In [1]:
```
##### Copyright 2019 The TensorFlow Authors.

#Licensed under the Apache License, Version 2.0 (the "License");

#@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.
```

In [2]:
```
# !pip install git+https://github.com/tensorflow/examples.git
# !pip install tensorflow_datasets
```

In [3]:
```
import tensorflow as tf
```

In [4]:
```
from __future__ import absolute_import, division, print_function, unicode_li
terals

from tensorflow_examples.models.pix2pix import pix2pix

from keras.layers import Activation

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')
```

In [5]:
```
# Загрузка датасета Oxford-IIIT Pets
dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
```

In [6]:
```
def normalize(input_image, input_mask):
  input_image = tf.cast(input_image, tf.float32) / 255.0
  input_mask -= 1
  return input_image, input_mask
```

In [7]:
```
@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
```

In [8]:
```
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
```

In [9]:
```
# Датасет уже содержит необходимые тестовый и тренеровочный сплиты, поэтому
давайте использовать их.
```

In [10]:
```
TRAIN_LENGTH = info.splits['train'].num_examples
BATCH_SIZE = 128
BUFFER_SIZE = 20000
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
```

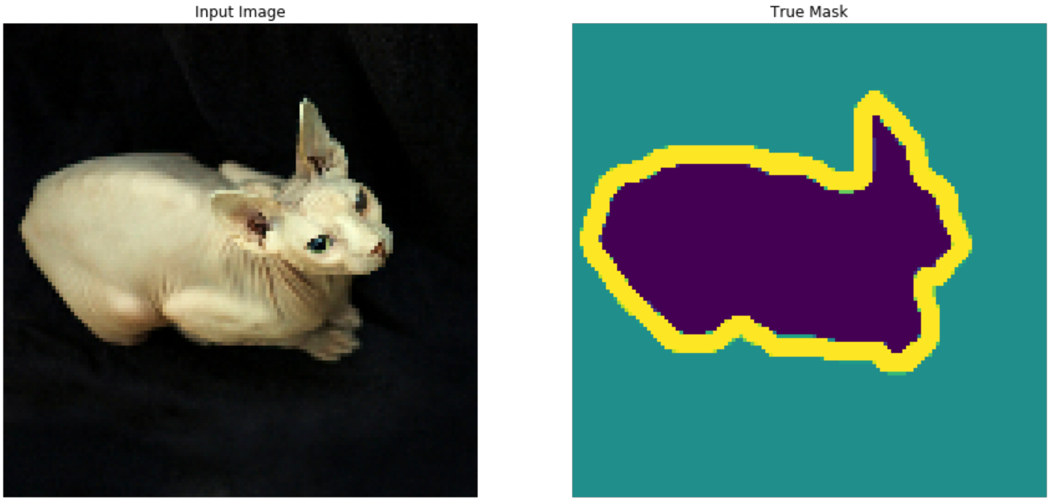
In [11]:
```
train = dataset['train'].map(load_image_train, num_parallel_calls=tf.data.ex
perimental.AUTOTUNE)
test = dataset['test'].map(load_image_test)
```

In [12]:
```
train_dataset = train.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat
()
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTO
TUNE)
test_dataset = test.batch(BATCH_SIZE)
```

In [13]:
```
# Давайте посмотрим на пример изображения из датасета и соотвествующую ему м
аску из датасета.
```

In [14]:
```
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()
```

In [15]:
```
for image, mask in train.take(1):
  sample_image, sample_mask = image, mask
display([sample_image, sample_mask])
```



In [16]:
```
# Определение модели
# энкодером будет предтренированный MobileNetV2
```

In [17]:
```
OUTPUT_CHANNELS = 3
```

In [18]:
```
base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3], in
clude_top=False)

# 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
```

In [19]:
```
# Декодер/апсемплер это просто серия апсемпл блоков имплементированны в Tens
orFlow examples.
```

In [20]:
```
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
]
```

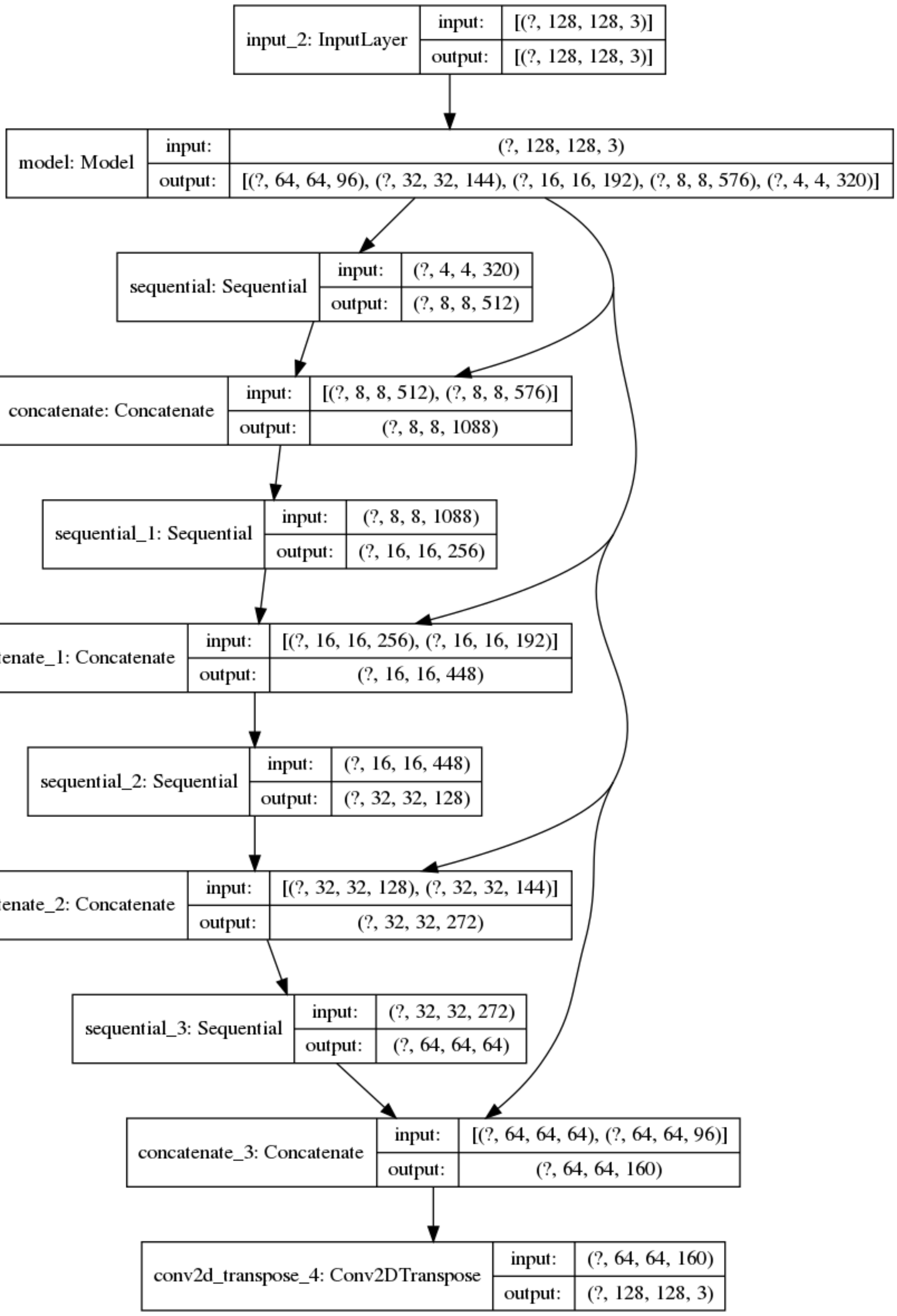
In [21]:
```
def unet_model(output_channels):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])
  x = inputs

  # 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)
```

In [22]:
```
# Тренировка модели
```

In [23]:
```
model = unet_model(OUTPUT_CHANNELS)
model.compile(optimizer='SGD',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits
=True),
              metrics=['accuracy'])
```

In [24]:
```
tf.keras.utils.plot_model(model, show_shapes=True)
```

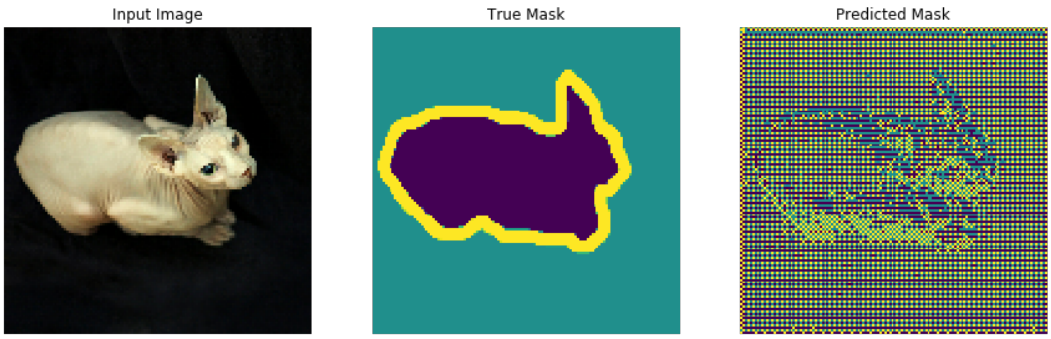
Out[24]:



In [25]:
```
def create_mask(pred_mask):
  pred_mask = tf.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]
```

In [26]:
```
def show_predictions(dataset=None, num=1):
  if dataset:
    for image, mask in dataset.take(num):
      pred_mask = model.predict(image)
      display([image[0], mask[0], create_mask(pred_mask)])
  else:
    display([sample_image, sample_mask,
             create_mask(model.predict(sample_image[tf.newaxis, ...]))])
```

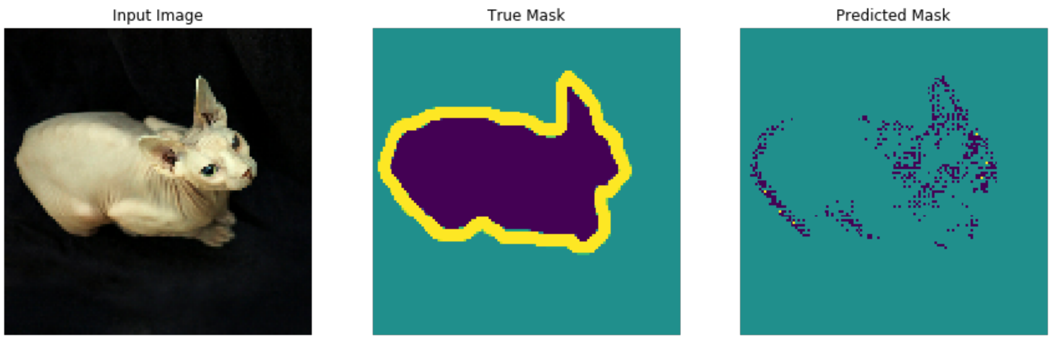
In [27]:
```
show_predictions()
```



In [28]:
```
# Давайте осуществлять мониторинг того как улучшается работа модели в процес
се обучения. Для завершения этой задачи callback функция определена ниже.
```

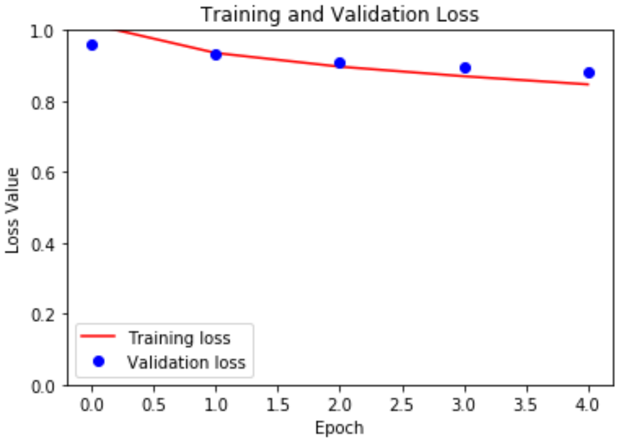
In [29]:
```
class DisplayCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    clear_output(wait=True)
    show_predictions()
    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))
```

In [30]:
```
EPOCHS = 5 # увеличьте при необходимости
VAL_SUBSPLITS = 5
VALIDATION_STEPS = info.splits['test'].num_examples//BATCH_SIZE//VAL_SUBSPLI
TS

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



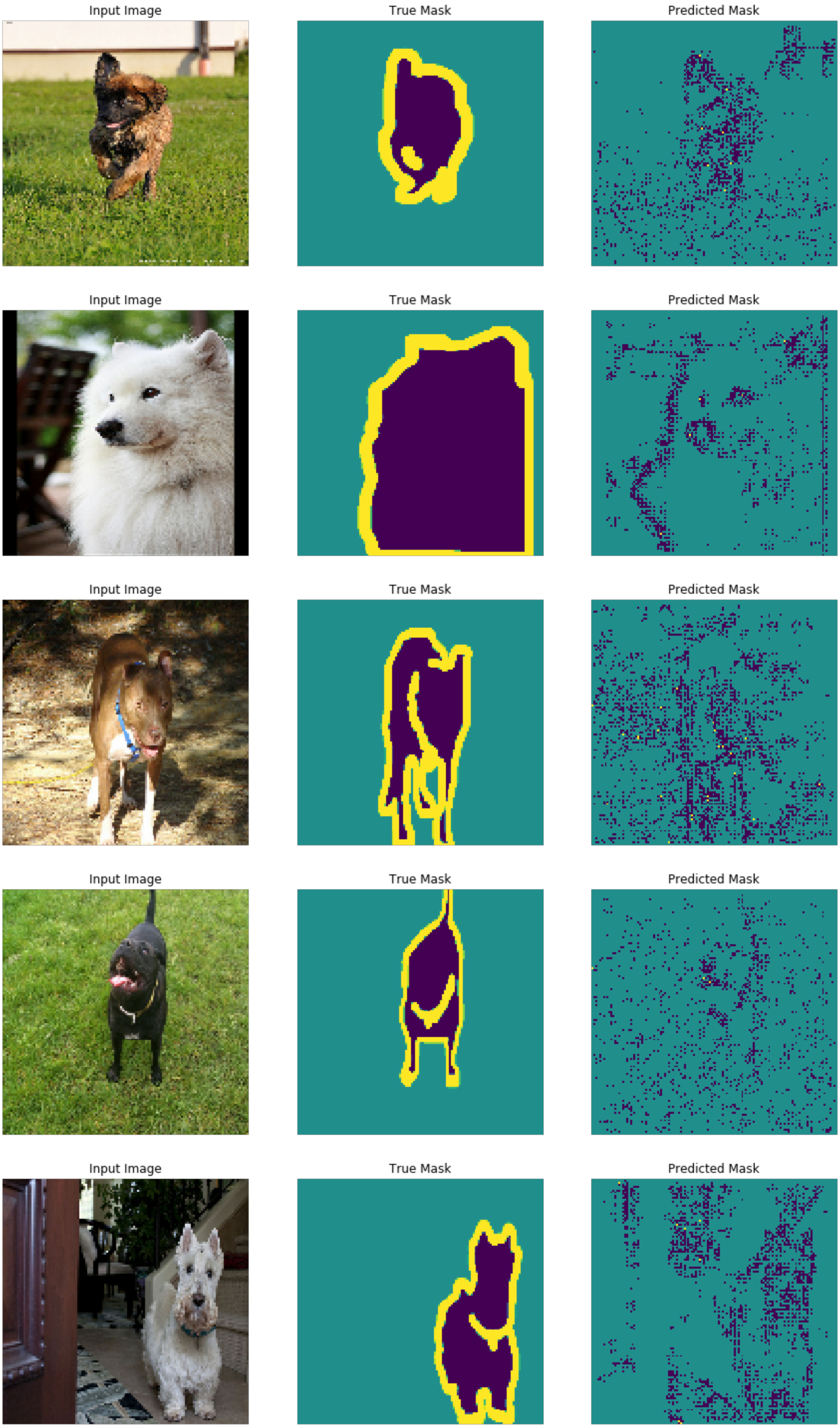

```
Sample Prediction after epoch 5

28/28 [==============================] - 301s 11s/step - loss: 0.8471 - accur
acy: 0.5960 - val_loss: 0.8809 - val_accuracy: 0.5744
```

In [31]:
```
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, 1])
plt.legend()
plt.show()
```



In [32]:
```
# Make predictions
```

In [33]:
```
show_predictions(test_dataset, 5)
```



```
In [ ]:
```