```
# import system libs
import os
import time
import random
import pathlib
import itertools
from glob import glob
from tqdm import tqdm_notebook, tnrange

# import data handling tools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
%matplotlib inline
from skimage.color import rgb2gray
from skimage.morphology import label
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from skimage.io import imread, imshow, concatenate_images

# import Deep learning Libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model, load_model, save_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.layers import Input, Activation, BatchNormalization, Dropout, Lambda, Conv2D,

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

print ('modules loaded')
```

```
modules loaded
```

```
IMG_CHANNELS, IMG_WIDTH, IMG_HEIGHT = 3, 512, 512
print ('Done')
```

```
Done
```

```
X = next(os.walk('/content/sample_data/images'))[2]
y = next(os.walk('/content/sample_data/masks'))[2]
print ('Done')
```

```
Done
```

```
X_ids = X[:-10]
y_ids = y[:-10]
print ('Done')
```

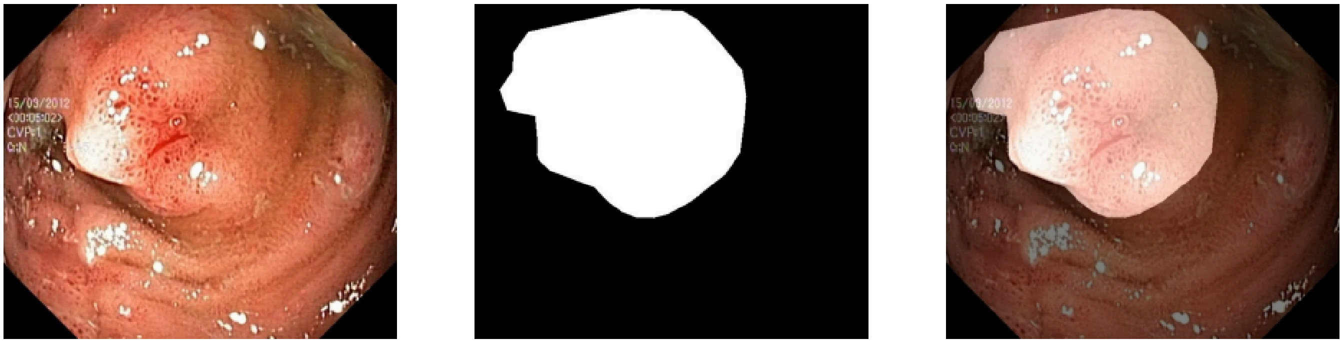
Done

```
X_tr = np.zeros((len(X_ids), 512, 512, 3), dtype=np.float32)
y_tr = np.zeros((len(y_ids), 512, 512, 1), dtype=np.uint8)
print ('Done')
```

Done

```
X_train = np.zeros((len(X_ids), 256, 256, 3), dtype=np.float32)
y_train = np.zeros((len(y_ids), 256, 256, 1), dtype=np.float32)

for n, id_ in enumerate(X_ids):
    image = tf.keras.preprocessing.image.load_img(f'/kaggle/input/kvasirseg/Kvasir-SEG/Kvasir-SEG/ir
    input_arr = tf.keras.preprocessing.image.img_to_array(image)[90:450,150:406]
   # input_arr = np.expand_dims(input_arr, axis=0)
    image = tf.keras.preprocessing.image.array_to_img(input_arr, ).resize((256, 256))
    X_train[n] = np.array(image)

for n, id_ in enumerate(y_ids):
    mask = tf.keras.preprocessing.image.load_img(f'/kaggle/input/kvasirseg/Kvasir-SEG/Kvasir-SEG/mas
                                              target_size=(IMG_HEIGHT, IMG_WIDTH), color_mode="g
    mask_arr = tf.keras.preprocessing.image.img_to_array(mask)[90:450,150:406]
   # mask_arr = np.expand_dims(mask_arr, axis=0)
    mask = tf.keras.preprocessing.image.array_to_img(mask_arr).resize((256, 256))
    #y_train[n] = np.array(image)[:, :, np.newaxis]
    y_train[n] = np.array(mask, dtype=np.float32)[:, :, np.newaxis] / 255.0  # Ensure masks are 0 or
```

```
plt.figure(figsize=(12, 8))
img=cv2.imread('/content/sample_data/images/cju0qkwl35piu0993l0dewei2.jpg')
msk=cv2.imread('/content/sample_data/masks/cju0qkwl35piu0993l0dewei2.jpg')

plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis(False)

plt.subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(msk, cv2.COLOR_BGR2RGB))
plt.axis(False)

plt.subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.imshow(cv2.cvtColor(msk, cv2.COLOR_BGR2RGB),alpha=0.5)
plt.axis(False)
plt.show()
```



```
def unet(input_size=(256, 256, 3)):
    inputs = Input(input_size)

    # First DownConvolution / Encoder Leg will begin, so start with Conv2D
    conv1 = Conv2D(filters=64, kernel_size=(3, 3), padding="same")(inputs)
    bn1 = Activation("relu")(conv1)
    conv1 = Conv2D(filters=64, kernel_size=(3, 3), padding="same")(bn1)
    bn1 = BatchNormalization(axis=3)(conv1)
    bn1 = Activation("relu")(bn1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(bn1)

    conv2 = Conv2D(filters=128, kernel_size=(3, 3), padding="same")(pool1)
    bn2 = Activation("relu")(conv2)
    conv2 = Conv2D(filters=128, kernel_size=(3, 3), padding="same")(bn2)
    bn2 = BatchNormalization(axis=3)(conv2)
    bn2 = Activation("relu")(bn2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(bn2)

    conv3 = Conv2D(filters=256, kernel_size=(3, 3), padding="same")(pool2)
    bn3 = Activation("relu")(conv3)
    conv3 = Conv2D(filters=256, kernel_size=(3, 3), padding="same")(bn3)
    bn3 = BatchNormalization(axis=3)(conv3)
    bn3 = Activation("relu")(bn3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(bn3)

    conv4 = Conv2D(filters=512, kernel_size=(3, 3), padding="same")(pool3)
    bn4 = Activation("relu")(conv4)
    conv4 = Conv2D(filters=512, kernel_size=(3, 3), padding="same")(bn4)
    bn4 = BatchNormalization(axis=3)(conv4)
    bn4 = Activation("relu")(bn4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(bn4)

    conv5 = Conv2D(filters=1024, kernel_size=(3, 3), padding="same")(pool4)
    bn5 = Activation("relu")(conv5)
    conv5 = Conv2D(filters=1024, kernel_size=(3, 3), padding="same")(bn5)
    bn5 = BatchNormalization(axis=3)(conv5)
    bn5 = Activation("relu")(bn5)

    """ Now UpConvolution / Decoder Leg will begin, so start with Conv2DTranspose
    The gray arrows (in the above image) indicate the skip connections that concatenate the encode
    """ After every concatenation we again apply two consecutive regular convolutions so that the

    up6 = concatenate([Conv2DTranspose(512, kernel_size=(2, 2), strides=(2, 2), padding="same")(bn
    conv6 = Conv2D(filters=512, kernel_size=(3, 3), padding="same")(up6)
```

```
    bn6 = Activation("relu")(conv6)
    conv6 = Conv2D(filters=512, kernel_size=(3, 3), padding="same")(bn6)
    bn6 = BatchNormalization(axis=3)(conv6)
    bn6 = Activation("relu")(bn6)

    up7 = concatenate([Conv2DTranspose(256, kernel_size=(2, 2), strides=(2, 2), padding="same")(bn
    conv7 = Conv2D(filters=256, kernel_size=(3, 3), padding="same")(up7)
    bn7 = Activation("relu")(conv7)
    conv7 = Conv2D(filters=256, kernel_size=(3, 3), padding="same")(bn7)
    bn7 = BatchNormalization(axis=3)(conv7)
    bn7 = Activation("relu")(bn7)

    up8 = concatenate([Conv2DTranspose(128, kernel_size=(2, 2), strides=(2, 2), padding="same")(bn
    conv8 = Conv2D(filters=128, kernel_size=(3, 3), padding="same")(up8)
    bn8 = Activation("relu")(conv8)
    conv8 = Conv2D(filters=128, kernel_size=(3, 3), padding="same")(bn8)
    bn8 = BatchNormalization(axis=3)(conv8)
    bn8 = Activation("relu")(bn8)

    up9 = concatenate([Conv2DTranspose(64, kernel_size=(2, 2), strides=(2, 2), padding="same")(bn8
    conv9 = Conv2D(filters=64, kernel_size=(3, 3), padding="same")(up9)
    bn9 = Activation("relu")(conv9)
    conv9 = Conv2D(filters=64, kernel_size=(3, 3), padding="same")(bn9)
    bn9 = BatchNormalization(axis=3)(conv9)
    bn9 = Activation("relu")(bn9)

    conv10 = Conv2D(filters=1, kernel_size=(1, 1), activation="sigmoid")(bn9)

    return Model(inputs=[inputs], outputs=[conv10])


def dice_coef(y_true, y_pred, smooth=100):
    y_true_flatten = K.flatten(y_true)
    y_pred_flatten = K.flatten(y_pred)

    intersection = K.sum(y_true_flatten * y_pred_flatten)
    union = K.sum(y_true_flatten) + K.sum(y_pred_flatten)
    return (2 * intersection + smooth) / (union + smooth)

# function to create dice loss
def dice_loss(y_true, y_pred, smooth=100):
    return -dice_coef(y_true, y_pred, smooth)

# function to create iou coefficient
def iou_coef(y_true, y_pred, smooth=100):
    intersection = K.sum(y_true * y_pred)
    sum = K.sum(y_true + y_pred)
    iou = (intersection + smooth) / (sum - intersection + smooth)
    return iou


model = unet()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', iou_coef, dice_co
# model.compile(Adamax(learning_rate= 0.001), loss= dice_loss, metrics= ['accuracy', iou_coef, dic

model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 256, 256, 3)	0	-
conv2d (Conv2D)	(None, 256, 256, 64)	1,792	input_layer[0][0]
activation (Activation)	(None, 256, 256, 64)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 64)	36,928	activation[0][0]
batch_normalization (BatchNormalization)	(None, 256, 256, 64)	256	conv2d_1[0][0]
activation_1 (Activation)	(None, 256, 256, 64)	0	batch_normalization[0
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0	activation_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 128)	73,856	max_pooling2d[0][0]
activation_2 (Activation)	(None, 128, 128, 128)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 128)	147,584	activation_2[0][0]
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 128)	512	conv2d_3[0][0]
activation_3 (Activation)	(None, 128, 128, 128)	0	batch_normalization_1
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0	activation_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 256)	295,168	max_pooling2d_1[0][0]
activation_4 (Activation)	(None, 64, 64, 256)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 256)	590,080	activation_4[0][0]
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 256)	1,024	conv2d_5[0][0]
activation_5 (Activation)	(None, 64, 64, 256)	0	batch_normalization_2
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 256)	0	activation_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 512)	1,180,160	max_pooling2d_2[0][0]
activation_6 (Activation)	(None, 32, 32, 512)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 512)	2,359,808	activation_6[0][0]

Start coding or generate with AI.

Layer (type)	Output Shape	Param #	Connected to
batch_normalization_3	(None, 32, 32, 512)	2,048	conv2d_7[0][0]

Start coding or generate with AI.

Layer (type)	Output Shape	Param #	Connected to
activation_7 (Activation)	(None, 32, 32, 512)	0	batch_normalization_3

```python
def plot_training(hist):
    '''
    This function take training model and plot history of accuracy and losses with the best epoch
    '''

    # Define needed variables
    tr_acc = hist.history['accuracy']
    tr_iou = hist.history['iou_coef']
    tr_dice = hist.history['dice_coef']
    tr_loss = hist.history['loss']

    val_acc = hist.history['val_accuracy']
    val_iou = hist.history['val_iou_coef']
    val_dice = hist.history['val_dice_coef']
    val_loss = hist.history['val_loss']

    index_acc = np.argmax(val_acc)
    acc_highest = val_acc[index_acc]
    index_iou = np.argmax(iou_coef)
    iou_highest = val_iou[index_iou]
    index_dice = np.argmax(dice_coef)
    dice_highest = val_dice[index_dice]
    index_loss = np.argmin(val_loss)
    val_lowest = val_loss[index_loss]

    Epochs = [i+1 for i in range(len(tr_acc))]

    acc_label = f'best epoch= {str(index_acc + 1)}'
    iou_label = f'best epoch= {str(index_iou + 1)}'
    dice_label = f'best epoch= {str(index_dice + 1)}'
    loss_label = f'best epoch= {str(index_loss + 1)}'

    # Plot training history
    plt.figure(figsize= (20, 20))
    plt.style.use('fivethirtyeight')

    # Training Accuracy
    plt.subplot(2, 2, 1)
    plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
    plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
    plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Training IoU
    plt.subplot(2, 2, 2)
    plt.plot(Epochs, tr_iou, 'r', label= 'Training IoU')
    plt.plot(Epochs, val_iou, 'g', label= 'Validation IoU')
    plt.scatter(index_iou + 1 , iou_highest, s= 150, c= 'blue', label= iou_label)
    plt.title('Training and Validation IoU Coefficient')
    plt.xlabel('Epochs')
    plt.ylabel('IoU')
    plt.legend()

    # Training Dice
    plt.subplot(2, 2, 3)
    plt.plot(Epochs, tr_dice, 'r', label= 'Training Dice')
```

```
    plt.plot(Epochs, val_dice, 'g', label= 'Validation Dice')
    plt.scatter(index_dice + 1 , dice_highest, s= 150, c= 'blue', label= dice_label)
    plt.title('Training and Validation Dice Coefficient')
    plt.xlabel('Epochs')
    plt.ylabel('Dice')
    plt.legend()

    # Training Loss
    plt.subplot(2, 2, 4)
    plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
    plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
    plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout
    plt.show()
```

```
img = tf.keras.preprocessing.image.load_img(r"/content/sample_data/images/cju0qx73cjw570799j4n5cjz
input_array = tf.keras.preprocessing.image.img_to_array(img)
input_array = np.array([input_array])  # Convert single image to a batch.
predictions = model.predict(input_array)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

```
plt.figure(figsize=(15, 12))
plt.subplot(1, 3, 1)
plt.imshow(np.squeeze(img))
plt.axis(False)
plt.subplot(1, 3, 2)
plt.imshow(np.squeeze(predictions))
plt.axis(False)
plt.subplot(1, 3, 3)
plt.imshow(np.squeeze(img))
# plt.imshow(msk,alpha=0.5)
plt.imshow(np.squeeze(predictions), alpha=0.5)
plt.axis(False)
plt.show()
```

```
subject = 'Kvasir Segmentation'
save_path = './'


save_id = str(f'{subject} model.h5')
model_save_loc = os.path.join(save_path, save_id)
model.save(model_save_loc)
print(f'model was saved as {model_save_loc}')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.sav
model was saved as ./Kvasir Segmentation model.h5
```

```
segmentation_model = load_model('Kvasir Segmentation model.h5', compile=False)
segmentation_model.compile(Adamax(learning_rate=0.001), loss=dice_loss, metrics=['accuracy', iou_c
```

```
# Preprocess the image for segmentation
def preprocess_segmentation_image(image):
    img_array = tf.keras.preprocessing.image.img_to_array(image)
    image = cv2.cvtColor(img_array, cv2.COLOR_RGB2BGR)
    img = cv2.resize(image, (256, 256))
    img = img / 255.0
    img = img[np.newaxis, :, :, :]
```

```
from PIL import Image
```

```
import numpy as np
import cv2
from PIL import Image

# Preprocess function to handle the image correctly
def preprocess_segmentation_image(image):
    # Resize to 256x256
    img_resized = cv2.resize(image, (256, 256))

    # Normalize to [0, 1]
    img_resized = img_resized / 255.0

    # Add a batch dimension (for TensorFlow model input)
    img_resized = np.expand_dims(img_resized, axis=0)  # Shape: (1, 256, 256, 3)

    return img_resized

# Read the image using PIL
image_file = r"/content/sample_data/images/cju1h89h6xbnx08352k2790o9.jpg"
pil_image = Image.open(image_file).convert('RGB')
open_cv_image = np.array(pil_image)  # Convert to OpenCV format (NumPy array)

# Convert RGB to BGR (if needed)
open_cv_image = open_cv_image[:, :, ::-1].copy()

# Perform preprocessing (resizing, normalization, and adding batch dimension)
img = preprocess_segmentation_image(open_cv_image)

# Check the shape of the preprocessed image
print(f"Preprocessed image shape: {img.shape}")  # Should be (1, 256, 256, 3)
```

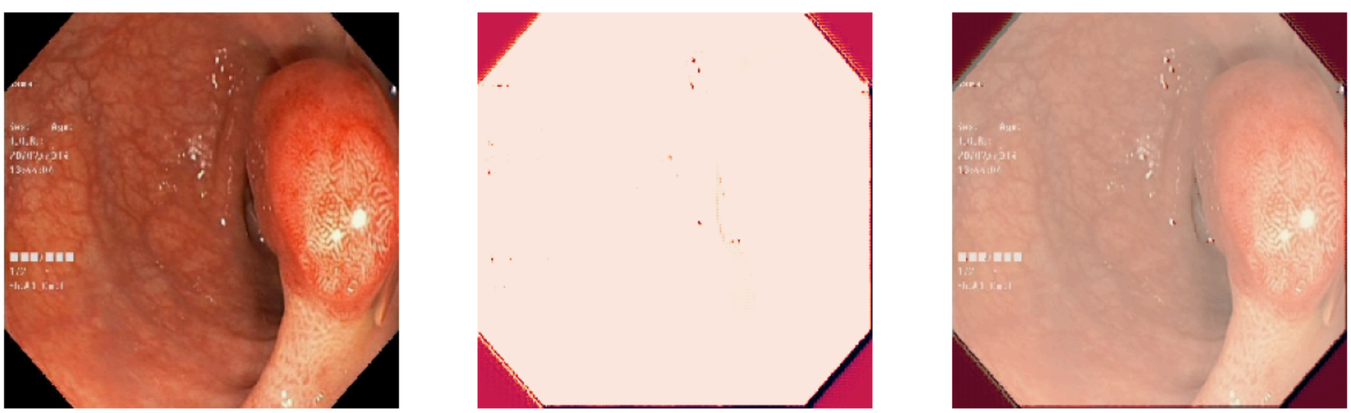
```
# Perform image segmentation using the pre-trained model
segmented_image = segmentation_model.predict(img)

# Print segmentation results (for debugging)
print("Segmentation completed.")
```

```
Preprocessed image shape: (1, 256, 256, 3)
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Segmentation completed.
```

```
img = tf.keras.preprocessing.image.load_img(r"/content/sample_data/cju0s2a9ekvms080138tjjpxr.jpg",
input_array = tf.keras.preprocessing.image.img_to_array(img)
input_array = np.array([input_array])  # Convert single image to a batch.
predictions = segmentation_model.predict(input_array)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

```
plt.figure(figsize=(15, 12))
plt.subplot(1, 3, 1)
plt.imshow(np.squeeze(img))
plt.axis(False)
plt.subplot(1, 3, 2)
plt.imshow(np.squeeze(predictions))
plt.axis(False)
plt.subplot(1, 3, 3)
plt.imshow(np.squeeze(img))
# plt.imshow(msk,alpha=0.5)
plt.imshow(np.squeeze(predictions), alpha=0.5)
plt.axis(False)
plt.show()
```



```
predicted_mask = np.squeeze(predictions)  # Remove batch dimension if needed (shape becomes 256x25

# Apply a threshold to convert the grayscale mask to binary (black and white)
binary_mask = (predicted_mask > 0.5).astype(np.uint8)  # Convert to 0 or 1 (binary)

# Display the binary mask
plt.imshow(binary_mask, cmap='gray')
plt.axis(False)
plt.show()
```