In [1]:
```
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Input, UpSampling2D, Concatenate
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [2]:
```
# Paths to the data folders
train_dir = "C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training"
test_dir = "C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing"

# Image dimensions and number of classes
img_dim = 128  # Adjust based on the image resolution and detail needed
num_classes = 4  # Four classes: glioma, meningioma, no tumor, pituitary
```

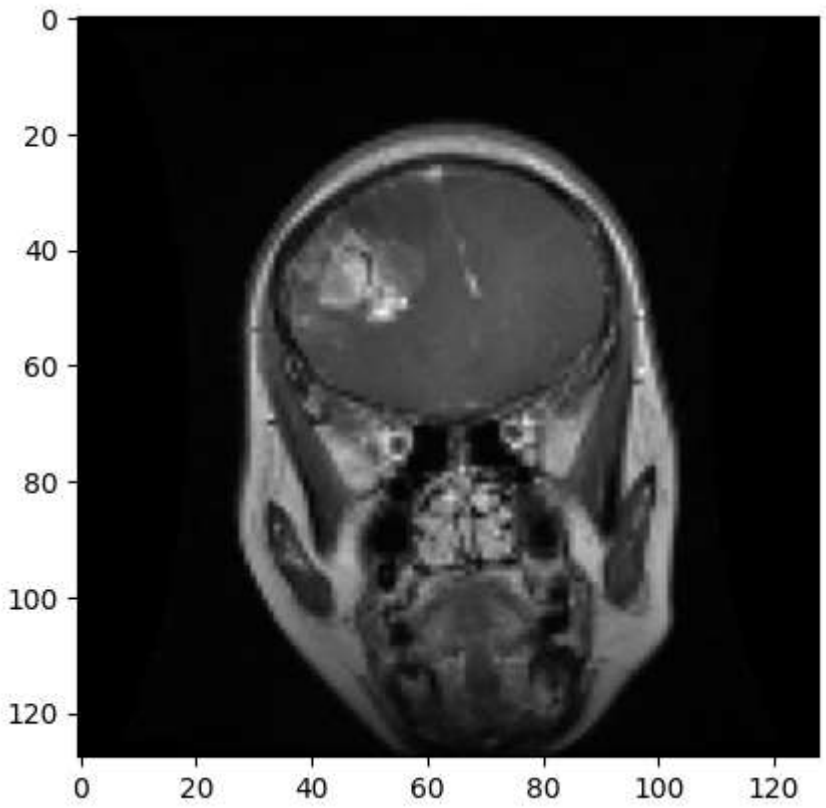
In [3]:
```
# Define the label mapping for each tumor type
label_map = {
    'glioma': 0,
    'meningioma': 1,
    'notumor': 2,
    'pituitary': 3
}
```

In [4]:
```
def load_data(data_dir, img_dim=128):
    images = []
    labels = []

    # Iterate through each folder in the directory
    for label_name, label_value in label_map.items():
        folder_path = os.path.join(data_dir, label_name)
        print(f"Loading images from: {folder_path}")

        # Check if the folder exists
        if not os.path.exists(folder_path):
            print(f"Warning: Folder {folder_path} does not exist.")
            continue
```

```
        # Loop through each image file in the folder
        for image_file in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image_file)

            # Load each image in grayscale
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            # Check if the image was loaded successfully
            if image is None:
                print(f"Warning: Image {image_path} could not be loaded.")
                continue

            # Resize and normalize the image
            image = cv2.resize(image, (img_dim, img_dim)) / 255.0
            images.append(image)
            labels.append(label_value)

    # Convert lists to numpy arrays and add a channel dimension for grayscale images
    images = np.array(images).reshape(-1, img_dim, img_dim, 1)
    labels = np.array(labels)

    return images, labels

# Example usage
train_dir = "C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training"
test_dir = "C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing"
train_images, train_labels = load_data(train_dir)
test_images, test_labels = load_data(test_dir)

# Print shape of the loaded data to verify
print("Training data shape:", train_images.shape)
print("Training labels shape:", train_labels.shape)
print("Testing data shape:", test_images.shape)
print("Testing labels shape:", test_labels.shape)
```

```
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\glioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\meningioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\notumor
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\pituitary
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing\glioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing\meningioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing\notumor
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing\pituitary
Training data shape: (5712, 128, 128, 1)
Training labels shape: (5712,)
Testing data shape: (1311, 128, 128, 1)
Testing labels shape: (1311,)
```

In [5]:
```
# Show the first image in the training set
plt.imshow(train_images[0], cmap='gray')
plt.show()
```



In [6]:
```
train_images[0]
```

Out[6]:
```
array([[[0.],
        [0.],
        [0.],
        ...,
        [0.],
        [0.],
        [0.]],

       [[0.],
        [0.],
        [0.],
        ...,
        [0.],
        [0.],
        [0.]],

       [[0.],
        [0.],
        [0.],
        ...,
        [0.],
        [0.],
        [0.]],

       ...,

       [[0.],
        [0.],
        [0.],
        ...,
        [0.],
        [0.],
        [0.]],

       [[0.],
        [0.],
        [0.],
        ...,
        [0.],
        [0.],
        [0.]],
```

```
       [[0.],
        [0.],
        [0.],
        ...,
        [0.],
        [0.],
        [0.]]])
```

In [7]:
```
reverse_label_map = {v: k for k, v in label_map.items()}  # Reverse mapping

# Load data as before
train_images, train_labels = load_data(train_dir)

# Convert numeric labels back to text labels for printing
text_labels = [reverse_label_map[label] for label in train_labels]

# Print the first 10 labels
print(text_labels[:10])
```

```
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\glioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\meningioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\notumor
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\pituitary
['glioma', 'glioma', 'glioma', 'glioma', 'glioma', 'glioma', 'glioma', 'glioma', 'glioma', 'glioma']
```

In [8]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Paths to the data folders
train_dir = "C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training"
test_dir = "C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing"

# Load and preprocess images
train_images, train_labels = load_data(train_dir)

# Check shapes before splitting
print("Shape of train_images before splitting:", train_images.shape)
print("Shape of train_labels before splitting:", train_labels.shape)

# Split the data into training and validation sets
train_images, val_images, train_labels, val_labels = train_test_split(
    train_images, train_labels, test_size=0.2, random_state=42
```

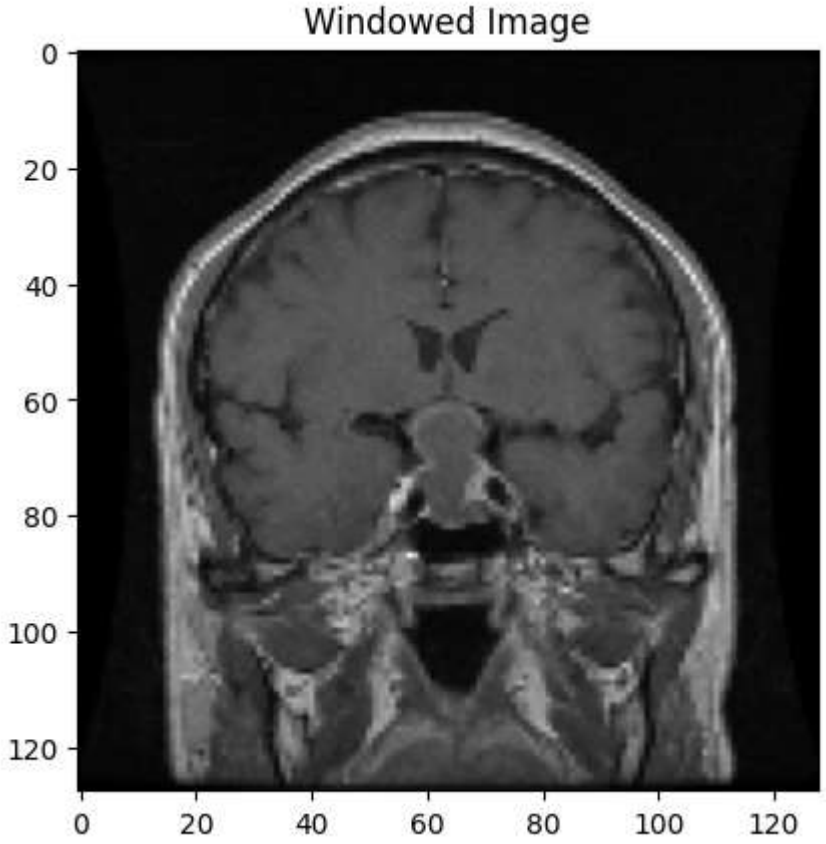
```
)

# Verify the shapes after splitting
print("Shape of train_images after splitting:", train_images.shape)
print("Shape of val_images after splitting:", val_images.shape)
print("Shape of train_labels after splitting:", train_labels.shape)
print("Shape of val_labels after splitting:", val_labels.shape)

# One-hot encode the labels after splitting
train_labels = to_categorical(train_labels, num_classes=num_classes)
val_labels = to_categorical(val_labels, num_classes=num_classes)

# Verify the shapes after encoding
print("Shape of train_labels after encoding:", train_labels.shape)
print("Shape of val_labels after encoding:", val_labels.shape)
```

```
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\glioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\meningioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\notumor
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Training\pituitary
Shape of train_images before splitting: (5712, 128, 128, 1)
Shape of train_labels before splitting: (5712,)
Shape of train_images after splitting: (4569, 128, 128, 1)
Shape of val_images after splitting: (1143, 128, 128, 1)
Shape of train_labels after splitting: (4569,)
Shape of val_labels after splitting: (1143,)
Shape of train_labels after encoding: (4569, 4)
Shape of val_labels after encoding: (1143, 4)
```

In [9]:

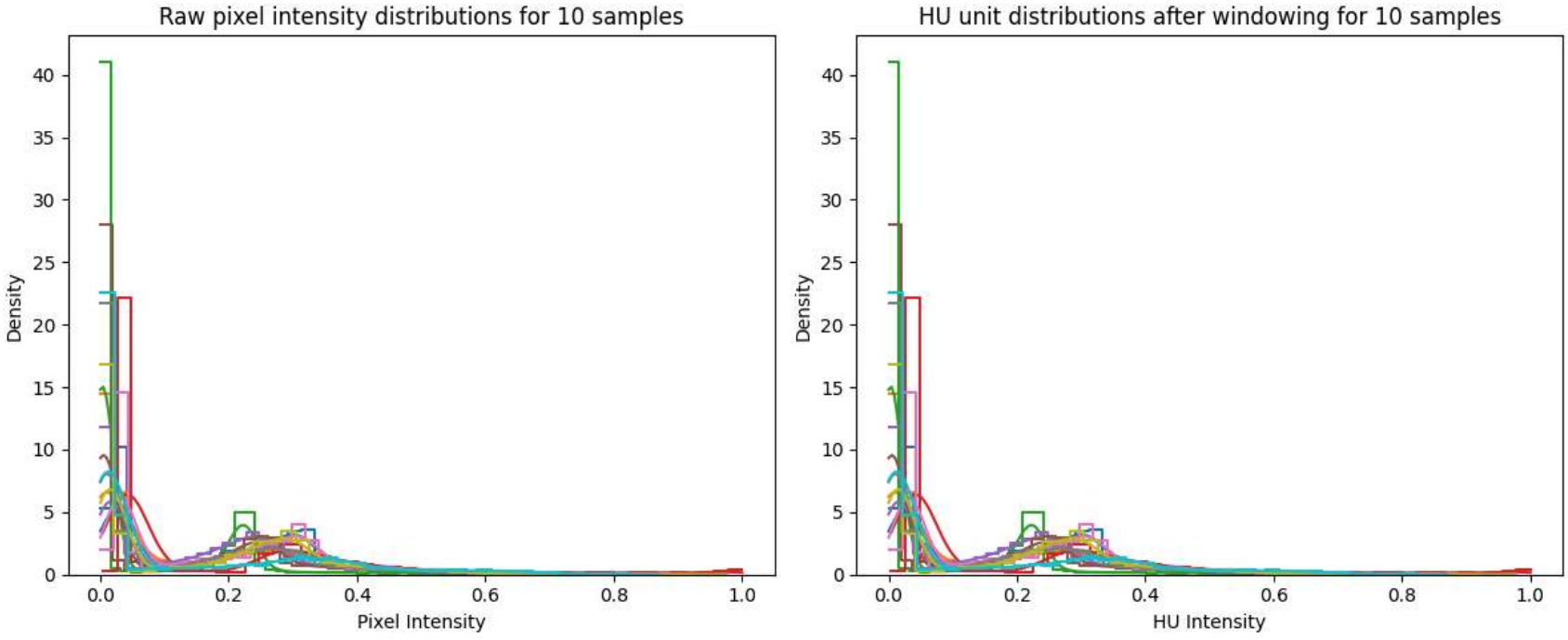
```
def apply_window(hu_image, center, width):
    hu_image = hu_image.copy()
    min_value = center - width // 2
    max_value = center + width // 2
    hu_image[hu_image < min_value] = min_value
    hu_image[hu_image > max_value] = max_value
    return hu_image

# Example usage with a specific window level and width
# Set appropriate window center and width for the tissue you want to highlight
window_center = 50  # Example value, adjust based on analysis
window_width = 100  # Example value, adjust based on analysis
```

```
# Apply windowing to each image in the training set
windowed_images = [apply_window(img, window_center, window_width) for img in train_images]

# Display a sample windowed image
plt.imshow(windowed_images[0].squeeze(), cmap='gray')
plt.title("Windowed Image")
plt.show()
```

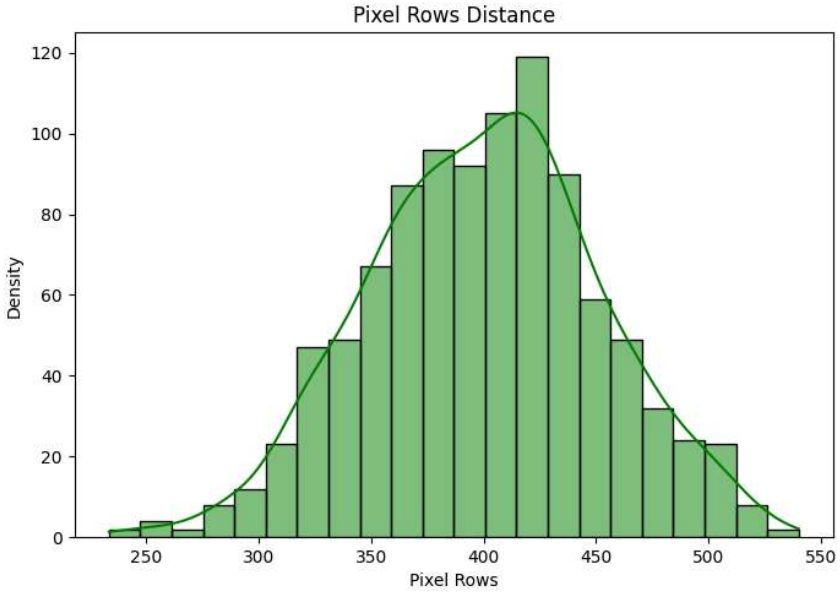


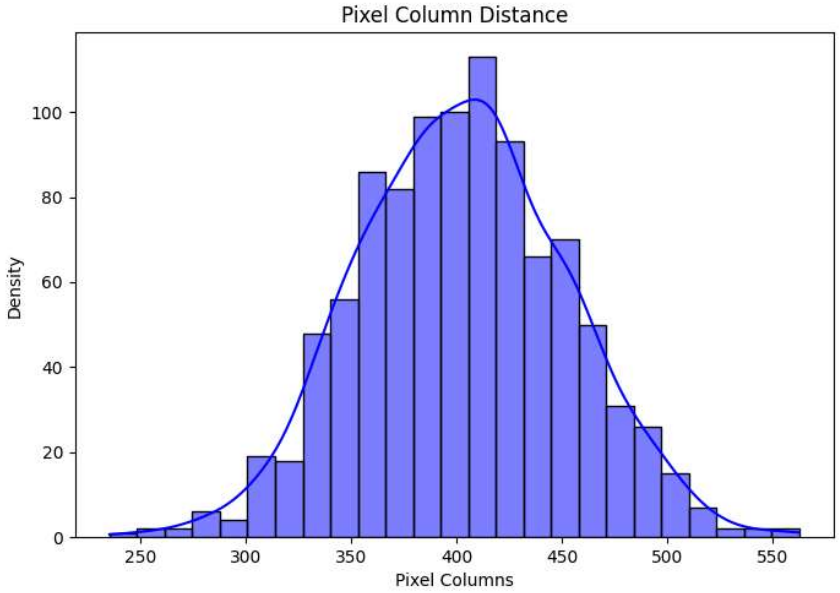
In [10]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Function to plot histograms of pixel intensities
def plot_histogram_analysis(images, num_samples=10):
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))
```

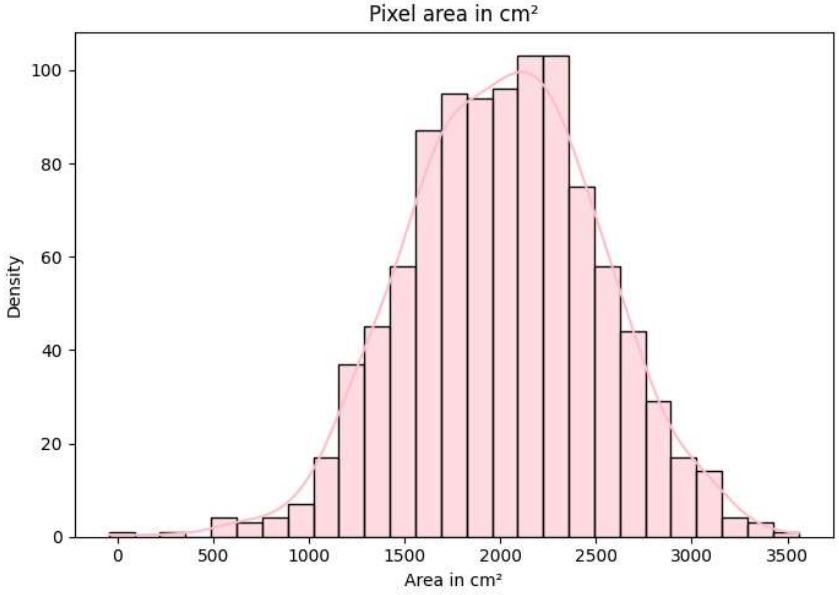
```
    # Plot histogram for raw pixel distributions (random samples)
    for i in range(num_samples):
        sns.histplot(images[i].flatten(), kde=True, ax=ax[0], stat="density", element="step", fill=False)
    ax[0].set_title(f"Raw pixel intensity distributions for {num_samples} samples")
    ax[0].set_xlabel("Pixel Intensity")
    ax[0].set_ylabel("Density")

    # If windowing is applied, repeat for the processed images (after windowing)
    # Assuming `windowed_images` contains images after windowing
    for i in range(num_samples):
        sns.histplot(windowed_images[i].flatten(), kde=True, ax=ax[1], stat="density", element="step", fill=False)
    ax[1].set_title(f"HU unit distributions after windowing for {num_samples} samples")
    ax[1].set_xlabel("HU Intensity")
    ax[1].set_ylabel("Density")

    plt.tight_layout()
    plt.show()

# Example usage (assuming `train_images` is the original data and `windowed_images` is the windowed data)
plot_histogram_analysis(train_images, num_samples=10)
```

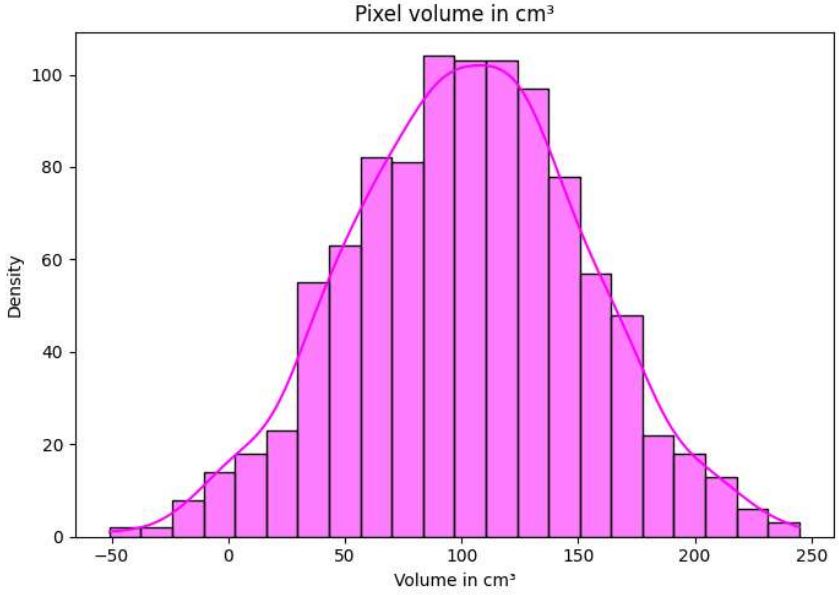


In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Generate example data for demonstration
pixel_rows = np.random.normal(400, 50, 1000)  # Simulate pixel row data
pixel_columns = np.random.normal(400, 50, 1000)  # Simulate pixel column data
slice_area = np.random.normal(2000, 500, 1000)  # Simulate pixel slice area data
slice_volume = np.random.normal(100, 50, 1000)  # Simulate pixelslice volume data

# Create a 2x2 grid of plots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Plot each metric in a separate subplot
sns.histplot(pixel_rows, kde=True, ax=axes[0, 0], color='green')
axes[0, 0].set_title("Pixel Rows Distance")
axes[0, 0].set_xlabel("Pixel Rows")
axes[0, 0].set_ylabel("Density")

sns.histplot(pixel_columns, kde=True, ax=axes[0, 1], color='blue')
axes[0, 1].set_title("Pixel Column Distance")
axes[0, 1].set_xlabel("Pixel Columns")
axes[0, 1].set_ylabel("Density")

sns.histplot(slice_area, kde=True, ax=axes[1, 0], color='pink')
axes[1, 0].set_title("Pixel area in cm²")
axes[1, 0].set_xlabel("Area in cm²")
axes[1, 0].set_ylabel("Density")

sns.histplot(slice_volume, kde=True, ax=axes[1, 1], color='magenta')
axes[1, 1].set_title("Pixel volume in cm³")
axes[1, 1].set_xlabel("Volume in cm³")
axes[1, 1].set_ylabel("Density")

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```









In [12]:
```
import pandas as pd

# Initialize a list to store the data
data = []
```

```
# Loop over each image and its corresponding label
for image, label in zip(train_images, train_labels):
    # Flatten the image to a 1D array of features
    features = image.flatten()
    # Append the features and label to the data list
    data.append([label] + features.tolist())

# Convert the data list to a DataFrame
# The first column will be the label, and the rest will be the pixel values as features
df = pd.DataFrame(data, columns=['label'] + [f'pixel_{i}' for i in range(features.size)])

# Display the first few rows of the DataFrame to verify
print(df.head())
```

```
                  label   pixel_0   pixel_1   pixel_2   pixel_3   pixel_4  \
0  [0.0, 0.0, 0.0, 1.0]  0.000000  0.000000  0.003922  0.003922  0.003922
1  [0.0, 0.0, 0.0, 1.0]  0.000000  0.000000  0.000000  0.000000  0.000000
2  [1.0, 0.0, 0.0, 0.0]  0.000000  0.000000  0.000000  0.000000  0.000000
3  [0.0, 0.0, 1.0, 0.0]  0.039216  0.039216  0.039216  0.039216  0.039216
4  [1.0, 0.0, 0.0, 0.0]  0.000000  0.000000  0.000000  0.000000  0.000000

    pixel_5   pixel_6   pixel_7   pixel_8  ...  pixel_16374  pixel_16375  \
0  0.003922  0.003922  0.003922  0.003922  ...     0.000000     0.000000
1  0.000000  0.000000  0.000000  0.000000  ...     0.000000     0.000000
2  0.000000  0.000000  0.000000  0.000000  ...     0.000000     0.000000
3  0.039216  0.039216  0.039216  0.039216  ...     0.039216     0.039216
4  0.000000  0.000000  0.000000  0.003922  ...     0.000000     0.000000

   pixel_16376  pixel_16377  pixel_16378  pixel_16379  pixel_16380  \
0     0.000000     0.000000     0.003922     0.003922     0.000000
1     0.000000     0.000000     0.000000     0.000000     0.000000
2     0.000000     0.000000     0.000000     0.000000     0.000000
3     0.039216     0.039216     0.039216     0.039216     0.039216
4     0.000000     0.000000     0.000000     0.000000     0.000000

   pixel_16381  pixel_16382  pixel_16383
0     0.000000     0.000000     0.000000
1     0.000000     0.000000     0.000000
2     0.000000     0.000000     0.000000
3     0.039216     0.039216     0.039216
4     0.000000     0.000000     0.000000

[5 rows x 16385 columns]
```

In [13]:
```python
#Model Training
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Build and compile the classification model
classification_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_dim, img_dim, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
```

```
    Flatten(),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')  # num_classes = 4 for multi-class classification
])
classification_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
classification_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 128)               3211392
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 516
=================================================================
Total params: 3,304,580
Trainable params: 3,304,580
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```
#Data Augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data augmentation for classification
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
```

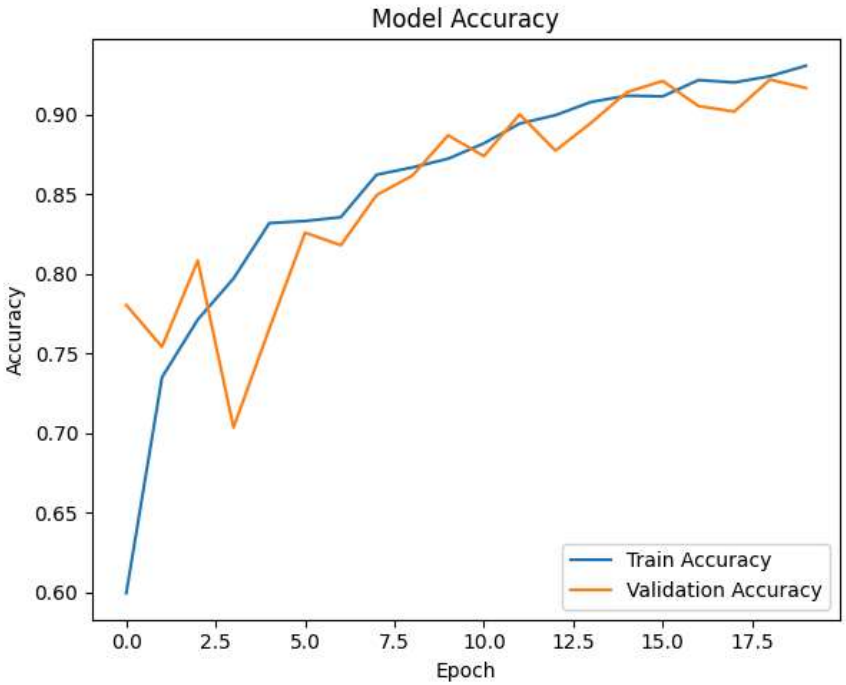
```
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Assuming train_images and train_labels are already loaded
train_generator = train_datagen.flow(train_images, train_labels, batch_size=32)
history_classification = classification_model.fit(
    train_generator,
    epochs=20,
    validation_data=(val_images, val_labels)  # Assuming val_images and val_labels are already split
)
```

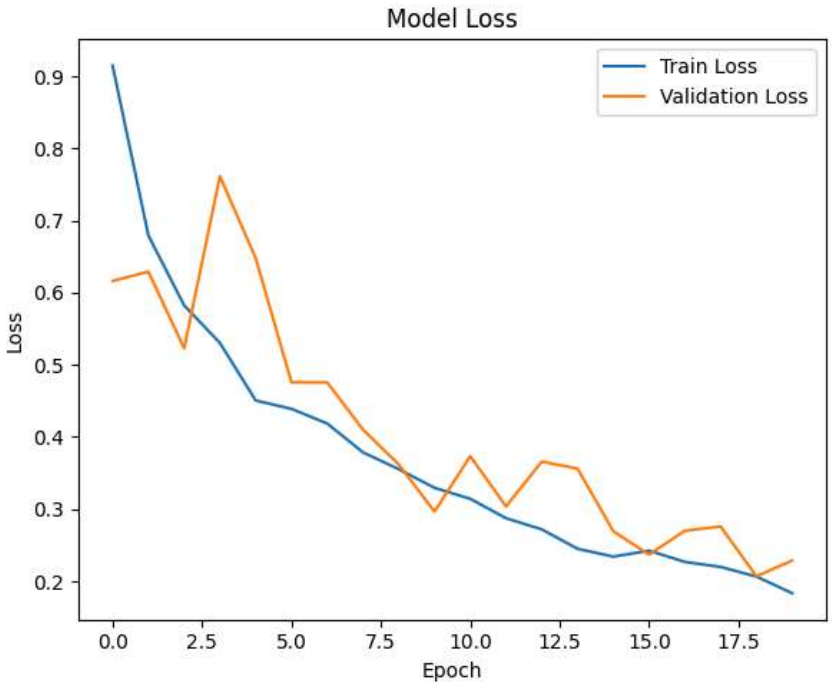
```
Epoch 1/20
143/143 [==============================] - 110s 772ms/step - loss: 0.9148 - accuracy: 0.5993 - val_loss: 0.6164 - val
_accuracy: 0.7804
Epoch 2/20
143/143 [==============================] - 63s 440ms/step - loss: 0.6793 - accuracy: 0.7350 - val_loss: 0.6293 - val_
accuracy: 0.7542
Epoch 3/20
143/143 [==============================] - 62s 431ms/step - loss: 0.5825 - accuracy: 0.7713 - val_loss: 0.5230 - val_
accuracy: 0.8084
Epoch 4/20
143/143 [==============================] - 63s 443ms/step - loss: 0.5305 - accuracy: 0.7971 - val_loss: 0.7615 - val_
accuracy: 0.7034
Epoch 5/20
143/143 [==============================] - 62s 435ms/step - loss: 0.4507 - accuracy: 0.8319 - val_loss: 0.6481 - val_
accuracy: 0.7655
Epoch 6/20
143/143 [==============================] - 60s 423ms/step - loss: 0.4391 - accuracy: 0.8332 - val_loss: 0.4758 - val_
accuracy: 0.8259
Epoch 7/20
143/143 [==============================] - 61s 425ms/step - loss: 0.4187 - accuracy: 0.8356 - val_loss: 0.4756 - val_
accuracy: 0.8180
Epoch 8/20
143/143 [==============================] - 66s 463ms/step - loss: 0.3787 - accuracy: 0.8623 - val_loss: 0.4102 - val_
accuracy: 0.8495
Epoch 9/20
143/143 [==============================] - 62s 434ms/step - loss: 0.3554 - accuracy: 0.8669 - val_loss: 0.3620 - val_
accuracy: 0.8618
Epoch 10/20
143/143 [==============================] - 64s 445ms/step - loss: 0.3295 - accuracy: 0.8724 - val_loss: 0.2965 - val_
accuracy: 0.8871
Epoch 11/20
143/143 [==============================] - 63s 443ms/step - loss: 0.3143 - accuracy: 0.8820 - val_loss: 0.3735 - val_
accuracy: 0.8740
Epoch 12/20
143/143 [==============================] - 62s 433ms/step - loss: 0.2874 - accuracy: 0.8945 - val_loss: 0.3038 - val_
accuracy: 0.9003
Epoch 13/20
143/143 [==============================] - 64s 444ms/step - loss: 0.2719 - accuracy: 0.8998 - val_loss: 0.3659 - val_
accuracy: 0.8775
Epoch 14/20
143/143 [==============================] - 63s 440ms/step - loss: 0.2450 - accuracy: 0.9081 - val_loss: 0.3561 - val_
accuracy: 0.8950
```

```
Epoch 15/20
143/143 [==============================] - 63s 441ms/step - loss: 0.2342 - accuracy: 0.9120 - val_loss: 0.2693 - val_
accuracy: 0.9143
Epoch 16/20
143/143 [==============================] - 63s 441ms/step - loss: 0.2424 - accuracy: 0.9116 - val_loss: 0.2370 - val_
accuracy: 0.9213
Epoch 17/20
143/143 [==============================] - 67s 466ms/step - loss: 0.2268 - accuracy: 0.9219 - val_loss: 0.2700 - val_
accuracy: 0.9055
Epoch 18/20
143/143 [==============================] - 67s 466ms/step - loss: 0.2198 - accuracy: 0.9203 - val_loss: 0.2759 - val_
accuracy: 0.9020
Epoch 19/20
143/143 [==============================] - 65s 451ms/step - loss: 0.2065 - accuracy: 0.9243 - val_loss: 0.2069 - val_
accuracy: 0.9221
Epoch 20/20
143/143 [==============================] - 62s 432ms/step - loss: 0.1834 - accuracy: 0.9308 - val_loss: 0.2287 - val_
accuracy: 0.9169
```

In [15]:

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history_classification.history['accuracy'], label='Train Accuracy')
plt.plot(history_classification.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history_classification.history['loss'], label='Train Loss')
plt.plot(history_classification.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')

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





In [17]:
```python
# Testing Initiates
import numpy as np
import cv2
import os
from tensorflow.keras.utils import to_categorical

# Paths to the test data directory
test_dir = "C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing"

# Define your image dimension and label mapping
img_dim = 128  # Confirming 128x128 as per your training setup
num_classes = 4  # number of classes in your classification task
label_map = {
    'glioma': 0,
    'meningioma': 1,
    'notumor': 2,
    'pituitary': 3
}

def load_test_data(data_dir, img_dim=128):
```

```
    images = []
    labels = []

    for label_name, label_value in label_map.items():
        folder_path = os.path.join(data_dir, label_name)
        print(f"Loading images from: {folder_path}")

        if not os.path.exists(folder_path):
            print(f"Warning: Folder {folder_path} does not exist.")
            continue

        for image_file in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image_file)
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            if image is None:
                print(f"Warning: Image {image_path} could not be loaded.")
                continue

            # Resize to 128x128 as per training and normalize
            image = cv2.resize(image, (img_dim, img_dim)) / 255.0
            images.append(image)
            labels.append(label_value)

    # Convert to numpy arrays and reshape for model input
    images = np.array(images).reshape(-1, img_dim, img_dim, 1)
    labels = np.array(labels)

    return images, labels

# Load the test data
test_images, test_labels = load_test_data(test_dir)
print("Loaded test images shape:", test_images.shape)  # Check if it's (batch_size, 128, 128, 1)
print("Loaded test labels shape:", test_labels.shape)

# Convert test labels to categorical format (one-hot encoding)
test_labels_cat = to_categorical(test_labels, num_classes=num_classes)

# Double-check model's expected input shape
print("Model's expected input shape:", classification_model.input_shape)
```

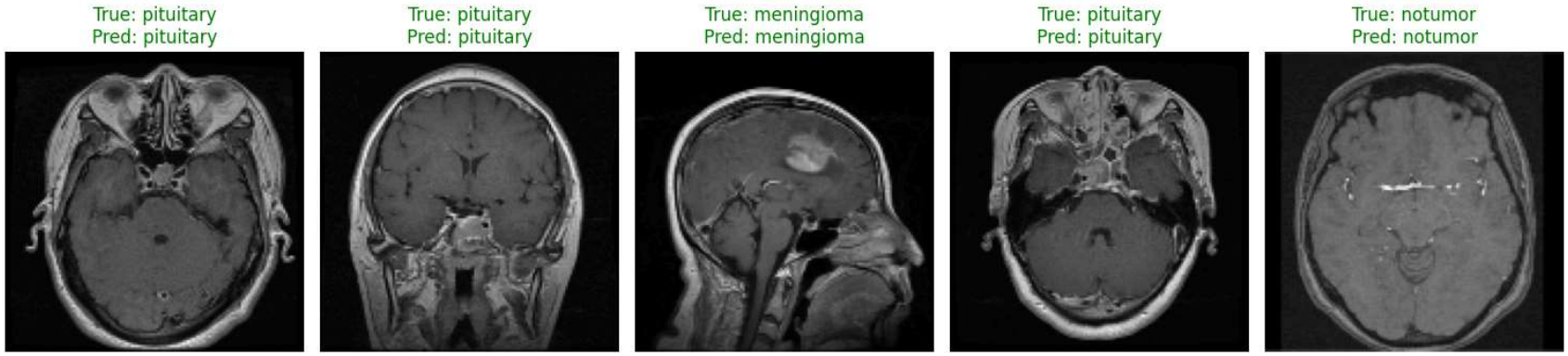
```
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing\glioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing\meningioma
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing\notumor
Loading images from: C:/College/Final Sem/Emerging Trends In Data Technology/Brain MRI/Testing\pituitary
Loaded test images shape: (1311, 128, 128, 1)
Loaded test labels shape: (1311,)
Model's expected input shape: (None, 128, 128, 1)
```

In [18]:
```
# Evaluate the model on the test set
test_loss, test_accuracy = classification_model.evaluate(test_images, test_labels_cat, verbose=1)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
41/41 [==============================] - 4s 101ms/step - loss: 0.2330 - accuracy: 0.9161
Test Accuracy: 0.9161
Test Loss: 0.2330
```

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the reverse mapping for label names
reverse_label_map = {v: k for k, v in label_map.items()}

# Make predictions on the test set
predictions = classification_model.predict(test_images)
predicted_classes = np.argmax(predictions, axis=1)  # Get the index of the class with the highest probability
actual_classes = np.argmax(test_labels_cat, axis=1)  # Convert one-hot encoded labels back to class indices

# Function to display a few test images with predictions
def display_predictions(test_images, actual_classes, predicted_classes, num_images=5):
    plt.figure(figsize=(15, 15))
    for i in range(num_images):
        # Select a random index for visualization
        idx = np.random.randint(0, len(test_images))

        # Display the image
        plt.subplot(1, num_images, i + 1)
        plt.imshow(test_images[idx].reshape(128, 128), cmap='gray')  # Reshape to 128x128 for display
        plt.axis('off')

        # Show actual and predicted labels
        actual_label = reverse_label_map[actual_classes[idx]]
```

```
        predicted_label = reverse_label_map[predicted_classes[idx]]

        # Set title with tumor types
        title_color = "green" if actual_label == predicted_label else "red"
        plt.title(f"True: {actual_label}\nPred: {predicted_label}", color=title_color)

    plt.tight_layout()
    plt.show()

# Display some test images with predictions
display_predictions(test_images, actual_classes, predicted_classes, num_images=5)
```



In [57]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classificatior
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming `classification_model` is your trained model and `test_images` and `test_labels` are your test set.
# Make predictions on the test set
test_predictions = np.argmax(classification_model.predict(test_images), axis=1)

# Calculate accuracy, precision, recall, and F1 score
accuracy = accuracy_score(test_labels, test_predictions)
precision = precision_score(test_labels, test_predictions, average='weighted')
recall = recall_score(test_labels, test_predictions, average='weighted')
f1 = f1_score(test_labels, test_predictions, average='weighted')

# Print the classification report for detailed metrics
print("Classification Report:")
```

```
print(classification_report(test_labels, test_predictions, target_names=['glioma', 'meningioma', 'pituitary', 'no tur

# Display the overall metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

      glioma       0.95      0.88      0.91       300
  meningioma       0.88      0.78      0.82       306
   pituitary       0.91      1.00      0.95       405
    no tumor       0.93      0.99      0.96       300

    accuracy                           0.92      1311
   macro avg       0.92      0.91      0.91      1311
weighted avg       0.92      0.92      0.91      1311

Accuracy: 0.9161
Precision: 0.9154
Recall: 0.9161
F1 Score: 0.9140
```

In [ ]:
```
#Segmentation
```

In [21]:
```
!pip install scikit-image
```

```
Collecting scikit-image
  Downloading scikit_image-0.21.0-cp38-cp38-win_amd64.whl.metadata (14 kB)
Requirement already satisfied: numpy>=1.21.1 in c:\college\anaconda\envs\myenv\lib\site-packages (from scikit-image)
(1.23.5)
Requirement already satisfied: scipy>=1.8 in c:\college\anaconda\envs\myenv\lib\site-packages (from scikit-image) (1.
10.1)
Collecting networkx>=2.8 (from scikit-image)
  Downloading networkx-3.1-py3-none-any.whl.metadata (5.3 kB)
Requirement already satisfied: pillow>=9.0.1 in c:\college\anaconda\envs\myenv\lib\site-packages (from scikit-image)
(10.4.0)
Collecting imageio>=2.27 (from scikit-image)
  Downloading imageio-2.35.1-py3-none-any.whl.metadata (4.9 kB)
Collecting tifffile>=2022.8.12 (from scikit-image)
  Downloading tifffile-2023.7.10-py3-none-any.whl.metadata (31 kB)
Collecting PyWavelets>=1.1.1 (from scikit-image)
  Downloading PyWavelets-1.4.1-cp38-cp38-win_amd64.whl.metadata (1.9 kB)
Requirement already satisfied: packaging>=21 in c:\college\anaconda\envs\myenv\lib\site-packages (from scikit-image)
(24.1)
Collecting lazy_loader>=0.2 (from scikit-image)
  Downloading lazy_loader-0.4-py3-none-any.whl.metadata (7.6 kB)
Downloading scikit_image-0.21.0-cp38-cp38-win_amd64.whl (22.7 MB)
   ---------------------------------------- 0.0/22.7 MB ? eta -:--:--
   --- ------------------------------------ 1.8/22.7 MB 9.1 MB/s eta 0:00:03
   ------- -------------------------------- 4.2/22.7 MB 10.5 MB/s eta 0:00:02
   ------------ --------------------------- 6.8/22.7 MB 10.7 MB/s eta 0:00:02
   --------------- ------------------------ 9.4/22.7 MB 11.1 MB/s eta 0:00:02
   ------------------- -------------------- 11.0/22.7 MB 11.1 MB/s eta 0:00:02
   ---------------------- ----------------- 12.8/22.7 MB 10.1 MB/s eta 0:00:01
   ------------------------ --------------- 14.7/22.7 MB 9.8 MB/s eta 0:00:01
   ---------------------------- ---------- 16.5/22.7 MB 9.7 MB/s eta 0:00:01
   ------------------------------- -------- 18.6/22.7 MB 9.7 MB/s eta 0:00:01
   ---------------------------------- ----- 21.0/22.7 MB 9.8 MB/s eta 0:00:01
   ----------------------------------------  22.5/22.7 MB 10.0 MB/s eta 0:00:01
   ---------------------------------------- 22.7/22.7 MB 9.6 MB/s eta 0:00:00
Downloading imageio-2.35.1-py3-none-any.whl (315 kB)
Downloading lazy_loader-0.4-py3-none-any.whl (12 kB)
Downloading networkx-3.1-py3-none-any.whl (2.1 MB)
   ---------------------------------------- 0.0/2.1 MB ? eta -:--:--
   ---------------------------------------- 2.1/2.1 MB 12.9 MB/s eta 0:00:00
Downloading PyWavelets-1.4.1-cp38-cp38-win_amd64.whl (4.2 MB)
   ---------------------------------------- 0.0/4.2 MB ? eta -:--:--
   ------------------------ --------------- 2.6/4.2 MB 12.6 MB/s eta 0:00:01
```

```
   ---------------------------------------- 4.2/4.2 MB 11.9 MB/s eta 0:00:00
Downloading tifffile-2023.7.10-py3-none-any.whl (220 kB)
Installing collected packages: tifffile, PyWavelets, networkx, lazy_loader, imageio, scikit-image
Successfully installed PyWavelets-1.4.1 imageio-2.35.1 lazy_loader-0.4 networkx-3.1 scikit-image-0.21.0 tifffile-202
3.7.10
```

In [54]:

```python
import cv2
import numpy as np
from skimage import morphology, measure
from sklearn.cluster import KMeans

def normalize_image(image):
    # Normalize the image intensities to range [0, 1]
    return (image - np.min(image)) / (np.max(image) - np.min(image))

def apply_kmeans(image, n_clusters=2):
    # Flatten the image and apply KMeans clustering
    flat_image = image.flatten().reshape(-1, 1)
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(flat_image)
    clustered = kmeans.labels_.reshape(image.shape)
    return clustered

def threshold_image(clustered_image):
    # Convert clustered image to binary based on cluster label
    return (clustered_image == 1).astype(np.uint8)  # Assuming 1 is the tumor cluster label

def apply_morphology(binary_image):
    # Perform morphological operations to remove noise and enhance the tumor area
    morphed = morphology.binary_opening(binary_image, morphology.disk(3))
    morphed = morphology.binary_closing(morphed, morphology.disk(3))
    return morphed

def label_regions(morphed_image):
    # Label connected regions in the morphed binary image
    labeled_image, num_regions = measure.label(morphed_image, return_num=True, connectivity=2)
    return labeled_image, num_regions

def create_tumor_mask(labeled_image, tumor_cluster_label=1):
    # Create a mask for the tumor region based on labeled regions
    tumor_mask = (labeled_image == tumor_cluster_label).astype(np.uint8)
    return tumor_mask
```

```
def apply_mask_on_image(image, mask):
    # Apply the tumor mask on the original image (overlay)
    return image * mask
```

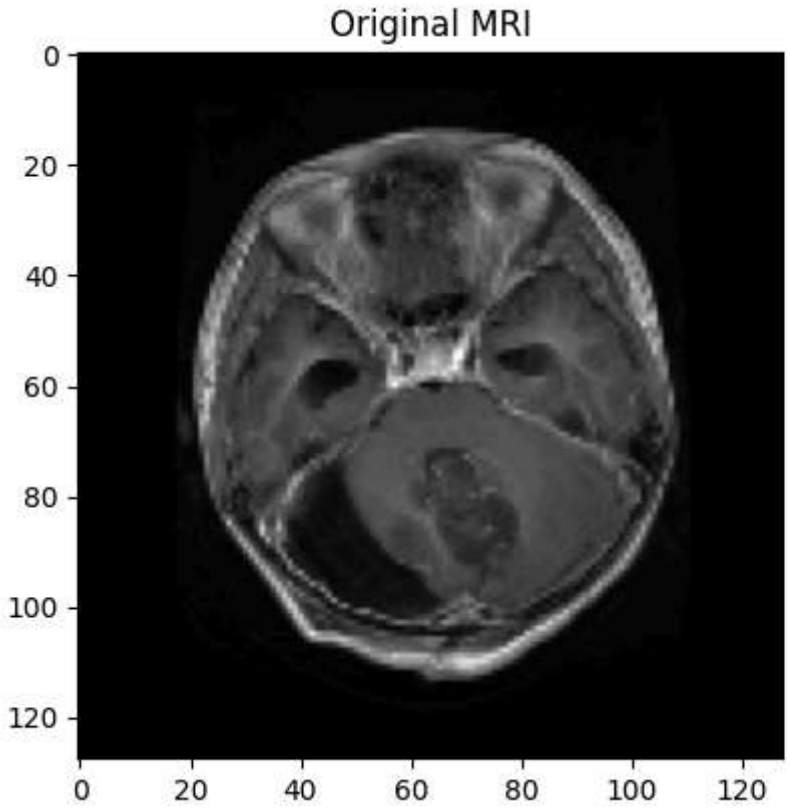
In [ ]:
```
def segment_tumor(image):
    # Step 1: Normalize the image
    norm_image = normalize_image(image)

    # Step 2: Apply K-means clustering
    clustered_image = apply_kmeans(norm_image, n_clusters=2)  # Assuming 2 clusters: tumor and non-tumor

    # Step 3: Threshold the clustered image
    binary_image = threshold_image(clustered_image)

    # Step 4: Morphological operations
    morphed_image = apply_morphology(binary_image)

    # Step 5: Label different regions
    labeled_image, num_regions = label_regions(morphed_image)

    # Step 6: Create tumor mask (assuming tumor is labeled with cluster '1' here)
    tumor_mask = create_tumor_mask(labeled_image, tumor_cluster_label=1)

    # Step 7: Apply mask on the original image
    final_segmented_image = apply_mask_on_image(image, tumor_mask)

    return final_segmented_image
```

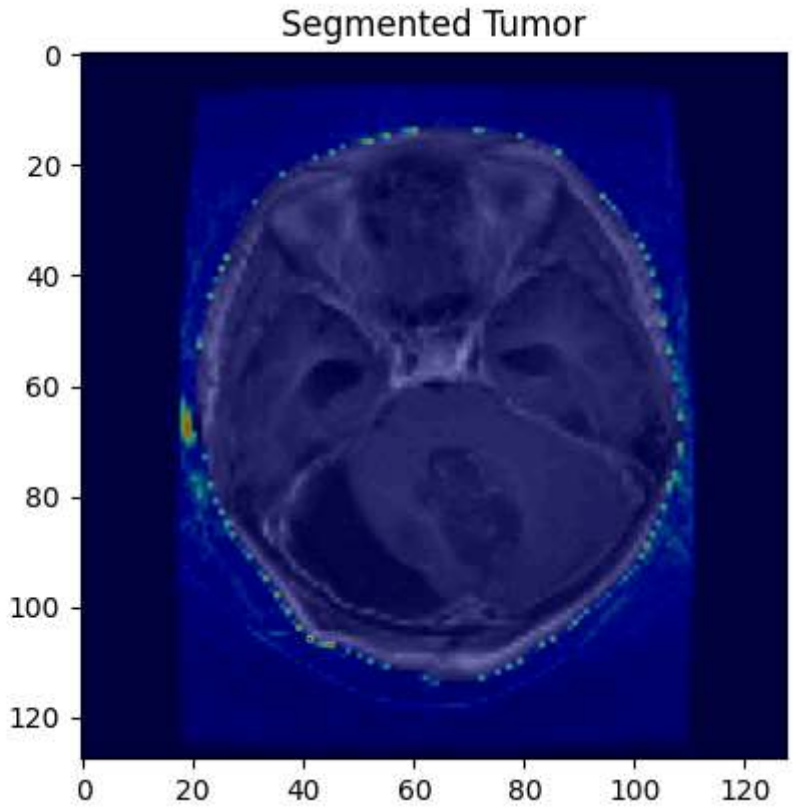
In [56]:
```
import matplotlib.pyplot as plt
import numpy as np

# Assuming your trained classification model is named 'classification_model'
# and the 'segment_tumor' function is already defined as shown in previous messages.

# Define the tumor type mapping
tumor_types = {0: "glioma", 1: "meningioma", 2: "pituitary", 3: "no tumor"}

def classify_and_segment(image):
    # Step 1: Classify the tumor type
    prediction = classification_model.predict(image.reshape(1, img_dim, img_dim, 1))
    tumor_type = np.argmax(prediction)  # Gets the predicted class label
```

```
    # Output classification result
    classified_type = tumor_types[tumor_type]
    print(f"Classification Result: {classified_type}")

    # Step 2: Proceed to segmentation only if a tumor is detected
    if classified_type != "no tumor":
        print("Tumor detected. Starting segmentation...")

        # Step 3: Segment the tumor
        tumor_mask = segment_tumor(image)  # Use the function to get the tumor mask

        # Step 4: Display segmented tumor overlay
        plt.figure(figsize=(10, 5))

        # Original image
        plt.subplot(1, 2, 1)
        plt.imshow(image.squeeze(), cmap='gray')
        plt.title("Original MRI")

        # Segmented tumor overlay
        plt.subplot(1, 2, 2)
        plt.imshow(image.squeeze(), cmap='gray')
        plt.imshow(tumor_mask, cmap='jet', alpha=0.5)  # Overlay mask
        plt.title("Segmented Tumor")

        plt.show()

        return classified_type, tumor_mask
    else:
        print("No tumor detected. Skipping segmentation.")
        return classified_type, None

# Loop over test images and classify/segment each one
for i, test_image in enumerate(test_images[:15]):  # Limit to 10 images for display purposes
    print(f"\nProcessing test image {i + 1}...")
    classify_and_segment(test_image.squeeze())
```

```
Processing test image 1...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

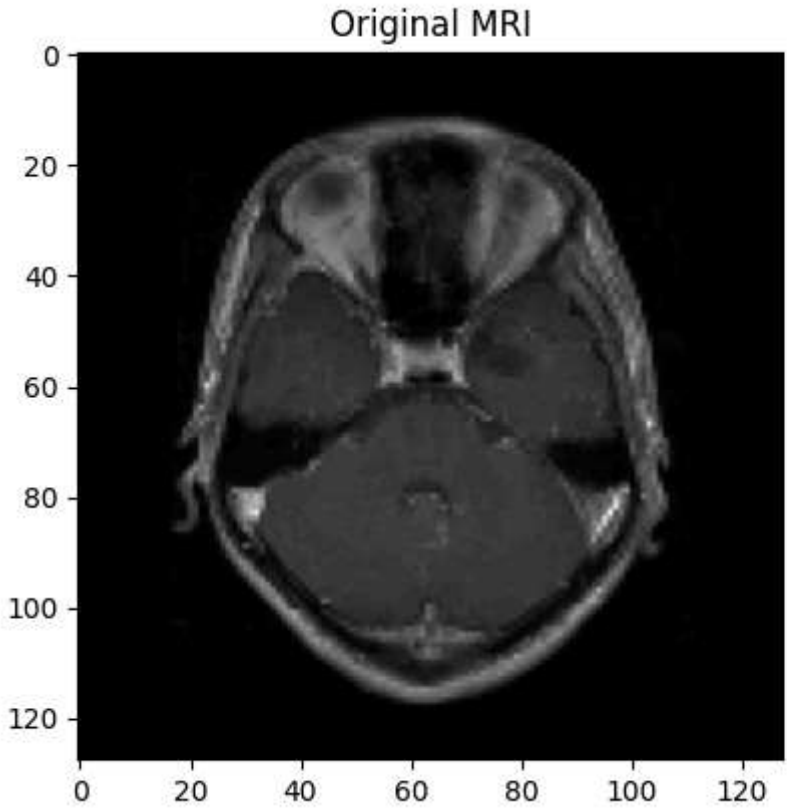
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

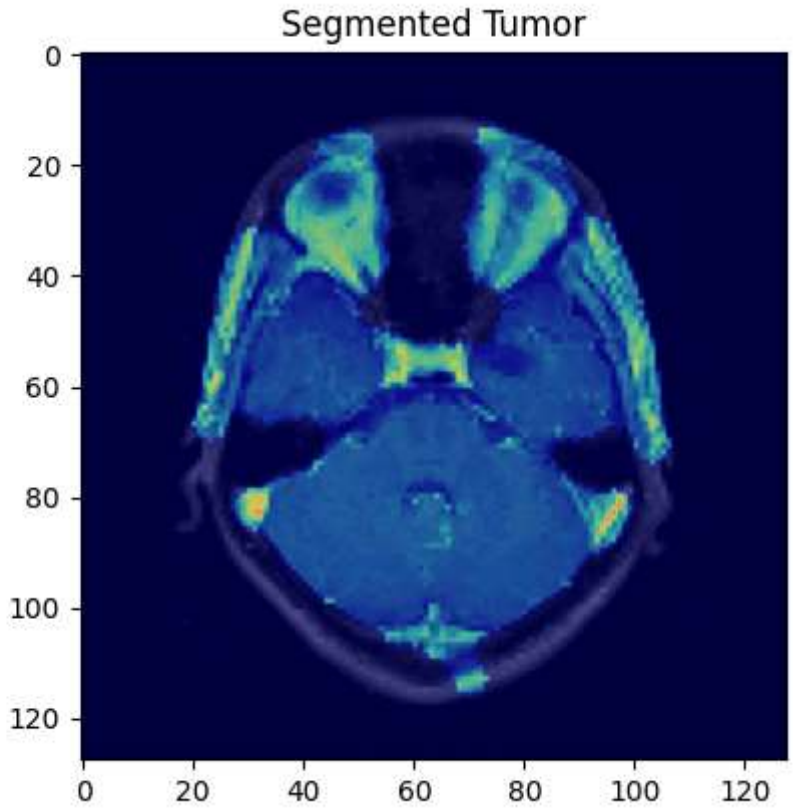





```
Processing test image 2...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

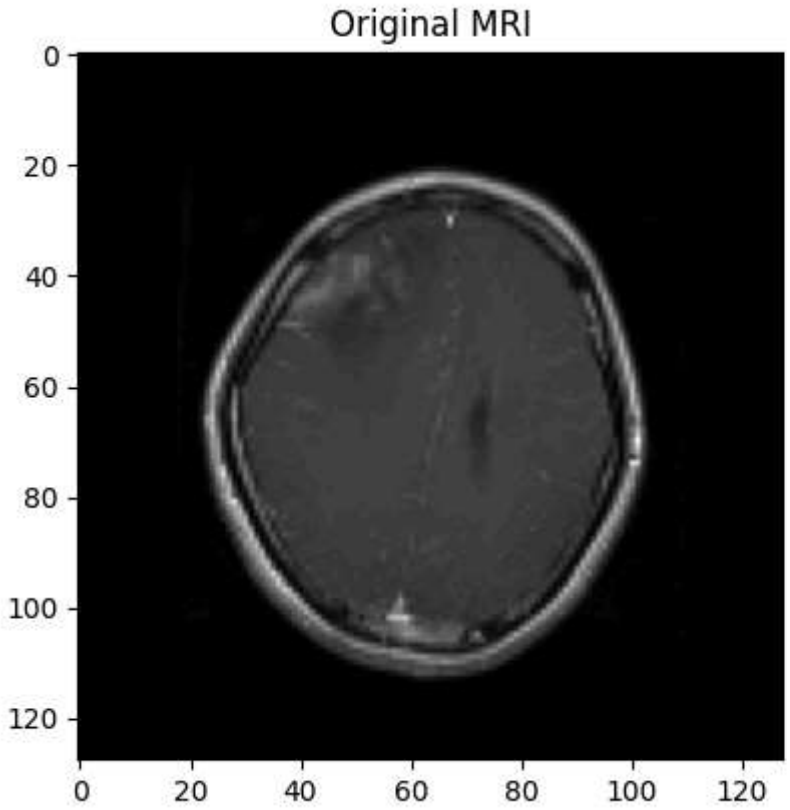
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

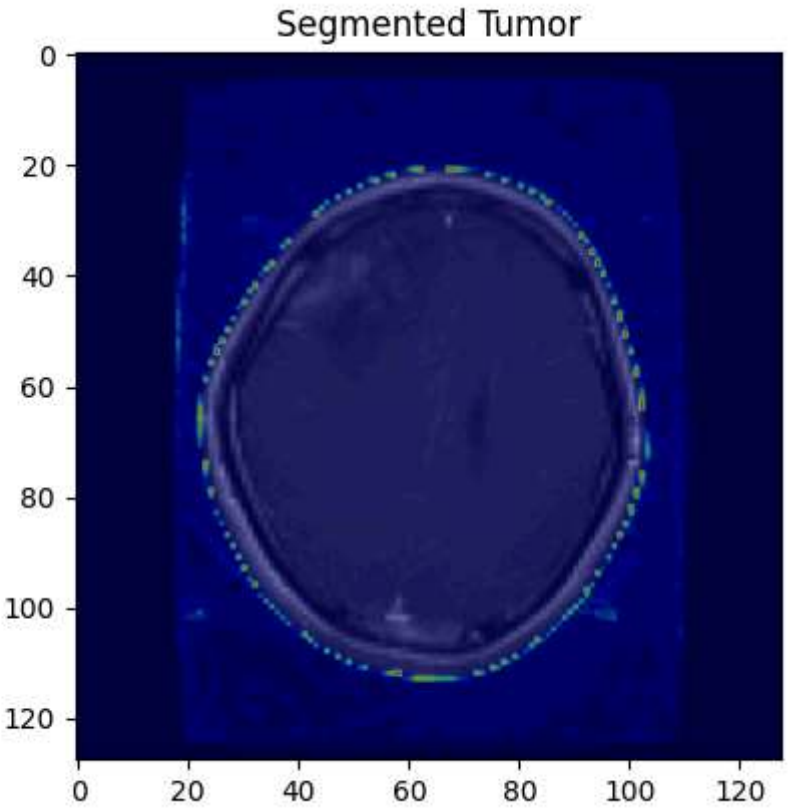





```
Processing test image 3...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

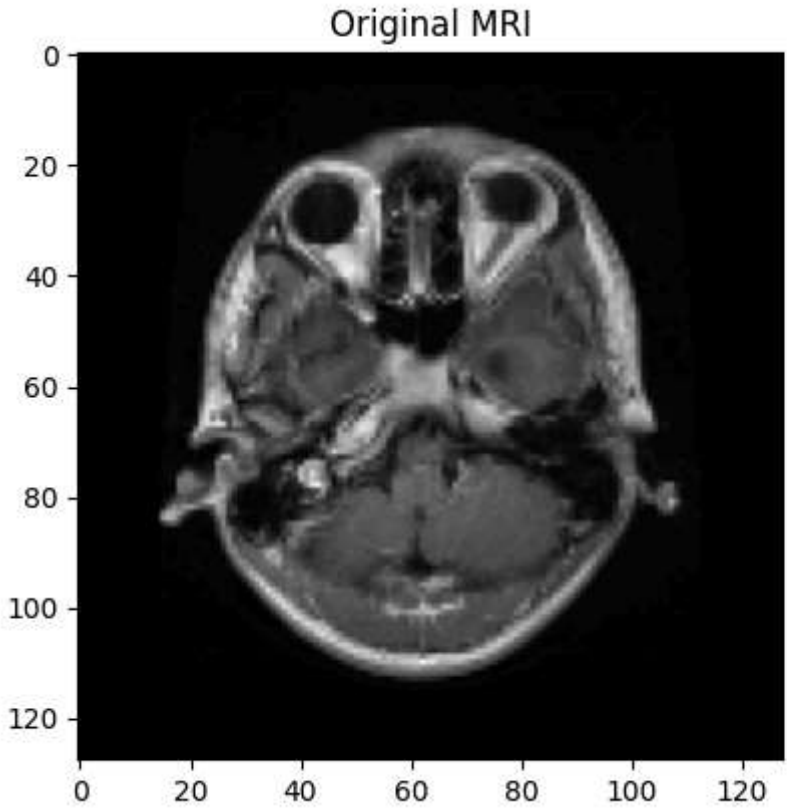
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

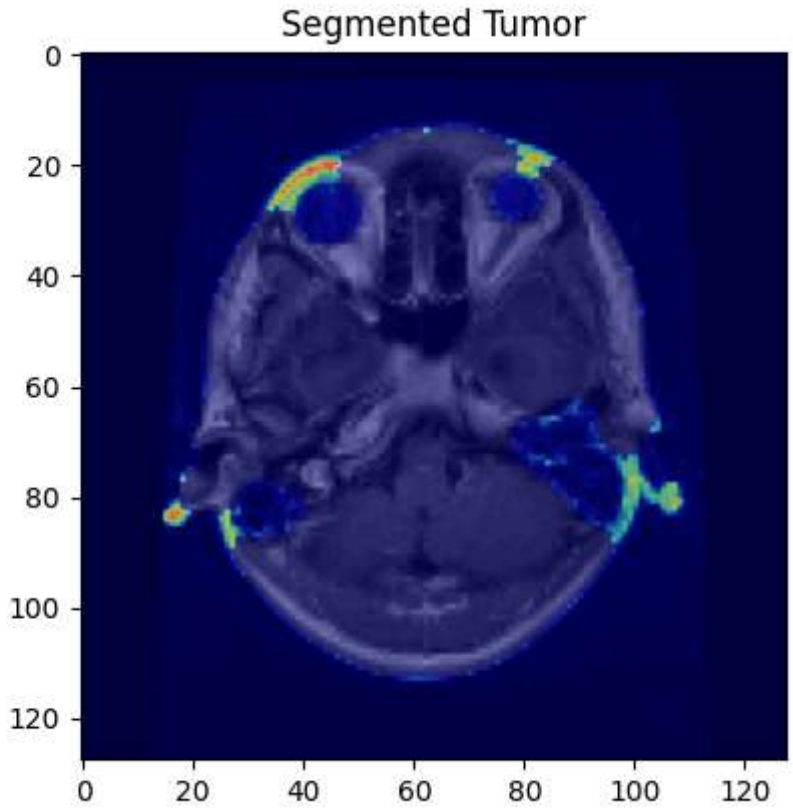





```
Processing test image 4...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

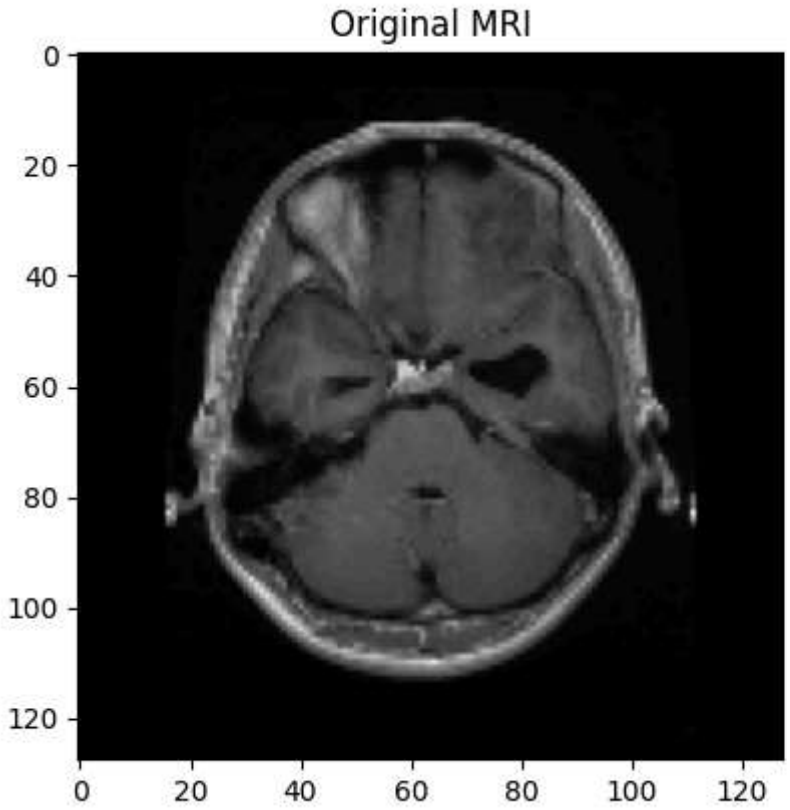
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

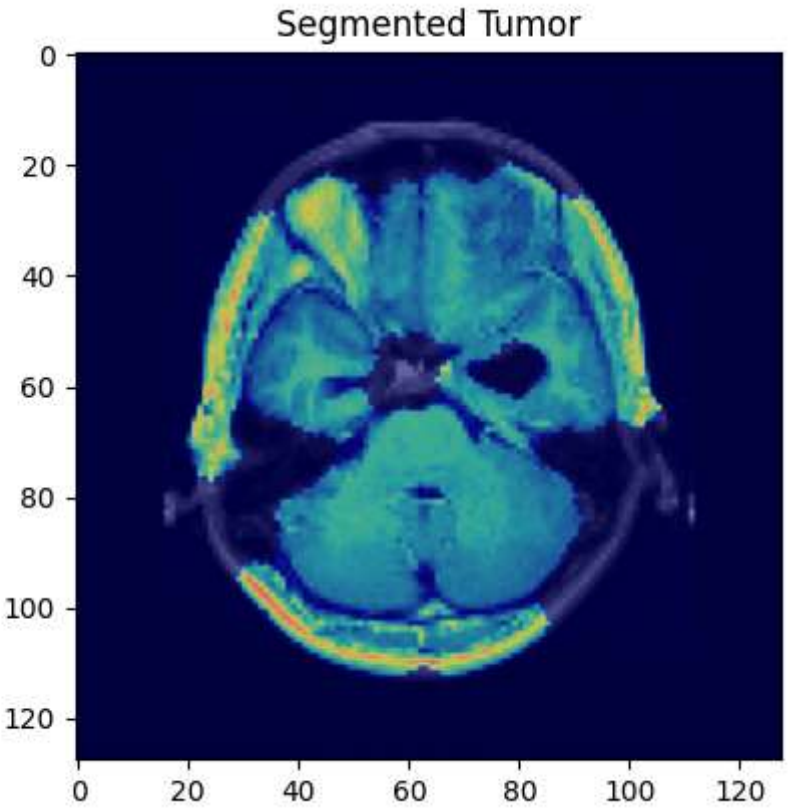





```
Processing test image 5...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

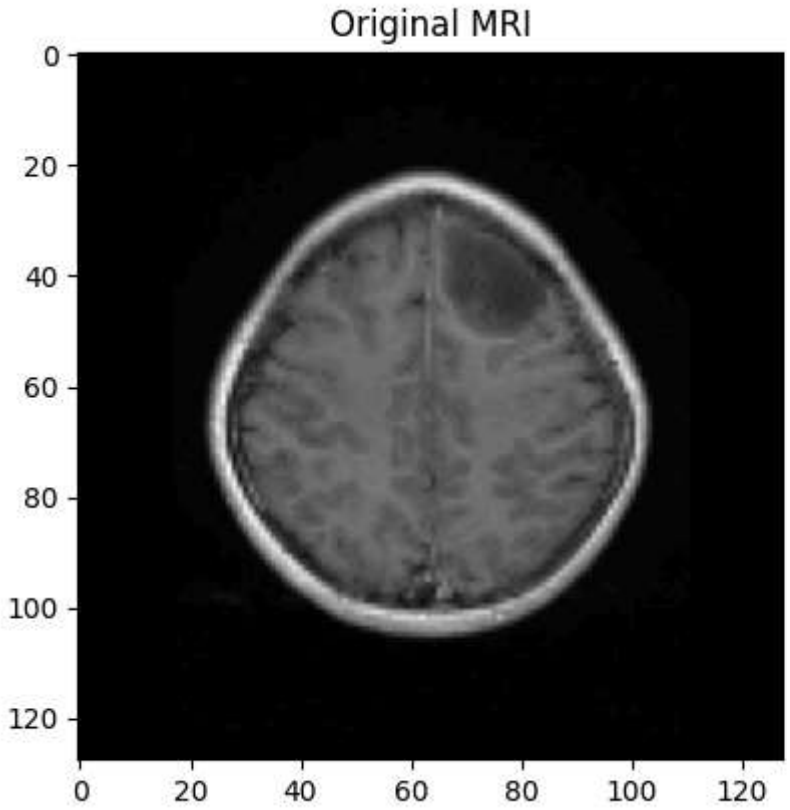
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

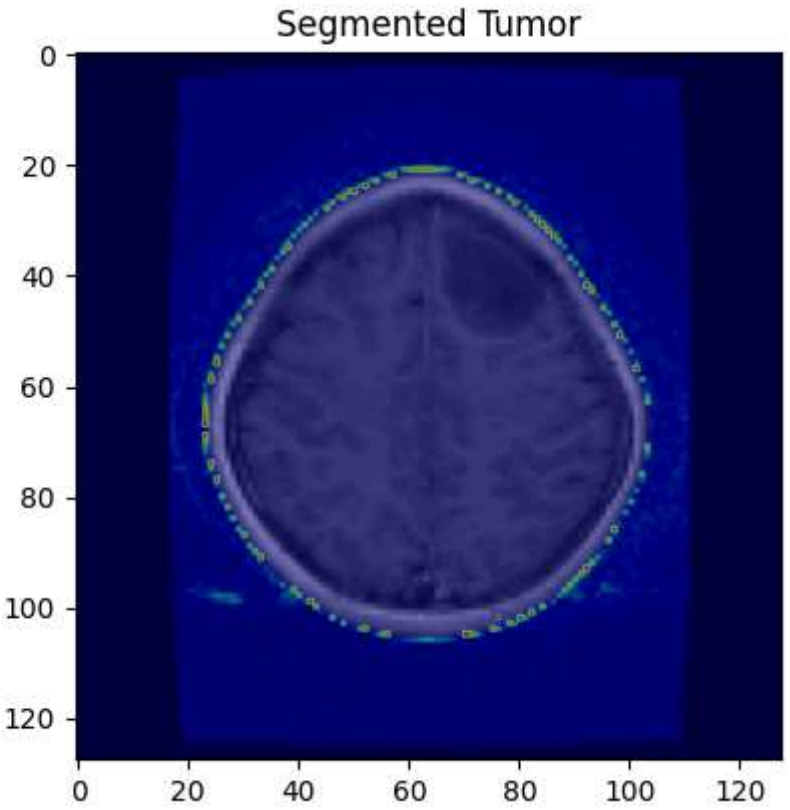





```
Processing test image 6...
Classification Result: meningioma
Tumor detected. Starting segmentation...
```

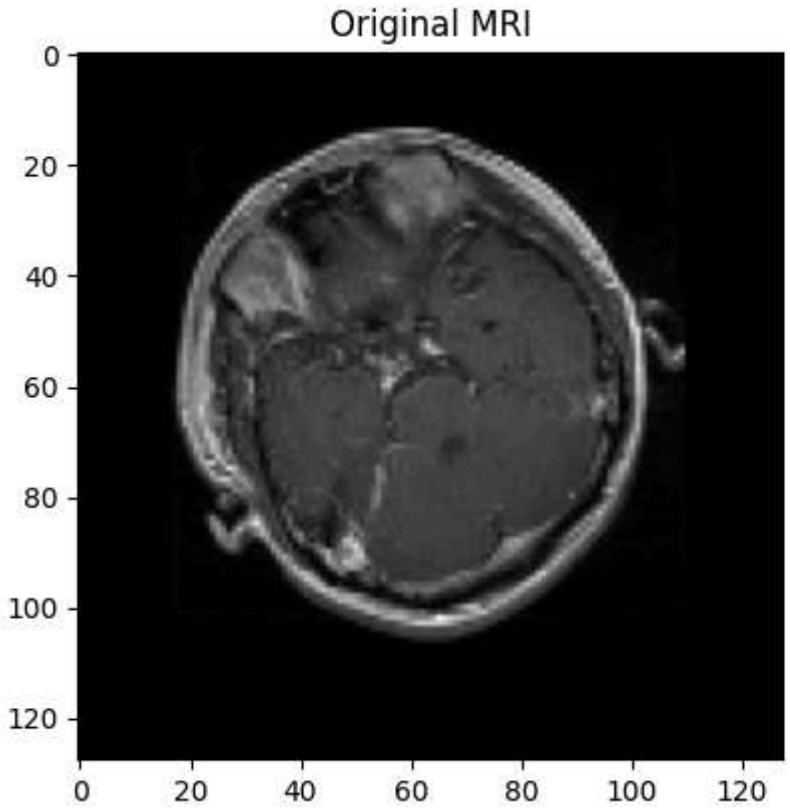
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

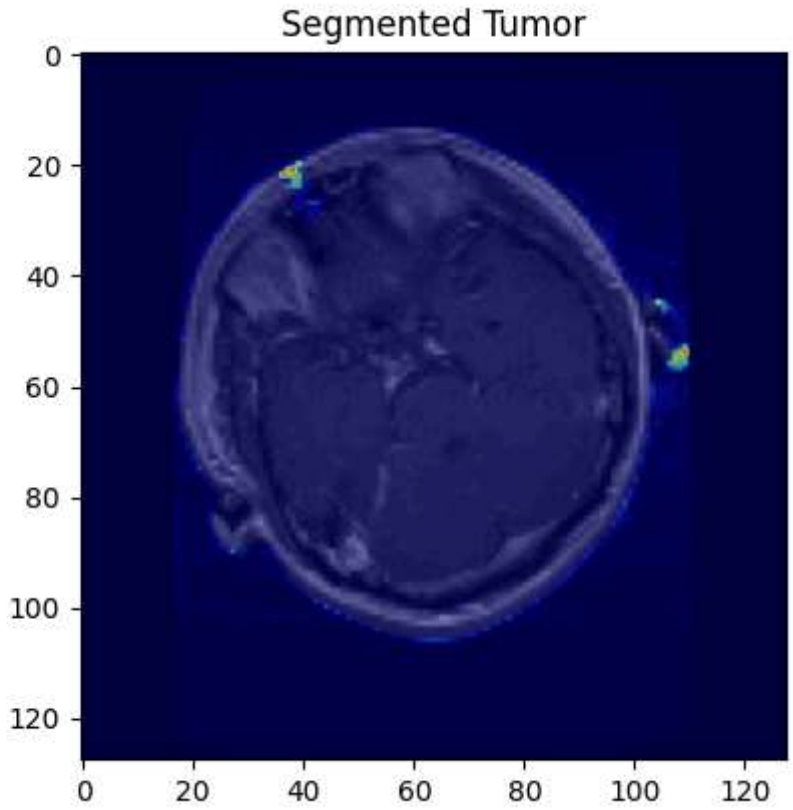





```
Processing test image 7...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

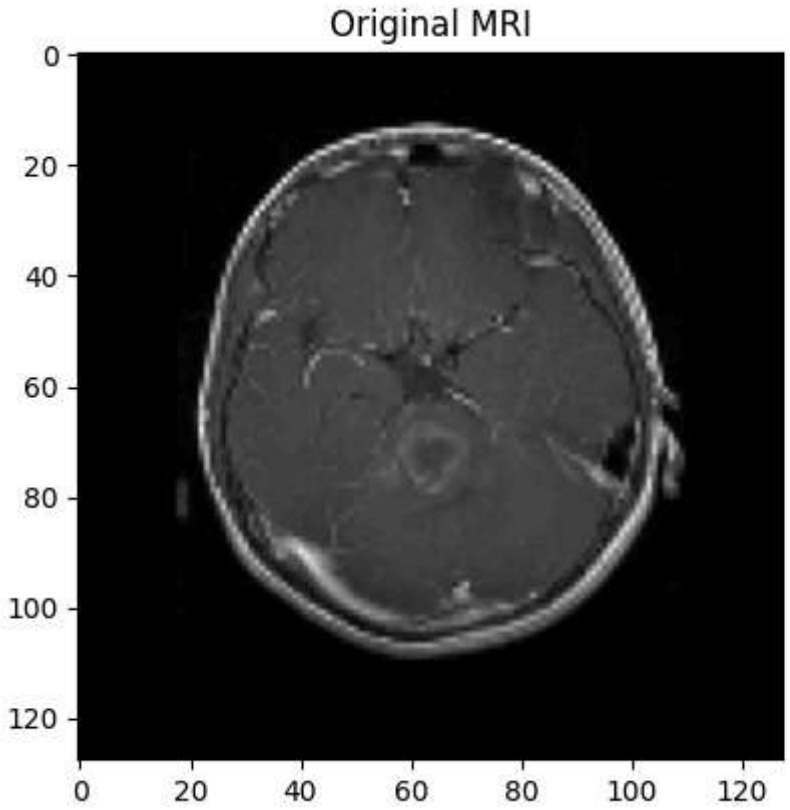
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

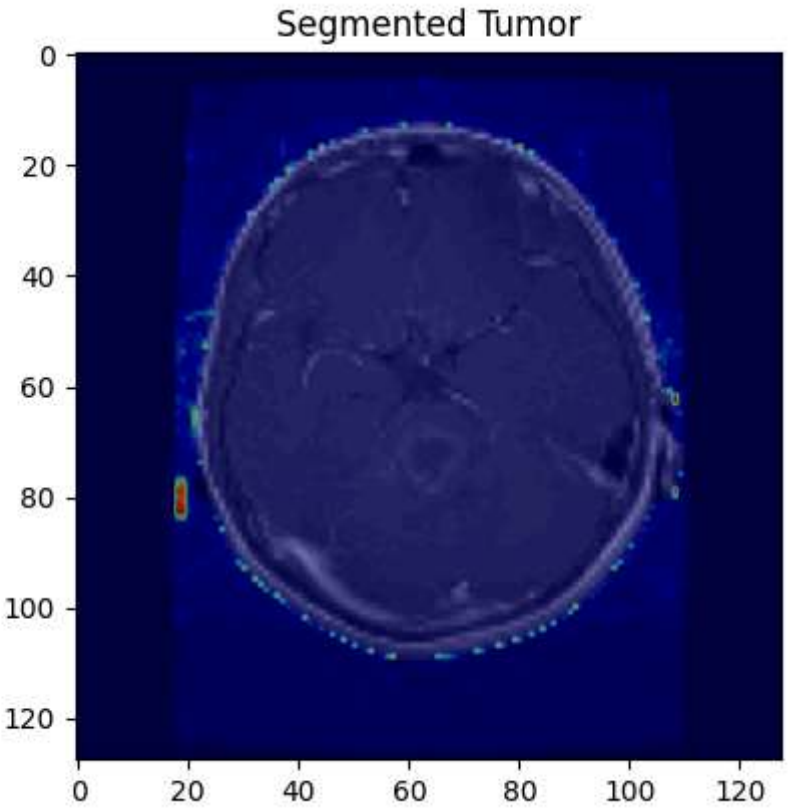





```
Processing test image 8...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

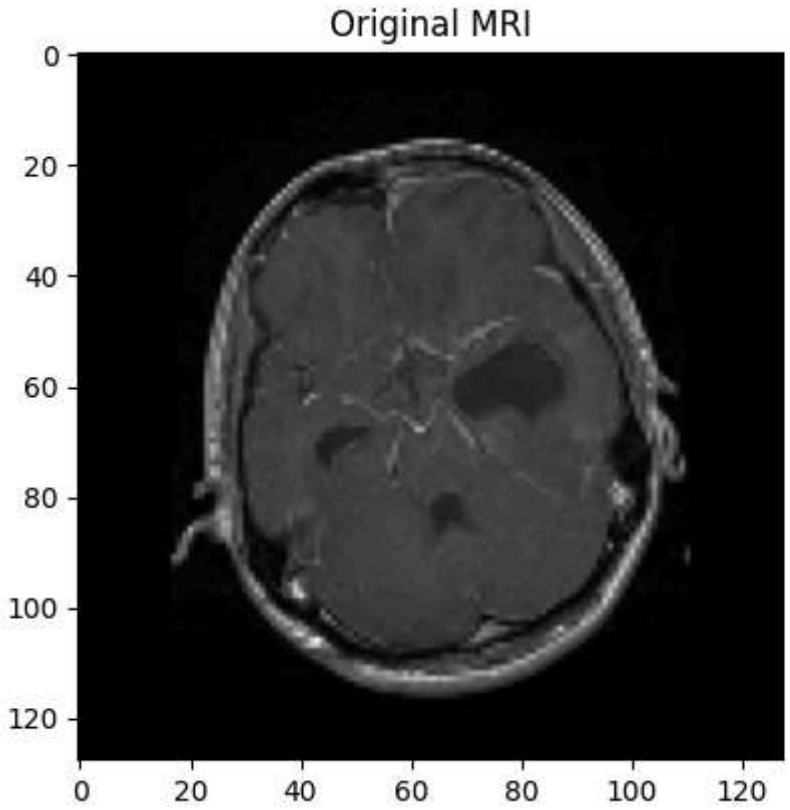
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

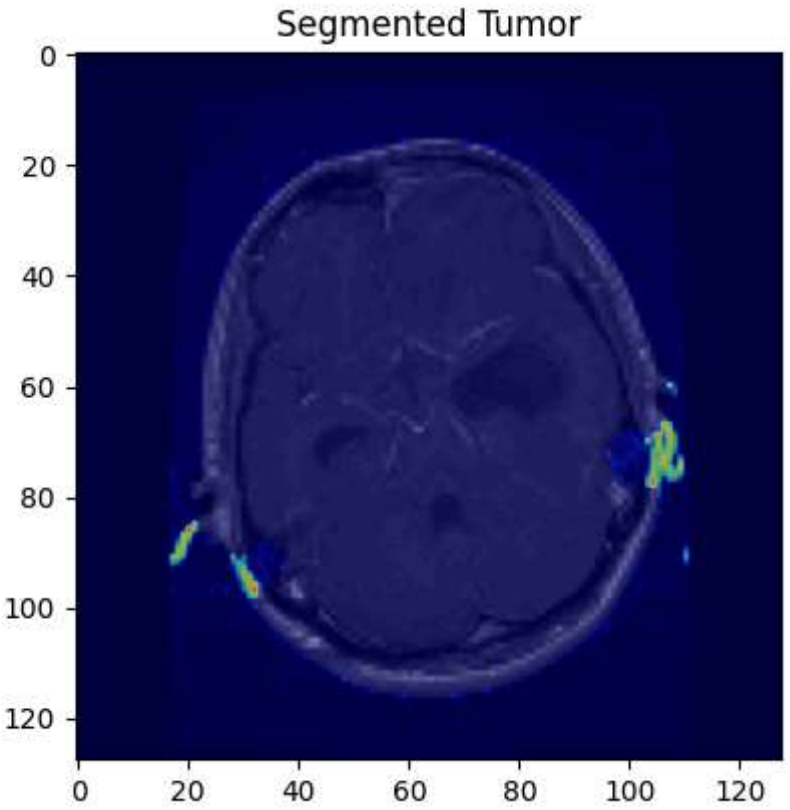





```
Processing test image 9...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

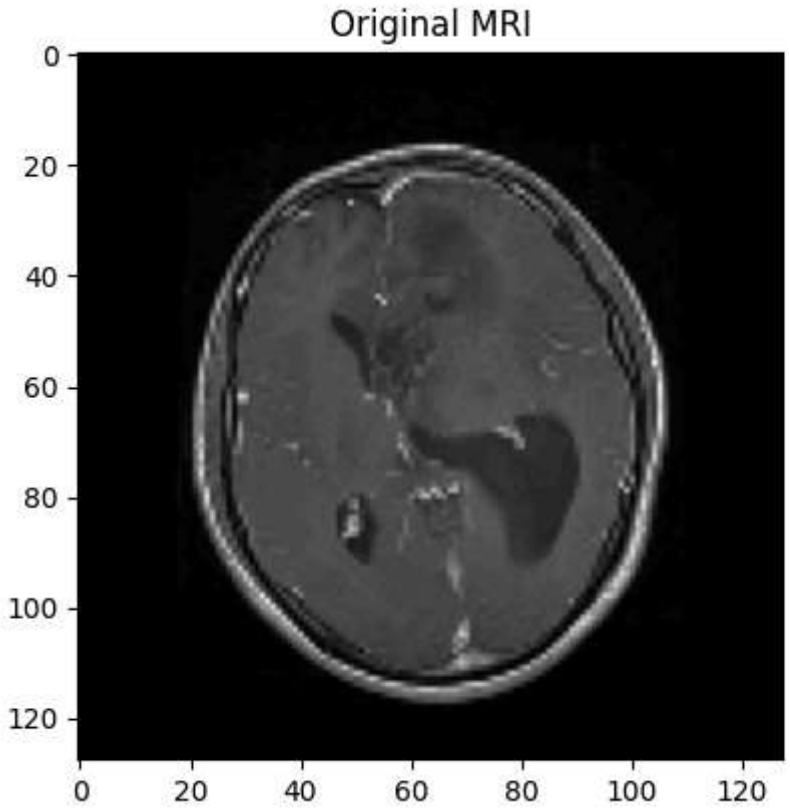
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

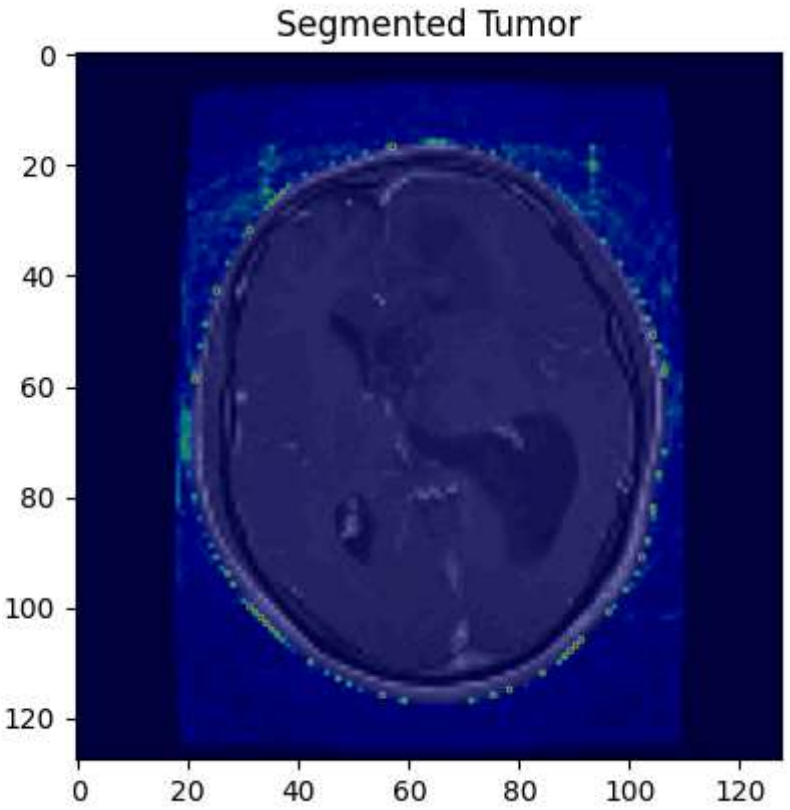





```
Processing test image 10...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

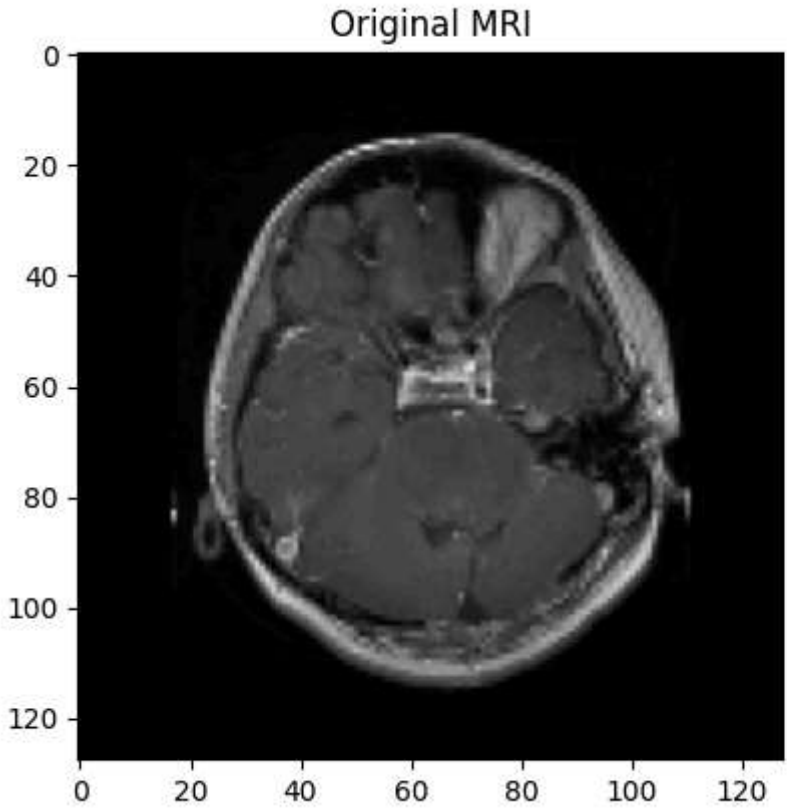
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

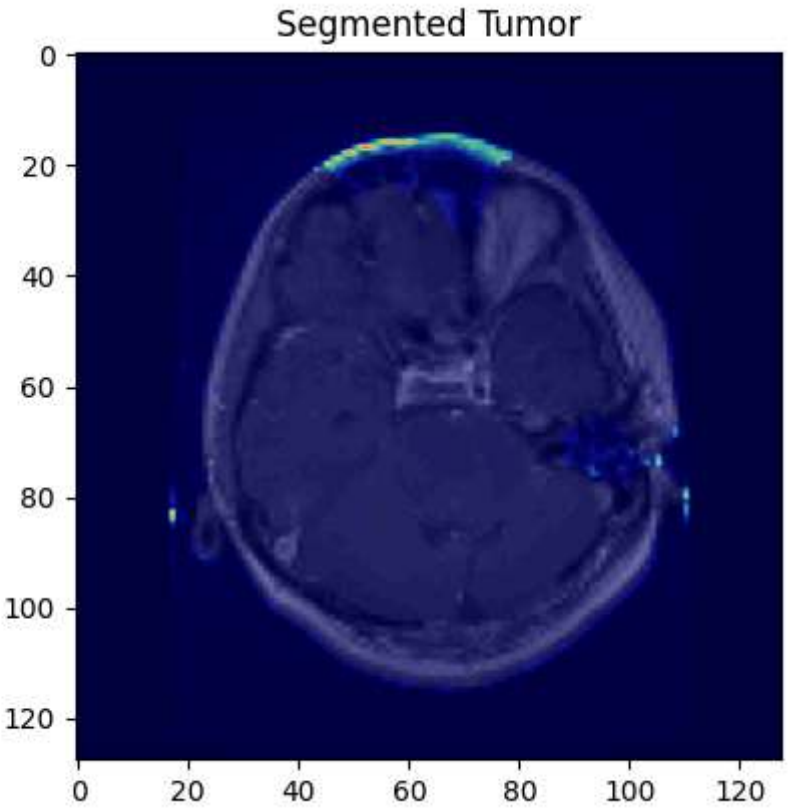





```
Processing test image 11...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

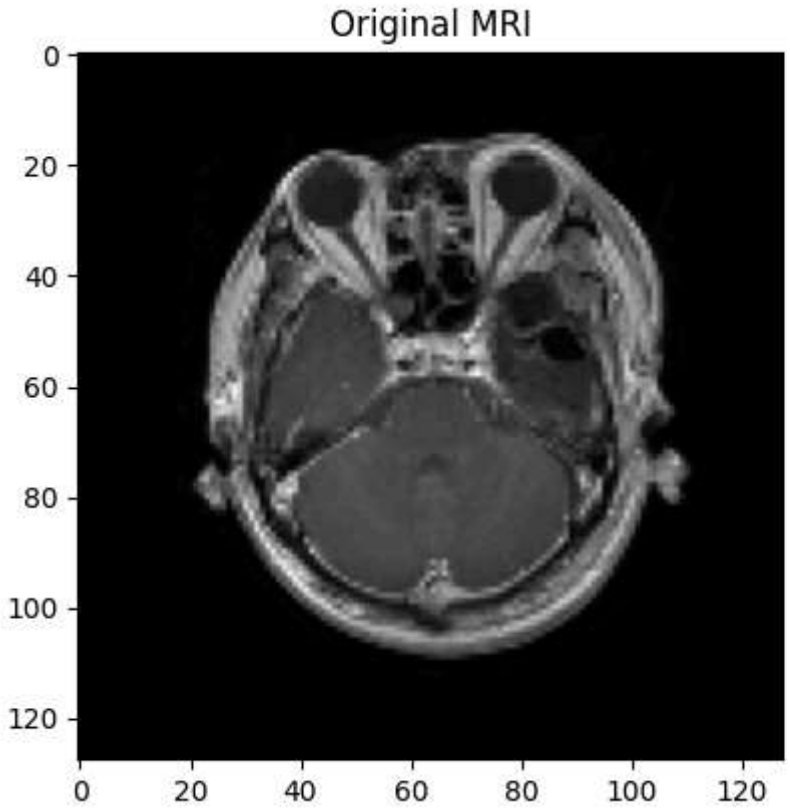
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

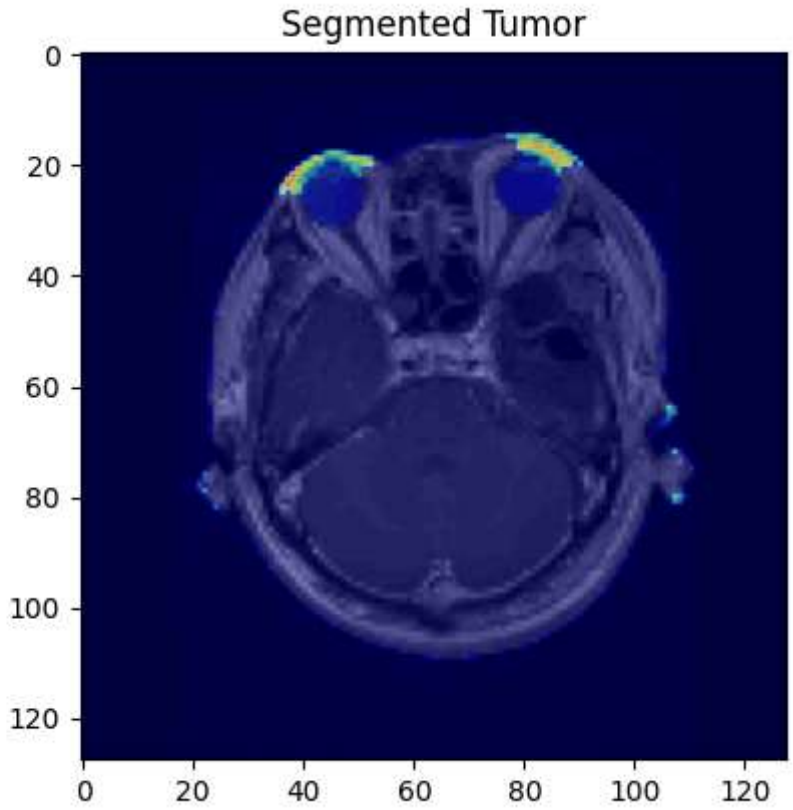





```
Processing test image 12...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

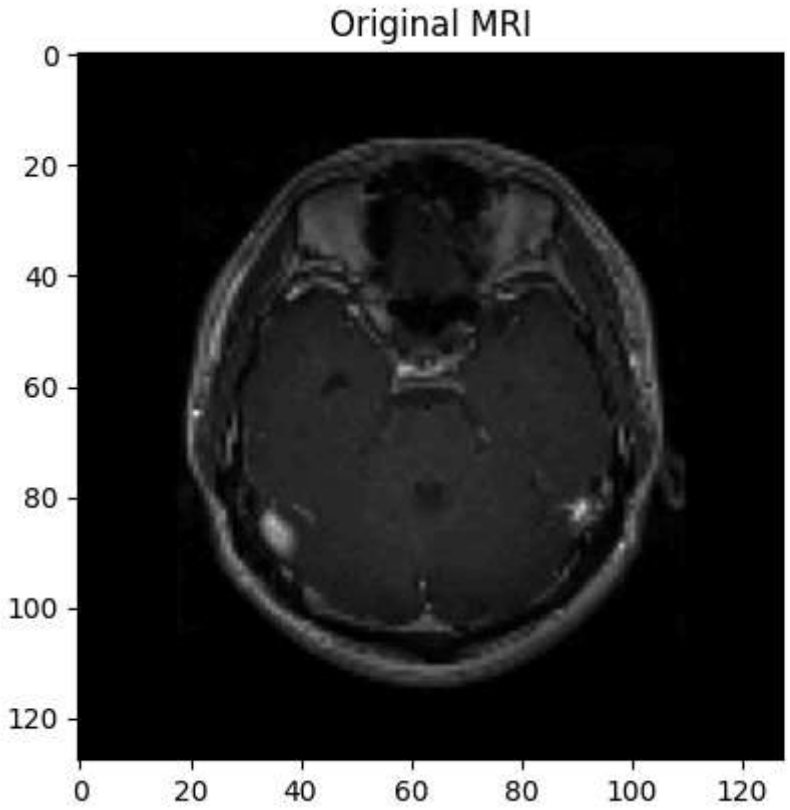
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

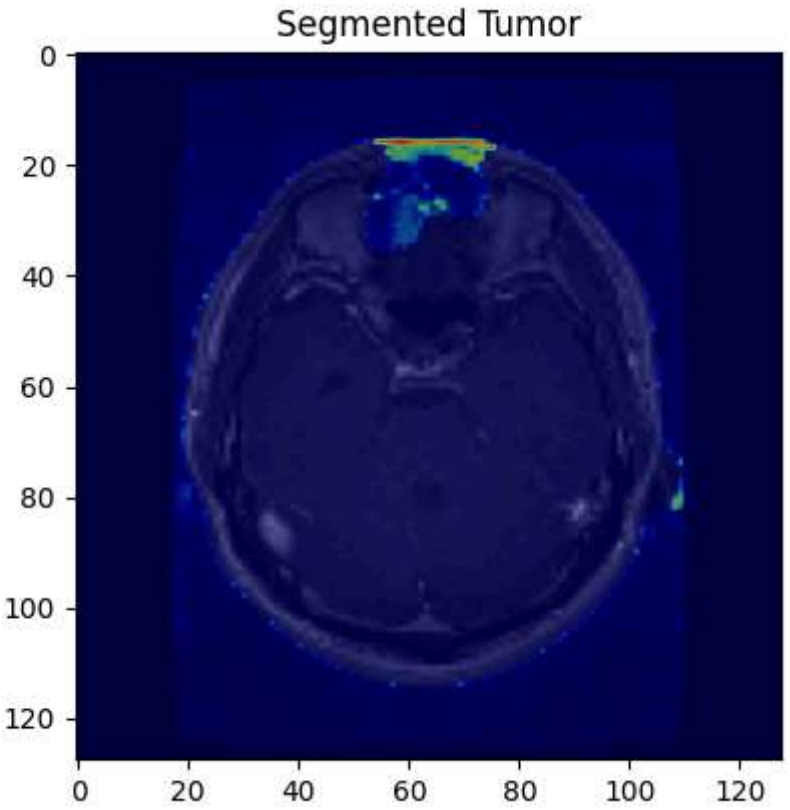





```
Processing test image 13...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

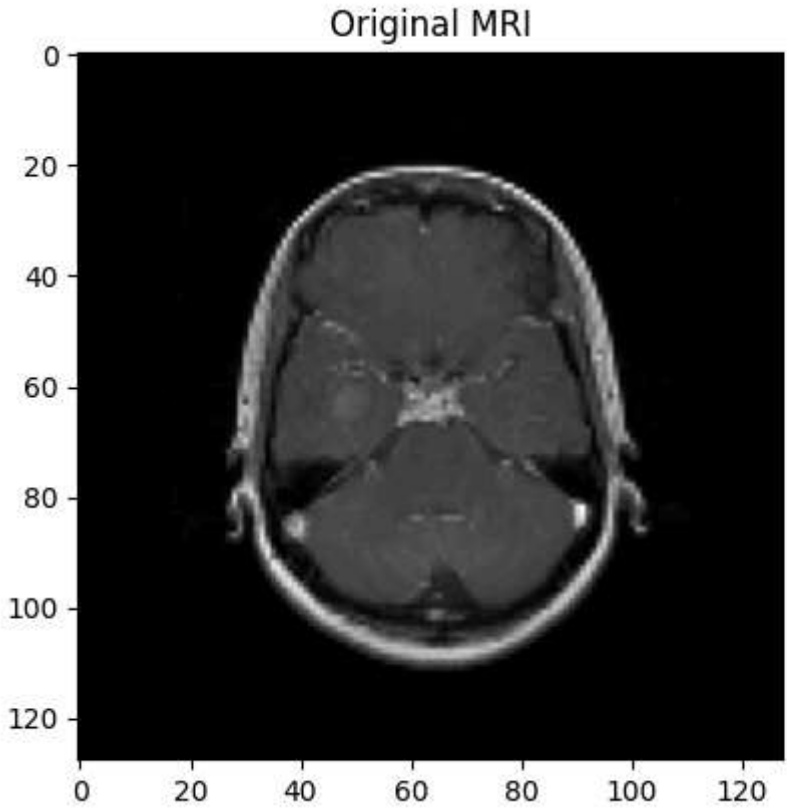
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

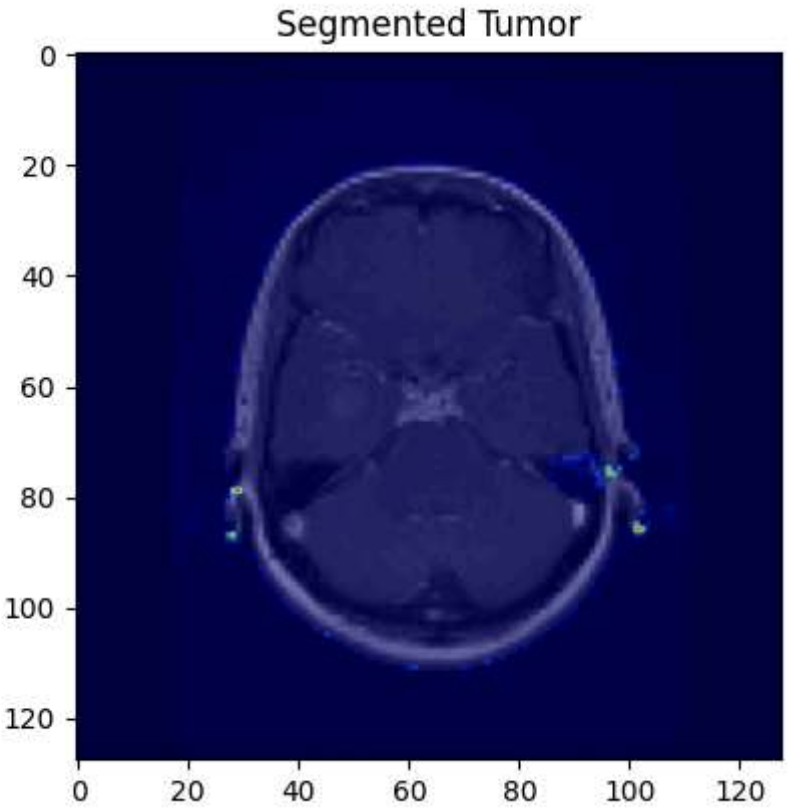





```
Processing test image 14...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

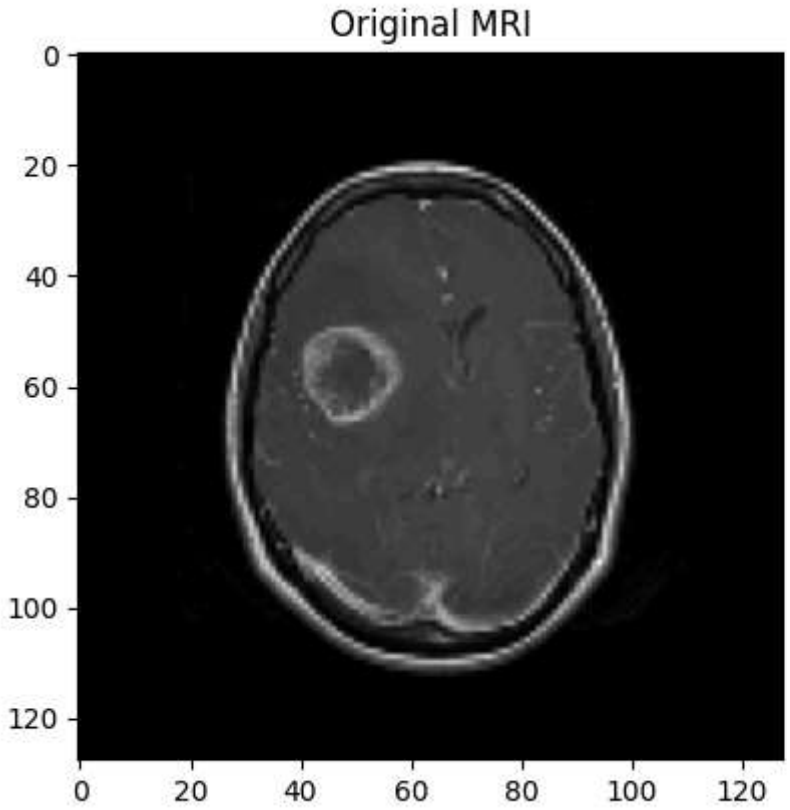
```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

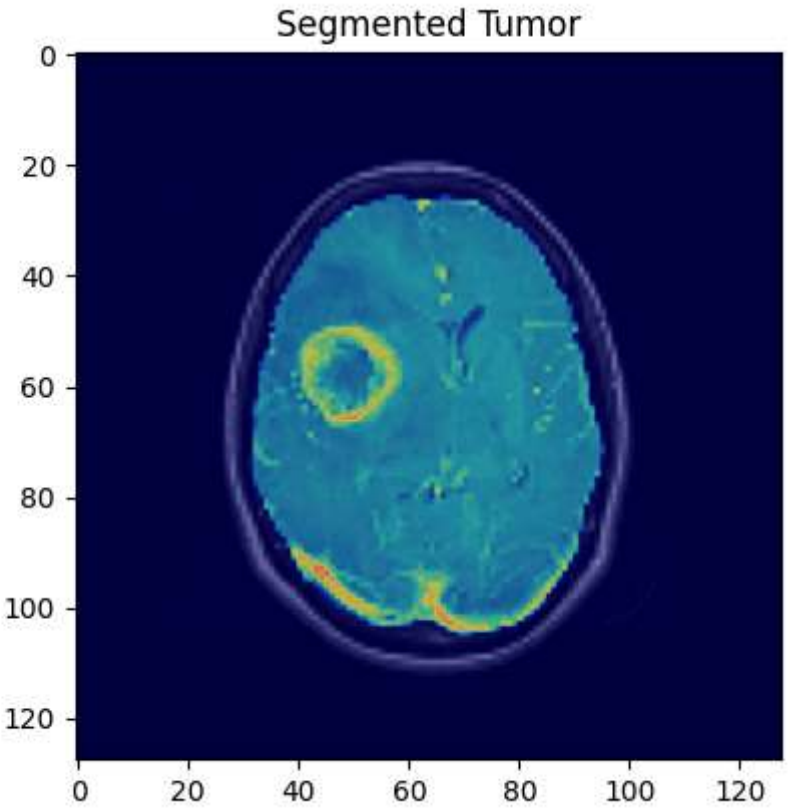





```
Processing test image 15...
Classification Result: glioma
Tumor detected. Starting segmentation...
```

```
C:\College\Anaconda\envs\myenv\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```


Original MRI
0
20
40
60
80
100
120
0
20
40
60
80
100
120


Segmented Tumor
0
20
40
60
80
100
120
0
20
40
60
80
100
120

In [ ]:

In [ ]: