index

December 7, 2020

1 Detecting Tumors in MRI Brain Scans with Convolutional Neural Networks

1.1 Table of Contents

2 Business Understanding

According to the National Institute of Health, "[brain tumors] occur when something goes wrong with genes that regulate cell growth, allowing cells to grow and divide out of control... Depending on its type, a growing tumor may not cause any symptoms or can kill or displace healthy cells or disrupt their function" (Brain and Spinal Cord Tumors, 2020).

Brain tumors are diagnosed first by a neurological exam, and then through imaging methods including MRI scans. Currently, images are analyzed my MRI technicians before being sent to the doctor for a final analysis. If necessary, a biopsy is done to confirm a diagnosis. A biopsy is a surgical procedure where a small sample of tissue is extracted. Depending on the location of the suspected tumor, this can be dangerous to the patient, or impossible to perform if in a particularly sensitive area.

With advances in image classification techniques, preliminary analyses can be aided by computers through algorithms like those created in this project. This can reduce the need for potentially dangerous biopsies, allowing doctors and patients to focus on the next step, treatment. Doctors, patients, and MRI technicians stand to benefit from classification algorithms.

3 Data Understanding

Thousands of MRI brain scans were used in this project. The scans were sourced from two Kaggle datasets and BrainDevelopment.org: Kaggle 2018, Kaggle 2020, and BrainDevelopment.org. Included in these scans are brains with and without brain tumors present, of various section and scan types. The three types of sections included are frontal, medial, and horizontal.

Various scan types include Proton Density and Transverse Magnetization. Different types of scans are useful for detecting different types of tissue in different regions of the brain. Some tissue types are more visible under contrast material or propagate magnetization differently.

In this project, a scan with a tumor was considered Class 0, and a scan with a tumor present was considered Class 1. As a result, recall score was the prioritized along with accuracy in this type of neural network, as the effects of false negative can be much more harmful than a false positive.

4 Data Science Process

4.0.1 This notebook contains the code necessary to load and classify MRI images as containing a brain tumor or being tumor-free.

The process is as follows:

4.1 Imports

```
[1]:  # import libraries/packages
      import numpy as np
      import os
      import cv2
      import matplotlib.pyplot as plt
      import seaborn as sns


      from keras.preprocessing.image import ImageDataGenerator
      from keras.models import Sequential
      from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout,␣
       ↪BatchNormalization
      from keras.callbacks import ReduceLROnPlateau
      from sklearn.metrics import classification_report, confusion_matrix
      from skimage.segmentation import mark_boundaries
      import math
      import lime.lime_image as li
```

4.2 Data Preparation

After sourcing data, paths to the images were saved and used to split data into train, validation, and testing images. Using the paths, images were loaded using openCV and duplicates were removed

2

with custom functions. The images were then resized and reshaped to work with the format of Convolutional Neural Network. To create a more robust model, ImageDataGenerator was used to create slightly altered images used in tandem with original images while training the model.

```python
[2]: # import custom functions
     import sys
     sys.path.append('../../src')

     from data_functions import *
```

5 Get Image Paths

5.0.1 Custom function returns all image files within given directory

```python
[3]: # get paths from base directory
     base_dir = 'D:/MRI_data/Kaggle'
     absent_paths, present_paths = get_img_paths(base_dir)
```

5.0.2 Split Paths into Train, Validation, and Test sets with sklearn train_test_split

```python
[4]: # define train size for data set
     train_size = 0.8 # eighty percent
     test_size = 0.15 # fifteen percent
     val_size = 0.05 # five percent

     # calculate test size for second split
     test_size2 = test_size/(1-train_size)

     # split absent paths into train and test
     absent_train, absent_test = train_test_split(absent_paths,␣
      ↪train_size=train_size, random_state=2021)

     # split absent test paths into test and validation
     absent_test, absent_val = train_test_split(absent_test, train_size=test_size2,␣
      ↪random_state=2021)

     # split present paths into train and test
     present_train, present_test = train_test_split(present_paths,␣
      ↪train_size=train_size, random_state=2021)

     # split present paths into test and val
     present_test, present_val = train_test_split(present_test,␣
      ↪train_size=test_size2, random_state=2021)
```

```python
[5]: # combine train and test and validation lists (respectively)
     train = absent_train
     train.extend(present_train)
```

```
test = absent_test
test.extend(present_test)

val = absent_val
val.extend(present_val)
```

6 Load Image Data

6.0.1 Custom function loads data given path and class specified

```
[6]: # load image data
     train_img_data = get_data(train)

     val_img_data = get_data(val)

     test_img_data = get_data(test)
```

7 Remove Duplicates

7.0.1 Custom function compares each image and removes duplicates. This is necessary for correctly fitting the model.

```
[7]: # remove duplicates from each set
     unique_train_img, train_duplicate = remove_duplicates(train_img_data)
     unique_val_img, val_duplicate = remove_duplicates(val_img_data)
     unique_test_img, test_duplicate = remove_duplicates(test_img_data)
```

### 7.0.2 View samples of scans with and without tumors present

```
[37]: plt.rcParams["savefig.transparent"] = True
      labels = ['No Tumor', 'Tumor Present']

      # create figure hold images
      fig, ax = plt.subplots(1,2, figsize=(10,5))
      ax = ax.flatten()
      # plot image data from first [1] train image data (tumor-free brain)
      ax[0].imshow(unique_train_img[100][0], cmap='gray')
      ax[0].set_title(labels[unique_train_img[100][1]])
      ax[0].axis('off')

      # plot image data from last [-1] train image data (tumor patient brain)

      ax[1].imshow(unique_train_img[-22][0], cmap='gray')
      ax[1].set_title(labels[unique_train_img[-22][1]])
      ax[1].axis('off')
```

```
# save figure
plt.savefig("scan_comparison.png", transparent=True)
```
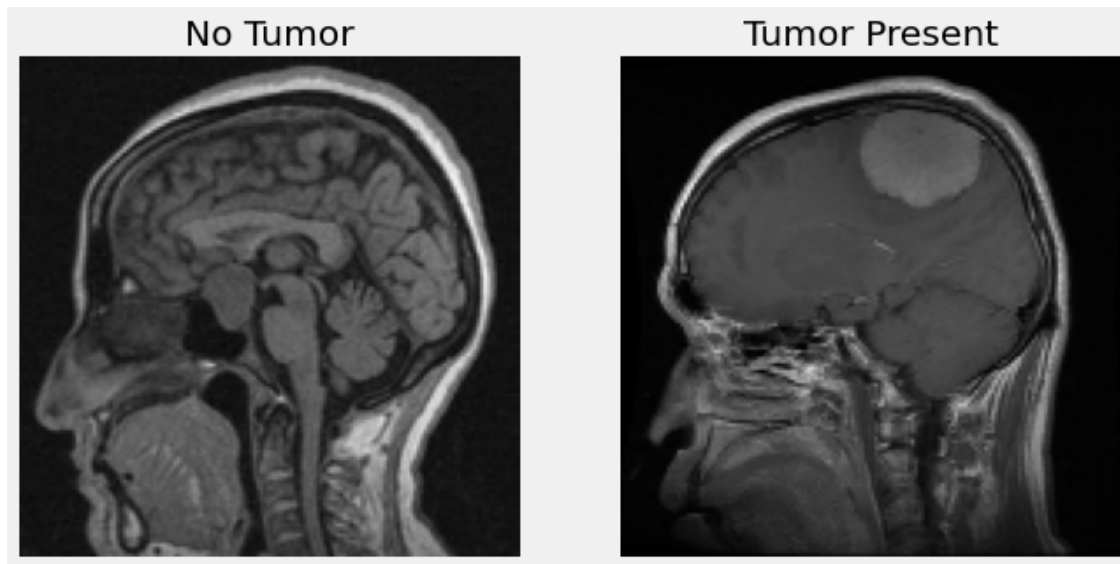

8   Reshape Image Data

```
[9]: # separate image data and identifiers
     x_train = []
     y_train = []

     x_val = []
     y_val = []

     x_test = []
     y_test = []

     for data, label, _ in unique_train_img:
         x_train.append(data)
         y_train.append(label)

     for data, label, _ in unique_val_img:
         x_val.append(data)
         y_val.append(label)

     for data, label, _ in unique_test_img:
         x_test.append(data)
         y_test.append(label)
```

```
[10]:  # scale data and convert to np.array for efficiency

       x_train = np.array(x_train)/255

       x_test = np.array(x_test)/255

       x_val = np.array(x_val)/255
```

```
[11]:  img_size = 150

       # resize data for deep learning
       x_train = x_train.reshape(-1, img_size, img_size, 3)

       x_test = x_test.reshape(-1, img_size, img_size, 3)

       x_val = x_val.reshape(-1, img_size, img_size, 3)
```

8.1 ImageDataGenerator

```
[12]:  datagen = ImageDataGenerator(\
                                   featurewise_center=False,
                                   samplewise_center=False,
                                   featurewise_std_normalization=False,
                                   samplewise_std_normalization=False,
                                   zca_whitening=False,
                                   rotation_range=30,
                                   zoom_range=0.25,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   horizontal_flip=True,
                                   vertical_flip=False)
```

9 Modeling

A Convolutional Neural Network makes it possible to process images in the form of pixels as input and to predict the desired classification as output. The development of Convolutional Neural Network (CNN) layers trains a model for significant gains in the ability to classify images and detect objects in a picture. Multiple processing layers use image analysis filters, or convolutions as the model is trained.

The convolutional layers help extract the spatial features in an image. The layers have a weight-sharing technique, which helps in reducing computation efforts.

A Convolution Neural Network (CNN) is built on three broad strategies:

1) Learn features using Convolution layer

2) Reduce computational costs by down sample the image and reduce dimensionality using Max-Pooling(subsampling)

3) Fully connected layer to equip the network with classification capabilities

9.1 First Simple Model (FSM)

```
[13]: metrics = ['Recall', 'accuracy']
      learning_rate_reduction_loss = ReduceLROnPlateau(monitor='val_loss',␣
      ↪patience=2, verbose=1, factor=0.3, min_lr=10**-9)
      learning_rate_reduction_acc = ReduceLROnPlateau(monitor='val_accuracy',␣
      ↪patience=2, verbose=1, factor=0.3, min_lr=10**-9)
```

```
[14]: metrics = ['Recall', 'accuracy']
      fsm = Sequential()
      fsm.add(Conv2D(32, (3,3), strides=1, padding='same', activation='relu',␣
      ↪input_shape=(150,150,3)))
      fsm.add(Flatten())
      fsm.add(Dense(units=1, activation='sigmoid'))
      fsm.compile(optimizer='adam', loss='binary_crossentropy', metrics=metrics)
      fsm.summary()
```

```
Model: "sequential"

_____
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      896

_____
 flatten (Flatten)           (None, 720000)            0

_____
 dense (Dense)               (None, 1)                 720001
=================================================================
Total params: 720,897
Trainable params: 720,897
Non-trainable params: 0

_____
```

```
[15]: epochs = 15
      fsm_history = fsm.fit(datagen.flow(x_train, y_train), epochs=epochs,␣
      ↪validation_data=datagen.flow(x_val, y_val),␣
      ↪callbacks=[learning_rate_reduction_loss, learning_rate_reduction_acc])
```

```
Epoch 1/15
80/80 [==============================] - 30s 370ms/step - loss: 0.6270 - recall:
0.9574 - accuracy: 0.8331 - val_loss: 0.5018 - val_recall: 1.0000 -
val_accuracy: 0.8343
Epoch 2/15
80/80 [==============================] - 29s 368ms/step - loss: 0.3801 - recall:
0.9872 - accuracy: 0.8610 - val_loss: 0.4586 - val_recall: 0.9658 -
val_accuracy: 0.8229
Epoch 3/15
```

```
80/80 [==============================] - 28s 350ms/step - loss: 0.3499 - recall:
0.9757 - accuracy: 0.8614 - val_loss: 0.4174 - val_recall: 0.9863 -
val_accuracy: 0.8400
Epoch 4/15
80/80 [==============================] - 31s 393ms/step - loss: 0.3473 - recall:
0.9766 - accuracy: 0.8665 - val_loss: 0.4471 - val_recall: 0.9795 -
val_accuracy: 0.8286
Epoch 5/15
80/80 [==============================] - ETA: 0s - loss: 0.3244 - recall: 0.9748
- accuracy: 0.8701
Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
80/80 [==============================] - 29s 358ms/step - loss: 0.3244 - recall:
0.9748 - accuracy: 0.8701 - val_loss: 0.4365 - val_recall: 0.9863 -
val_accuracy: 0.8514
Epoch 6/15
80/80 [==============================] - 29s 360ms/step - loss: 0.3223 - recall:
0.9739 - accuracy: 0.8705 - val_loss: 0.3928 - val_recall: 0.9384 -
val_accuracy: 0.8286
Epoch 7/15
80/80 [==============================] - 31s 393ms/step - loss: 0.3192 - recall:
0.9688 - accuracy: 0.8673 - val_loss: 0.3489 - val_recall: 0.9726 -
val_accuracy: 0.8686
Epoch 8/15
80/80 [==============================] - 34s 420ms/step - loss: 0.3075 - recall:
0.9711 - accuracy: 0.8740 - val_loss: 0.4081 - val_recall: 0.9452 -
val_accuracy: 0.8514
Epoch 9/15
80/80 [==============================] - ETA: 0s - loss: 0.3145 - recall: 0.9652
- accuracy: 0.8681
Epoch 00009: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.

Epoch 00009: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
80/80 [==============================] - 27s 341ms/step - loss: 0.3145 - recall:
0.9652 - accuracy: 0.8681 - val_loss: 0.3805 - val_recall: 0.9795 -
val_accuracy: 0.8514
Epoch 10/15
80/80 [==============================] - 28s 354ms/step - loss: 0.3030 - recall:
0.9693 - accuracy: 0.8685 - val_loss: 0.3903 - val_recall: 0.9521 -
val_accuracy: 0.8343
Epoch 11/15
80/80 [==============================] - ETA: 0s - loss: 0.3144 - recall: 0.9688
- accuracy: 0.8709
Epoch 00011: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.

Epoch 00011: ReduceLROnPlateau reducing learning rate to 2.429999949526973e-06.
80/80 [==============================] - 21s 269ms/step - loss: 0.3144 - recall:
0.9688 - accuracy: 0.8709 - val_loss: 0.3955 - val_recall: 0.9452 -
val_accuracy: 0.8400
```

```
Epoch 12/15
80/80 [==============================] - 18s 227ms/step - loss: 0.3027 - recall:
0.9720 - accuracy: 0.8764 - val_loss: 0.3709 - val_recall: 0.9589 -
val_accuracy: 0.8457
Epoch 13/15
80/80 [==============================] - ETA: 0s - loss: 0.3032 - recall: 0.9720
- accuracy: 0.8713
Epoch 00013: ReduceLROnPlateau reducing learning rate to 7.289999985005124e-07.

Epoch 00013: ReduceLROnPlateau reducing learning rate to 2.1870000637136398e-07.
80/80 [==============================] - 18s 229ms/step - loss: 0.3032 - recall:
0.9720 - accuracy: 0.8713 - val_loss: 0.3739 - val_recall: 0.9658 -
val_accuracy: 0.8514
Epoch 14/15
80/80 [==============================] - 18s 227ms/step - loss: 0.3124 - recall:
0.9665 - accuracy: 0.8677 - val_loss: 0.3635 - val_recall: 0.9658 -
val_accuracy: 0.8514
Epoch 15/15
80/80 [==============================] - ETA: 0s - loss: 0.3040 - recall: 0.9734
- accuracy: 0.8744
Epoch 00015: ReduceLROnPlateau reducing learning rate to 6.561000276406048e-08.
80/80 [==============================] - 18s 229ms/step - loss: 0.3040 - recall:
0.9734 - accuracy: 0.8744 - val_loss: 0.3429 - val_recall: 0.9658 -
val_accuracy: 0.8514
```

10 Final Model

```python
[16]: metrics = ['Recall', 'accuracy']

model= Sequential()
model.add(Conv2D(32, (3,3), strides =1, padding = 'same', activation ='relu',
 →input_shape = (150,150,3)))

model.add(MaxPool2D((2,2), strides=2, padding='same'))
model.add(Conv2D(64, (2,2), strides =1, padding ='same', activation='relu'))
model.add(Dropout(0.15))
model.add(MaxPool2D((2,2), strides=2, padding='same'))
model.add(Conv2D(64, (3,3), strides =1, padding ='same', activation='relu'))
model.add(Dropout(0.2))
model.add(MaxPool2D((2,2), strides=2, padding='same'))
model.add(Conv2D(64, (3,3), strides =1, padding ='same', activation='selu'))

model.add(Flatten())

model.add(Dense(units=128, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(units=1, activation='sigmoid'))
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=metrics)
model.summary()
```

```
Model: "sequential_1"
_____
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 150, 150, 32)      896
_____
max_pooling2d (MaxPooling2D) (None, 75, 75, 32)        0
_____
conv2d_2 (Conv2D)            (None, 75, 75, 64)        8256
_____
dropout (Dropout)            (None, 75, 75, 64)        0
_____
max_pooling2d_1 (MaxPooling2 (None, 38, 38, 64)        0
_____
conv2d_3 (Conv2D)            (None, 38, 38, 64)        36928
_____
dropout_1 (Dropout)          (None, 38, 38, 64)        0
_____
max_pooling2d_2 (MaxPooling2 (None, 19, 19, 64)        0
_____
conv2d_4 (Conv2D)            (None, 19, 19, 64)        36928
_____
flatten_1 (Flatten)          (None, 23104)             0
_____
dense_1 (Dense)              (None, 128)               2957440
_____
dropout_2 (Dropout)          (None, 128)               0
_____
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 3,040,577
Trainable params: 3,040,577
Non-trainable params: 0
_____
```

```
[17]: epochs = 15
      model_history = model.fit(datagen.flow(x_train, y_train), epochs=epochs,␣
        ↪validation_data=datagen.flow(x_val, y_val),␣
        ↪callbacks=[learning_rate_reduction_loss, learning_rate_reduction_acc])
```

```
Epoch 1/15
80/80 [==============================] - 51s 638ms/step - loss: 0.8403 - recall:
0.9739 - accuracy: 0.8390 - val_loss: 0.4378 - val_recall: 1.0000 -
val_accuracy: 0.8343
Epoch 2/15
```

```
80/80 [==============================] - 51s 635ms/step - loss: 0.3930 - recall:
0.9885 - accuracy: 0.8571 - val_loss: 0.4573 - val_recall: 0.9932 -
val_accuracy: 0.8286
Epoch 3/15
80/80 [==============================] - 54s 680ms/step - loss: 0.3768 - recall:
0.9853 - accuracy: 0.8528 - val_loss: 0.4219 - val_recall: 1.0000 -
val_accuracy: 0.8457
Epoch 4/15
80/80 [==============================] - 53s 658ms/step - loss: 0.3498 - recall:
0.9863 - accuracy: 0.8634 - val_loss: 0.4504 - val_recall: 0.8288 -
val_accuracy: 0.7886
Epoch 5/15
80/80 [==============================] - ETA: 0s - loss: 0.3222 - recall: 0.9707
- accuracy: 0.8642
Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
80/80 [==============================] - 53s 661ms/step - loss: 0.3222 - recall:
0.9707 - accuracy: 0.8642 - val_loss: 0.3789 - val_recall: 0.9110 -
val_accuracy: 0.8343
Epoch 6/15
80/80 [==============================] - 51s 635ms/step - loss: 0.2627 - recall:
0.9661 - accuracy: 0.8799 - val_loss: 0.3982 - val_recall: 0.8767 -
val_accuracy: 0.8457
Epoch 7/15
80/80 [==============================] - ETA: 0s - loss: 0.2536 - recall: 0.9629
- accuracy: 0.8823
Epoch 00007: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
80/80 [==============================] - 51s 641ms/step - loss: 0.2536 - recall:
0.9629 - accuracy: 0.8823 - val_loss: 0.3520 - val_recall: 0.9110 -
val_accuracy: 0.8400
Epoch 8/15
80/80 [==============================] - 53s 666ms/step - loss: 0.2283 - recall:
0.9620 - accuracy: 0.8933 - val_loss: 0.3144 - val_recall: 0.9658 -
val_accuracy: 0.8857
Epoch 9/15
80/80 [==============================] - 53s 662ms/step - loss: 0.2273 - recall:
0.9569 - accuracy: 0.8909 - val_loss: 0.3477 - val_recall: 0.9247 -
val_accuracy: 0.8629
Epoch 10/15
80/80 [==============================] - ETA: 0s - loss: 0.2186 - recall: 0.9643
- accuracy: 0.8969
Epoch 00010: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.

Epoch 00010: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.
80/80 [==============================] - 51s 640ms/step - loss: 0.2186 - recall:
0.9643 - accuracy: 0.8969 - val_loss: 0.3331 - val_recall: 0.9315 -
val_accuracy: 0.8857
Epoch 11/15
80/80 [==============================] - 51s 638ms/step - loss: 0.2237 - recall:
```

```
0.9610 - accuracy: 0.8953 - val_loss: 0.3400 - val_recall: 0.9384 -
val_accuracy: 0.8800
Epoch 12/15
80/80 [==============================] - ETA: 0s - loss: 0.2231 - recall: 0.9597
- accuracy: 0.8917
Epoch 00012: ReduceLROnPlateau reducing learning rate to 2.429999949526973e-06.

Epoch 00012: ReduceLROnPlateau reducing learning rate to 7.289999985005124e-07.
80/80 [==============================] - 51s 640ms/step - loss: 0.2231 - recall:
0.9597 - accuracy: 0.8917 - val_loss: 0.3601 - val_recall: 0.9041 -
val_accuracy: 0.8457
Epoch 13/15
80/80 [==============================] - 52s 652ms/step - loss: 0.2140 - recall:
0.9638 - accuracy: 0.8976 - val_loss: 0.3174 - val_recall: 0.9452 -
val_accuracy: 0.8857
Epoch 14/15
80/80 [==============================] - ETA: 0s - loss: 0.2187 - recall: 0.9633
- accuracy: 0.9012
Epoch 00014: ReduceLROnPlateau reducing learning rate to 2.1870000637136398e-07.

Epoch 00014: ReduceLROnPlateau reducing learning rate to 6.561000276406048e-08.
80/80 [==============================] - 52s 652ms/step - loss: 0.2187 - recall:
0.9633 - accuracy: 0.9012 - val_loss: 0.3384 - val_recall: 0.9041 -
val_accuracy: 0.8343
Epoch 15/15
80/80 [==============================] - 52s 656ms/step - loss: 0.2190 - recall:
0.9615 - accuracy: 0.8945 - val_loss: 0.3567 - val_recall: 0.9247 -
val_accuracy: 0.8629
```

11 Visualize Model Performance

11.0.1 Display changes in accuracy by epoch to help understand model learning progress
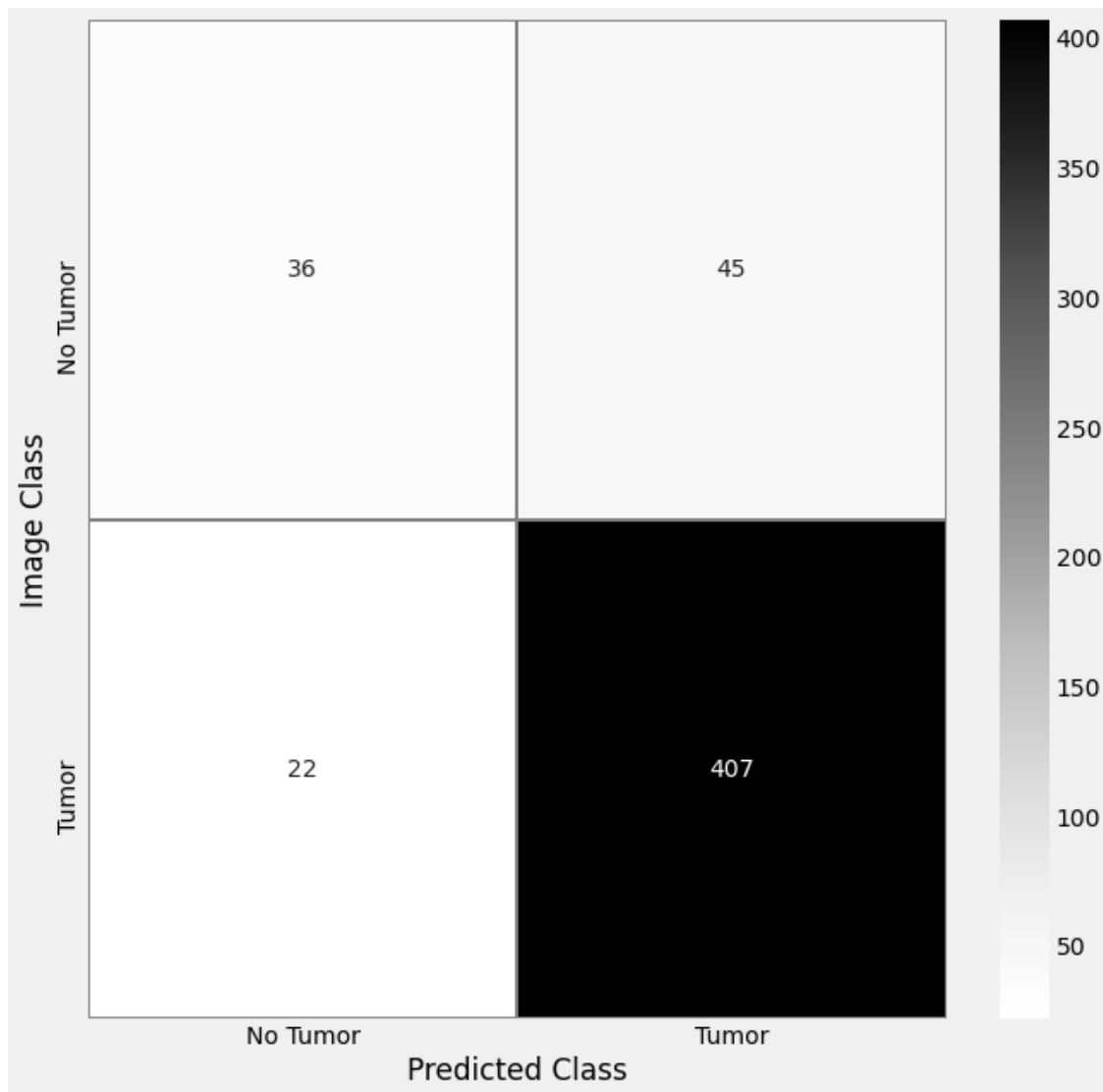
```
[18]: plot_metrics(fsm.history, path='../figures/metrics_fsm')
```
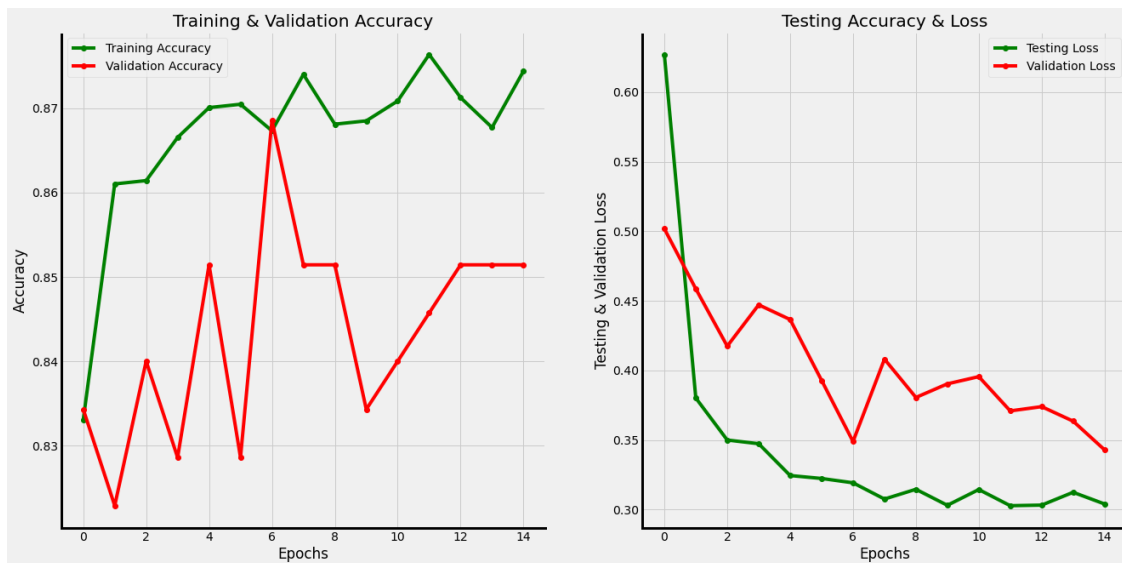
[19]:
```python
plot_confusion_matrix(fsm, x_test, y_test, path='../figures/
↪confusion_matrix_fsm')
```

```
              precision    recall  f1-score   support

    No Tumor       0.62      0.44      0.52        81
       Tumor       0.90      0.95      0.92       429

    accuracy                          0.87       510
   macro avg       0.76      0.70      0.72       510
weighted avg       0.86      0.87      0.86       510

[[ 36  45]
 [ 22 407]]
```
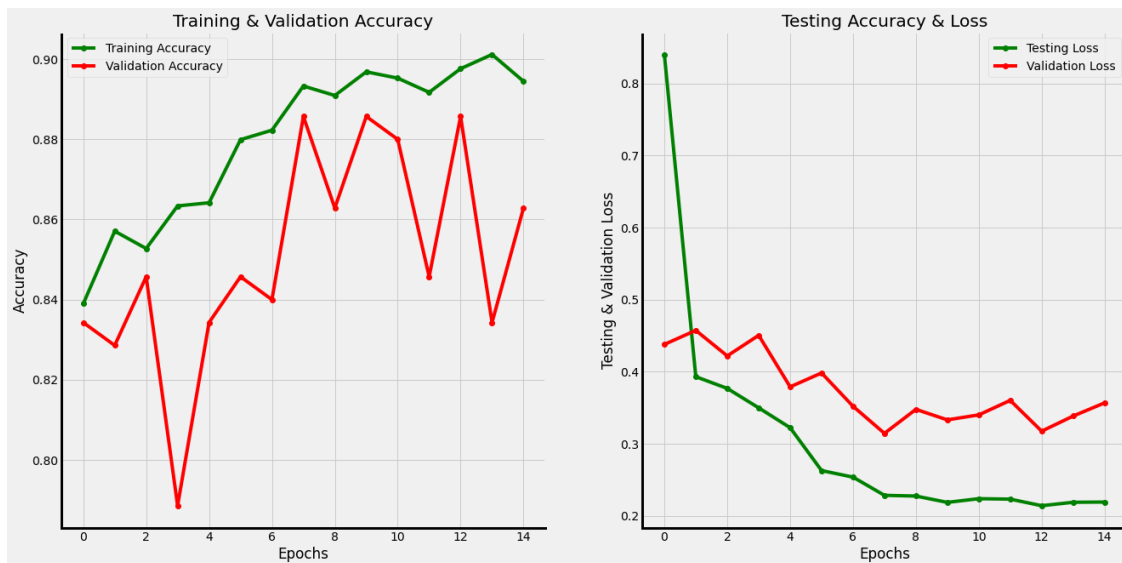
11.0.2 Create a confusion matrix representing the model's corect and incorrect predictions

```
[20]: plot_metrics(model.history, path='../figures/metrics_final')
```

```
[21]: plot_confusion_matrix(model, x_test, y_test, path='../figures/
      ↪confusion_matrix_final')
```

```
              precision    recall  f1-score   support

    No Tumor       0.64      0.69      0.67        81
       Tumor       0.94      0.93      0.93       429

    accuracy                          0.89       510
   macro avg       0.79      0.81      0.80       510
weighted avg       0.89      0.89      0.89       510

[[ 56  25]
 [ 31 398]]
```
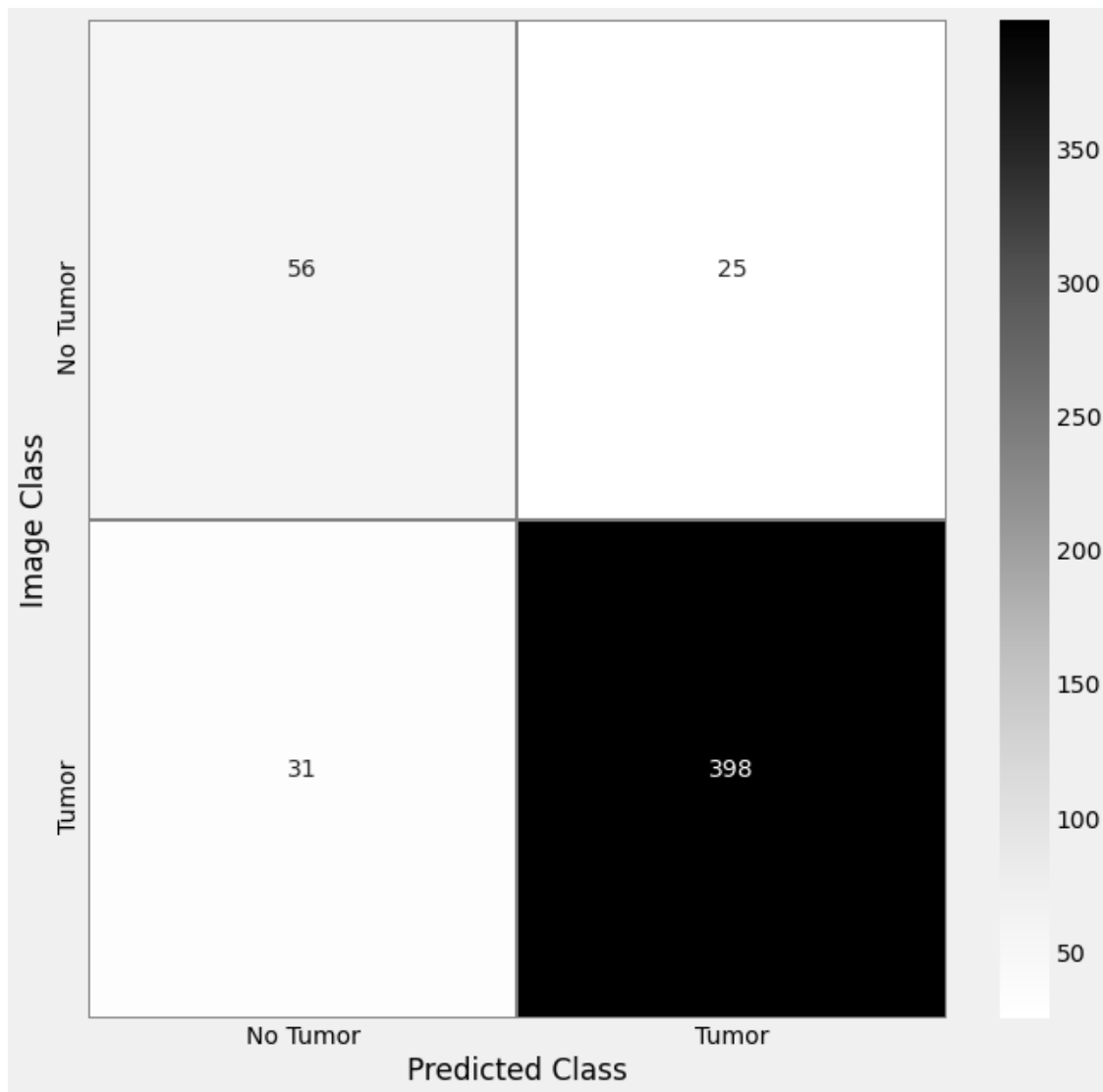
12 Analyze Model Performance

12.0.1 Get all image classifications and compare to predictions. Store for use in identifying error types.

```
[22]: # get model predictions
predictions = (model.predict(x_test) > 0.5).astype('int32')
# identify image classifications
good_preds0 = []
good_preds1 = []
type_1 = []
type_2 = []
```

```python
for i in range(len(y_test)):

    if y_test[i] == predictions[i]:
        if y_test[i] == 0:
            good_preds0.append(i)

        else:
            good_preds1.append(i)

    elif y_test[i] == 1:
        type_1.append(i)

    else:
        type_2.append(i)

good_preds0 = np.array(good_preds0)
good_preds1 = np.array(good_preds1)
type_1 = np.array(type_1)
type_2 = np.array(type_2)
```

12.1 TIME for LIME

12.1.1 Use LIME techniques to visualize model's analysis of images

To better understand how the model is classifying the images, implement LIME. LIME is an acronym for Local Interpretable Model-agnostic Explanations. The way LIME works is it breaks the image into several regions, called "superpixels", and the model classifies the image with the various superpixels turned on and off. Superpixels are then given weights based on their importance to the model's classification. Below are patient scans with tumors present. Using LIME, we see with the superpixel the model had given the most weight is picking up on the region containing the tumor, evidence the model is picking up on relevant regions when identifying the presence of tumors. (Note, this is not true for all images in dataset, particularly those misclassified)

```
[49]: num = 17
      lime_image(model, x_test[good_preds1[num]], min_superpixels=0,␣
       ↪max_superpixels=2, path=f'../figures/tumor_correct[{num}]', figsize=(15,10))
```
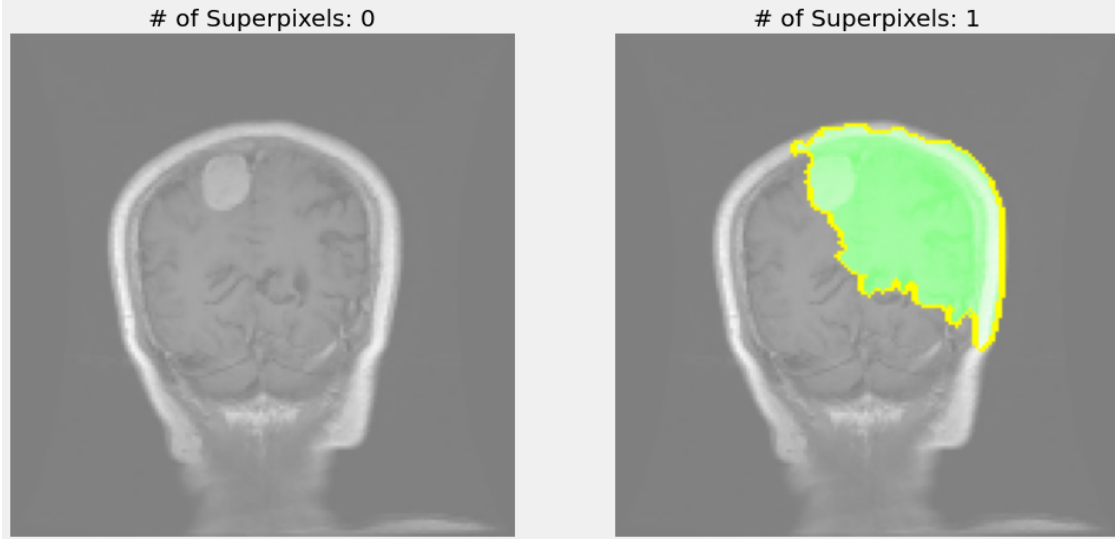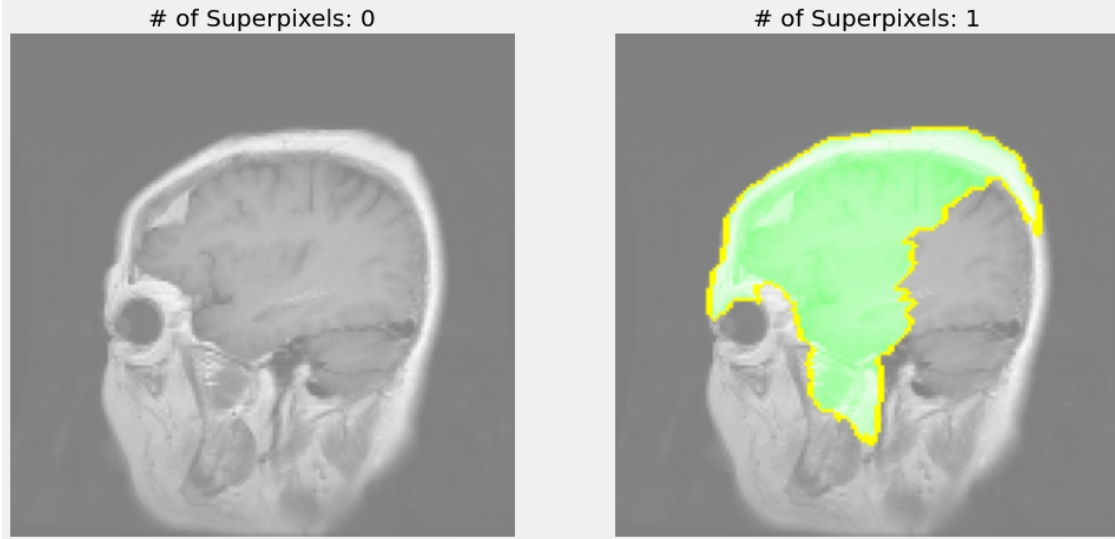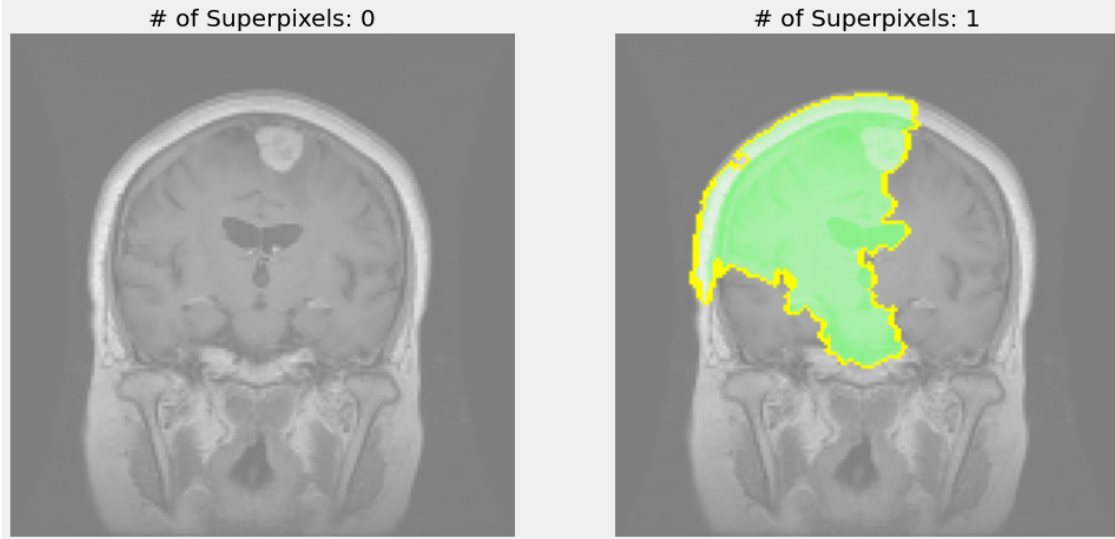
```
rows: 1  columns: 2
len(ax) = 2
m_end: 2  n_end: 0
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))
```

Different Features Analyzed by Model

# of Superpixels: 0   # of Superpixels: 1

```
[50]: num = 18
      lime_image(model, x_test[good_preds1[num]], min_superpixels=0,␣
       ↪max_superpixels=2, path=f'../figures/tumor_correct[{num}]', figsize=(15,10))
```

```
rows: 1  columns: 2
len(ax) = 2
m_end: 2  n_end: 0
```

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))
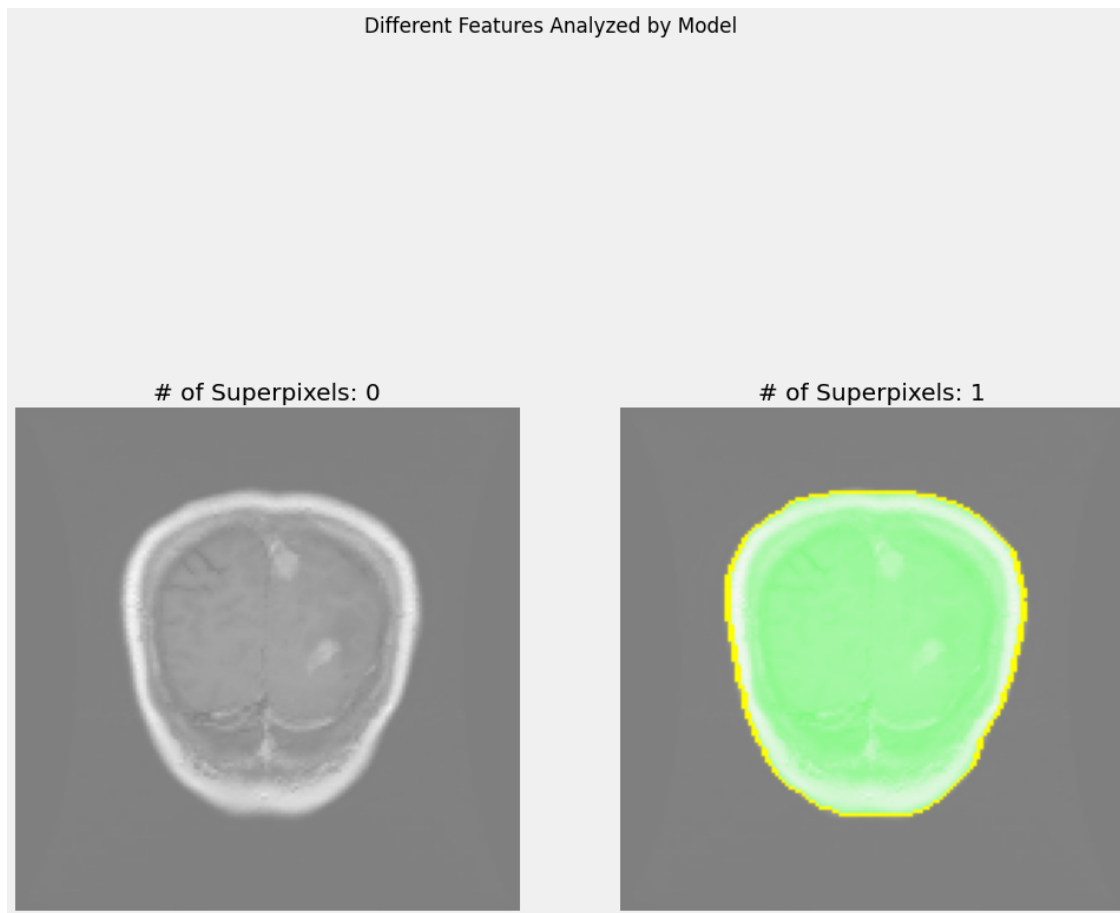
Different Features Analyzed by Model

# of Superpixels: 0   # of Superpixels: 1

[52]:
```python
num = 20
lime_image(model, x_test[good_preds1[num]], min_superpixels=0,
    max_superpixels=2, path=f'../figures/tumor_correct[{num}]', figsize=(15,10))
```

```
rows: 1  columns: 2
len(ax) = 2
m_end: 2  n_end: 0
```

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))

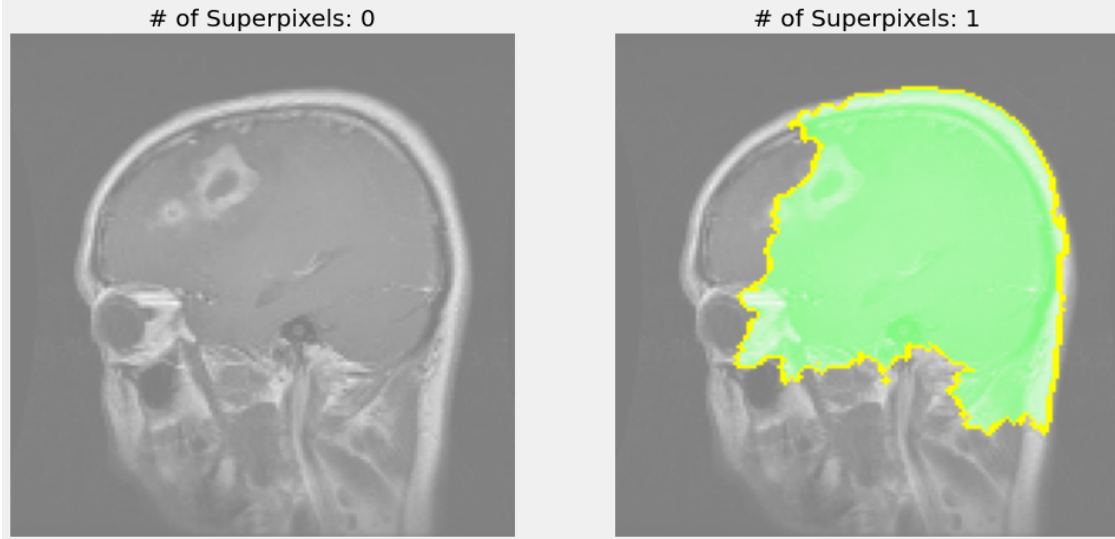HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))

```
[57]: num = 25
      lime_image(model, x_test[good_preds1[num]], min_superpixels=0,␣
       ↪max_superpixels=2, path=f'../figures/tumor_correct[{num}]', figsize=(15,10))
```

rows: 1   columns: 2
len(ax) = 2
m_end: 2   n_end: 0

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))

Different Features Analyzed by Model

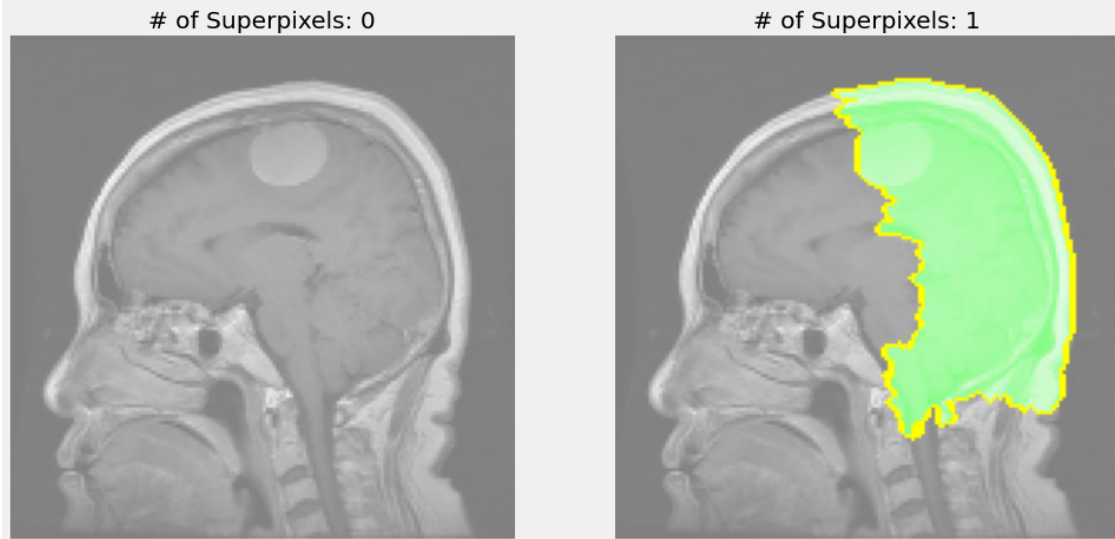# of Superpixels: 0   # of Superpixels: 1

```
[47]: num = 26
      lime_image(model, x_test[good_preds1[num]], min_superpixels=0,
      ↪max_superpixels=2, path=f'../figures/tumor_correct[{num}]', figsize=(15,10))
```

```
rows: 1  columns: 2
len(ax) = 2
m_end: 2  n_end: 0
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))
```

Different Features Analyzed by Model

# of Superpixels: 0   # of Superpixels: 1

```
[59]: num = 28
      lime_image(model, x_test[good_preds1[num]], min_superpixels=0,
       ↪max_superpixels=2, path=f'../figures/tumor_correct[{num}]', figsize=(15,10))
```

```
rows: 1  columns: 2
len(ax) = 2
m_end: 2  n_end: 0
```

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))

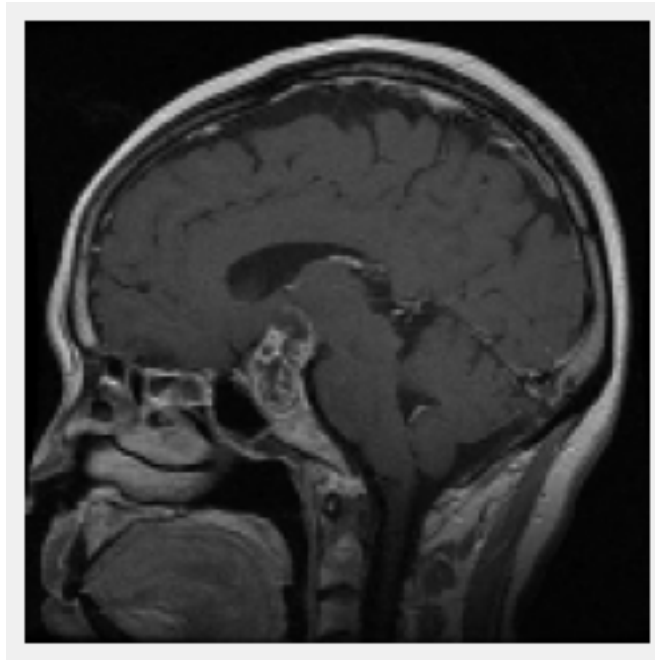HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))

Different Features Analyzed by Model

# of Superpixels: 0   # of Superpixels: 1

```
[71]: plt.rcParams["savefig.transparent"] = True
      labels = ['No Tumor', 'Tumor Present']

      # create figure hold images
      fig, ax = plt.subplots()
      # plot image data from first [1] train image data (tumor-free brain)
      ax.imshow(x_test[good_preds1[32]], cmap='gray')
      # ax[0].set_title(labels[unique_train_img[100][1]])
      ax.axis('off')

      # save figure
      plt.savefig("../../report/figures/identified.png", transparent=True)
```
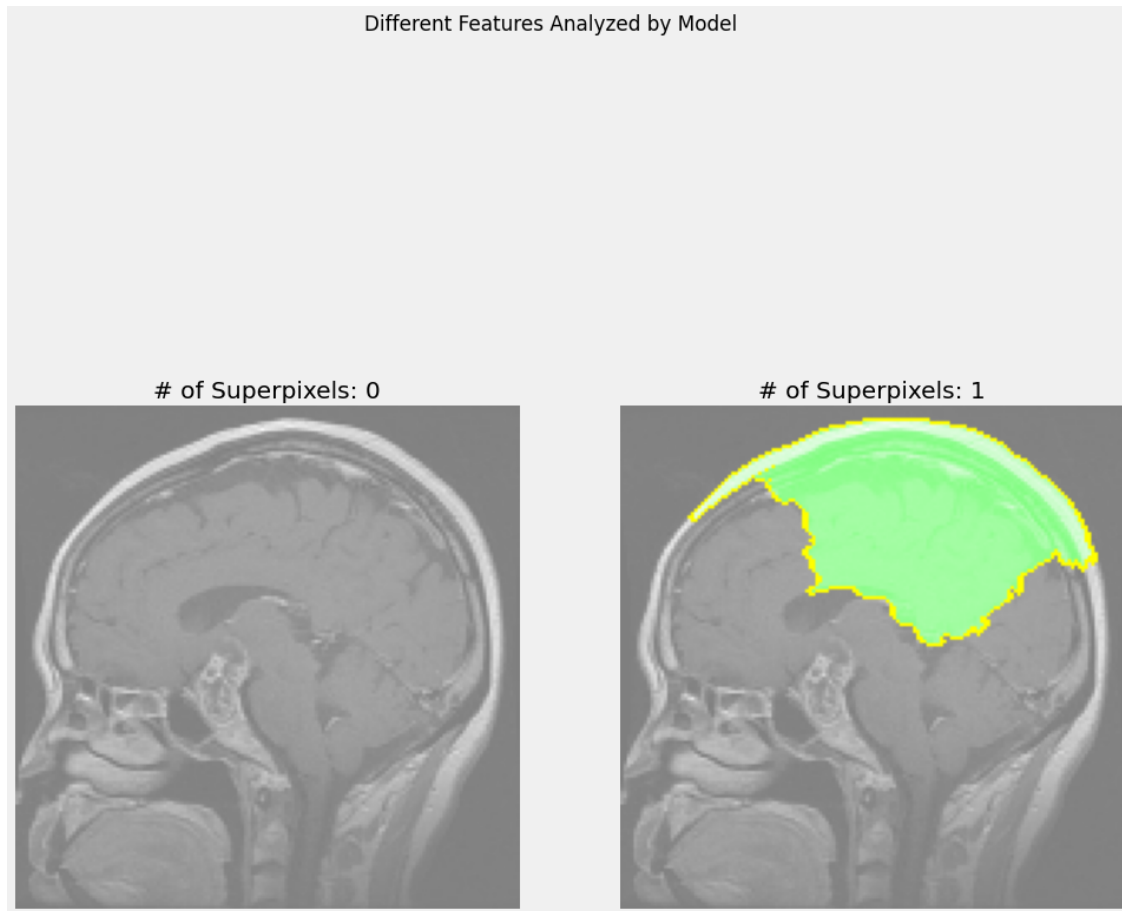
```
[63]: num = 32
      lime_image(model, x_test[good_preds1[num]], min_superpixels=0,␣
       ↪max_superpixels=2, path=f'../figures/tumor_correct[{num}]', figsize=(15,10))
```

```
rows: 1  columns: 2
len(ax) = 2
m_end: 2  n_end: 0
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0), HTML(value='')))
```

Different Features Analyzed by Model

13 Conclusion

Between the CNN's shown, we've seen good recall and improved accuracy. This weas followed by verification with LIME that the model is detecting areas with tumors. With more data and tuning, the model shows potential for higher accuracy and recall. In addition, creating variations of the model may lead to more advanced diagnoses, such as classify tumors by location, and determining if a tumor is benign or malignant.