**Code:**

**Program:**

import os

import sys

import random

import numpy as np

import cv2

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

#Data Generator

class DataGen(keras.utils.Sequence):

def \_\_init\_\_(self, ids, path, batch\_size=8, image\_size=128):

self.ids = ids

self.path = path

self.batch\_size = batch\_size

self.image\_size = image\_size

self.on\_epoch\_end()

def \_\_load\_\_(self, id\_name):

## Path

image\_path = os.path.join(self.path, "Data", id\_name) + ".tif"

mask\_path = os.path.join(self.path, "Mask", id\_name) + "\_mask.tif"

## Reading Image

image = cv2.imread(image\_path, 1)

image = cv2.resize(image, (self.image\_size, self.image\_size))

\_mask\_image = cv2.imread(mask\_path, -1)

mask = cv2.resize(\_mask\_image, (self.image\_size, self.image\_size)) #128x128

## Normalizaing

image = image/255.0

mask = mask/255.0

return image, mask

def \_\_getitem\_\_(self, index):

if(index+1)\*self.batch\_size > len(self.ids):

self.batch\_size = len(self.ids) - index\*self.batch\_size

files\_batch = self.ids[index\*self.batch\_size : (index+1)\*self.batch\_size]

image = []

mask = []

for id\_name in files\_batch:

\_img, \_mask = self.\_\_load\_\_(id\_name)

image.append(\_img)

mask.append(\_mask)

image = np.array(image)

mask = np.array(mask)

return image, mask

def on\_epoch\_end(self):

pass

def \_\_len\_\_(self):

return int(np.ceil(len(self.ids)/float(self.batch\_size)))

#Callback

target\_data = []

loss = []

class History\_LAW(keras.callbacks.Callback):

i = 0

def on\_epoch\_end(self, epoch, logs={}):

self.i += 1

def on\_train\_batch\_end(self, batch, logs=None):

if self.i == 99:

if logs['loss'] > .10:

target\_data.append(batch)

image\_size = 512

train\_path = "/content/drive/My Drive/Dataset\_Brain\_Tumour (copy)/"

epochs = 100

batch\_size = 1

## Training Ids

train\_ids = next(os.walk(train\_path))[1]

image\_ids=[]

for r, d, f in os.walk(train\_path):

for file in f:

if "mask.tif" in file:

fil=(file.split("\_mask.tif"))[0]

image\_ids.append(fil)

count = len(image\_ids)

test\_size = int(count \* (15 / 100))

validation\_size = int(count \* (15 / 100))

train\_size = int(count \* (70 / 100))

test\_data = image\_ids[:test\_size]

train = image\_ids[test\_size:]

valid\_data = train[:validation\_size]

train\_data = train[validation\_size:]

def down\_block(x, filters,rate=0.25, kernel\_size=(3, 3), padding="same", strides=1):

c = keras.layers.Conv2D(filters, kernel\_size, padding=padding, strides=strides, activation="relu")(x)

c = keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True,beta\_initializer='zeros', gamma\_initializer='ones',moving\_mean\_initializer='zeros', moving\_variance\_initializer='ones', beta\_regularizer=None, gamma\_regularizer=None,beta\_constraint=None,gamma\_constraint=None)(c)

c = keras.layers.Dropout(rate)(c)

p = keras.layers.MaxPool2D((2, 2), (2, 2))(c)

return c, p

def up\_block(x, skip, filters,rate=0.25, kernel\_size=(3, 3), padding="same", strides=1):

us = keras.layers.UpSampling2D((2, 2))(x)

concat = keras.layers.Concatenate()([us, skip])

c = keras.layers.Conv2D(filters, kernel\_size, padding=padding, strides=strides, activation="relu")(concat)

c = keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True,beta\_initializer='zeros', gamma\_initializer='ones',moving\_mean\_initializer='zeros', moving\_variance\_initializer='ones', beta\_regularizer=None, gamma\_regularizer=None,beta\_constraint=None,gamma\_constraint=None)(c)

c = keras.layers.Dropout(rate)(c)

return c

def bottleneck(x, filters,rate=0.25, kernel\_size=(3, 3), padding="same", strides=1):

c = keras.layers.Conv2D(filters, kernel\_size, padding=padding, strides=strides, activation="relu")(x)

c = keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True,beta\_initializer='zeros', gamma\_initializer='ones',moving\_mean\_initializer='zeros', moving\_variance\_initializer='ones', beta\_regularizer=None, gamma\_regularizer=None,beta\_constraint=None,gamma\_constraint=None)(c)

c = keras.layers.Conv2D(filters, kernel\_size, padding=padding, strides=strides, activation="relu")(c)

c = keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True,beta\_initializer='zeros', gamma\_initializer='ones',moving\_mean\_initializer='zeros', moving\_variance\_initializer='ones', beta\_regularizer=None, gamma\_regularizer=None,beta\_constraint=None,gamma\_constraint=None)(c)

c = keras.layers.Dropout(rate)(c)

return c

def UNet():

f = [8, 16, 32, 64, 128, 256, 512, 1024]

inputs = keras.layers.Input((image\_size, image\_size, 3))

p0 = inputs

c1, p1 = down\_block(p0, f[0],0.10)

bn = bottleneck(p1, f[3],0.10)

u1 = up\_block(bn, c1, f[0],0.10)

outputs = keras.layers.Conv2D(1, (1, 1), padding="same", activation="sigmoid")(u1)

model = keras.models.Model(inputs, outputs)

return model

#Compile Model

model = UNet()

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

model.summary()

train\_gen = DataGen(train\_data, train\_path, image\_size=image\_size, batch\_size=batch\_size)

valid\_gen = DataGen(valid\_data, train\_path, image\_size=image\_size, batch\_size=batch\_size)

train\_steps = len(train\_data)//batch\_size

valid\_steps = len(valid\_data)//batch\_size

model\_Hist = History\_LAW()

seqModel=model.fit\_generator(train\_gen, validation\_data=valid\_gen, steps\_per\_epoch=train\_steps, validation\_steps=valid\_steps,

epochs=epochs, callbacks = [model\_Hist])

train\_loss1 = seqModel.history['loss']

val\_loss1 = seqModel.history['val\_loss']

train\_acc1 = seqModel.history['acc']

val\_acc1 = seqModel.history['val\_acc']

plt.plot(train\_loss1)

plt.plot(val\_loss1 )

plt.title('Model 1 loss')

plt.ylabel('Percentage')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

plt.plot(train\_acc1)

plt.plot(val\_acc1 )

plt.title('Model 1 Accuracy')

plt.ylabel('Percentage')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

new\_train\_data = []

for i in target\_data:

new\_train\_data.append(train\_data[i])

def UNet2():

f = [8, 16, 32, 64, 128, 256, 512, 1024]

inputs = keras.layers.Input((image\_size, image\_size, 3))

p0 = inputs

c1, p1 = down\_block(p0, f[0],,0.10, (5,5))

bn = bottleneck(p1, f[3],0.10,(7,7))

u1 = up\_block(bn, c1, f[0], ,0.10,(5,5))

outputs = keras.layers.Conv2D(1, (1, 1), padding="same", activation="sigmoid")(u1)

model = keras.models.Model(inputs, outputs)

return model

model2 = UNet2()

model2.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["acc"])

model2.summary()

train\_gen = DataGen(new\_train\_data, train\_path, image\_size=image\_size, batch\_size=batch\_size)

valid\_gen = DataGen(valid\_data, train\_path, image\_size=image\_size, batch\_size=batch\_size)

train\_steps = len(new\_train\_data)//batch\_size

valid\_steps = len(valid\_data)//batch\_size

model\_Hist = History\_LAW()

target\_data.clear()

seqModel = model2.fit\_generator(train\_gen, validation\_data=valid\_gen, steps\_per\_epoch=train\_steps, validation\_steps=valid\_steps,

epochs=epochs, callbacks = [model\_Hist])

train\_loss2 = seqModel.history['loss']

val\_loss2 = seqModel.history['val\_loss']

train\_acc2 = seqModel.history['acc']

val\_acc2 = seqModel.history['val\_acc']

plt.plot(train\_loss2)

plt.plot(val\_loss2 )

plt.title('Model 2 loss')

plt.ylabel('Percentage')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

plt.plot(train\_acc2)

plt.plot(val\_acc2 )

plt.title('Model 2 Accuracy')

plt.ylabel('Percentage')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

def UNet3():

f = [8, 16, 32, 64, 128, 256, 512, 1024]

inputs = keras.layers.Input((image\_size, image\_size, 3))

p0 = inputs

c1, p1 = down\_block(p0, f[0] ,0.10)

c2, p2 = down\_block(p1, f[1],,0.10,(5,5))

bn = bottleneck(p2, f[3], ,0.10,(7,7))

u1 = up\_block(bn, c2, f[1],,0.10,(5,5))

u2 = up\_block(u1, c1,,0.10, f[0])

outputs = keras.layers.Conv2D(1, (1, 1), padding="same", activation="sigmoid")(u2)

model = keras.models.Model(inputs, outputs)

return model

model3 = UNet3()

model3.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["acc"])

model3.summary()

train\_gen = DataGen(new\_train\_data, train\_path, image\_size=image\_size, batch\_size=batch\_size)

valid\_gen = DataGen(valid\_data, train\_path, image\_size=image\_size, batch\_size=batch\_size)

train\_steps = len(new\_train\_data)//batch\_size

valid\_steps = len(valid\_data)//batch\_size

model\_Hist = History\_LAW()

target\_data.clear()

seqModel = model3.fit\_generator(train\_gen, validation\_data=valid\_gen, steps\_per\_epoch=train\_steps, validation\_steps=valid\_steps,

epochs=epochs, callbacks = [model\_Hist])

train\_loss3 = seqModel.history['loss']

val\_loss3 = seqModel.history['val\_loss']

train\_acc3 = seqModel.history['acc']

val\_acc3 = seqModel.history['val\_acc']

plt.plot(train\_loss3)

plt.plot(val\_loss3 )

plt.title('Model 3 loss')

plt.ylabel('Percentage')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

plt.plot(train\_acc3)

plt.plot(val\_acc3 )

plt.title('Model 3 Accuracy')

plt.ylabel('Percentage')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

## Save the Weights

model.save\_weights("Model2.h5")

j=0

for id in valid\_data:

## Dataset for prediction

x, y = valid\_gen.\_\_getitem\_\_(j)

result1 = model.predict(x)

result1 = result1 > 0.5

result2 = model2.predict(x)

result2 = result2 > 0.5

result3 = model3.predict(x)

result3 = result3 > 0.5

j += 2

if np.count\_nonzero(result1)>=np.count\_nonzero(result2) and np.count\_nonzero(result1>=result3):

result = result1

if np.count\_nonzero(result2>=result1) and np.count\_nonzero(result2>=result3):

result = result2

if np.count\_nonzero(result3)>=np.count\_nonzero(result2) and np.count\_nonzero(result3)>=np.count\_nonzero(result1):

result = result3

i=0

while i < len(x):

fig.subplots\_adjust(hspace=0.4, wspace=0.4)

ax = fig.add\_subplot(1, 3, 1)

ax.imshow(np.reshape(y[i]\*255, (image\_size, image\_size)), cmap="gray")

ax.set\_title('Mask')

ax = fig.add\_subplot(1, 3, 2)

ax.imshow(np.reshape(result[i]\*255, (image\_size, image\_size)), cmap="gray")

ax.set\_title('Result')

ax = fig.add\_subplot(1, 3, 3)

ax.imshow(x[i])

ax.set\_title('Image')

plt.show()

i += 1