Практическое задание

Попробуйте обучить нейронную сеть U-Net на любом другом датасете.

Опишите результата. Что помогло повысить точность?

*Попробуйте свои силы в задаче Carvana на Kaggle - https://www.kaggle.com/c/carvana-image-masking-challenge/overview4.

*Сделайте свою реализацию U-Net на TensorFlow

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

```
Collecting git+https://github.com/tensorflow/examples.git
  Cloning https://github.com/tensorflow/examples.git to /tmp/pip-req-build-vid46t2f
  Running command git clone -q https://github.com/tensorflow/examples.git /tmp/pip-re
Requirement already satisfied: absl-py in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from te
```

```
try:
  %tensorflow_version only exists in Colab.
  #%tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf
```

```
Unknown TensorFlow version: only exists in Colab.
Currently selected TF version: 2.x
Available versions:
 * 1.x
 * 2.x
```

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, Model

import tensorflow.compat.v2 as tf
from __future__ import absolute_import, division, print_function, unicode_literals
from tensorflow.keras.layers import Conv2D, Input
from tensorflow_examples.models.pix2pix import pix2pix
from tensorflow.keras.layers import Dense, Flatten, Concatenate
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')
```

```
ds_names = ( 'caltech_birds2010',)
dataset_all, info = tfds.load(ds_names[0], split = ['train[:10%]','test[-10%:]'], with_inf
```

```
Downloading and preparing dataset caltech_birds2010/0.1.1 (download: 659.14 MiB, gene
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-108-a5c354a43244> in <module>()
      1 ds_names = ( 'caltech_birds2010',)
----> 2 dataset_all, info = tfds.load(ds_names[0], split =
['train[:10%]','test[-10%:]'], with_info=True)

                                  9 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/lib/io/file_io.py in
seek(self, offset, whence, position)
    165           "Invalid whence argument: {}. Valid values are 0, 1, or
2.".format(
    166               whence))
--> 167     self._read_buf.seek(offset)
    168
    169   def readline(self):

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
info
```

```
tfds.core.DatasetInfo(
    name='caltech_birds2010',
    version=0.1.1,
    description='Caltech-UCSD Birds 200 (CUB-200) is an image dataset with photos
of 200 bird species (mostly North American). The total number of
categories of birds is 200 and there are 6033 images in the 2010
dataset and 11,788 images in the 2011 dataset.
Annotations include bounding boxes, segmentation labels.',
    homepage='http://www.vision.caltech.edu/visipedia/CUB-200.html',
    features=FeaturesDict({
        'bbox': BBoxFeature(shape=(4,), dtype=tf.float32),
        'image': Image(shape=(None, None, 3), dtype=tf.uint8),
        'image/filename': Text(shape=(), dtype=tf.string),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=200),
        'label_name': Text(shape=(), dtype=tf.string),
        'segmentation_mask': Image(shape=(None, None, 1), dtype=tf.uint8),
    }),
    total_num_examples=6033,
    splits={
        'test': 3033,
        'train': 3000,
    },
    supervised_keys=('image', 'label'),
    citation="""@techreport{WelinderEtal2010,
    Author = {P. Welinder and S. Branson and T. Mita and C. Wah and F. Schroff and S
    Institution = {California Institute of Technology},
    Number = {CNS-TR-2010-001},
    Title = {{Caltech-UCSD Birds 200}},
    Year = {2010}
    }""",
    redistribution_info=,
)
```

Следующий код выполнит простую аугументацию данных посредством переворота изображений. В дополнение изображение будет нормализовано к 0 и 1. Пиксели сегментационной маски будут помечены {1, 2, 3}, но для удобства из данного цифрового ряда будет вычтено по 1 и в итоге получиться {0, 1, 2}

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


@tf.function
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


@tf.function
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
```

Датасет уже содержит необходимые тестовый и тренировочный сплиты

```
TRAIN_LENGTH = info.splits['train'].num_examples
BATCH_SIZE = 16
BUFFER_SIZE = 128
STEPS_PER_EPOCH = TRAIN_LENGTH // 10 // BATCH_SIZE


train = dataset_all[0].map(load_image_train, num_parallel_calls=tf.data.experimental.AUTOT
test = dataset_all[1].map(load_image_test)


train_dataset = train.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
test_dataset = test.batch(BATCH_SIZE)
```

```
test_dataset
```

```
<BatchDataset shapes: ((None, 128, 128, 3), (None, 128, 128, 1)), types: (tf.float32,
```

Посмотрим на пример изображения из датасета и соотвествующую ему маску из датасета.

```
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 test_dataset.take(1):
  sample_image, sample_mask = image, mask
display([sample_image[11,:,:,:], sample_mask[11,:,:,:]])
```


```
sample_mask += 1
```

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

```
(1.0, 2.0)
```

```
sample_mask.dtype
```

```
tf.float32
```

Определение модели

Будем использовать модифицированный U-Net. В качестве энкодера будет использоваться предтренированный MobileNetV2. Декодером будет апсемпл блок уже имплементированный в TensorFlow examples Pix2pix tutorial.

Причина по который будет испольваться три канала заключается в том что у нас 3 возможных лейбла на каждый пиксель. Можно это воспринимать как классиификацию где кажедый пиксель будет принадлежать одному из трех классов.

```
OUTPUT_CHANNELS = 3
```

Как упоминалось ранее энкодером будет предтренированный MobileNetV2, который подготовлен и готов к использованию - tf.keras.applications. Энкодер состоит из определенных аутпутов из средних слоев модели. Энкодр не будет участвовать в процессе тренировкие модели.

```
base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3], include_top=Fals

# Use the activations of these layers
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
layers = [base_model.get_layer(name).output for name in layer_names]

# Create the feature extraction model
down_stack = tf.keras.Model(inputs=base_model.input, outputs=layers)

down_stack.trainable = False
```

Декодер/апсемплер это просто серия апсемпл блоков имплементированнхы в TensorFlow examples

```
up_stack = [
    pix2pix.upsample(512, 3),  # 4x4 -> 8x8
    pix2pix.upsample(256, 3),  # 8x8 -> 16x16
    pix2pix.upsample(128, 3),  # 16x16 -> 32x32
```

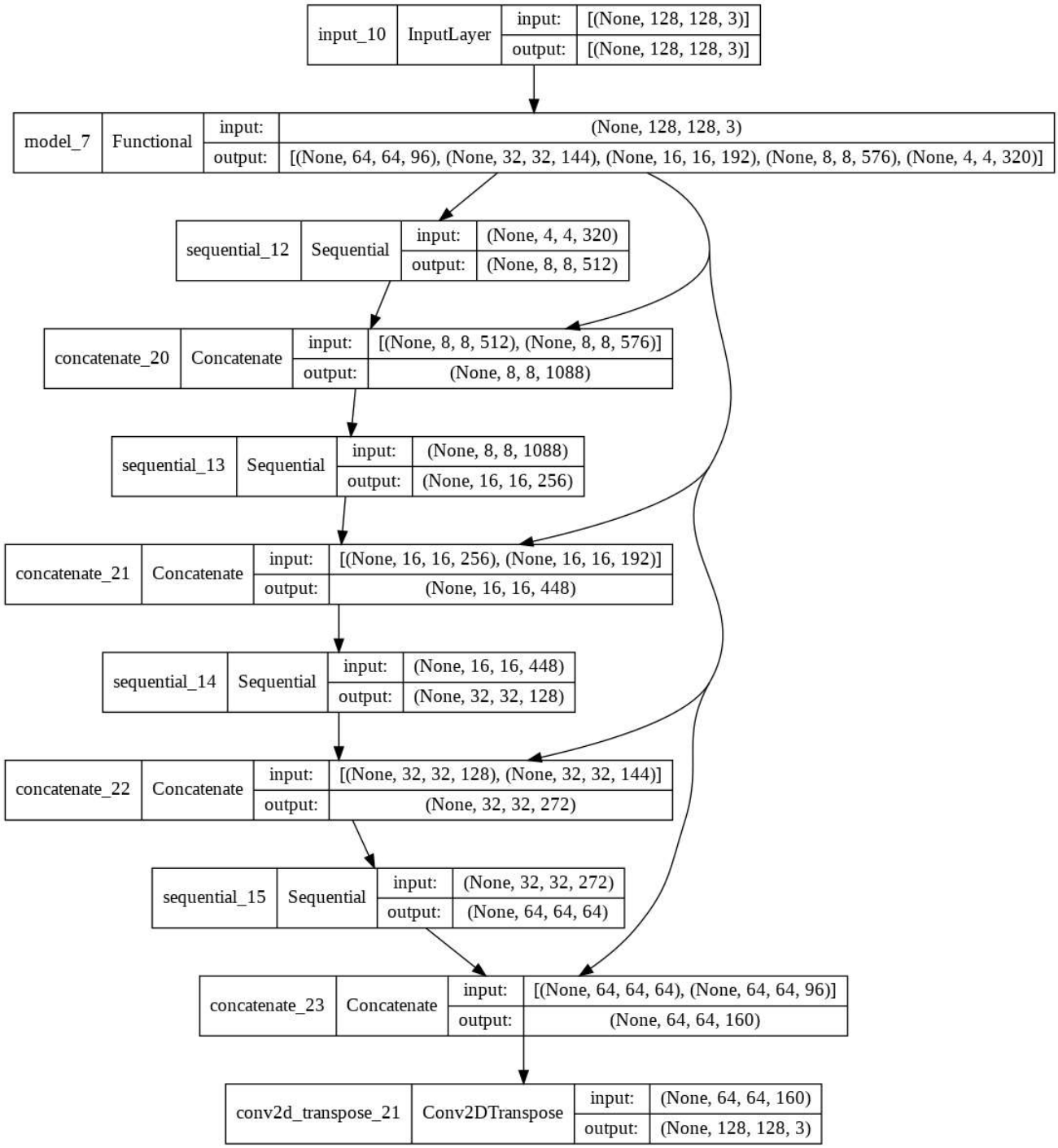
```
    pix2pix.upsample(64, 3),   # 32x32 -> 64x64
]



def unet_model(output_channels):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])
  x = inputs

  # Downsampling through the model
  skips = down_stack(x)
  x = skips[-1]
  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # This is the last layer of the model
  last = tf.keras.layers.Conv2DTranspose(
      output_channels, 3, strides=2,
      padding='same')  #64x64 -> 128x128

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

Тренировка модели

```
model = unet_model(OUTPUT_CHANNELS)
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              # loss = 'binary_crossentropy',
              metrics=['accuracy'])
```

Посмотрим на получившуюся архитектуру модели.

```
tf.keras.utils.plot_model(model, show_shapes=True)
```



попробуем сделать предсказание с помощью нашей модели до того как началось обучение.

```
def create_mask(pred_mask):
  pred_mask = tf.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]


def show_predictions(dataset=None, num=1):
```

```
  if dataset:
    for image, mask in dataset.take(num):
      print(1)
      pred_mask = model.predict(image)
      display([image[0], mask[0], create_mask(pred_mask)])
  else:
    display([sample_image[0,:,:,:], sample_mask[0,:,:,:],
             create_mask(model.predict(sample_image[:,:,:]))])


show_predictions()
```

```
WARNING:tensorflow:5 out of the last 11 calls to <function Model.make_predict_functic
WARNING:tensorflow:5 out of the last 11 calls to <function Model.make_predict_functic
```

Буду осуществлять мониторинг того как улучшается работа модели в процессе обучения. Для завершения этой задачи callback функция определена ниже.

```
class DisplayCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    clear_output(wait=True)
    show_predictions()
    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))


train_dataset
```

```
<PrefetchDataset shapes: ((None, 128, 128, 3), (None, 128, 128, 1)), types: (tf.float
```

```
test_dataset
```

```
<BatchDataset shapes: ((None, 128, 128, 3), (None, 128, 128, 1)), types: (tf.float32,
```

```
info.splits['test'].num_examples//10//16//5
```

```
3
```

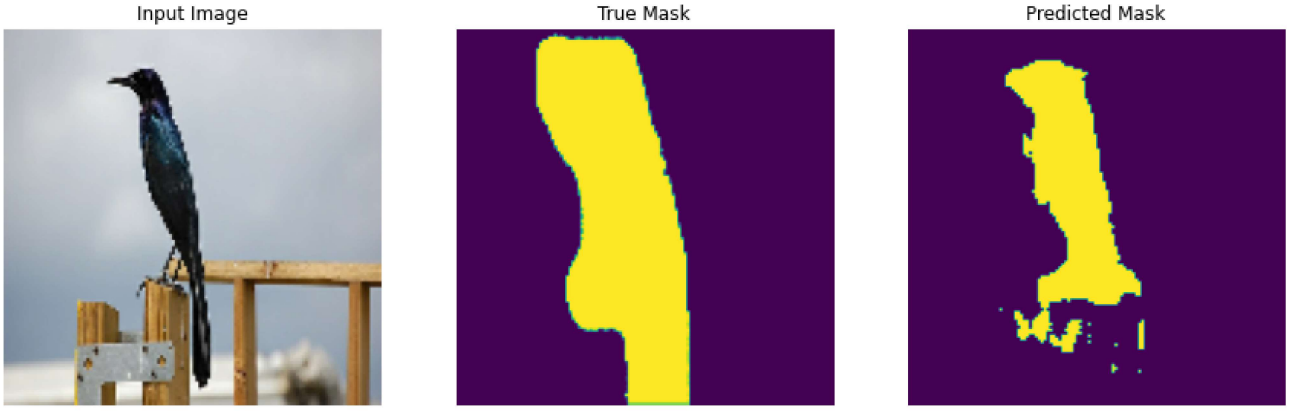
```
EPOCHS = 10
VAL_SUBSPLITS = 5
VALIDATION_STEPS = info.splits['test'].num_examples//10//BATCH_SIZE//VAL_SUBSPLITS

model_history = model.fit(train_dataset, epochs=EPOCHS,
                          steps_per_epoch=STEPS_PER_EPOCH,
                          validation_steps=VALIDATION_STEPS,
                          validation_data=test_dataset,
                          callbacks=[DisplayCallback()]
                          )
```


```
Sample Prediction after epoch 10

18/18 [==============================] - 19s 1s/step - loss: 0.0403 - accuracy: 0.986
```

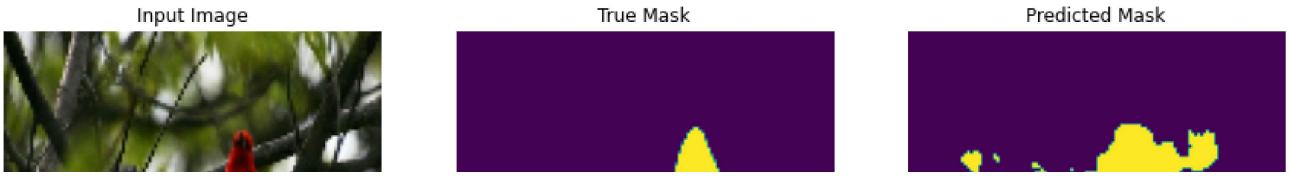
```
show_predictions(test_dataset, 3)
```

1

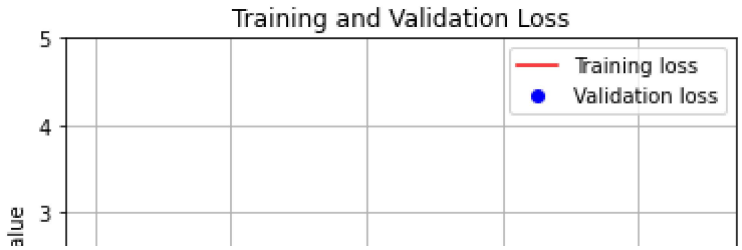
1



1




```
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']

epochs = range(EPOCHS)

plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'bo', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.ylim([0, 5])
plt.grid('on')
plt.legend()
plt.show()
```

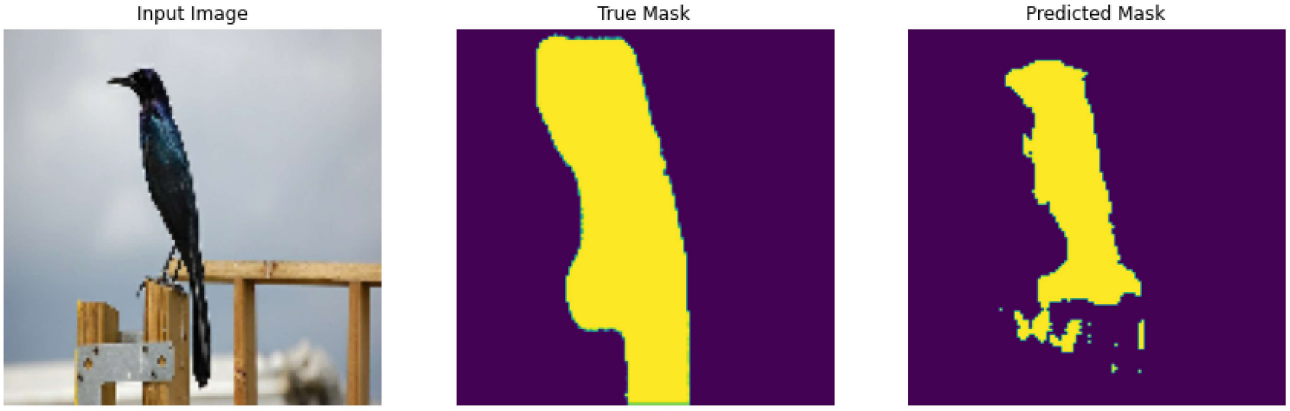


несколько предсказаний

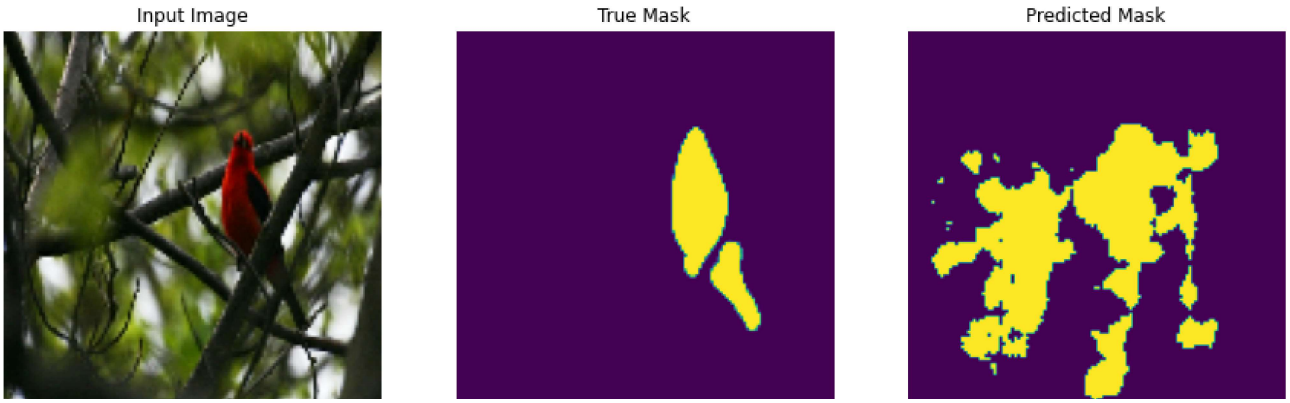
```
show_predictions(test_dataset, 5)
```

1

1



1



Вывод

для лучшего качества нужно намного больше эпох
с увеличением эпох с 5 до 15 и дообучением еще на 10 результат стал намного лучше
хорошо было бы увеличить до 1000 и посмотреть, но колаб не хочет столько обучать

1