Group Name: Innovators

Group:09

ML+CV Combined Project: Cell Segmentation

Group Members

Aadya Chinubhai AU2040232

Abhishu Oza AU2040027

Razin Karimi AU2040230

Nihar Jani AU2040205

Tasks Performed:

- ## ***Implementation of Unet***

```
Import numpy as np
Import tensorflow as tf
From tensorflow.keras import backend as K
From tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Conv2DTranspose,
concatenate
From tensorflow.keras.models import Model
From keras import metrics
Import keras
Import os
Import random
From tqdm import tqdm
From zipfile import ZipFile
From skimage.io import imread, imshow
From skimage.transform import resize
Import matplotlib.pyplot as plt
Import pandas as pd
Class UNet:

  Def __init__(self, input_shape_,  activation_ = "relu"):
    Self.activation = activation_
    Self.input_shape = input_shape_

  Def dice_coef(self, y_true, y_pred, smooth=100):
    Y_true = tf.cast(y_true, tf.float32)
    Y_pred = tf.cast(y_pred, tf.float32)

    Y_true_f = K.flatten(y_true)
    Y_pred_f = K.flatten(y_pred)
    Intersection = K.sum(y_true_f * y_pred_f)
    Dice = (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)
    Return dice

  Def dice_coef_loss(self, y_true, y_pred):
    Return -self.dice_coef(y_true, y_pred)

  Def iou(self, y_true, y_pred):
    Y_true = tf.cast(y_true, tf.float32)
    Y_pred = tf.cast(y_pred, tf.float32)
```

```
        Def f(y_true, y_pred):
            Intersection = (y_true * y_pred).sum()
            Union = y_true.sum() + y_pred.sum() – intersection
            X = (intersection + 1e-15) / (union + 1e-15)
            X = x.astype(np.float32)
            Return x

        Return tf.numpy_function(f, [y_true, y_pred], tf.float32)

    Def buildModel(self):

        # Lets create the DownSampling Blocks
        Kernel_initializer =  ‘he_uniform’

        Inputs = tf.keras.Input(shape = self.input_shape)

        # Block – 1

        S = inputs

        #Contraction path
        C1 = Conv2D(16, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer, 
padding=’same’)(s)
#         c1 = Dropout(0.1)(c1)
        C1 = Conv2D(16, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer, 
padding=’same’)(c1)
        P1 = MaxPooling2D((2, 2))(c1)

        C2 = Conv2D(32, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer, 
padding=’same’)(p1)
#         c2 = Dropout(0.1)(c2)
        C2 = Conv2D(32, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer, 
padding=’same’)(c2)
        P2 = MaxPooling2D((2, 2))(c2)

        C3 = Conv2D(64, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer, 
padding=’same’)(p2)
#         c3 = Dropout(0.2)(c3)
        C3 = Conv2D(64, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer, 
padding=’same’)(c3)
        P3 = MaxPooling2D((2, 2))(c3)
    C4 = Conv2D(128, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(p3)
#       c4 = Dropout(0.2)(c4)
    C4 = Conv2D(128, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(c4)
    P4 = MaxPooling2D(pool_size=(2, 2))(c4)

    C5 = Conv2D(256, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(p4)
#       c5 = Dropout(0.3)(c5)
    C5 = Conv2D(256, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(c5)

    #Expansive path
    U6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding=’same’)(c5)
    U6 = concatenate([u6, c4])
    C6 = Conv2D(128, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(u6)
#       c6 = Dropout(0.2)(c6)
    C6 = Conv2D(128, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(c6)

    U7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding=’same’)(c6)
    U7 = concatenate([u7, c3])
    C7 = Conv2D(64, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(u7)
#       c7 = Dropout(0.2)(c7)
    C7 = Conv2D(64, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(c7)

    U8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding=’same’)(c7)
    U8 = concatenate([u8, c2])
    C8 = Conv2D(32, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(u8)
#       c8 = Dropout(0.1)(c8)
    C8 = Conv2D(32, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(c8)

    U9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding=’same’)(c8)
    U9 = concatenate([u9, c1])
    C9 = Conv2D(16, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer,
padding=’same’)(u9)
#       c9 = Dropout(0.1)(c9)
```

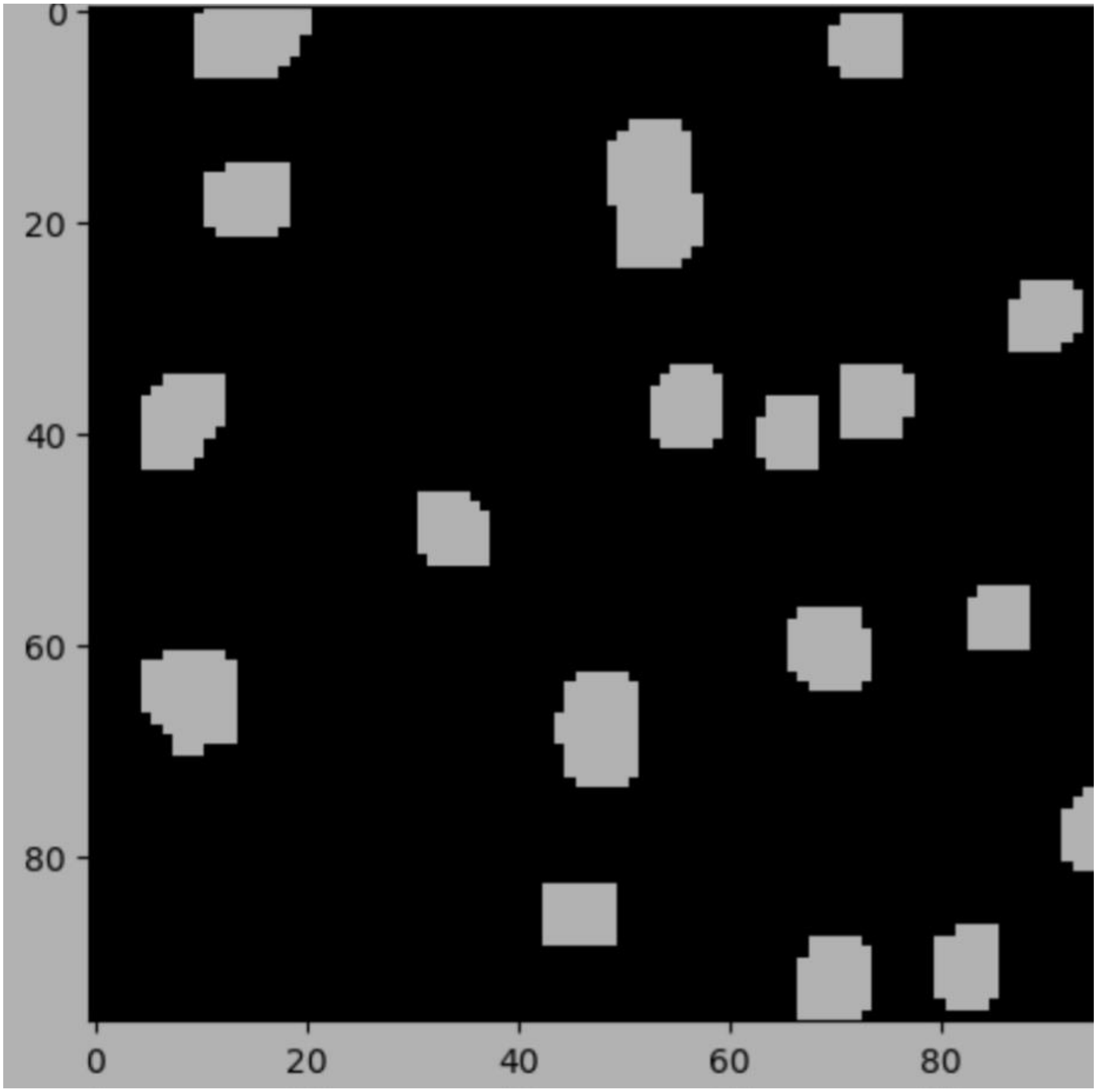
```
    C9 = Conv2D(16, (3, 3), activation= self.activation, kernel_initializer=kernel_initializer, padding=’same’)(c9)

    Outputs = Conv2D(1, (1, 1), activation=’sigmoid’)(c9)

    Model = Model(inputs=[inputs], outputs=[outputs])
    #compile model outside of this function to make it flexible.
    Model.summary()

    Return model.
```

- ***Output Images***



- ***<u>Things to do in future</u>***

  Apply different activation function in given algorithm and see different outputs as well as examine them.