brain-tumor-seg-unet-depplabv3

[52]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

```
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/name_mapping_validation_data.csv
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/survival_evaluation.csv
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_084/BraTS20_Validation_084_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_084/BraTS20_Validation_084_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_084/BraTS20_Validation_084_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_084/BraTS20_Validation_084_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
```

```
I_BraTS2020_ValidationData/BraTS20_Validation_118/BraTS20_Validation_118_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_118/BraTS20_Validation_118_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_118/BraTS20_Validation_118_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_118/BraTS20_Validation_118_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_111/BraTS20_Validation_111_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_111/BraTS20_Validation_111_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_111/BraTS20_Validation_111_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_111/BraTS20_Validation_111_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_060/BraTS20_Validation_060_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_060/BraTS20_Validation_060_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_060/BraTS20_Validation_060_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_060/BraTS20_Validation_060_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_034/BraTS20_Validation_034_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_034/BraTS20_Validation_034_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_034/BraTS20_Validation_034_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_034/BraTS20_Validation_034_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_045/BraTS20_Validation_045_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_045/BraTS20_Validation_045_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_045/BraTS20_Validation_045_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
```

```
I_BraTS2020_ValidationData/BraTS20_Validation_045/BraTS20_Validation_045_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_027/BraTS20_Validation_027_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_027/BraTS20_Validation_027_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_027/BraTS20_Validation_027_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_027/BraTS20_Validation_027_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_073/BraTS20_Validation_073_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_073/BraTS20_Validation_073_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_073/BraTS20_Validation_073_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_073/BraTS20_Validation_073_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_014/BraTS20_Validation_014_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_014/BraTS20_Validation_014_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_014/BraTS20_Validation_014_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_014/BraTS20_Validation_014_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_047/BraTS20_Validation_047_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_047/BraTS20_Validation_047_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_047/BraTS20_Validation_047_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_047/BraTS20_Validation_047_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_096/BraTS20_Validation_096_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_096/BraTS20_Validation_096_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
```

```
I_BraTS2020_ValidationData/BraTS20_Validation_096/BraTS20_Validation_096_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_096/BraTS20_Validation_096_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_068/BraTS20_Validation_068_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_068/BraTS20_Validation_068_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_068/BraTS20_Validation_068_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_068/BraTS20_Validation_068_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_090/BraTS20_Validation_090_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_090/BraTS20_Validation_090_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_090/BraTS20_Validation_090_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_090/BraTS20_Validation_090_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_038/BraTS20_Validation_038_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_038/BraTS20_Validation_038_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_038/BraTS20_Validation_038_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_038/BraTS20_Validation_038_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_017/BraTS20_Validation_017_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_017/BraTS20_Validation_017_flair.n
ii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_017/BraTS20_Validation_017_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_017/BraTS20_Validation_017_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_069/BraTS20_Validation_069_t1ce.ni
i
/kaggle/input/brats20-dataset-training-validation/BraTS2020_ValidationData/MICCA
I_BraTS2020_ValidationData/BraTS20_Validation_069/BraTS20_Validation_069_t2.nii
```

```
BraTS2020_TrainingData/BraTS20_Training_178/BraTS20_Training_178_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_178/BraTS20_Training_178_t1ce.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_178/BraTS20_Training_178_seg.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_178/BraTS20_Training_178_flair.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_355/BraTS20_Training_355_flair.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_355/W39_1998.09.19_Segm.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_355/BraTS20_Training_355_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_355/BraTS20_Training_355_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_355/BraTS20_Training_355_t1ce.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_285/BraTS20_Training_285_seg.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_285/BraTS20_Training_285_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_285/BraTS20_Training_285_t1ce.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_285/BraTS20_Training_285_flair.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_285/BraTS20_Training_285_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_297/BraTS20_Training_297_t2.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_297/BraTS20_Training_297_t1.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_297/BraTS20_Training_297_seg.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_297/BraTS20_Training_297_t1ce.nii
/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_
BraTS2020_TrainingData/BraTS20_Training_297/BraTS20_Training_297_flair.nii
/kaggle/input/images/Metastasis head scan MRI tumor.JPG
/kaggle/input/images/mri image with.jpg
```

[53]:
```
pip install nibabel
```

```
Requirement already satisfied: nibabel in /opt/conda/lib/python3.10/site-
packages (5.2.1)
Requirement already satisfied: numpy>=1.20 in /opt/conda/lib/python3.10/site-
packages (from nibabel) (1.26.4)
Requirement already satisfied: packaging>=17 in /opt/conda/lib/python3.10/site-
packages (from nibabel) (21.3)
```

```
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.10/site-packages (from packaging>=17->nibabel) (3.1.1)
Note: you may need to restart the kernel to use updated packages.
```

[54]:
```
import os
import numpy as np
import nibabel as nib
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow.keras.backend as K
import tensorflow as tf
from tensorflow.keras.callbacks import (EarlyStopping, ReduceLROnPlateau,␣
 ↪ModelCheckpoint)
from tensorflow.keras.layers import *
```

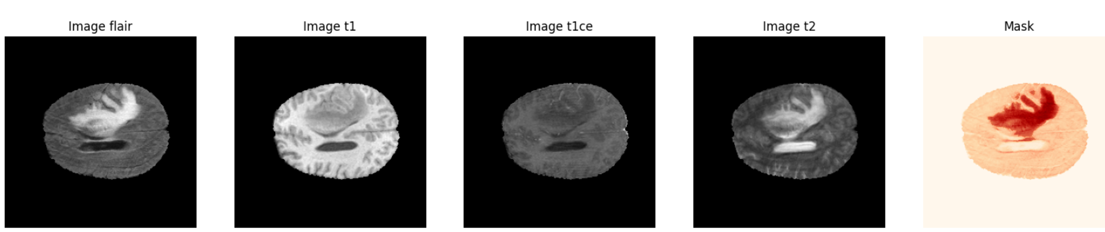
[55]:
```
TRAIN_DATASET_PATH = '../input/brats20-dataset-training-validation/
 ↪BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/'
VALIDATION_DATASET_PATH = '../input/brats20-dataset-training-validation/
 ↪BraTS2020_ValidationData/MICCAI_BraTS2020_ValidationData/'
# DEFINE seg-areas
SEGMENT_CLASSES = {
    0 : 'NOT tumor',
    1 : 'NECROTIC/CORE', # or NON-ENHANCING tumor CORE
    2 : 'EDEMA',
    3 : 'ENHANCING' # original 4 -> converted into 3 later
}
IMG_SIZE=128
# there are 155 slices per volume
# to start at 5 and use 145 slices means we will skip the first 5 and last 5
VOLUME_SLICES = 100
VOLUME_START_AT = 22 # first slice of volume that we will include
```

[56]:
```
def show_imgs(paths, i):
    sub_path = sorted(os.listdir(TRAIN_DATASET_PATH + paths[i]))

    image_flair = nib.load(TRAIN_DATASET_PATH + paths[i]+ '/' +sub_path[0]).
 ↪get_fdata()
    mask = nib.load(TRAIN_DATASET_PATH + paths[i]+ '/' + sub_path[1]).
 ↪get_fdata()
    image_t1 = nib.load(TRAIN_DATASET_PATH + paths[i]+ '/' + sub_path[2]).
 ↪get_fdata()
    image_t1ce = nib.load(TRAIN_DATASET_PATH + paths[i]+ '/' + sub_path[3]).
 ↪get_fdata()
    image_t2 = nib.load(TRAIN_DATASET_PATH + paths[i]+ '/' + sub_path[4]).
 ↪get_fdata()
```

```python
    print(f"Height of the image: {image_flair.shape[0]}")
    print(f"width of the image: {image_flair.shape[1]}")
    print(f"number of slices of volume of the image: {image_flair.shape[-1]}")
    print()

    fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1,5, figsize = (20, 10))
    slice_w = 25
    ax1.imshow(image_flair[:,:,image_flair.shape[0]//2-slice_w], cmap = 'gray')
    ax1.set_title('Image flair')
    ax1.axis(False)
    ax2.imshow(image_t1[:,:,image_t1.shape[0]//2-slice_w], cmap = 'gray')
    ax2.set_title('Image t1')
    ax2.axis(False)
    ax3.imshow(image_t1ce[:,:,image_t1ce.shape[0]//2-slice_w], cmap = 'gray')
    ax3.set_title('Image t1ce')
    ax3.axis(False)
    ax4.imshow(image_t2[:,:,image_t2.shape[0]//2-slice_w], cmap = 'gray')
    ax4.set_title('Image t2')
    ax4.axis(False)
    ax5.imshow(image_flair[:,:,image_flair.shape[0]//2-slice_w], cmap="OrRd", alpha=1.0)
    ax5.imshow(mask[:,:,mask.shape[0]//2-slice_w], alpha=0.2, cmap="OrRd")
    ax5.set_title('Mask')
    ax5.axis(False)
    print()
#     plt.imshow(mask, cmap='reds')
    plt.show()
```

[57]:
```python
path = sorted(os.listdir('/kaggle/input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData'))
show_imgs(path, 0) # with Tumor
```

```
Height of the image: 240
width of the image: 240
number of slices of volume of the image: 155
```



[58]:
```
train_and_val_directories = [f.path for f in os.scandir(TRAIN_DATASET_PATH) if␣
↪f.is_dir()]

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

[59]:
```
tf.keras.applications.MobileNetV2(input_shape=(128,128,3), include_top=False,␣
↪weights='imagenet').summary
```

[59]:
```
<bound method Model.summary of <Functional name=mobilenetv2_1.00_128,
built=True>>
```

[60]:
```
def generate_data(list_ids, batch_size, img_size):
    while True:
        'Generates data containing batch_size samples' # X : (n_samples, *dim,␣
↪n_channels)
        indexes = np.arange(len(list_ids))
    #     indexes = indexes[index*batch_size:(index+1)*batch_size]
        np.random.shuffle(indexes)
        Batch_ids = [list_ids[k] for k in indexes[:batch_size]]

        # Initialization
        X = np.zeros((batch_size*VOLUME_SLICES, img_size[0], img_size[1],␣
↪img_size[2]))
        y = np.zeros((batch_size*VOLUME_SLICES, 240, 240))
        Y = np.zeros((batch_size*VOLUME_SLICES, img_size[0], img_size[1], 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+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]

        # Generate masks
        y[y==4] = 3
        mask = tf.one_hot(y, len(SEGMENT_CLASSES))
        Y = tf.image.resize(mask, (IMG_SIZE, IMG_SIZE))
        yield X/np.max(X), Y


def data_generator_wrapper(list_ids, batch_size=1, img_size=(IMG_SIZE,␣
↪IMG_SIZE, 2)):
    if len(list_ids)==0 or batch_size<=0: return None
    return generate_data(list_ids, batch_size, img_size)
```

[61]:
```
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())
```

[62]:

```
import tensorflow.keras.backend as K

K.clear_session()
```

[63]:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Concatenate,␣
↪Conv2DTranspose, UpSampling2D

def DeepLabV3Plus_UNet(input_shape, num_classes):
    def UNet(input_tensor):
        # Downsample path
        conv1 = Conv2D(32, (3, 3), activation='relu',␣
↪padding='same')(input_tensor)
        pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

        conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)
        pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
```

```
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool3)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    # Bridge
    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool4)
    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv5)

    # Upsample path
    up6 = Conv2DTranspose(256, (2, 2), strides=(2, 2),␣
↪padding='same')(conv5)
    concat6 = Concatenate()([up6, conv4])
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(concat6)

    up7 = Conv2DTranspose(128, (2, 2), strides=(2, 2),␣
↪padding='same')(conv6)
    concat7 = Concatenate()([up7, conv3])
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(concat7)

    up8 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(conv7)
    concat8 = Concatenate()([up8, conv2])
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(concat8)

    up9 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(conv8)
    concat9 = Concatenate()([up9, conv1])
    conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(concat9)

    return conv9

def DeepLabV3Plus(input_tensor):
    # DeepLabv3+ implementation (without ASPP module)
    # Modify this part to include ASPP module
    x = Conv2D(filters=3, kernel_size=3, padding='same',␣
↪activation='relu')(input_tensor)
    # Backbone (e.g., ResNet, MobileNetV2)
    backbone = tf.keras.applications.MobileNetV2(input_shape=(128,128,3),␣
↪include_top=False, weights='imagenet')
    backbone_output = backbone(x)

    # Upsampling
    upsample1 = UpSampling2D((4, 4))(backbone_output)

    # U-Net path
    unet_output = UNet(upsample1)
```

```
        # upsampling and Convolutional layers
        upsample2 = UpSampling2D((8, 8))(unet_output)
        x = Conv2D(64, (3, 3), padding='same', activation='relu')(upsample2)
        x = Conv2D(32, (3, 3), padding='same', activation='relu')(x)

        # Final prediction
        output = Conv2D(num_classes, (1, 1), activation='softmax')(x)

        return output

    # Input tensor
    input_tensor = Input(shape=input_shape)

    # DeepLabv3+ with U-Net
    deep_lab_unet_output = DeepLabV3Plus(input_tensor)

    # Create the model
    model = tf.keras.Model(inputs=input_tensor, outputs=deep_lab_unet_output)

    return model

# Input tensor
input_shape = (128, 128, 3)
num_classes = 3

# Create model
model = DeepLabV3Plus_UNet(input_shape, num_classes)
```

[64]:
```
input_shape = (IMG_SIZE,IMG_SIZE, 2)
num_classes = len(SEGMENT_CLASSES)
model_2 = DeepLabV3Plus_UNet(input_shape, num_classes)
model_2.summary()
```

Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 128, 128, 2)	0	-
conv2d_14 (Conv2D)	(None, 128, 128, 3)	57	input_layer_2[0]…
mobilenetv2_1.00_1… (Functional)	(None, 4, 4, 1280)	2,257,984	conv2d_14[0][0]

up_sampling2d_2 (UpSampling2D)	(None, 16, 16, 1280)	0	mobilenetv2_1.00…
conv2d_15 (Conv2D)	(None, 16, 16, 32)	368,672	up_sampling2d_2[…
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 32)	0	conv2d_15[0][0]
conv2d_16 (Conv2D)	(None, 8, 8, 64)	18,496	max_pooling2d_4[…
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 64)	0	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 4, 4, 128)	73,856	max_pooling2d_5[…
max_pooling2d_6 (MaxPooling2D)	(None, 2, 2, 128)	0	conv2d_17[0][0]
conv2d_18 (Conv2D)	(None, 2, 2, 256)	295,168	max_pooling2d_6[…
max_pooling2d_7 (MaxPooling2D)	(None, 1, 1, 256)	0	conv2d_18[0][0]
conv2d_19 (Conv2D)	(None, 1, 1, 512)	1,180,160	max_pooling2d_7[…
conv2d_20 (Conv2D)	(None, 1, 1, 512)	2,359,808	conv2d_19[0][0]
conv2d_transpose_4 (Conv2DTranspose)	(None, 2, 2, 256)	524,544	conv2d_20[0][0]
concatenate_4 (Concatenate)	(None, 2, 2, 512)	0	conv2d_transpose… conv2d_18[0][0]
conv2d_21 (Conv2D)	(None, 2, 2, 256)	1,179,904	concatenate_4[0]…
conv2d_transpose_5 (Conv2DTranspose)	(None, 4, 4, 128)	131,200	conv2d_21[0][0]
concatenate_5 (Concatenate)	(None, 4, 4, 256)	0	conv2d_transpose… conv2d_17[0][0]
conv2d_22 (Conv2D)	(None, 4, 4, 128)	295,040	concatenate_5[0]…
conv2d_transpose_6 (Conv2DTranspose)	(None, 8, 8, 64)	32,832	conv2d_22[0][0]
concatenate_6	(None, 8, 8, 128)	0	conv2d_transpose…

(Concatenate)			conv2d_16[0][0]
conv2d_23 (Conv2D)	(None, 8, 8, 64)	73,792	concatenate_6[0]…
conv2d_transpose_7 (Conv2DTranspose)	(None, 16, 16, 32)	8,224	conv2d_23[0][0]
concatenate_7 (Concatenate)	(None, 16, 16, 64)	0	conv2d_transpose… conv2d_15[0][0]
conv2d_24 (Conv2D)	(None, 16, 16, 32)	18,464	concatenate_7[0]…
up_sampling2d_3 (UpSampling2D)	(None, 128, 128, 32)	0	conv2d_24[0][0]
conv2d_25 (Conv2D)	(None, 128, 128, 64)	18,496	up_sampling2d_3[…
conv2d_26 (Conv2D)	(None, 128, 128, 32)	18,464	conv2d_25[0][0]
conv2d_27 (Conv2D)	(None, 128, 128, 4)	132	conv2d_26[0][0]

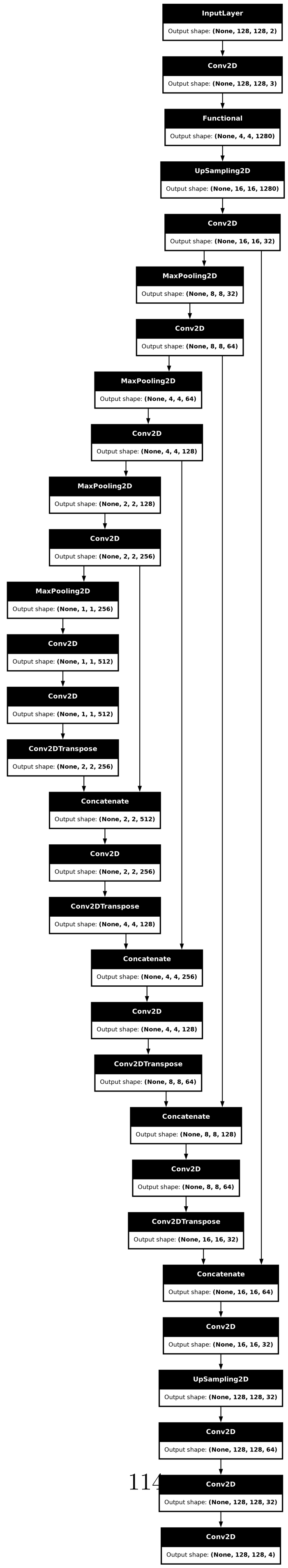
Total params: 8,855,293 (33.78 MB)

Trainable params: 8,821,181 (33.65 MB)

Non-trainable params: 34,112 (133.25 KB)

[65]:
```
from tensorflow.keras.utils import plot_model

# Plot the model structure
tf.keras.utils.plot_model(model_2, to_file='deeplabv3plus_model.png',␣
 ↪show_shapes=True)
```

[65]:


InputLayer
Output shape: (None, 128, 128, 2)
Conv2D
Output shape: (None, 128, 128, 3)
Functional
Output shape: (None, 4, 4, 1280)
UpSampling2D
Output shape: (None, 16, 16, 1280)
Conv2D
Output shape: (None, 16, 16, 32)
MaxPooling2D
Output shape: (None, 8, 8, 32)
Conv2D
Output shape: (None, 8, 8, 64)
MaxPooling2D
Output shape: (None, 4, 4, 64)
Conv2D
Output shape: (None, 4, 4, 128)
MaxPooling2D
Output shape: (None, 2, 2, 128)
Conv2D
Output shape: (None, 2, 2, 256)
MaxPooling2D
Output shape: (None, 1, 1, 256)
Conv2D
Output shape: (None, 1, 1, 512)
Conv2D
Output shape: (None, 1, 1, 512)
Conv2DTranspose
Output shape: (None, 2, 2, 256)
Concatenate
Output shape: (None, 2, 2, 512)
Conv2D
Output shape: (None, 2, 2, 256)
Conv2DTranspose
Output shape: (None, 4, 4, 128)
Concatenate
Output shape: (None, 4, 4, 256)
Conv2D
Output shape: (None, 4, 4, 128)
Conv2DTranspose
Output shape: (None, 8, 8, 64)
Concatenate
Output shape: (None, 8, 8, 128)
Conv2D
Output shape: (None, 8, 8, 64)
Conv2DTranspose
Output shape: (None, 16, 16, 32)
Concatenate
Output shape: (None, 16, 16, 64)
Conv2D
Output shape: (None, 16, 16, 32)
UpSampling2D
Output shape: (None, 128, 128, 32)
Conv2D
Output shape: (None, 128, 128, 64)
Conv2D
Output shape: (None, 128, 128, 32)
Conv2D
Output shape: (None, 128, 128, 4)

[66]:
```
model_2.compile(loss="categorical_crossentropy", optimizer='adam',
              metrics = ['accuracy', tf.keras.metrics.
 ↪MeanIoU(num_classes=len(SEGMENT_CLASSES)),
                         dice_coef, precision, sensitivity, specificity,␣
 ↪dice_coef_necrotic,
                         dice_coef_edema ,dice_coef_enhancing])
```

[67]:
```
early_stopping_cb = EarlyStopping(patience=5, restore_best_weights=True,␣
 ↪verbose=1)
checkpoints_cb = ModelCheckpoint("model_weights_vir.keras",␣
 ↪save_best_only=True, verbose=1)
reducee_lr_cb = ReduceLROnPlateau(patience=3, verbose=1)



callbackss = [checkpoints_cb, reducee_lr_cb, early_stopping_cb]
```

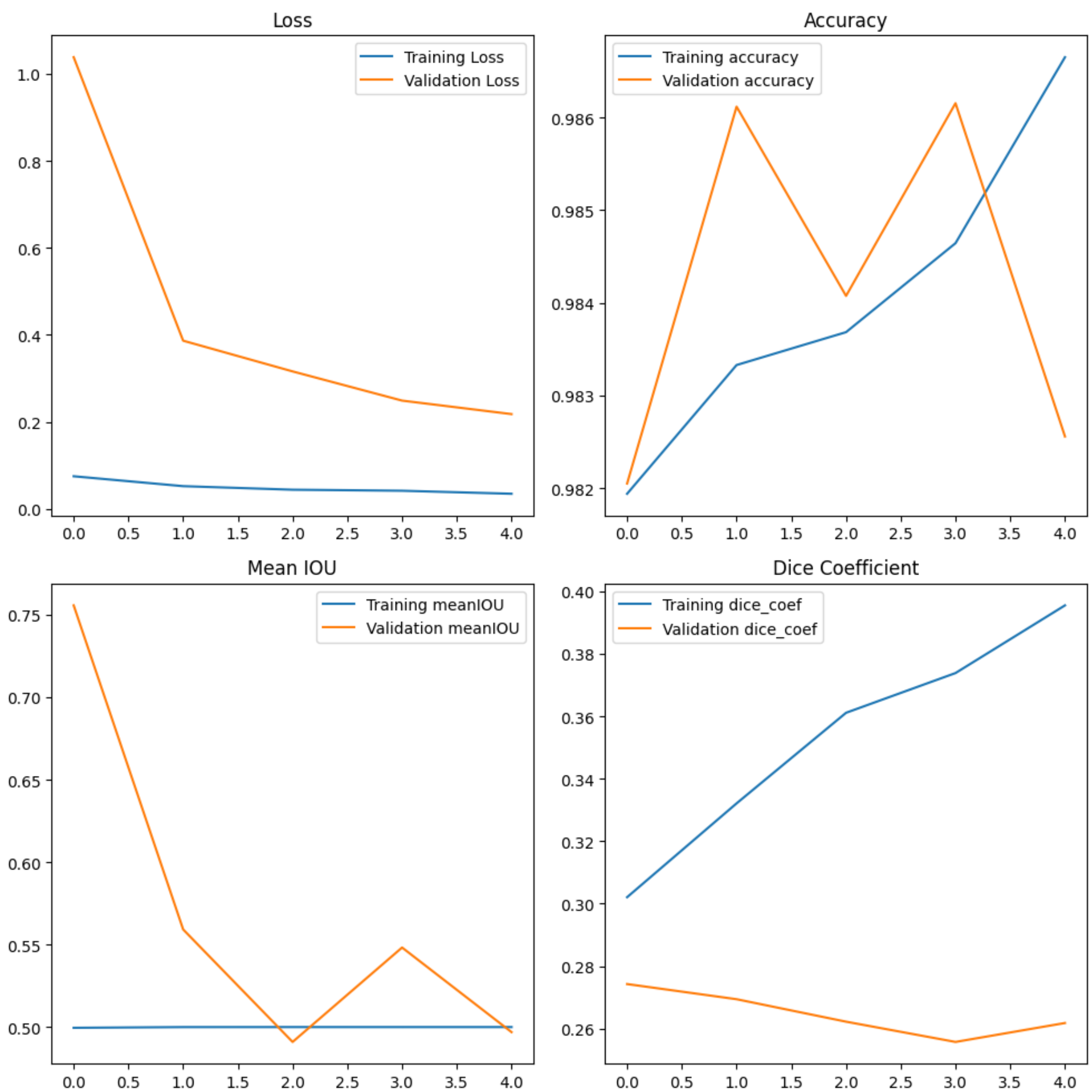
[68]:
```
batch_size=1
history = model_2.fit(data_generator_wrapper(train_ids, batch_size=batch_size),
                    epochs=5,
                    steps_per_epoch=max(1, len(train_ids)//batch_size),
                    validation_data=data_generator_wrapper(val_ids,␣
 ↪batch_size=batch_size),
                    validation_steps=max(1, len(val_ids)//batch_size),
                    initial_epoch=0,
                    callbacks = callbackss)
```

```
Epoch 1/5
249/249              0s 342ms/step -
accuracy: 0.9749 - dice_coef: 0.2784 - dice_coef_edema: 0.1482 -
dice_coef_enhancing: 0.0919 - dice_coef_necrotic: 0.0876 - loss: 0.1291 -
mean_io_u: 0.4633 - precision: 0.9826 - sensitivity: 0.9562 - specificity:
0.9947
Epoch 1: val_loss improved from inf to 1.03828, saving model to
model_weights_vir.keras
249/249              174s 472ms/step -
accuracy: 0.9749 - dice_coef: 0.2784 - dice_coef_edema: 0.1485 -
dice_coef_enhancing: 0.0921 - dice_coef_necrotic: 0.0878 - loss: 0.1288 -
mean_io_u: 0.4634 - precision: 0.9826 - sensitivity: 0.9563 - specificity:
0.9947 - val_accuracy: 0.9821 - val_dice_coef: 0.2743 - val_dice_coef_edema:
1.0159e-04 - val_dice_coef_enhancing: 1.4078e-05 - val_dice_coef_necrotic:
1.1262e-05 - val_loss: 1.0383 - val_mean_io_u: 0.7557 - val_precision: 0.9820 -
val_sensitivity: 0.9821 - val_specificity: 0.9940 - learning_rate: 0.0010
Epoch 2/5
249/249              0s 304ms/step -
```

```
accuracy: 0.9837 - dice_coef: 0.3216 - dice_coef_edema: 0.2460 -
dice_coef_enhancing: 0.1722 - dice_coef_necrotic: 0.1958 - loss: 0.0571 -
mean_io_u: 0.4974 - precision: 0.9890 - sensitivity: 0.9803 - specificity:
0.9963
Epoch 2: val_loss improved from 1.03828 to 0.38662, saving model to
model_weights_vir.keras
249/249            97s 389ms/step -
accuracy: 0.9837 - dice_coef: 0.3216 - dice_coef_edema: 0.2461 -
dice_coef_enhancing: 0.1723 - dice_coef_necrotic: 0.1959 - loss: 0.0570 -
mean_io_u: 0.4974 - precision: 0.9890 - sensitivity: 0.9803 - specificity:
0.9963 - val_accuracy: 0.9861 - val_dice_coef: 0.2694 - val_dice_coef_edema:
1.2912e-04 - val_dice_coef_enhancing: 2.3665e-07 - val_dice_coef_necrotic:
2.2190e-06 - val_loss: 0.3866 - val_mean_io_u: 0.5592 - val_precision: 0.9861 -
val_sensitivity: 0.9861 - val_specificity: 0.9954 - learning_rate: 0.0010
Epoch 3/5
249/249            0s 302ms/step -
accuracy: 0.9832 - dice_coef: 0.3624 - dice_coef_edema: 0.3499 -
dice_coef_enhancing: 0.2384 - dice_coef_necrotic: 0.2695 - loss: 0.0430 -
mean_io_u: 0.5042 - precision: 0.9923 - sensitivity: 0.9789 - specificity:
0.9973
Epoch 3: val_loss improved from 0.38662 to 0.31601, saving model to
model_weights_vir.keras
249/249            97s 391ms/step -
accuracy: 0.9832 - dice_coef: 0.3624 - dice_coef_edema: 0.3499 -
dice_coef_enhancing: 0.2384 - dice_coef_necrotic: 0.2695 - loss: 0.0430 -
mean_io_u: 0.5042 - precision: 0.9923 - sensitivity: 0.9789 - specificity:
0.9973 - val_accuracy: 0.9841 - val_dice_coef: 0.2622 - val_dice_coef_edema:
2.6873e-04 - val_dice_coef_enhancing: 1.9987e-05 - val_dice_coef_necrotic:
4.7138e-05 - val_loss: 0.3160 - val_mean_io_u: 0.4910 - val_precision: 0.9841 -
val_sensitivity: 0.9841 - val_specificity: 0.9947 - learning_rate: 0.0010
Epoch 4/5
249/249            0s 298ms/step -
accuracy: 0.9848 - dice_coef: 0.3752 - dice_coef_edema: 0.3764 -
dice_coef_enhancing: 0.2569 - dice_coef_necrotic: 0.3036 - loss: 0.0405 -
mean_io_u: 0.5015 - precision: 0.9924 - sensitivity: 0.9802 - specificity:
0.9973
Epoch 4: val_loss improved from 0.31601 to 0.24891, saving model to
model_weights_vir.keras
249/249            96s 385ms/step -
accuracy: 0.9848 - dice_coef: 0.3752 - dice_coef_edema: 0.3764 -
dice_coef_enhancing: 0.2569 - dice_coef_necrotic: 0.3036 - loss: 0.0405 -
mean_io_u: 0.5015 - precision: 0.9924 - sensitivity: 0.9802 - specificity:
0.9973 - val_accuracy: 0.9862 - val_dice_coef: 0.2557 - val_dice_coef_edema:
0.0026 - val_dice_coef_enhancing: 1.0390e-04 - val_dice_coef_necrotic:
1.9870e-04 - val_loss: 0.2489 - val_mean_io_u: 0.5482 - val_precision: 0.9862 -
val_sensitivity: 0.9862 - val_specificity: 0.9954 - learning_rate: 0.0010
Epoch 5/5
249/249            0s 299ms/step -
```

```
accuracy: 0.9874 - dice_coef: 0.3859 - dice_coef_edema: 0.4061 -
dice_coef_enhancing: 0.2639 - dice_coef_necrotic: 0.3230 - loss: 0.0330 -
mean_io_u: 0.5102 - precision: 0.9942 - sensitivity: 0.9827 - specificity:
0.9979
Epoch 5: val_loss improved from 0.24891 to 0.21790, saving model to
model_weights_vir.keras
249/249            96s 384ms/step -
accuracy: 0.9874 - dice_coef: 0.3859 - dice_coef_edema: 0.4061 -
dice_coef_enhancing: 0.2640 - dice_coef_necrotic: 0.3231 - loss: 0.0330 -
mean_io_u: 0.5102 - precision: 0.9942 - sensitivity: 0.9827 - specificity:
0.9979 - val_accuracy: 0.9826 - val_dice_coef: 0.2618 - val_dice_coef_edema:
0.0486 - val_dice_coef_enhancing: 0.0100 - val_dice_coef_necrotic: 0.0122 -
val_loss: 0.2179 - val_mean_io_u: 0.4969 - val_precision: 0.9828 -
val_sensitivity: 0.9823 - val_specificity: 0.9943 - learning_rate: 0.0010
Restoring model weights from the end of the best epoch: 5.
```

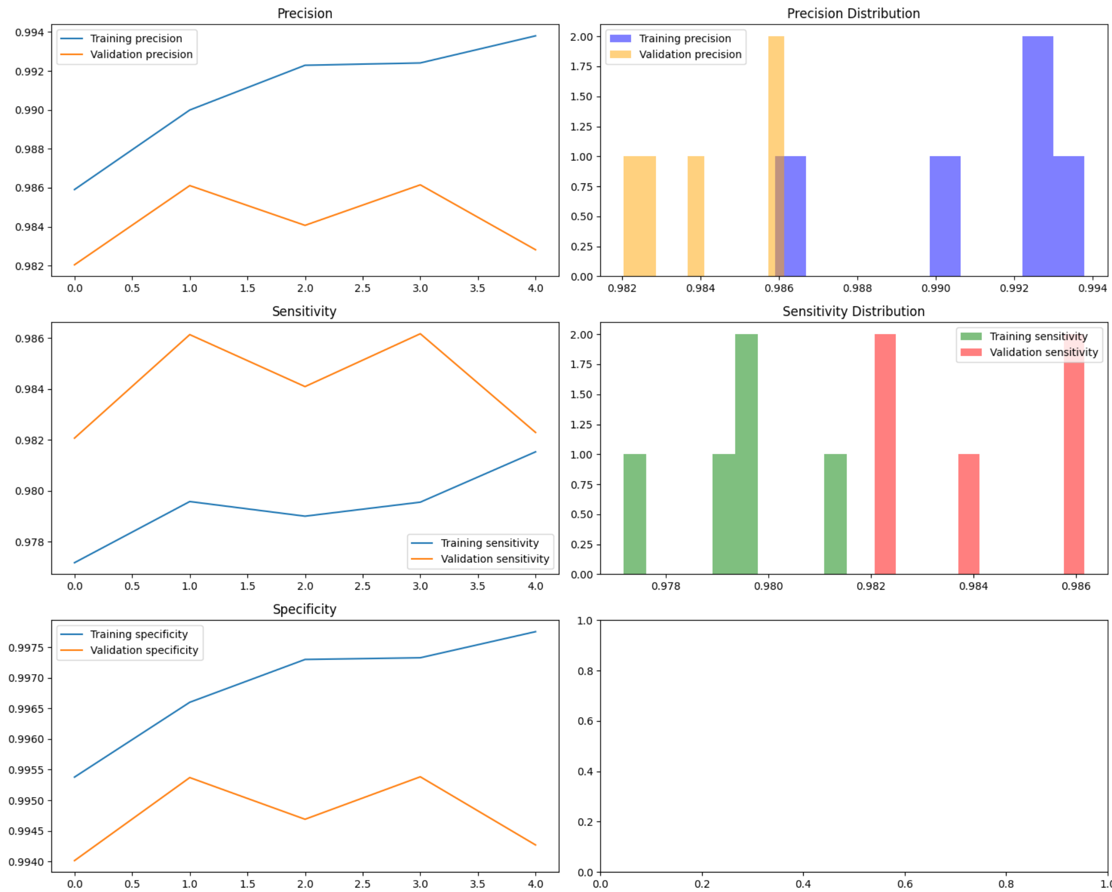
[69]:
```python
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
ax[0, 0].plot(history.history['loss'], label="Training Loss")
ax[0, 0].plot(history.history['val_loss'], label='Validation Loss')
ax[0, 0].set_title('Loss')
ax[0, 0].legend()

ax[0, 1].plot(history.history['accuracy'], label="Training accuracy")
ax[0, 1].plot(history.history['val_accuracy'], label='Validation accuracy')
ax[0, 1].set_title("Accuracy")
ax[0, 1].legend()

ax[1, 0].plot(history.history['mean_io_u'], label="Training meanIOU")
ax[1, 0].plot(history.history['val_mean_io_u'], label='Validation meanIOU')
ax[1, 0].set_title("Mean IOU")
ax[1, 0].legend()

ax[1, 1].plot(history.history['dice_coef'], label="Training dice_coef")
ax[1, 1].plot(history.history['val_dice_coef'], label='Validation dice_coef')
ax[1, 1].set_title("Dice Coefficient")
ax[1, 1].legend()

plt.tight_layout()
```



[70]:
```python
import matplotlib.pyplot as plt
```

[71]:
```python
fig, ax = plt.subplots(3, 2, figsize=(15, 12))

# Plot precision
ax[0, 0].plot(history.history['precision'], label="Training precision")
ax[0, 0].plot(history.history['val_precision'], label='Validation precision')
ax[0, 0].set_title('Precision')
ax[0, 0].legend()

# Plot sensitivity
ax[1, 0].plot(history.history['sensitivity'], label="Training sensitivity")
```

```
ax[1, 0].plot(history.history['val_sensitivity'], label='Validation␣
↪sensitivity')
ax[1, 0].set_title("Sensitivity")
ax[1, 0].legend()

# Plot specificity
ax[2, 0].plot(history.history['specificity'], label="Training specificity")
ax[2, 0].plot(history.history['val_specificity'], label='Validation␣
↪specificity')
ax[2, 0].set_title("Specificity")
ax[2, 0].legend()

# Plot histogram of precision
ax[0, 1].hist(history.history['precision'], bins=10, alpha=0.5, color='blue',␣
↪label='Training precision')
ax[0, 1].hist(history.history['val_precision'], bins=10, alpha=0.5,␣
↪color='orange', label='Validation precision')
ax[0, 1].set_title('Precision Distribution')
ax[0, 1].legend()

# Plot histogram of sensitivity
ax[1, 1].hist(history.history['sensitivity'], bins=10, alpha=0.5,␣
↪color='green', label='Training sensitivity')
ax[1, 1].hist(history.history['val_sensitivity'], bins=10, alpha=0.5,␣
↪color='red', label='Validation sensitivity')
ax[1, 1].set_title('Sensitivity Distribution')
ax[1, 1].legend()

plt.tight_layout()
plt.show()
```



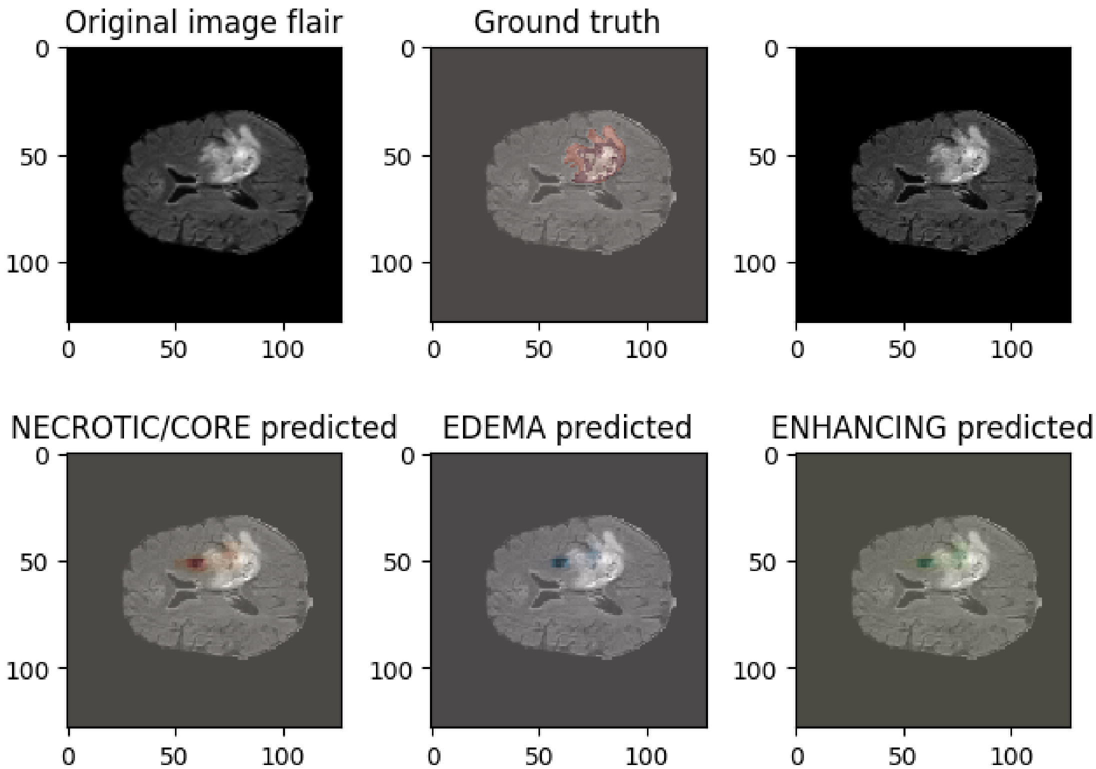
[72]:

```
# function to predict the brain tumor image from segment classes and pass it to␣
↪showing segmented image

def predict_tumors(case, start_slice=60):
    path = TRAIN_DATASET_PATH + "/" + case
    X = np.empty((VOLUME_SLICES, IMG_SIZE, IMG_SIZE, 2))

    flair = nib.load(path + "/" + case + "_flair.nii").get_fdata()
    mask = nib.load(path + "/" + case + "_seg.nii").get_fdata()
    ce = nib.load(path + "/" + case + "_t1ce.nii").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))

    pred = model_2.predict(X / np.max(X), verbose=1)
```

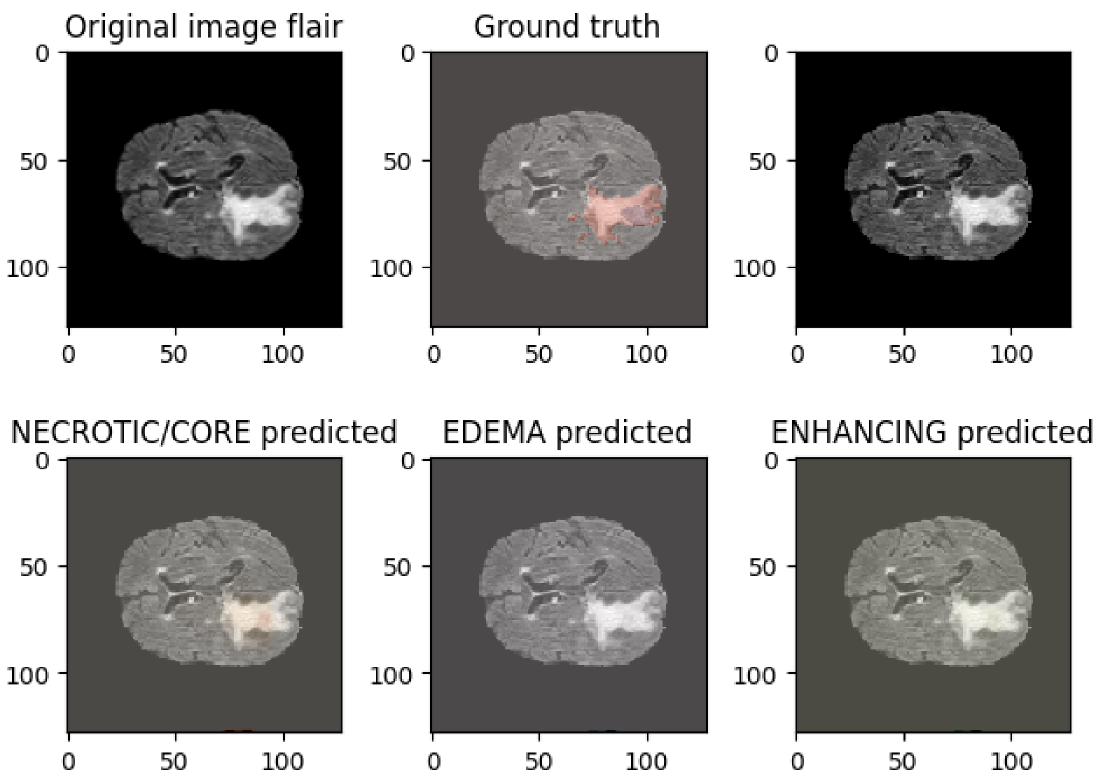
```python
    core = pred[:, :, :, 1]
    edema = pred[:, :, :, 2]
    enhancing = pred[:, :, :, 3]

    f, ax = plt.subplots(2, 3)

    for i in range(2):  # for each image, add brain background
        for j in range(3):
            ax[i, j].imshow(cv2.resize(flair[:, :,␣
↪start_slice+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE)), cmap="gray",␣
↪interpolation='none')

    ax[0, 0].imshow(cv2.resize(flair[:, :, start_slice+VOLUME_START_AT],␣
↪(IMG_SIZE, IMG_SIZE)), cmap="gray")
    ax[0, 0].title.set_text('Original image flair')

    mask = cv2.resize(mask[:, :, start_slice+VOLUME_START_AT], (IMG_SIZE,␣
↪IMG_SIZE), interpolation=cv2.INTER_NEAREST)
    ax[0, 1].imshow(mask, cmap="Reds", interpolation='none', alpha=0.3)
    ax[0, 1].title.set_text('Ground truth')

    ax[1, 0].imshow(edema[start_slice, :, :], cmap="OrRd",␣
↪interpolation='none', alpha=0.3)
    ax[1, 0].title.set_text(f'{SEGMENT_CLASSES[1]} predicted')

    ax[1, 1].imshow(core[start_slice, :, :], cmap="PuBu", interpolation='none',␣
↪alpha=0.3)
    ax[1, 1].title.set_text(f'{SEGMENT_CLASSES[2]} predicted')

    ax[1, 2].imshow(enhancing[start_slice, :, :], cmap="YlGn",␣
↪interpolation='none', alpha=0.3)
    ax[1, 2].title.set_text(f'{SEGMENT_CLASSES[3]} predicted')



    plt.tight_layout()
    plt.show()
    print("\n")
```

[73]:
```python
for ids in np.random.choice(train_ids, size=5, replace=False):
    predict_tumors(ids)
```

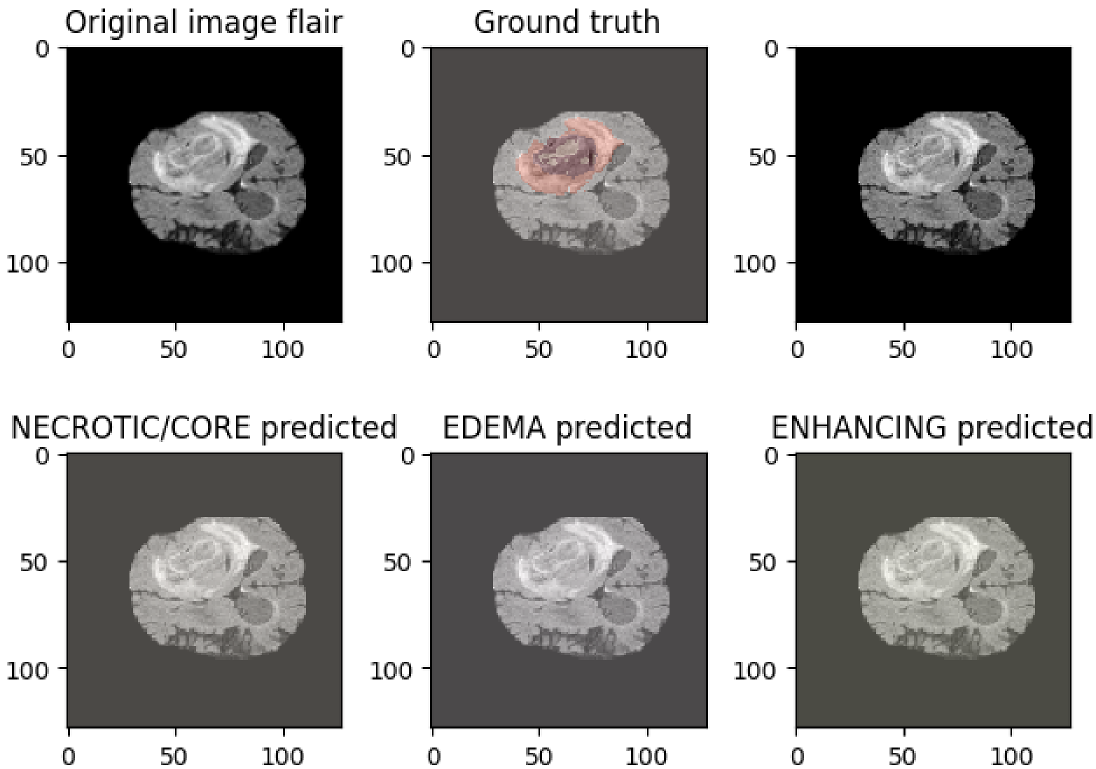
```
4/4              23s 3s/step
```



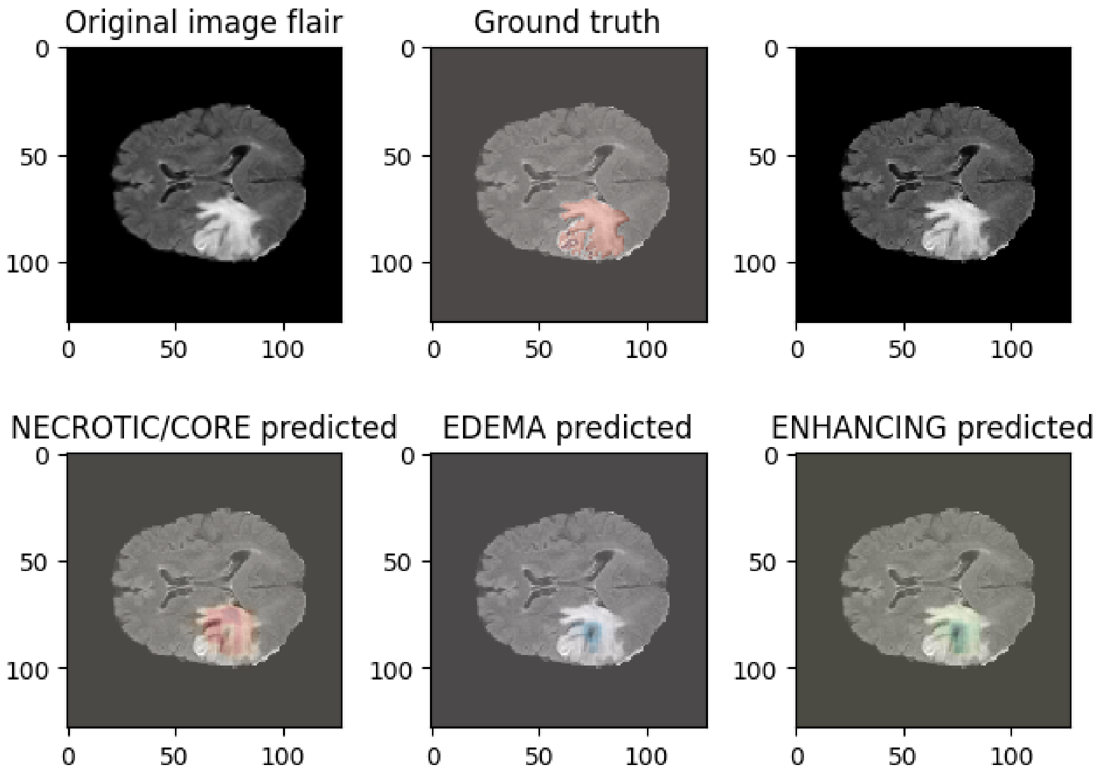

```
4/4              0s 30ms/step
```



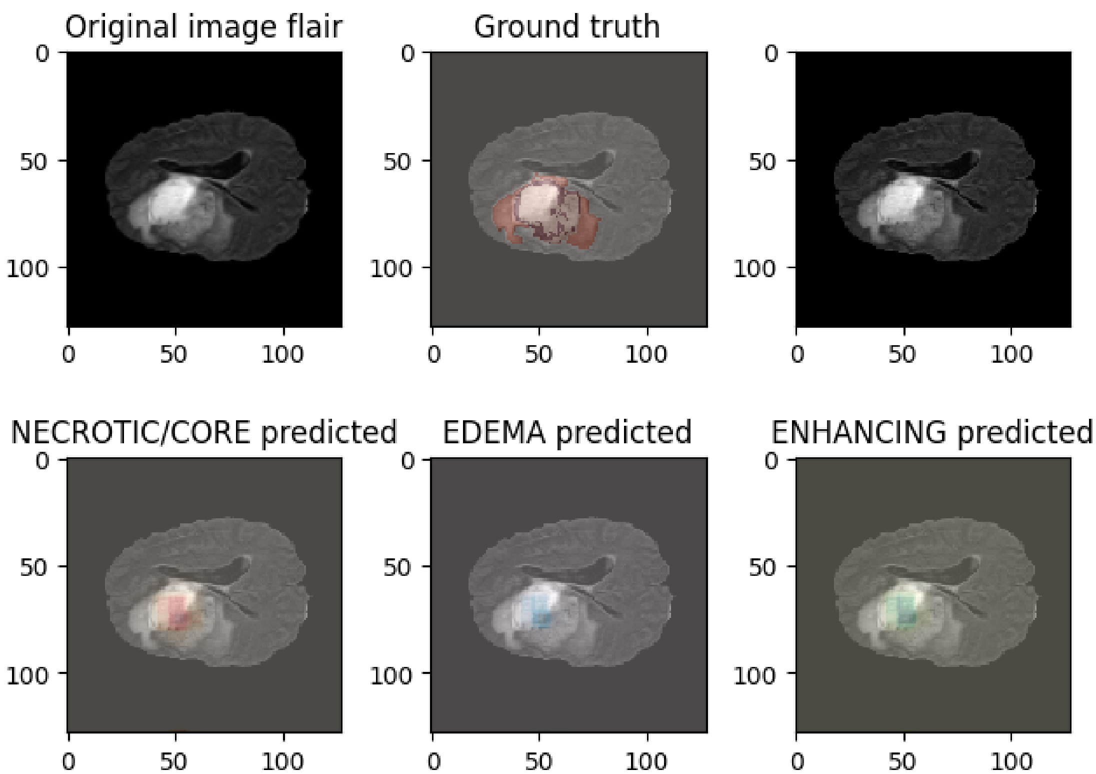

```
4/4                 0s 30ms/step
```




```
4/4              0s 27ms/step
```




```
4/4                0s 30ms/step
```



[74]:
```
import pandas as pd

# Get the accuracy, loss, sensitivity, specificity, and precision from the␣
↪history
accuracy = history.history['accuracy']
loss = history.history['loss']
sensitivity = history.history['sensitivity']
specificity = history.history['specificity']
precision = history.history['precision']

# Create a DataFrame
df = pd.DataFrame({'Epoch': range(1, len(accuracy) + 1), 'Accuracy': accuracy,␣
↪'Loss': loss, 'Sensitivity': sensitivity, 'Specificity': specificity,␣
↪'Precision': precision})

# Print the DataFrame
print(df)
```

```
   Epoch  Accuracy      Loss  Sensitivity  Specificity  Precision
```

0	1	0.981940	0.075013	0.977179	0.995378	0.985908
1	2	0.983329	0.052263	0.979582	0.996600	0.989992
2	3	0.983684	0.044084	0.979006	0.997299	0.992289
3	4	0.984644	0.041712	0.979557	0.997327	0.992409
4	5	0.986651	0.034793	0.981534	0.997753	0.993801

[75]:

```
import os
import nibabel as nib
import numpy as np
from tabulate import tabulate

# Directory paths where the NIfTI files are located
nifti_dir_train = "/kaggle/input/brats20-dataset-training-validation/
↪BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/BraTS20_Training_001"
nifti_dir_val = "/kaggle/input/brats20-dataset-training-validation/
↪BraTS2020_ValidationData/MICCAI_BraTS2020_ValidationData/
↪BraTS20_Validation_001/"

# Initialize a list to store the modalities and standard deviations
modalities_list = []

# Iterate over the files in the training data directory
for file_name_train in os.listdir(nifti_dir_train):
    # Check if the file is a NIfTI file
    if file_name_train.endswith(".nii") or file_name_train.endswith(".nii.gz"):
        # Get the file path
        file_path_train = os.path.join(nifti_dir_train, file_name_train)
        # Load the NIfTI file
        nifti_img_train = nib.load(file_path_train)
        # Get the image data array
        image_data_train = nifti_img_train.get_fdata()
        # Calculate the mean and standard deviation of the modalities
        modalities_mean_train = np.mean(image_data_train, axis=(0, 1, 2))
        modalities_std_train = np.std(image_data_train, axis=(0, 1, 2))
        # Append the modalities and standard deviations to the list
        modalities_list.append([file_name_train, modalities_mean_train,␣
↪modalities_std_train])

# Iterate over the files in the validation data directory
for file_name_val in os.listdir(nifti_dir_val):
    # Check if the file is a NIfTI file
    if file_name_val.endswith(".nii") or file_name_val.endswith(".nii.gz"):
        # Get the file path
        file_path_val = os.path.join(nifti_dir_val, file_name_val)
        # Load the NIfTI file
        nifti_img_val = nib.load(file_path_val)
        # Get the image data array
```

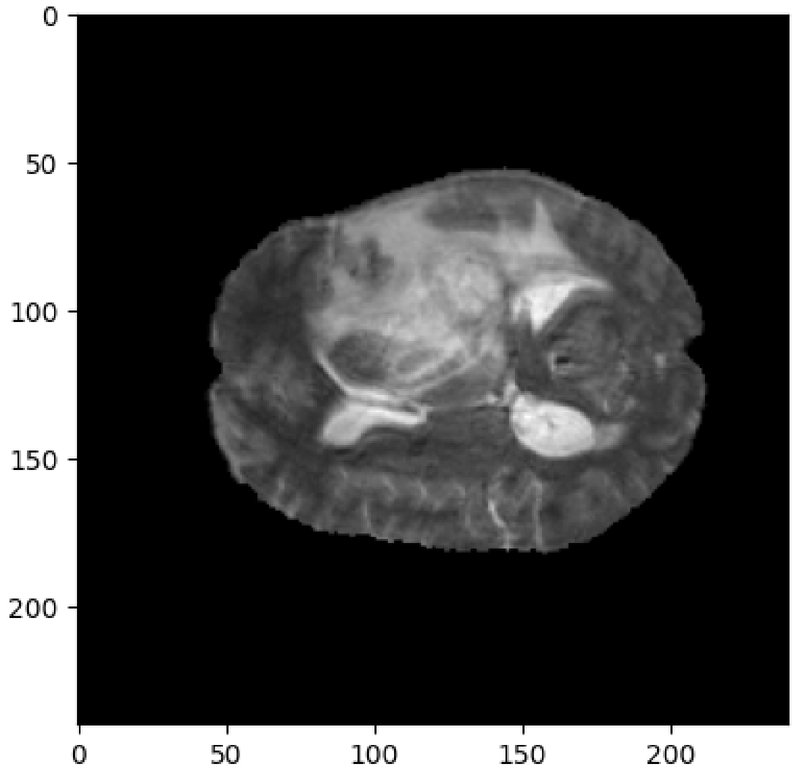
```
        image_data_val = nifti_img_val.get_fdata()
        # Calculate the mean and standard deviation of the modalities
        modalities_mean_val = np.mean(image_data_val, axis=(0, 1, 2))
        modalities_std_val = np.std(image_data_val, axis=(0, 1, 2))
        # Append the modalities and standard deviations to the list
        modalities_list.append([file_name_val, modalities_mean_val,␣
↪modalities_std_val])

# Define the table headers
headers = ["NIfTI File", "Mean", "Standard Deviation"]

# Print the table
print(tabulate(modalities_list, headers, tablefmt="grid"))
```

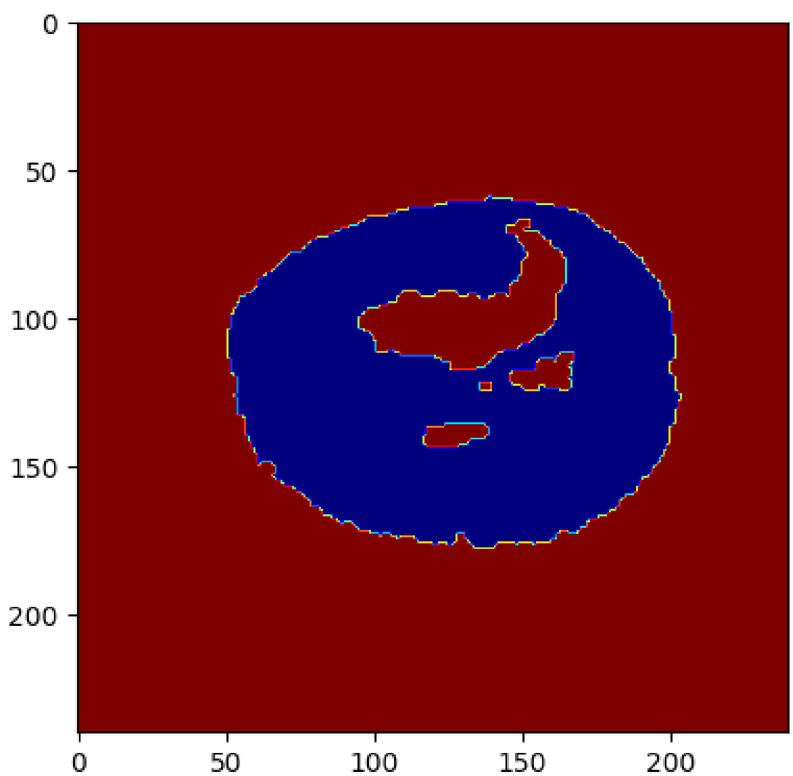
```
+-----------------------------------+------------+----------------------+
| NIfTI File                        |       Mean |   Standard Deviation |
+===================================+============+======================+
| BraTS20_Training_001_t2.nii       | 17.2514    |            44.9792   |
+-----------------------------------+------------+----------------------+
| BraTS20_Training_001_t1ce.nii     | 62.7716    |           155.079    |
+-----------------------------------+------------+----------------------+
| BraTS20_Training_001_t1.nii       | 53.2871    |           130.785    |
+-----------------------------------+------------+----------------------+
| BraTS20_Training_001_seg.nii      |  0.0519712 |             0.352661 |
+-----------------------------------+------------+----------------------+
| BraTS20_Training_001_flair.nii    | 26.0219    |            66.7654   |
+-----------------------------------+------------+----------------------+
| BraTS20_Validation_001_t2.nii     | 30.1645    |            72.3258   |
+-----------------------------------+------------+----------------------+
| BraTS20_Validation_001_t1.nii     | 58.3632    |           130.018    |
+-----------------------------------+------------+----------------------+
| BraTS20_Validation_001_t1ce.nii   | 66.2126    |           149.149    |
+-----------------------------------+------------+----------------------+
| BraTS20_Validation_001_flair.nii  | 37.6599    |            89.208    |
+-----------------------------------+------------+----------------------+
```

[85]:

```
import matplotlib.pyplot as plt
import nibabel as nib
import SimpleITK as sitk

# Provide the path to your input image
image_path = "/kaggle/input/brats20-dataset-training-validation/
↪BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/BraTS20_Training_001/
↪BraTS20_Training_001_t2.nii"

# Load the image
image = nib.load(image_path)
```

```
image_data = image.get_fdata()

# Display slices of the image
num_slices = image_data.shape[-1]  # Number of slices in the image
mid_slice = num_slices // 2  # Select the middle slice or adjust as needed

# Display the selected slice
plt.imshow(image_data[..., mid_slice], cmap='gray')
plt.axis('on')
plt.show()

# Load the image using SimpleITK
image_sitk = sitk.ReadImage(image_path)

# Apply thresholding to segment active tumor
threshold = sitk.BinaryThresholdImageFilter()
threshold.SetLowerThreshold(1)  # Adjust the threshold value as per your data
threshold.SetUpperThreshold(100)  # Adjust the threshold value as per your data
threshold.SetInsideValue(0)
threshold.SetOutsideValue(1)
segmented_image = threshold.Execute(image_sitk)

# Optional: Apply morphological operations for refinement
morphology = sitk.BinaryMorphologicalOpeningImageFilter()
morphology.SetKernelRadius(2)  # Adjust the kernel radius as per your␣
 ↪requirement
segmented_image = morphology.Execute(segmented_image)

# Save the segmented active tumor image
output_path = "image2.nii"
sitk.WriteImage(segmented_image, output_path)

# Load the segmented active tumor image
segmented_image_data = nib.load(output_path).get_fdata()

# Choose a slice to display (assuming a 2D image or selecting a slice from a 3D␣
 ↪image)
slice_index = 100  # Adjust the slice index as needed

# Display the segmented active tumor image
plt.imshow(segmented_image_data[:, :, slice_index], cmap='jet')
plt.axis('on')
plt.show()
```


0
50
100
150
200
0
50
100
150
200


0
50
100
150
200
0
50
100
150
200