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
import seaborn as sns
import pywt

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

# Function to apply orthogonal wavelet transform and INT function
def apply_wavelet_transform(image, wavelet='db1', level=1):
    # Perform wavelet transform
    coeffs = pywt.wavedec2(image, wavelet=wavelet, level=level)

    # Reconstruct the image from the wavelet coefficients
    reconstructed_image = pywt.waverec2(coeffs, wavelet=wavelet)

    # Apply INT function to convert pixel values to integers
    reconstructed_image = np.uint8(reconstructed_image)

    return reconstructed_image

# Load and preprocess data with wavelet transform and INT function
X_vgg_wavelet = []
Y_vgg_wavelet = []
for cls in classes:
    pth = 'brain_tumor/Training/' + cls
    for j in os.listdir(pth):
        img_vgg = cv2.imread(pth + '/' + j)
        img_vgg = cv2.resize(img_vgg, (224, 224))

        img_wavelet = np.zeros_like(img_vgg, dtype=np.uint8)
        for channel in range(3):
            img_wavelet[:, :, channel] = apply_wavelet_transform(img_vgg[:,

        X_vgg_wavelet.append(img_wavelet)
        Y_vgg_wavelet.append(classes[cls])

X_vgg_wavelet = np.array(X_vgg_wavelet)
Y_vgg_wavelet = np.array(Y_vgg_wavelet)

# Preprocess data for VGG-16
```

Loading [MathJax]/extensions/Safe.js

```
X_vgg_wavelet = preprocess_input(X_vgg_wavelet)
Y_vgg_wavelet = tf.keras.utils.to_categorical(Y_vgg_wavelet, num_classes=2)

# Split the dataset
xtrain_vgg, xtest_vgg, ytrain_vgg, ytest_vgg = train_test_split(
    X_vgg_wavelet, Y_vgg_wavelet, random_state=10, test_size=.20)

# Build Enhanced 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_vgg = (224, 224, 3)
vgg16_model = build_vgg16_model(input_shape_vgg)
vgg16_model.compile(optimizer='adam', loss='categorical_crossentropy', metri

# Train model
history = vgg16_model.fit(xtrain_vgg, ytrain_vgg, epochs=10, batch_size=32,
                          validation_data=(xtest_vgg, ytest_vgg))

# Evaluate model
test_loss, test_acc = vgg16_model.evaluate(xtest_vgg, ytest_vgg)
print(f"Enhanced 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()
```

Loading [MathJax]/extensions/Safe.js

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

# Enhanced 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('Enhanced Confusion Matrix\nVGG-16 with Wavelet Transform', fontsi
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')
```

Loading [MathJax]/extensions/Safe.js

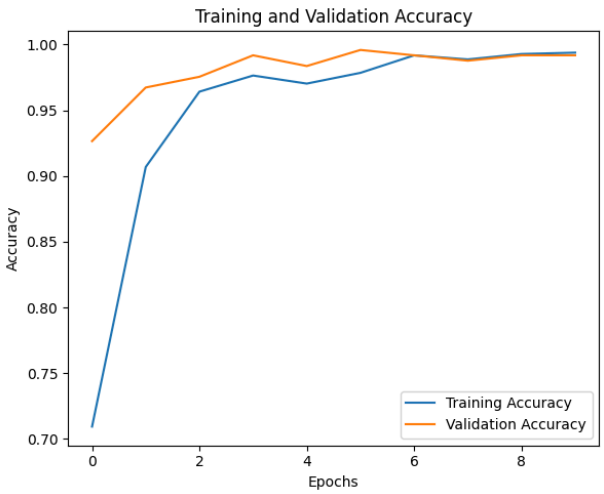
```
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()

# Detailed Precision-Recall Curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', lw=2,
        label=f'Precision-Recall curve (AP = {avg_precision:.2f})')
plt.fill_between(recall, precision, alpha=0.2, color='blue')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Detailed Precision-Recall Curve')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()


# [Keep all previous imports and code until the evaluation part]

# After all the evaluation code, add this:

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 (same as training pipeline)
        print("Preprocessing image...")
        img = cv2.resize(img, (224, 224))

        # Apply wavelet transform to each channel
        img_wavelet = np.zeros_like(img, dtype=np.uint8)
        for channel in range(3):
            img_wavelet[:, :, channel] = apply_wavelet_transform(img[:, :, c

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

        # Prepare for VGG model
```

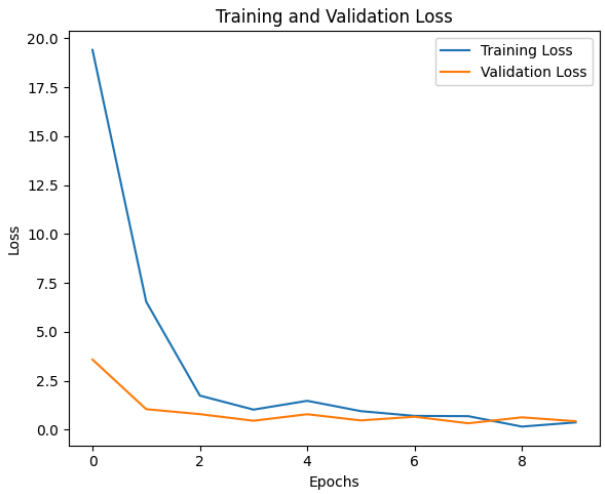
Loading [MathJax]/extensions/Safe.js

```
        img_processed = preprocess_input(img_wavelet)

        # 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(4).jpg"  # Make sure this image is in your working direc
predict_specific_image(vgg16_model, image_path)
```

Loading [MathJax]/extensions/Safe.js

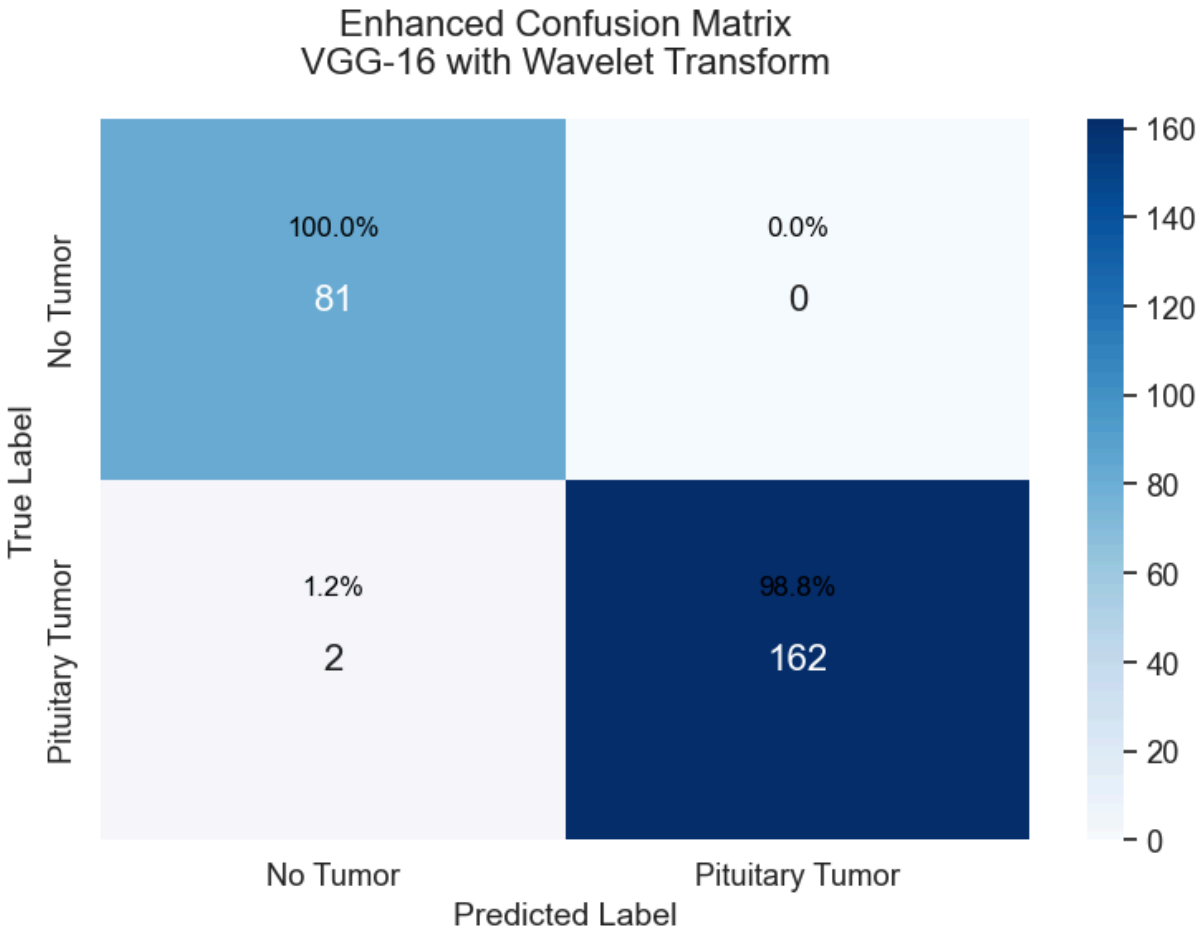
```
Epoch 1/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 134s 4s/step - accuracy: 0.5953 - loss: 24.3097 -
val_accuracy: 0.9265 - val_loss: 3.5817
Epoch 2/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 131s 4s/step - accuracy: 0.8869 - loss: 7.7435 -
val_accuracy: 0.9673 - val_loss: 1.0449
Epoch 3/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 210s 7s/step - accuracy: 0.9643 - loss: 2.0380 -
val_accuracy: 0.9755 - val_loss: 0.7900
Epoch 4/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 216s 7s/step - accuracy: 0.9778 - loss: 0.8808 -
val_accuracy: 0.9918 - val_loss: 0.4537
Epoch 5/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 218s 7s/step - accuracy: 0.9693 - loss: 1.8243 -
val_accuracy: 0.9837 - val_loss: 0.7860
Epoch 6/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 211s 7s/step - accuracy: 0.9766 - loss: 0.8270 -
val_accuracy: 0.9959 - val_loss: 0.4703
Epoch 7/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 266s 7s/step - accuracy: 0.9912 - loss: 0.4660 -
val_accuracy: 0.9918 - val_loss: 0.6581
Epoch 8/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 219s 7s/step - accuracy: 0.9846 - loss: 1.1801 -
val_accuracy: 0.9878 - val_loss: 0.3233
Epoch 9/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 209s 7s/step - accuracy: 0.9896 - loss: 0.2701 -
val_accuracy: 0.9918 - val_loss: 0.6278
Epoch 10/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 209s 7s/step - accuracy: 0.9922 - loss: 0.5588 -
val_accuracy: 0.9918 - val_loss: 0.4286
8/8 ━━━━━━━━━━━━━━━━━━━━ 41s 5s/step - accuracy: 0.9977 - loss: 0.1322
Enhanced VGG-16 Test Accuracy: 0.9918
```





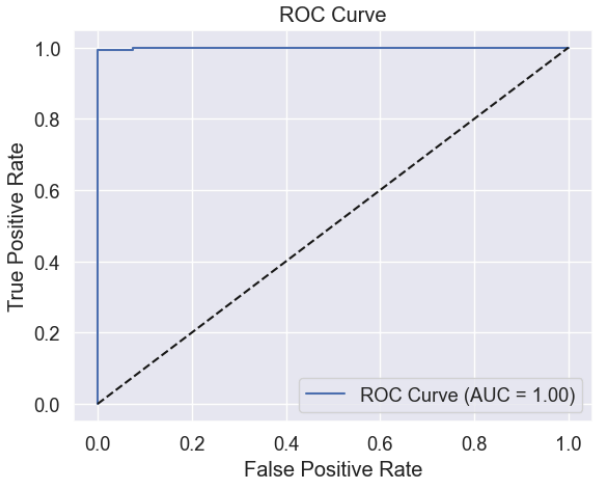

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 42s 5s/step
```

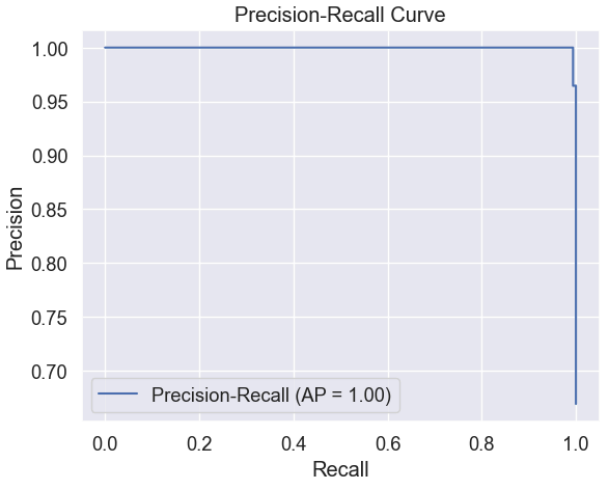
Loading [MathJax]/extensions/Safe.js



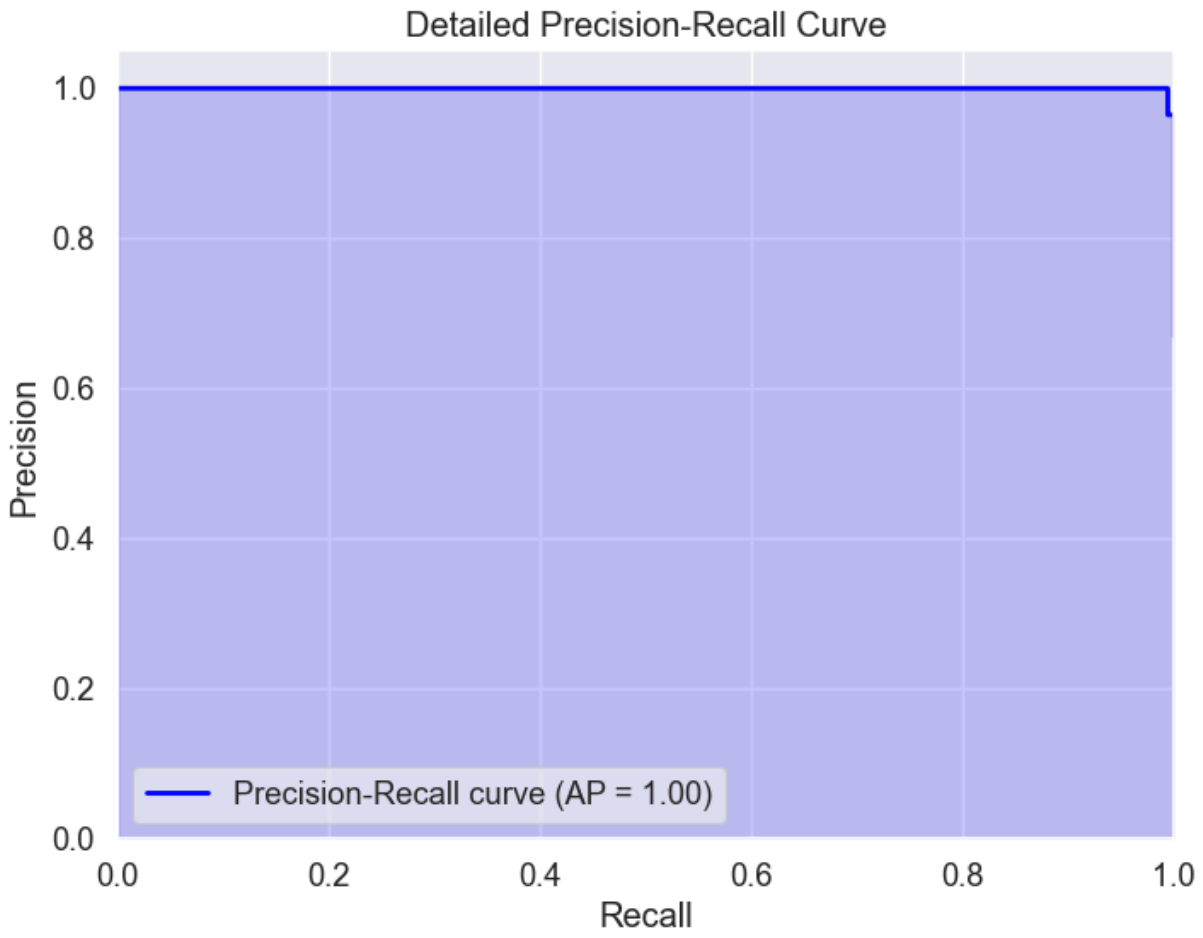
Classification Report:

	precision	recall	f1-score	support
No Tumor	0.98	1.00	0.99	81
Pituitary Tumor	1.00	0.99	0.99	164
accuracy			0.99	245
macro avg	0.99	0.99	0.99	245
weighted avg	0.99	0.99	0.99	245





Loading [MathJax]/extensions/Safe.js

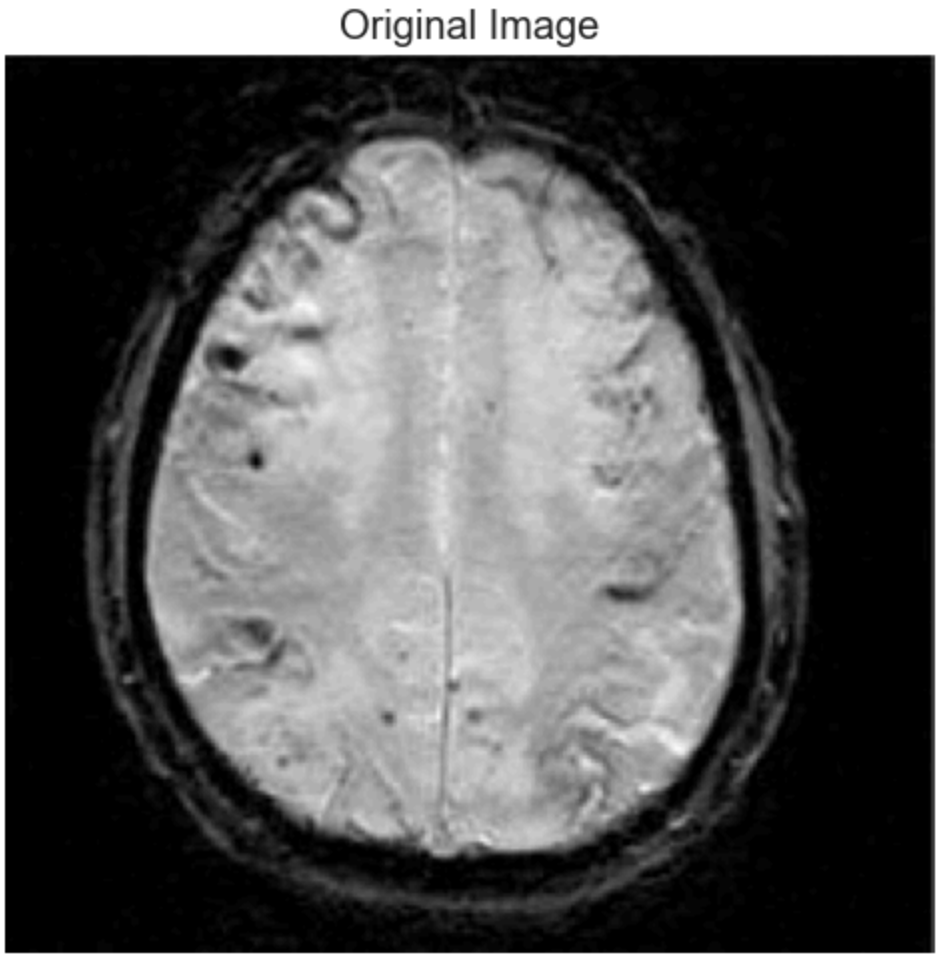



```
Processing image: image(4).jpg

Error processing image: Image not found or invalid path
Please ensure:
1. The image file exists at the specified path
2. The file is a valid image format (jpg, png, etc.)
3. You have read permissions for the file
```

In [2]:
```
# 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)
```

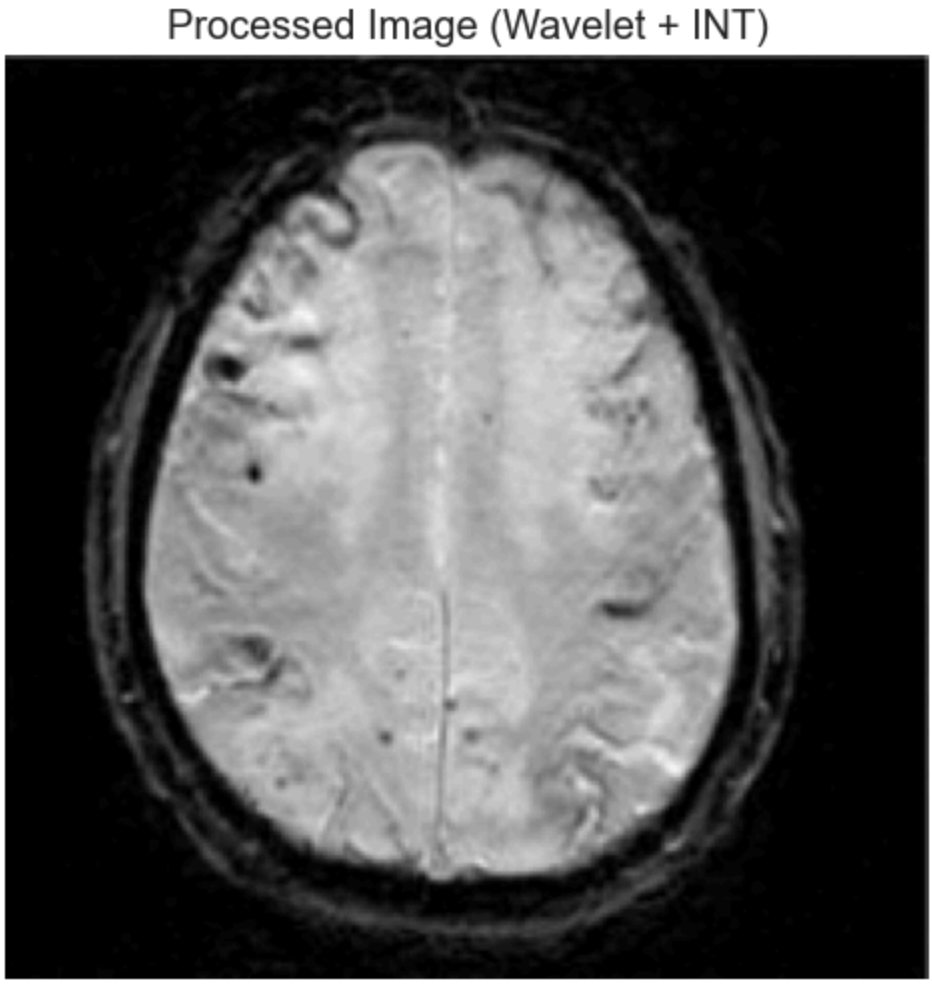
```
Processing image: image(12).jpg
```

Loading [MathJax]/extensions/Safe.js

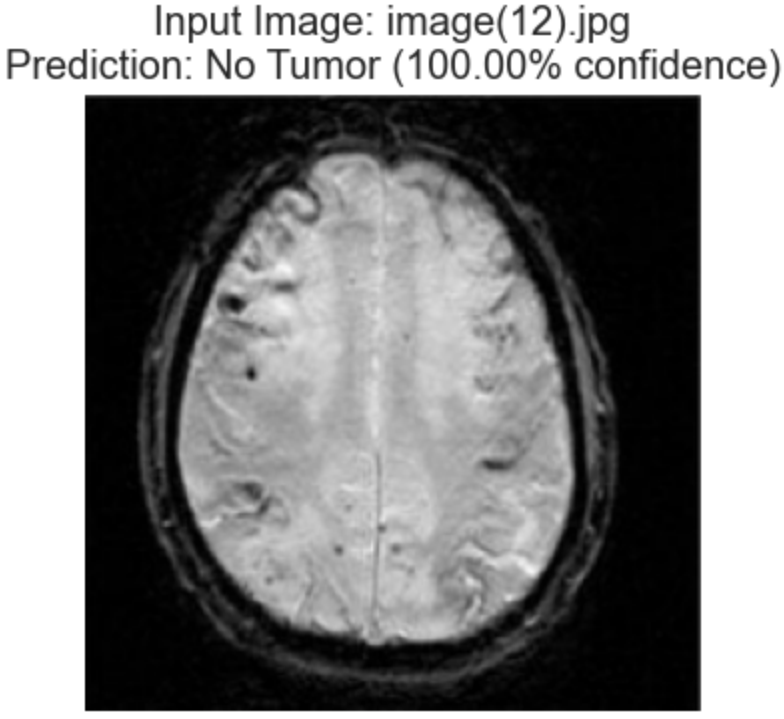



```
Preprocessing image...
```

Loading [MathJax]/extensions/Safe.js




```
Making prediction...
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 525ms/step
```



Loading [MathJax]/extensions/Safe.js

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

In [ ]:

Loading [MathJax]/extensions/Safe.js