CODE

import numpy as np

import pandas as pd

import os

import glob as gb

import os

import matplotlib.pyplot as plt

from tqdm import tqdm

import cv2

from skimage.io import imread, imshow

from skimage.transform import resize

from google.colab import drive

drive.mount('/content/drive')

TRAIN\_PATH ='/content/drive/MyDrive/input/nails\_segmentation/images/'

IMG\_WIDTH = 256

IMG\_HEIGHT = 256

IMG\_CHANNELS = 3

train\_ids = gb.glob(TRAIN\_PATH+'\*.jpg')

X\_train = np.zeros((len(train\_ids), IMG\_HEIGHT, IMG\_WIDTH, IMG\_CHANNELS), dtype=np.uint8)

for n, id\_ in tqdm(enumerate(train\_ids), total=len(train\_ids)):

    img = imread(id\_)[:,:,:IMG\_CHANNELS]

    img = resize(img, (IMG\_HEIGHT, IMG\_WIDTH), mode='constant', preserve\_range=True)

    X\_train[n] = img

plt.figure(figsize=(20,20))

for n ,i in enumerate(list(np.random.randint(0,len(X\_train),16))) :

    plt.subplot(4,4,n+1)

    plt.imshow(X\_train[i])

    plt.axis('off')

    plt.title(i)

TRAIN\_PATH ='/content/drive/MyDrive/input/nails\_segmentation/labels/'

IMG\_WIDTH = 256

IMG\_HEIGHT = 256

IMG\_CHANNELS = 1

train\_ids = gb.glob(TRAIN\_PATH+'\*.jpg')

Y\_train = np.zeros((len(train\_ids), IMG\_HEIGHT, IMG\_WIDTH, 1), dtype=np.bool)

for n, id\_ in tqdm(enumerate(train\_ids), total=len(train\_ids)):

    mask = imread(id\_)[:,:,:1]

    mask = resize(mask, (IMG\_HEIGHT, IMG\_WIDTH), mode='constant', preserve\_range=True)

    Y\_train[n] = mask

plt.figure(figsize=(20,20))

for n ,i in enumerate(list(np.random.randint(0,len(Y\_train),16))) :

    plt.subplot(4,4,n+1)

    plt.imshow(Y\_train[i],cmap='gray')

    plt.axis('off')

    plt.title(i)

import tensorflow

from tensorflow.keras.models import Model

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

def unet(IMG\_HEIGHT, IMG\_WIDTH, IMG\_CHANNELS):

#Build the model

    inputs = Input((IMG\_HEIGHT, IMG\_WIDTH, IMG\_CHANNELS))

    #Contraction path

    c2 = Conv2D(64, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(inputs)

    c2 = Dropout(0.1)(c2)

    c2 = Conv2D(64, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c2)

    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(128, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(p2)

    c3 = Dropout(0.2)(c3)

    c3 = Conv2D(128, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c3)

    p3 = MaxPooling2D((2, 2))(c3)

    c4 = Conv2D(256, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(p3)

    c4 = Dropout(0.2)(c4)

    c4 = Conv2D(256, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c4)

    #Expansive path

    u7 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c4)

    u7 = concatenate([u7, c3])

    c7 = Conv2D(128, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(u7)

    c7 = Dropout(0.2)(c7)

    c7 = Conv2D(128, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c7)

    u8 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c7)

    u8 = concatenate([u8, c2])

    c8 = Conv2D(64, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(u8)

    c8 = Dropout(0.1)(c8)

    c8 = Conv2D(64, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c8)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c8)

    model = Model(inputs=[inputs], outputs=[outputs])

    model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

    model.summary()

    return model

model=unet(256,256,3)

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

EarlyStop=EarlyStopping(patience=10,restore\_best\_weights=True)

Reduce\_LR=ReduceLROnPlateau(monitor='val\_accuracy',verbose=2,factor=0.5,min\_lr=0.00001)

model\_check=ModelCheckpoint('model.hdf5',monitor='val\_loss',verbose=1,save\_best\_only=True)

callback=[EarlyStop , Reduce\_LR,model\_check]

history=model.fit(X\_train,Y\_train,validation\_split=0.1,batch\_size=8,steps\_per\_epoch=len(X\_train)//8,epochs=25,

                 callbacks=callback,verbose=1,shuffle=True)

#plotting training values

import seaborn as sns

sns.set()

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(loss) + 1)

#accuracy plot

plt.plot(epochs, acc, color='green', label='Training Accuracy')

plt.plot(epochs, val\_acc, color='blue', label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend()

plt.figure()

#loss plot

plt.plot(epochs, loss, color='green', label='Training Loss')

plt.plot(epochs, val\_loss, color='red', label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

y\_pred=model.predict(X\_train)

y\_pred\_1=(y\_pred > 0.3).astype(np.uint8)

plt.imshow(y\_pred\_1[5],cmap='gray')

plt.show()

plt.imshow(Y\_train[5],cmap='gray')

plt.show()

# import necessary libraries

import tensorflow as tf

from tensorflow.keras.preprocessing import image

import numpy as np

# load your own dataset and map categories to labels

train\_data = tf.keras.preprocessing.image\_dataset\_from\_directory(

    '/content/drive/MyDrive/nails/train',

    validation\_split=0.2,

    subset='training',

    seed=123,

    image\_size=(224, 224),

    batch\_size=32)

categories = train\_data.class\_names

num\_categories = len(categories)

# load pre-trained ResNet50 model

model = tf.keras.applications.ResNet50(

    include\_top=True,

    weights='imagenet')

# load image to predict

img\_path = '/content/drive/MyDrive/nails/train/NailFungus/acute-paronychia-10\_jpg.rf.30fe6a1eb24243b296b136fba85b7d63.jpg'

img = image.load\_img(img\_path, target\_size=(224, 224))

x = image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0)

x = tf.keras.applications.resnet50.preprocess\_input(x)

# make prediction

preds = model.predict(x)

categories = ['Normal','Bening','Malignant']

# example predicted label

predicted\_label = 0

# map predicted label to category name

category = categories[predicted\_label]

print("The image is classified as:", category)