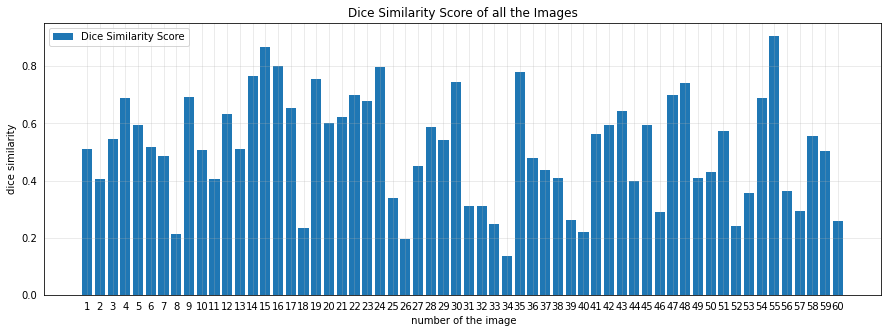
CMP9135M Computer Vision

## Task 1: Image Segmentation and Detection

**1.1**

|  |  |  |  |
| --- | --- | --- | --- |
| Skin Image Name | Colored Skin Image | Binary Skin Image | Dice Similarity |
| ISIC\_0000019 | A picture containing text, floor  Description automatically generated |  | 0.6873050525748563 |
| ISIC\_0000095 | A close up of a person's eye  Description automatically generated with medium confidence | A picture containing silhouette, cave  Description automatically generated | 0.24747184235200276 |
| ISIC\_0000214 | A close-up of a person's skin  Description automatically generated with low confidence | A picture containing silhouette  Description automatically generated | 0.5572162341982702 |

**1.2**

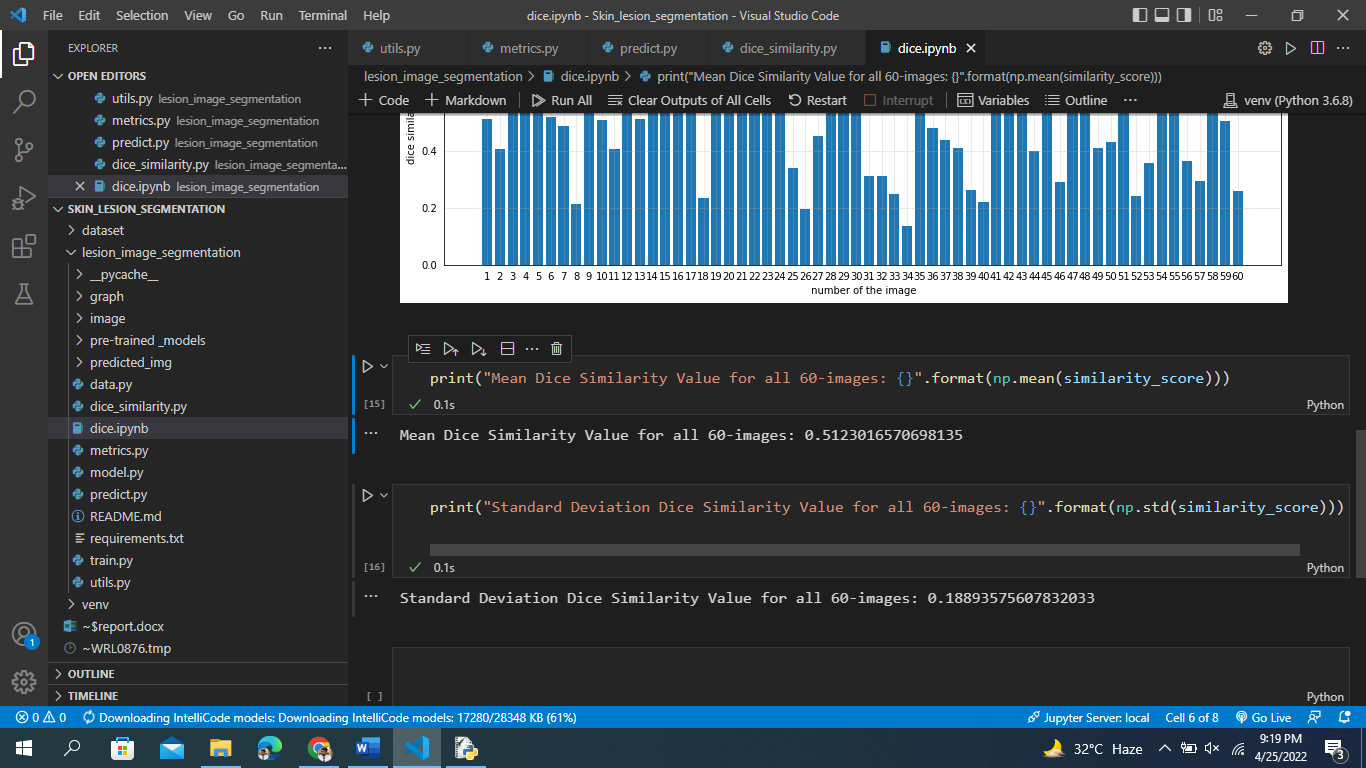
**1.3**

**Mean Dice Similarity:**

**Graphical user interface

Description automatically generated**

**Standard Deviation of Dice Similarity:**

****

**Implementation Steps:**

1. **Import required libraries**

import numpy as np

import cv2

import tensorflow as tf

from tensorflow.keras.layers import \*

from tensorflow.keras.models import Model

from tensorflow.keras.applications import \*

1. **Building the Convolution Block**

def conv\_block(inputs, filters):

    x = inputs

    x = Conv2D(filters, (3, 3), padding="same")(x)

    x = BatchNormalization()(x)

    x = Activation('relu')(x)

    x = Conv2D(filters, (3, 3), padding="same")(x)

    x = BatchNormalization()(x)

    x = Activation('relu')(x)

    x = squeeze\_excite\_block(x)

    return x

1. **Encoder, Decoder Block**

def decoder1(inputs, skip\_connections):

    num\_filters = [256, 128, 64, 32]

    skip\_connections.reverse()

    x = inputs

    for i, f in enumerate(num\_filters):

        x = UpSampling2D((2, 2), interpolation='bilinear')(x)

        x = Concatenate()([x, skip\_connections[i]])

        x = conv\_block(x, f)

    return x

def encoder2(inputs):

    num\_filters = [32, 64, 128, 256]

    skip\_connections = []

    x = inputs

    for i, f in enumerate(num\_filters):

        x = conv\_block(x, f)

        skip\_connections.append(x)

        x = MaxPool2D((2, 2))(x)

    return x, skip\_connections

def decoder2(inputs, skip\_1, skip\_2):

    num\_filters = [256, 128, 64, 32]

    skip\_2.reverse()

    x = inputs

    for i, f in enumerate(num\_filters):

        x = UpSampling2D((2, 2), interpolation='bilinear')(x)

        x = Concatenate()([x, skip\_1[i], skip\_2[i]])

        x = conv\_block(x, f)

    return x

1. **Construct the Double U-net Architecture**

def create\_double\_u\_net(input\_shape):

    inputs = Input(input\_shape)

    x, skip\_1 = encoder1(inputs)

    x = ASPP(x, 64)

    x = decoder1(x, skip\_1)

    outputs1 = output\_block(x)

    x = inputs \* outputs1

    x, skip\_2 = encoder2(x)

    x = ASPP(x, 64)

    x = decoder2(x, skip\_1, skip\_2)

    outputs2 = output\_block(x)

    outputs = Concatenate()([outputs1, outputs2])

    model = Model(inputs, outputs)

    return model

def create\_unet(input\_shape):

    inputs = tf.keras.Input(input\_shape)

    conv1 = Conv2D(64, kernel\_size=(3,3), activation='relu', padding='same')(inputs)

    conv2 = Conv2D(64, kernel\_size=(3,3), activation='relu', padding='same')(conv1)

    pool2 = MaxPool2D(pool\_size=(2,2),padding='same')(conv2)

    conv3 = Conv2D(128, kernel\_size=3, activation='relu', padding='same')(pool2)

    conv4 = Conv2D(128, kernel\_size=3, activation='relu', padding='same')(conv3)

    pool4 = MaxPool2D(pool\_size=(2,2),padding='same')(conv4)

    conv5 = Conv2D(256, kernel\_size=(3,3), activation='relu', padding='same')(pool4)

    conv6 = Conv2D(256, kernel\_size=(3,3), activation='relu', padding='same')(conv5)

    pool6 = MaxPool2D(pool\_size=(2,2))(conv6)

    conv7 = Conv2D(512, kernel\_size=(3,3), activation='relu', padding='same')(pool6)

    conv8 = Conv2D(512, kernel\_size=(3,3), activation='relu', padding='same')(conv7)

    pool8 = MaxPool2D(pool\_size=(2,2))(conv8)

    conv9 = Conv2D(1024, kernel\_size=(3,3), activation='relu', padding='same')(pool8)

    conv10 = Conv2D(1024, kernel\_size=(3,3), activation='relu', padding='same')(conv9)

    up11 = UpSampling2D(size=(2, 2))(conv10)

    up11 = Conv2D(512, kernel\_size=(3,3), activation='relu', padding='same')(up11)

    up11 = Concatenate(axis=-1)([conv8,up11])

    conv12 = Conv2D(512, kernel\_size=(3,3), activation='relu', padding='same')(up11)

    conv13 = Conv2D(512, kernel\_size=(3,3), activation='relu', padding='same')(conv12)

    up14 = UpSampling2D((2,2))(conv13)

    up14 = Conv2D(256, kernel\_size=(3,3), activation='relu', padding='same')(up14)

    up14 = Concatenate(axis=-1)([conv6,up14])

    conv15 = Conv2D(256, kernel\_size=(3,3), activation='relu', padding='same')(up14)

    conv16 = Conv2D(256, kernel\_size=(3,3), activation='relu', padding='same')(conv15)

    up17 = UpSampling2D((2,2))(conv16)

    up17 = Conv2D(128, kernel\_size=(3,3), activation='relu', padding='same')(up17)

    up17 = Concatenate(axis=-1)([conv4,up17])

    conv18 = Conv2D(128, kernel\_size=(3,3), activation='relu', padding='same')(up17)

    conv19 = Conv2D(128, kernel\_size=(3,3), activation='relu', padding='same')(conv18)

    up18 = UpSampling2D((2,2))(conv19)

    up18 = Conv2D(64, kernel\_size=(3,3), activation='relu', padding='same')(up18)

    up18 = Concatenate(axis=-1)([conv2,up18])

    conv19 = Conv2D(64, kernel\_size=(3,3), activation='relu', padding='same')(up18)

    conv20 = Conv2D(64, kernel\_size=(3,3), activation='relu', padding='same')(conv19)

    conv21 = Conv2D(1, kernel\_size=(1,1), activation='sigmoid', padding = 'same')(conv20)

    output = conv21

    model = tf.keras.Model(inputs,output)

    return output

1. **Train the Model**

model = create\_double\_u\_net((192, 256, 3))

loss\_function = DiceLoss()

model.compile('Adam', loss=loss\_function, metrics=[iou])

model.fit(train, epochs=100, validation\_data=validation,

          initial\_epoch=0,

          verbose=1,

          callbacks=[tb, mc])

1. **Predict the Model**

file\_path\_model = 'pre-trained\_models/model.h5'

trained\_model = tf.keras.models.load\_model(file\_path\_model, compile=False)

size = (512, 384)

images\_folder = 'image'

file\_img = os.listdir(images\_folder)

abc = []

for index, i in enumerate(file\_img):

    name = images\_folder + '/'+i

    img2 = cv2.imread(name, cv2.IMREAD\_COLOR)

    out\_put = get\_predict(trained\_model, img2)

    out\_put = out\_put.astype(np.uint8)

    out\_put = np.stack([out\_put, out\_put, out\_put], axis=-1)

    abc.append(out\_put)

    cv2.imwrite(f"predicted\_img/{i}", out\_put)

1. **Dice Similarity Calculation**

y\_pred = cv2.imread('predicted\_img/ISIC\_0000214.jpg')

y\_true = cv2.imread('image/ISIC\_0000214.jpg')

y\_true = cv2.cvtColor(y\_true, cv2.COLOR\_BGR2GRAY)

y\_pred = cv2.cvtColor(y\_pred, cv2.COLOR\_BGR2GRAY)

y\_true = cv2.resize(y\_true, (224, 224))

y\_pred = cv2.resize(y\_pred, (224, 224))

def dice(pred, true, k = 1):

    intersection = np.sum(pred[true!=k]) \* 2.0

    dice = intersection / (np.sum(pred) + np.sum(true))

    return dice

dice\_score = dice(y\_pred, y\_true, k=255)

print ("Dice Similarity Score: {}".format(dice\_score))

# Appendix

1. <https://challenge.isic-archive.com/data/>
2. <https://vnopenai.github.io/ai-doctor/vision/skin-lesion-segmentation/intro-and-data/>
3. https://arxiv.org/pdf/2006.04868.pdf
4. https://arxiv.org/pdf/1606.00915v2.pdf
5. https://arxiv.org/abs/1709.01507
6. https://github.com/DebeshJha/2020-CBMS-DoubleU-Net
7. https://challenge2018.isic-archive.com/