Experiment: 4

Aim:

To implement the U-Net architecture for performing image segmentation on a pet dataset, where each image is segmented to identify specific regions, such as the pet's body or background.

Theory:

U-Net is a convolutional neural network architecture designed specifically for image segmentation tasks, where the goal is to classify each pixel in an image. It consists of two main paths:

Contracting Path (Encoder): This path captures context in the image through a series of convolutional and pooling layers, progressively reducing the spatial dimensions while increasing feature depth.

Expanding Path (Decoder): It restores the spatial dimensions by upsampling and uses skip connections to combine low-level features from the encoder with high-level features, improving the precision of the segmentation.

U-Net is effective for medical imaging, object segmentation, and other pixel-wise classification tasks due to its ability to leverage both local and global information.

Importing Necessary libraries

```
!pip install tensorflow_datasets
```

```
Requirement already satisfied: tensorflow_datasets in
/usr/local/lib/python3.10/dist-packages (4.9.6)
Requirement already satisfied: absl-py in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(1.4.0)
Requirement already satisfied: click in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(8.1.7)
Requirement already satisfied: dm-tree in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(0.1.8)
Requirement already satisfied: immutabledict in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(4.2.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(1.26.4)
Requirement already satisfied: promise in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(2.3)
Requirement already satisfied: protobuf>=3.20 in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(3.20.3)
```

```
Requirement already satisfied: psutil in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(5.9.5)
Requirement already satisfied: pyarrow in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(16.1.0)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(2.32.3)
Requirement already satisfied: simple-parsing in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(0.1.6)
Requirement already satisfied: tensorflow-metadata in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(1.16.1)
Requirement already satisfied: termcolor in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(2.5.0)
Requirement already satisfied: toml in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(0.10.2)
Requirement already satisfied: tqdm in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(4.66.5)
Requirement already satisfied: wrapt in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(1.16.0)
Requirement already satisfied: array-record>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow_datasets)
(0.5.1)
Requirement already satisfied: etils>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from
etils[enp,epath,epy,etree]>=1.6.0; python_version < "3.11"-
>tensorflow_datasets) (1.9.4)
Requirement already satisfied: typing_extensions in
/usr/local/lib/python3.10/dist-packages (from
etils[enp,epath,epy,etree]>=1.6.0; python_version < "3.11"-
>tensorflow_datasets) (4.12.2)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.10/dist-packages (from
etils[enp,epath,epy,etree]>=1.6.0; python_version < "3.11"-
>tensorflow_datasets) (2024.6.1)
Requirement already satisfied: importlib_resources in
/usr/local/lib/python3.10/dist-packages (from
etils[enp,epath,epy,etree]>=1.6.0; python_version < "3.11"-
>tensorflow_datasets) (6.4.5)
Requirement already satisfied: zipp in
/usr/local/lib/python3.10/dist-packages (from
etils[enp,epath,epy,etree]>=1.6.0; python_version < "3.11"-
>tensorflow_datasets) (3.20.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>tensorflow_datasets) (3.4.0)
```

```
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>tensorflow_datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>tensorflow_datasets) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>tensorflow_datasets) (2024.8.30)
Requirement already satisfied: six in
/usr/local/lib/python3.10/dist-packages (from promise-
>tensorflow_datasets) (1.16.0)
Requirement already satisfied: docstring-parser<1.0,>=0.15 in
/usr/local/lib/python3.10/dist-packages (from simple-parsing-
>tensorflow_datasets) (0.16)
```

```
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
```

Preparing the data named TFDS

37 category pet dataset with roughly 200 images for each class. The images have a large variations in scale, pose and lighting. All images have an associated ground truth annotation of breed, head ROI, and pixel level trimap segmentation.

##**For more information visit below link** Link: https://www.robots.ox.ac.uk/~vgg/data/pets/

```
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/tensorflow_datasets/oxford_iiit_pet/3.2.0...
```

{"model_id":"4cf6cfcb332346c7b53f7038c99d218a","version_major":2,"version_minor":0}

{"model_id":"d37dd00544da4e618eee0b3c52aa90a8","version_major":2,"version_minor":0}

{"model_id":"659416d841124224b92932fb8d16238f","version_major":2,"version_minor":0}

```
print(info)
```

```
print(dataset)
```

```
print(dataset["train"])
```

Pre-processing

```
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

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

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

    return input_image, input_mask
```

Loading the Training and Test Dataset

```
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

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

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)

print(train_dataset)
```

Splitting the dataset

```
BATCH_SIZE = 64
BUFFER_SIZE = 1000

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)

print(train_batches)
```

#Data Visualization

```
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()

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])
```

#Designing U-Net

```
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

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
```

```
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

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

unet_model = build_unet_model()

unet_model.summary()

Model: "U-Net"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━
━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)               ┃ Output Shape            ┃         Param
# ┃ Connected to            ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━
━━╇━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ input_layer_1              │ (None, 128, 128, 3)     │
0 │ -                       │
│ (InputLayer)               │                         │
                          │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ conv2d_19 (Conv2D)         │ (None, 128, 128, 64)    │
1,792 │ input_layer_1[0][0]     │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ conv2d_20 (Conv2D)         │ (None, 128, 128, 64)    │
36,928 │ conv2d_19[0][0]         │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ max_pooling2d_4            │ (None, 64, 64, 64)      │
0 │ conv2d_20[0][0]         │
│ (MaxPooling2D)             │                         │
                          │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ dropout_8 (Dropout)        │ (None, 64, 64, 64)      │
0 │ max_pooling2d_4[0][0]   │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ conv2d_21 (Conv2D)         │ (None, 64, 64, 128)     │
73,856 │ dropout_8[0][0]         │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ conv2d_22 (Conv2D)         │ (None, 64, 64, 128)     │
147,584 │ conv2d_21[0][0]         │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ max_pooling2d_5            │ (None, 32, 32, 128)     │
0 │ conv2d_22[0][0]         │
│ (MaxPooling2D)             │                         │
                          │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ dropout_9 (Dropout)        │ (None, 32, 32, 128)     │
0 │ max_pooling2d_5[0][0]   │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ conv2d_23 (Conv2D)         │ (None, 32, 32, 256)     │
295,168 │ dropout_9[0][0]         │
├────────────────────────────┼─────────────────────────┼──────────────
──┼─────────────────────────┤
│ conv2d_24 (Conv2D)         │ (None, 32, 32, 256)     │
590,080 │ conv2d_23[0][0]         │
```

```
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ max_pooling2d_6           │ (None, 16, 16, 256)     │
0 │ conv2d_24[0][0]         │
│ (MaxPooling2D)            │                         │
│                         │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ dropout_10 (Dropout)      │ (None, 16, 16, 256)     │
0 │ max_pooling2d_6[0][0]   │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ conv2d_25 (Conv2D)        │ (None, 16, 16, 512)     │
1,180,160 │ dropout_10[0][0]        │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ conv2d_26 (Conv2D)        │ (None, 16, 16, 512)     │
2,359,808 │ conv2d_25[0][0]         │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ max_pooling2d_7           │ (None, 8, 8, 512)       │
0 │ conv2d_26[0][0]         │
│ (MaxPooling2D)            │                         │
│                         │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ dropout_11 (Dropout)      │ (None, 8, 8, 512)       │
0 │ max_pooling2d_7[0][0]   │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ conv2d_27 (Conv2D)        │ (None, 8, 8, 1024)      │
4,719,616 │ dropout_11[0][0]        │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ conv2d_28 (Conv2D)        │ (None, 8, 8, 1024)      │
9,438,208 │ conv2d_27[0][0]         │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ conv2d_transpose_4        │ (None, 16, 16, 512)     │
4,719,104 │ conv2d_28[0][0]         │
│ (Conv2DTranspose)         │                         │
│                         │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ concatenate_4             │ (None, 16, 16, 1024)    │
0 │ conv2d_transpose_4[0]… │
│ (Concatenate)             │                         │
│ conv2d_26[0][0]         │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
│ dropout_12 (Dropout)      │ (None, 16, 16, 1024)    │
0 │ concatenate_4[0][0]     │
├───────────────────────────┼─────────────────────────┼──────────
──┼─────────────────────────┤
```

```
│ conv2d_29 (Conv2D)          │ (None, 16, 16, 512)       │
4,719,104 │ dropout_12[0][0]       │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ conv2d_30 (Conv2D)          │ (None, 16, 16, 512)       │
2,359,808 │ conv2d_29[0][0]        │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ conv2d_transpose_5          │ (None, 32, 32, 256)       │
1,179,904 │ conv2d_30[0][0]        │
│ (Conv2DTranspose)           │                           │
                         │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ concatenate_5               │ (None, 32, 32, 512)       │
0 │ conv2d_transpose_5[0]… │
│ (Concatenate)               │                           │
│ conv2d_24[0][0]          │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ dropout_13 (Dropout)        │ (None, 32, 32, 512)       │
0 │ concatenate_5[0][0]    │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ conv2d_31 (Conv2D)          │ (None, 32, 32, 256)       │
1,179,904 │ dropout_13[0][0]       │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ conv2d_32 (Conv2D)          │ (None, 32, 32, 256)       │
590,080 │ conv2d_31[0][0]        │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ conv2d_transpose_6          │ (None, 64, 64, 128)       │
295,040 │ conv2d_32[0][0]        │
│ (Conv2DTranspose)           │                           │
                         │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ concatenate_6               │ (None, 64, 64, 256)       │
0 │ conv2d_transpose_6[0]… │
│ (Concatenate)               │                           │
│ conv2d_22[0][0]          │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ dropout_14 (Dropout)        │ (None, 64, 64, 256)       │
0 │ concatenate_6[0][0]    │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ conv2d_33 (Conv2D)          │ (None, 64, 64, 128)       │
295,040 │ dropout_14[0][0]       │
├─────────────────────────────┼───────────────────────────┼───────────
──┼──────────────────────────┤
│ conv2d_34 (Conv2D)          │ (None, 64, 64, 128)       │
147,584 │ conv2d_33[0][0]        │
```

```
├──────────────────────────────┼──────────────────────────┼─────────────
──┼──────────────────────────┤
│ conv2d_transpose_7           │ (None, 128, 128, 64)     │
73,792 │ conv2d_34[0][0]           │
│ (Conv2DTranspose)            │                          │
│                           │
├──────────────────────────────┼──────────────────────────┼─────────────
──┼──────────────────────────┤
│ concatenate_7                │ (None, 128, 128, 128)    │
0 │ conv2d_transpose_7[0]… │
│ (Concatenate)                │                          │
│ conv2d_20[0][0]           │
├──────────────────────────────┼──────────────────────────┼─────────────
──┼──────────────────────────┤
│ dropout_15 (Dropout)         │ (None, 128, 128, 128)    │
0 │ concatenate_7[0][0]    │
├──────────────────────────────┼──────────────────────────┼─────────────
──┼──────────────────────────┤
│ conv2d_35 (Conv2D)           │ (None, 128, 128, 64)     │
73,792 │ dropout_15[0][0]          │
├──────────────────────────────┼──────────────────────────┼─────────────
──┼──────────────────────────┤
│ conv2d_36 (Conv2D)           │ (None, 128, 128, 64)     │
36,928 │ conv2d_35[0][0]           │
├──────────────────────────────┼──────────────────────────┼─────────────
──┼──────────────────────────┤
│ conv2d_37 (Conv2D)           │ (None, 128, 128, 3)      │
195 │ conv2d_36[0][0]        │
└──────────────────────────────┴──────────────────────────┴─────────────
──┴──────────────────────────┘

 Total params: 34,513,475 (131.66 MB)

 Trainable params: 34,513,475 (131.66 MB)

 Non-trainable params: 0 (0.00 B)
```

#**Training the Model**

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

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,
```

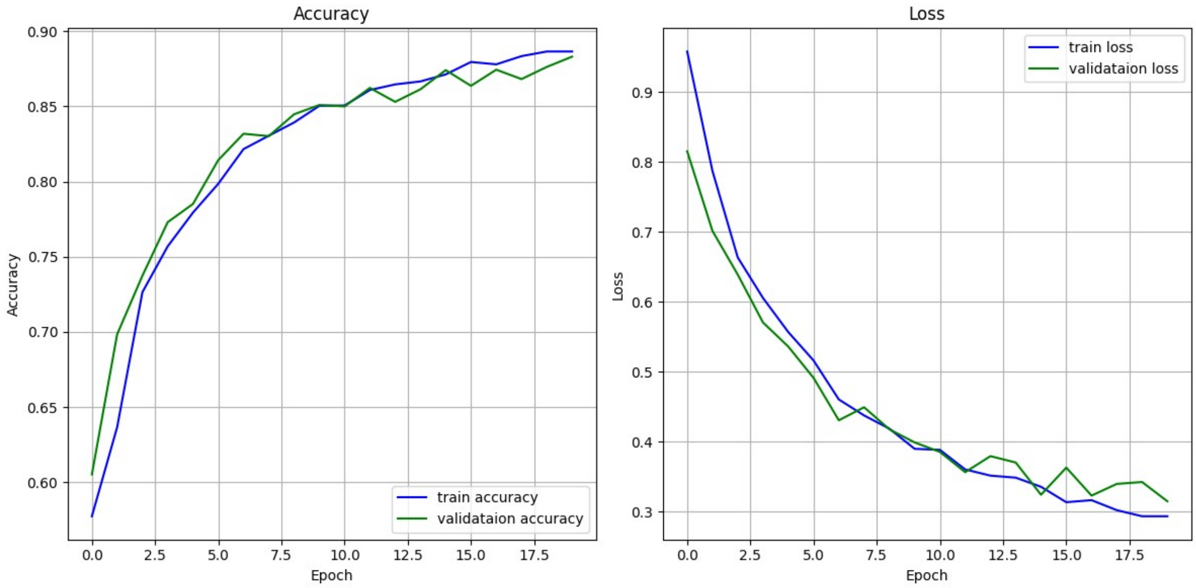
```
                                epochs=NUM_EPOCHS,
                                steps_per_epoch=STEPS_PER_EPOCH,
                                validation_steps=VALIDATION_STEPS,
                                validation_data=validation_batches)

Epoch 1/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 177s 1s/step - accuracy: 0.5558 - loss:
1.0744 - val_accuracy: 0.6053 - val_loss: 0.8153
Epoch 2/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 107s 886ms/step - accuracy: 0.6163 -
loss: 0.8125 - val_accuracy: 0.6985 - val_loss: 0.7014
Epoch 3/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 50s 885ms/step - accuracy: 0.7143 - loss:
0.6875 - val_accuracy: 0.7374 - val_loss: 0.6389
Epoch 4/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 50s 884ms/step - accuracy: 0.7514 - loss:
0.6160 - val_accuracy: 0.7730 - val_loss: 0.5703
Epoch 5/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 76s 1s/step - accuracy: 0.7730 - loss:
0.5706 - val_accuracy: 0.7851 - val_loss: 0.5359
Epoch 6/20

/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran
out of data; interrupting training. Make sure that your dataset or
generator can generate at least `steps_per_epoch * epochs` batches.
You may need to use the `.repeat()` function when building your
dataset.
  self.gen.throw(typ, value, traceback)

57/57 ━━━━━━━━━━━━━━━━━━━━ 51s 891ms/step - accuracy: 0.7923 - loss:
0.5303 - val_accuracy: 0.8142 - val_loss: 0.4911
Epoch 7/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 50s 885ms/step - accuracy: 0.8181 - loss:
0.4677 - val_accuracy: 0.8317 - val_loss: 0.4304
Epoch 8/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 50s 886ms/step - accuracy: 0.8255 - loss:
0.4473 - val_accuracy: 0.8301 - val_loss: 0.4490
Epoch 9/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 51s 888ms/step - accuracy: 0.8350 - loss:
0.4326 - val_accuracy: 0.8447 - val_loss: 0.4175
Epoch 10/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 49s 856ms/step - accuracy: 0.8505 - loss:
0.3907 - val_accuracy: 0.8508 - val_loss: 0.3987
Epoch 11/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 51s 894ms/step - accuracy: 0.8476 - loss:
0.3946 - val_accuracy: 0.8499 - val_loss: 0.3852
Epoch 12/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 50s 886ms/step - accuracy: 0.8593 - loss:
0.3633 - val_accuracy: 0.8622 - val_loss: 0.3562
Epoch 13/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 51s 890ms/step - accuracy: 0.8649 - loss:
0.3500 - val_accuracy: 0.8529 - val_loss: 0.3791
Epoch 14/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 50s 886ms/step - accuracy: 0.8634 - loss:
```

```
0.3568 - val_accuracy: 0.8613 - val_loss: 0.3700
Epoch 15/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 49s 851ms/step - accuracy: 0.8709 - loss:
0.3343 - val_accuracy: 0.8740 - val_loss: 0.3239
Epoch 16/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 51s 891ms/step - accuracy: 0.8756 - loss:
0.3221 - val_accuracy: 0.8636 - val_loss: 0.3627
Epoch 17/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 50s 886ms/step - accuracy: 0.8771 - loss:
0.3171 - val_accuracy: 0.8743 - val_loss: 0.3227
Epoch 18/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 51s 887ms/step - accuracy: 0.8815 - loss:
0.3053 - val_accuracy: 0.8680 - val_loss: 0.3394
Epoch 19/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 51s 889ms/step - accuracy: 0.8856 - loss:
0.2943 - val_accuracy: 0.8761 - val_loss: 0.3421
Epoch 20/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 49s 854ms/step - accuracy: 0.8848 - loss:
0.2969 - val_accuracy: 0.8830 - val_loss: 0.3145
```

#**Learning Curves**

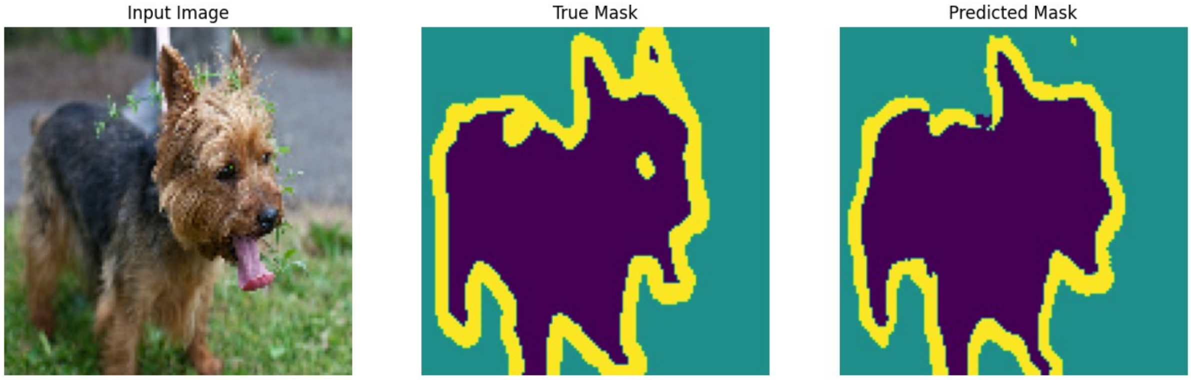
```
def display_learning_curves(history):
    acc = history.history["accuracy"]
    val_acc = history.history["val_accuracy"]

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]

    epochs_range = range(NUM_EPOCHS)

    fig = plt.figure(figsize=(12,6))

    plt.subplot(1,2,1)
    plt.plot(epochs_range, acc, 'b', label="train accuracy")
    plt.plot(epochs_range, val_acc,'g', label="validataion
accuracy")
    plt.title("Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.grid()
    plt.legend(loc="lower right")

    plt.subplot(1,2,2)
    plt.plot(epochs_range, loss, 'b', label="train loss")
    plt.plot(epochs_range, val_loss,'g', label="validataion loss")
    plt.title("Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.grid()
    plt.legend(loc="upper right")
```

```
    fig.tight_layout()
    plt.show()

# Display learning curves
display_learning_curves(unet_model.history)
```

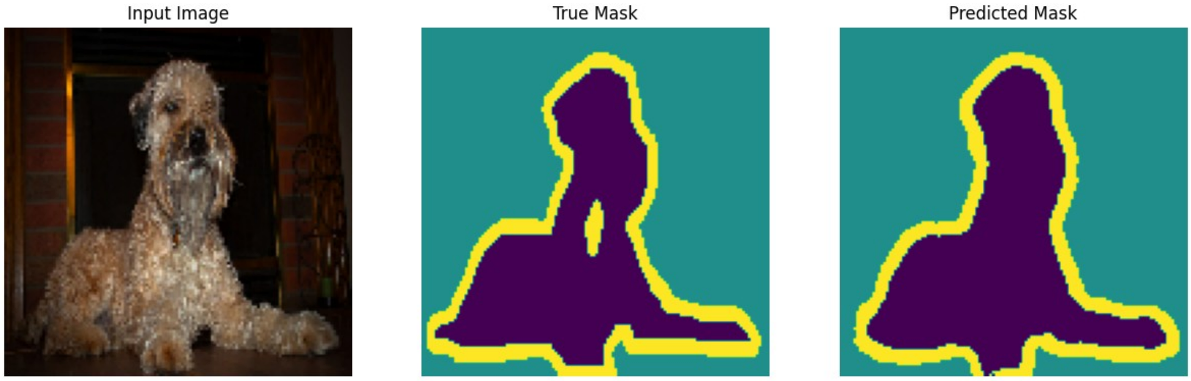


#**Prediction**

```
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):
      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, ...]))])

count = 0
for i in test_batches:
    count +=1
print("number of batches:", count)

number of batches: 11

show_predictions(test_batches.skip(5), 3)

2/2 ━━━━━━━━━━━━━━━━━━━━ 18s 119ms/step
```




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
```




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step
```