BRAIN MRI SEGMENTATION

- ***Pre-processing and Model building***

In [ ]:
```
#import libraries
import os
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

#importing Deep learning libraries
import tensorflow as tf
from tensorflow.keras.utils import normalize
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dropout, UpSampling2D,
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
```

In [ ]:
```
dataset_path='/content/drive/MyDrive/project_datasets/kaggle_3m'
```

In [ ]:
```
#load image mask pairs
image_paths = []
mask_paths = []

for patient_folder in os.listdir(dataset_path):
    folder_path = os.path.join(dataset_path, patient_folder)
    if os.path.isdir(folder_path):
        for file in os.listdir(folder_path):
            if file.endswith(".tif") and "_mask" not in file:
                img_path = os.path.join(folder_path, file)
                mask_path = img_path.replace(".tif", "_mask.tif")
                if os.path.exists(mask_path):
                    image_paths.append(img_path)
                    mask_paths.append(mask_path)

print(f"Total images: {len(image_paths)}, Total masks: {len(mask_paths)}")
```

```
Total images: 3929, Total masks: 3929
```

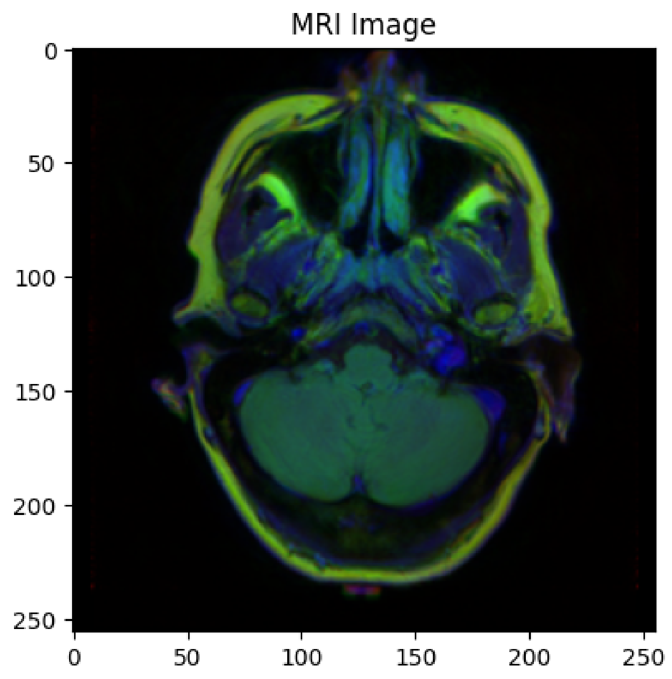
In [ ]:
```
x=np.load('/content/drive/MyDrive/project_datasets/x.npy')
y=np.load('/content/drive/MyDrive/project_datasets/y.npy')
```

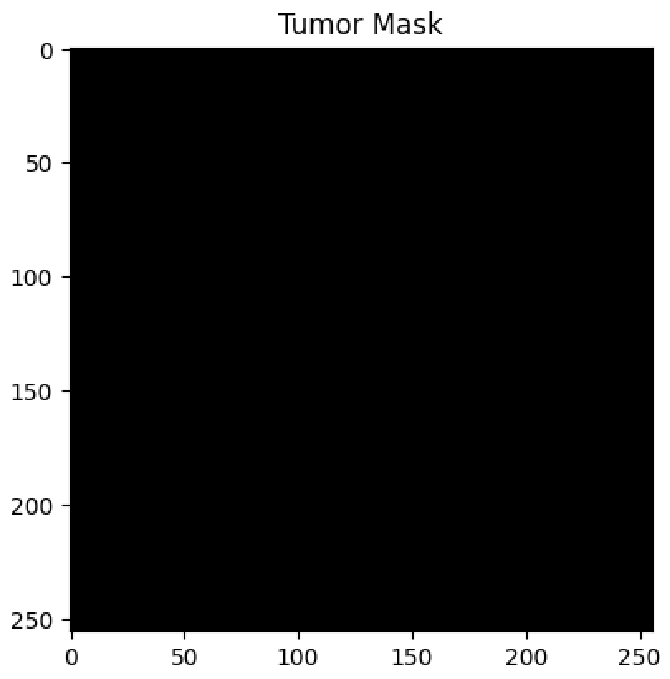
In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
#visualize a few samples
import random
r = random.randint(0, len(image_paths)-1)
img = cv2.imread(image_paths[r])
mask = cv2.imread(mask_paths[r], cv2.IMREAD_GRAYSCALE)

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('MRI Image')

plt.subplot(1,2,2)
plt.imshow(mask, cmap='gray')
plt.title('Tumor Mask')
plt.show()
```





In [ ]:

```
#preprocess images and Masks
IMG_SIZE = 128  # resize for speed (you can increase to 256 later)

def preprocess(img_path, mask_path):
    img = cv2.imread(img_path)
    mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    mask = cv2.resize(mask, (IMG_SIZE, IMG_SIZE))

    img = normalize(img, axis=-1)
    mask = np.expand_dims(mask, axis=-1)
    mask = mask / 255.0
    return img, mask
```

In [ ]:

```
x = []
y = []

for i in range(len(image_paths)):
    img, mask = preprocess(image_paths[i], mask_paths[i])
    x.append(img)
    y.append(mask)

x = np.array(x)
y = np.array(y)

print("Dataset Shape:", x.shape, y.shape)
```

```
Dataset Shape: (3929, 128, 128, 3) (3929, 128, 128, 1)
```

In [ ]:

```
#saving the x and y array in my local system
import numpy as np
np.save('/content/drive/MyDrive/project_datasets/x.npy',x)
np.save('/content/drive/MyDrive/project_datasets/y.npy',y)
```

In [ ]:

```
#train test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42
```

In [ ]:

```
#Builting a u-net model
def unet_model(input_size=(128,128,3)):
    inputs = Input(input_size)

    # Encoder
    c1 = Conv2D(16, 3, activation='relu', padding='same')(inputs)
    c1 = Conv2D(16, 3, activation='relu', padding='same')(c1)
    p1 = MaxPooling2D(pool_size=(2, 2))(c1)

    c2 = Conv2D(32, 3, activation='relu', padding='same')(p1)
    c2 = Conv2D(32, 3, activation='relu', padding='same')(c2)
    p2 = MaxPooling2D(pool_size=(2, 2))(c2)

    # Bottleneck
    c3 = Conv2D(64, 3, activation='relu', padding='same')(p2)
    c3 = Conv2D(64, 3, activation='relu', padding='same')(c3)

    # Decoder
    u4 = UpSampling2D(size=(2, 2))(c3)
    u4 = concatenate([u4, c2])
    c4 = Conv2D(32, 3, activation='relu', padding='same')(u4)
    c4 = Conv2D(32, 3, activation='relu', padding='same')(c4)

    u5 = UpSampling2D(size=(2, 2))(c4)
    u5 = concatenate([u5, c1])
    c5 = Conv2D(16, 3, activation='relu', padding='same')(u5)
    c5 = Conv2D(16, 3, activation='relu', padding='same')(c5)

    output = Conv2D(1, 1, activation='sigmoid')(c5)

    model = Model(inputs=[inputs], outputs=[output])
    return model
```

In [ ]:

```
segmentation_model = unet_model()
segmentation_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur
segmentation_model.summary()
```

```
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)         ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer          │ (None, 128, 128,  │          0 │ -                 │
│ (InputLayer)         │ 3)                │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d (Conv2D)      │ (None, 128, 128,  │        448 │ input_layer[0][0] │
│                      │ 16)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_1 (Conv2D)    │ (None, 128, 128,  │      2,320 │ conv2d[0][0]      │
│                      │ 16)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ max_pooling2d        │ (None, 64, 64,    │          0 │ conv2d_1[0][0]    │
│ (MaxPooling2D)       │ 16)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_2 (Conv2D)    │ (None, 64, 64,    │      4,640 │ max_pooling2d[0]… │
│                      │ 32)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_3 (Conv2D)    │ (None, 64, 64,    │      9,248 │ conv2d_2[0][0]    │
│                      │ 32)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ max_pooling2d_1      │ (None, 32, 32,    │          0 │ conv2d_3[0][0]    │
│ (MaxPooling2D)       │ 32)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_4 (Conv2D)    │ (None, 32, 32,    │     18,496 │ max_pooling2d_1[… │
│                      │ 64)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_5 (Conv2D)    │ (None, 32, 32,    │     36,928 │ conv2d_4[0][0]    │
│                      │ 64)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ up_sampling2d        │ (None, 64, 64,    │          0 │ conv2d_5[0][0]    │
│ (UpSampling2D)       │ 64)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ concatenate          │ (None, 64, 64,    │          0 │ up_sampling2d[0]… │
│ (Concatenate)        │ 96)               │            │ conv2d_3[0][0]    │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_6 (Conv2D)    │ (None, 64, 64,    │     27,680 │ concatenate[0][0] │
│                      │ 32)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_7 (Conv2D)    │ (None, 64, 64,    │      9,248 │ conv2d_6[0][0]    │
│                      │ 32)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ up_sampling2d_1      │ (None, 128, 128,  │          0 │ conv2d_7[0][0]    │
│ (UpSampling2D)       │ 32)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ concatenate_1        │ (None, 128, 128,  │          0 │ up_sampling2d_1[… │
│ (Concatenate)        │ 48)               │            │ conv2d_1[0][0]    │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_8 (Conv2D)    │ (None, 128, 128,  │      6,928 │ concatenate_1[0]… │
│                      │ 16)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_9 (Conv2D)    │ (None, 128, 128,  │      2,320 │ conv2d_8[0][0]    │
│                      │ 16)               │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_10 (Conv2D)   │ (None, 128, 128,  │         17 │ conv2d_9[0][0]    │
│                      │ 1)                │            │                   │
└──────────────────────┴───────────────────┴────────────┴───────────────────┘
 Total params: 118,273 (462.00 KB)
 Trainable params: 118,273 (462.00 KB)
 Non-trainable params: 0 (0.00 B)
```

In [ ]:

```python
#train the model
callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=3)
]

history = segmentation_model.fit(
    x_train, y_train,
    validation_data=(x_test, y_test),
    batch_size=16,
    epochs=50,
    callbacks=callbacks
)
```

```
Epoch 1/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 29s 80ms/step - accuracy: 0.9367 - loss: 0.1672 - val_accur
acy: 0.9877 - val_loss: 0.0400 - learning_rate: 0.0010
Epoch 2/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 29ms/step - accuracy: 0.9897 - loss: 0.0302 - val_accura
cy: 0.9906 - val_loss: 0.0283 - learning_rate: 0.0010
Epoch 3/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9921 - loss: 0.0256 - val_accura
cy: 0.9911 - val_loss: 0.0273 - learning_rate: 0.0010
Epoch 4/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 10s 29ms/step - accuracy: 0.9923 - loss: 0.0237 - val_accur
acy: 0.9916 - val_loss: 0.0266 - learning_rate: 0.0010
Epoch 5/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9925 - loss: 0.0228 - val_accura
cy: 0.9927 - val_loss: 0.0215 - learning_rate: 0.0010
Epoch 6/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 29ms/step - accuracy: 0.9926 - loss: 0.0213 - val_accura
cy: 0.9913 - val_loss: 0.0245 - learning_rate: 0.0010
Epoch 7/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9933 - loss: 0.0186 - val_accura
cy: 0.9930 - val_loss: 0.0196 - learning_rate: 0.0010
Epoch 8/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 29ms/step - accuracy: 0.9926 - loss: 0.0215 - val_accura
cy: 0.9931 - val_loss: 0.0196 - learning_rate: 0.0010
Epoch 9/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9937 - loss: 0.0176 - val_accura
cy: 0.9931 - val_loss: 0.0208 - learning_rate: 0.0010
Epoch 10/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9939 - loss: 0.0166 - val_accura
cy: 0.9918 - val_loss: 0.0226 - learning_rate: 0.0010
Epoch 11/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9943 - loss: 0.0153 - val_accura
cy: 0.9938 - val_loss: 0.0168 - learning_rate: 3.0000e-04
Epoch 12/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 10s 30ms/step - accuracy: 0.9944 - loss: 0.0147 - val_accur
acy: 0.9938 - val_loss: 0.0166 - learning_rate: 3.0000e-04
Epoch 13/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9941 - loss: 0.0153 - val_accura
cy: 0.9940 - val_loss: 0.0161 - learning_rate: 3.0000e-04
Epoch 14/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9945 - loss: 0.0144 - val_accura
cy: 0.9939 - val_loss: 0.0165 - learning_rate: 3.0000e-04
Epoch 15/50
```

```
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9946 - loss: 0.0142 - val_accura
cy: 0.9941 - val_loss: 0.0156 - learning_rate: 3.0000e-04
Epoch 16/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9946 - loss: 0.0140 - val_accura
cy: 0.9942 - val_loss: 0.0152 - learning_rate: 3.0000e-04
Epoch 17/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9947 - loss: 0.0141 - val_accura
cy: 0.9939 - val_loss: 0.0165 - learning_rate: 3.0000e-04
Epoch 18/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9947 - loss: 0.0138 - val_accura
cy: 0.9933 - val_loss: 0.0185 - learning_rate: 3.0000e-04
Epoch 19/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9947 - loss: 0.0135 - val_accura
cy: 0.9945 - val_loss: 0.0147 - learning_rate: 3.0000e-04
Epoch 20/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9948 - loss: 0.0138 - val_accura
cy: 0.9936 - val_loss: 0.0168 - learning_rate: 3.0000e-04
Epoch 21/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9944 - loss: 0.0149 - val_accura
cy: 0.9947 - val_loss: 0.0144 - learning_rate: 3.0000e-04
Epoch 22/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9950 - loss: 0.0129 - val_accura
cy: 0.9948 - val_loss: 0.0137 - learning_rate: 3.0000e-04
Epoch 23/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9947 - loss: 0.0131 - val_accura
cy: 0.9945 - val_loss: 0.0150 - learning_rate: 3.0000e-04
Epoch 24/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9948 - loss: 0.0133 - val_accura
cy: 0.9949 - val_loss: 0.0138 - learning_rate: 3.0000e-04
Epoch 25/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9953 - loss: 0.0119 - val_accura
cy: 0.9949 - val_loss: 0.0136 - learning_rate: 3.0000e-04
Epoch 26/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9952 - loss: 0.0120 - val_accura
cy: 0.9950 - val_loss: 0.0134 - learning_rate: 3.0000e-04
Epoch 27/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9954 - loss: 0.0114 - val_accura
cy: 0.9950 - val_loss: 0.0132 - learning_rate: 3.0000e-04
Epoch 28/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9952 - loss: 0.0119 - val_accura
cy: 0.9949 - val_loss: 0.0134 - learning_rate: 3.0000e-04
Epoch 29/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9954 - loss: 0.0114 - val_accura
cy: 0.9953 - val_loss: 0.0124 - learning_rate: 3.0000e-04
Epoch 30/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9956 - loss: 0.0109 - val_accura
cy: 0.9951 - val_loss: 0.0127 - learning_rate: 3.0000e-04
Epoch 31/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9955 - loss: 0.0111 - val_accura
cy: 0.9952 - val_loss: 0.0135 - learning_rate: 3.0000e-04
Epoch 32/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9958 - loss: 0.0104 - val_accura
cy: 0.9949 - val_loss: 0.0134 - learning_rate: 3.0000e-04
Epoch 33/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9958 - loss: 0.0104 - val_accura
cy: 0.9954 - val_loss: 0.0121 - learning_rate: 9.0000e-05
Epoch 34/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9957 - loss: 0.0106 - val_accura
cy: 0.9952 - val_loss: 0.0128 - learning_rate: 9.0000e-05
```

```
Epoch 35/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9957 - loss: 0.0102 - val_accura
cy: 0.9953 - val_loss: 0.0120 - learning_rate: 9.0000e-05
Epoch 36/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9959 - loss: 0.0097 - val_accura
cy: 0.9954 - val_loss: 0.0119 - learning_rate: 9.0000e-05
Epoch 37/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9960 - loss: 0.0097 - val_accura
cy: 0.9954 - val_loss: 0.0119 - learning_rate: 9.0000e-05
Epoch 38/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9959 - loss: 0.0101 - val_accura
cy: 0.9954 - val_loss: 0.0118 - learning_rate: 9.0000e-05
Epoch 39/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9962 - loss: 0.0093 - val_accura
cy: 0.9954 - val_loss: 0.0118 - learning_rate: 9.0000e-05
Epoch 40/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9963 - loss: 0.0088 - val_accura
cy: 0.9954 - val_loss: 0.0117 - learning_rate: 2.7000e-05
Epoch 41/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 32ms/step - accuracy: 0.9961 - loss: 0.0094 - val_accura
cy: 0.9954 - val_loss: 0.0118 - learning_rate: 2.7000e-05
Epoch 42/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9964 - loss: 0.0086 - val_accura
cy: 0.9955 - val_loss: 0.0118 - learning_rate: 2.7000e-05
Epoch 43/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9962 - loss: 0.0092 - val_accura
cy: 0.9954 - val_loss: 0.0117 - learning_rate: 2.7000e-05
Epoch 44/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9962 - loss: 0.0089 - val_accura
cy: 0.9955 - val_loss: 0.0117 - learning_rate: 8.1000e-06
Epoch 45/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9962 - loss: 0.0091 - val_accura
cy: 0.9955 - val_loss: 0.0117 - learning_rate: 8.1000e-06
Epoch 46/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9962 - loss: 0.0091 - val_accura
cy: 0.9955 - val_loss: 0.0117 - learning_rate: 8.1000e-06
Epoch 47/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9963 - loss: 0.0090 - val_accura
cy: 0.9955 - val_loss: 0.0117 - learning_rate: 2.4300e-06
Epoch 48/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9961 - loss: 0.0096 - val_accura
cy: 0.9955 - val_loss: 0.0117 - learning_rate: 2.4300e-06
Epoch 49/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9963 - loss: 0.0087 - val_accura
cy: 0.9955 - val_loss: 0.0117 - learning_rate: 2.4300e-06
Epoch 50/50
197/197 ━━━━━━━━━━━━━━━━━━━━ 6s 32ms/step - accuracy: 0.9962 - loss: 0.0090 - val_accura
cy: 0.9955 - val_loss: 0.0117 - learning_rate: 7.2900e-07
```

In [ ]:

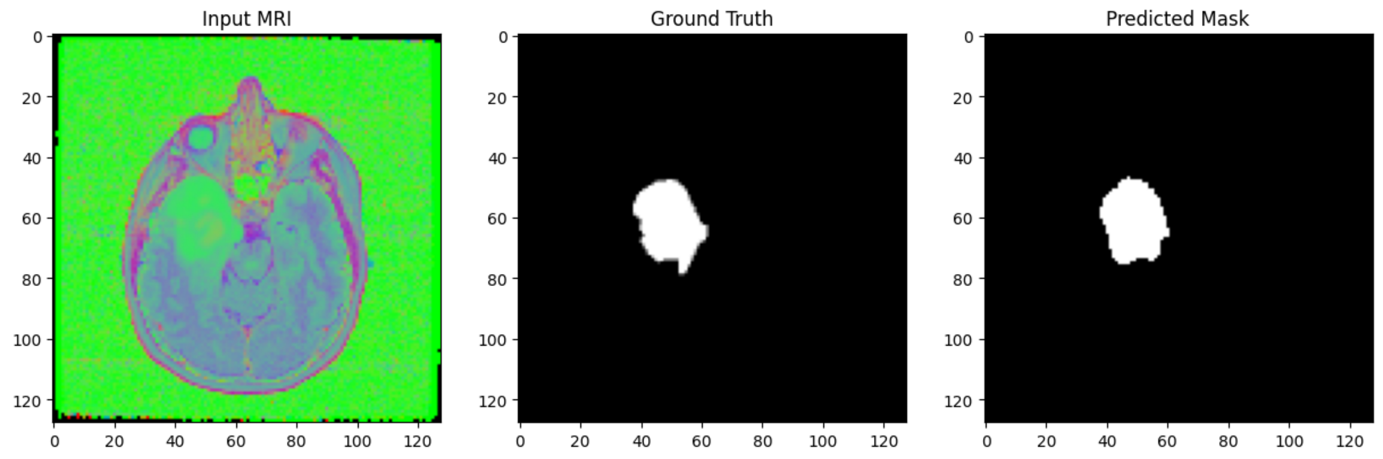
```
#saving the segmented model
segmentation_model.save('/content/drive/MyDrive/project_datasets/segmented_model(50epoch
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.savi
ng.save_model(model)`. This file format is considered legacy. We recommend using instead
the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model
(model, 'my_model.keras')`.
```

In [ ]:

```
#Evaluate and predictions
preds = segmentation_model.predict(x_test)
preds = (preds > 0.5).astype(np.uint8)

i = random.randint(0, len(preds)-1)
plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
plt.imshow(x_test[i])
plt.title('Input MRI')

plt.subplot(1,3,2)
plt.imshow(np.squeeze(y_test[i]), cmap='gray')
plt.title('Ground Truth')

plt.subplot(1,3,3)
plt.imshow(np.squeeze(preds[i]), cmap='gray')
plt.title('Predicted Mask')
plt.show()
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 4s 74ms/step
```



In [ ]:

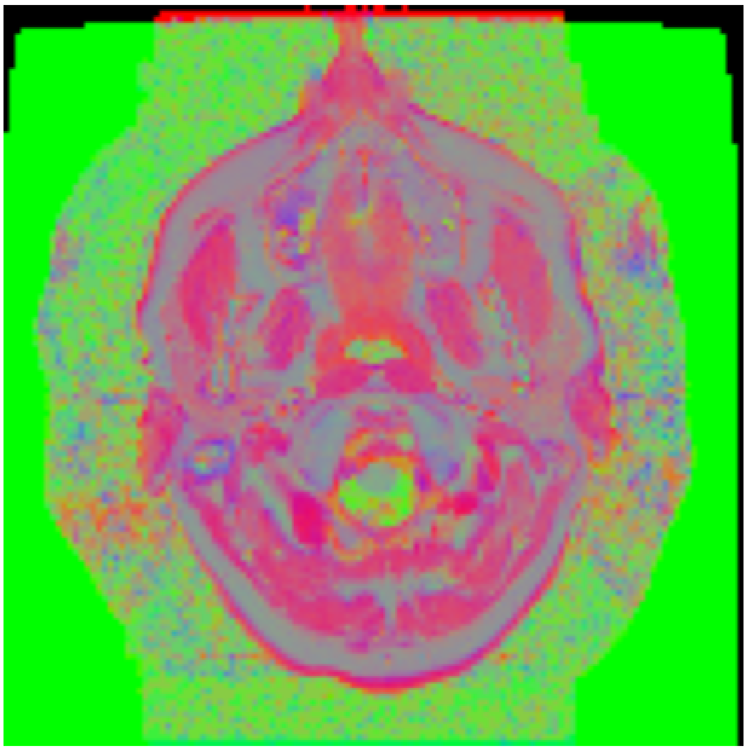
```
from tensorflow.keras.models import Model
import numpy as np
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf

def grad_cam(model, image, layer_name):
    grad_model = Model([model.inputs], [model.get_layer(layer_name).output, model.output

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(np.expand_dims(image, axis=0))

        # If predictions is a list, take the first output
        if isinstance(predictions, list):
            predictions = predictions[0]

        # Target channel
        target_output = predictions[:, :, :, 0]

    grads = tape.gradient(target_output, conv_outputs)[0]
    weights = tf.reduce_mean(grads, axis=[0, 1])
    weights = tf.reshape(weights, [-1])
    cam = np.ones(conv_outputs.shape[1:3], dtype=np.float32)
    for i, w in enumerate(weights):
        cam += w.numpy() * conv_outputs[0, :, :, i].numpy()

    cam = cv2.resize(cam, (128, 128))
    cam = np.maximum(cam, 0)
    cam = cam / (cam.max() + 1e-8)
    return cam
```

In [ ]:

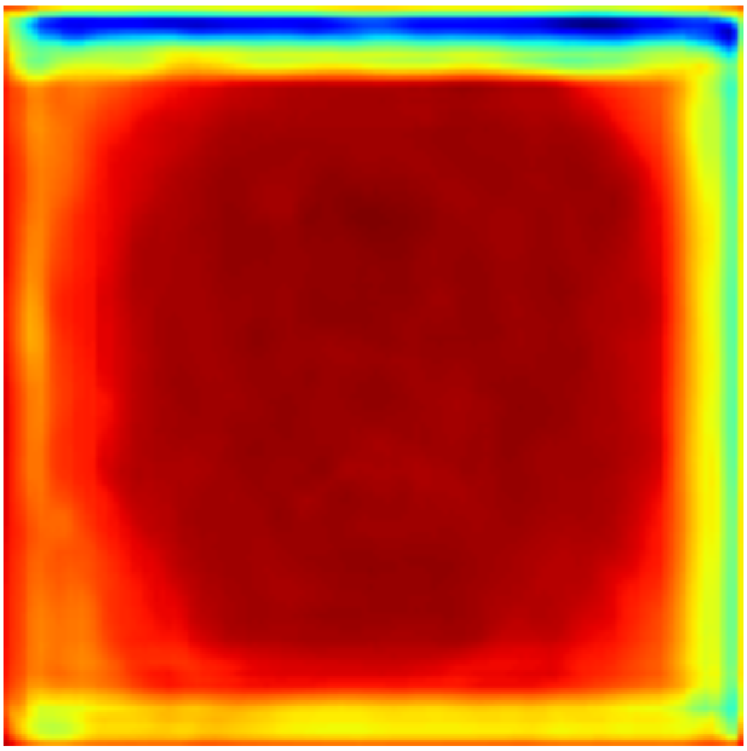
```
# Pick a sample image
i = 0  # index of the test image you want to visualize
sample_img = x_test[i]  # make sure x_test is your dataset (numpy array)

# Generate Grad-CAM heatmap
heatmap = grad_cam(segmentation_model, sample_img, 'conv2d_9')  # replace 'conv2d_9' wit

# Display the original image
plt.imshow(sample_img)
plt.title("Original Image")
plt.axis('off')
plt.show()

# Display the Grad-CAM heatmap
plt.imshow(heatmap, cmap='jet')
plt.title("Grad-CAM Heatmap")
plt.axis('off')
plt.show()

# Overlay heatmap on the image
plt.imshow(sample_img)
plt.imshow(heatmap, cmap='jet', alpha=0.5)  # adjust alpha for transparency
plt.title("Grad-CAM Overlay")
plt.axis('off')
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/models/functional.py:241: UserWarning:
The structure of `inputs` doesn't match the expected structure.
Expected: [['keras_tensor']]
Received: inputs=Tensor(shape=(1, 128, 128, 3))
  warnings.warn(msg)
```

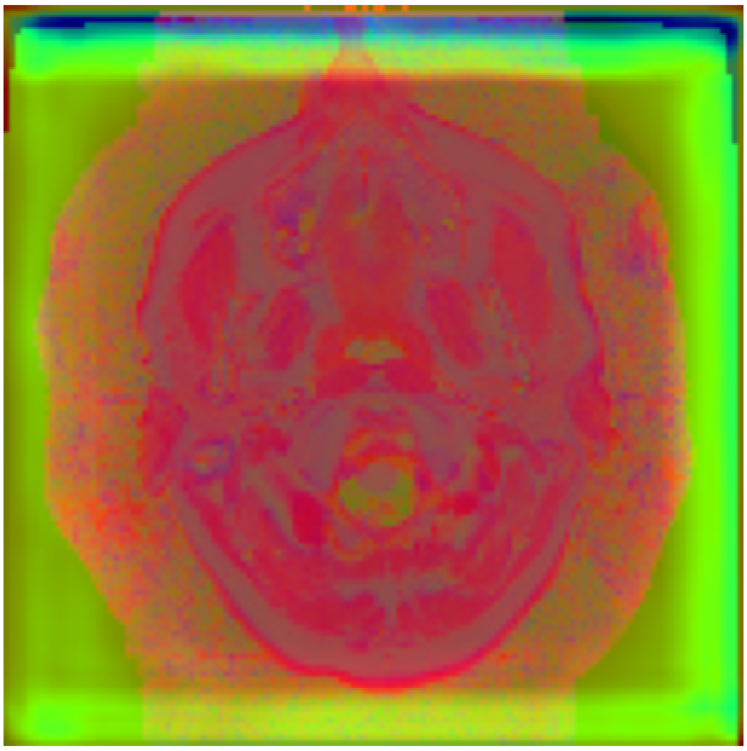
Original Image



Grad-CAM Heatmap

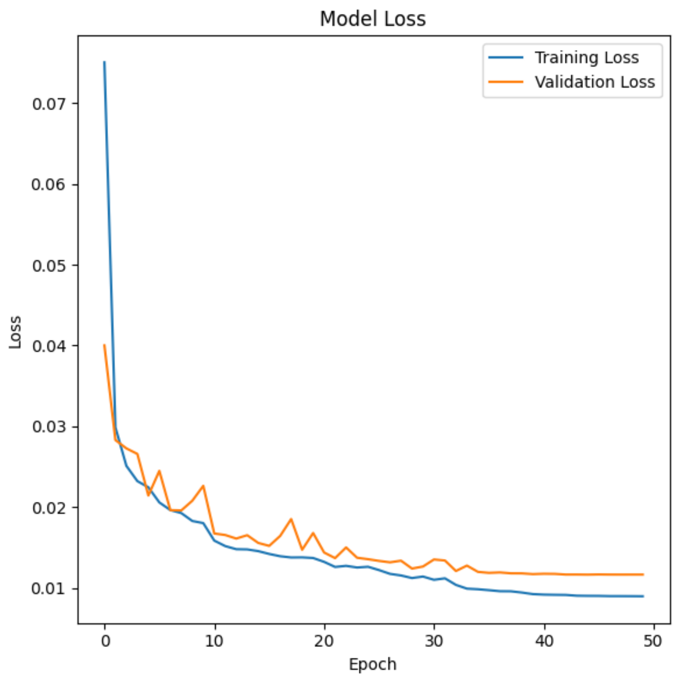


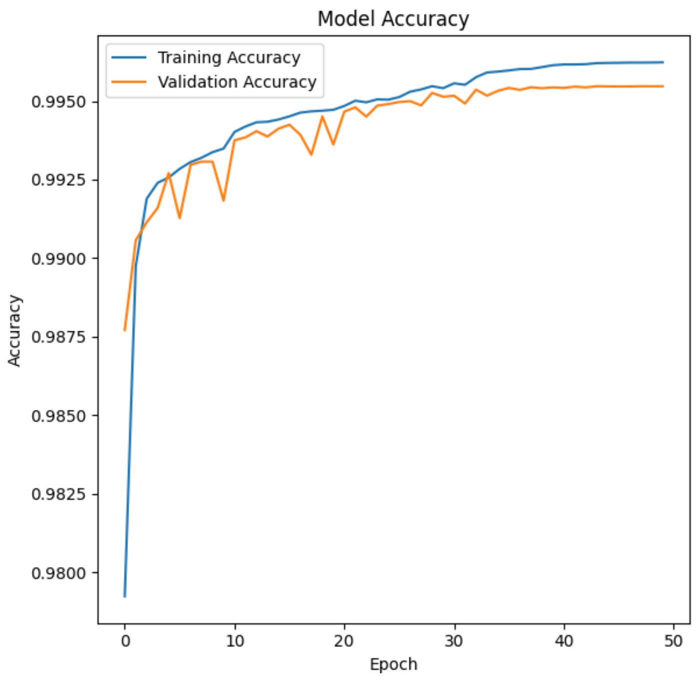
Grad-CAM Overlay



In [ ]:

```
# Plotting learning curves
plt.figure(figsize=(12, 6))

# Plot Loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot Accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

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





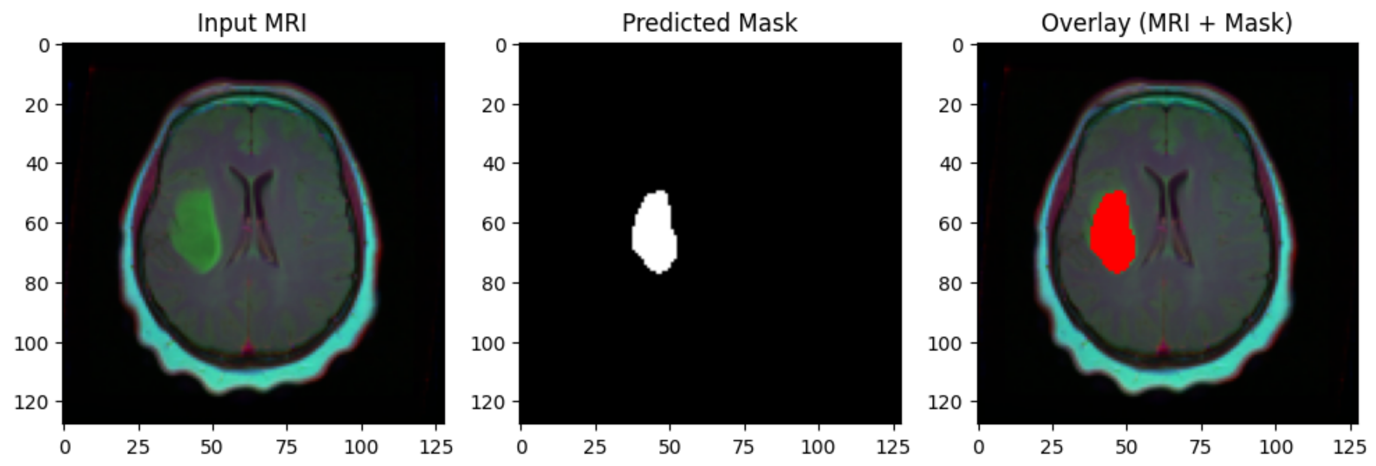
In [ ]:

```
pred_mask1 = segmentation_model.predict(input_img)
pred_mask1 = (pred_mask1 > 0.5).astype(np.uint8)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 798ms/step
```

In [ ]:

```
plt.figure(figsize=(12,5))

plt.subplot(1, 3, 1)
plt.imshow(new_img_resized)
plt.title("Input MRI")

plt.subplot(1, 3, 2)
plt.imshow(np.squeeze(pred_mask1), cmap='gray')
plt.title("Predicted Mask")

plt.subplot(1, 3, 3)
# Overlay mask on MRI (for better understanding)
overlay = new_img_resized.copy()
overlay[np.squeeze(pred_mask1)==1] = [255, 0, 0]  # red overlay
plt.imshow(overlay)
plt.title("Overlay (MRI + Mask)")

plt.show()
```



In [ ]:

```python
def iou_score(y_true, y_pred):
    intersection = np.logical_and(y_true, y_pred)
    union = np.logical_or(y_true, y_pred)
    # Add a small epsilon to avoid division by zero
    return np.sum(intersection) / (np.sum(union) + 1e-8)

# Find an index where y_test is not all zeros to get a meaningful IoU
valid_indices = [idx for idx, mask in enumerate(y_test) if np.sum(mask) > 0]

if valid_indices:
    i = random.choice(valid_indices)
    iou = iou_score(y_test[i], preds[i])
    print("IoU Score:", iou)
else:
    print("No valid test samples with masks found to calculate IoU.")
```

```
IoU Score: 0.8663594469846461
```

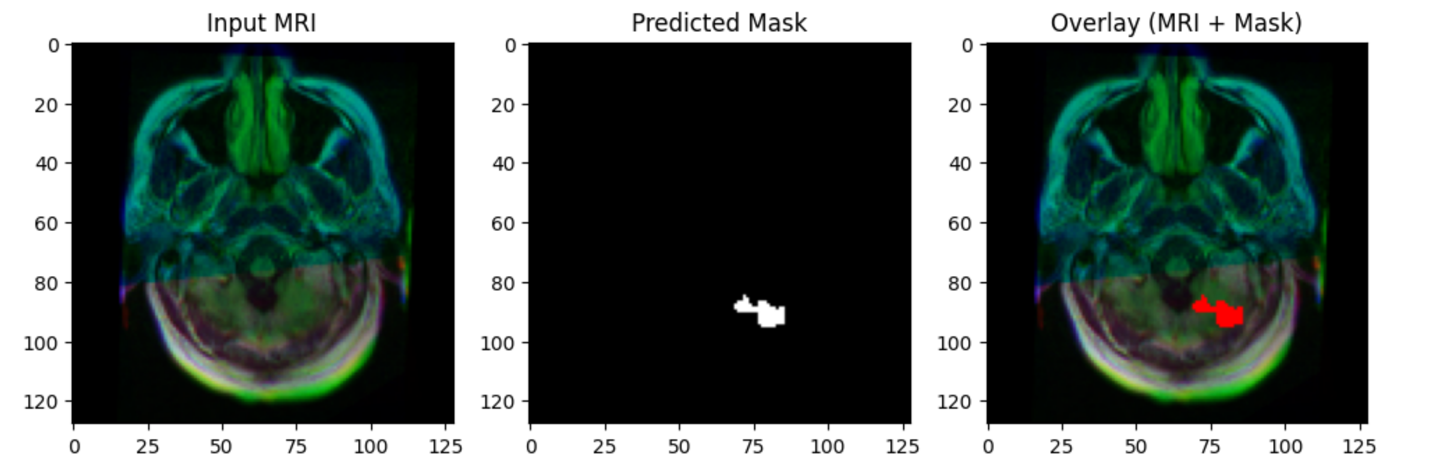
In [ ]:

```python
#prediction for new image
def predict_new_image(model, img_path, img_size=128):
    import numpy as np
    import cv2
    import matplotlib.pyplot as plt

    # Load and preprocess
    img = cv2.imread(img_path)
    img_resized = cv2.resize(img, (img_size, img_size))
    img_norm = img_resized / 255.0
    input_img = np.expand_dims(img_norm, axis=0)

    # Predict
    pred_mask = model.predict(input_img)
    pred_mask = (pred_mask > 0.5).astype(np.uint8)

    # Visualize
    plt.figure(figsize=(12,5))
    plt.subplot(1, 3, 1)
    plt.imshow(img_resized)
    plt.title("Input MRI")

    plt.subplot(1, 3, 2)
    plt.imshow(np.squeeze(pred_mask), cmap='gray')
    plt.title("Predicted Mask")

    plt.subplot(1, 3, 3)
    overlay = img_resized.copy()
    overlay[np.squeeze(pred_mask)==1] = [255, 0, 0]
    plt.imshow(overlay)
    plt.title("Overlay (MRI + Mask)")

    plt.show()
```

In [ ]:

```
predict_new_image(segmentation_model,'/content/drive/MyDrive/project_datasets/kaggle_3m/
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```



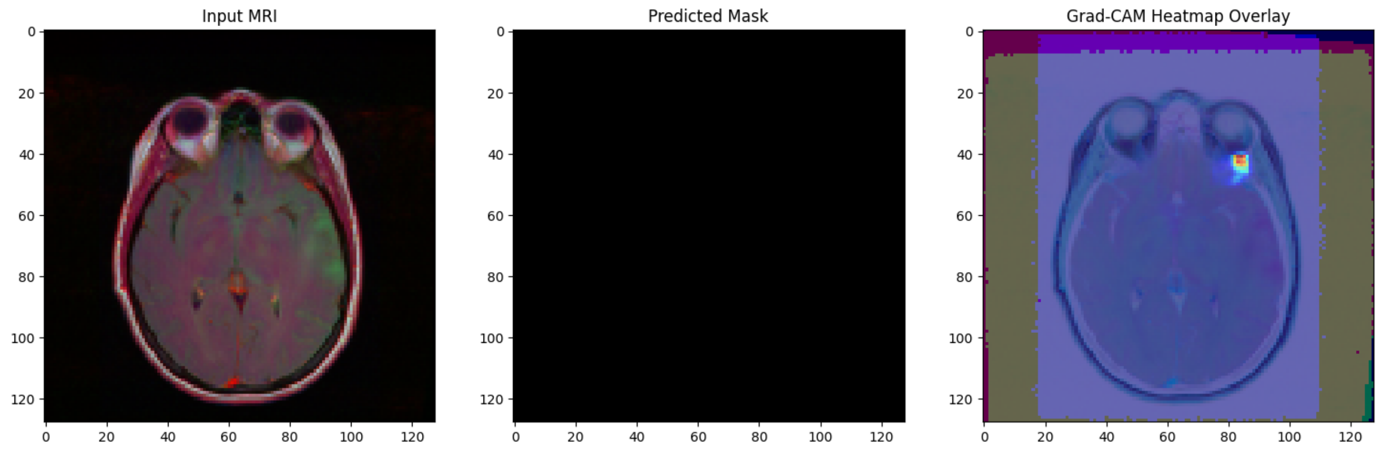
In [ ]:

```
import numpy as np
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt

def predict_and_explain(model, img_path, layer_name, img_size=128):

    #Load and preprocess the image
    img = cv2.imread(img_path)
    if img is None:
        raise ValueError(f"Image not found at: {img_path}")

    img_resized = cv2.resize(img, (img_size, img_size))
    img_norm = img_resized / 255.0
    input_img = np.expand_dims(img_norm, axis=0)

    #prediction of the segmented mask
    pred_mask = model.predict(input_img)
    pred_mask = (pred_mask > 0.5).astype(np.uint8)
    pred_mask_vis = np.squeeze(pred_mask)

    #GRID Cam
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(input_img)
        loss = tf.reduce_mean(predictions)
    grads = tape.gradient(loss, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]

    cam = np.zeros(conv_outputs.shape[:2], dtype=np.float32)
    for i, w in enumerate(pooled_grads):
        cam += w * conv_outputs[:, :, i]

    cam = np.maximum(cam, 0)
    cam = cam / np.max(cam)
    cam_resized = cv2.resize(cam, (img_size, img_size))

    #Heatmap overlay
    heatmap = cv2.applyColorMap(np.uint8(255 * cam_resized), cv2.COLORMAP_JET)
    heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)
    overlay = np.uint8(0.6 * heatmap + 0.4 * (img_resized * 255))

    #Output of all results
    plt.figure(figsize=(18, 6))

    plt.subplot(1, 3, 1)
    plt.imshow(img_resized)
    plt.title("Input MRI")

    plt.subplot(1, 3, 2)
    plt.imshow(pred_mask_vis, cmap='gray')
    plt.title("Predicted Mask")

    plt.subplot(1, 3, 3)
    plt.imshow(overlay)
    plt.title("Grad-CAM Heatmap Overlay")

    plt.show()

    return pred_mask_vis, cam_resized
```

In [ ]:

```
predict_and_explain(segmentation_model, "/content/drive/MyDrive/project_datasets/kaggle_
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```



Out[ ]:

```
(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]], dtype=uint8),
 array([[2.8216601e-07, 6.2412994e-08, 3.6319452e-08, ..., 1.7096623e-14,
         1.9968087e-14, 2.1958299e-12],
        [1.0777206e-12, 2.7534917e-14, 1.1563726e-13, ..., 1.7303560e-22,
         1.4883633e-22, 2.8622207e-19],
        [6.7293712e-14, 3.1987560e-19, 3.7160029e-20, ..., 0.0000000e+00,
         0.0000000e+00, 1.0484234e-26],
        ...,
        [2.1721213e-05, 6.8095822e-09, 1.9004179e-10, ..., 2.6528504e-21,
         3.0075729e-23, 8.5637667e-14],
        [1.6931085e-04, 8.9123961e-07, 4.0168629e-08, ..., 1.4027646e-17,
         4.4203958e-19, 1.3441402e-10],
        [4.8891362e-03, 2.7909124e-04, 1.7668830e-05, ..., 5.4767635e-11,
         2.8128586e-11, 9.0796874e-07]], dtype=float32))
```

In [ ]: