In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import (accuracy_score, roc_curve, auc, f1_score,
                             recall_score, precision_score, confusion_matrix,
                             classification_report, precision_recall_curve,
                             average_precision_score)
import os
import cv2
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import seaborn as sns

# Define tumor classes
path = os.listdir('brain_tumor/Training/')
classes = {'no_tumor': 0, 'pituitary_tumor': 1}

# Load and preprocess data
X = []
Y = []
for cls in classes:
    pth = 'brain_tumor/Training/' + cls
    for j in os.listdir(pth):
        img = cv2.imread(pth + '/' + j)
        img = cv2.resize(img, (224, 224))
        X.append(img)
        Y.append(classes[cls])

X = np.array(X)
Y = np.array(Y)

# Preprocess data for VGG-16
X = preprocess_input(X)
Y = tf.keras.utils.to_categorical(Y, num_classes=2)

# Split the dataset
xtrain, xtest, ytrain, ytest = train_test_split(
    X, Y, random_state=10, test_size=.20)

# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(xtrain)
```

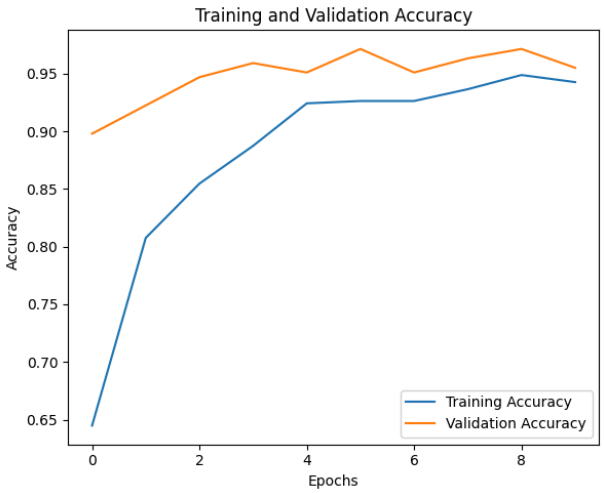
```python
# Build VGG-16 model
def build_vgg16_model(input_shape):
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=ir
    base_model.trainable = False
    model = models.Sequential([
        base_model,
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(2, activation='softmax')
    ])
    return model

# Initialize and compile model
input_shape = (224, 224, 3)
vgg16_model = build_vgg16_model(input_shape)
vgg16_model.compile(optimizer='adam',
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])

# Train model
history = vgg16_model.fit(
    datagen.flow(xtrain, ytrain, batch_size=32),
    epochs=10,
    validation_data=(xtest, ytest)
)

# Evaluate model
test_loss, test_acc = vgg16_model.evaluate(xtest, ytest)
print(f"VGG-16 Test Accuracy: {test_acc:.4f}")

# Plot training history
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

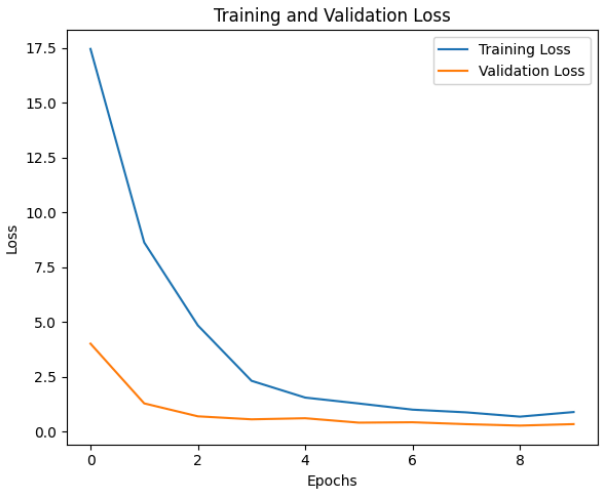
```
# Predictions
y_pred = vgg16_model.predict(xtest)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(ytest, axis=1)

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
ax = sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
                 cbar=True, annot_kws={"size": 16},
                 xticklabels=['No Tumor', 'Pituitary Tumor'],
                 yticklabels=['No Tumor', 'Pituitary Tumor'])

plt.title('Confusion Matrix\nVGG-16 Model', fontsize=16, pad=20)
plt.xlabel('Predicted Label', fontsize=14)
plt.ylabel('True Label', fontsize=14)

# Add percentages
conf_matrix_percent = conf_matrix.astype('float') / conf_matrix.sum(axis=1)[
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(j+0.5, i+0.3, f"{conf_matrix_percent[i,j]*100:.1f}%",
                ha="center", va="center", color="black", fontsize=12)

plt.tight_layout()
plt.show()

# Classification Report
print("Classification Report:")
print(classification_report(y_true, y_pred_classes, target_names=['No Tumor'

# ROC and Precision-Recall Curves
y_probs = y_pred[:, 1]
fpr, tpr, _ = roc_curve(y_true, y_probs)
roc_auc = auc(fpr, tpr)

precision, recall, _ = precision_recall_curve(y_true, y_probs)
avg_precision = average_precision_score(y_true, y_probs)

# Plot both curves
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(recall, precision, label=f'Precision-Recall (AP = {avg_precision:.2
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
```

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

def predict_specific_image(model, image_path):
    """Function to make prediction on a specific image"""
    try:
        print(f"\nProcessing image: {image_path}")

        # Load and display original image
        img = cv2.imread(image_path)
        if img is None:
            raise FileNotFoundError("Image not found or invalid path")

        plt.figure(figsize=(6, 6))
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title("Original Image")
        plt.axis('off')
        plt.show()

        # Preprocess the image
        print("Preprocessing image...")
        img = cv2.resize(img, (224, 224))
        img_processed = preprocess_input(img)

        # Display processed image
        plt.figure(figsize=(6, 6))
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title("Processed Image")
        plt.axis('off')
        plt.show()

        # Make prediction
        print("Making prediction...")
        pred = model.predict(np.expand_dims(img_processed, axis=0))
        pred_class = np.argmax(pred, axis=1)[0]
        confidence = np.max(pred) * 100
        class_name = 'No Tumor' if pred_class == 0 else 'Pituitary Tumor'

        # Display final result
        plt.figure(figsize=(8, 4))
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title(f"Input Image: {image_path}\nPrediction: {class_name} ({cc
        plt.axis('off')
        plt.show()

        print("\n" + "="*50)
        print("Prediction Result:")
        print(f"Class: {class_name}")
        print(f"Confidence: {confidence:.2f}%")
        print("="*50)

    except Exception as e:
        print(f"\nError processing image: {str(e)}")
        print("Please ensure:")
        print("1. The image file exists at the specified path")
        print("2. The file is a valid image format (jpg, png, etc.)")
```

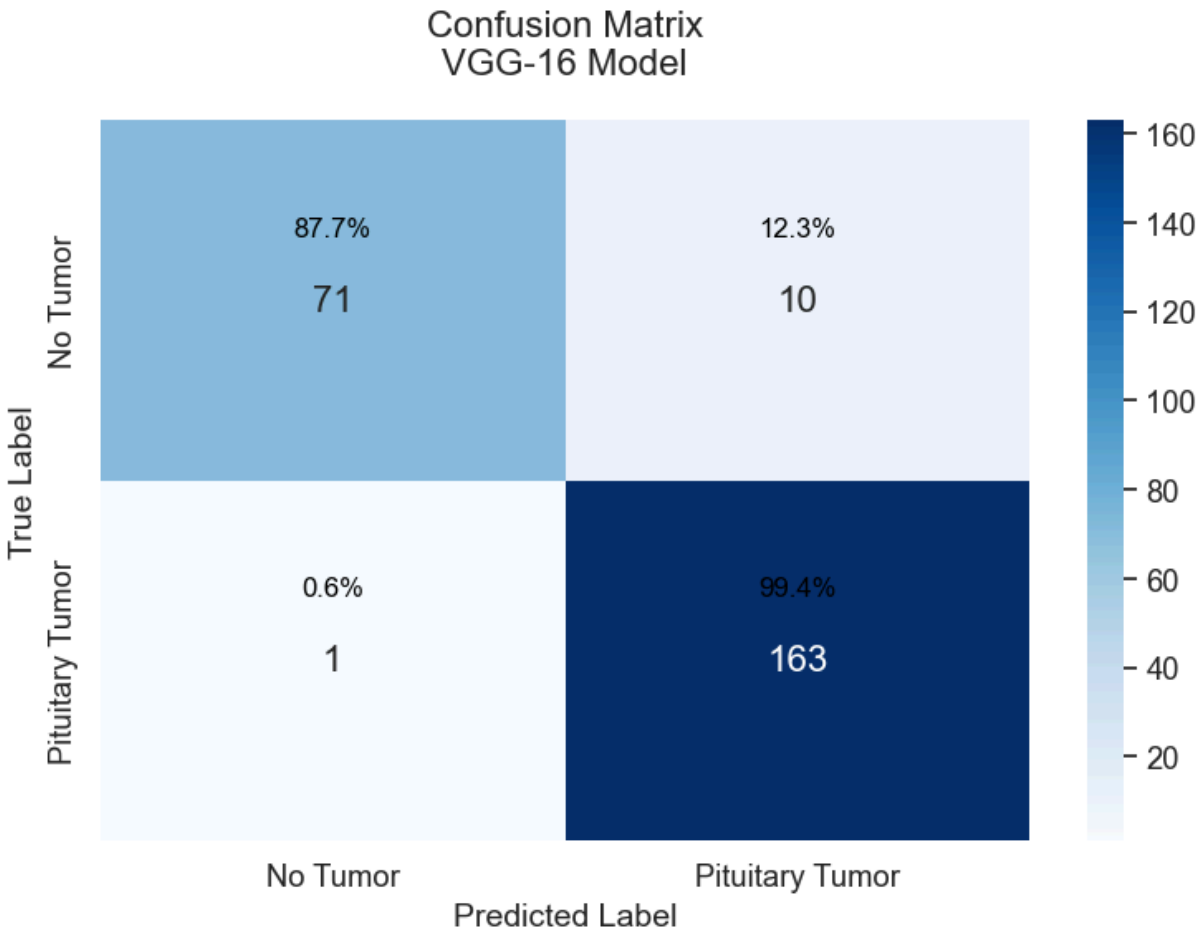
```
        print("3. You have read permissions for the file")

# Test your specific image
image_path = "image(12).jpg"  # Make sure this image is in your working dire
predict_specific_image(vgg16_model, image_path)
```

```
C:\Users\Ananda kumar sahoo\AppData\Local\Programs\Python\Python312\Lib\site
-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWa
rning: Your `PyDataset` class should call `super().__init__(**kwargs)` in it
s constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max
_queue_size`. Do not pass these arguments to `fit()`, as they will be ignore
d.
  self._warn_if_super_not_called()
Epoch 1/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 135s 4s/step - accuracy: 0.5914 - loss: 18.1663 -
val_accuracy: 0.8980 - val_loss: 4.0181
Epoch 2/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 127s 4s/step - accuracy: 0.7710 - loss: 11.9202 -
val_accuracy: 0.9224 - val_loss: 1.2947
Epoch 3/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 129s 4s/step - accuracy: 0.8421 - loss: 5.6723 -
val_accuracy: 0.9469 - val_loss: 0.7052
Epoch 4/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 128s 4s/step - accuracy: 0.8937 - loss: 2.3793 -
val_accuracy: 0.9592 - val_loss: 0.5686
Epoch 5/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 129s 4s/step - accuracy: 0.9271 - loss: 1.6117 -
val_accuracy: 0.9510 - val_loss: 0.6177
Epoch 6/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 129s 4s/step - accuracy: 0.9306 - loss: 1.6510 -
val_accuracy: 0.9714 - val_loss: 0.4168
Epoch 7/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 133s 4s/step - accuracy: 0.9316 - loss: 1.0222 -
val_accuracy: 0.9510 - val_loss: 0.4345
Epoch 8/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 124s 4s/step - accuracy: 0.9346 - loss: 0.8433 -
val_accuracy: 0.9633 - val_loss: 0.3476
Epoch 9/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 125s 4s/step - accuracy: 0.9537 - loss: 0.7445 -
val_accuracy: 0.9714 - val_loss: 0.2828
Epoch 10/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 129s 4s/step - accuracy: 0.9460 - loss: 0.9938 -
val_accuracy: 0.9551 - val_loss: 0.3497
8/8 ━━━━━━━━━━━━━━━━━━━━ 24s 3s/step - accuracy: 0.9407 - loss: 0.3838
VGG-16 Test Accuracy: 0.9551
```

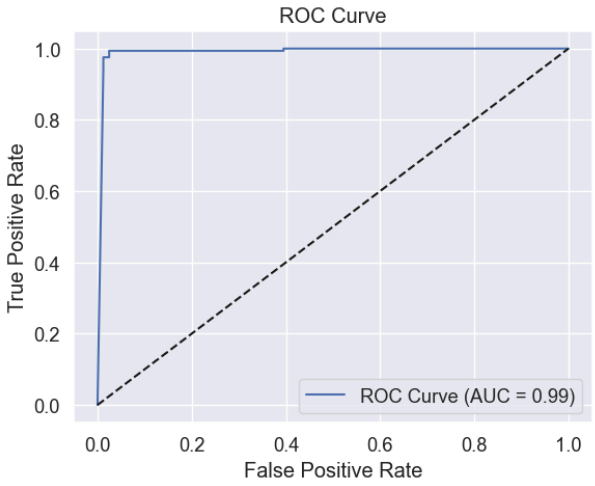


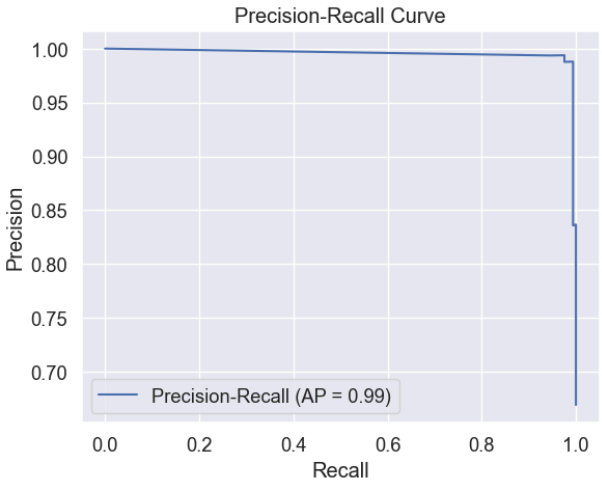


8/8 ━━━━━━━━━━━━━━━━━━━━ **25s** 3s/step

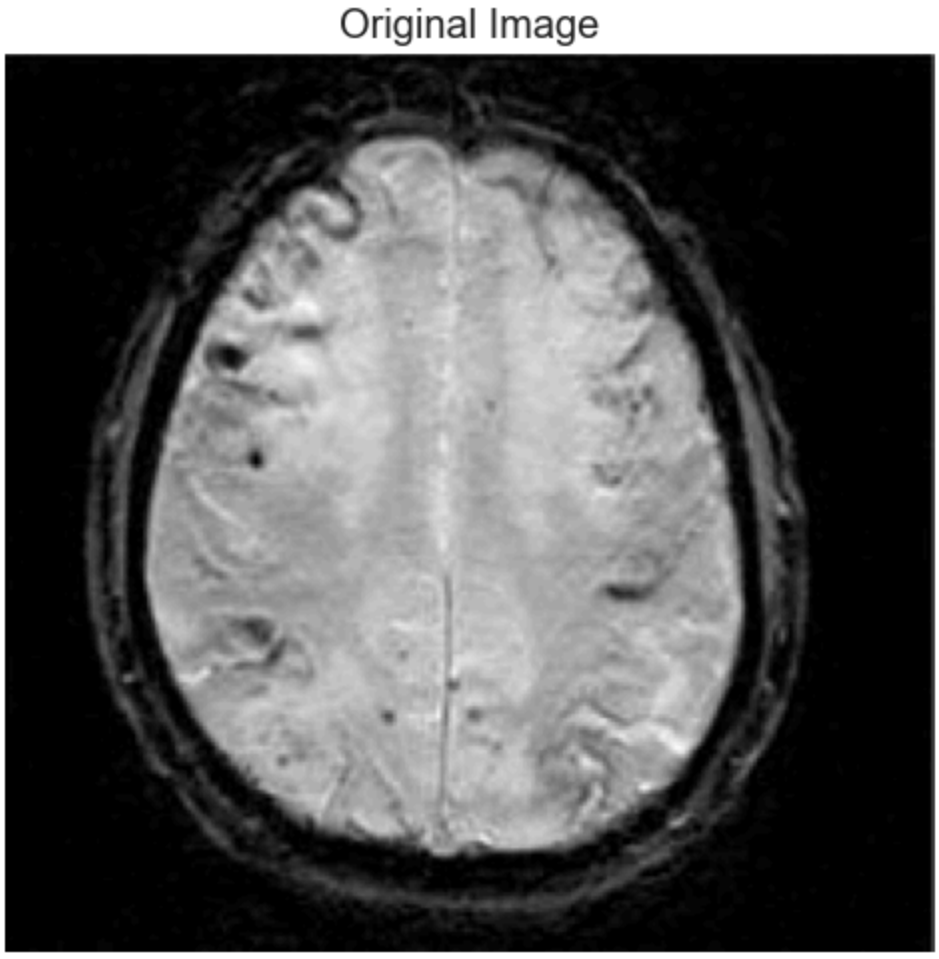



```
Classification Report:
                precision    recall  f1-score   support

      No Tumor       0.99      0.88      0.93        81
Pituitary Tumor      0.94      0.99      0.97       164

      accuracy                           0.96       245
     macro avg       0.96      0.94      0.95       245
  weighted avg       0.96      0.96      0.95       245
```

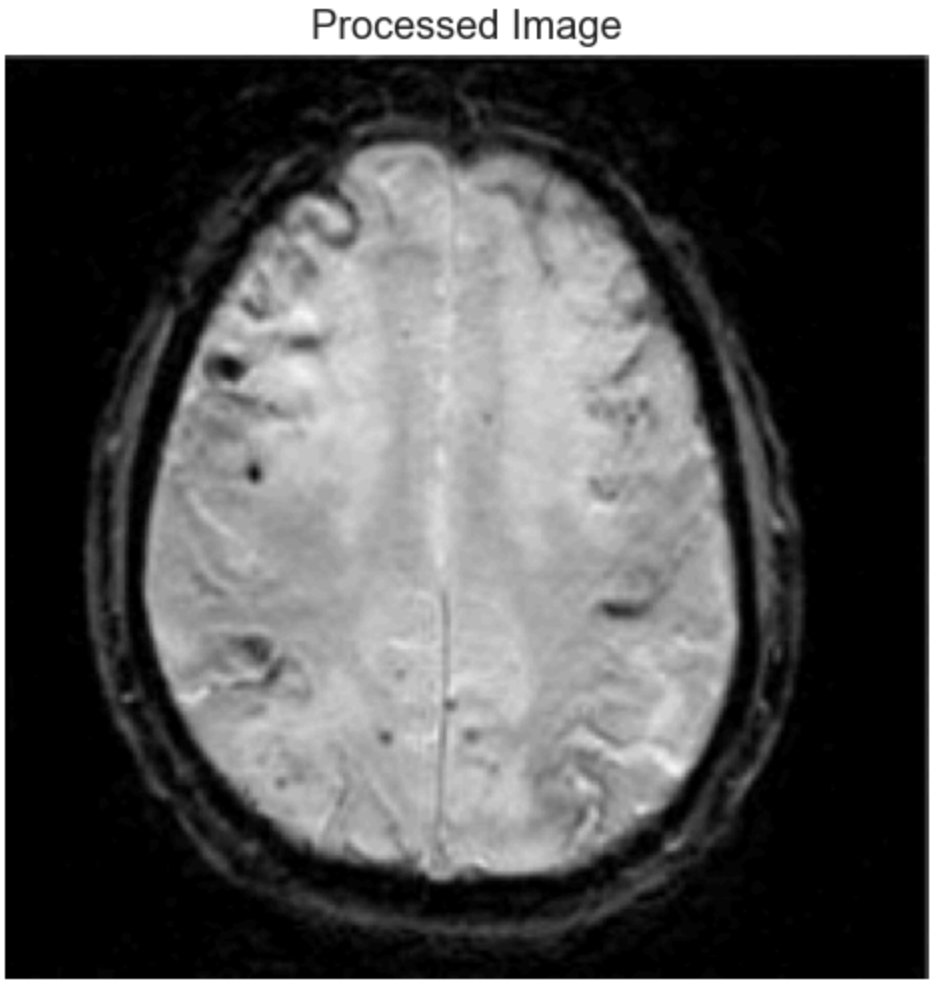




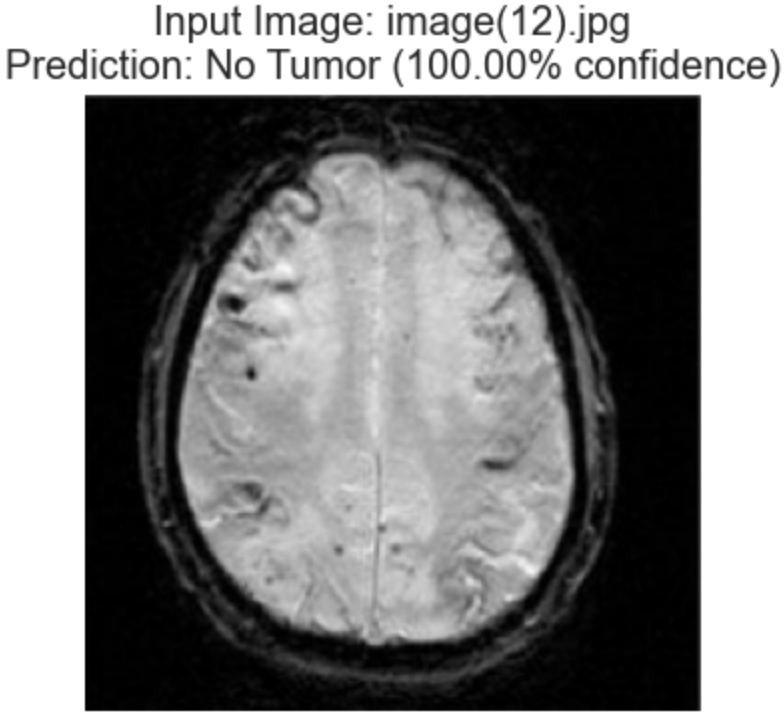

```
Processing image: image(12).jpg
```




```
Preprocessing image...
```




```
Making prediction...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 275ms/step
```



```
==================================================
Prediction Result:
Class: No Tumor
Confidence: 100.00%
==================================================
```

In [ ]: