U-Net Pet Segmentation

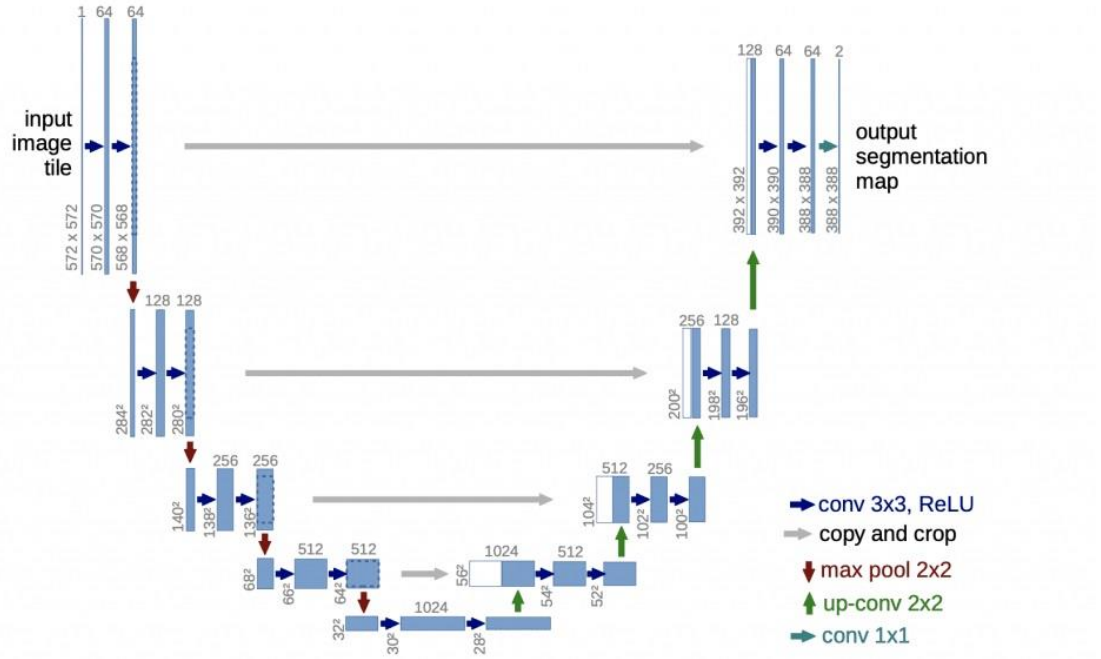
دو نوع تقسیم بندی تصویر وجود دارد:

1. تقسیم بندی معنایی (Semantic segmentation): هر پیکسل را با یک برچسب طبقه بندی میکند.
2. تقسیم‌بندی نمونه (Instance segmentation): هر پیکسل را طبقه‌بندی میکند و هر نمونه شی را متمایز میکند.

U-Net یک تکنیک تقسیم بندی معنایی است که در ابتدا برای تقسیم بندی تصویربرداری پزشکی پیشنهاد شده است. این یکی از مدل‌های تقسیم‌بندی یادگیری عمیق قبلی است و معماری U-Net نیز در بسیاری از انواع GAN مانند Generator Pix2Pix استفاده می‌شود.

معماری U-Net

U-Net در مقاله U-Net: Convolutional Networks for Biomedical Image Segmentation معرفی شد. معماری مدل نسبتاً ساده است: یک Encoding (برای نمونه برداری پایین) و یک Decoding (برای نمونه برداری بالا) با اتصالات پرش. همانطور که شکل زیر نشان می دهد، شکل آن مانند حرف U است از این رو U-Net نامیده می شود.



فلش‌های خاکستری نشان‌دهنده اتصالات پرش است که نقشه ویژگی Encoding را با Decoding پیوند می‌دهد، که به جریان برگشتی گرادیان‌ها برای بهبود آموزش کمک می‌کند.

اکنون که درک اولیه ای از تقسیم بندی معنایی و معماری U-Net داریم، بیایید یک U-Net را با TensorFlow 2 / Keras پیاده سازی کنیم.

ما از Colab برای آموزش مدل استفاده خواهیم کرد، بنابراین مطمئن شوید که "Hardware accelerator" را روی " / GPU under Runtime change runtime type" تنظیم کرده اید. سپس کتابخانه ها و بسته ها را وارد میکنیم که این پروژه به آنها بستگی دارد:

Importing the required libraries and packages:

In [1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import numpy as np
```

مجموعه داده

ما از مجموعه داده های حیوانات خانگی Oxford-IIIT که به عنوان بخشی از مجموعه داده های TensorFlow (TFDS) موجود است، استفاده خواهیم کرد. می توان آن را به راحتی با TFDS بارگیری کرد، و سپس با کمی پیش پردازش داده، آماده برای آموزش مدل های تقسیم بندی می شود.
فقط با یک خط کد، می توانیم از tfds برای بارگذاری مجموعه داده با مشخص کردن نام مجموعه داده استفاده کنیم و با تنظیم with_info=True اطلاعات مجموعه داده را دریافت کنیم:

Loading the Oxford-IIIT pet dataset:

In [2]:
```
dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
```

```
Downloading and preparing dataset 773.52 MiB (download: 773.52 MiB, generated: 774.69 MiB, total: 1.51 GiB) to /root/tensor
flow_datasets/oxford_iiit_pet/3.2.0...
Dl Completed...: 0 url [00:00, ? url/s]
Dl Size...: 0 MiB [00:00, ? MiB/s]
Extraction completed...: 0 file [00:00, ? file/s]
Generating splits...:   0%|          | 0/2 [00:00<?, ? splits/s]
Generating train examples...:   0%|          | 0/3680 [00:00<?, ? examples/s]
Shuffling /root/tensorflow_datasets/oxford_iiit_pet/3.2.0.incompleteVAF5VR/oxford_iiit_pet-train.tfrecord*...:…
Generating test examples...:   0%|          | 0/3669 [00:00<?, ? examples/s]
Shuffling /root/tensorflow_datasets/oxford_iiit_pet/3.2.0.incompleteVAF5VR/oxford_iiit_pet-test.tfrecord*...: …
Dataset oxford_iiit_pet downloaded and prepared to /root/tensorflow_datasets/oxford_iiit_pet/3.2.0. Subsequent calls will r
euse this data.
```

اطلاعات مجموعه داده را با print(info) چاپ میکنیم، و ما انواع اطلاعات دقیق در مورد مجموعه داده حیوانات خانگی آکسفورد را خواهیم دید. به عنوان مثال، در شکل زیر می‌توانیم ببینیم که در مجموع 7349 تصویر با تقسیم test/train split وجود دارد.

Take a look at the dataset info. Note the test and train data split is already built in the dataset.

In [3]:
```
print (info)
```

```
tfds.core.DatasetInfo(
    name='oxford_iiit_pet',
    full_name='oxford_iiit_pet/3.2.0',
    description="""
    The Oxford-IIIT pet dataset is a 37 category pet image dataset with roughly 200
    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/',
    data_dir=PosixGPath('/tmp/tmpdok286twtfds'),
    file_format=tfrecord,
    download_size=773.52 MiB,
    dataset_size=774.69 MiB,
    features=FeaturesDict({
        'file_name': Text(shape=(), dtype=string),
        'image': Image(shape=(None, None, 3), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=37),
        'segmentation_mask': Image(shape=(None, None, 1), dtype=uint8),
        'species': ClassLabel(shape=(), dtype=int64, num_classes=2),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=3669, num_shards=4>,
        'train': <SplitInfo num_examples=3680, num_shards=4>,
    },
    citation="""@InProceedings{parkhi12a,
      author       = "Parkhi, O. M. and Vedaldi, A. and Zisserman, A. and Jawahar, C.~V.",
      title        = "Cats and Dogs",
      booktitle    = "IEEE Conference on Computer Vision and Pattern Recognition",
      year         = "2012",
    }""",
)
```

اجازه دهید قبل از شروع آموزش U-Net با آن، ابتدا چند تغییر در داده های دانلود شده ایجاد کنیم.
ابتدا باید اندازه تصاویر و ماسک ها را به 128×128 تغییر دهیم:

Data preprocessing

In [6]:
```
def resize(input_image, input_mask):
    input_image = tf.image.resize(input_image, (128, 128), method="nearest")
    input_mask = tf.image.resize(input_mask, (128, 128), method="nearest")

    return input_image, input_mask
```

سپس یک تابع برای تقویت مجموعه داده با چرخاندن افقی آنها ایجاد می کنیم:

In [7]:
```
def augment(input_image, input_mask):
    if tf.random.uniform(()) > 0.5:
        # Random flipping of the image and mask
        input_image = tf.image.flip_left_right(input_image)
        input_mask = tf.image.flip_left_right(input_mask)

    return input_image, input_mask
```

ما یک تابع برای نرمال سازی مجموعه داده با مقیاس بندی تصاویر در محدوده [-1، 1] و کاهش ماسک تصویر به میزان 1 ایجاد می کنیم:

In [8]:
```
def normalize(input_image, input_mask):
    input_image = tf.cast(input_image, tf.float32) / 255.0
    input_mask -= 1

    return input_image, input_mask
```

ما دو تابع را برای پیش پردازش مجموعه داده های Train و Test ایجاد می کنیم، با تفاوت جزئی بین این دو. ما فقط در مجموعه داده های آموزشی Image augmentation را انجام می دهیم.

In [9]:
```
def load_image_train(datapoint):
    input_image = datapoint["image"]
    input_mask = datapoint["segmentation_mask"]
    input_image, input_mask = resize(input_image, input_mask)
    input_image, input_mask = augment(input_image, input_mask)
    input_image, input_mask = normalize(input_image, input_mask)

    return input_image, input_mask
```

In [10]:
```
def load_image_test(datapoint):
    input_image = datapoint["image"]
    input_mask = datapoint["segmentation_mask"]
    input_image, input_mask = resize(input_image, input_mask)
    input_image, input_mask = normalize(input_image, input_mask)

    return input_image, input_mask
```

اکنون با استفاده از تابع map() آماده ساخت یک Pipeline ورودی با tf.data هستیم:

In [11]:
```
train_dataset = dataset["train"].map(load_image_train, num_parallel_calls=tf.data.AUTOTUNE)
test_dataset = dataset["test"].map(load_image_test, num_parallel_calls=tf.data.AUTOTUNE)
```

In [12]:
```
print(train_dataset)
```

```
<_ParallelMapDataset element_spec=(TensorSpec(shape=(128, 128, 3), dtype=tf.float32, name=None), TensorSpec(shape=(128, 12
8, 1), dtype=tf.uint8, name=None))>
```

اگر print(train_dataset) را اجرا کنیم، متوجه می شویم که تصویر به شکل 128×128×3 با نوع داده tf.float32 است در حالی که ماسک تصویر به شکل 128×128×1 با نوع داده tf.uint8 است.
ما اندازه دسته ای (batch size) 64 و اندازه بافر (buffer size) 1000 را برای ایجاد دسته ای از مجموعه داده های Train و Test تعریف می کنیم. با مجموعه داده اصلی TFDS، 3680 نمونه آموزشی و 3669 نمونه تست وجود دارد که بیشتر به مجموعه‌های اعتبارسنجی/آزمون تقسیم می‌شوند. برای آموزش مدل U-Net از train_batches و validation_batches استفاده خواهیم کرد. پس از پایان آموزش، از test_batches برای آزمایش پیش‌بینی‌های مدل استفاده می‌کنیم.

In [13]:
```
BATCH_SIZE = 64
BUFFER_SIZE = 1000
```

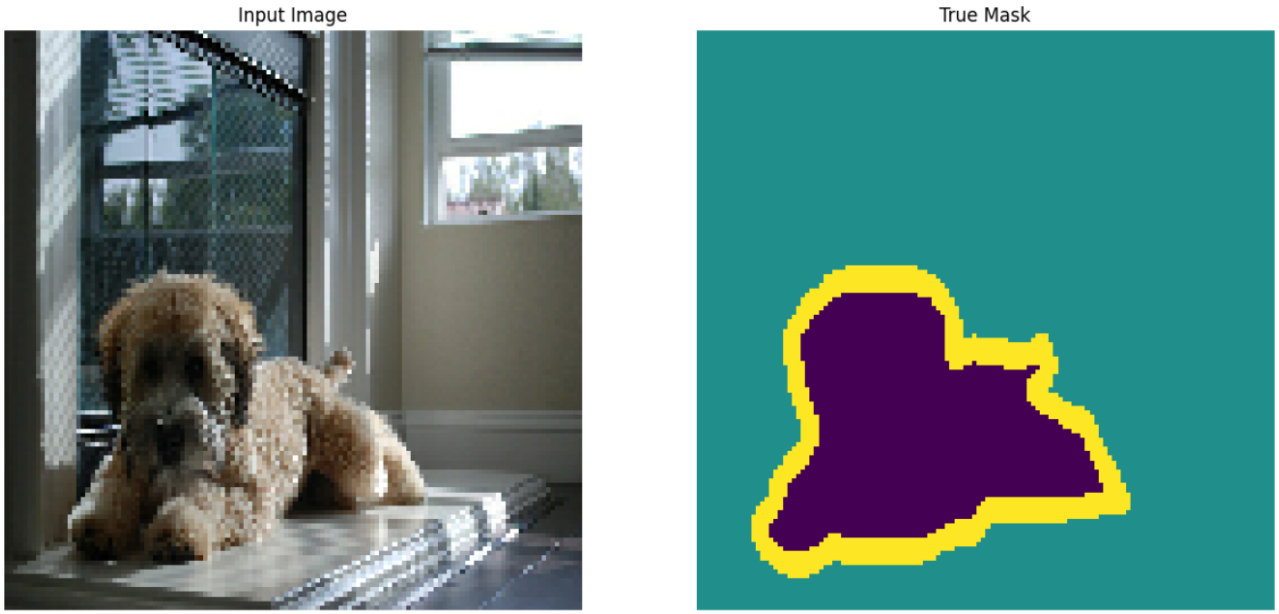
In [14]:
```
train_batches = train_dataset.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()
train_batches = train_batches.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
validation_batches = test_dataset.take(3000).batch(BATCH_SIZE)
test_batches = test_dataset.skip(3000).take(669).batch(BATCH_SIZE)
```

تجسم سازی دیتا

اکنون مجموعه داده ها برای آموزش آماده هستند. بیایید یک تصویر نمونه تصادفی و ماسک آن را از مجموعه داده آموزشی visualize کنیم تا ایده ای از نحوه به نظر رسیدن داده ها داشته باشیم.

In [16]:
```
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.utils.array_to_img(display_list[i]))
    plt.axis("off")
  plt.show()
```

In [17]:
```
sample_batch = next(iter(test_batches))
random_index = np.random.choice(sample_batch[0].shape[0])
sample_image, sample_mask = sample_batch[0][random_index], sample_batch[1][random_index]
display([sample_image, sample_mask])
```



نمونه تصویر ورودی یک گربه به شکل 128×128×3 می باشد. ماسک واقعی سه بخش دارد: پس‌زمینه سبز, شی پیش زمینه بنفش (یک گربه)، و طرح زرد. شکل هم تصویر ورودی اصلی و هم تصویر ماسک واقعی را نشان می دهد.

معماری مدل

اکنون که داده ها را برای آموزش آماده کرده ایم، بیایید معماری مدل U-Net را تعریف کنیم. همانطور که قبلا ذکر شد، U-Net به شکل یک حرف U است که دارای Encoding، Decoding و اتصالات پرش بین آنها است. بنابراین ما چند بلوک برای ساخت مدل U-Net ایجاد خواهیم کرد.

ساخت بلوک

ابتدا یک تابع double_conv_block با لایه های Conv2D-ReLU-Conv2D-ReLU ایجاد می کنیم که هم در Encoding (یا مسیر contracting) و هم در گلوگاه (bottleneck) U-Net استفاده می کنیم.

U-Net Building blocks

Create the building blocks for making the components U-Net model.

In [18]:
```
def double_conv_block(x, n_filters):

    # Conv2D then ReLU activation
    x = layers.Conv2D(n_filters, 3, padding = "same", activation = "relu", kernel_initializer = "he_normal")(x)
    # Conv2D then ReLU activation
    x = layers.Conv2D(n_filters, 3, padding = "same", activation = "relu", kernel_initializer = "he_normal")(x)

    return x
```

سپس یک تابع downsample_block برای downsampling یا استخراج ویژگی تعریف می کنیم تا در Encoding استفاده شود.

In [19]:
```
def downsample_block(x, n_filters):
    f = double_conv_block(x, n_filters)
    p = layers.MaxPool2D(2)(f)
    p = layers.Dropout(0.3)(p)

    return f, p
```

در نهایت، یک تابع upsample_block برای decoding (یا مسیر expanding) U-Net تعریف می کنیم.

In [20]:
```
def upsample_block(x, conv_features, n_filters):
    # upsample
    x = layers.Conv2DTranspose(n_filters, 3, 2, padding="same")(x)
    # concatenate
    x = layers.concatenate([x, conv_features])
    # dropout
    x = layers.Dropout(0.3)(x)
    # Conv2D twice with ReLU activation
    x = double_conv_block(x, n_filters)

    return x
```

مدل U-Net

سه گزینه برای ساخت مدل Keras وجود دارد که در وبلاگ آدریان (Adrian's blog) و مستندات Keras توضیح داده شده است:

1) Sequential API: ساده‌ترین و مبتدی پسندترین، چیدن لایه‌ها به صورت متوالی.
2) Functional API: انعطاف‌پذیرتر است و توپولوژی غیر خطی، لایه‌های مشترک و ورودی‌ها یا خروجی‌های متعدد را امکان‌پذیر می‌کند.
3) Model subclassing: انعطاف پذیرترین و بهترین برای مدل های پیچیده که نیاز به حلقه های آموزشی سفارشی دارند.

U-Net یک معماری نسبتا ساده دارد. با این حال، برای ایجاد اتصالات پرش بین encoding و decoding، باید چند لایه را به هم متصل کنیم. بنابراین Keras Functional API برای این منظور مناسب ترین است.

ابتدا یک تابع build_unet_model ایجاد می کنیم، ورودی ها، لایه های encoding، گلوگاه (bottleneck)، لایه های decoding و در نهایت لایه خروجی را با Conv2D با فعال سازی softmax مشخص می کنیم. توجه داشته باشید که شکل تصویر ورودی 128×128×3 است. خروجی دارای سه کانال مربوط به سه کلاس است که مدل هر پیکسل را برای آنها طبقه بندی می کند: پس زمینه، شی پیش زمینه و طرح کلی شی.

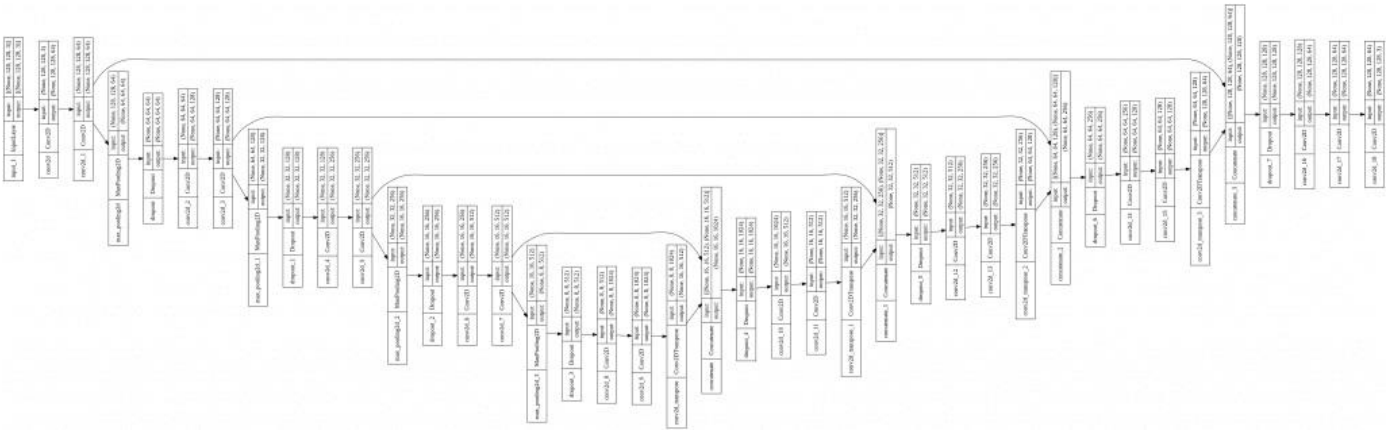
Build the U-Net Model

In [21]:
```
def build_unet_model():

    # inputs
    inputs = layers.Input(shape=(128,128,3))

    # encoder: contracting path - downsample
    # 1 - downsample
    f1, p1 = downsample_block(inputs, 64)
    # 2 - downsample
    f2, p2 = downsample_block(p1, 128)
    # 3 - downsample
    f3, p3 = downsample_block(p2, 256)
    # 4 - downsample
    f4, p4 = downsample_block(p3, 512)

    # 5 - bottleneck
    bottleneck = double_conv_block(p4, 1024)

    # decoder: expanding path - upsample
    # 6 - upsample
    u6 = upsample_block(bottleneck, f4, 512)
    # 7 - upsample
    u7 = upsample_block(u6, f3, 256)
    # 8 - upsample
    u8 = upsample_block(u7, f2, 128)
    # 9 - upsample
    u9 = upsample_block(u8, f1, 64)

    # outputs
    outputs = layers.Conv2D(3, 1, padding="same", activation = "softmax")(u9)

    # unet model with Keras Functional API
    unet_model = tf.keras.Model(inputs, outputs, name="U-Net")

    return unet_model
```

برای ایجاد مدل unet_model تابع build_unet_model را فراخوانی می کنیم:

In [22]:
```
unet_model = build_unet_model()
```

و می‌توانیم معماری مدل را با model.summary() تجســـم کنیم تا جزئیات مدل را ببینیم. و ما می توانیم از یک تابع Keras Utils به نام plot_model برای تولید یک نمودار بصـری تر، از جمله اتصـالات پرش اسـتفاده کنیم. تصـویر تولید شـده در Colab، 90 درجه چرخیده است تا بتوانید معماری U شکل را در شکل زیر ببینید:



کامپایل و آموزش U-Net

برای کامپایل unet_model، بهینه‌ساز، loss function و معیارهای دقت (accuracy) را برای ردیابی در طول آموزش مشخص می‌کنیم:

Compile and Train U-Net

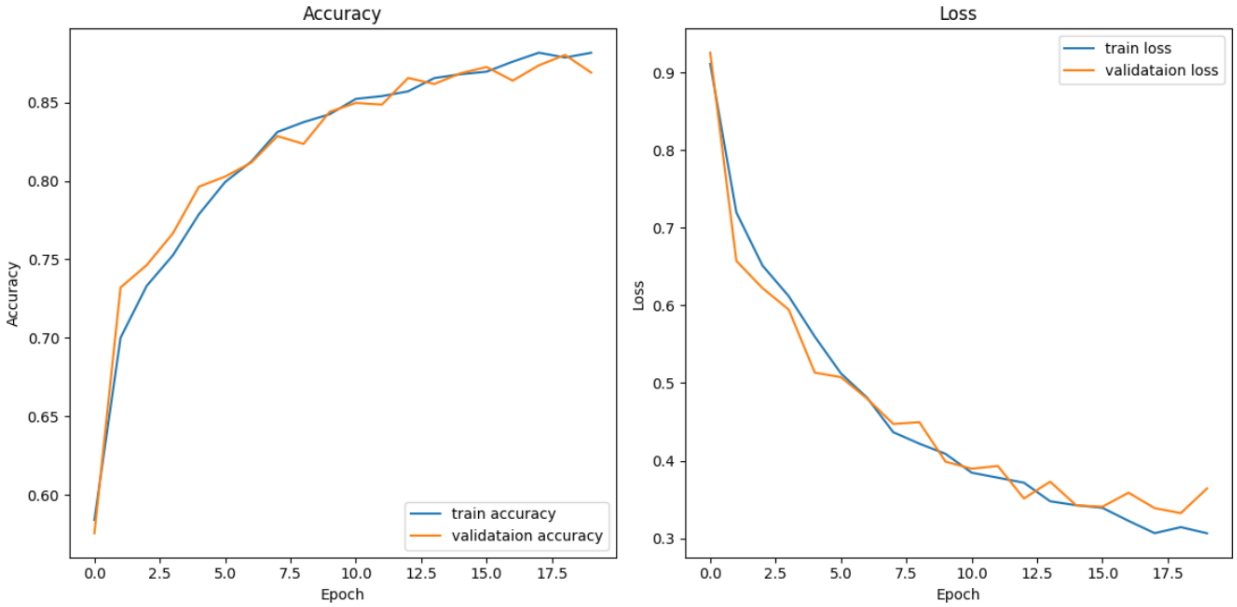
In [25]:
```
unet_model.compile(optimizer=tf.keras.optimizers.Adam(),
                   loss="sparse_categorical_crossentropy",
                   metrics="accuracy")
```

ما unet_model را با فراخوانی model.fit() و آموزش آن برای 20 epochs آموزش می دهیم.

In [26]:
```
NUM_EPOCHS = 20

TRAIN_LENGTH = info.splits["train"].num_examples
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE

VAL_SUBSPLITS = 5
TEST_LENTH = info.splits["test"].num_examples
VALIDATION_STEPS = TEST_LENTH // BATCH_SIZE // VAL_SUBSPLITS

model_history = unet_model.fit(train_batches,
                               epochs=NUM_EPOCHS,
                               steps_per_epoch=STEPS_PER_EPOCH,
                               validation_steps=VALIDATION_STEPS,
                               validation_data=validation_batches)
```

```
Epoch 1/20
57/57 [==============================] - 103s 1s/step - loss: 0.9110 - accuracy: 0.5842 - val_loss: 0.9254 - val_accuracy:
0.5757
Epoch 2/20
57/57 [==============================] - 72s 1s/step - loss: 0.7196 - accuracy: 0.7001 - val_loss: 0.6573 - val_accuracy:
0.7321
Epoch 3/20
57/57 [==============================] - 58s 1s/step - loss: 0.6513 - accuracy: 0.7332 - val_loss: 0.6222 - val_accuracy:
0.7464
Epoch 4/20
57/57 [==============================] - 60s 1s/step - loss: 0.6120 - accuracy: 0.7526 - val_loss: 0.5947 - val_accuracy:
0.7666
Epoch 5/20
57/57 [==============================] - 60s 1s/step - loss: 0.5597 - accuracy: 0.7788 - val_loss: 0.5134 - val_accuracy:
0.7963
Epoch 6/20
57/57 [==============================] - 58s 1s/step - loss: 0.5123 - accuracy: 0.7993 - val_loss: 0.5076 - val_accuracy:
0.8027
```

پس از آموزش برای 20 دوره، ما دقت آموزشـــی و دقت اعتبار (validation) 0.88 ~ را دریافت می کنیم. منحنی یادگیری در طول آموزش نشان می‌دهد که مدل در مجموعه داده‌های آموزشی و مجموعه اعتبارسنجی به خوبی عمل می‌کند، که نشان می‌دهد مدل به خوبی و بدون overfitting تعمیم می‌یابد.



پیش بینی

اکنون که آموزش unet_model را به پایان رساندیم، اجازه دهید از آن برای پیش‌بینی چند تصویر نمونه از مجموعه داده آزمایشی استفاده کنیم.

Predictions with U-Net model

Let's try the trained U-Net model on a few samples from the test dataset.

In [29]:
```
def create_mask(pred_mask):
  pred_mask = tf.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]
```

In [30]:
```
def show_predictions(dataset=None, num=1):
  if dataset:
    for image, mask in dataset.take(num):
      pred_mask = unet_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 [31]:
```
count = 0
for i in test_batches:
    count +=1
print("number of batches:", count)
```

```
number of batches: 11
```

برای تصاویر ورودی، ماسک های واقعی، و ماسک های پیش بینی شده توسط مدل U-Net که آموزش دادیم، به شکل زیر مراجعه کنید.