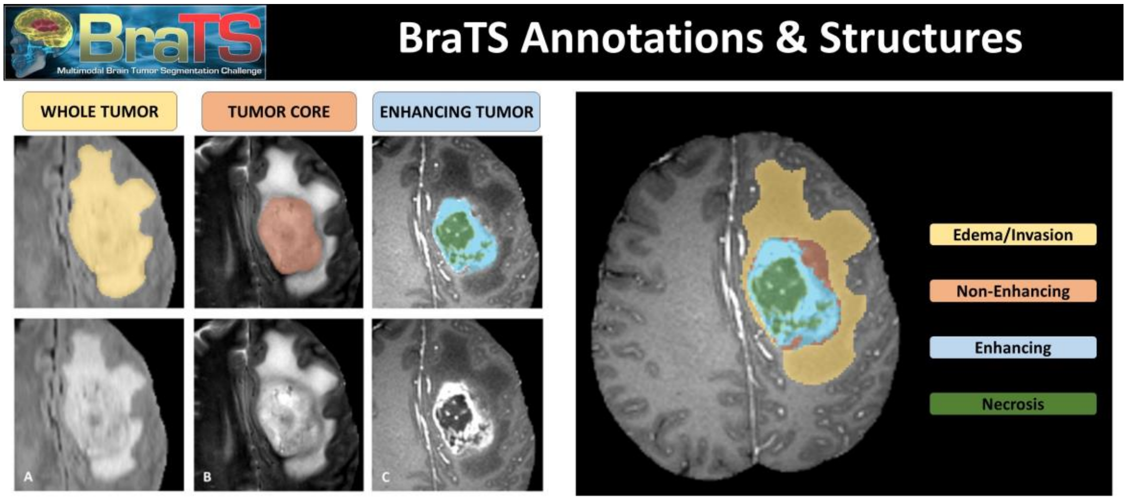
Brain Tumor Segmentation BraTS 2020 Dataset



Problem definiton

Segmentation of gliomas in pre-operative MRI scans.

Each pixel on image must be labeled:

- Pixel is part of a tumor area (1 or 2 or 3) -> can be one of multiple classes / sub-regions
- Anything else -> pixel is not on a tumor region (0)

The sub-regions of tumor considered for evaluation are: 1) the "enhancing tumor" (ET), 2) the "tumor core" (TC), and 3) the "whole tumor" (WT) The provided segmentation labels have values of 1 for NCR & NET, 2 for ED, 4 for ET, and 0 for everything else.

```
import os
import cv2
import glob
import PIL
import shutil
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from skimage import data
from skimage.util import montage
import skimage.transform as skTrans
from skimage.transform import rotate
from skimage.transform import resize
from PIL import Image, ImageOps
```

```
import nilearn as nl
import nibabel as nib
import nilearn.plotting as nlplt
!pip install git+https://github.com/miykael/gif_your_nifti
import gif_your_nifti.core as gif2nif


import keras
import keras.backend as K
from keras.callbacks import CSVLogger
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau,
EarlyStopping, TensorBoard
from tensorflow.keras.layers.experimental import preprocessing

np.set_printoptions(precision=3, suppress=True)
```

```
Collecting git+https://github.com/miykael/gif_your_nifti
  Cloning https://github.com/miykael/gif_your_nifti to /tmp/pip-req-build-
aw647g58
  Running command git clone -q https://github.com/miykael/gif_your_nifti
/tmp/pip-req-build-aw647g58
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-
packages (from gif-your-nifti==0.2.2) (1.19.5)
Requirement already satisfied: nibabel in /opt/conda/lib/python3.7/site-
packages (from gif-your-nifti==0.2.2) (3.2.1)
Requirement already satisfied: imageio<3 in /opt/conda/lib/python3.7/site-
packages (from gif-your-nifti==0.2.2) (2.9.0)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.7/site-
packages (from gif-your-nifti==0.2.2) (3.3.3)
Requirement already satisfied: scikit-image in /opt/conda/lib/python3.7/site-
packages (from gif-your-nifti==0.2.2) (0.18.1)
Requirement already satisfied: pillow in /opt/conda/lib/python3.7/site-
packages (from imageio<3->gif-your-nifti==0.2.2) (7.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-
packages (from matplotlib->gif-your-nifti==0.2.2) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib->gif-your-
nifti==0.2.2) (1.3.1)
Requirement already satisfied: python-dateutil>=2.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib->gif-your-
```

```
nifti==0.2.2) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/opt/conda/lib/python3.7/site-packages (from matplotlib->gif-your-
nifti==0.2.2) (2.4.7)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages
(from cycler>=0.10->matplotlib->gif-your-nifti==0.2.2) (1.15.0)
Requirement already satisfied: packaging>=14.3 in
/opt/conda/lib/python3.7/site-packages (from nibabel->gif-your-nifti==0.2.2)
(20.8)
Requirement already satisfied: networkx>=2.0 in
/opt/conda/lib/python3.7/site-packages (from scikit-image->gif-your-
nifti==0.2.2) (2.5)
Requirement already satisfied: PyWavelets>=1.1.1 in
/opt/conda/lib/python3.7/site-packages (from scikit-image->gif-your-
nifti==0.2.2) (1.1.1)
Requirement already satisfied: scipy>=1.0.1 in /opt/conda/lib/python3.7/site-
packages (from scikit-image->gif-your-nifti==0.2.2) (1.5.4)
Requirement already satisfied: tifffile>=2019.7.26 in
/opt/conda/lib/python3.7/site-packages (from scikit-image->gif-your-
nifti==0.2.2) (2021.2.1)
Requirement already satisfied: decorator>=4.3.0 in
/opt/conda/lib/python3.7/site-packages (from networkx>=2.0->scikit-image-
>gif-your-nifti==0.2.2) (4.4.2)
Building wheels for collected packages: gif-your-nifti
  Building wheel for gif-your-nifti (setup.py) ... e=gif_your_nifti-0.2.2-
py3-none-any.whl size=6634
sha256=c43944de372984e36d22f1d46e123143283e2888842fdd6a58782c71ca4b75aa
  Stored in directory: /tmp/pip-ephem-wheel-cache-
2mz6u469/wheels/4a/8c/d1/b228c3b67231f7459e8f70d73f4dadaf65cd90692d41f43e88
Successfully built gif-your-nifti
Installing collected packages: gif-your-nifti
Successfully installed gif-your-nifti-0.2.2
WARNING: You are using pip version 21.0.1; however, version 24.0 is
available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip
install --upgrade pip' command.
```

```
  SEGMENT_CLASSES = {
    0 : 'NOT tumor',
    1 : 'NECROTIC/CORE',
    2 : 'EDEMA',
    3 : 'ENHANCING'
}

VOLUME_SLICES = 100
VOLUME_START_AT = 22
```

Image data descriptions

All BraTS multimodal scans are available as NIfTI files (.nii.gz) -> commonly used medical imaging format to store brain imagin data obtained using MRI and describe different MRI settings

1. **T1**: T1-weighted, native image, sagittal or axial 2D acquisitions, with 1–6 mm slice thickness.
1. **T1c**: T1-weighted, contrast-enhanced (Gadolinium) image, with 3D acquisition and 1 mm isotropic voxel size for most patients.
2. **T2**: T2-weighted image, axial 2D acquisition, with 2–6 mm slice thickness.
3. **FLAIR**: T2-weighted FLAIR image, axial, coronal, or sagittal 2D acquisitions, 2–6 mm slice thickness.

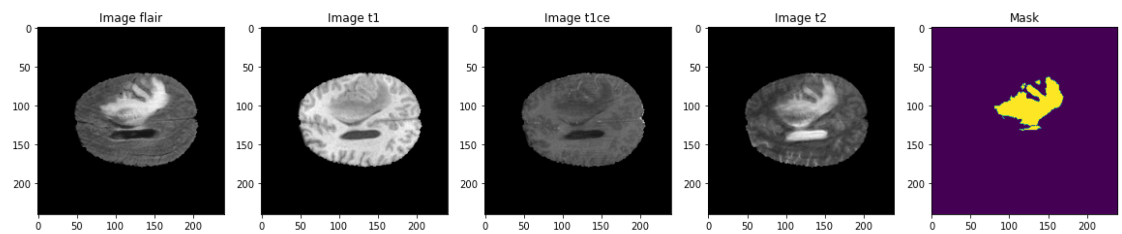
Data were acquired with different clinical protocols and various scanners from multiple (n=19) institutions.

All the imaging datasets have been segmented manually, by one to four raters, following the same annotation protocol, and their annotations were approved by experienced neuro-radiologists. Annotations comprise the GD-enhancing tumor (ET — label 4), the peritumoral edema (ED — label 2), and the necrotic and non-enhancing tumor core (NCR/NET — label 1), as described both in the BraTS 2012-2013 TMI paper and in the latest BraTS summarizing paper. The provided data are distributed after their pre-processing, i.e., co-registered to the same anatomical template, interpolated to the same resolution (1 mm^3) and skull-stripped.

```
TRAIN_DATASET_PATH = '../input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/'
VALIDATION_DATASET_PATH = '../input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCAI_BraTS2020_ValidationData'

test_image_flair=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20_Training_001_flair.nii').get_fdata()
test_image_t1=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20_Training_001_t1.nii').get_fdata()
test_image_t1ce=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20_Training_001_t1ce.nii').get_fdata()
test_image_t2=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20_Training_001_t2.nii').get_fdata()
test_mask=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20_Training_001_seg.nii').get_fdata()


fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1,5, figsize = (20, 10))
slice_w = 25
ax1.imshow(test_image_flair[:,:,test_image_flair.shape[0]//2-slice_w], cmap = 'gray')
ax1.set_title('Image flair')
```

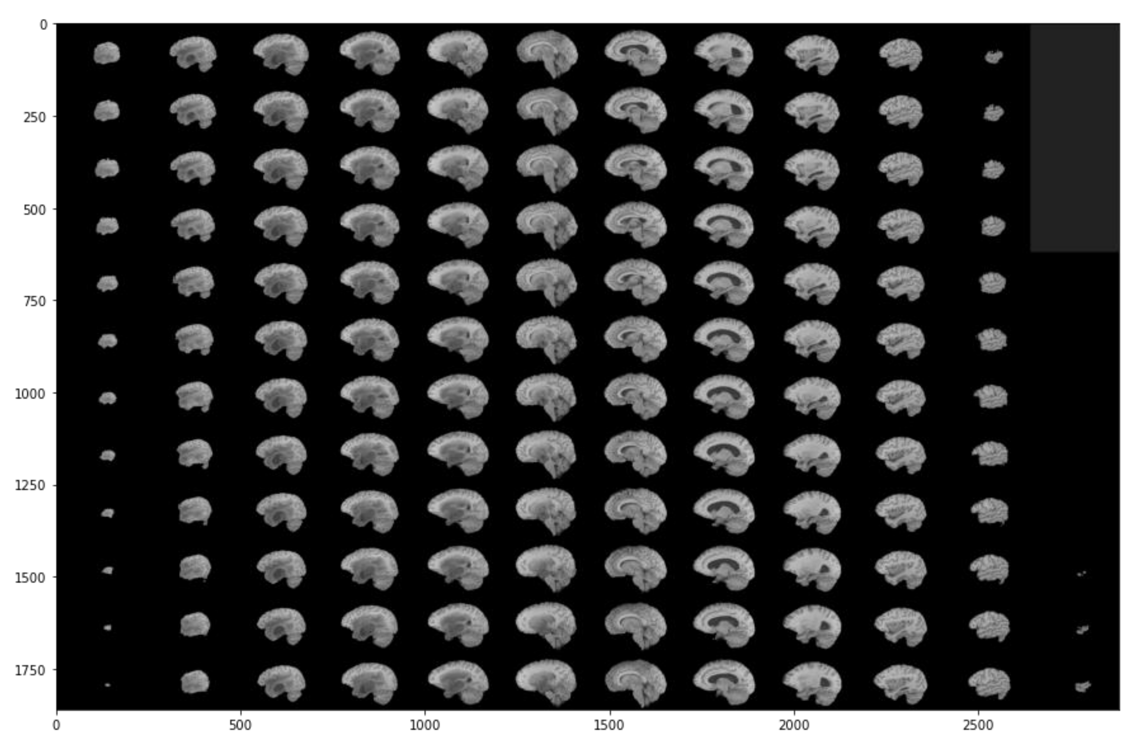
```
ax2.imshow(test_image_t1[:,:,test_image_t1.shape[0]//2-slice_w], cmap =
'gray')
ax2.set_title('Image t1')
ax3.imshow(test_image_t1ce[:,:,test_image_t1ce.shape[0]//2-slice_w], cmap =
'gray')
ax3.set_title('Image t1ce')
ax4.imshow(test_image_t2[:,:,test_image_t2.shape[0]//2-slice_w], cmap =
'gray')
ax4.set_title('Image t2')
ax5.imshow(test_mask[:,:,test_mask.shape[0]//2-slice_w])
ax5.set_title('Mask')
```

```
Text(0.5, 1.0, 'Mask')
```



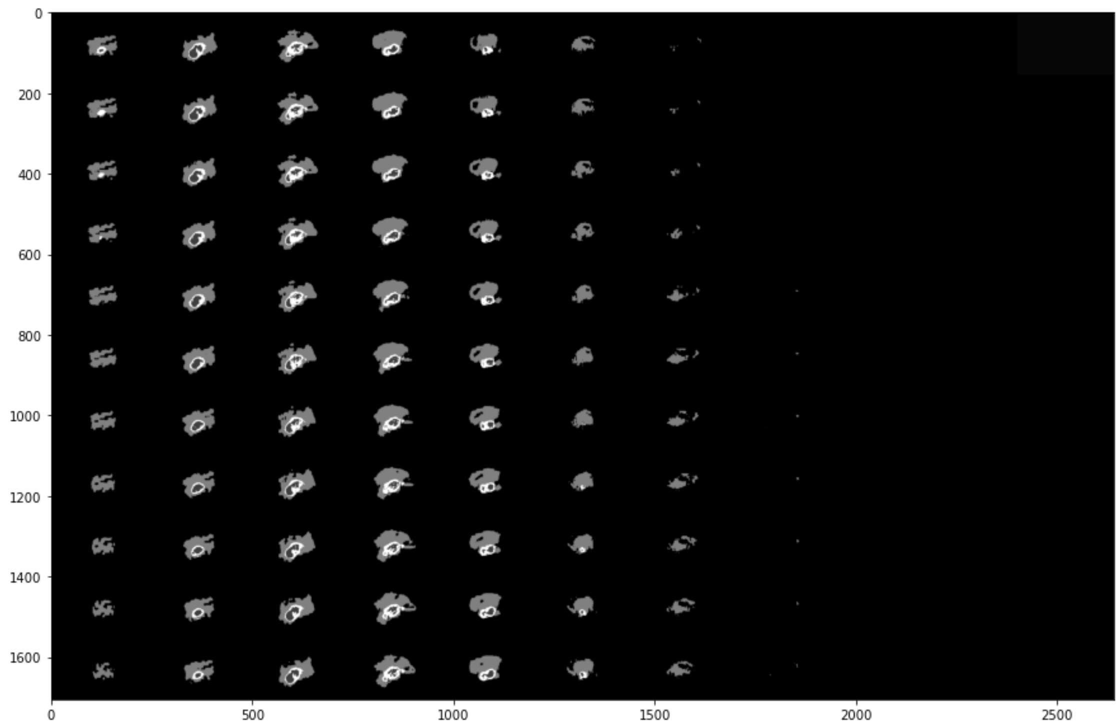

```
fig, ax1 = plt.subplots(1, 1, figsize = (15,15))
ax1.imshow(rotate(montage(test_image_t1[50:-50,:,:]), 90, resize=True), cmap
='gray')
```

```
<matplotlib.image.AxesImage at 0x7a7b9a3732d0>
```



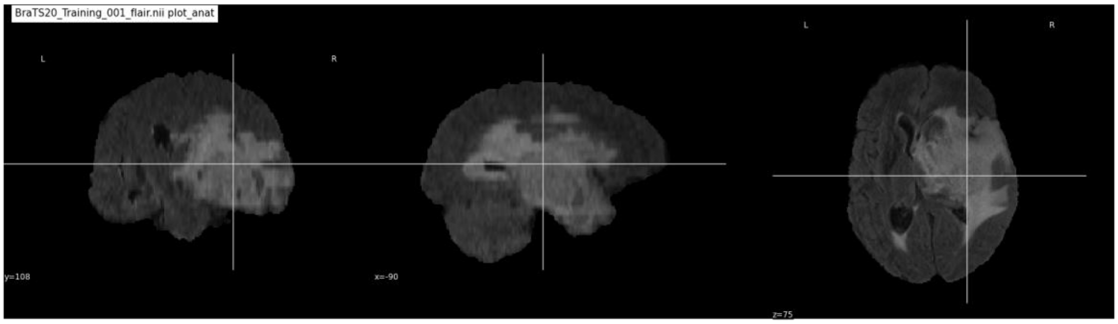
```
fig, ax1 = plt.subplots(1, 1, figsize = (15,15))
ax1.imshow(rotate(montage(test_mask[60:-60,:,:]), 90, resize=True), cmap
='gray')
```

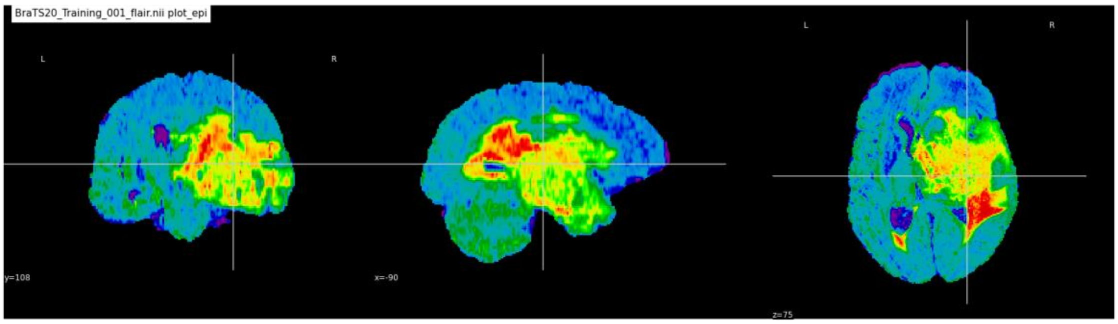
```
<matplotlib.image.AxesImage at 0x7a7bdb744250>
```

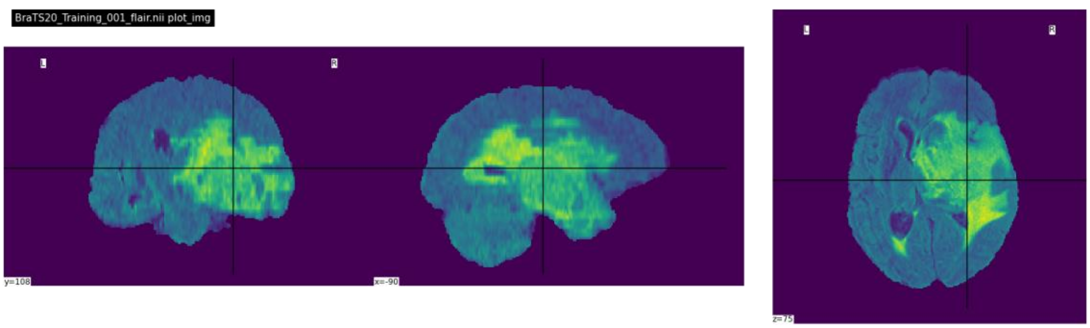


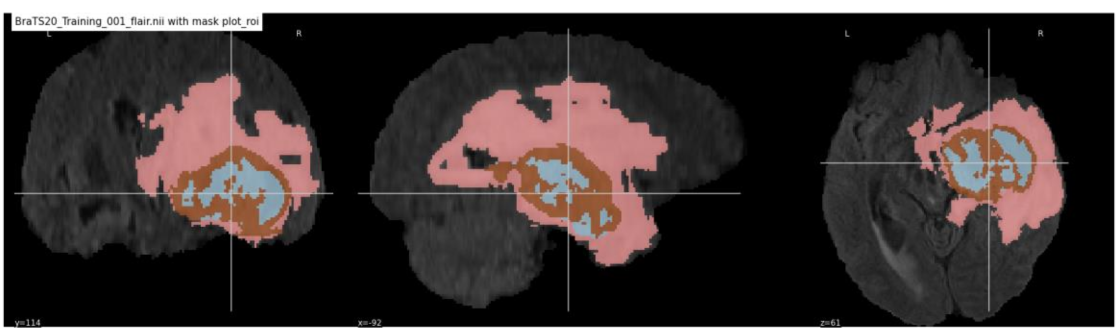

```
shutil.copy2(TRAIN_DATASET_PATH +
'BraTS20_Training_001/BraTS20_Training_001_flair.nii',
'./test_gif_BraTS20_Training_001_flair.nii')
gif2nif.write_gif_normal('./test_gif_BraTS20_Training_001_flair.nii')

niimg = nl.image.load_img(TRAIN_DATASET_PATH +
'BraTS20_Training_001/BraTS20_Training_001_flair.nii')
nimask = nl.image.load_img(TRAIN_DATASET_PATH +
'BraTS20_Training_001/BraTS20_Training_001_seg.nii')

fig, axes = plt.subplots(nrows=4, figsize=(30, 40))


nlplt.plot_anat(niimg,
                title='BraTS20_Training_001_flair.nii plot_anat',
                axes=axes[0])

nlplt.plot_epi(niimg,
               title='BraTS20_Training_001_flair.nii plot_epi',
               axes=axes[1])

nlplt.plot_img(niimg,
```

```
                title='BraTS20_Training_001_flair.nii plot_img',
                axes=axes[2])

nlplt.plot_roi(nimask,
                title='BraTS20_Training_001_flair.nii with mask plot_roi',
                bg_img=niimg,
                axes=axes[3], cmap='Paired')

plt.show()
```










```
def dice_coef(y_true, y_pred, smooth=1.0):
    class_num = 4
    for i in range(class_num):
        y_true_f = K.flatten(y_true[:,:,:,i])
        y_pred_f = K.flatten(y_pred[:,:,:,i])
        intersection = K.sum(y_true_f * y_pred_f)
        loss = ((2. * intersection + smooth) / (K.sum(y_true_f) +
```

```
K.sum(y_pred_f) + smooth))
        if i == 0:
            total_loss = loss
        else:
            total_loss = total_loss + loss
    total_loss = total_loss / class_num
    return total_loss

def dice_coef_necrotic(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[:,:,:,1] * y_pred[:,:,:,1]))
    return (2. * intersection) / (K.sum(K.square(y_true[:,:,:,1])) +
K.sum(K.square(y_pred[:,:,:,1])) + epsilon)

def dice_coef_edema(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[:,:,:,2] * y_pred[:,:,:,2]))
    return (2. * intersection) / (K.sum(K.square(y_true[:,:,:,2])) +
K.sum(K.square(y_pred[:,:,:,2])) + epsilon)

def dice_coef_enhancing(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[:,:,:,3] * y_pred[:,:,:,3]))
    return (2. * intersection) / (K.sum(K.square(y_true[:,:,:,3])) +
K.sum(K.square(y_pred[:,:,:,3])) + epsilon)

def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision

def sensitivity(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    return true_positives / (possible_positives + K.epsilon())

def specificity(y_true, y_pred):
    true_negatives = K.sum(K.round(K.clip((1-y_true) * (1-y_pred), 0, 1)))
    possible_negatives = K.sum(K.round(K.clip(1-y_true, 0, 1)))
    return true_negatives / (possible_negatives + K.epsilon())

IMG_SIZE=128

from tensorflow.keras import layers, models, Input
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D,
concatenate, Dropout, BatchNormalization

def build_improved_unet(inputs, ker_init, dropout):
    def conv_block(input_tensor, num_filters):
        conv = Conv2D(num_filters, 3, activation='relu', padding='same',
kernel_initializer=ker_init)(input_tensor)
        conv = BatchNormalization()(conv)
```

```
        conv = Conv2D(num_filters, 3, activation='relu', padding='same',
kernel_initializer=ker_init)(conv)
        conv = BatchNormalization()(conv)
        return conv

    conv1 = conv_block(inputs, 32)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = conv_block(pool1, 64)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = conv_block(pool2, 128)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = conv_block(pool3, 256)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = conv_block(pool4, 512)
    drop5 = Dropout(dropout)(conv5)

    up6 = Conv2D(256, 2, activation='relu', padding='same',
kernel_initializer=ker_init)(UpSampling2D(size=(2, 2))(drop5))
    merge6 = concatenate([conv4, up6], axis=3)
    conv6 = conv_block(merge6, 256)

    up7 = Conv2D(128, 2, activation='relu', padding='same',
kernel_initializer=ker_init)(UpSampling2D(size=(2, 2))(conv6))
    merge7 = concatenate([conv3, up7], axis=3)
    conv7 = conv_block(merge7, 128)

    up8 = Conv2D(64, 2, activation='relu', padding='same',
kernel_initializer=ker_init)(UpSampling2D(size=(2, 2))(conv7))
    merge8 = concatenate([conv2, up8], axis=3)
    conv8 = conv_block(merge8, 64)

    up9 = Conv2D(32, 2, activation='relu', padding='same',
kernel_initializer=ker_init)(UpSampling2D(size=(2, 2))(conv8))
    merge9 = concatenate([conv1, up9], axis=3)
    conv9 = conv_block(merge9, 32)

    conv10 = Conv2D(4, (1, 1), activation='softmax')(conv9)

    return models.Model(inputs=inputs, outputs=conv10)

input_layer = Input((IMG_SIZE, IMG_SIZE, 2))
improved_model = build_improved_unet(input_layer, 'he_normal', 0.2)
improved_model.compile(loss="categorical_crossentropy",
optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics=['accuracy',
```

```
tf.keras.metrics.MeanIoU(num_classes=4), dice_coef, precision, sensitivity,
specificity, dice_coef_necrotic, dice_coef_edema, dice_coef_enhancing])

improved_model.summary()

Model: "model"
__________________________________________________________________________________
____________________
Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
====================
input_1 (InputLayer)           [(None, 128, 128, 2) 0

__________________________________________________________________________________
____________________
conv2d (Conv2D)                (None, 128, 128, 32) 608
input_1[0][0]
__________________________________________________________________________________
____________________
batch_normalization (BatchNorma (None, 128, 128, 32) 128         conv2d[0][0]
__________________________________________________________________________________
____________________
conv2d_1 (Conv2D)              (None, 128, 128, 32) 9248
batch_normalization[0][0]
__________________________________________________________________________________
____________________
batch_normalization_1 (BatchNor (None, 128, 128, 32) 128
conv2d_1[0][0]
__________________________________________________________________________________
____________________
max_pooling2d (MaxPooling2D)   (None, 64, 64, 32)   0
batch_normalization_1[0][0]
__________________________________________________________________________________
____________________
conv2d_2 (Conv2D)              (None, 64, 64, 64)   18496
max_pooling2d[0][0]
__________________________________________________________________________________
____________________
batch_normalization_2 (BatchNor (None, 64, 64, 64)   256
conv2d_2[0][0]
__________________________________________________________________________________
____________________
conv2d_3 (Conv2D)              (None, 64, 64, 64)   36928
batch_normalization_2[0][0]
__________________________________________________________________________________
____________________
batch_normalization_3 (BatchNor (None, 64, 64, 64)   256
conv2d_3[0][0]
__________________________________________________________________________________
____________________
max_pooling2d_1 (MaxPooling2D)  (None, 32, 32, 64)   0
```

```
batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 32, 32, 128)  73856
max_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 32, 32, 128)  512
conv2d_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 32, 32, 128)  147584
batch_normalization_4[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 32, 32, 128)  512
conv2d_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 16, 16, 128)  0
batch_normalization_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 16, 16, 256)  295168
max_pooling2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 16, 16, 256)  1024
conv2d_6[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 16, 16, 256)  590080
batch_normalization_6[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 16, 16, 256)  1024
conv2d_7[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 8, 8, 256)    0
batch_normalization_7[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 8, 8, 512)    1180160
max_pooling2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 8, 8, 512)    2048
conv2d_8[0][0]
__________________________________________________________________________________________________
```

```
conv2d_9 (Conv2D)                (None, 8, 8, 512)    2359808
batch_normalization_8[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 8, 8, 512)    2048
conv2d_9[0][0]
__________________________________________________________________________________________________
dropout (Dropout)                (None, 8, 8, 512)    0
batch_normalization_9[0][0]
__________________________________________________________________________________________________
up_sampling2d (UpSampling2D)    (None, 16, 16, 512)  0
dropout[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)               (None, 16, 16, 256)  524544
up_sampling2d[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)        (None, 16, 16, 512)  0
batch_normalization_7[0][0]

conv2d_10[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)               (None, 16, 16, 256)  1179904
concatenate[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 16, 16, 256)  1024
conv2d_11[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)               (None, 16, 16, 256)  590080
batch_normalization_10[0][0]
__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 16, 16, 256)  1024
conv2d_12[0][0]
__________________________________________________________________________________________________
up_sampling2d_1 (UpSampling2D)  (None, 32, 32, 256)  0
batch_normalization_11[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)               (None, 32, 32, 128)  131200
up_sampling2d_1[0][0]
__________________________________________________________________________________________________
```

```
________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 32, 32, 256)  0
batch_normalization_5[0][0]

conv2d_13[0][0]
________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 32, 32, 128)  295040
concatenate_1[0][0]
________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 32, 32, 128)  512
conv2d_14[0][0]
________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 32, 32, 128)  147584
batch_normalization_12[0][0]
________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 32, 32, 128)  512
conv2d_15[0][0]
________________________________________________________________________________________________
up_sampling2d_2 (UpSampling2D)  (None, 64, 64, 128)  0
batch_normalization_13[0][0]
________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 64, 64, 64)   32832
up_sampling2d_2[0][0]
________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 64, 64, 128)  0
batch_normalization_3[0][0]

conv2d_16[0][0]
________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 64, 64, 64)   73792
concatenate_2[0][0]
________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 64, 64, 64)   256
conv2d_17[0][0]
________________________________________________________________________________________________
conv2d_18 (Conv2D)              (None, 64, 64, 64)   36928
batch_normalization_14[0][0]
________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 64, 64, 64)   256
```

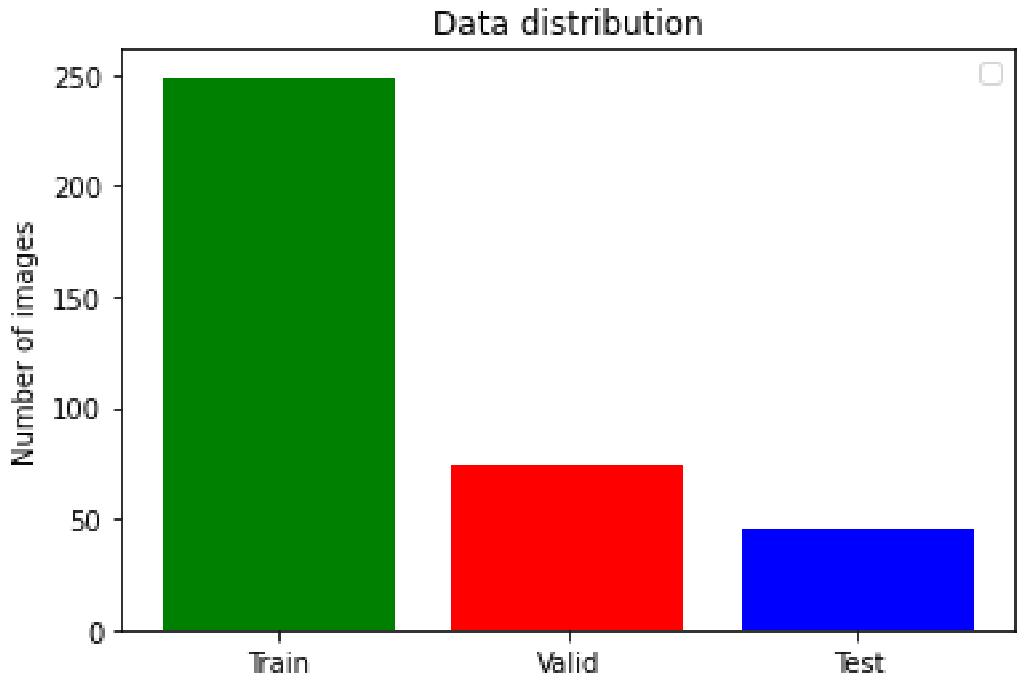
```
conv2d_18[0][0]
__________________________________________________________________________________________________
up_sampling2d_3 (UpSampling2D)  (None, 128, 128, 64) 0
batch_normalization_15[0][0]
__________________________________________________________________________________________________
conv2d_19 (Conv2D)              (None, 128, 128, 32) 8224
up_sampling2d_3[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 128, 128, 64) 0
batch_normalization_1[0][0]

conv2d_19[0][0]
__________________________________________________________________________________________________
conv2d_20 (Conv2D)              (None, 128, 128, 32) 18464
concatenate_3[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 128, 128, 32) 128
conv2d_20[0][0]
__________________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, 128, 128, 32) 9248
batch_normalization_16[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 128, 128, 32) 128
conv2d_21[0][0]
__________________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 128, 128, 4)  132
batch_normalization_17[0][0]
==================================================================================================
Total params: 7,771,684
Trainable params: 7,765,796
Non-trainable params: 5,888
__________________________________________________________________________________________________

def build_unet(inputs, ker_init, dropout):
    conv1 = Conv2D(32, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(inputs)
    conv1 = Conv2D(32, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv1)

    pool = MaxPooling2D(pool_size=(2, 2))(conv1)
```

```
    conv = Conv2D(64, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(pool)
    conv = Conv2D(64, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv)

    pool1 = MaxPooling2D(pool_size=(2, 2))(conv)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(pool1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv2)

    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(pool2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv3)


    pool4 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv5 = Conv2D(512, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(pool4)
    conv5 = Conv2D(512, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv5)
    drop5 = Dropout(dropout)(conv5)

    up7 = Conv2D(256, 2, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(UpSampling2D(size = (2,2))(drop5))
    merge7 = concatenate([conv3,up7], axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(merge7)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv7)

    up8 = Conv2D(128, 2, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(UpSampling2D(size = (2,2))(conv7))
    merge8 = concatenate([conv2,up8], axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(merge8)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv8)

    up9 = Conv2D(64, 2, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(UpSampling2D(size = (2,2))(conv8))
    merge9 = concatenate([conv,up9], axis = 3)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(merge9)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv9)
```

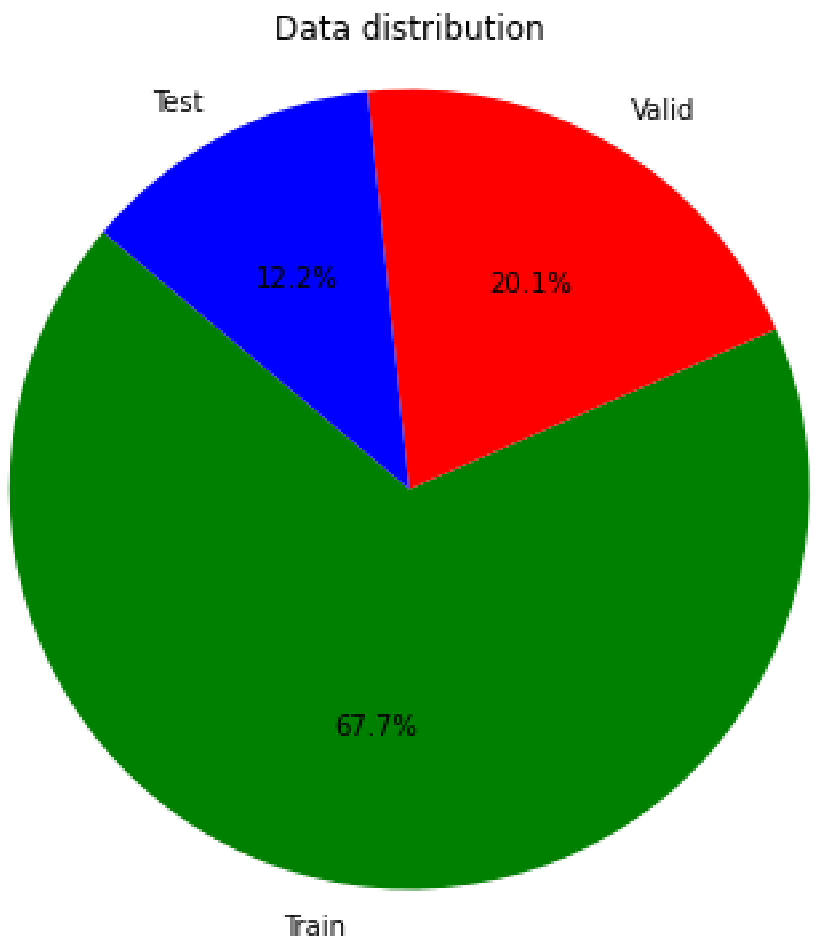
```
    up = Conv2D(32, 2, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(UpSampling2D(size = (2,2))(conv9))
    merge = concatenate([conv1,up], axis = 3)
    conv = Conv2D(32, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(merge)
    conv = Conv2D(32, 3, activation = 'relu', padding = 'same',
kernel_initializer = ker_init)(conv)

    conv10 = Conv2D(4, (1,1), activation = 'softmax')(conv)

    return Model(inputs = inputs, outputs = conv10)

input_layer = Input((IMG_SIZE, IMG_SIZE, 2))

model = build_unet(input_layer, 'he_normal', 0.2)
model.compile(loss="categorical_crossentropy",
optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics =
['accuracy',tf.keras.metrics.MeanIoU(num_classes=4), dice_coef, precision,
sensitivity, specificity, dice_coef_necrotic, dice_coef_edema
,dice_coef_enhancing] )

plot_model(improved_model,
           show_shapes = True,
           show_dtype=False,
           show_layer_names = True,
           rankdir = 'TB',
           expand_nested = False,
           dpi = 70)


train_and_val_directories = [f.path for f in os.scandir(TRAIN_DATASET_PATH)
if f.is_dir()]

train_and_val_directories.remove(TRAIN_DATASET_PATH+'BraTS20_Training_355')


def pathListIntoIds(dirList):
    x = []
    for i in range(0,len(dirList)):
        x.append(dirList[i][dirList[i].rfind('/')+1:])
    return x

train_and_test_ids = pathListIntoIds(train_and_val_directories);


train_test_ids, val_ids = train_test_split(train_and_test_ids,test_size=0.2)
train_ids, test_ids = train_test_split(train_test_ids,test_size=0.15)
```

```
class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, dim=(IMG_SIZE,IMG_SIZE), batch_size = 1,
n_channels = 2, shuffle=True):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.list_IDs = list_IDs
        self.n_channels = n_channels
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        indexes =
self.indexes[index*self.batch_size:(index+1)*self.batch_size]
        Batch_ids = [self.list_IDs[k] for k in indexes]
        X, y = self.__data_generation(Batch_ids)

        return X, y

    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)

    def __data_generation(self, Batch_ids):
        'Generates data containing batch_size samples'
        X = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim,
self.n_channels))
        y = np.zeros((self.batch_size*VOLUME_SLICES, 240, 240))
        Y = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, 4))


        for c, i in enumerate(Batch_ids):
            case_path = os.path.join(TRAIN_DATASET_PATH, i)

            data_path = os.path.join(case_path, f'{i}_flair.nii');
            flair = nib.load(data_path).get_fdata()

            data_path = os.path.join(case_path, f'{i}_t1ce.nii');
            ce = nib.load(data_path).get_fdata()

            data_path = os.path.join(case_path, f'{i}_seg.nii');
```

```
            seg = nib.load(data_path).get_fdata()

            for j in range(VOLUME_SLICES):
                 X[j +VOLUME_SLICES*c,:,:,0] =
cv2.resize(flair[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE));
                 X[j +VOLUME_SLICES*c,:,:,1] =
cv2.resize(ce[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE));

                 y[j +VOLUME_SLICES*c] = seg[:,:,j+VOLUME_START_AT];

        y[y==4] = 3;
        mask = tf.one_hot(y, 4);
        Y = tf.image.resize(mask, (IMG_SIZE, IMG_SIZE));
        return X/np.max(X), Y

training_generator = DataGenerator(train_ids)
valid_generator = DataGenerator(val_ids)
test_generator = DataGenerator(test_ids)

def showDataLayout():
    plt.bar(["Train","Valid","Test"],
    [len(train_ids), len(val_ids), len(test_ids)], align='center',color=[
'green','red', 'blue'])
    plt.legend()

    plt.ylabel('Number of images')
    plt.title('Data distribution')

    plt.show()

showDataLayout()
```




```
def showDataLayout():
    labels = ["Train", "Valid", "Test"]
    sizes = [len(train_ids), len(val_ids), len(test_ids)]
    colors = ['green', 'red', 'blue']

    plt.figure(figsize=(6, 6))
    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',
startangle=140)
    plt.title('Data distribution')
    plt.axis('equal')

    plt.show()

showDataLayout()
```




```
csv_logger = CSVLogger('training.log', separator=',', append=False)
callbacks = [
      keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=2, min_lr=0.000001, verbose=1),
        csv_logger
    ]

K.clear_session()
history = improved_model.fit(training_generator,
                    epochs=5,
                     steps_per_epoch=len(train_ids),
                     callbacks= callbacks,
                     validation_data = valid_generator
                     )
```

```
Epoch 1/5
249/249 [==============================] - 216s 833ms/step - loss: 0.8919 -
accuracy: 0.8354 - mean_io_u: 0.3756 - dice_coef: 0.1800 - precision: 0.7696
- sensitivity: 0.4265 - specificity: 0.9878 - dice_coef_necrotic: 0.0579 -
dice_coef_edema: 0.2102 - dice_coef_enhancing: 0.0956 - val_loss: 3.9397 -
val_accuracy: 0.6902 - val_mean_io_u: 0.3756 - val_dice_coef: 0.2108 -
val_precision: 0.8034 - val_sensitivity: 0.6487 - val_specificity: 0.9470 -
```

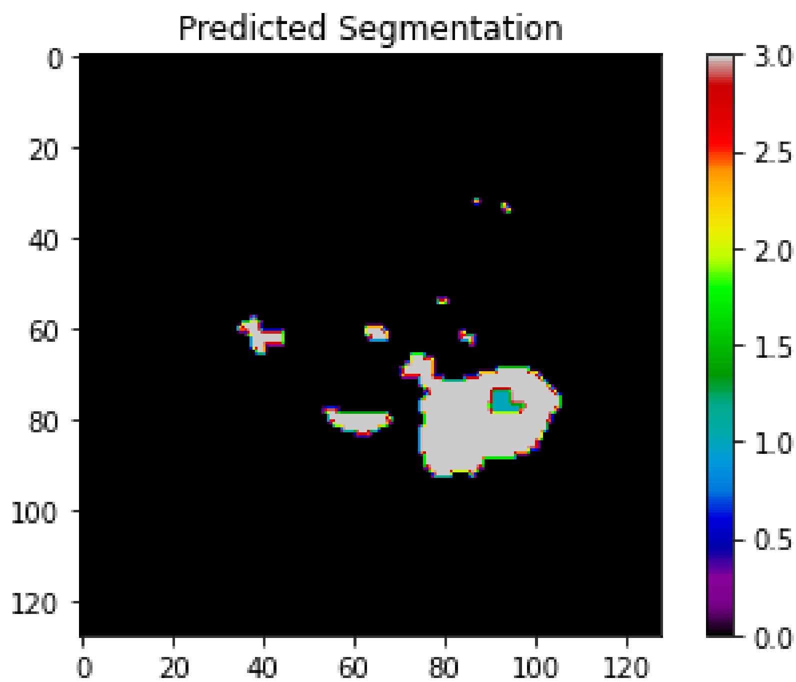
```
val_dice_coef_necrotic: 0.0473 - val_dice_coef_edema: 0.1420 -
val_dice_coef_enhancing: 0.0503
Epoch 2/5
249/249 [==============================] - 118s 472ms/step - loss: 0.0831 -
accuracy: 0.9882 - mean_io_u: 0.3756 - dice_coef: 0.3458 - precision: 0.9905
- sensitivity: 0.9856 - specificity: 0.9968 - dice_coef_necrotic: 0.2357 -
dice_coef_edema: 0.5405 - dice_coef_enhancing: 0.4251 - val_loss: 1.1279 -
val_accuracy: 0.9465 - val_mean_io_u: 0.3756 - val_dice_coef: 0.2988 -
val_precision: 0.9523 - val_sensitivity: 0.9425 - val_specificity: 0.9844 -
val_dice_coef_necrotic: 0.0863 - val_dice_coef_edema: 0.2563 -
val_dice_coef_enhancing: 0.2466
Epoch 3/5
249/249 [==============================] - 117s 469ms/step - loss: 0.0461 -
accuracy: 0.9894 - mean_io_u: 0.3756 - dice_coef: 0.4208 - precision: 0.9910
- sensitivity: 0.9872 - specificity: 0.9970 - dice_coef_necrotic: 0.2761 -
dice_coef_edema: 0.5894 - dice_coef_enhancing: 0.5515 - val_loss: 2.3090 -
val_accuracy: 0.9433 - val_mean_io_u: 0.3756 - val_dice_coef: 0.3214 -
val_precision: 0.9510 - val_sensitivity: 0.9400 - val_specificity: 0.9841 -
val_dice_coef_necrotic: 0.1185 - val_dice_coef_edema: 0.1589 -
val_dice_coef_enhancing: 0.3189
Epoch 4/5
249/249 [==============================] - 117s 470ms/step - loss: 0.0377 -
accuracy: 0.9899 - mean_io_u: 0.3756 - dice_coef: 0.4502 - precision: 0.9913
- sensitivity: 0.9879 - specificity: 0.9971 - dice_coef_necrotic: 0.3002 -
dice_coef_edema: 0.5732 - dice_coef_enhancing: 0.5718 - val_loss: 0.5554 -
val_accuracy: 0.9547 - val_mean_io_u: 0.3756 - val_dice_coef: 0.3183 -
val_precision: 0.9627 - val_sensitivity: 0.9502 - val_specificity: 0.9878 -
val_dice_coef_necrotic: 0.0779 - val_dice_coef_edema: 0.1482 -
val_dice_coef_enhancing: 0.2835
Epoch 5/5
249/249 [==============================] - 117s 470ms/step - loss: 0.0315 -
accuracy: 0.9905 - mean_io_u: 0.3756 - dice_coef: 0.4872 - precision: 0.9918
- sensitivity: 0.9886 - specificity: 0.9972 - dice_coef_necrotic: 0.3310 -
dice_coef_edema: 0.6118 - dice_coef_enhancing: 0.6151 - val_loss: 0.6838 -
val_accuracy: 0.9528 - val_mean_io_u: 0.3756 - val_dice_coef: 0.3379 -
val_precision: 0.9588 - val_sensitivity: 0.9500 - val_specificity: 0.9866 -
val_dice_coef_necrotic: 0.1131 - val_dice_coef_edema: 0.1659 -
val_dice_coef_enhancing: 0.2944

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-08b3f0c01a12> in <module>
      6                       validation_data = valid_generator
      7                       )
----> 8 model.save("3D_MRI_Brain_tumor_segmentation.h5")

NameError: name 'model' is not defined
```

```
improved_model.save('3D_MRI_Brain_tumor_segmentation.h5')
print("Model saved successfully as 3D_MRI_Brain_tumor_segmentation.h5")
```

```
Model saved successfully as 3D_MRI_Brain_tumor_segmentation.h5
```

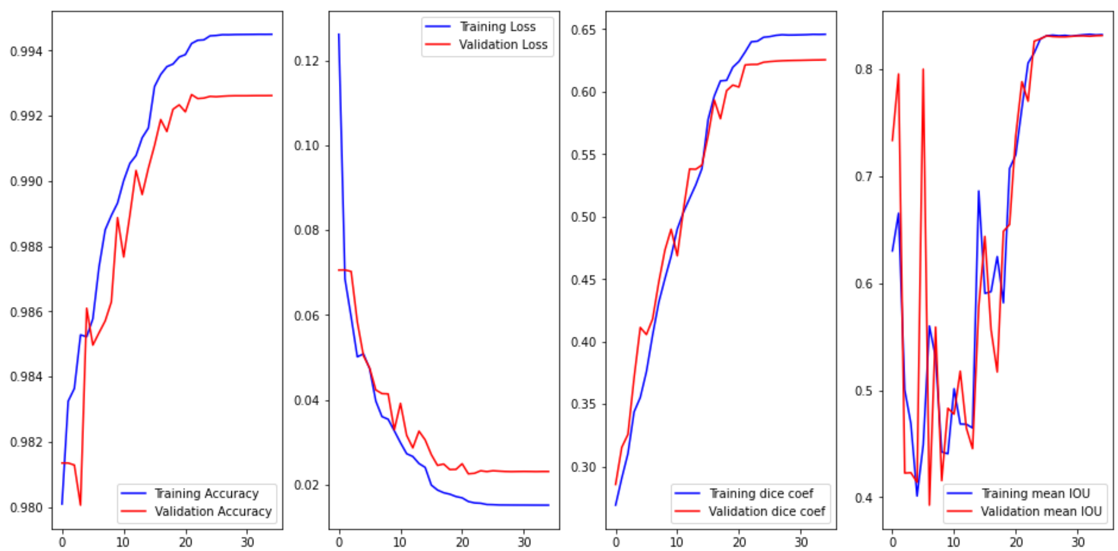
```python
import tensorflow as tf
import numpy as np
import nibabel as nib
import cv2

model = tf.keras.models.load_model("3D_MRI_Brain_tumor_segmentation.h5",
custom_objects={
    'dice_coef': dice_coef,
    'precision': precision,
    'sensitivity': sensitivity,
    'specificity': specificity,
    'dice_coef_necrotic': dice_coef_necrotic,
    'dice_coef_edema': dice_coef_edema,
    'dice_coef_enhancing': dice_coef_enhancing
})

def preprocess_image(image_file, slice_index=None):

    img = nib.load(image_file).get_fdata()

    if slice_index is not None:
        img = img[:, :, slice_index]

    img_resized = cv2.resize(img, (IMG_SIZE, IMG_SIZE))

    img_resized = img_resized / np.max(img_resized)

    return img_resized

def predict(image_paths, slice_index):
    X = np.zeros((1, IMG_SIZE, IMG_SIZE, 2))

    X[0, :, :, 0] = preprocess_image(image_paths[0], slice_index)
    X[0, :, :, 1] = preprocess_image(image_paths[1], slice_index)

    pred = model.predict(X)

    return np.argmax(pred[0], axis=-1)

image_paths = [
    '/kaggle/input/brats20-dataset-training-
validation/BraTS2020_ValidationData/MICCAI_BraTS2020_ValidationData/BraTS20_V
alidation_010/BraTS20_Validation_010_flair.nii',
    '/kaggle/input/brats20-dataset-training-
validation/BraTS2020_ValidationData/MICCAI_BraTS2020_ValidationData/BraTS20_V
alidation_010/BraTS20_Validation_010_t1ce.nii'
]
```

```python
slice_index = 75
prediction = predict(image_paths, slice_index)

import numpy as np

def get_classification(pred):
    class_predictions = np.argmax(pred, axis=-1)
    return class_predictions

get_classification(prediction)

array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 87, 93,
       94,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0, 79,  0,  0,  0, 38, 36, 35, 37, 38, 38, 38, 39, 73, 73,
       72, 71, 71, 75, 75, 76, 77, 76, 75, 75, 54, 55, 55, 56, 58, 61, 75,
       75, 75, 75, 76, 76, 76, 77, 78,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0])

import matplotlib.pyplot as plt

def visualize_prediction(prediction):
    plt.imshow(prediction, cmap='nipy_spectral')
    plt.title('Predicted Segmentation')
    plt.colorbar()
    plt.show()

visualize_prediction(prediction)
```




```
model =
keras.models.load_model('../input/modelperclasseval/model_per_class.h5',
                                   custom_objects={ 'accuracy' :
tf.keras.metrics.MeanIoU(num_classes=4),
                                                   "dice_coef": dice_coef,
                                                   "precision": precision,
                                                   "sensitivity":sensitivity,
                                                   "specificity":specificity,
                                                   "dice_coef_necrotic":
dice_coef_necrotic,
                                                   "dice_coef_edema":
dice_coef_edema,
                                                   "dice_coef_enhancing":
dice_coef_enhancing
                                                  }, compile=False)

history = pd.read_csv('../input/modelperclasseval/training_per_class.log',
sep=',', engine='python')

hist=history

acc=hist['accuracy']
val_acc=hist['val_accuracy']

epoch=range(len(acc))

loss=hist['loss']
val_loss=hist['val_loss']
```

```
train_dice=hist['dice_coef']
val_dice=hist['val_dice_coef']

f,ax=plt.subplots(1,4,figsize=(16,8))

ax[0].plot(epoch,acc,'b',label='Training Accuracy')
ax[0].plot(epoch,val_acc,'r',label='Validation Accuracy')
ax[0].legend()

ax[1].plot(epoch,loss,'b',label='Training Loss')
ax[1].plot(epoch,val_loss,'r',label='Validation Loss')
ax[1].legend()

ax[2].plot(epoch,train_dice,'b',label='Training dice coef')
ax[2].plot(epoch,val_dice,'r',label='Validation dice coef')
ax[2].legend()

ax[3].plot(epoch,hist['mean_io_u'],'b',label='Training mean IOU')
ax[3].plot(epoch,hist['val_mean_io_u'],'r',label='Validation mean IOU')
ax[3].legend()

plt.show()
```

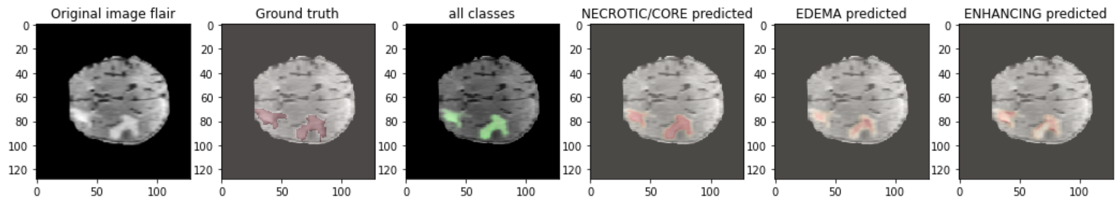



```
def imageLoader(path):
    image = nib.load(path).get_fdata()
    X = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, self.n_channels))
    for j in range(VOLUME_SLICES):
        X[j +VOLUME_SLICES*c,:,:,0] =
cv2.resize(image[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE));
        X[j +VOLUME_SLICES*c,:,:,1] = cv2.resize(ce[:,:,j+VOLUME_START_AT],
(IMG_SIZE, IMG_SIZE));
```

```
        y[j +VOLUME_SLICES*c] = seg[:,:,j+VOLUME_START_AT];
    return np.array(image)

def loadDataFromDir(path, list_of_files, mriType, n_images):
    scans = []
    masks = []
    for i in list_of_files[:n_images]:
        fullPath = glob.glob( i + '/*'+ mriType +'*')[0]
        currentScanVolume = imageLoader(fullPath)
        currentMaskVolume = imageLoader( glob.glob( i + '/*seg*')[0] )

        for j in range(0, currentScanVolume.shape[2]):
            scan_img = cv2.resize(currentScanVolume[:,:,j],
dsize=(IMG_SIZE,IMG_SIZE), interpolation=cv2.INTER_AREA).astype('uint8')
            mask_img = cv2.resize(currentMaskVolume[:,:,j],
dsize=(IMG_SIZE,IMG_SIZE), interpolation=cv2.INTER_AREA).astype('uint8')
            scans.append(scan_img[..., np.newaxis])
            masks.append(mask_img[..., np.newaxis])
    return np.array(scans, dtype='float32'), np.array(masks, dtype='float32')

def predictByPath(case_path,case):
    files = next(os.walk(case_path))[2]
    X = np.empty((VOLUME_SLICES, IMG_SIZE, IMG_SIZE, 2))

    vol_path = os.path.join(case_path, f'BraTS20_Training_{case}_flair.nii');
    flair=nib.load(vol_path).get_fdata()

    vol_path = os.path.join(case_path, f'BraTS20_Training_{case}_t1ce.nii');
    ce=nib.load(vol_path).get_fdata()

    for j in range(VOLUME_SLICES):
        X[j,:,:,0] = cv2.resize(flair[:,:,j+VOLUME_START_AT],
(IMG_SIZE,IMG_SIZE))
        X[j,:,:,1] = cv2.resize(ce[:,:,j+VOLUME_START_AT],
(IMG_SIZE,IMG_SIZE))

    return model.predict(X/np.max(X), verbose=1)

def showPredictsById(case, start_slice = 60):
    path = f"../input/brats20-dataset-training-
validation/BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/BraTS20_Train
ing_{case}"
    gt = nib.load(os.path.join(path,
f'BraTS20_Training_{case}_seg.nii')).get_fdata()
    origImage = nib.load(os.path.join(path,
f'BraTS20_Training_{case}_flair.nii')).get_fdata()
    p = predictByPath(path,case)
```

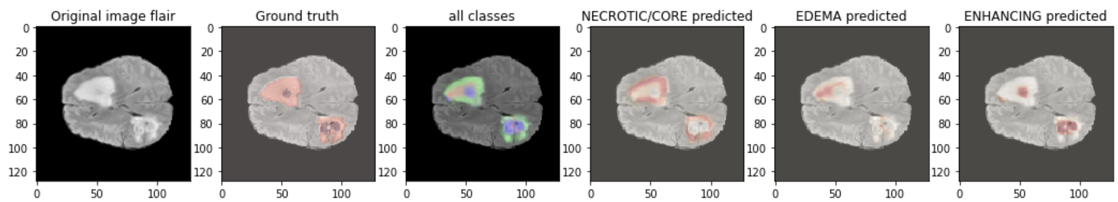
```
    core = p[:,:,:,1]
    edema= p[:,:,:,2]
    enhancing = p[:,:,:,3]

    plt.figure(figsize=(18, 50))
    f, axarr = plt.subplots(1,6, figsize = (18, 50))

    for i in range(6):

axarr[i].imshow(cv2.resize(origImage[:,:,start_slice+VOLUME_START_AT],
(IMG_SIZE, IMG_SIZE)), cmap="gray", interpolation='none')

    axarr[0].imshow(cv2.resize(origImage[:,:,start_slice+VOLUME_START_AT],
(IMG_SIZE, IMG_SIZE)), cmap="gray")
    axarr[0].title.set_text('Original image flair')
    curr_gt=cv2.resize(gt[:,:,start_slice+VOLUME_START_AT], (IMG_SIZE,
IMG_SIZE), interpolation = cv2.INTER_NEAREST)
    axarr[1].imshow(curr_gt, cmap="Reds", interpolation='none', alpha=0.3) #
,alpha=0.3,cmap='Reds'
    axarr[1].title.set_text('Ground truth')
    axarr[2].imshow(p[start_slice,:,:,1:4], cmap="Reds",
interpolation='none', alpha=0.3)
    axarr[2].title.set_text('all classes')
    axarr[3].imshow(edema[start_slice,:,:], cmap="OrRd",
interpolation='none', alpha=0.3)
    axarr[3].title.set_text(f'{SEGMENT_CLASSES[1]} predicted')
    axarr[4].imshow(core[start_slice,:,], cmap="OrRd", interpolation='none',
alpha=0.3)
    axarr[4].title.set_text(f'{SEGMENT_CLASSES[2]} predicted')
    axarr[5].imshow(enhancing[start_slice,:,], cmap="OrRd",
interpolation='none', alpha=0.3)
    axarr[5].title.set_text(f'{SEGMENT_CLASSES[3]} predicted')
    plt.show()


showPredictsById(case=test_ids[0][-3:])
showPredictsById(case=test_ids[1][-3:])
showPredictsById(case=test_ids[2][-3:])
showPredictsById(case=test_ids[3][-3:])
showPredictsById(case=test_ids[4][-3:])
showPredictsById(case=test_ids[5][-3:])
showPredictsById(case=test_ids[6][-3:])
```

```
4/4 [==============================] - 1s 96ms/step

<Figure size 1296x3600 with 0 Axes>
```



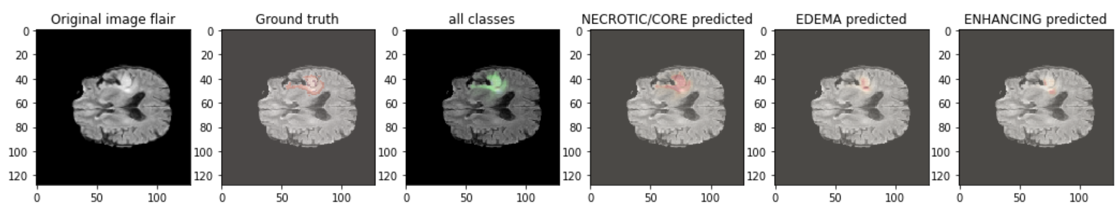
4/4 [==============================] - 0s 33ms/step

<Figure size 1296x3600 with 0 Axes>



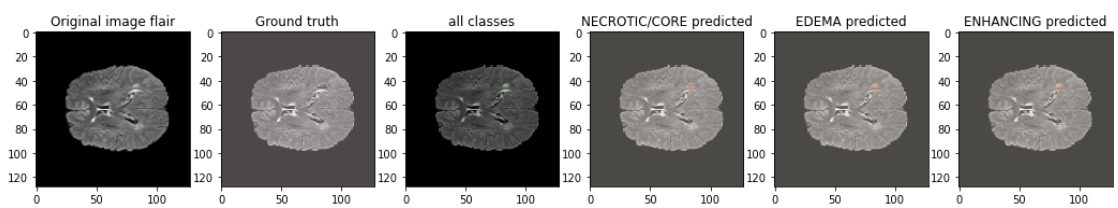
4/4 [==============================] - 0s 33ms/step

<Figure size 1296x3600 with 0 Axes>



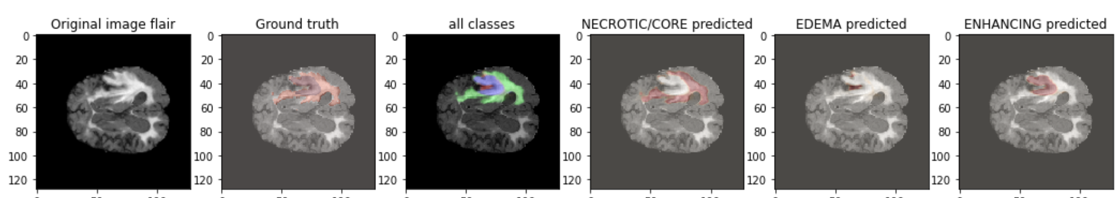
4/4 [==============================] - 0s 33ms/step

<Figure size 1296x3600 with 0 Axes>



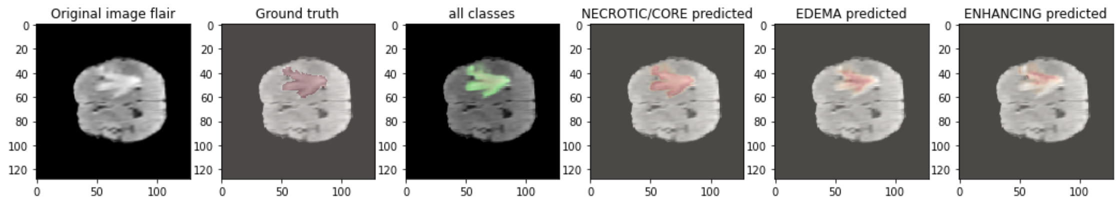
4/4 [==============================] - 0s 33ms/step

<Figure size 1296x3600 with 0 Axes>



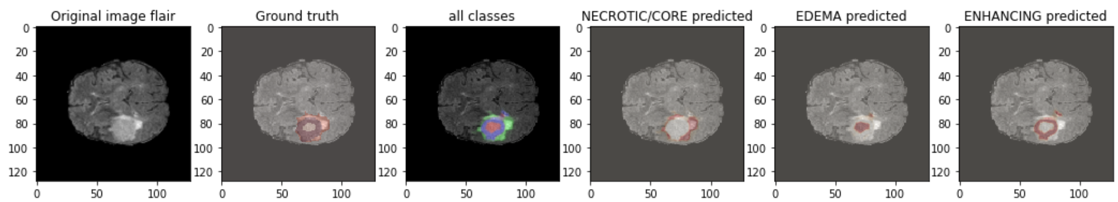
4/4 [==============================] - 0s 33ms/step

<Figure size 1296x3600 with 0 Axes>




```
4/4 [==============================] - 0s 33ms/step
```

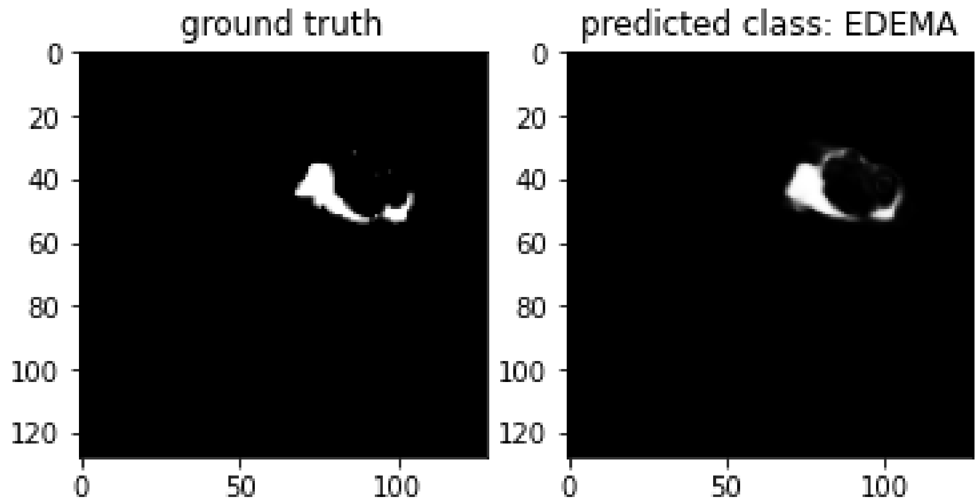
<Figure size 1296x3600 with 0 Axes>




```
case = case=test_ids[3][-3:]
path = f"../input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/BraTS20_Training_{case}"
gt = nib.load(os.path.join(path, f'BraTS20_Training_{case}_seg.nii')).get_fdata()
p = predictByPath(path,case)

core = p[:,:,:,1]
edema= p[:,:,:,2]
enhancing = p[:,:,:,3]

i=40
eval_class = 2

gt[gt != eval_class] = 1

resized_gt = cv2.resize(gt[:,:,i+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE))

plt.figure()
f, axarr = plt.subplots(1,2)
axarr[0].imshow(resized_gt, cmap="gray")
axarr[0].title.set_text('ground truth')
axarr[1].imshow(p[i,:,:,eval_class], cmap="gray")
axarr[1].title.set_text(f'predicted class: {SEGMENT_CLASSES[eval_class]}')
plt.show()
```

```
4/4 [==============================] - 0s 36ms/step
```

<Figure size 432x288 with 0 Axes>




```
improved_model.compile(loss="categorical_crossentropy",
optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics =
['accuracy',tf.keras.metrics.MeanIoU(num_classes=4), dice_coef, precision,
sensitivity, specificity, dice_coef_necrotic, dice_coef_edema,
dice_coef_enhancing] )
print("Evaluate on test data")
results = improved_model.evaluate(test_generator, batch_size=100, callbacks=
callbacks)
print("test loss, test acc:", results)
```

```
Evaluate on test data
45/45 [==============================] - 25s 538ms/step - loss: 1.3613 -
accuracy: 0.9620 - mean_io_u_1: 0.3755 - dice_coef: 0.3311 - precision:
0.9663 - sensitivity: 0.9603 - specificity: 0.9890 - dice_coef_necrotic:
0.1275 - dice_coef_edema: 0.1491 - dice_coef_enhancing: 0.2706
test loss, test acc: [1.4527249336242676, 0.9519450664520264,
0.37558794021606445, 0.33874499797821045, 0.9571685791015625,
0.9493675231933594, 0.9859769940376282, 0.122862309217453,
0.18301647901535034, 0.2746582329273224]
```