Setup env

In [ ]:
```
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


# neural imaging
import nilearn as nl
import nibabel as nib
import nilearn.plotting as nlplt
!pip install git+https://github.com/miykael/gif_your_nifti # nifti to gif
import gif_your_nifti.core as gif2nif


# ml libs
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
from tensorflow.keras.layers.experimental import preprocessing


# Make numpy printouts easier to read.
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-
160eymo5
  Running command git clone -q https://github.com/miykael/gif_your_nifti /
tmp/pip-req-build-160eymo5
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-pack
ages (from gif-your-nifti==0.2.2) (1.19.5)
Requirement already satisfied: nibabel in /opt/conda/lib/python3.7/site-pa
ckages (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/si
te-packages (from gif-your-nifti==0.2.2) (0.18.1)
Requirement already satisfied: pillow in /opt/conda/lib/python3.7/site-pac
kages (from imageio<3->gif-your-nifti==0.2.2) (7.2.0)
Requirement already satisfied: python-dateutil>=2.1 in /opt/conda/lib/pyth
on3.7/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: cycler>=0.10 in /opt/conda/lib/python3.7/si
te-packages (from matplotlib->gif-your-nifti==0.2.2) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python
3.7/site-packages (from matplotlib->gif-your-nifti==0.2.2) (1.3.1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packag
es (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: PyWavelets>=1.1.1 in /opt/conda/lib/python
3.7/site-packages (from scikit-image->gif-your-nifti==0.2.2) (1.1.1)
Requirement already satisfied: tifffile>=2019.7.26 in /opt/conda/lib/pytho
n3.7/site-packages (from scikit-image->gif-your-nifti==0.2.2) (2021.2.1)
Requirement already satisfied: networkx>=2.0 in /opt/conda/lib/python3.7/s
ite-packages (from scikit-image->gif-your-nifti==0.2.2) (2.5)
Requirement already satisfied: scipy>=1.0.1 in /opt/conda/lib/python3.7/si
te-packages (from scikit-image->gif-your-nifti==0.2.2) (1.5.4)
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) ... done
  Created wheel for gif-your-nifti: filename=gif_your_nifti-0.2.2-py3-none
-any.whl size=6634 sha256=1dc34e20147dd0594de0ac77341ea58aab5b1a6da4caf54e
adc56f4803e686c0
  Stored in directory: /tmp/pip-ephem-wheel-cache-gvjm75pj/wheels/4a/8c/d
1/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 availa
ble.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip ins
tall --upgrade pip' command.
```

In [2]:

```
# DEFINE seg-areas
SEGMENT_CLASSES = {
    0 : 'NOT tumor',
    1 : 'NECROTIC/CORE', # or NON-ENHANCING tumor CORE
```

```
    2 : 'EDEMA',
    3 : 'ENHANCING' # original 4 -> converted into 3 later
}

# there are 155 slices per volume
# to start at 5 and use 145 slices means we will skip the first 5 and las
VOLUME_SLICES = 100
VOLUME_START_AT = 22 # first slice of volume that we will include
```

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

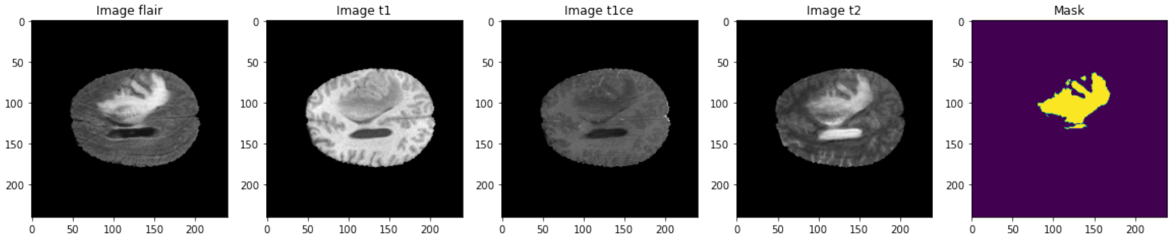
Data were acquired with different clinical protocols and various scanners from multiple (n=19) institutions.

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.

In [3]:
```
TRAIN_DATASET_PATH = '../input/brats20-dataset-training-validation/BraTS2
VALIDATION_DATASET_PATH = '../input/brats20-dataset-training-validation/B

test_image_flair=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraT
test_image_t1=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20
test_image_t1ce=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS
test_image_t2=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20
test_mask=nib.load(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20_Tra


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], cm
ax1.set_title('Image flair')
ax2.imshow(test_image_t1[:,:,test_image_t1.shape[0]//2-slice_w], cmap = '
ax2.set_title('Image t1')
ax3.imshow(test_image_t1ce[:,:,test_image_t1ce.shape[0]//2-slice_w], cmap
```

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

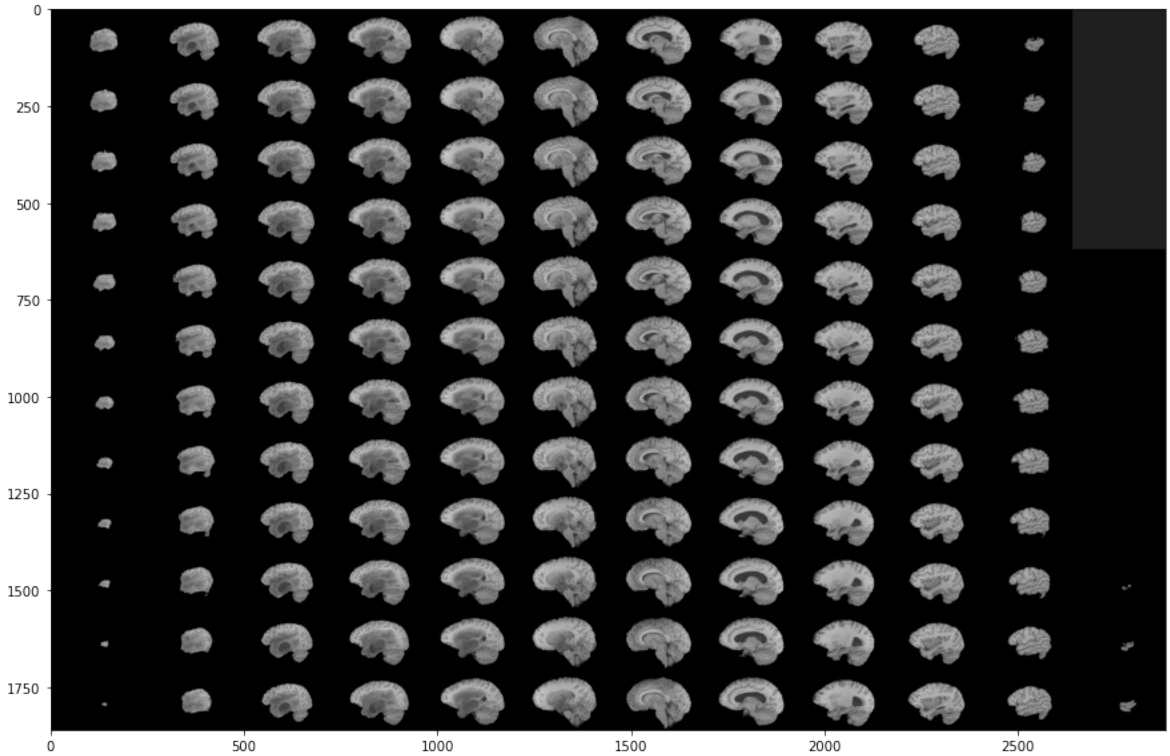
Out[3]: Text(0.5, 1.0, 'Mask')



Show whole nifti data -> print each slice from 3d data

In [4]:
```
# Skip 50:-50 slices since there is not much to see
fig, ax1 = plt.subplots(1, 1, figsize = (15,15))
ax1.imshow(rotate(montage(test_image_t1[50:-50,:,:]), 90, resize=True), c
```

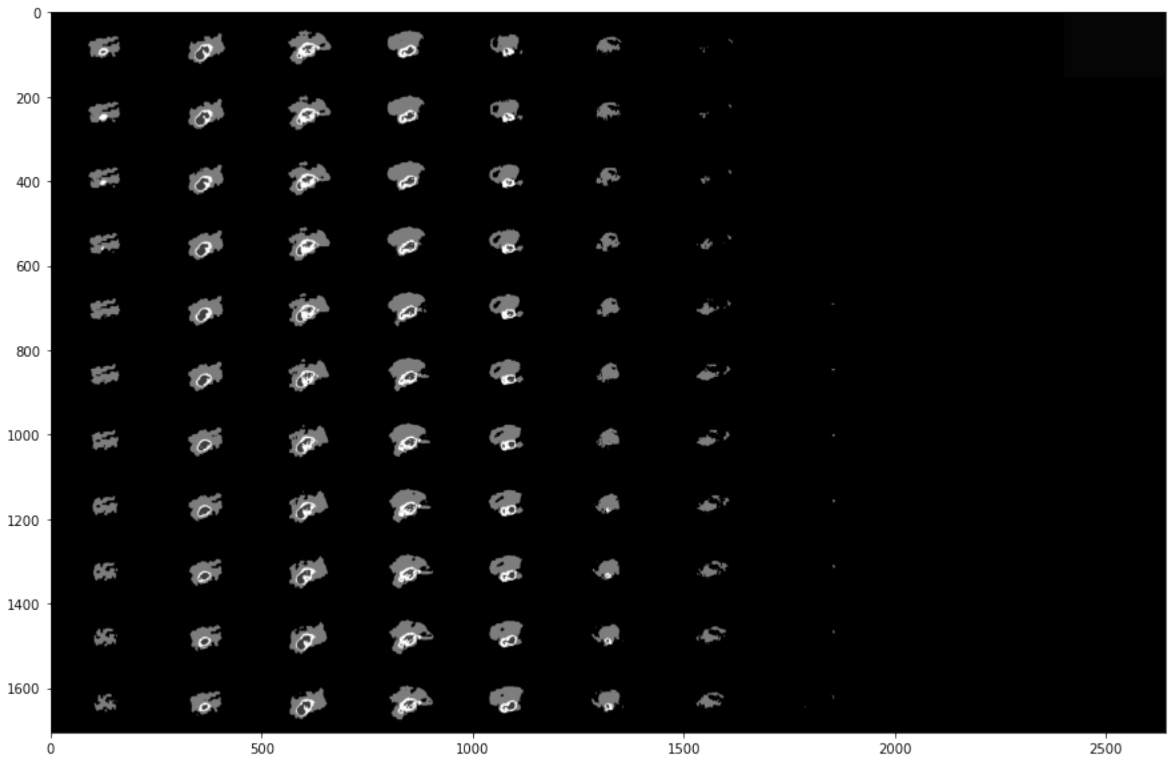
Out[4]: <matplotlib.image.AxesImage at 0x7bd3c8373510>



Show segment of tumor for each above slice

In [5]:
```
# Skip 50:-50 slices since there is not much to see
fig, ax1 = plt.subplots(1, 1, figsize = (15,15))
ax1.imshow(rotate(montage(test_mask[60:-60,:,:]), 90, resize=True), cmap
```

Out[5]: <matplotlib.image.AxesImage at 0x7bd3c86df750>



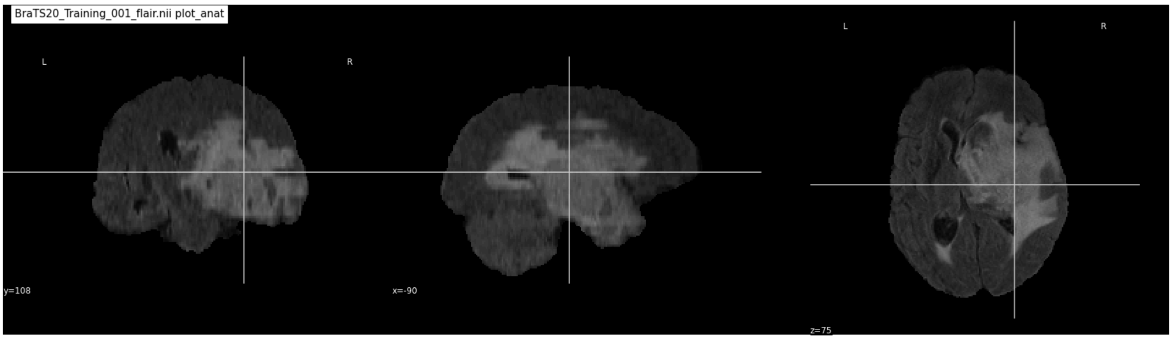
In [6]:
```
shutil.copy2(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraTS20_Training_
gif2nif.write_gif_normal('./test_gif_BraTS20_Training_001_flair.nii')
```

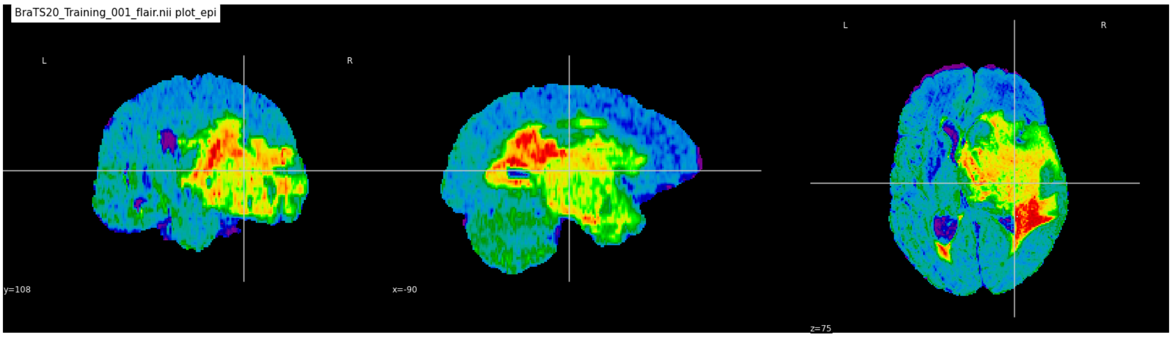
Gif representation of slices in 3D volume

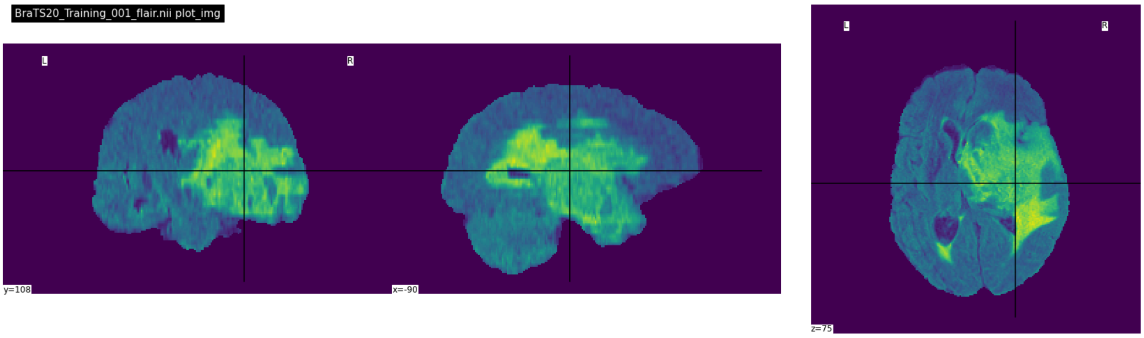
Show segments of tumor using different effects

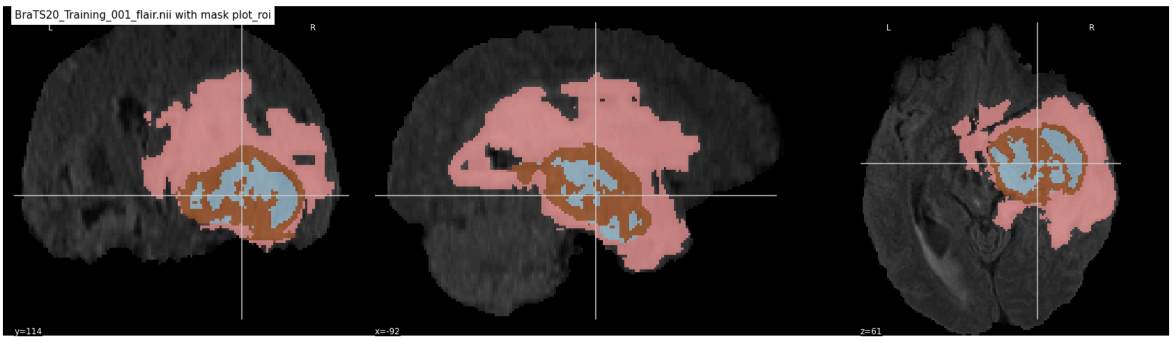
In [7]:
```
niimg = nl.image.load_img(TRAIN_DATASET_PATH + 'BraTS20_Training_001/BraT
nimask = nl.image.load_img(TRAIN_DATASET_PATH + 'BraTS20_Training_001/Bra

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()
```









In [8]:
```
# dice loss as defined above for 4 classes
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
   #    K.print_tensor(loss, message='loss value for class {} : '.format
        if i == 0:
            total_loss = loss
        else:
            total_loss = total_loss + loss
    total_loss = total_loss / class_num
#    K.print_tensor(total_loss, message=' total dice coef: ')
    return total_loss



# define per class evaluation of dice coef
# inspired by https://github.com/keras-team/keras/issues/9395
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.su

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.su

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.su



# Computing Precision
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


# Computing Sensitivity
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())


# Computing Specificity
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())
```

In [9]:

```
IMG_SIZE=128
```

In [10]:

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

    pool = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_in
    conv = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_in
```

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

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


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

    up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_in
    merge7 = concatenate([conv3,up7], axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_

    up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_in
    merge8 = concatenate([conv2,up8], axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_

    up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_ini
    merge9 = concatenate([conv,up9], axis = 3)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_i
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_i

    up = Conv2D(32, 2, activation = 'relu', padding = 'same', kernel_init
    merge = concatenate([conv1,up], axis = 3)
    conv = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_in
    conv = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_in

    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
```

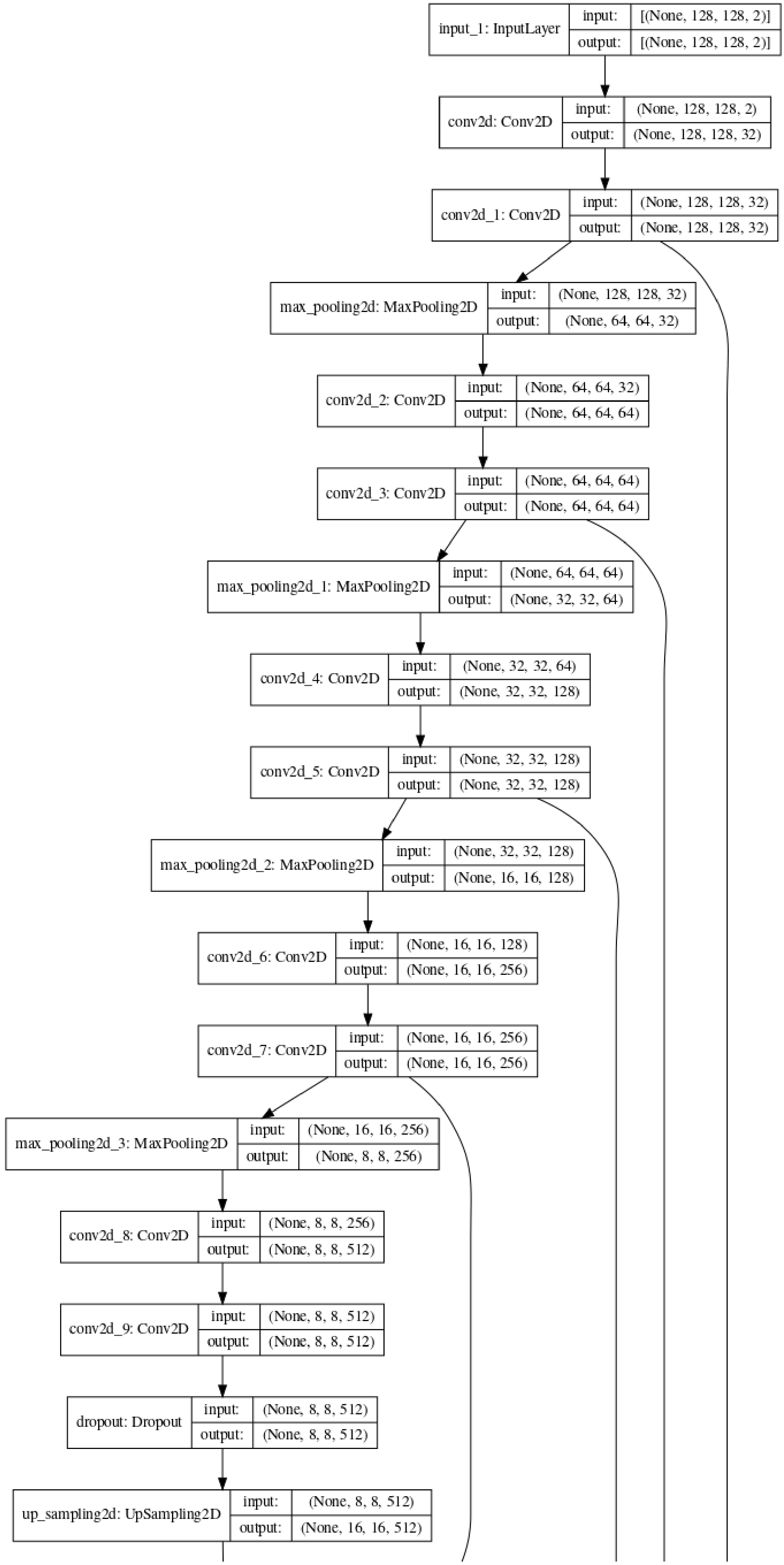
model architecture

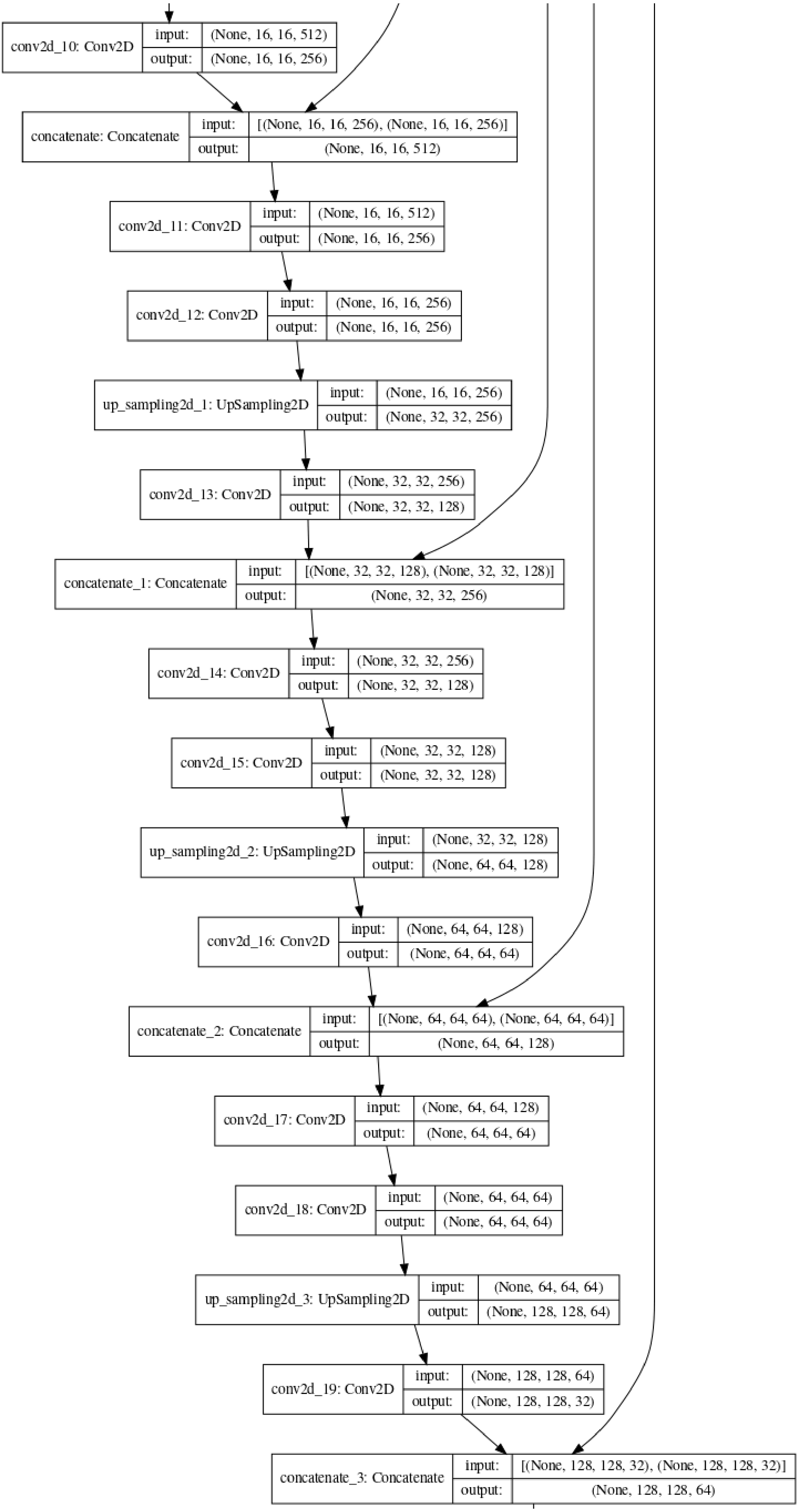
If you are about to use U-NET, I suggest to check out this awesome library that I found later, after manual implementation of U-NET keras-unet-collection, which also contains implementation of dice loss, tversky loss and many more!

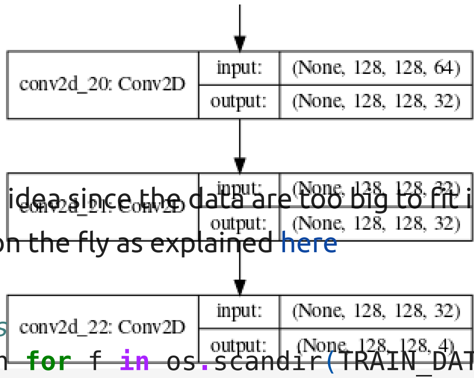
In [11]:

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

Out[11]:


input_1: InputLayer
input: [(None, 128, 128, 2)]
output: [(None, 128, 128, 2)]
conv2d: Conv2D
input: (None, 128, 128, 2)
output: (None, 128, 128, 32)
conv2d_1: Conv2D
input: (None, 128, 128, 32)
output: (None, 128, 128, 32)
max_pooling2d: MaxPooling2D
input: (None, 128, 128, 32)
output: (None, 64, 64, 32)
conv2d_2: Conv2D
input: (None, 64, 64, 32)
output: (None, 64, 64, 64)
conv2d_3: Conv2D
input: (None, 64, 64, 64)
output: (None, 64, 64, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 64, 64, 64)
output: (None, 32, 32, 64)
conv2d_4: Conv2D
input: (None, 32, 32, 64)
output: (None, 32, 32, 128)
conv2d_5: Conv2D
input: (None, 32, 32, 128)
output: (None, 32, 32, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 32, 32, 128)
output: (None, 16, 16, 128)
conv2d_6: Conv2D
input: (None, 16, 16, 128)
output: (None, 16, 16, 256)
conv2d_7: Conv2D
input: (None, 16, 16, 256)
output: (None, 16, 16, 256)
max_pooling2d_3: MaxPooling2D
input: (None, 16, 16, 256)
output: (None, 8, 8, 256)
conv2d_8: Conv2D
input: (None, 8, 8, 256)
output: (None, 8, 8, 512)
conv2d_9: Conv2D
input: (None, 8, 8, 512)
output: (None, 8, 8, 512)
dropout: Dropout
input: (None, 8, 8, 512)
output: (None, 8, 8, 512)
up_sampling2d: UpSampling2D
input: (None, 8, 8, 512)
output: (None, 16, 16, 512)


conv2d_10: Conv2D
input: (None, 16, 16, 512)
output: (None, 16, 16, 256)
concatenate: Concatenate
input: [(None, 16, 16, 256), (None, 16, 16, 256)]
output: (None, 16, 16, 512)
conv2d_11: Conv2D
input: (None, 16, 16, 512)
output: (None, 16, 16, 256)
conv2d_12: Conv2D
input: (None, 16, 16, 256)
output: (None, 16, 16, 256)
up_sampling2d_1: UpSampling2D
input: (None, 16, 16, 256)
output: (None, 32, 32, 256)
conv2d_13: Conv2D
input: (None, 32, 32, 256)
output: (None, 32, 32, 128)
concatenate_1: Concatenate
input: [(None, 32, 32, 128), (None, 32, 32, 128)]
output: (None, 32, 32, 256)
conv2d_14: Conv2D
input: (None, 32, 32, 256)
output: (None, 32, 32, 128)
conv2d_15: Conv2D
input: (None, 32, 32, 128)
output: (None, 32, 32, 128)
up_sampling2d_2: UpSampling2D
input: (None, 32, 32, 128)
output: (None, 64, 64, 128)
conv2d_16: Conv2D
input: (None, 64, 64, 128)
output: (None, 64, 64, 64)
concatenate_2: Concatenate
input: [(None, 64, 64, 64), (None, 64, 64, 64)]
output: (None, 64, 64, 128)
conv2d_17: Conv2D
input: (None, 64, 64, 128)
output: (None, 64, 64, 64)
conv2d_18: Conv2D
input: (None, 64, 64, 64)
output: (None, 64, 64, 64)
up_sampling2d_3: UpSampling2D
input: (None, 64, 64, 64)
output: (None, 128, 128, 64)
conv2d_19: Conv2D
input: (None, 128, 128, 64)
output: (None, 128, 128, 32)
concatenate_3: Concatenate
input: [(None, 128, 128, 32), (None, 128, 128, 32)]
output: (None, 128, 128, 64)



Load data

Loading all data into memory is not a good idea since the data are too big to fit in. So we will create dataGenerators - load data on the fly as explained here

In [12]:
```
# lists of directories with studies
train_and_val_directories = [f.path for f in os.scandir(TRAIN_DATASET_PAT

# file BraTS20_Training_355 has ill formatted name for for seg.nii file
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
train_ids, test_ids = train_test_split(train_test_ids,test_size=0.15)
```

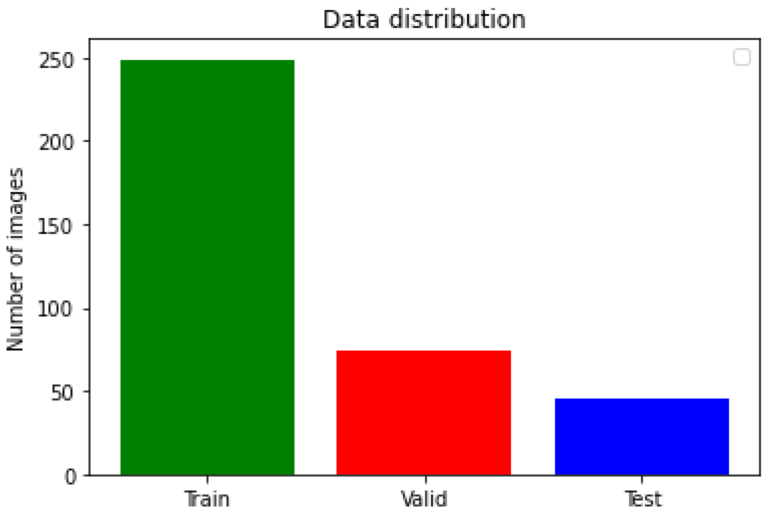
Override Keras sequence DataGenerator class

In [13]:
```
class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, dim=(IMG_SIZE,IMG_SIZE), batch_size = 1,
        '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'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch

        # Find list of IDs
        Batch_ids = [self.list_IDs[k] for k in indexes]

        # Generate data
        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 : (n_samples,
        # Initialization
        X = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, self.n_ch
        y = np.zeros((self.batch_size*VOLUME_SLICES, 240, 240))
        Y = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, 4))


        # Generate data
        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+VOL
                 X[j +VOLUME_SLICES*c,:,:,1] = cv2.resize(ce[:,:,j+VOLUME

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

        # Generate masks
        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)
```

Number of data used for training / testing / validation

In [14]:
```
# show number of data for each dir
def showDataLayout():
    plt.bar(["Train","Valid","Test"],
    [len(train_ids), len(val_ids), len(test_ids)], align='center',color=[
    plt.legend()

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

    plt.show()

showDataLayout()
```



Add callback for training process

In [15]:
```
csv_logger = CSVLogger('training.log', separator=',', append=False)


callbacks = [
#     keras.callbacks.EarlyStopping(monitor='loss', min_delta=0,
#                               patience=2, verbose=1, mode='auto'),
      keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=2, min_lr=0.000001, verbose=1),
#  keras.callbacks.ModelCheckpoint(filepath = 'model_.{epoch:02d}-{val_lo
#                             verbose=1, save_best_only=True, save_weight
        csv_logger
    ]
```

Train model

My best model was trained with 81% accuracy on mean IOU and 65.5% on Dice loss
I will load this pretrained model instead of training again

In [16]:
```
K.clear_session()
# I train by 35 epochs but for time i will use 15 epochs
history =  model.fit(training_generator,
                    epochs=15,
                     steps_per_epoch=len(train_ids),
                     callbacks= callbacks,
                     validation_data = valid_generator
                     )
model.save("3D_MRI_Brain_tumor_segmentation.h5")
```

```
Epoch 1/15
249/249 [==============================] - 226s 876ms/step - loss: 0.1987
- accuracy: 0.9626 - mean_io_u: 0.5461 - dice_coef: 0.2544 - precision: 0.
9519 - sensitivity: 0.9273 - specificity: 0.9946 - dice_coef_necrotic: 0.0
356 - dice_coef_edema: 0.1066 - dice_coef_enhancing: 0.0285 - val_loss: 0.
0874 - val_accuracy: 0.9817 - val_mean_io_u: 0.4345 - val_dice_coef: 0.271
1 - val_precision: 0.9815 - val_sensitivity: 0.9815 - val_specificity: 0.9
938 - val_dice_coef_necrotic: 0.0650 - val_dice_coef_edema: 0.1986 - val_d
ice_coef_enhancing: 0.0427
Epoch 2/15
249/249 [==============================] - 109s 436ms/step - loss: 0.0780
- accuracy: 0.9831 - mean_io_u: 0.5588 - dice_coef: 0.2743 - precision: 0.
9830 - sensitivity: 0.9826 - specificity: 0.9943 - dice_coef_necrotic: 0.0
635 - dice_coef_edema: 0.1525 - dice_coef_enhancing: 0.0474 - val_loss: 0.
0871 - val_accuracy: 0.9817 - val_mean_io_u: 0.4048 - val_dice_coef: 0.262
1 - val_precision: 0.9817 - val_sensitivity: 0.9817 - val_specificity: 0.9
939 - val_dice_coef_necrotic: 0.0280 - val_dice_coef_edema: 0.0877 - val_d
ice_coef_enhancing: 0.0200
Epoch 3/15
249/249 [==============================] - 108s 433ms/step - loss: 0.0829
- accuracy: 0.9838 - mean_io_u: 0.4547 - dice_coef: 0.2697 - precision: 0.
9836 - sensitivity: 0.9833 - specificity: 0.9946 - dice_coef_necrotic: 0.0
486 - dice_coef_edema: 0.1413 - dice_coef_enhancing: 0.0333 - val_loss: 0.
0856 - val_accuracy: 0.9817 - val_mean_io_u: 0.3756 - val_dice_coef: 0.266
0 - val_precision: 0.9816 - val_sensitivity: 0.9816 - val_specificity: 0.9
939 - val_dice_coef_necrotic: 0.0411 - val_dice_coef_edema: 0.1226 - val_d
ice_coef_enhancing: 0.0287
Epoch 4/15
249/249 [==============================] - 108s 433ms/step - loss: 0.0773
- accuracy: 0.9835 - mean_io_u: 0.5191 - dice_coef: 0.2705 - precision: 0.
9833 - sensitivity: 0.9833 - specificity: 0.9945 - dice_coef_necrotic: 0.0
470 - dice_coef_edema: 0.1366 - dice_coef_enhancing: 0.0303 - val_loss: 0.
0788 - val_accuracy: 0.9817 - val_mean_io_u: 0.3792 - val_dice_coef: 0.280
0 - val_precision: 0.9815 - val_sensitivity: 0.9815 - val_specificity: 0.9
939 - val_dice_coef_necrotic: 0.0745 - val_dice_coef_edema: 0.1769 - val_d
ice_coef_enhancing: 0.0612
Epoch 5/15
249/249 [==============================] - 108s 433ms/step - loss: 0.0682
- accuracy: 0.9835 - mean_io_u: 0.4179 - dice_coef: 0.2820 - precision: 0.
9833 - sensitivity: 0.9834 - specificity: 0.9945 - dice_coef_necrotic: 0.0
699 - dice_coef_edema: 0.1765 - dice_coef_enhancing: 0.0514 - val_loss: 0.
0753 - val_accuracy: 0.9817 - val_mean_io_u: 0.3892 - val_dice_coef: 0.275
6 - val_precision: 0.9816 - val_sensitivity: 0.9816 - val_specificity: 0.9
939 - val_dice_coef_necrotic: 0.0575 - val_dice_coef_edema: 0.1308 - val_d
ice_coef_enhancing: 0.0537
Epoch 6/15
249/249 [==============================] - 108s 433ms/step - loss: 0.0619
- accuracy: 0.9846 - mean_io_u: 0.4051 - dice_coef: 0.2831 - precision: 0.
9844 - sensitivity: 0.9844 - specificity: 0.9948 - dice_coef_necrotic: 0.0
665 - dice_coef_edema: 0.1677 - dice_coef_enhancing: 0.0696 - val_loss: 0.
0739 - val_accuracy: 0.9817 - val_mean_io_u: 0.6914 - val_dice_coef: 0.274
5 - val_precision: 0.9816 - val_sensitivity: 0.9817 - val_specificity: 0.9
939 - val_dice_coef_necrotic: 0.0579 - val_dice_coef_edema: 0.1130 - val_d
ice_coef_enhancing: 0.0497
Epoch 7/15
249/249 [==============================] - 108s 434ms/step - loss: 0.0626
- accuracy: 0.9847 - mean_io_u: 0.5119 - dice_coef: 0.2816 - precision: 0.
9846 - sensitivity: 0.9842 - specificity: 0.9949 - dice_coef_necrotic: 0.0
645 - dice_coef_edema: 0.1547 - dice_coef_enhancing: 0.0771 - val_loss: 0.
0809 - val_accuracy: 0.9817 - val_mean_io_u: 0.3756 - val_dice_coef: 0.273
```

```
9 - val_precision: 0.9816 - val_sensitivity: 0.9816 - val_specificity: 0.9
939 - val_dice_coef_necrotic: 0.0567 - val_dice_coef_edema: 0.1366 - val_d
ice_coef_enhancing: 0.0167
Epoch 8/15
249/249 [==============================] - 108s 433ms/step - loss: 0.0686
- accuracy: 0.9836 - mean_io_u: 0.4553 - dice_coef: 0.2791 - precision: 0.
9834 - sensitivity: 0.9834 - specificity: 0.9945 - dice_coef_necrotic: 0.0
664 - dice_coef_edema: 0.1506 - dice_coef_enhancing: 0.0644 - val_loss: 0.
0734 - val_accuracy: 0.9817 - val_mean_io_u: 0.8061 - val_dice_coef: 0.282
3 - val_precision: 0.9816 - val_sensitivity: 0.9816 - val_specificity: 0.9
939 - val_dice_coef_necrotic: 0.0577 - val_dice_coef_edema: 0.1434 - val_d
ice_coef_enhancing: 0.1087
Epoch 9/15
249/249 [==============================] - 108s 434ms/step - loss: 0.0602
- accuracy: 0.9845 - mean_io_u: 0.6425 - dice_coef: 0.2902 - precision: 0.
9845 - sensitivity: 0.9843 - specificity: 0.9948 - dice_coef_necrotic: 0.0
673 - dice_coef_edema: 0.1650 - dice_coef_enhancing: 0.1230 - val_loss: 0.
0776 - val_accuracy: 0.9817 - val_mean_io_u: 0.3756 - val_dice_coef: 0.275
0 - val_precision: 0.9817 - val_sensitivity: 0.9817 - val_specificity: 0.9
939 - val_dice_coef_necrotic: 0.0474 - val_dice_coef_edema: 0.1153 - val_d
ice_coef_enhancing: 0.0604
Epoch 10/15
249/249 [==============================] - 108s 433ms/step - loss: 0.0609
- accuracy: 0.9841 - mean_io_u: 0.4086 - dice_coef: 0.2984 - precision: 0.
9841 - sensitivity: 0.9839 - specificity: 0.9947 - dice_coef_necrotic: 0.0
693 - dice_coef_edema: 0.1742 - dice_coef_enhancing: 0.1708 - val_loss: 0.
0754 - val_accuracy: 0.9820 - val_mean_io_u: 0.3771 - val_dice_coef: 0.295
2 - val_precision: 0.9820 - val_sensitivity: 0.9814 - val_specificity: 0.9
940 - val_dice_coef_necrotic: 0.1086 - val_dice_coef_edema: 0.2227 - val_d
ice_coef_enhancing: 0.2103

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.000200000009499
49026.
Epoch 11/15
249/249 [==============================] - 108s 435ms/step - loss: 0.0601
- accuracy: 0.9839 - mean_io_u: 0.3866 - dice_coef: 0.3099 - precision: 0.
9842 - sensitivity: 0.9835 - specificity: 0.9947 - dice_coef_necrotic: 0.1
016 - dice_coef_edema: 0.2096 - dice_coef_enhancing: 0.2139 - val_loss: 0.
0611 - val_accuracy: 0.9830 - val_mean_io_u: 0.4008 - val_dice_coef: 0.336
6 - val_precision: 0.9852 - val_sensitivity: 0.9814 - val_specificity: 0.9
951 - val_dice_coef_necrotic: 0.1261 - val_dice_coef_edema: 0.2313 - val_d
ice_coef_enhancing: 0.3190
Epoch 12/15
249/249 [==============================] - 109s 436ms/step - loss: 0.0507
- accuracy: 0.9864 - mean_io_u: 0.3958 - dice_coef: 0.3460 - precision: 0.
9878 - sensitivity: 0.9848 - specificity: 0.9959 - dice_coef_necrotic: 0.1
290 - dice_coef_edema: 0.2901 - dice_coef_enhancing: 0.3048 - val_loss: 0.
0598 - val_accuracy: 0.9831 - val_mean_io_u: 0.3958 - val_dice_coef: 0.349
7 - val_precision: 0.9859 - val_sensitivity: 0.9809 - val_specificity: 0.9
953 - val_dice_coef_necrotic: 0.1613 - val_dice_coef_edema: 0.2784 - val_d
ice_coef_enhancing: 0.3477
Epoch 13/15
249/249 [==============================] - 108s 433ms/step - loss: 0.0498
- accuracy: 0.9863 - mean_io_u: 0.3953 - dice_coef: 0.3688 - precision: 0.
9882 - sensitivity: 0.9843 - specificity: 0.9960 - dice_coef_necrotic: 0.1
698 - dice_coef_edema: 0.3336 - dice_coef_enhancing: 0.3691 - val_loss: 0.
0678 - val_accuracy: 0.9778 - val_mean_io_u: 0.3929 - val_dice_coef: 0.353
4 - val_precision: 0.9847 - val_sensitivity: 0.9726 - val_specificity: 0.9
949 - val_dice_coef_necrotic: 0.2272 - val_dice_coef_edema: 0.2670 - val_d
ice_coef_enhancing: 0.3658
```

```
Epoch 14/15
249/249 [==============================] - 108s 433ms/step - loss: 0.0505
- accuracy: 0.9862 - mean_io_u: 0.4057 - dice_coef: 0.3538 - precision: 0.
9887 - sensitivity: 0.9836 - specificity: 0.9962 - dice_coef_necrotic: 0.1
806 - dice_coef_edema: 0.2785 - dice_coef_enhancing: 0.3280 - val_loss: 0.
0593 - val_accuracy: 0.9812 - val_mean_io_u: 0.4538 - val_dice_coef: 0.390
1 - val_precision: 0.9870 - val_sensitivity: 0.9773 - val_specificity: 0.9
956 - val_dice_coef_necrotic: 0.2542 - val_dice_coef_edema: 0.3558 - val_d
ice_coef_enhancing: 0.4339
Epoch 15/15
249/249 [==============================] - 108s 432ms/step - loss: 0.0466
- accuracy: 0.9868 - mean_io_u: 0.4614 - dice_coef: 0.3999 - precision: 0.
9893 - sensitivity: 0.9842 - specificity: 0.9964 - dice_coef_necrotic: 0.2
372 - dice_coef_edema: 0.3898 - dice_coef_enhancing: 0.4180 - val_loss: 0.
0545 - val_accuracy: 0.9844 - val_mean_io_u: 0.4979 - val_dice_coef: 0.388
0 - val_precision: 0.9866 - val_sensitivity: 0.9825 - val_specificity: 0.9
955 - val_dice_coef_necrotic: 0.1900 - val_dice_coef_edema: 0.3700 - val_d
ice_coef_enhancing: 0.3962
```

Visualize the training process

In [17]:

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

# Load the saved model
model = tf.keras.models.load_model("3D_MRI_Brain_tumor_segmentation.h5",
    '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):
    # Load the NIfTI file
    img = nib.load(image_file).get_fdata()

    # Select a specific slice if needed
    if slice_index is not None:
        img = img[:, :, slice_index]

    # Resize the image to (IMG_SIZE, IMG_SIZE)
    img_resized = cv2.resize(img, (IMG_SIZE, IMG_SIZE))

    # Normalize the image
    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))

    # Process specific slices from the images
    X[0, :, :, 0] = preprocess_image(image_paths[0], slice_index)
    X[0, :, :, 1] = preprocess_image(image_paths[1], slice_index)
```

```
    # Make prediction
    pred = model.predict(X)

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

# Example usage
image_paths = [
    '/kaggle/input/brats20-dataset-training-validation/BraTS2020_Validati
    '/kaggle/input/brats20-dataset-training-validation/BraTS2020_Validati
]

slice_index = 75  # Example slice index
prediction = predict(image_paths, slice_index)
```

In [18]:

```
import numpy as np

def get_classification(pred):
    # Get the class with the highest probability for each pixel
    class_predictions = np.argmax(pred, axis=-1)
    return class_predictions
```

In [19]:

```
get_classification(prediction)
```

Out[19]:

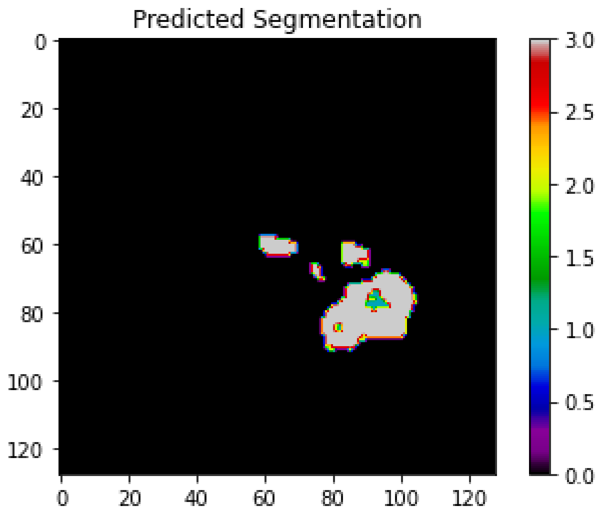
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,
        0,  0,  0,  0,  0,  0,  0, 59, 59, 59, 59, 60, 61, 83, 83, 74, 7
4,
       74, 75, 76, 89, 85, 84, 84, 84, 83, 82, 80, 79, 78, 78, 77, 77, 7
7,
       77, 77, 78, 78, 78, 80, 79,  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,  0])
```

In [20]:

```
import matplotlib.pyplot as plt

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

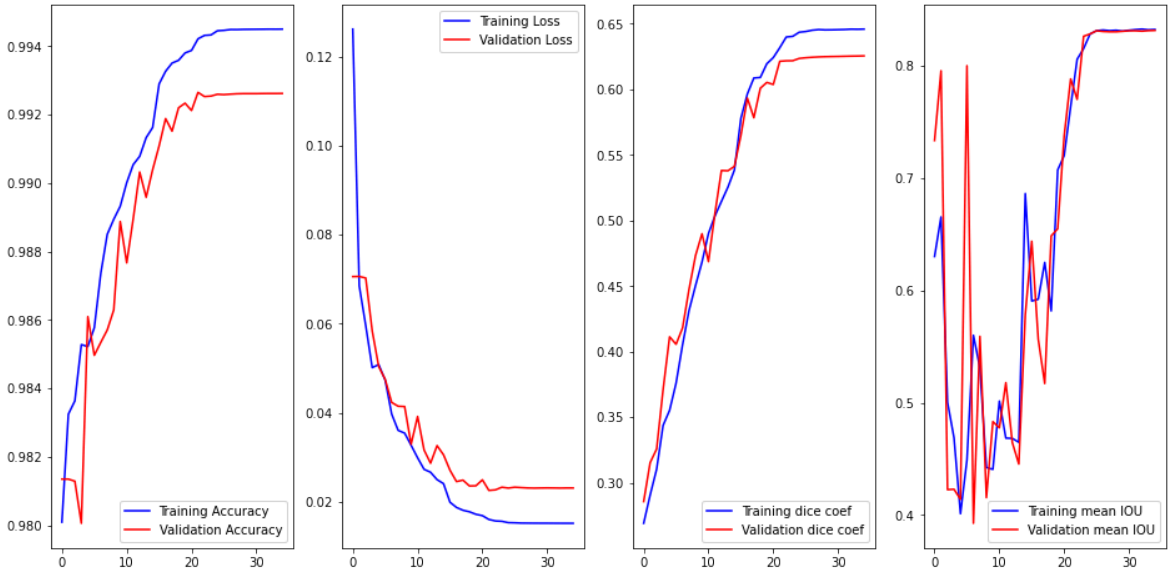
In [21]:

```
visualize_prediction(prediction)
```



In [22]:
```
############ load trained model ################
model = keras.models.load_model('../input/modelperclasseval/model_per_cla
                                   custom_objects={ 'accuracy' : tf.keras
                                                   "dice_coef": dice_coef
                                                   "precision": precision
                                                   "sensitivity":sensitiv
                                                   "specificity":specific
                                                   "dice_coef_necrotic":
                                                   "dice_coef_edema": dic
                                                   "dice_coef_enhancing":
                                                  }, compile=False)

history = pd.read_csv('../input/modelperclasseval/training_per_class.log'

hist=history

############### ########## ####### #######

# hist=history.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()
```



Prediction examples

In [23]:

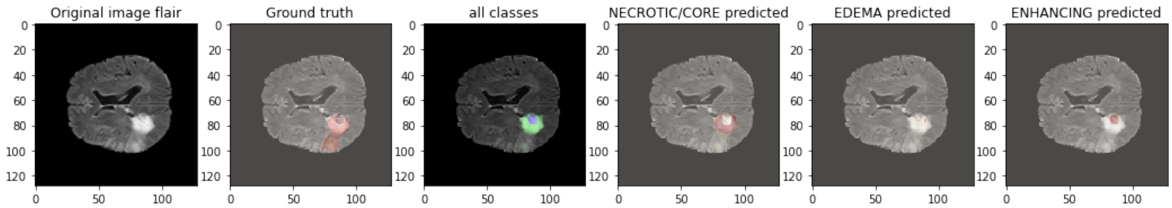
```
# mri type must one of 1) flair 2) t1 3) t1ce 4) t2 ------- or even 5) se
# returns volume of specified study at `path`
def imageLoader(path):
    image = nib.load(path).get_fdata()
    X = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, self.n_channe
    for j in range(VOLUME_SLICES):
        X[j +VOLUME_SLICES*c,:,:,0] = cv2.resize(image[:,:,j+VOLUME_START
        X[j +VOLUME_SLICES*c,:,:,1] = cv2.resize(ce[:,:,j+VOLUME_START_AT

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


# load nifti file at `path`
# and load each slice with mask from volume
# choose the mri type & resize to `IMG_SIZE`
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 each slice in 3D volume, find also it's mask
        for j in range(0, currentScanVolume.shape[2]):
            scan_img = cv2.resize(currentScanVolume[:,:,j], dsize=(IMG_SI
            mask_img = cv2.resize(currentMaskVolume[:,:,j], dsize=(IMG_SI
            scans.append(scan_img[..., np.newaxis])
            masks.append(mask_img[..., np.newaxis])
    return np.array(scans, dtype='float32'), np.array(masks, dtype='float
```

```
#brains_list_test, masks_list_test = loadDataFromDir(VALIDATION_DATASET_P
```

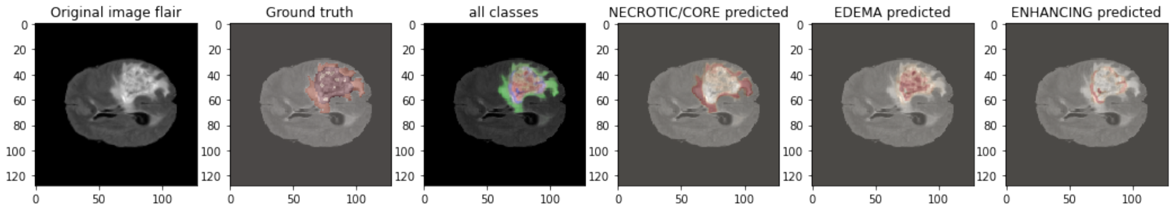
In [24]:
```
def predictByPath(case_path,case):
    files = next(os.walk(case_path))[2]
    X = np.empty((VOLUME_SLICES, IMG_SIZE, IMG_SIZE, 2))
  #  y = np.empty((VOLUME_SLICES, IMG_SIZE, IMG_SIZE))

    vol_path = os.path.join(case_path, f'BraTS20_Training_{case}_flair.ni
    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()

 #   vol_path = os.path.join(case_path, f'BraTS20_Training_{case}_seg.nii
 #   seg=nib.load(vol_path).get_fdata()


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

  #  model.evaluate(x=X,y=y[:,:,:,0], callbacks= callbacks)
    return model.predict(X/np.max(X), verbose=1)


def showPredictsById(case, start_slice = 60):
    path = f"../input/brats20-dataset-training-validation/BraTS2020_Train
    gt = nib.load(os.path.join(path, f'BraTS20_Training_{case}_seg.nii'))
    origImage = nib.load(os.path.join(path, f'BraTS20_Training_{case}_fla
    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): # for each image, add brain background
        axarr[i].imshow(cv2.resize(origImage[:,:,start_slice+VOLUME_START

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


# mask = np.zeros((10,10))
# mask[3:-3, 3:-3] = 1 # white square in black background
# im = mask + np.random.randn(10,10) * 0.01 # random image
# masked = np.ma.masked_where(mask == 0, mask)

# plt.figure()
# plt.subplot(1,2,1)
# plt.imshow(im, 'gray', interpolation='none')
# plt.subplot(1,2,2)
# plt.imshow(im, 'gray', interpolation='none')
# plt.imshow(masked, 'jet', interpolation='none', alpha=0.7)
# plt.show()
```

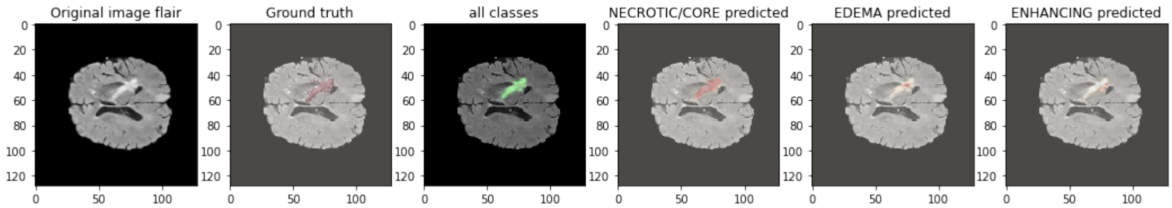
```
4/4 [==============================] - 1s 96ms/step
<Figure size 1296x3600 with 0 Axes>
```



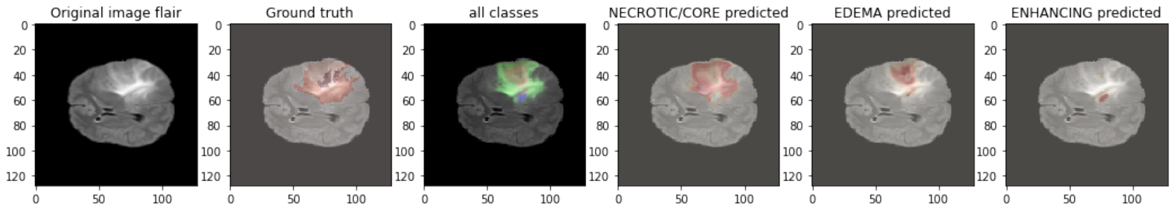

```
4/4 [==============================] - 0s 33ms/step
<Figure size 1296x3600 with 0 Axes>
```



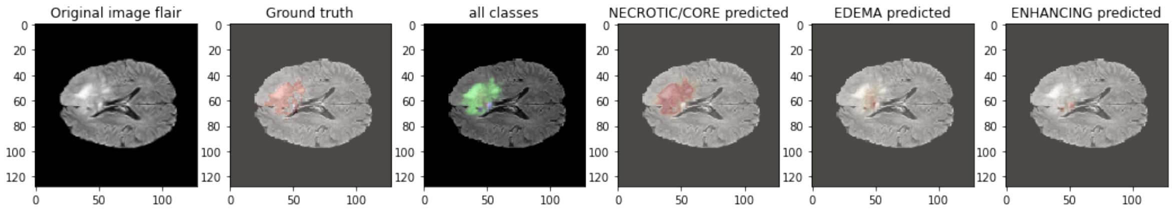

```
4/4 [==============================] - 0s 33ms/step
<Figure size 1296x3600 with 0 Axes>
```



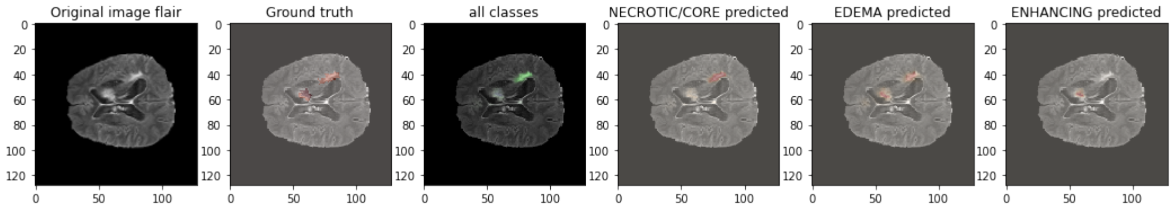

```
4/4 [==============================] - 0s 33ms/step
<Figure size 1296x3600 with 0 Axes>
```



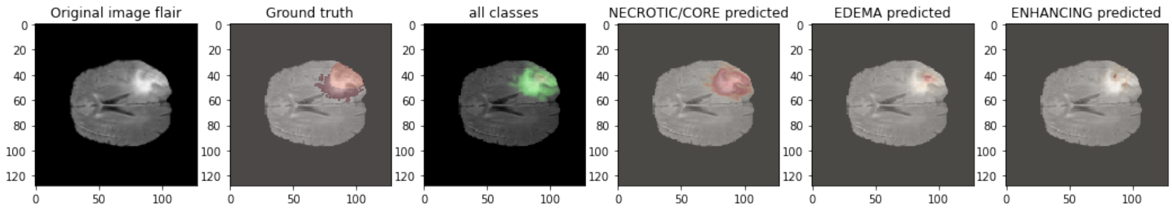

```
4/4 [==============================] - 0s 33ms/step
<Figure size 1296x3600 with 0 Axes>
```




```
4/4 [==============================] - 0s 33ms/step
<Figure size 1296x3600 with 0 Axes>
```




```
4/4 [==============================] - 0s 34ms/step
<Figure size 1296x3600 with 0 Axes>
```

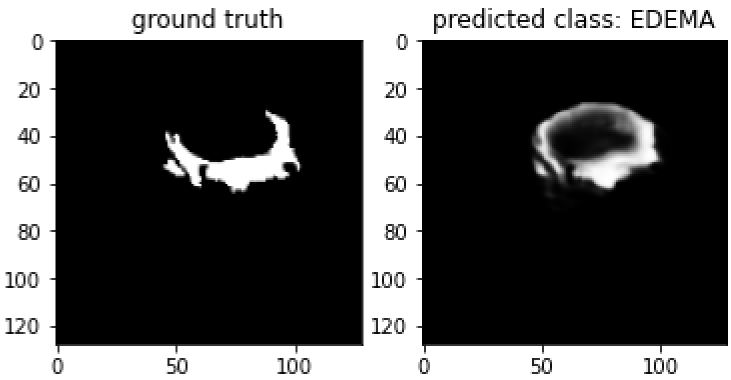


Evaluation

In [25]:

```
case = case=test_ids[3][-3:]
path = f"../input/brats20-dataset-training-validation/BraTS2020_TrainingD
gt = nib.load(os.path.join(path, f'BraTS20_Training_{case}_seg.nii')).get
p = predictByPath(path,case)


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


i=40 # slice at
eval_class = 2 #     0 : 'NOT tumor',  1 : 'ENHANCING',    2 : 'CORE',



gt[gt != eval_class] = 1 # use only one class for per class evaluation

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 33ms/step
<Figure size 432x288 with 0 Axes>
```



In [26]:
```
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(test_generator, batch_size=100, callbacks= callb
print("test loss, test acc:", results)
```

```
Evaluate on test data
45/45 [==============================] - 28s 602ms/step - loss: 0.0208 - a
ccuracy: 0.9928 - mean_io_u_1: 0.8334 - dice_coef: 0.6126 - precision: 0.9
931 - sensitivity: 0.9914 - specificity: 0.9977 - dice_coef_necrotic: 0.59
36 - dice_coef_edema: 0.7004 - dice_coef_enhancing: 0.6472
test loss, test acc: [0.020345302298665047, 0.9929292798042297, 0.83286988
73519897, 0.6156396865844727, 0.9933792948722839, 0.9915380477905273, 0.99
77688193321228, 0.6048883199691772, 0.7391760349273682, 0.625171601772308
3]
```

In [ ]: