# MACHINE LEARNING

100% IMPLEMENTATION

Paargav Shanker Su
2018103048 | CSE 'Q' BATCH

Shankar Kumar S
2018103063 | CSE 'Q' BATCH

## Title

Image Segmentation for Brain Tumor Detection

## Abstract

In image related analysis and tasks such as image segmentation, image classification, image generation etc, deep convolutional neural networks have been proven to be very effective. We make use of a convolution neural network architecture that consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. This type of architecture is capable of image segmentation with very high efficacy. This type or architecture provides several advantages for segmentation tasks: first, this model allows for the use of global location and context at the same time. Second, it works with very few training samples and provides better performance for segmentation tasks. We intend to apply this novel deep learning architecture to automate nucleus detection. By automating brain tumour detection, it enables us to help unlock cures faster. On top of detecting the tumours, our project will also be able to segment the tumour from the image.

## Introduction

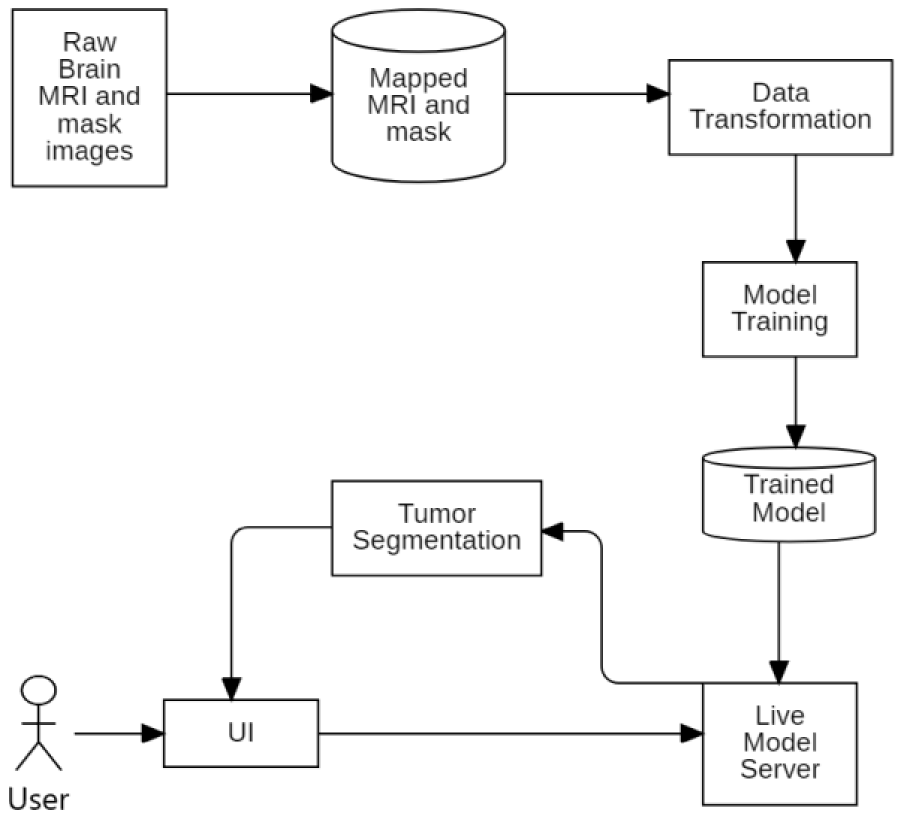
Several conventional computer vision techniques have been outperformed by deep learning in areas such as image classification, segmentation, tracking etc. Convolutional Neural Networks (CNN) is one of the most famous deep learning architectures. Its true effectiveness came to the surface when it is trained on more powerful machines with GPUs and leveraging large amount of training data. We generally use deep CNN models for image classification. But in the medical field, image segmentation has its own significance. For example, image segmentation is widely used in the localization of cancerous and defected regions in MRI and CT scans.

Segmentation of brain tumors allows visualization of the size and position of a tumor within the brain, and also provides for comparison of pre-operative and post-operative images or visualization of changes in a tumor's size and shape through a treatment time period. CNN models are used along with cross entropy loss as a pixel-wise measure in medical image segmentation

The most popular deep CNN architectures for medical image segmentation is based on an encoder-decoder architecture. The widely used models in this domain is U-Net and V-Net architectures. We will be using U-Net architecture to detect and localize brain tumors.

## Overall Architecture

### Project Architecture



## Module Architecture

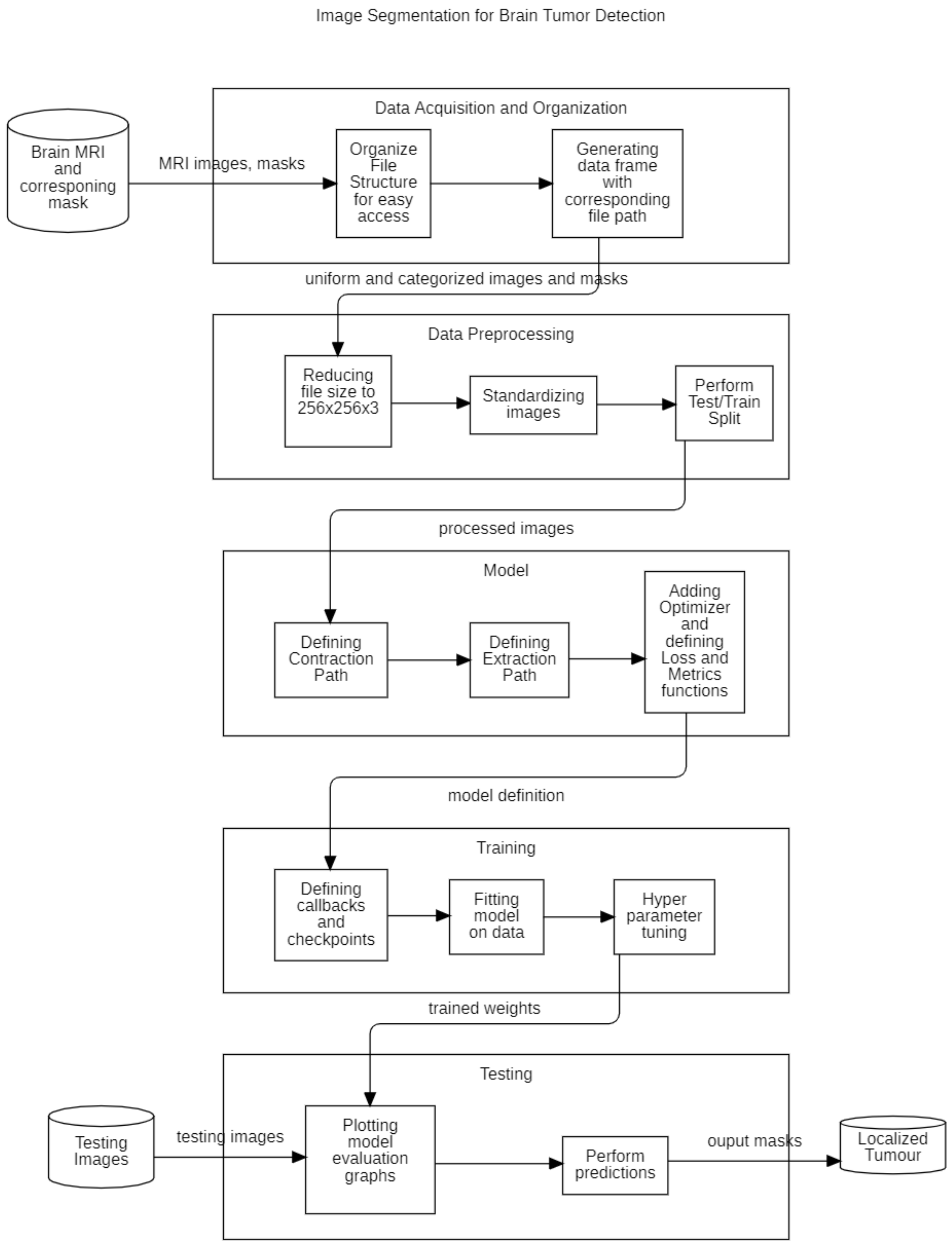
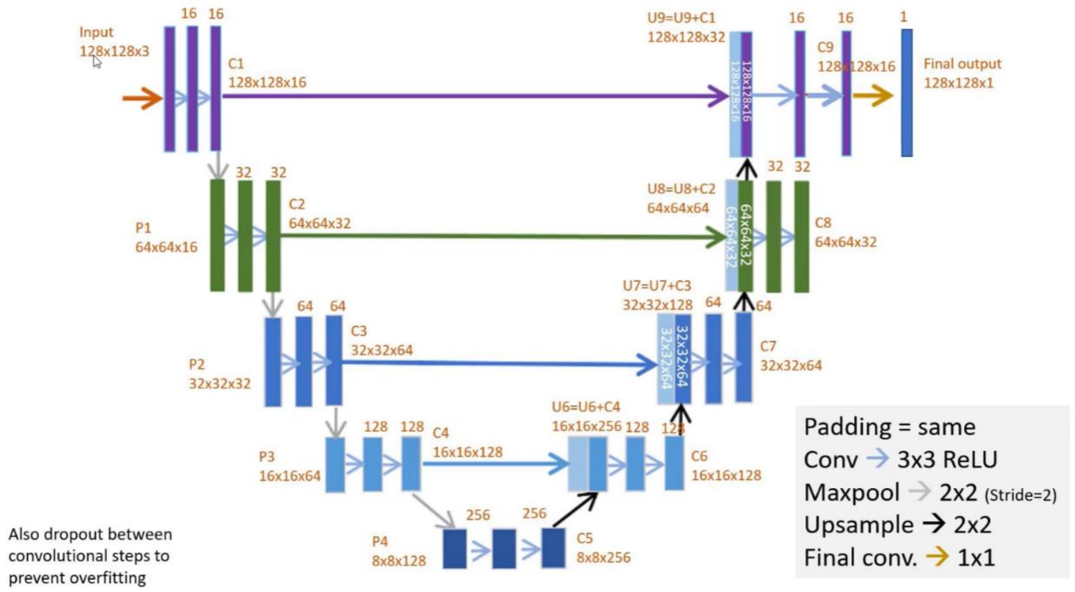


Image Segmentation for Brain Tumor Detection
Brain MRI and corresponing mask
MRI images, masks
Data Acquisition and Organization
Organize File Structure for easy access
Generating data frame with corresponding file path
uniform and categorized images and masks
Data Preprocessing
Reducing file size to 256x256x3
Standardizing images
Perform Test/Train Split
processed images
Model
Defining Contraction Path
Defining Extraction Path
Adding Optimizer and defining Loss and Metrics functions
model definition
Training
Defining callbacks and checkpoints
Fitting model on data
Hyper parameter tuning
trained weights
Testing
Testing Images
testing images
Plotting model evaluation graphs
Perform predictions
ouput masks
Localized Tumour

## Model Architecture



## Modules

**Input -** Images

**Output -** Masks

**<u>Module 1 - Data Acquisition and organization</u>**

- First, we take messy and disorganized images and their corresponding training masks.
- Then we organize the file structure for easy access and efficient access.
- Data frame with corresponding file path is generated.

**Module Input -** Messy and disorganized images and training masks

**Module Output** - Uniform, categorized and refactored images and training masks

**<u>Module 2 - Data Preprocessing</u>**

- Each of the images are read.
- Images are resized to 256x256x3.
- Then we standardize those images.
- After that we perform train/test split.

**Module Input -** Unprocessed images

**Module Output -** Processed Images in uniform size. Data is split into test and train

**<u>Module 3 – Model</u>**

- Here we define the contraction path of the model architecture.
- Then we define the extraction path of the model architecture.
- And then we add optimizer and define loss and metric functions.

**Module Input -** Image size

**Module Output -** Defined model ready for training

**Module 4 – Training**

- Callbacks and checkpoints are defined.
- Model is fitted on the data.
- Hyper parameter tuning is performed.

**Module Input -** Training images and defined model

**Module Output -** Trained weights

**Module 5 – Testing**

- Here we plot model evaluation graphs.
- And then we perform predictions on test images.

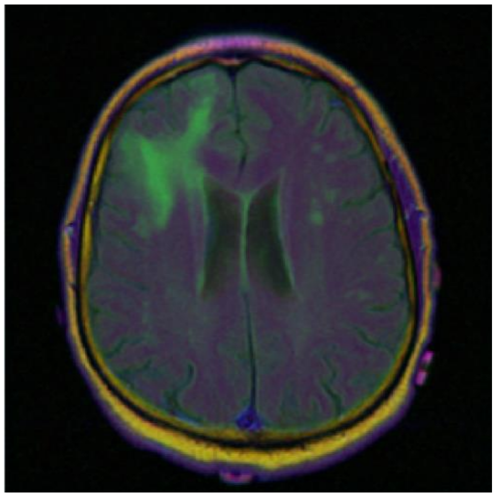
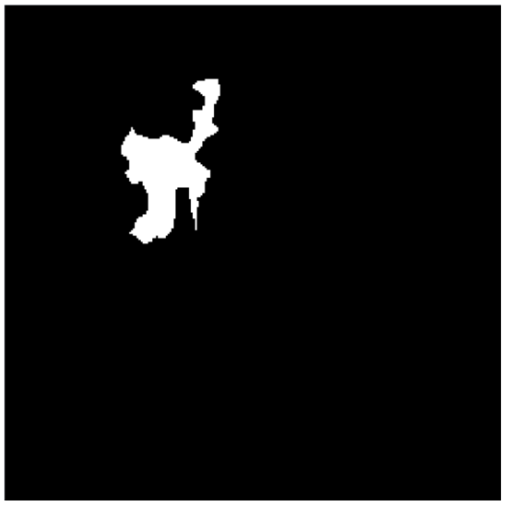
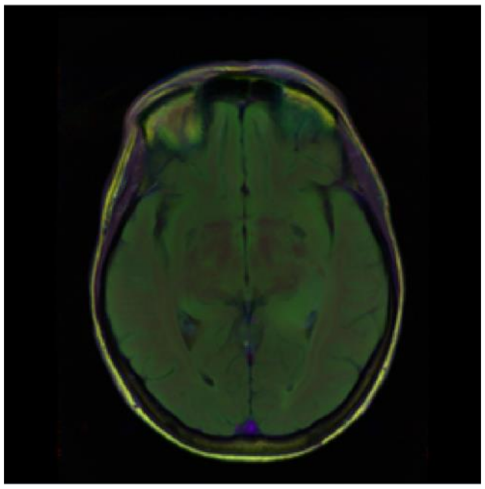
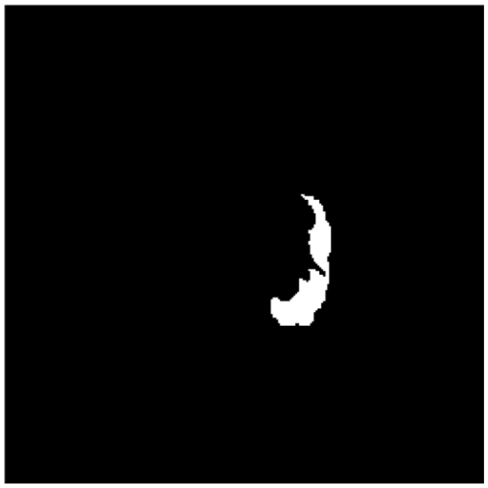
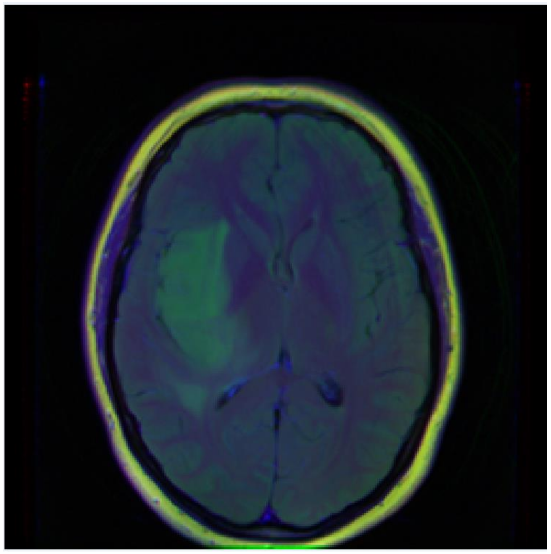
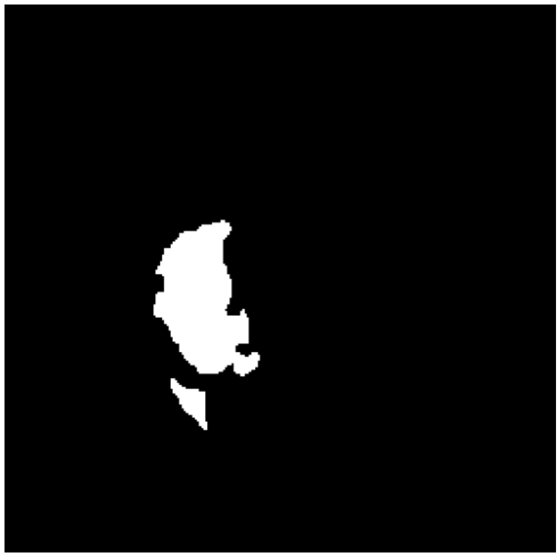
**Module Input -** Testing images

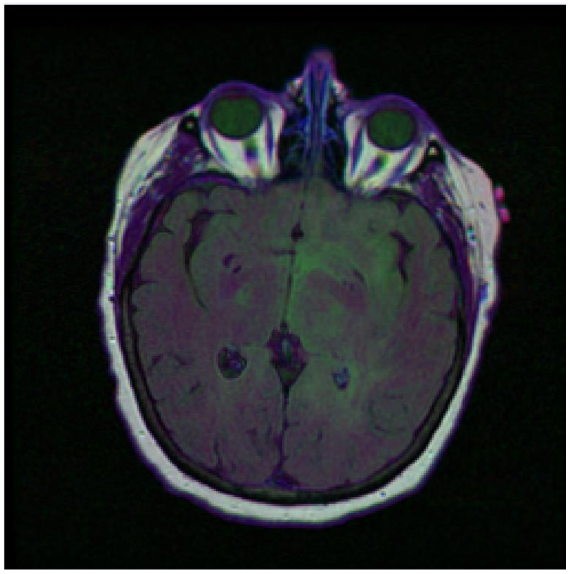
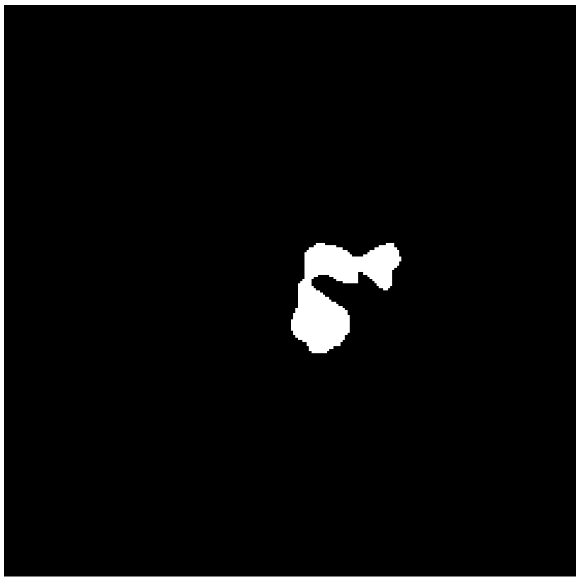
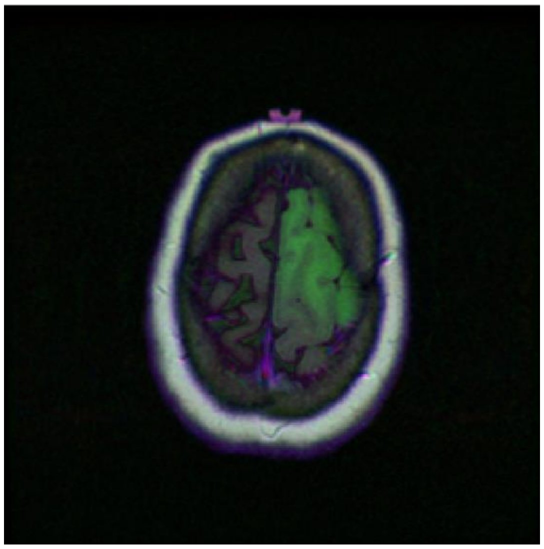
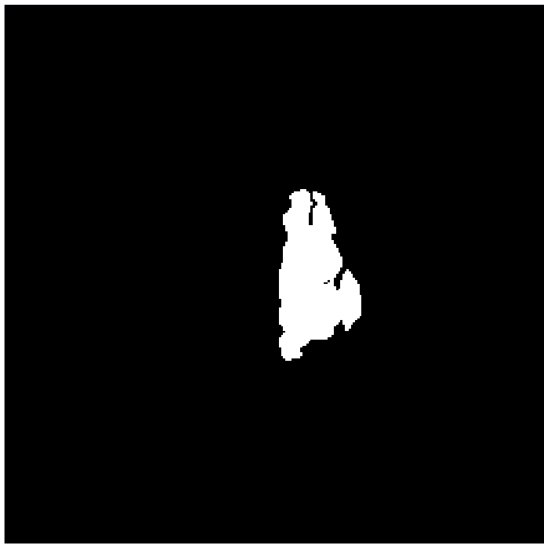
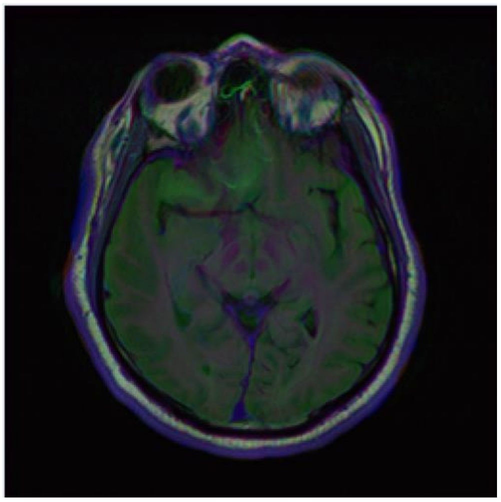
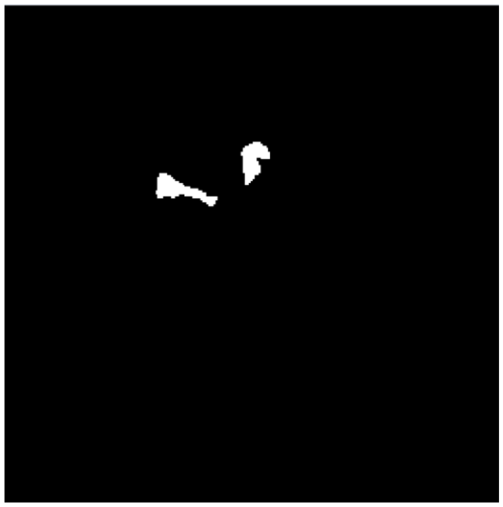
**Module Output -** Output masks

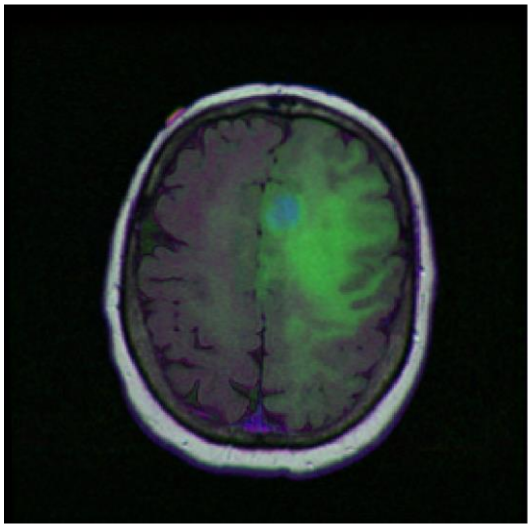
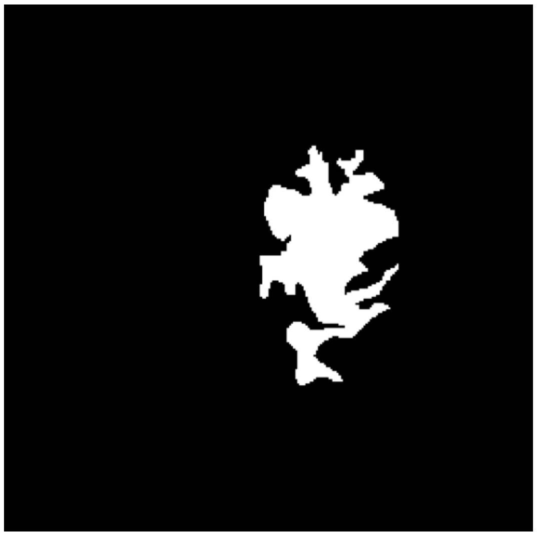
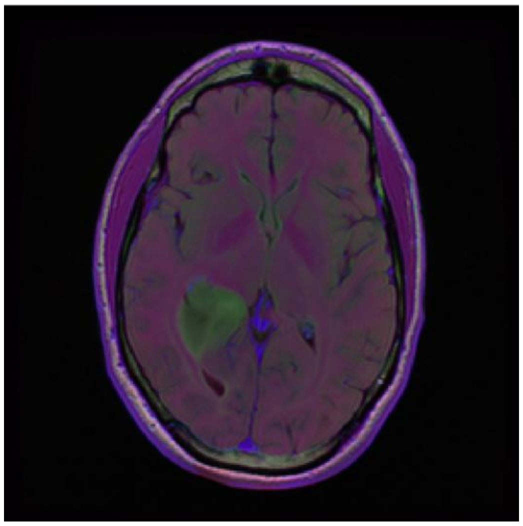
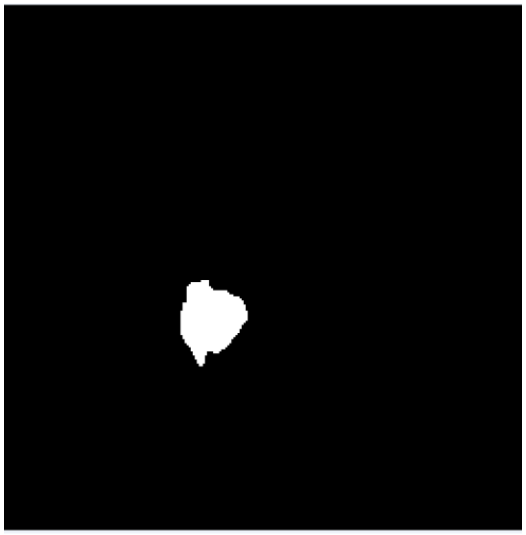
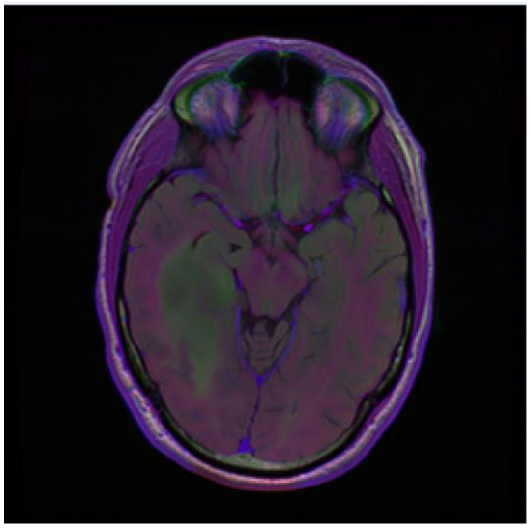
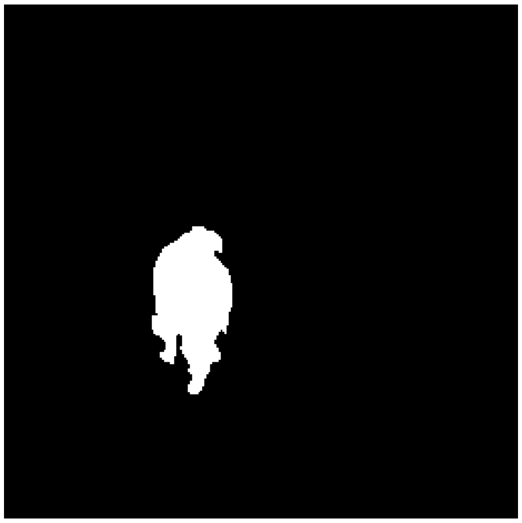
## Dataset and Implementation Details

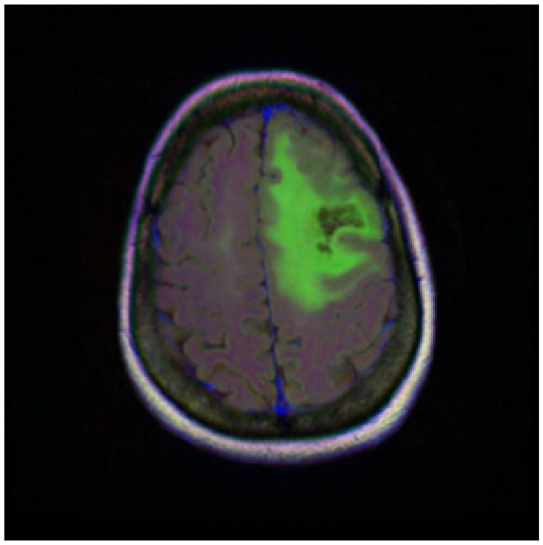
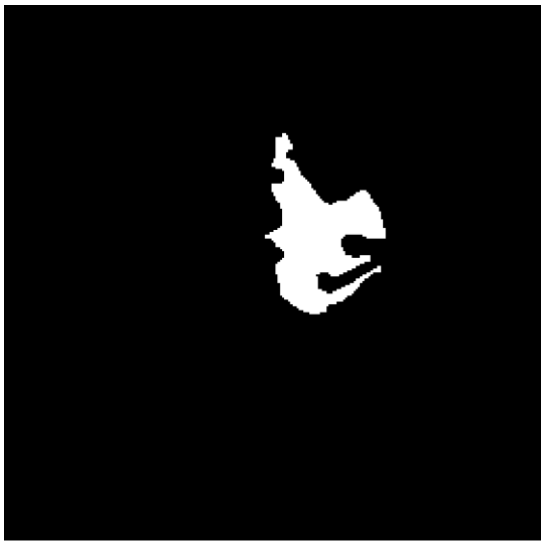
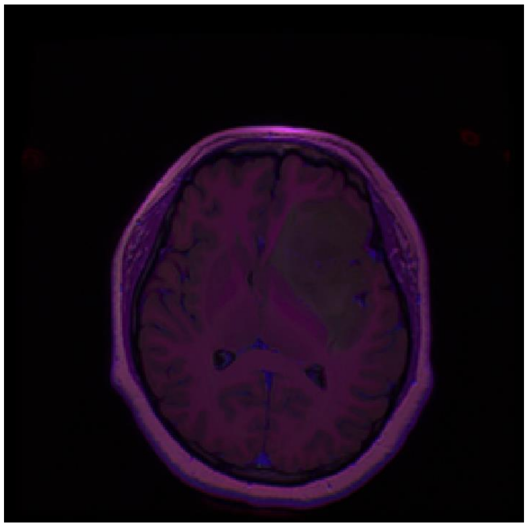
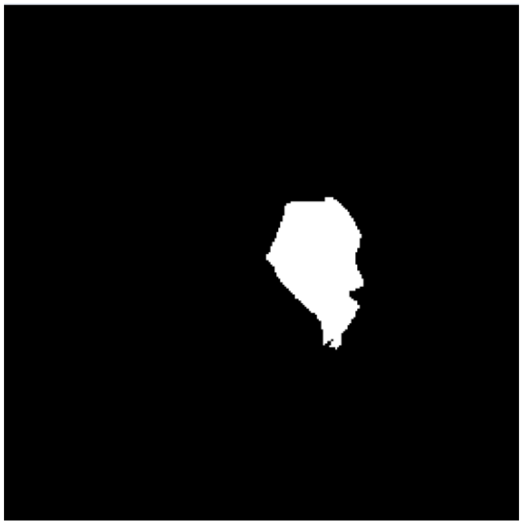
The Brain MRI dataset that we have considered has 3929 brain MRI images. Each of the brain MRI image has a corresponding mask image associated with it. Thus, the total number of images in the data set is 7858 images. The dataset comprises of brain MRI images that have a tumor and also contains brain MRI images that do not have a tumor. If a particular brain MRI image has a tumor, the corresponding mask has the tumor segmented with white pixels. Whereas if the brain MRI image does not have a tumor, the corresponding mask is a plain black image.

https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation

| Brain MRI Image | Mask Image |
|---|---|
|  |  |
|  |  |
|  |  |

| | |
|---|---|
|  |  |
|  |  |
|  |  |

| | |
|---|---|
|  |  |
|  |  |
|  |  |

| | |
|---|---|
|  |  |
|  |  |

## Implementation Details

### Packages

**Numpy –** used to work with arrays

**Pandas –** used to analyze data and structure them into dataframes

**Matplotlib –** used to create interactive visualizations

**Seaborn –** provides high level interface or statistical graphs

**cv2 –** used to work with images

**Tensorflow –** used to build, train and deploy machine learning models

**Tensorflow.keras –** high level tensorflow api which is used to define and train neural network models

**Random –** used to generate random numbers

**Sklearn.model_selection.train_test_split –** used to split train and test data

### Algorithm

Input image is reshaped into a 256x256x3 matrix

**Contraction Path:**

- Convolution and max pool layers are applied numerous times on the input image which will result in extracted features
- At the end of this, our model will be able to detect nuclei in the image
- The results of the contraction path is now fed into the expansion path

**Expansion path:**

- The matrix is fed into numerous Transposed Convolution layers and upsampling layers
- At the end of this, the model will result in a 128x128x16 matrix

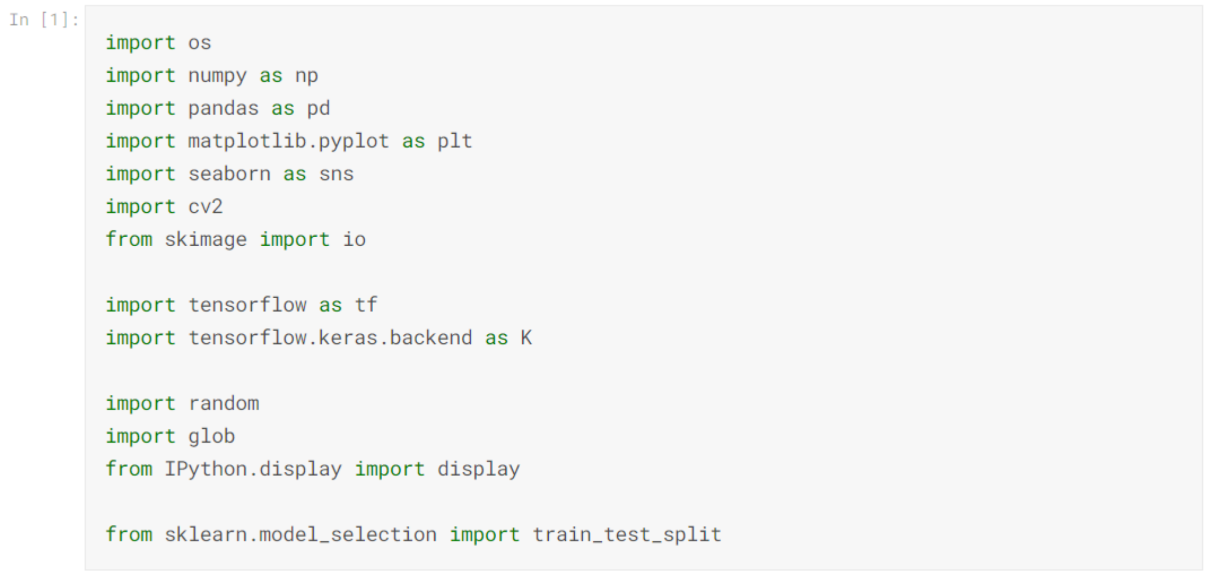
Dimensionality reduction is applied on the matrix which results in a 128x128x1 image which corresponds to the output masks which contain the segmentation for the nuclei.

Trained is performed for several epochs and in batches, until the model converges and results in minimum loss.

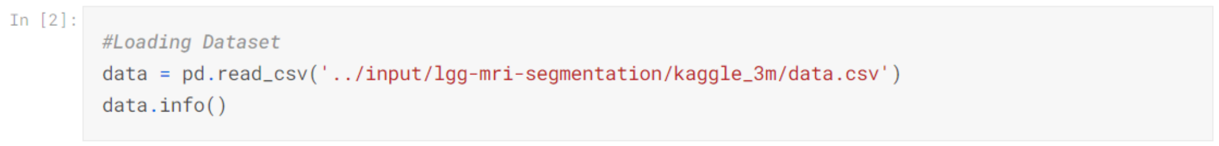
## Screenshots

### MODULE 1 - Data Acquisition and Organization

#### Importing Libraries and Dataset



```
In [1]:
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
from skimage import io

import tensorflow as tf
import tensorflow.keras.backend as K

import random
import glob
from IPython.display import display

from sklearn.model_selection import train_test_split
```



```
In [2]:
#Loading Dataset
data = pd.read_csv('../input/lgg-mri-segmentation/kaggle_3m/data.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110 entries, 0 to 109
Data columns (total 18 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Patient                   110 non-null    object
 1   RNASeqCluster             92 non-null     float64
 2   MethylationCluster        109 non-null    float64
 3   miRNACluster              110 non-null    int64
 4   CNCluster                 108 non-null    float64
 5   RPPACluster               98 non-null     float64
 6   OncosignCluster           105 non-null    float64
 7   COCCluster                110 non-null    int64
 8   histological_type         109 non-null    float64
 9   neoplasm_histologic_grade 109 non-null    float64
 10  tumor_tissue_site         109 non-null    float64
 11  laterality                109 non-null    float64
 12  tumor_location            109 non-null    float64
 13  gender                    109 non-null    float64
 14  age_at_initial_pathologic 109 non-null    float64
 15  race                      108 non-null    float64
 16  ethnicity                 102 non-null    float64
 17  death01                   109 non-null    float64
dtypes: float64(15), int64(2), object(1)
memory usage: 15.6+ KB
```

In [3]:

```
#Displaying dataset
data.head(10)
```

Out[3]:

| | Patient | RNASeqCluster | MethylationCluster | miRNACluster | CNCluster | RPPACluster | OncosignCluster | COCCluste |
|---|---|---|---|---|---|---|---|---|
| 0 | TCGA_CS_4941 | 2.0 | 4.0 | 2 | 2.0 | NaN | 3.0 | 2 |
| 1 | TCGA_CS_4942 | 1.0 | 5.0 | 2 | 1.0 | 1.0 | 2.0 | 1 |
| 2 | TCGA_CS_4943 | 1.0 | 5.0 | 2 | 1.0 | 2.0 | 2.0 | 1 |
| 3 | TCGA_CS_4944 | NaN | 5.0 | 2 | 1.0 | 2.0 | 1.0 | 1 |
| 4 | TCGA_CS_5393 | 4.0 | 5.0 | 2 | 1.0 | 2.0 | 3.0 | 1 |
| 5 | TCGA_CS_5395 | 2.0 | 4.0 | 2 | 2.0 | NaN | 3.0 | 2 |
| 6 | TCGA_CS_5396 | 3.0 | 3.0 | 2 | 3.0 | 2.0 | 2.0 | 3 |
| 7 | TCGA_CS_5397 | NaN | 4.0 | 1 | 2.0 | 3.0 | 3.0 | 2 |
| 8 | TCGA_CS_6186 | 2.0 | 4.0 | 1 | 2.0 | 1.0 | 3.0 | 2 |
| 9 | TCGA_CS_6188 | 2.0 | 4.0 | 3 | 2.0 | 3.0 | 3.0 | 2 |

## Mapping patient id to filepath

In [4]:
```python
#Creating arrays of filename with its path
data_map = []
for sub_dir_path in glob.glob("/kaggle/input/lgg-mri-segmentation/kaggle_3m/"+"*"):
    #if os.path.isdir(sub_path_dir):
    try:
        dir_name = sub_dir_path.split('/')[-1]
        for filename in os.listdir(sub_dir_path):
            image_path = sub_dir_path + '/' + filename
            data_map.extend([dir_name, image_path])
    except Exception as e:
        print(e)
```

```
[Errno 20] Not a directory: '/kaggle/input/lgg-mri-segmentation/kaggle_3m/README.md'
[Errno 20] Not a directory: '/kaggle/input/lgg-mri-segmentation/kaggle_3m/data.csv'
```

In [5]:
```python
#Converting array to a dataframe
df = pd.DataFrame({"patient_id" : data_map[::2],
                   "path" : data_map[1::2]})
df.head()
```

Out[5]:

| | patient_id | path |
|---|---|---|
| 0 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... |
| 1 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... |
| 2 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... |
| 3 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... |
| 4 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... |

## Mapping brain MRI image to corresponding mask

In [6]:

```
#Sorting and mapping Brain MRI to corresponding mask
df_imgs = df[~df['path'].str.contains("mask")]
df_masks = df[df['path'].str.contains("mask")]

# File path line length images for later sorting
BASE_LEN = 89 # len(/kaggle/input/lgg-mri-segmentation/kaggle_3m/TCGA_DU_6404_19850629/TCGA_DU_64
04_19850629_ <-!!!43.tif)
END_IMG_LEN = 4 # len(/kaggle/input/lgg-mri-segmentation/kaggle_3m/TCGA_DU_6404_19850629/TCGA_DU_
6404_19850629_43 !!!->.tif)
END_MASK_LEN = 9 # (/kaggle/input/lgg-mri-segmentation/kaggle_3m/TCGA_DU_6404_19850629/TCGA_DU_64
04_19850629_43 !!!->_mask.tif)

# Data sorting
imgs = sorted(df_imgs["path"].values, key=lambda x : int(x[BASE_LEN:-END_IMG_LEN]))
masks = sorted(df_masks["path"].values, key=lambda x : int(x[BASE_LEN:-END_MASK_LEN]))
```

## Feature Extraction of presence/absence of tumor

### Final dataset - Segregating Brain MRI by Presence/Absence of Tumor

In [7]:

```
#Final Dataset with groudtruth labels of whether tumor exists or not
#Final dataframe
brain_df = pd.DataFrame({"patient_id": df_imgs.patient_id.values,
                         "image_path": imgs,
                         "mask_path": masks
                        })
def pos_neg_diagnosis(mask_path):
    #/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...
    value = np.max(cv2.imread(mask_path))
    if value > 0 :
        return 1
    else:
        return 0

brain_df['mask'] = brain_df['mask_path'].apply(lambda x: pos_neg_diagnosis(x))
brain_df
```

Out[7]:

| | patient_id | image_path | mask_path | mask |
|---|---|---|---|---|
| 0 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| 1 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| 2 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| 3 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| 4 | TCGA_DU_7010_19860307 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| ... | ... | ... | ... | ... |
| 3924 | TCGA_DU_7306_19930512 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| 3925 | TCGA_DU_7306_19930512 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| 3926 | TCGA_DU_7306_19930512 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| 3927 | TCGA_DU_7306_19930512 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |
| 3928 | TCGA_DU_7306_19930512 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 0 |

3929 rows × 4 columns

## Performing Data Visualization

In [8]:

```
#Visualization of number of images with and without tumors
class_count = brain_df['mask'].value_counts()
class_count
```

Out[8]:

```
0    2556
1    1373
Name: mask, dtype: int64
```

In [9]:

```
class_count = class_count.to_frame()
class_count
```

Out[9]:

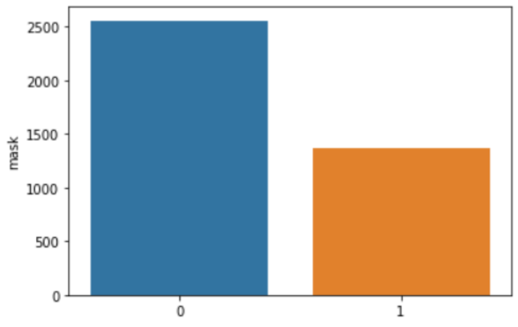
| | mask |
|---|---|
| 0 | 2556 |
| 1 | 1373 |

In [10]:

```
sns.barplot(x=class_count.index ,y='mask', data=class_count)
```

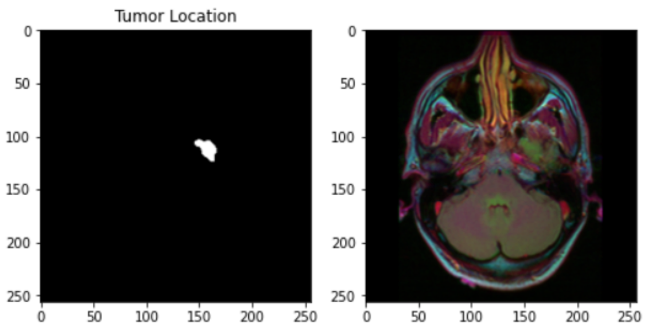
Out[10]:

```
<AxesSubplot:ylabel='mask'>
```

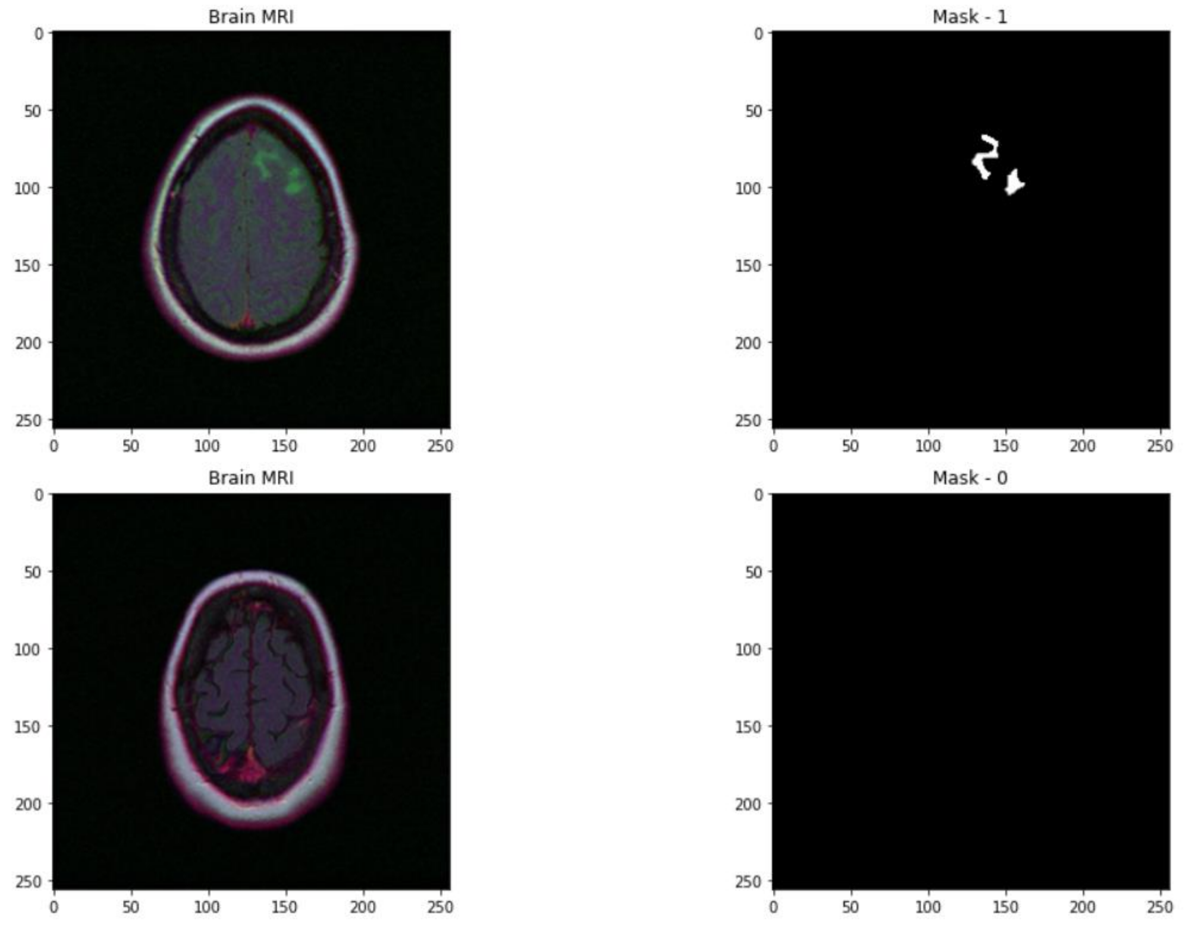


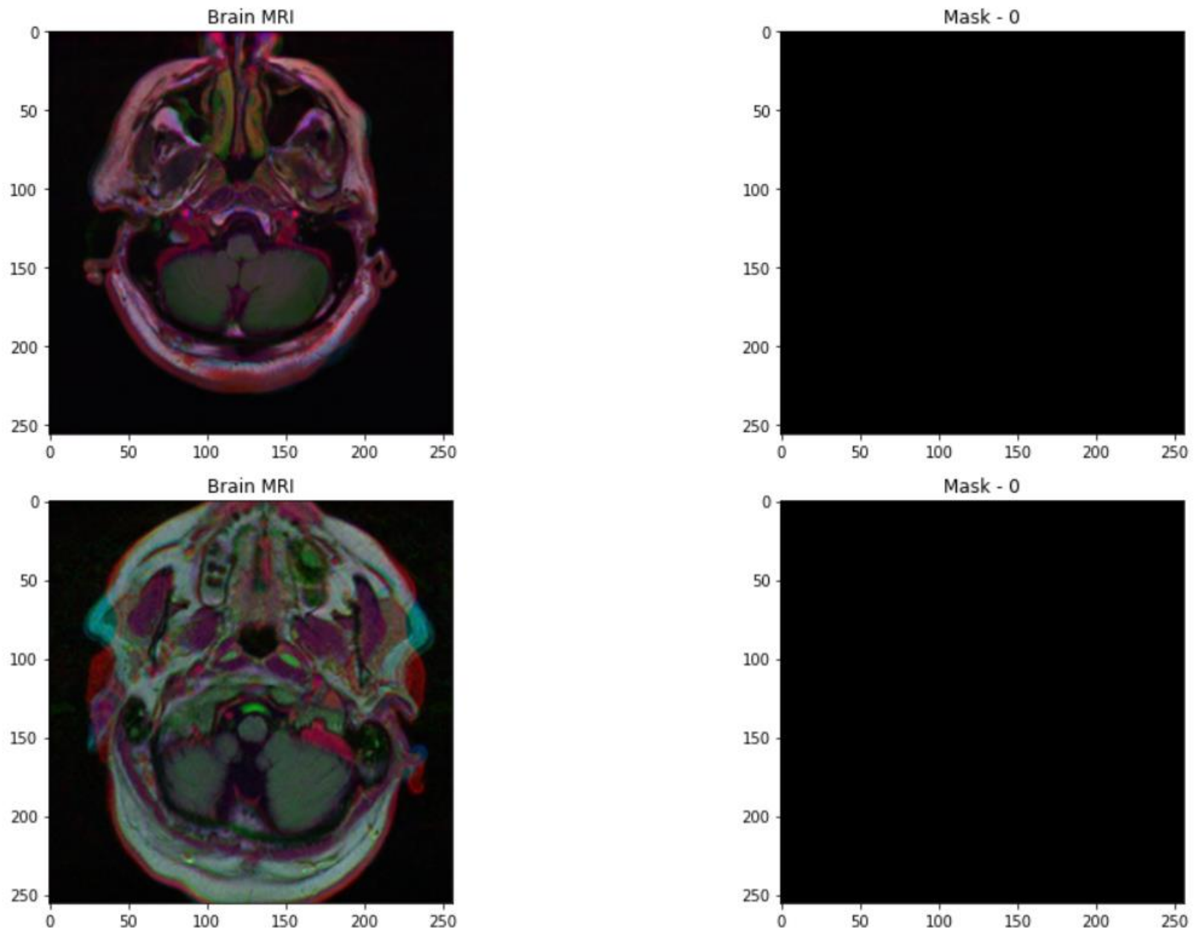
In [11]:

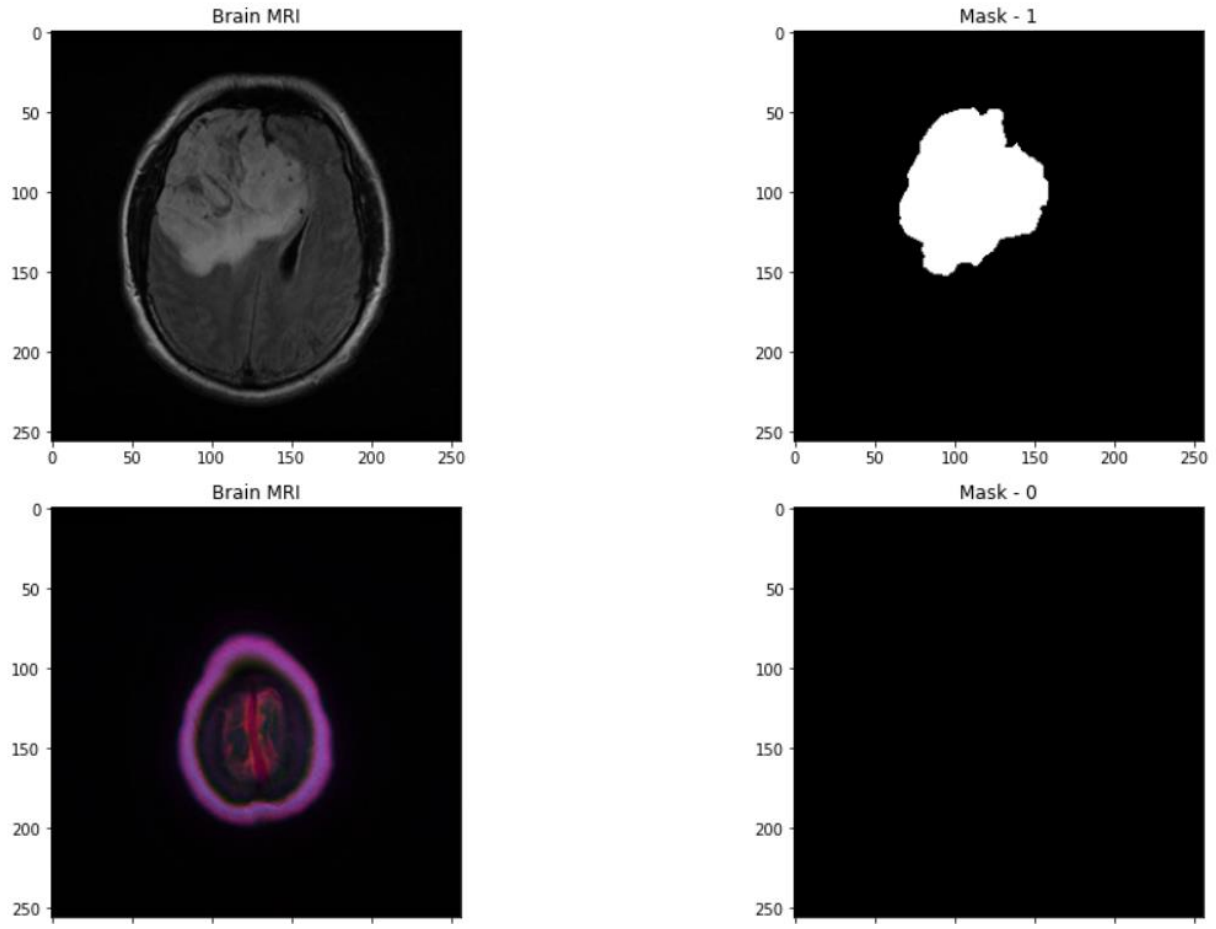
```
#Plotting Single Brain MRI image and mask with tumour
for i in range(len(brain_df)):
    if cv2.imread(brain_df.mask_path[i]).max() > 0:
        break

plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.imshow(cv2.imread(brain_df.mask_path[i]));
plt.title('Tumor Location')

plt.subplot(1,2,2)
plt.imshow(cv2.imread(brain_df.image_path[i]));
```



In [12]:
```
# More Plot examples
fig, axs = plt.subplots(6,2, figsize=(16,26))
count = 0
for x in range(6):
  i = random.randint(0, len(brain_df)) # select a random index
  axs[count][0].title.set_text("Brain MRI") # set title
  axs[count][0].imshow(cv2.imread(brain_df.image_path[i])) # show MRI
  axs[count][1].title.set_text("Mask - " + str(brain_df['mask'][i])) # plot title on the mask (0
or 1)
  axs[count][1].imshow(cv2.imread(brain_df.mask_path[i])) # Show corresponding mask
  count += 1

fig.tight_layout()
```

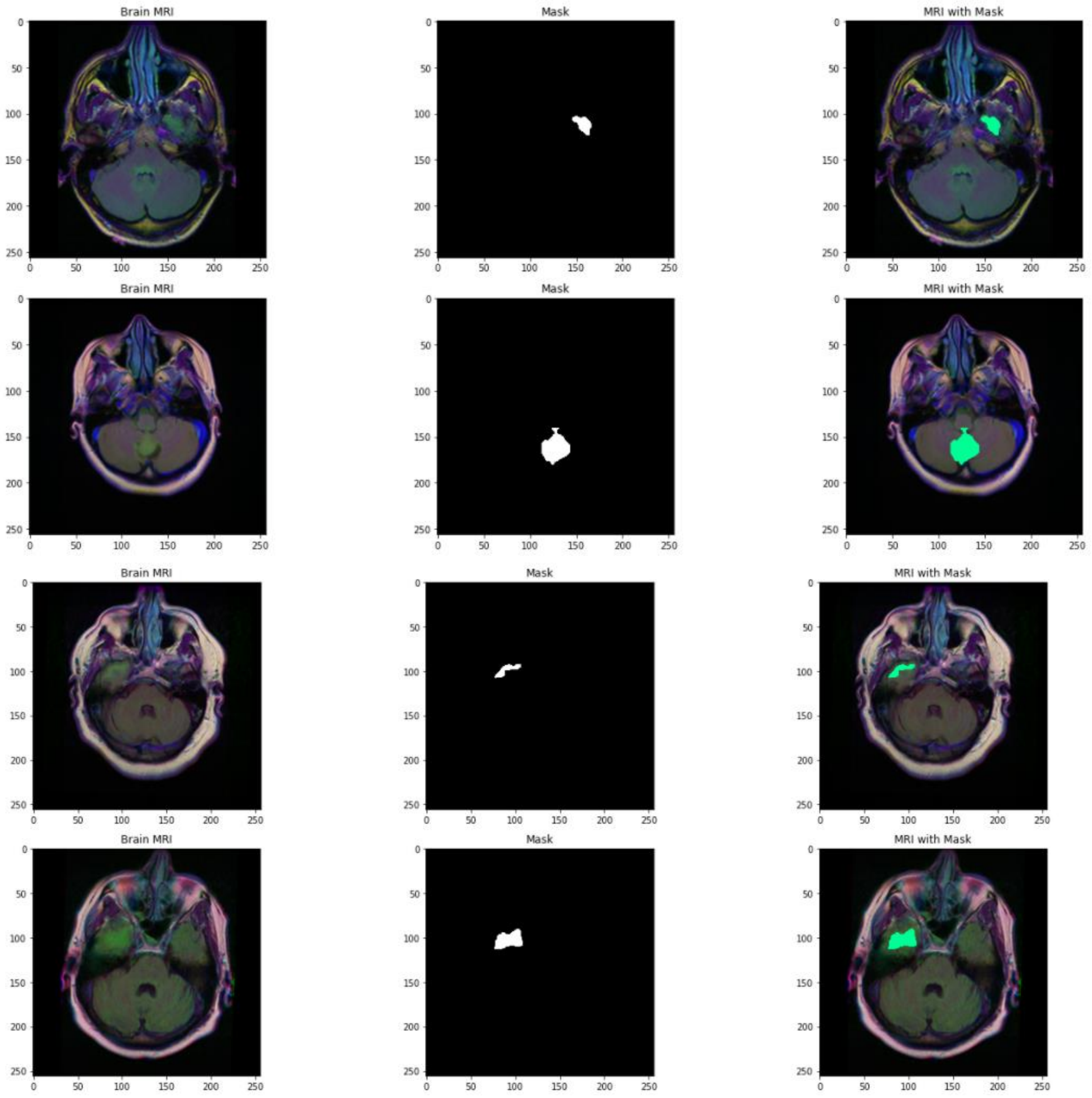


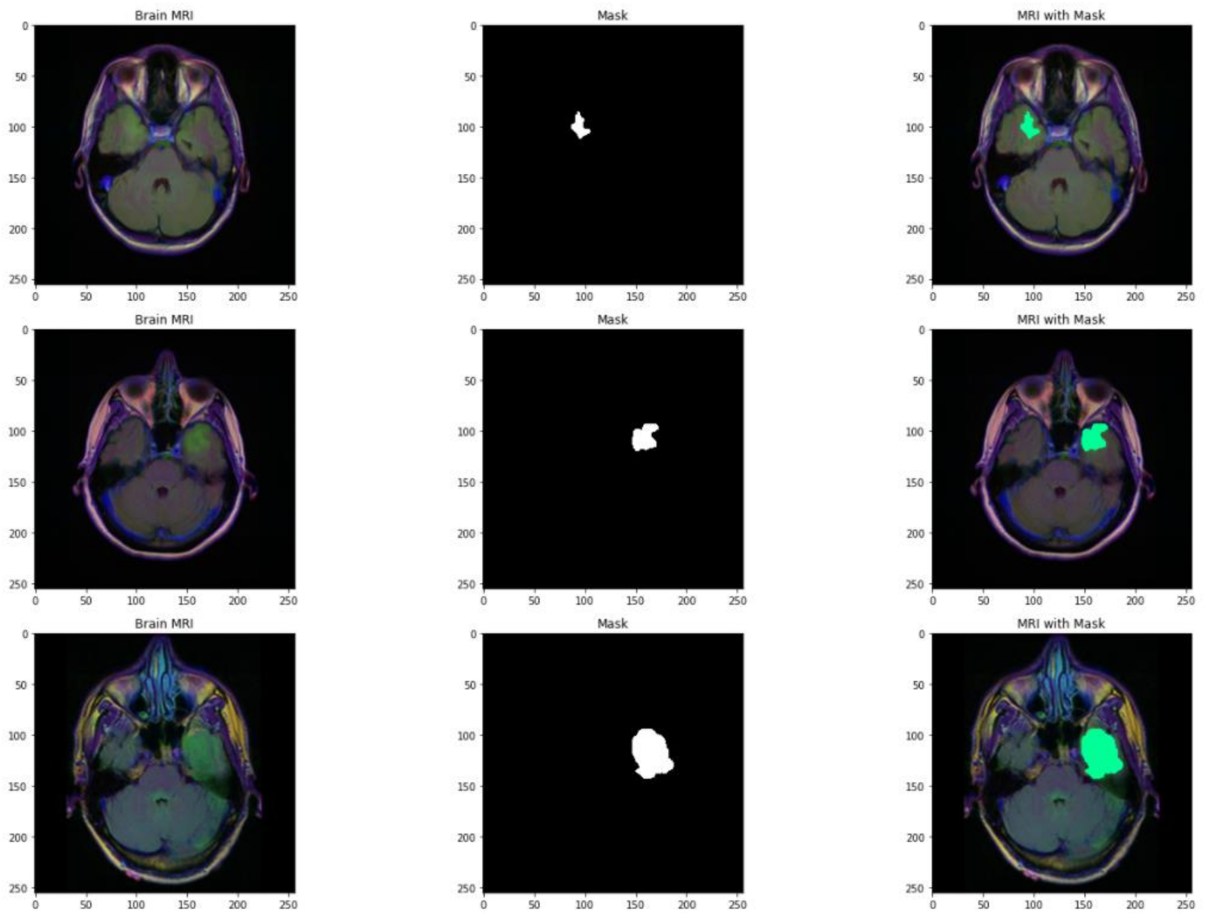

Brain MRI
Mask - 0
Brain MRI
Mask - 0
0
50
100
150
200
250

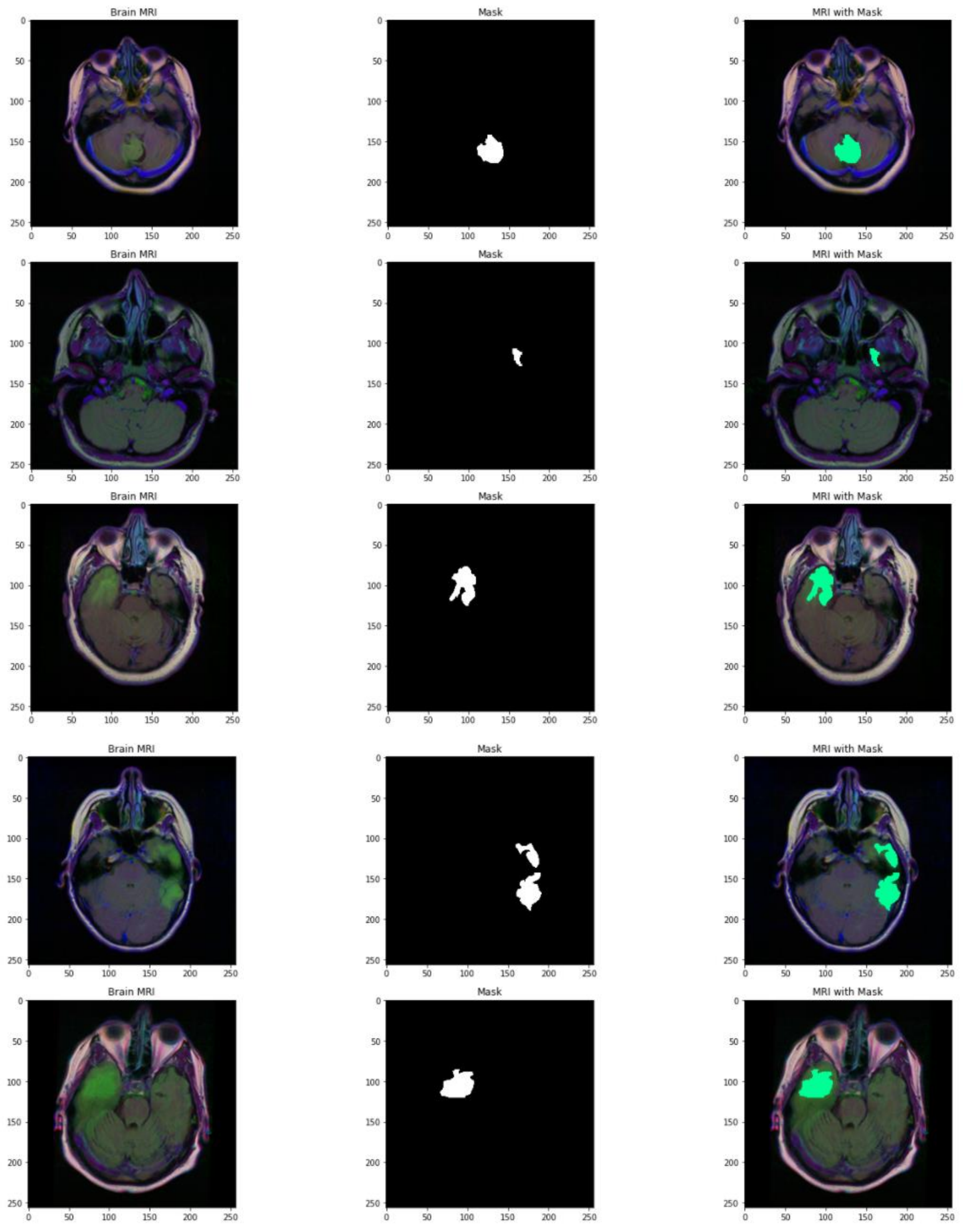


In [13]:

```python
count = 0
i = 0
fig,axs = plt.subplots(12,3, figsize=(20,50))
for mask in brain_df['mask']:
    if (mask==1):
        img = io.imread(brain_df.image_path[i])
        axs[count][0].title.set_text("Brain MRI")
        axs[count][0].imshow(img)

        mask = io.imread(brain_df.mask_path[i])
        axs[count][1].title.set_text("Mask")
        axs[count][1].imshow(mask, cmap='gray')

        img[mask==255] = (0,255,150)  # change pixel color at the position of mask
        axs[count][2].title.set_text("MRI with Mask")
        axs[count][2].imshow(img)
        count +=1
    i += 1
    if (count==12):
        break

fig.tight_layout()
```


Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask


Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask


Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask

## MODULE 2 - Data Preprocessing

### Performing train and test split

In [14]:

```
brain_df_train = brain_df.drop(columns=['patient_id'])
brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))
brain_df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3929 entries, 0 to 3928
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  3929 non-null   object
 1   mask_path   3929 non-null   object
 2   mask        3929 non-null   object
dtypes: object(3)
memory usage: 92.2+ KB
```

In [16]:

```
brain_df_mask = brain_df[brain_df['mask'] == 1]
brain_df_mask.shape
```

Out[16]:

```
(1373, 4)
```

In [17]:

```
brain_df_mask.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1373 entries, 537 to 3802
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   patient_id  1373 non-null   object
 1   image_path  1373 non-null   object
 2   mask_path   1373 non-null   object
 3   mask        1373 non-null   int64 
dtypes: int64(1), object(3)
memory usage: 53.6+ KB
```

In [18]:

```python
# Creating test, train and val sets
X_train, X_val = train_test_split(brain_df_mask, test_size=0.15)
X_test, X_val = train_test_split(X_val, test_size=0.5)
print("Train size is {}, valid size is {} & test size is {}".format(len(X_train), len(X_val), l
en(X_test)))

train_ids = list(X_train.image_path)
train_mask = list(X_train.mask_path)

val_ids = list(X_val.image_path)
val_mask= list(X_val.mask_path)
```

```
Train size is 1167, valid size is 103 & test size is 103
```

## Resizing and standardizing images

In [19]:

```python
class DataGenerator(tf.keras.utils.Sequence):
  def __init__(self, ids , mask, image_dir = './', batch_size = 16, img_h = 256, img_w = 256, s
huffle = True):

    self.ids = ids
    self.mask = mask
    self.image_dir = image_dir
    self.batch_size = batch_size
    self.img_h = img_h
    self.img_w = img_w
    self.shuffle = shuffle
    self.on_epoch_end()

  def __len__(self):
    'Get the number of batches per epoch'
    return int(np.floor(len(self.ids)) / self.batch_size)
  def __getitem__(self, index):
    'Generate a batch of data'

    #generate index of batch_size length
    indexes = self.indexes[index* self.batch_size : (index+1) * self.batch_size]

    #get the ImageId corresponding to the indexes created above based on batch size
    list_ids = [self.ids[i] for i in indexes]

    #get the MaskId corresponding to the indexes created above based on batch size
    list_mask = [self.mask[i] for i in indexes]


    #generate data for the X(features) and y(label)
    X, y = self.__data_generation(list_ids, list_mask)

    #returning the data
    return X, y
```

```python
  def on_epoch_end(self):
    'Used for updating the indices after each epoch, once at the beginning as well as at the en
d of each epoch'

    #getting the array of indices based on the input dataframe
    self.indexes = np.arange(len(self.ids))

    #if shuffle is true, shuffle the indices
    if self.shuffle:
      np.random.shuffle(self.indexes)

  def __data_generation(self, list_ids, list_mask):
    'generate the data corresponding the indexes in a given batch of images'

    # create empty arrays of shape (batch_size,height,width,depth)
    #Depth is 3 for input and depth is taken as 1 for output becasue mask consist only of 1 chann
el.
    X = np.empty((self.batch_size, self.img_h, self.img_w, 3))
    y = np.empty((self.batch_size, self.img_h, self.img_w, 1))

    #iterate through the dataframe rows, whose size is equal to the batch_size
    for i in range(len(list_ids)):
      #path of the image
      img_path = str(list_ids[i])

      #mask path
      mask_path = str(list_mask[i])
```

```
        #reading the original image and the corresponding mask image
        img = io.imread(img_path)
        mask = io.imread(mask_path)

        #resizing and coverting them to array of type float64
        img = cv2.resize(img,(self.img_h,self.img_w))
        img = np.array(img, dtype = np.float64)

        mask = cv2.resize(mask,(self.img_h,self.img_w))
        mask = np.array(mask, dtype = np.float64)

        #standardising
        img -= img.mean()
        img /= img.std()

        mask -= mask.mean()
        mask /= mask.std()

        #Adding image to the empty array
        X[i,] = img

        #expanding the dimnesion of the image from (256,256) to (256,256,1)
        y[i,] = np.expand_dims(mask, axis = 2)
```

```
    y = (y > 0).astype(int)
    X = (X > 0).astype(int)

    return X, y

train_data = DataGenerator(train_ids, train_mask)
val_data = DataGenerator(val_ids, val_mask)
```

# MODULE 3 - Model

## Dice Coefficient Metric

In [20]:

```
#Defining dice coefficiat metric
def dice_coef(y_true, y_pred):
    """
    Parameters
    ----------
    y_true : numpy array of actual masks
    y_pred : numpy array of predicted masks
    """
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + K.epsilon()) / (K.sum(y_true_f) + K.sum(y_pred_f) + K.epsilo
n())
```

## Defining contraction path

In [21]:

```
#U-NET MODEL
inputs = tf.keras.layers.Input((256,256,3))


#NORMALIZING THE INPUT
s = tf.keras.layers.Lambda(lambda x: x/255)(inputs)

#CONTRACTION PATH
#Convolution-1
c1 = tf.keras.layers.Conv2D(16, (3,3), activation='relu', kernel_initializer='he_normal', pa
dding='same')(s)
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(16, (3,3), activation='relu', kernel_initializer='he_normal', pa
dding='same')(c1)
#MaxPool-1
p1 = tf.keras.layers.MaxPool2D((2,2))(c1)
```

```
#Convolution-2
c2 = tf.keras.layers.Conv2D(32, (3,3), activation='relu', kernel_initializer='he_normal', pa
dding='same')(p1)
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(32, (3,3), activation='relu', kernel_initializer='he_normal', pa
dding='same')(c2)
#MaxPool-2
p2 = tf.keras.layers.MaxPool2D((2,2))(c2)


#Convolution-3
c3 = tf.keras.layers.Conv2D(64, (3,3), activation='relu', kernel_initializer='he_normal', pa
dding='same')(p2)
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(64, (3,3), activation='relu', kernel_initializer='he_normal', pa
dding='same')(c3)
#MaxPool-3
p3 = tf.keras.layers.MaxPool2D((2,2))(c3)
```

```
#Convolution-4
c4 = tf.keras.layers.Conv2D(128, (3,3), activation='relu', kernel_initializer='he_normal', p
adding='same')(p3)
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(128, (3,3), activation='relu', kernel_initializer='he_normal', p
adding='same')(c4)
#MaxPool-4
p4 = tf.keras.layers.MaxPool2D((2,2))(c4)


#Convolution-5
c5 = tf.keras.layers.Conv2D(256, (3,3), activation='relu', kernel_initializer='he_normal', p
adding='same')(p4)
c5 = tf.keras.layers.Dropout(0.3)(c5)
c5 = tf.keras.layers.Conv2D(256, (3,3), activation='relu', kernel_initializer='he_normal', p
adding='same')(c5)
```

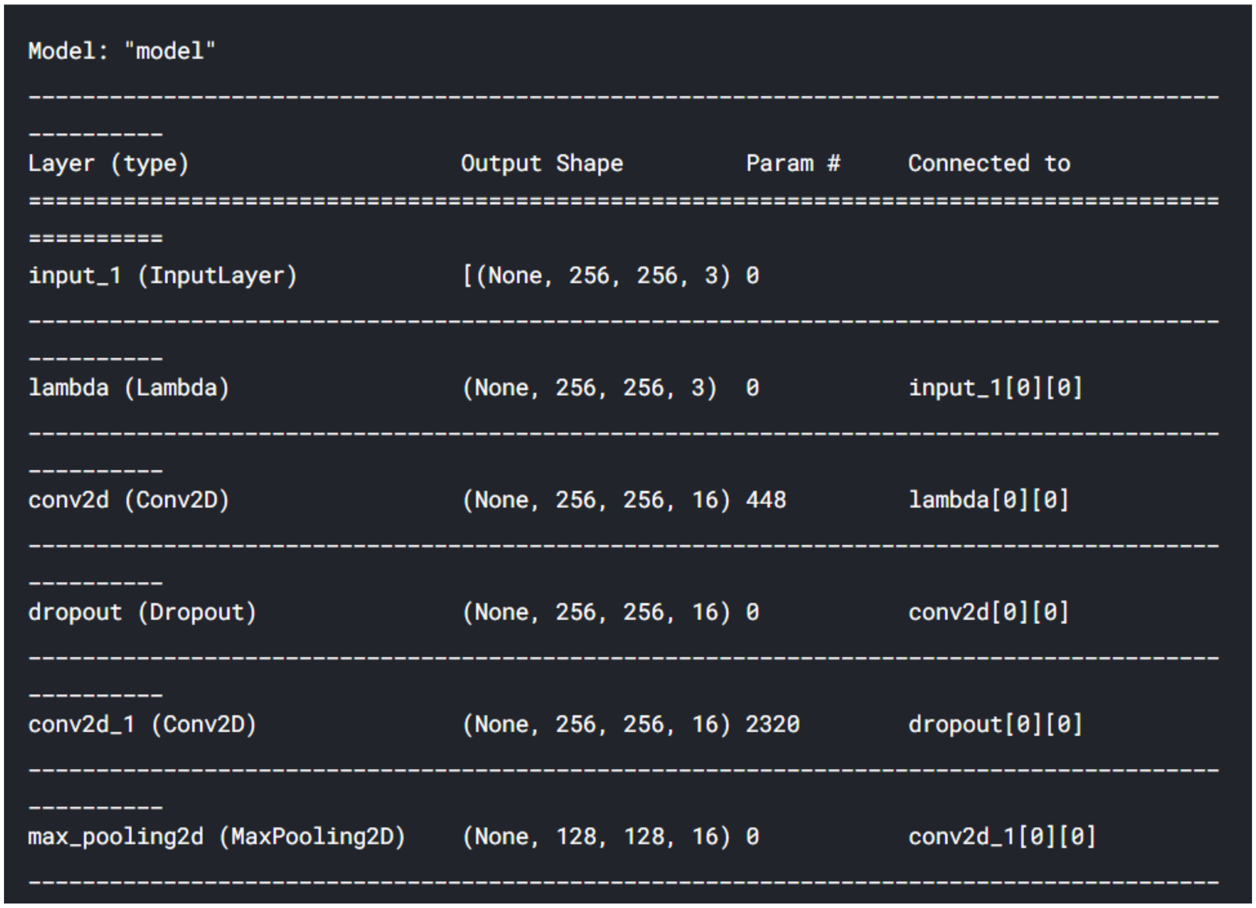
## Defining expansion path

```
#EXPANSION PATH
#ConvolutionTranspose-1 (upsample)
u6 = tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c5)
u6 = tf.keras.layers.concatenate([u6, c4])
#Convolution-6
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(c6)


#ConvolutionTranspose-2 (upsample)
u7 = tf.keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c6)
u7 = tf.keras.layers.concatenate([u7, c3])
#Convolution-7
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', p
adding='same')(u7)
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', p
adding='same')(c7)
```

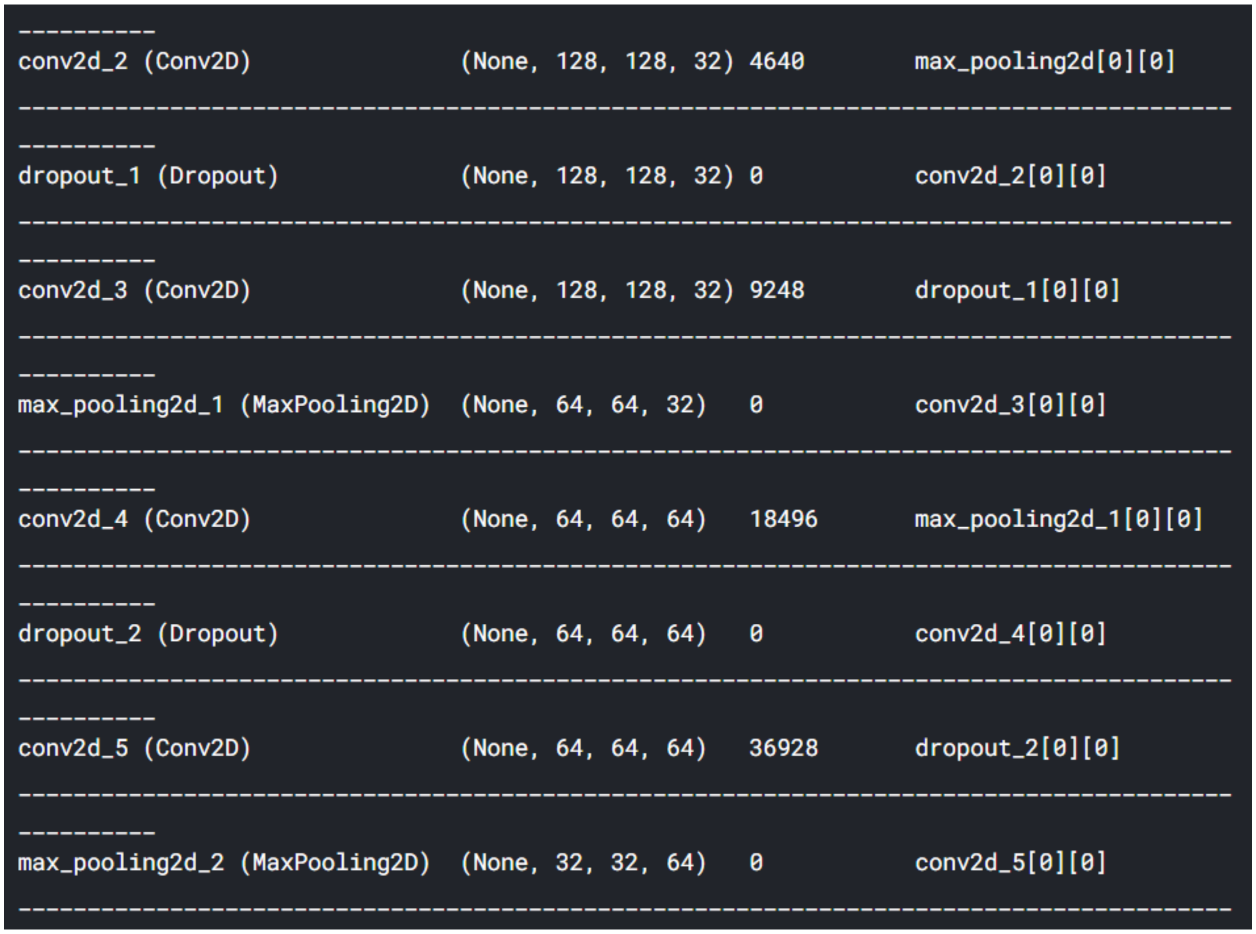
```
#ConvolutionTranspose-3 (upsample)
u8 = tf.keras.layers.Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c7)
u8 = tf.keras.layers.concatenate([u8, c2])
#Convolution-8
c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', p
adding='same')(u8)
c8 = tf.keras.layers.Dropout(0.1)(c8)
c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', p
adding='same')(c8)


#ConvolutionTranspose-4 (upsample)
u9 = tf.keras.layers.Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c8)
u9 = tf.keras.layers.concatenate([u9, c1], axis=3)
#Convolution-9
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', p
adding='same')(u9)
c9 = tf.keras.layers.Dropout(0.1)(c9)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', p
adding='same')(c9)
```

## Adding Optimizer and loss metrics

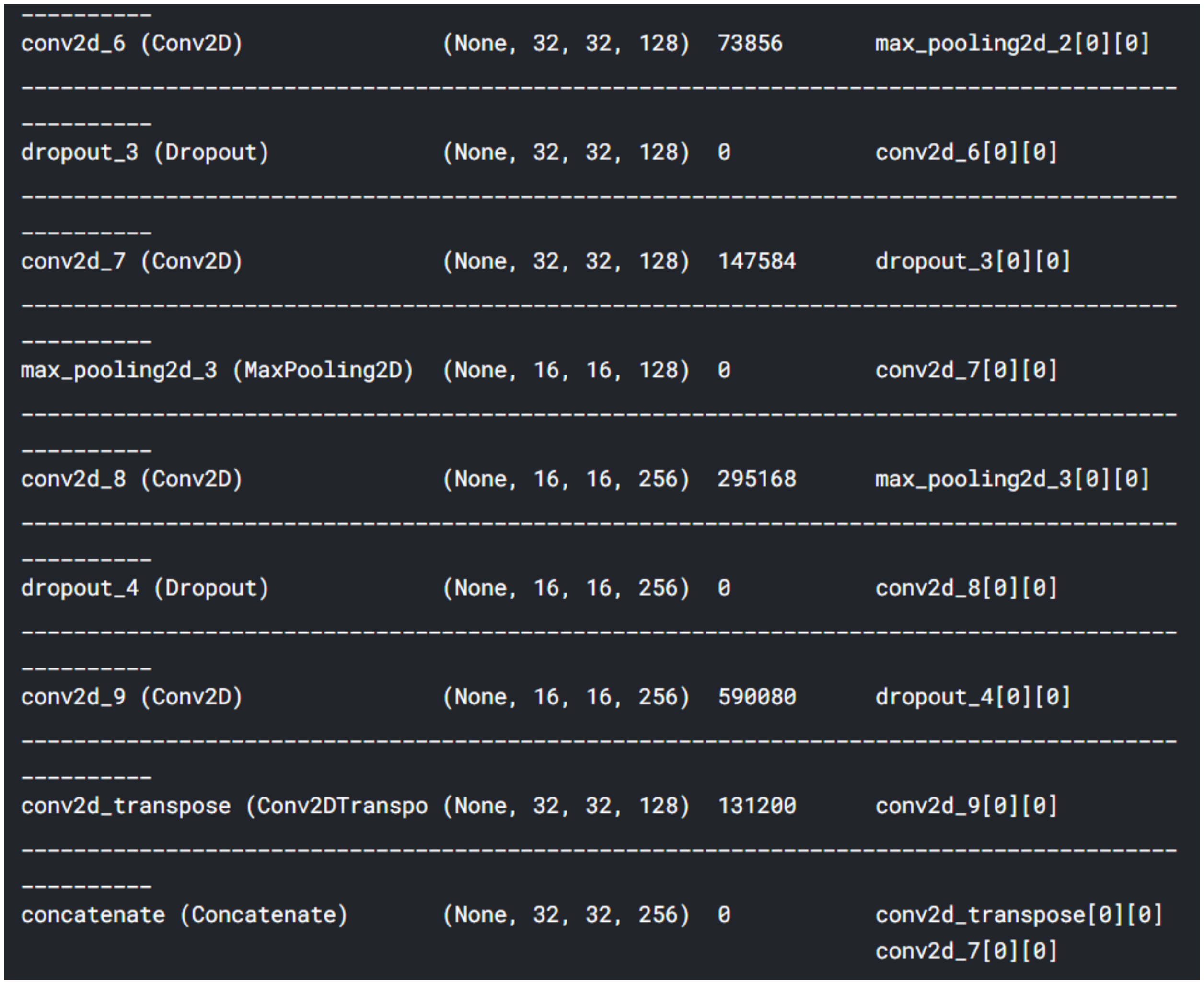
```
outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
model = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[dice_coef,'accuracy'])
model.summary()
```

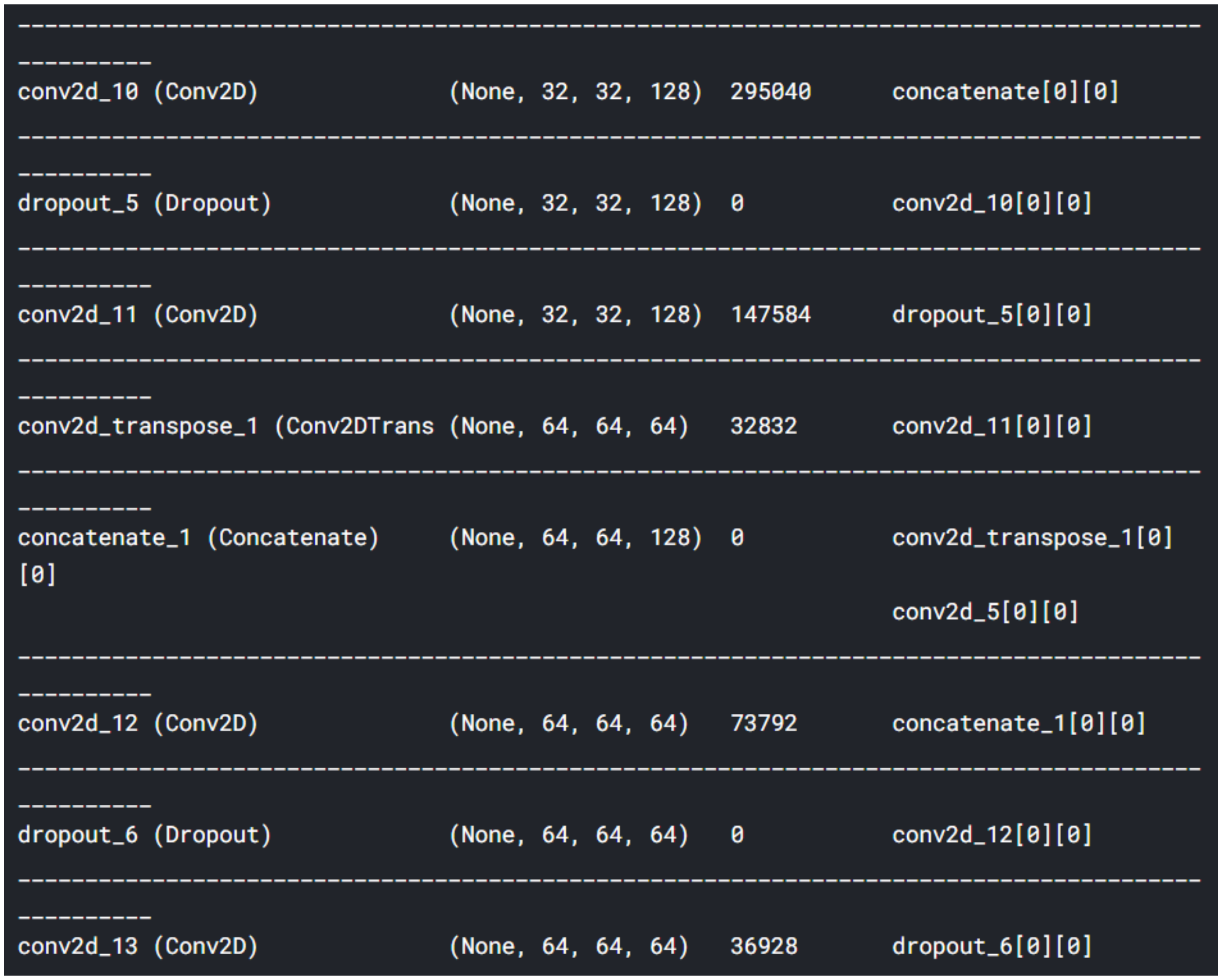


```
Model: "model"
________________________________________________________________________________________
__________
Layer (type)                    Output Shape         Param #     Connected to
========================================================================================
==========
input_1 (InputLayer)            [(None, 256, 256, 3) 0

________________________________________________________________________________________
__________
lambda (Lambda)                 (None, 256, 256, 3)  0           input_1[0][0]

________________________________________________________________________________________
__________
conv2d (Conv2D)                 (None, 256, 256, 16) 448         lambda[0][0]

________________________________________________________________________________________
__________
dropout (Dropout)               (None, 256, 256, 16) 0           conv2d[0][0]

________________________________________________________________________________________
__________
conv2d_1 (Conv2D)               (None, 256, 256, 16) 2320        dropout[0][0]

________________________________________________________________________________________
__________
max_pooling2d (MaxPooling2D)    (None, 128, 128, 16) 0           conv2d_1[0][0]

________________________________________________________________________________________
```

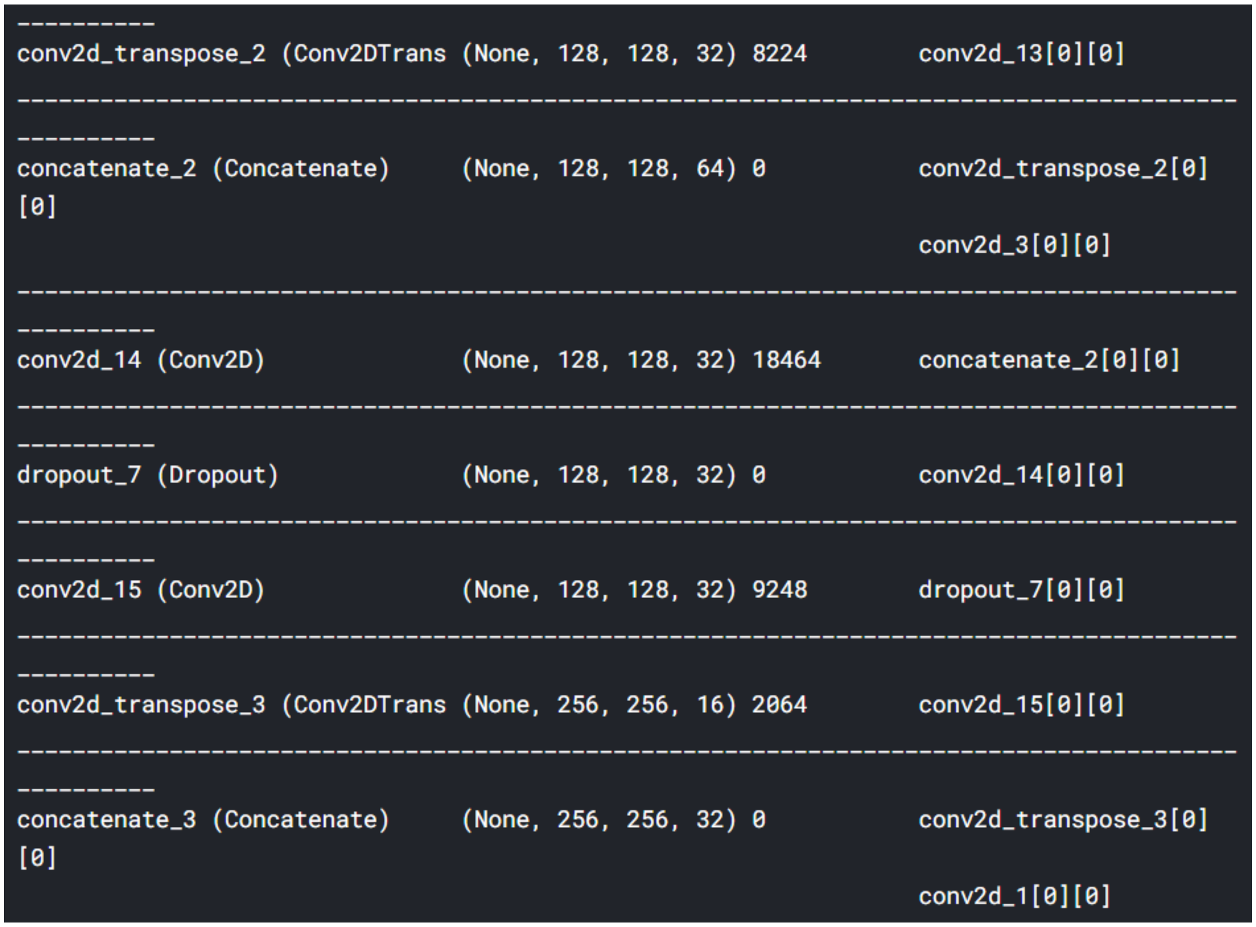


```
----------
conv2d_2 (Conv2D)             (None, 128, 128, 32) 4640       max_pooling2d[0][0]
----------------------------------------------------------------------------------------
----------
dropout_1 (Dropout)           (None, 128, 128, 32) 0          conv2d_2[0][0]
----------------------------------------------------------------------------------------
----------
conv2d_3 (Conv2D)             (None, 128, 128, 32) 9248       dropout_1[0][0]
----------------------------------------------------------------------------------------
----------
max_pooling2d_1 (MaxPooling2D)  (None, 64, 64, 32)   0          conv2d_3[0][0]
----------------------------------------------------------------------------------------
----------
conv2d_4 (Conv2D)             (None, 64, 64, 64)   18496      max_pooling2d_1[0][0]
----------------------------------------------------------------------------------------
----------
dropout_2 (Dropout)           (None, 64, 64, 64)   0          conv2d_4[0][0]
----------------------------------------------------------------------------------------
----------
conv2d_5 (Conv2D)             (None, 64, 64, 64)   36928      dropout_2[0][0]
----------------------------------------------------------------------------------------
----------
max_pooling2d_2 (MaxPooling2D)  (None, 32, 32, 64)   0          conv2d_5[0][0]
----------------------------------------------------------------------------------------
```

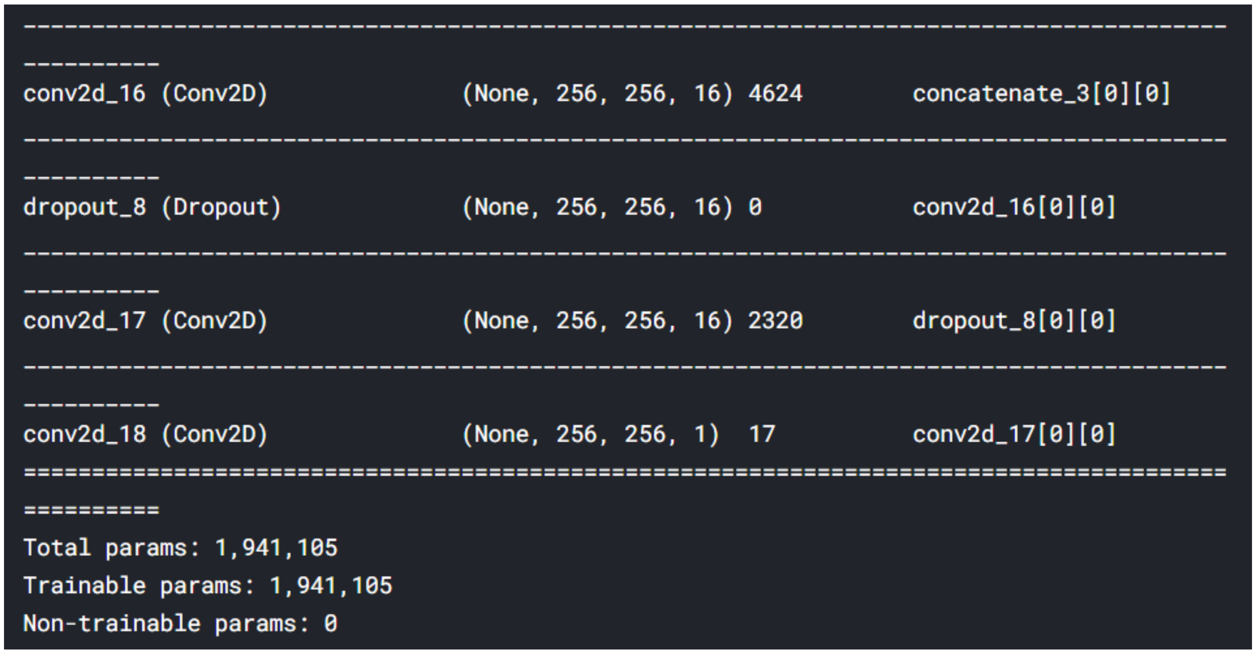

```
----------
conv2d_6 (Conv2D)                (None, 32, 32, 128)  73856       max_pooling2d_2[0][0]
__________________________________________________________________________________________
----------
dropout_3 (Dropout)              (None, 32, 32, 128)  0           conv2d_6[0][0]
__________________________________________________________________________________________
----------
conv2d_7 (Conv2D)                (None, 32, 32, 128)  147584      dropout_3[0][0]
__________________________________________________________________________________________
----------
max_pooling2d_3 (MaxPooling2D)   (None, 16, 16, 128)  0           conv2d_7[0][0]
__________________________________________________________________________________________
----------
conv2d_8 (Conv2D)                (None, 16, 16, 256)  295168      max_pooling2d_3[0][0]
__________________________________________________________________________________________
----------
dropout_4 (Dropout)              (None, 16, 16, 256)  0           conv2d_8[0][0]
__________________________________________________________________________________________
----------
conv2d_9 (Conv2D)                (None, 16, 16, 256)  590080      dropout_4[0][0]
__________________________________________________________________________________________
----------
conv2d_transpose (Conv2DTranspo (None, 32, 32, 128)  131200      conv2d_9[0][0]
__________________________________________________________________________________________
----------
concatenate (Concatenate)        (None, 32, 32, 256)  0           conv2d_transpose[0][0]
                                                                  conv2d_7[0][0]
```



```
----------------------------------------------------------------------------------------
----------
conv2d_10 (Conv2D)              (None, 32, 32, 128)  295040      concatenate[0][0]
----------------------------------------------------------------------------------------
----------
dropout_5 (Dropout)             (None, 32, 32, 128)  0           conv2d_10[0][0]
----------------------------------------------------------------------------------------
----------
conv2d_11 (Conv2D)              (None, 32, 32, 128)  147584      dropout_5[0][0]
----------------------------------------------------------------------------------------
----------
conv2d_transpose_1 (Conv2DTrans (None, 64, 64, 64)   32832       conv2d_11[0][0]
----------------------------------------------------------------------------------------
----------
concatenate_1 (Concatenate)     (None, 64, 64, 128)  0           conv2d_transpose_1[0]
[0]

                                                                 conv2d_5[0][0]
----------------------------------------------------------------------------------------
----------
conv2d_12 (Conv2D)              (None, 64, 64, 64)   73792       concatenate_1[0][0]
----------------------------------------------------------------------------------------
----------
dropout_6 (Dropout)             (None, 64, 64, 64)   0           conv2d_12[0][0]
----------------------------------------------------------------------------------------
----------
conv2d_13 (Conv2D)              (None, 64, 64, 64)   36928       dropout_6[0][0]
```



```
----------
conv2d_transpose_2 (Conv2DTrans (None, 128, 128, 32) 8224        conv2d_13[0][0]
__________________________________________________________________________________________________
----------
concatenate_2 (Concatenate)     (None, 128, 128, 64) 0           conv2d_transpose_2[0]
[0]
                                                                 conv2d_3[0][0]
__________________________________________________________________________________________________
----------
conv2d_14 (Conv2D)              (None, 128, 128, 32) 18464       concatenate_2[0][0]
__________________________________________________________________________________________________
----------
dropout_7 (Dropout)             (None, 128, 128, 32) 0           conv2d_14[0][0]
__________________________________________________________________________________________________
----------
conv2d_15 (Conv2D)              (None, 128, 128, 32) 9248        dropout_7[0][0]
__________________________________________________________________________________________________
----------
conv2d_transpose_3 (Conv2DTrans (None, 256, 256, 16) 2064        conv2d_15[0][0]
__________________________________________________________________________________________________
----------
concatenate_3 (Concatenate)     (None, 256, 256, 32) 0           conv2d_transpose_3[0]
[0]
                                                                 conv2d_1[0][0]
```



```
__________________________________________________________________________________________________
----------
conv2d_16 (Conv2D)              (None, 256, 256, 16) 4624        concatenate_3[0][0]
__________________________________________________________________________________________________
----------
dropout_8 (Dropout)             (None, 256, 256, 16) 0           conv2d_16[0][0]
__________________________________________________________________________________________________
----------
conv2d_17 (Conv2D)              (None, 256, 256, 16) 2320        dropout_8[0][0]
__________________________________________________________________________________________________
----------
conv2d_18 (Conv2D)              (None, 256, 256, 1)  17          conv2d_17[0][0]
==================================================================================================
==========
Total params: 1,941,105
Trainable params: 1,941,105
Non-trainable params: 0
```

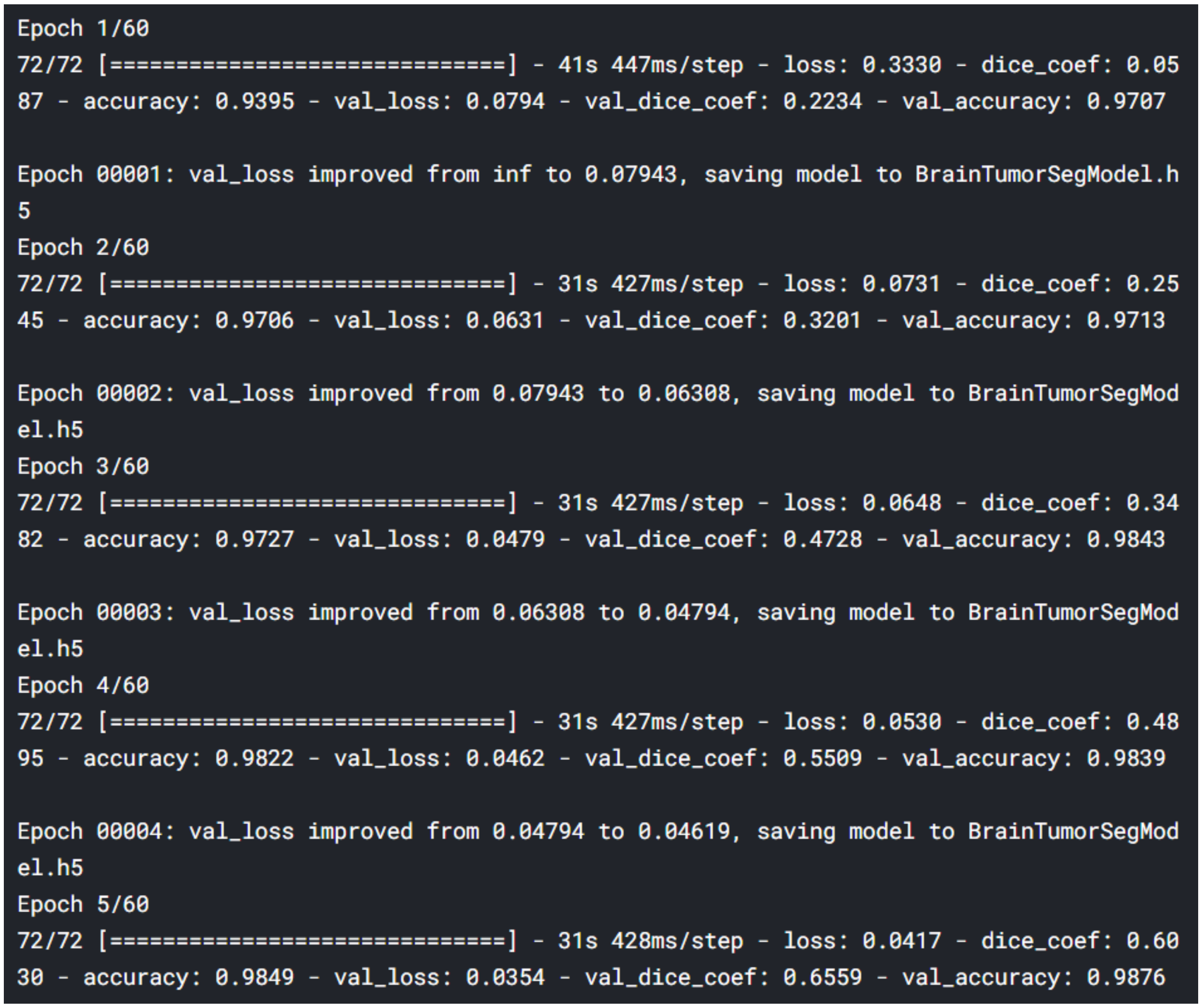
## MODULE 4 - Training

### Defining callbacks and checkpoints

In [22]:
```
#Defining callbacks for the model
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode="mi
n"),
    tf.keras.callbacks.TensorBoard(log_dir='logs'),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                              mode='min',
                              verbose=1,
                              patience=5,
                              min_delta=0.0001,
                              factor=0.2
                             ),
    tf.keras.callbacks.ModelCheckpoint('BrainTumorSegModel.h5', monitor='val_loss', verbose=
1, save_best_only=True)
]
```

### Fitting model on data

In [23]:
```
#Fitting model on data
h = model.fit(train_data,
                  epochs = 60,
                  validation_data = val_data,
                  callbacks=callbacks
                 )
```



```
Epoch 1/60
72/72 [==============================] - 41s 447ms/step - loss: 0.3330 - dice_coef: 0.05
87 - accuracy: 0.9395 - val_loss: 0.0794 - val_dice_coef: 0.2234 - val_accuracy: 0.9707

Epoch 00001: val_loss improved from inf to 0.07943, saving model to BrainTumorSegModel.h
5
Epoch 2/60
72/72 [==============================] - 31s 427ms/step - loss: 0.0731 - dice_coef: 0.25
45 - accuracy: 0.9706 - val_loss: 0.0631 - val_dice_coef: 0.3201 - val_accuracy: 0.9713

Epoch 00002: val_loss improved from 0.07943 to 0.06308, saving model to BrainTumorSegMod
el.h5
Epoch 3/60
72/72 [==============================] - 31s 427ms/step - loss: 0.0648 - dice_coef: 0.34
82 - accuracy: 0.9727 - val_loss: 0.0479 - val_dice_coef: 0.4728 - val_accuracy: 0.9843

Epoch 00003: val_loss improved from 0.06308 to 0.04794, saving model to BrainTumorSegMod
el.h5
Epoch 4/60
72/72 [==============================] - 31s 427ms/step - loss: 0.0530 - dice_coef: 0.48
95 - accuracy: 0.9822 - val_loss: 0.0462 - val_dice_coef: 0.5509 - val_accuracy: 0.9839

Epoch 00004: val_loss improved from 0.04794 to 0.04619, saving model to BrainTumorSegMod
el.h5
Epoch 5/60
72/72 [==============================] - 31s 428ms/step - loss: 0.0417 - dice_coef: 0.60
30 - accuracy: 0.9849 - val_loss: 0.0354 - val_dice_coef: 0.6559 - val_accuracy: 0.9876
```

```
Epoch 00047: val_loss did not improve from 0.01482
Epoch 48/60
72/72 [==============================] - 31s 428ms/step - loss: 0.0096 - dice_coef: 0.89
90 - accuracy: 0.9960 - val_loss: 0.0156 - val_dice_coef: 0.8707 - val_accuracy: 0.9941

Epoch 00048: val_loss did not improve from 0.01482
Epoch 49/60
72/72 [==============================] - 31s 428ms/step - loss: 0.0095 - dice_coef: 0.89
95 - accuracy: 0.9960 - val_loss: 0.0165 - val_dice_coef: 0.8641 - val_accuracy: 0.9938

Epoch 00049: val_loss did not improve from 0.01482
Epoch 50/60
72/72 [==============================] - 31s 427ms/step - loss: 0.0092 - dice_coef: 0.89
98 - accuracy: 0.9962 - val_loss: 0.0163 - val_dice_coef: 0.8683 - val_accuracy: 0.9938

Epoch 00050: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.

Epoch 00050: val_loss did not improve from 0.01482
Epoch 00050: early stopping
```

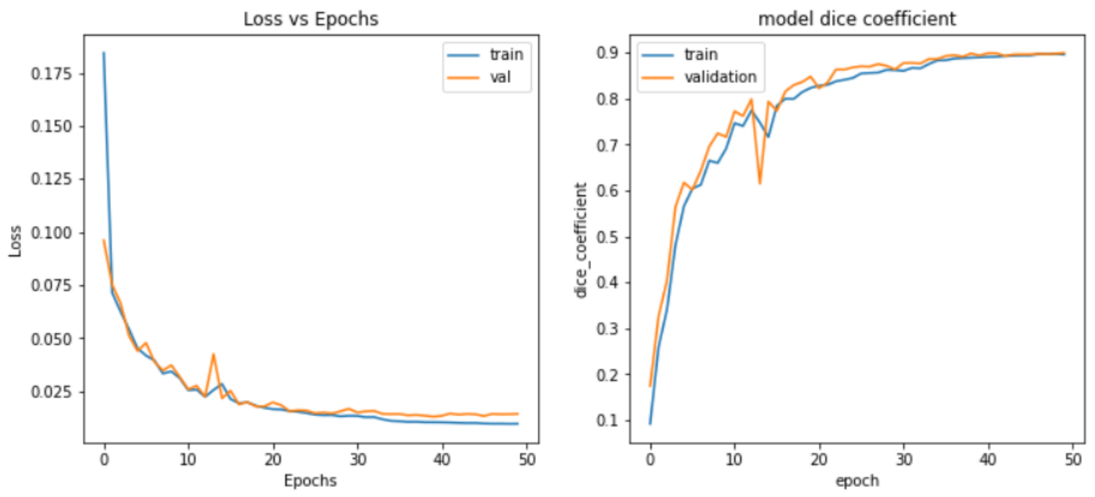
In [24]:
```
h.history.keys()
```

Out[24]:
```
dict_keys(['loss', 'dice_coef', 'accuracy', 'val_loss', 'val_dice_coef', 'val_accuracy',
'lr'])
```

## Plotting metrics

In [25]:

```
#Plotting metrics
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(h.history['loss']);
plt.plot(h.history['val_loss']);
plt.title("Loss vs Epochs");
plt.ylabel("Loss");
plt.xlabel("Epochs");
plt.legend(['train', 'val']);

plt.subplot(1,2,2)
plt.plot(h.history['dice_coef'])
plt.plot(h.history['val_dice_coef'])
plt.title('model dice coefficient')
plt.ylabel('dice_coefficient')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



## MODULE 5 - Testing

In [26]:

```
#Displaying accuracy obtained
test_ids = list(X_test.image_path)
test_mask = list(X_test.mask_path)
test_data = DataGenerator(test_ids, test_mask)
_, accuracy, dice_coefficient = model.evaluate(test_data)
print("Segmentation accuracy is {:.2f}%".format(accuracy*100))
print("Segmentation dice_coefficient is {:.2f}%".format(dice_coefficient*100))
```

```
6/6 [==============================] - 1s 165ms/step - loss: 0.0159 - dice_coef: 0.8738 - ac
curacy: 0.9940
Segmentation accuracy is 87.38%
Segmentation dice_coefficient is 99.40%
```

## Perform predictons

In [27]:

```
def prediction(test, model_seg):
    '''
    Predcition function which takes dataframe containing ImageID as Input prediction on the im
age
    '''
    # empty list to store results
    mask, image_id, has_mask = [], [], []

    #itetrating through each image in test data
    for i in test.image_path:
        #Creating a empty array of shape 1,256,256,1
        X = np.empty((1,256,256,3))
        # read the image
        img = io.imread(i)
        #resizing the image and coverting them to array of type float64
        img = cv2.resize(img, (256,256))
        img = np.array(img, dtype=np.float64)

        # standardising the image
        img -= img.mean()
        img /= img.std()
        #converting the shape of image from 256,256,3 to 1,256,256,3
        X[0,] = img
        #make prediction of mask
        predict = model_seg.predict(X)

        # if sum of predicted mask is 0 then there is no tumour
        if predict.round().astype(int).sum()==0:
            image_id.append(i)
            has_mask.append(0)
            mask.append('No mask')
        else:
        #if the sum of pixel values are more than 0, then there is tumour
            image_id.append(i)
            has_mask.append(1)
            mask.append(predict)

    return pd.DataFrame({'image_path': image_id,'predicted_mask': mask,'has_mask': has_mas
k})
```

In [28]:

```
# making prediction
df_pred = prediction(X_test, model)
df_pred
```

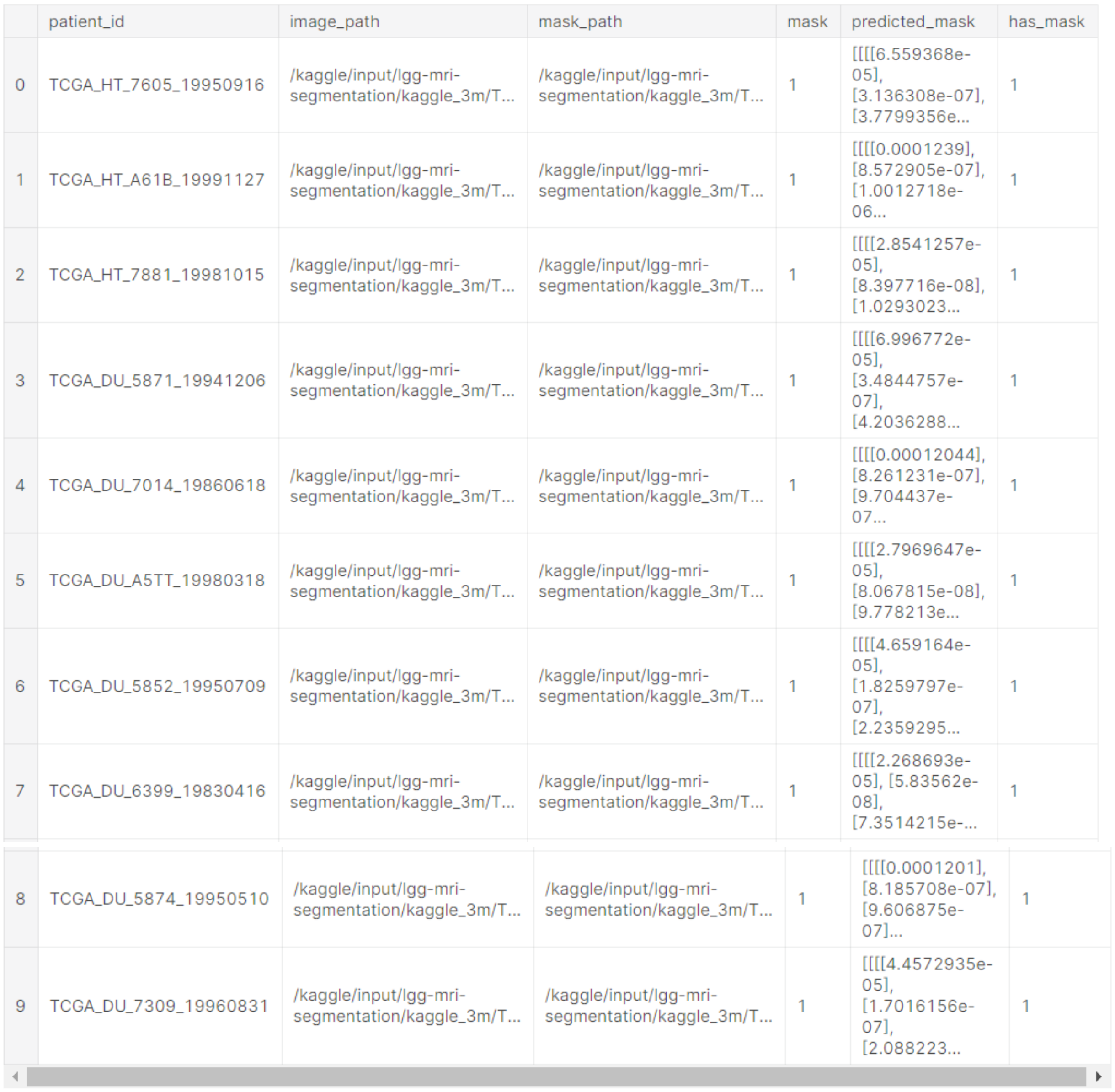
Out[28]:

| | image_path | predicted_mask | has_mask |
|---|---|---|---|
| 0 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | No mask | 0 |
| 1 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.00204809], [0.00014338], [8.759976e-05],... | 1 |
| 2 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.00169989], [0.00010918], [6.5773755e-05]... | 1 |
| 3 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.00194896], [0.00013319], [8.11708e-05], ... | 1 |
| 4 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.00171365], [0.00011058], [6.671945e-05],... | 1 |
| ... | ... | ... | ... |
| 98 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.00233464], [0.00017266], [0.00010627], [... | 1 |
| 99 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.0008736], [4.1779218e-05], [2.383704e-05... | 1 |
| 100 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.0016905], [0.00010822], [6.495547e-05], ... | 1 |
| 101 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.00116705], [6.3322e-05], [3.6960755e-05]... | 1 |
| 102 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | [[[[0.00196377], [0.00013468], [8.2080245e-05]... | 1 |

103 rows × 3 columns

In [29]:

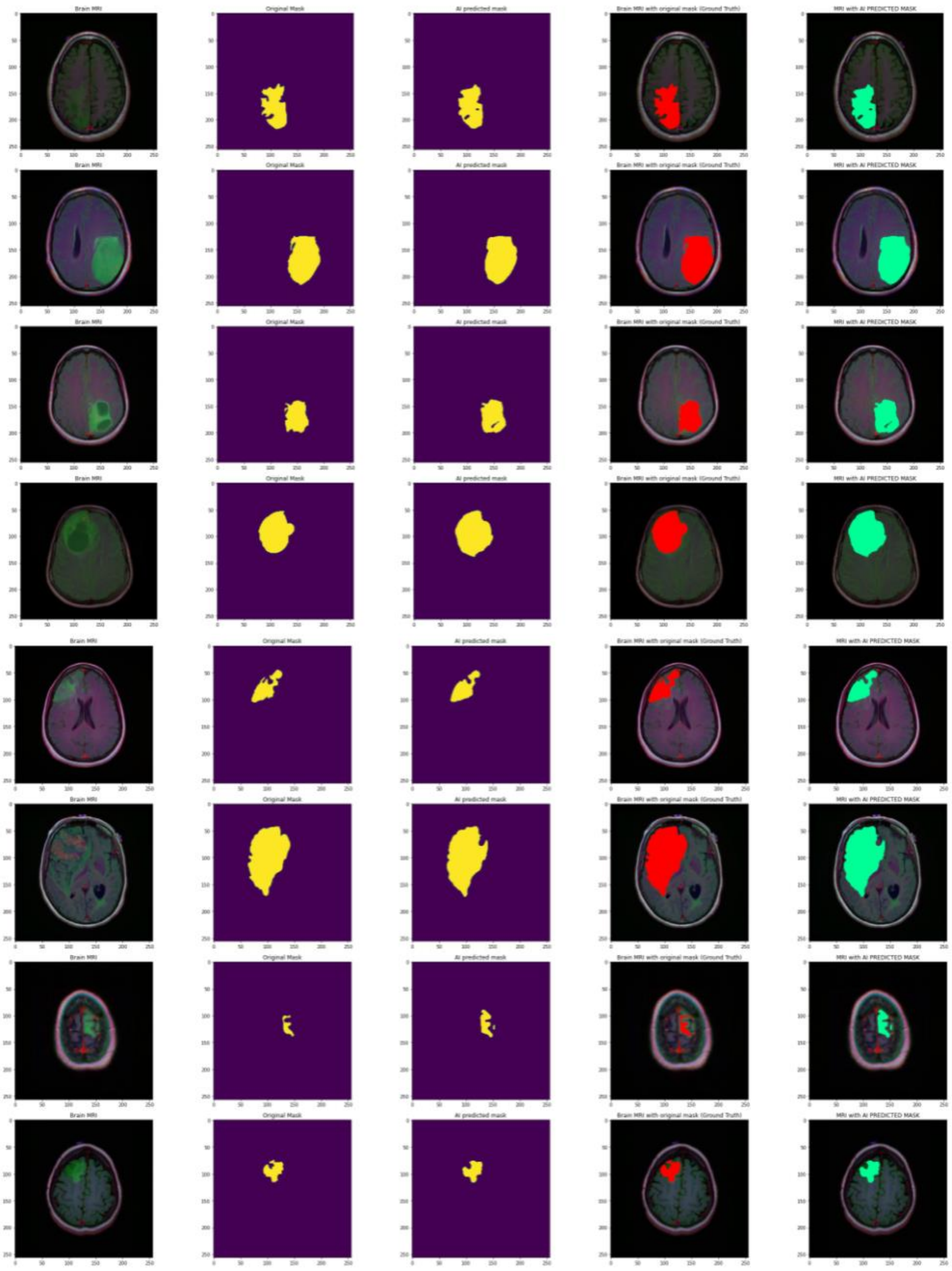
```
# merging original and prediction df
df_pred = X_test.merge(df_pred, on='image_path')
df_pred.head(10)
```



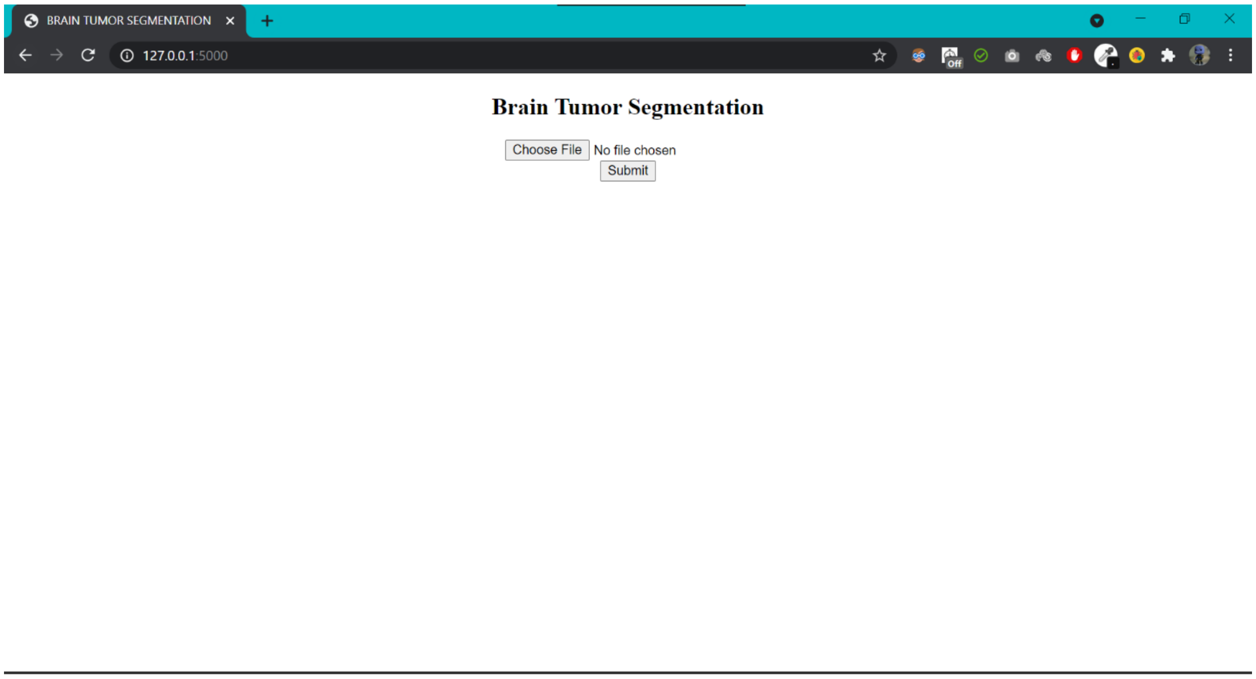
| | patient_id | image_path | mask_path | mask | predicted_mask | has_mask |
|---|---|---|---|---|---|---|
| 0 | TCGA_HT_7605_19950916 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[6.559368e-05], [3.136308e-07], [3.7799356e... | 1 |
| 1 | TCGA_HT_A61B_19991127 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[0.0001239], [8.572905e-07], [1.0012718e-06... | 1 |
| 2 | TCGA_HT_7881_19981015 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[2.8541257e-05], [8.397716e-08], [1.0293023... | 1 |
| 3 | TCGA_DU_5871_19941206 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[6.996772e-05], [3.4844757e-07], [4.2036288... | 1 |
| 4 | TCGA_DU_7014_19860618 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[0.00012044], [8.261231e-07], [9.704437e-07... | 1 |
| 5 | TCGA_DU_A5TT_19980318 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[2.7969647e-05], [8.067815e-08], [9.778213e... | 1 |
| 6 | TCGA_DU_5852_19950709 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[4.659164e-05], [1.8259797e-07], [2.2359295... | 1 |
| 7 | TCGA_DU_6399_19830416 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[2.268693e-05], [5.83562e-08], [7.3514215e-... | 1 |
| 8 | TCGA_DU_5874_19950510 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[0.0001201], [8.185708e-07], [9.606875e-07]... | 1 |
| 9 | TCGA_DU_7309_19960831 | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | /kaggle/input/lgg-mri-segmentation/kaggle_3m/T... | 1 | [[[[4.4572935e-05], [1.7016156e-07], [2.088223... | 1 |

In [30]:

```python
#visualizing prediction
count = 0
fig, axs = plt.subplots(15,5, figsize=(30,70))

for i in range(len(df_pred)):
    if df_pred.has_mask[i]==1 and count<15:
        #read mri images
        img = io.imread(df_pred.image_path[i])
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        axs[count][0].imshow(img)
        axs[count][0].title.set_text('Brain MRI')

        #read original mask
        mask = io.imread(df_pred.mask_path[i])
        axs[count][1].imshow(mask)
        axs[count][1].title.set_text('Original Mask')

        #read predicted mask
        pred = np.array(df_pred.predicted_mask[i]).squeeze().round()
        axs[count][2].imshow(pred)
        axs[count][2].title.set_text('AI predicted mask')
        #overlay original mask with MRI
        img[mask==255] = (255,0,0)
        axs[count][3].imshow(img)
        axs[count][3].title.set_text('Brain MRI with original mask (Ground Truth)')

        #overlay predicted mask and MRI
        img_ = io.imread(df_pred.image_path[i])
        img_ = cv2.cvtColor(img_, cv2.COLOR_BGR2RGB)
        img_[pred==1] = (0,255,150)
        axs[count][4].imshow(img_)
        axs[count][4].title.set_text('MRI with AI PREDICTED MASK')

        count +=1
    if (count==15):
        break

fig.tight_layout()
```


Brain MRI
Original Mask
AI predicted mask
Brain MRI with original mask (Ground Truth)
MRI with AI PREDICTED MASK

## Web Application

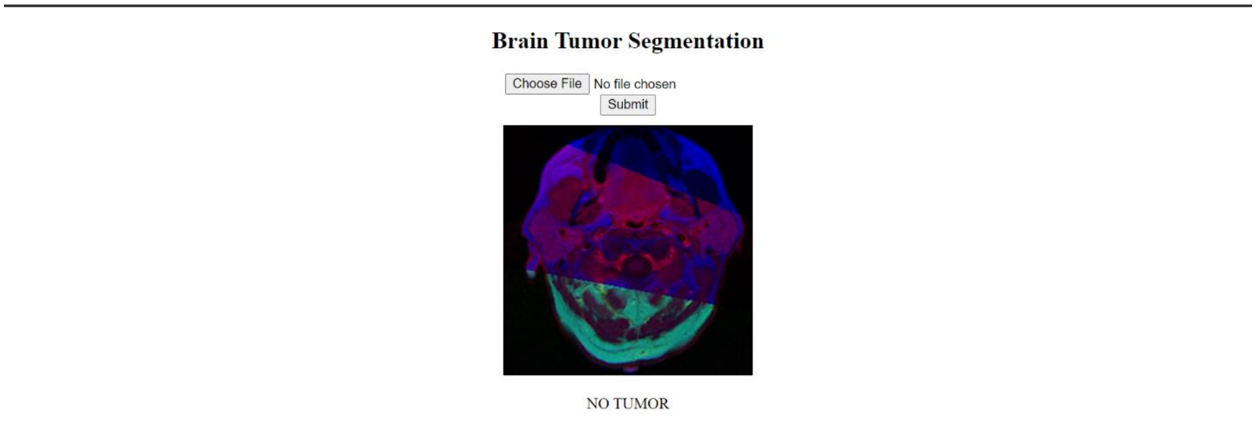

BRAIN TUMOR SEGMENTATION

127.0.0.1:5000

**Brain Tumor Segmentation**

Choose File No file chosen

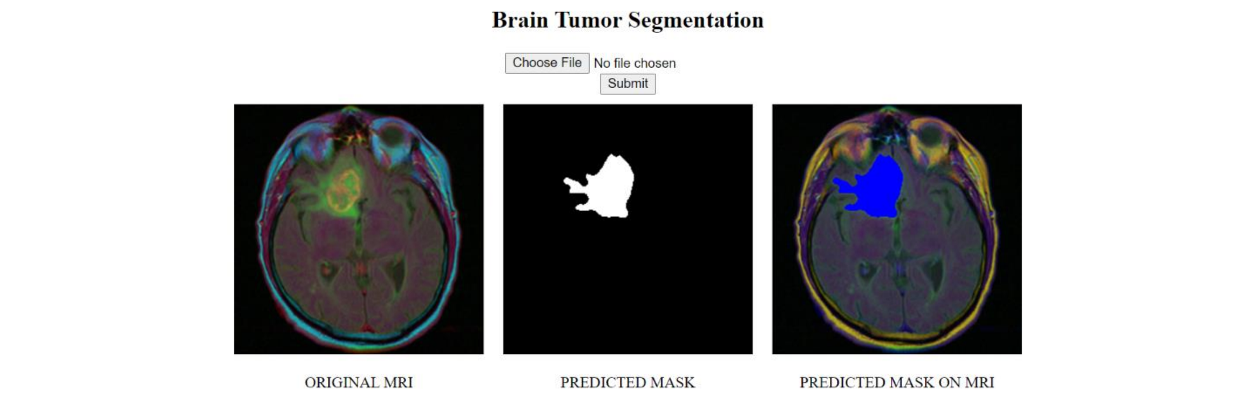
Submit


**Brain Tumor Segmentation**

Choose File No file chosen

Submit

NO TUMOR



## Performance Measure and Evaluation

### Metrics

**Accuracy -** Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. However, in the case of brain MRI segmentation, accuracy alone is not a sufficient evaluation metric because of class imbalance.

**Dice-Coefficient -** A common metric measure of overlap between the predicted and the ground truth. The calculation is 2 * the area of overlap (between the predicted and the ground truth) divided by the total area (of both predict and ground truth combined). This metric will be used together with the Binary cross-entropy as the loss function for training the model.

### Loss

**Binary Cross Entropy -** A common metric and loss function for binary classification for measuring the probability of misclassification. Binary cross entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.

## Conclusion and Results

| | |
|---|---|
| Dice Coefficient (on test data) | 99.4% |
| Accuracy (on test data) | 87.38% |

In this project, we have implemented a U-Net type of architecture, which is based on convolutional neural networks for medical image segmentation. Our proposed network has two parts, 1) an encoder part 2) decoder part of the network. Encoder is the first half of the model in which convolution blocks are followed by maxpool and down sampling to encode the input image into feature representations, thus learning the context of the image. Decoder is the second half of the model where the features (lower resolution) learnt by the encoder are projected onto the pixel space (higher resolution), thus enabling localization of the tumor on the MRI image. We have also added skip connections which enables the use of low-level features for tumor localization.

## References

1) Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computerassisted intervention (pp. 234-241).

2) Springer, Cham. S. Pereira, A. Pinto, V. Alves and C. A. Silva, "Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images," in IEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1240-1251, May 2016, doi: 10.1109/TMI.2016.2538465.