```
!pip install segmentation_models

import tensorflow as tf
import segmentation_models as sm
import glob
import cv2
import os
import numpy as np
from matplotlib import pyplot as plt
import keras
from sklearn.model_selection import train_test_split
```

- Provide environment variable SM_FRAMEWORK=keras / SM_FRAMEWORK=tf.keras before import segmentation_models
- Change framework sm.set_framework('keras') / sm.set_framework('tf.keras')

```
os.environ["SM_FRAMEWORK"] = "tf.keras"
sm.set_framework('tf.keras')
keras.backend.set_image_data_format('channels_last')
```

Data Preprocessing Pipeline

```
H = 256 # height of image
W = 256 # width of image

'''This function is used to return the list of path for images and
masks in
sorted order from the given directory respectively.'''
# function to return list of image paths and mask paths
def process_data(IMG_DIR, MASK_DIR):
    images = [os.path.join(IMG_DIR, x) for x in
sorted(os.listdir(IMG_DIR))]
    masks = [os.path.join(MASK_DIR, x) for x in
sorted(os.listdir(MASK_DIR))]

    return images, masks

'''This function is used to return splitted list of images and
corresponding
mask paths in train and test by providing test size.'''
# function to load data and train test split
def load_data(IMG_DIR, MASK_DIR):
    X, y = process_data(IMG_DIR, MASK_DIR)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test

'''This function is used to read images. It takes image path as input.
```

```
After reading image it is resized by width and height provide
above(256 x 256).
Next normalization is done by dividing each values with 255. And the
result is returned.'''
# function to read image
def read_image(x):
    x = cv2.imread(x, cv2.IMREAD_COLOR)
    x = cv2.resize(x, (W, H))
    x = x / 255.0
    x = x.astype(np.float32)
    return x

'''This function is used to read masks.'''
# function to read mask
def read_mask(x):
    x = cv2.imread(x, cv2.IMREAD_GRAYSCALE)
    x = cv2.resize(x, (W, H))
    x = x.astype(np.int32)
    return x

'''This function is used to generate tensorflow data pipeline.
The tensorflow data pipeline is mapped to function ‘preprocess’ .'''
# function for tensorflow dataset pipeline
def tf_dataset(x, y, batch=8):
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.shuffle(buffer_size=5000)
    dataset = dataset.map(preprocess)
    dataset = dataset.batch(batch)
    dataset = dataset.repeat()
    dataset = dataset.prefetch(2)
    return dataset

'''This function takes image and mask path.
It reads the image and mask as provided by paths.
Mask is one hot encoded for multi class segmentation (here 4
class).'''
# function to read image and mask amd create one hot encoding for mask
def preprocess(x, y):
    def f(x, y):
        x = x.decode()
        y = y.decode()

        image = read_image(x)
        mask = read_mask(y)

        return image, mask

    image, mask = tf.numpy_function(f, [x, y], [tf.float32, tf.int32])
    mask = tf.one_hot(mask, 4, dtype=tf.int32)
```

```
    image.set_shape([H, W, 3])
    mask.set_shape([H, W, 4])

    return image, mask
```

Load the dataset

```
'''RENDER_IMAGE_DIR_PATH: 'Path of image directory'
GROUND_MASK_DIR_PATH: 'Path of mask directory'

Here load_data function is called. This will load the dataset paths
and
split it into X_train, X_test, y_train, y_test '''

RENDER_IMAGE_DIR_PATH = '../input/artificial-lunar-rocky-landscape-
dataset/images/render'
GROUND_MASK_DIR_PATH = '../input/artificial-lunar-rocky-landscape-
dataset/images/clean'

X_train, X_test, y_train, y_test = load_data(RENDER_IMAGE_DIR_PATH,
GROUND_MASK_DIR_PATH)
print(f"Dataset:\n Train: {len(X_train)} \n Test: {len(X_test)}")
```

Generate tensorflow data pipeline

```
batch_size = 8

'''Here the tf_dataset function is called will generate the tensorflow
data pipeline.'''
# calling tf_dataset
train_dataset = tf_dataset(X_train, y_train, batch=batch_size)
valid_dataset = tf_dataset(X_test, y_test, batch=batch_size)
```

Creating U-net Architecture

```
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization,
Activation, MaxPool2D, UpSampling2D, Concatenate
from tensorflow.keras.models import Model

'''conv_block it is used to create one block with two convolution
layer
followed by BatchNormalization and activation function relu.
If the pooling is required then Maxpool2D is applied and return it
else not.'''
# function to create convolution block
def conv_block(inputs, filters, pool=True):
    x = Conv2D(filters, 3, padding="same")(inputs)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
```

```
    x = Conv2D(filters, 3, padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    if pool == True:
        p = MaxPool2D((2, 2))(x)
        return x, p
    else:
        return x

'''build_unet it is used to create the U-net architecture.'''
# function to build U-net
def build_unet(shape, num_classes):
    inputs = Input(shape)

    """ Encoder """
    x1, p1 = conv_block(inputs, 16, pool=True)
    x2, p2 = conv_block(p1, 32, pool=True)
    x3, p3 = conv_block(p2, 48, pool=True)
    x4, p4 = conv_block(p3, 64, pool=True)
    x5, p5 = conv_block(p3, 64, pool=True)
    x6, p6 = conv_block(p3, 64, pool=True)

    """ Bridge """
    b1 = conv_block(p4, 128, pool=False)

    """ Decoder """
    u1 = UpSampling2D((2, 2), interpolation="bilinear")(b1)
    c1 = Concatenate()([u1, x4])
    x5 = conv_block(c1, 64, pool=False)

    u2 = UpSampling2D((2, 2), interpolation="bilinear")(x5)
    c2 = Concatenate()([u2, x3])
    x6 = conv_block(c2, 48, pool=False)

    u3 = UpSampling2D((2, 2), interpolation="bilinear")(x6)
    c3 = Concatenate()([u3, x2])
    x7 = conv_block(c3, 32, pool=False)

    u4 = UpSampling2D((2, 2), interpolation="bilinear")(x7)
    c4 = Concatenate()([u4, x1])
    x8 = conv_block(c4, 16, pool=False)

    """ Output layer """
    output = Conv2D(num_classes, 1, padding="same",
activation="softmax")(x8)

    return Model(inputs, output)
```

```
# calling build_unet function
import segmentation_models as sm

BACKBONE = 'efficientnetb7'
input_shape = (256, 256, 3)
n_classes = 4
activation = 'softmax'

model = sm.Unet(backbone_name = BACKBONE,
                input_shape = input_shape,
                classes = n_classes,
                activation = activation,
                encoder_weights = 'imagenet')

model.summary()
```

Load model and compile

```
lr = 1e-4
batch_size = 16
epochs = 8

# metrics for result validation
metrics = [sm.metrics.IOUScore(threshold=0.5),
sm.metrics.FScore(threshold=0.5)]

# compiling the model
model.compile(loss = 'categorical_crossentropy',
              optimizer = tf.keras.optimizers.Adam(lr),
              metrics = metrics)

train_steps = len(X_train)//batch_size
valid_steps = len(X_test)//batch_size

# importing libraries
from tensorflow.keras.callbacks import ModelCheckpoint,
ReduceLROnPlateau, EarlyStopping, TensorBoard
from segmentation_models.metrics import iou_score
import datetime, os

""" Hyperparameters """
img_shape = (256, 256, 3)
num_classes = 4
lr = 1e-4
batch_size = 16
epochs = 8

""" Model """
model = build_unet(img_shape, num_classes)
model.compile(loss="categorical_crossentropy",
```

```
              optimizer=tf.keras.optimizers.Adam(lr),
              metrics=[iou_score])


train_steps = len(X_train)//batch_size
valid_steps = len(X_test)//batch_size


callbacks = [

tf.keras.callbacks.ModelCheckpoint(filepath=f'models/lunarModel_{curre
nt_datetime}.h5',
                          monitor='val_iou_score', verbose=0,
                          mode='max', save_best_model=True),

        tf.keras.callbacks.ReduceLROnPlateau(monitor="val_iou_score",
mode='max', patience=4,
                             factor=0.1, verbose=0, min_lr=1e-6),

        tf.keras.callbacks.EarlyStopping(monitor="val_iou_score",
patience=5, verbose=0, mode='max'),


tf.keras.callbacks.TensorBoard(f'models/logs_{current_datetime}')
    ]
```

Train model

```
'''model.fit is used to train the model'''
model_history = model.fit(train_dataset,
        steps_per_epoch=train_steps,
        validation_data=valid_dataset,
        validation_steps=valid_steps,
        epochs=epochs,
    )
```