```
import numpy as np
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
path='/content/drive/MyDrive/Queensland Dataset CE42/Queensland Dataset CE42'
```

```
import os
os.listdir(path)
```

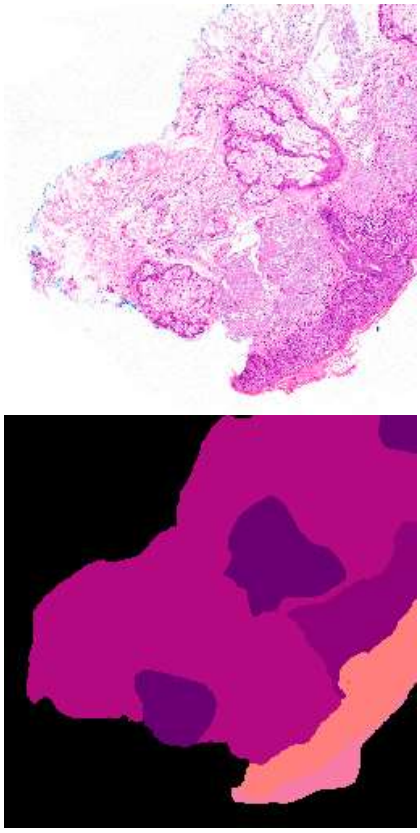
```
['IEC', 'BCC', 'SCC']
```

```
import cv2
```

```
dataset=[]
masks=[]

for i in os.listdir(path):
  for j in os.listdir(os.path.join(path,i)):
    if(j!='Masks'):
      for k in (sorted(os.listdir(os.path.join(path,i,j)))):
        img=cv2.imread(os.path.join(path,i,j,k))
        if(img is not None):
          dataset.append(np.array(img))


    elif(j=='Masks'):

      for k in (sorted(os.listdir(os.path.join(path,i,j)))):
         img_masks=cv2.imread(os.path.join(path,i,j,k))
         if(img is not None):
          masks.append(np.array(img_masks))



dataset=np.array(dataset)
masks=np.array(masks)
```

```
def gamma_correction(image, gamma):
    inv_gamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** inv_gamma) * 255 for i in np.arange(0, 256)]).astype("uint8")
    corrected_image = np.zeros_like(image)
    height, width, channels = image.shape
    for y in range(height):
        for x in range(width):
            for c in range(channels):
                corrected_image[y, x, c] = table[image[y, x, c]]
    return corrected_image
```

```
filtered_images=[]
for i in range(len(dataset)):
  gamma = 0.5
  corrected_image = gamma_correction(dataset[i], gamma)
  filtered_images.append(corrected_image)


filtered_images=np.array(filtered_images)


from google.colab.patches import cv2_imshow
new_image=dataset[2]
cv2_imshow(new_image)
mask_img=masks[2]
cv2_imshow(mask_img)
```



```
Label_dictionary = [
    (108, 0, 115),
    (145, 1, 122),
    (216, 47, 148),
    (254, 246, 242),
    (181, 9, 130),
    (236, 85, 157),
    (73, 0, 106),
    (248, 123, 168),
    (0, 0, 0),
    (127, 255, 255),
    (127, 255, 142),
    (255, 127, 127)
]

label_dict = {i: color for i, color in enumerate(Label_dictionary)}
```

```
def encode_image(image):
    encode_img = np.full(image.shape[:2], 8, dtype=np.uint8)
    reversed_image = image[:, :, ::-1]

    for k, y in enumerate(Label_dictionary):
        mask = np.all(reversed_image == y, axis=-1)
        encode_img[mask] = k

    return encode_img


def decode_image(image):
    if len(image.shape) == 2:
        row, col = image.shape
        num_classes = len(label_dict)
    else:
        row, col, num_classes = image.shape

    decode_img = np.zeros((row, col, 3), dtype=np.uint8)

    if len(image.shape) == 2:
        for i in range(row):
            for j in range(col):
                class_idx = image[i, j]
                if class_idx in label_dict:
                    decode_img[i, j] = label_dict[class_idx]
    else:
        for i in range(row):
            for j in range(col):
                class_idx = np.argmax(image[i, j])
                if class_idx in label_dict:
                    decode_img[i, j] = label_dict[class_idx]

    return decode_img



x=encode_image(masks[2])
cv2_imshow(masks[2])
cv2_imshow(x)
y=decode_image(x)
cv2_imshow(y)
```

```
np.unique(x)
```

```
array([ 0,  1,  4,  7,  8, 11], dtype=uint8)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(filtered_images,masks,test_size=0.2,random_state=42)
```

```
training_label_masks=[]
for i in y_train:
  encoded_img=encode_image(i)
  training_label_masks.append(encoded_img)
```

```
(np.array(training_label_masks)).shape
```

```
(1200, 256, 256)
```

```
import tensorflow as tf
training_label_masks = np.array(training_label_masks)
training_label_masks[training_label_masks >= 12] = 8
num_classes = 12
train_label_mask = tf.keras.utils.to_categorical(training_label_masks, num_classes=12)
```

```
train_label_mask.shape
```

```
(1200, 256, 256, 12)
```

```
testing_label_mask=[]
for i in range(len(x_test)):
  encoded_img=encode_image(x_test[i])
  testing_label_mask.append(encoded_img)
```

```
from tensorflow import keras
testing_label_mask = np.array(testing_label_mask)
testing_label_mask[testing_label_mask >= 12] = 8
num_classes = 12
test_label_mask = tf.keras.utils.to_categorical(testing_label_mask, num_classes=12)
```

```
print(" Training Shape :", x_train.shape, y_train.shape)
print(" Testing Shape :", x_test.shape,y_test.shape)
```

```
 Training Shape : (1200, 256, 256, 3) (1200, 256, 256, 3)
 Testing Shape : (300, 256, 256, 3) (300, 256, 256, 3)
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Conv2DTranspose, concatenate,Dropout
```

```
def unet_model(input_img,number_classes):
  inputs = Input(input_img)
  conv1=Conv2D(16,(3,3),padding='same',activation='relu')(inputs)
  conv1=Conv2D(16,(3,3),padding='same',activation='relu')(conv1)
  pool1=MaxPooling2D(pool_size=(2,2))(conv1)
  pool1 = Dropout(0.25)(pool1)
  conv2=Conv2D(32,(3,3),padding='same',activation='relu')(pool1)
  conv2=Conv2D(32,(3,3),padding='same',activation='relu')(conv2)
  pool2=MaxPooling2D(pool_size=(2,2))(conv2)
  pool2 = Dropout(0.25)(pool2)

  conv3=Conv2D(64,(3,3),padding='same',activation='relu')(pool2)
  conv3=Conv2D(64,(3,3),padding='same',activation='relu')(conv3)
  pool3=MaxPooling2D(pool_size=(2,2))(conv3)
  pool3 = Dropout(0.25)(pool3)
  conv4=Conv2D(128,(3,3),padding='same',activation='relu')(pool3)
  conv4=Conv2D(128,(3,3),padding='same',activation='relu')(conv4)
  pool4=MaxPooling2D(pool_size=(2,2))(conv4)
  pool4 = Dropout(0.25)(pool4)

  conv5=Conv2D(256,(3,3),padding='same',activation='relu')(pool4)
  conv5=Conv2D(256,(3,3),padding='same',activation='relu')(conv5)
  #decoder
  up6=Conv2DTranspose(128,(2,2),strides=(2,2),padding='same')(conv5)
  up6=concatenate([up6,conv4])
  conv6=Conv2D(128,(3,3),padding='same',activation='relu')(up6)
  conv6=Conv2D(128,(3,3),padding='same',activation='relu')(conv6)
  conv6 = Dropout(0.25)(conv6)
  up7=Conv2DTranspose(64,(2,2),strides=(2,2),padding='same')(conv6)
  up7=concatenate([up7,conv3])
  conv7=Conv2D(64,(3,3),padding='same',activation='relu')(up7)
  conv7=Conv2D(64,(3,3),padding='same',activation='relu')(conv7)
  conv7 = Dropout(0.25)(conv7)
  up8=Conv2DTranspose(32,(2,2),strides=(2,2),padding='same')(conv7)
  up8=concatenate([up8,conv2])
  conv8=Conv2D(32,(3,3),padding='same',activation='relu')(up8)
  conv8=Conv2D(32,(3,3),padding='same',activation='relu')(conv8)
  conv8 = Dropout(0.25)(conv8)
  up9=Conv2DTranspose(16,(2,2),strides=(2,2),padding='same')(conv8)
  up9=concatenate([up9,conv1])
  conv9=Conv2D(16,(3,3),padding='same',activation='relu')(up9)
  conv9=Conv2D(16,(3,3),padding='same',activation='relu')(conv9)
  conv9 = Dropout(0.25)(conv9)

  outputs=Conv2D(number_classes,(1,1),activation='softmax')(conv9)
  model=Model(inputs=inputs,outputs=outputs)
  return model



input_shape = (256, 256, 3)
num_classes = 12
model = unet_model(input_shape, num_classes)


model.summary()
```

```
conv2d_10 (Conv2D)          (None, 32, 32, 128)          295040    ['concatenate[0][0]']

conv2d_11 (Conv2D)          (None, 32, 32, 128)          147584    ['conv2d_10[0][0]']

dropout_4 (Dropout)         (None, 32, 32, 128)          0         ['conv2d_11[0][0]']

conv2d_transpose_1 (Conv2D  (None, 64, 64, 64)           32832     ['dropout_4[0][0]']
Transpose)

concatenate_1 (Concatenate  (None, 64, 64, 128)          0         ['conv2d_transpose_1[0][0]',
)                                                                   'conv2d_5[0][0]']

conv2d_12 (Conv2D)          (None, 64, 64, 64)           73792     ['concatenate_1[0][0]']

conv2d_13 (Conv2D)          (None, 64, 64, 64)           36928     ['conv2d_12[0][0]']

dropout_5 (Dropout)         (None, 64, 64, 64)           0         ['conv2d_13[0][0]']

conv2d_transpose_2 (Conv2D  (None, 128, 128, 32)         8224      ['dropout_5[0][0]']
Transpose)

concatenate_2 (Concatenate  (None, 128, 128, 64)         0         ['conv2d_transpose_2[0][0]',
)                                                                   'conv2d_3[0][0]']

conv2d_14 (Conv2D)          (None, 128, 128, 32)         18464     ['concatenate_2[0][0]']

conv2d_15 (Conv2D)          (None, 128, 128, 32)         9248      ['conv2d_14[0][0]']

dropout_6 (Dropout)         (None, 128, 128, 32)         0         ['conv2d_15[0][0]']

conv2d_transpose_3 (Conv2D  (None, 256, 256, 16)         2064      ['dropout_6[0][0]']
Transpose)

concatenate_3 (Concatenate  (None, 256, 256, 32)         0         ['conv2d_transpose_3[0][0]',
)                                                                   'conv2d_1[0][0]']

conv2d_16 (Conv2D)          (None, 256, 256, 16)         4624      ['concatenate_3[0][0]']

conv2d_17 (Conv2D)          (None, 256, 256, 16)         2320      ['conv2d_16[0][0]']

dropout_7 (Dropout)         (None, 256, 256, 16)         0         ['conv2d_17[0][0]']

conv2d_18 (Conv2D)          (None, 256, 256, 12)         204       ['dropout_7[0][0]']

==================================================================================================
Total params: 1941292 (7.41 MB)
Trainable params: 1941292 (7.41 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
x_train.shape
```

```
(1200, 256, 256, 3)
```

```
y_train.shape
```

```
(1200, 256, 256, 3)
```

```
from tensorflow.keras.callbacks import ReduceLROnPlateau,EarlyStopping

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.0001)
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```
from tensorflow.keras.utils import Sequence, to_categorical
batch_size = 32
class DataGenerator(Sequence):
    def __init__(self, images, masks, batch_size):
        self.images = images
        self.masks = masks
        self.batch_size = batch_size
        self.indices = np.arange(len(images))

    def __len__(self):
        return int(np.ceil(len(self.images) / self.batch_size))

    def __getitem__(self, index):
        batch_indices = self.indices[index * self.batch_size:(index + 1) * self.batch_size]
        batch_images = [self.images[i] for i in batch_indices]
        batch_masks = [self.masks[i] for i in batch_indices]
        return np.array(batch_images), np.array(batch_masks)

train_gen = DataGenerator(x_train, train_label_mask, batch_size)
test_gen = DataGenerator(x_test, test_label_mask, batch_size)
model.fit(train_gen, epochs=10, validation_data=test_gen)
```

```
Epoch 1/10
38/38 [==============================] - 52s 730ms/step - loss: 5.3546 - accuracy: 0.2375 - val_loss: 0.9684 -
Epoch 2/10
38/38 [==============================] - 17s 436ms/step - loss: 1.6096 - accuracy: 0.4877 - val_loss: 0.9442 -
Epoch 3/10
38/38 [==============================] - 17s 431ms/step - loss: 1.4703 - accuracy: 0.5522 - val_loss: 1.1772 -
Epoch 4/10
38/38 [==============================] - 17s 438ms/step - loss: 1.3633 - accuracy: 0.5796 - val_loss: 0.9797 -
Epoch 5/10
38/38 [==============================] - 17s 444ms/step - loss: 1.3065 - accuracy: 0.5985 - val_loss: 1.3295 -
Epoch 6/10
38/38 [==============================] - 17s 456ms/step - loss: 1.2227 - accuracy: 0.6209 - val_loss: 1.4220 -
Epoch 7/10
38/38 [==============================] - 17s 440ms/step - loss: 1.1906 - accuracy: 0.6315 - val_loss: 1.4516 -
Epoch 8/10
38/38 [==============================] - 17s 440ms/step - loss: 1.1525 - accuracy: 0.6409 - val_loss: 1.4903 -
Epoch 9/10
38/38 [==============================] - 17s 449ms/step - loss: 1.1745 - accuracy: 0.6412 - val_loss: 1.5438 -
Epoch 10/10
38/38 [==============================] - 17s 441ms/step - loss: 1.1410 - accuracy: 0.6469 - val_loss: 1.6339 -
<keras.src.callbacks.History at 0x7e842ee9baf0>
```

```
model.save('unet_model_new.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your mod
  saving_api.save_model(
```

```
prediction=model.predict(x_test)
```

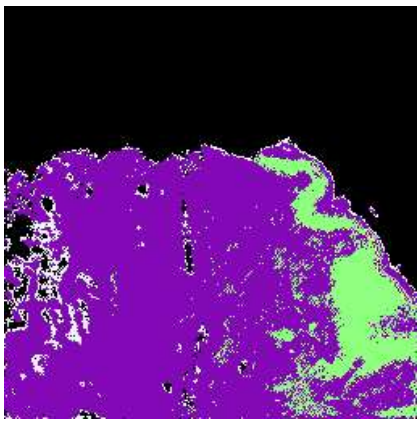
```
10/10 [==============================] - 1s 77ms/step
```

```
prediction[1].shape
```

```
(256, 256, 12)
```

```
cv2_imshow(y_test[11])
```

```
from google.colab.patches import cv2_imshow
new_img=decode_image(prediction[11])
cv2_imshow(new_img)
```



```
print(np.argmax(prediction[11],axis=-1))
```