PRACTICAL 6 : Implement a deep neural network (Ex: UNet/ SegNet) for Image Segmentation task

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_datasets as tfds
# import tensorflow_datasets as my_custom_name
import matplotlib.pyplot as plt
import numpy as np
```

```
import os
import requests
import tarfile
import shutil

# URL to download the dataset archive
url = 'http://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz'

# Directory to save the downloaded dataset
dataset_dir = '/workspace/All/DNN_Lab/Data/Oxford/oxford_iiit_pet'

# Create the directory if it doesn't exist
os.makedirs(dataset_dir, exist_ok=True)

# Download the dataset archive
r = requests.get(url)
with open(os.path.join(dataset_dir, 'images.tar.gz'), 'wb') as f:
    f.write(r.content)

# Extract the dataset archive
with tarfile.open(os.path.join(dataset_dir, 'images.tar.gz'), 'r:gz') as tar:
    tar.extractall(path=dataset_dir)

# After extraction, the dataset will be available in dataset_dir/images
# You can then proceed to load and preprocess the dataset using other libraries like numpy or pandas
```

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. Link: https://www.robots.ox.ac.uk/~vgg/data/pets/

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

```
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/tmpi1s8r38ltfds'),
        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",
```

```
        }""",
    )
```

```
print(dataset)
```

```
{'train': <_PrefetchDataset element_spec={'file_name': TensorSpec(shape=(), dtype=tf.string, name=None), 'image': TensorSpec(shape=
```

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

```
<_PrefetchDataset element_spec={'file_name': TensorSpec(shape=(), dtype=tf.string, name=None), 'image': TensorSpec(shape=(None, None
```

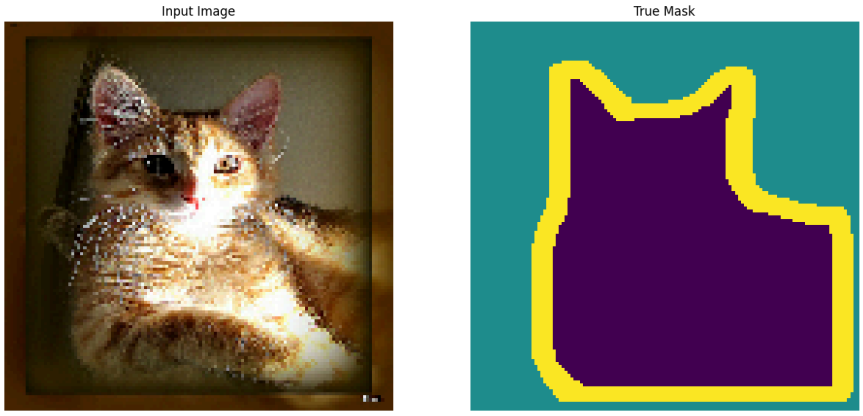
```
def resize(input_image, input_mask):
    input_image = tf.image.resize(input_image, (128, 128), method="nearest")
    input_mask = tf.image.resize(input_mask, (128, 128), method="nearest")
    return input_image, input_mask


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


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)
```

```
<_ParallelMapDataset element_spec=(TensorSpec(shape=(128, 128, 3), dtype=tf.float32, name=None), TensorSpec(shape=(128, 128, 1), dty
```

```
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)
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 128, 128, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None, 128, 128
```

```
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

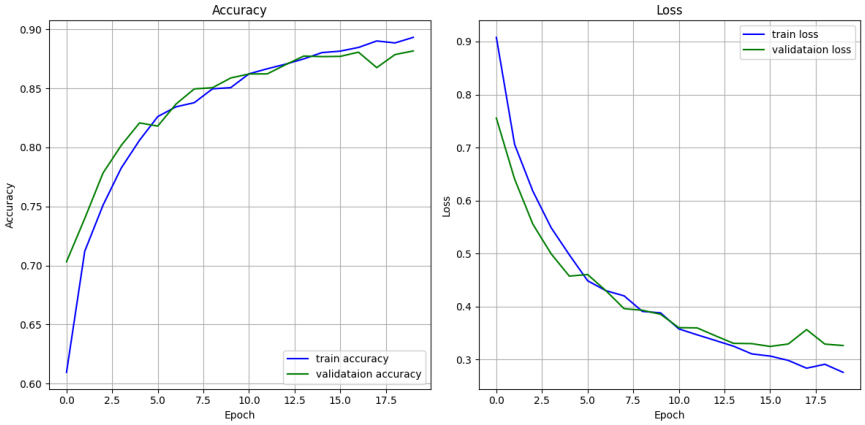
```
def double_conv_block(x, n_filters):
    x = layers.Conv2D(n_filters, 3, padding = "same", activation = "relu", kernel_initializer = "he_normal")(x)
    x = layers.Conv2D(n_filters, 3, padding = "same", activation = "relu", kernel_initializer = "he_normal")(x)
    return x


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

 # encoder: contracting path - downsample
    f1, p1 = downsample_block(inputs, 64)          # 1 - downsample
```

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

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

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

    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()
```

```
     dropout_3 (Dropout)         (None, 8, 8, 512)           0         ['max_pooling2d_3[0][0]']

     conv2d_8 (Conv2D)           (None, 8, 8, 1024)          4719616   ['dropout_3[0][0]']

     conv2d_9 (Conv2D)           (None, 8, 8, 1024)          9438208   ['conv2d_8[0][0]']

     conv2d_transpose (Conv2DTr  (None, 16, 16, 512)         4719104   ['conv2d_9[0][0]']
     anspose)

     concatenate (Concatenate)   (None, 16, 16, 1024)        0         ['conv2d_transpose[0][0]',
                                                                        'conv2d_7[0][0]']

     dropout_4 (Dropout)         (None, 16, 16, 1024)        0         ['concatenate[0][0]']

     conv2d_10 (Conv2D)          (None, 16, 16, 512)         4719104   ['dropout_4[0][0]']

     conv2d_11 (Conv2D)          (None, 16, 16, 512)         2359808   ['conv2d_10[0][0]']

     conv2d_transpose_1 (Conv2D  (None, 32, 32, 256)         1179904   ['conv2d_11[0][0]']
     Transpose)

     concatenate_1 (Concatenate  (None, 32, 32, 512)         0         ['conv2d_transpose_1[0][0]',
     )                                                                  'conv2d_5[0][0]']

     dropout_5 (Dropout)         (None, 32, 32, 512)         0         ['concatenate_1[0][0]']

     conv2d_12 (Conv2D)          (None, 32, 32, 256)         1179904   ['dropout_5[0][0]']

     conv2d_13 (Conv2D)          (None, 32, 32, 256)         590080    ['conv2d_12[0][0]']

     conv2d_transpose_2 (Conv2D  (None, 64, 64, 128)         295040    ['conv2d_13[0][0]']
     Transpose)

     concatenate_2 (Concatenate  (None, 64, 64, 256)         0         ['conv2d_transpose_2[0][0]',
     )                                                                  'conv2d_3[0][0]']

     dropout_6 (Dropout)         (None, 64, 64, 256)         0         ['concatenate_2[0][0]']

     conv2d_14 (Conv2D)          (None, 64, 64, 128)         295040    ['dropout_6[0][0]']

     conv2d_15 (Conv2D)          (None, 64, 64, 128)         147584    ['conv2d_14[0][0]']

     conv2d_transpose_3 (Conv2D  (None, 128, 128, 64)        73792     ['conv2d_15[0][0]']
     Transpose)

     concatenate_3 (Concatenate  (None, 128, 128, 128)       0         ['conv2d_transpose_3[0][0]',
     )                                                                  'conv2d_1[0][0]']

     dropout_7 (Dropout)         (None, 128, 128, 128)       0         ['concatenate_3[0][0]']

     conv2d_16 (Conv2D)          (None, 128, 128, 64)        73792     ['dropout_7[0][0]']

     conv2d_17 (Conv2D)          (None, 128, 128, 64)        36928     ['conv2d_16[0][0]']

     conv2d_18 (Conv2D)          (None, 128, 128, 3)         195       ['conv2d_17[0][0]']

    ==================================================================================================
```

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

```
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)


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)
```




```
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 [==============================] - 1s 9ms/step
```

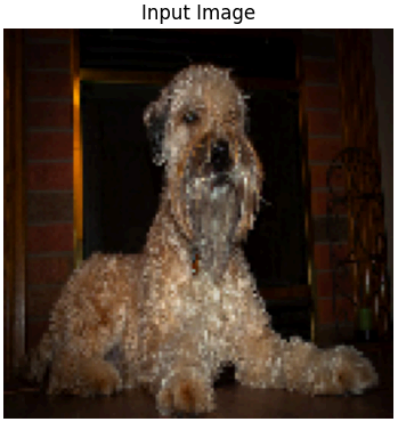
Input Image

True Mask

Predicted Mask

```
2/2 [==============================] - 0s 126ms/step
```

Input Image



True Mask

Predicted Mask

```
2/2 [==============================] - 0s 126ms/step
```

Input Image

True Mask

Predicted Mask