mask):
  input_image = tf.cast(input_image, tf.float32) / 255.0
  #input_mask -= 1
  return input_image, input_mask


def load_image_train(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

  if tf.random.uniform(()) > 0.5:
    input_image = tf.image.flip_left_right(input_image)
    input_mask = tf.image.flip_left_right(input_mask)

  input_image, input_mask = normalize(input_image, input_mask)

  return input_image, input_mask
```

```
def load_image_test(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

  input_image, input_mask = normalize(input_image, input_mask)

  return input_image, input_mask
```

можно использовать результаты анализа для формирования задачи обработки

```
TRAIN_LENGTH = info.splits['train'].num_examples
BATCH_SIZE = 16
BUFFER_SIZE = 128
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
```

можно брать данные частями (тут только первая часть набора, который прочли (их может быть больше, может быть test, validation и т.д. смотрим по info))

```
train = dataset[0].map(load_image_train, num_parallel_calls=tf.data.experimental.AUTOTUNE)
#test = dataset[1].map(load_image_test)
```

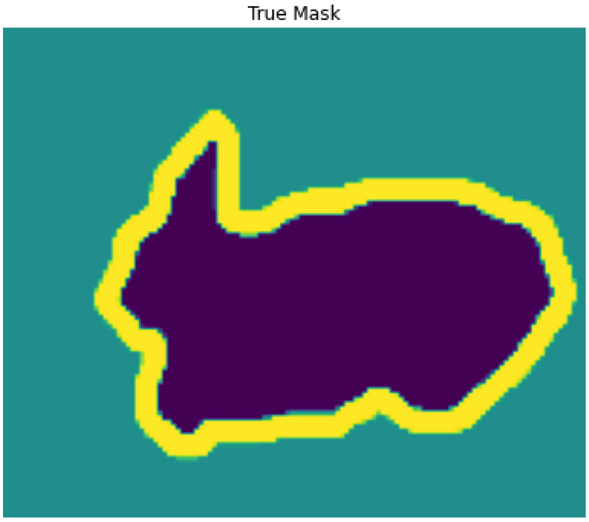
отрисуем

```
def display(display_list):
  plt.figure(figsize=(15, 15))

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

  for i in range(len(display_list)):
    plt.subplot(1, len(display_list), i+1)
    plt.title(title[i])
    plt.imshow(tf.keras.preprocessing.image.array_to_img(display_list[i]))
    plt.axis('off')
  plt.show()
```

```
for image, mask in train.take(1):
  sample_image, sample_mask = image, mask
display([sample_image, sample_mask])
```


Проверим результат обработки меток

```
sample_mask.numpy().min(),sample_mask.numpy().max()
```

```
(1.0, 3.0)
```

# GAN `Model

## Загрузка модулей

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
```

## строим mnist data

```
# mnist
```

```
class_ = 0
batch_size = 64
(x_train, y), (x_test, yt) = keras.datasets. mnist.load_data()
all_digits = np.concatenate([x_train, x_test])
all_digits = all_digits.astype("float32") / 255
ind_i = np.where(y == class_)
ind_it = np.where(yt == class_)
all_digits = all_digits[ind_i,:,:]
all_digits = np.reshape(all_digits, (-1, 28, 28, 1))
dataset = tf.data.Dataset.from_tensor_slices(all_digits)
dataset = dataset.shuffle(buffer_size=1024).batch(batch_size).prefetch(32)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mni
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

Строим discriminator размер карты 28x28 и бинарная классификация (настоящее изображение или генерировано).

```
discriminator = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(64, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(128, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.GlobalMaxPooling2D(),
        layers.Dense(1),
    ],
    name="discriminator",
)

discriminator.summary()
```

```
Model: "discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 64)        640

 leaky_re_lu (LeakyReLU)     (None, 14, 14, 64)        0

 conv2d_1 (Conv2D)           (None, 7, 7, 128)         73856

 leaky_re_lu_1 (LeakyReLU)   (None, 7, 7, 128)         0

 global_max_pooling2d (Globa  (None, 128)              0
 lMaxPooling2D)

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 74,625
```

```
Trainable params: 74,625
Non-trainable params: 0
_________________________________________________________________
```

## Строим generator

обратное по отношению к дискриминатору преобразование, меняем Conv2D на Conv2DTranspose .

```
latent_dim = 128

generator = keras.Sequential(
    [
        keras.Input(shape=(latent_dim,)),
        # строим размер входного вектора 8x8x128 map
        layers.Dense(7 * 7 * 128),
        layers.LeakyReLU(alpha=0.2),
        layers.Reshape((7, 7, 128)),
        layers.Conv2DTranspose(128, (4, 4), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(256, (4, 4), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(1, (7, 7), padding="same", activation="sigmoid"),
    ],
    name="generator",
)

generator.summary()
```

```
Model: "generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 6272)              809088

 leaky_re_lu_2 (LeakyReLU)   (None, 6272)              0

 reshape (Reshape)           (None, 7, 7, 128)         0

 conv2d_transpose (Conv2DTra  (None, 14, 14, 128)      262272
 nspose)

 leaky_re_lu_3 (LeakyReLU)   (None, 14, 14, 128)       0

 conv2d_transpose_1 (Conv2DT  (None, 28, 28, 256)      524544
 ranspose)

 leaky_re_lu_4 (LeakyReLU)   (None, 28, 28, 256)       0

 conv2d_2 (Conv2D)           (None, 28, 28, 1)         12545

=================================================================
Total params: 1,608,449
Trainable params: 1,608,449
```

```
Non-trainable params: 0
```

# Класс со своим этапом обучения train_step

```
class GAN(keras.Model):
    def __init__(self, discriminator, generator, latent_dim):
        super(GAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.latent_dim = latent_dim

    def compile(self, d_optimizer, g_optimizer, loss_fn):
        super(GAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.loss_fn = loss_fn

    def train_step(self, real_images):
        if isinstance(real_images, tuple):
            real_images = real_images[0]
        # берем случайный пример из скрытого пространства
        batch_size = tf.shape(real_images)[0]
        random_latent_vectors = tf.random.normal(shape=(batch_size, self.latent_dim))

        # Строим по нему фейковое изображение
        generated_images = self.generator(random_latent_vectors)

        # собрали с реальным в текзор
        combined_images = tf.concat([generated_images, real_images], axis=0)

        # задаем метки 1 и 0 соответственно
        labels = tf.concat(
            [tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))], axis=0
        )
        # Добавляем шум !!!
        labels += 0.05 * tf.random.uniform(tf.shape(labels))

        # учим discriminator
        with tf.GradientTape() as tape:
            predictions = self.discriminator(combined_images)
            d_loss = self.loss_fn(labels, predictions)
        grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights)
        )

        #Выбрали случайный пример в скрытом пространстве
        random_latent_vectors = tf.random.normal(shape=(batch_size, self.latent_dim))

        # собрали метки реальных изображений
        misleading_labels = tf.zeros((batch_size, 1))
```

```
        # Учим generator !
        with tf.GradientTape() as tape:
            predictions = self.discriminator(self.generator(random_latent_vectors))
            g_loss = self.loss_fn(misleading_labels, predictions)
        grads = tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(zip(grads, self.generator.trainable_weights))
        return {"d_loss": d_loss, "g_loss": g_loss}
```

## Callback для сохранения изображений по ходу обучения

```
class GANMonitor(keras.callbacks.Callback):
    def __init__(self, num_img=3, latent_dim=128):
        self.num_img = num_img
        self.latent_dim = latent_dim

    def on_epoch_end(self, epoch, logs=None):
        random_latent_vectors = tf.random.normal(shape=(self.num_img, self.latent_dim))
        generated_images = self.model.generator(random_latent_vectors)
        generated_images *= 255
        generated_images.numpy()
        for i in range(self.num_img):
            img = keras.preprocessing.image.array_to_img(generated_images[i])
            img.save("generated_img_{i}_{epoch}.png".format(i=i, epoch=epoch))
```

## Учим end-to-end модель

```
epochs = 50

gan = GAN(discriminator=discriminator, generator=generator, latent_dim=latent_dim)
gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.0003),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.0003),
    loss_fn=keras.losses.BinaryCrossentropy(from_logits=True),
)

gan.fit(
    dataset, epochs=epochs, callbacks=[GANMonitor(num_img=3, latent_dim=latent_dim)]
)
```

```
Epoch 1/50
93/93 [==============================] - 271s 3s/step - d_loss: 0.4114 - g_loss:
Epoch 2/50
93/93 [==============================] - 271s 3s/step - d_loss: 0.0948 - g_loss:
Epoch 3/50
93/93 [==============================] - 269s 3s/step - d_loss: 0.0013 - g_loss:
Epoch 4/50
```

```
93/93 [==============================] - 271s 3s/step - d_loss: -0.0523 - g_loss:
Epoch 5/50
93/93 [==============================] - 269s 3s/step - d_loss: -0.1538 - g_loss:
Epoch 6/50
93/93 [==============================] - 270s 3s/step - d_loss: -0.0594 - g_loss:
Epoch 7/50
93/93 [==============================] - 270s 3s/step - d_loss: 0.3141 - g_loss:
Epoch 8/50
93/93 [==============================] - 269s 3s/step - d_loss: 0.6420 - g_loss:
Epoch 9/50
93/93 [==============================] - 269s 3s/step - d_loss: 0.6221 - g_loss:
Epoch 10/50
93/93 [==============================] - 269s 3s/step - d_loss: 0.5549 - g_loss:
Epoch 11/50
93/93 [==============================] - 270s 3s/step - d_loss: 0.5439 - g_loss:
Epoch 12/50
93/93 [==============================] - 270s 3s/step - d_loss: 0.4779 - g_loss:
Epoch 13/50
93/93 [==============================] - 270s 3s/step - d_loss: 0.4665 - g_loss:
Epoch 14/50
93/93 [==============================] - 269s 3s/step - d_loss: 0.4833 - g_loss:
Epoch 15/50
93/93 [==============================] - 269s 3s/step - d_loss: 0.4787 - g_loss:
Epoch 16/50
93/93 [==============================] - 270s 3s/step - d_loss: 0.5518 - g_loss:
Epoch 17/50
93/93 [==============================] - 269s 3s/step - d_loss: 0.5087 - g_loss:
Epoch 18/50
93/93 [==============================] - 270s 3s/step - d_loss: 0.5486 - g_loss:
Epoch 19/50
93/93 [==============================] - 270s 3s/step - d_loss: 0.4854 - g_loss:
Epoch 20/50
93/93 [==============================] - 270s 3s/step - d_loss: 0.5732 - g_loss:
Epoch 21/50
93/93 [==============================] - 272s 3s/step - d_loss: 0.5081 - g_loss:
Epoch 22/50
93/93 [==============================] - 272s 3s/step - d_loss: 0.5108 - g_loss:
Epoch 23/50
93/93 [==============================] - 272s 3s/step - d_loss: 0.4132 - g_loss:
Epoch 24/50
93/93 [==============================] - 271s 3s/step - d_loss: 0.5141 - g_loss:
Epoch 25/50
93/93 [==============================] - 272s 3s/step - d_loss: 0.4319 - g_loss:
Epoch 26/50
93/93 [==============================] - 271s 3s/step - d_loss: 0.5680 - g_loss:
Epoch 27/50
93/93 [==============================] - 272s 3s/step - d_loss: 0.4894 - g_loss:
Epoch 28/50
93/93 [==============================] - 272s 3s/step - d_loss: 0.5306 - g_loss:
```

## Отображение последних сгенерированных изображений:

```
from IPython.display import Image, display
```

```
for i in range(1,5):
  display(Image("generated_img_0_"+str(i)+"9.png"))
```

```
i = 3
display(Image("generated_img_0_"+str(i)+"8.png"))
```