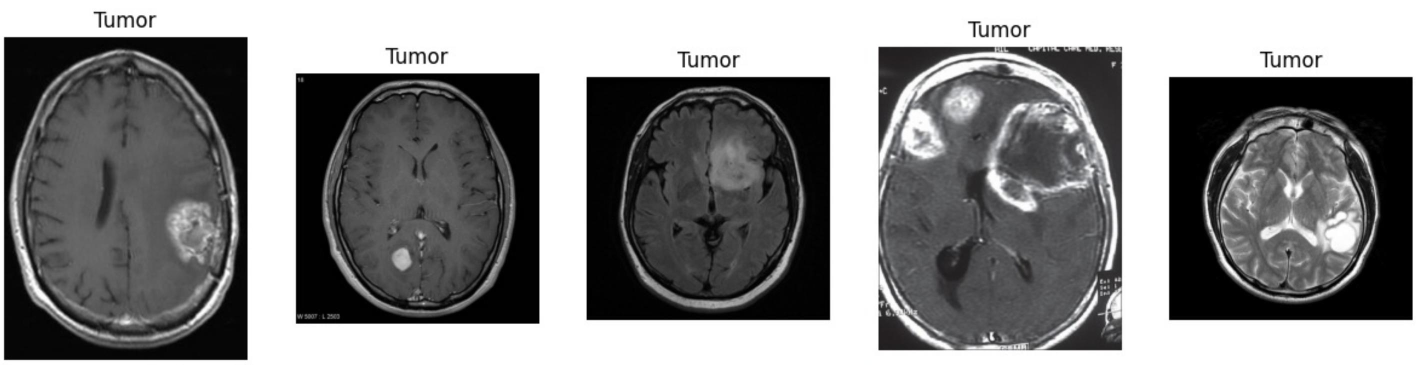
In [3]:

```python
import os
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import pandas as pd
import numpy as np


# Hardcoded directories for 'yes' and 'no'
yes_dir = '/Users/yannietchi/Desktop/comp-562-final-project/MRI data/yes'
no_dir = '/Users/yannietchi/Desktop/comp-562-final-project/MRI data/no'
```

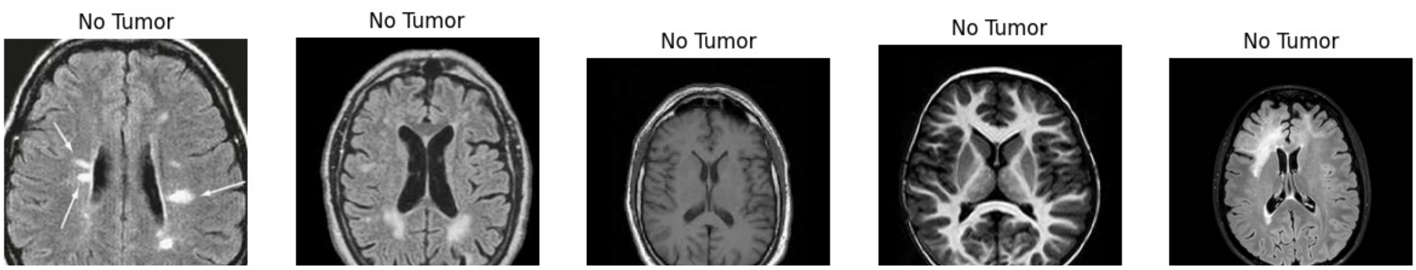
In [4]:

```python
# Function to display sample images from a folder
def display_images(folder, label, n=5):
    # Get all image files from the folder
    image_files = [os.path.join(folder, f) for f in os.listdir(folder) if os.path.isfile
(os.path.join(folder, f))]

    # Select up to `n` images to display
    image_files = image_files[:n]

    # Create a figure
    plt.figure(figsize=(15, 5))
    for i, img_path in enumerate(image_files):
        img = Image.open(img_path)  # Open the image
        plt.subplot(1, n, i + 1)    # Add a subplot
        plt.imshow(img, cmap='gray')  # Display the image
        plt.title(label)  # Title as the label
        plt.axis('off')   # Turn off axes
    plt.show()


# Display sample images from 'yes' and 'no'
print("Sample images from 'yes' (Tumor) dataset:")
display_images(yes_dir, label="Tumor")

print("Sample images from 'no' (No Tumor) dataset:")
display_images(no_dir, label="No Tumor")
```

```
Sample images from 'yes' (Tumor) dataset:
```




```
Sample images from 'no' (No Tumor) dataset:
```



In [5]:

```
image_paths = []
labels = []

# Process 'yes' folder
for filename in os.listdir(yes_dir):
    file_path = os.path.join(yes_dir, filename)
    if os.path.isfile(file_path):  # Ensure it's a file
        image_paths.append(file_path)
        labels.append(1)  # Label for tumor

# Process 'no' folder
for filename in os.listdir(no_dir):
    file_path = os.path.join(no_dir, filename)
    if os.path.isfile(file_path):  # Ensure it's a file
        image_paths.append(file_path)
        labels.append(0)  # Label for no tumor

# Create the dataframe
data = pd.DataFrame({'image_path': image_paths, 'label': labels})

# Display the first few rows

data
```

Out[5]:

	image_path	label
0	/Users/yannietchi/Desktop/comp-562-final-proje...	1
1	/Users/yannietchi/Desktop/comp-562-final-proje...	1
2	/Users/yannietchi/Desktop/comp-562-final-proje...	1
3	/Users/yannietchi/Desktop/comp-562-final-proje...	1
4	/Users/yannietchi/Desktop/comp-562-final-proje...	1
...	...	...
248	/Users/yannietchi/Desktop/comp-562-final-proje...	0
249	/Users/yannietchi/Desktop/comp-562-final-proje...	0
250	/Users/yannietchi/Desktop/comp-562-final-proje...	0
251	/Users/yannietchi/Desktop/comp-562-final-proje...	0
252	/Users/yannietchi/Desktop/comp-562-final-proje...	0

253 rows × 2 columns

Data Augmentation

In [6]:

```
# Define data augmentation for grayscale images
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Function to load and augment images
def augment_images(folder, label, augmentor, num_augments=3):
```

```
    data = []
    for file in os.listdir(folder):
        img_path = os.path.join(folder, file)
        if os.path.isfile(img_path):
            # Load and preprocess image
            img = Image.open(img_path).convert('L')  # Convert to grayscale
            img = np.array(img).reshape((1, img.size[1], img.size[0], 1))  # Reshape for
augmentation

            # Append original image to data
            data.append({'image': img.squeeze(), 'label': label})

            # Generate augmented images
            aug_iter = augmentor.flow(img, batch_size=1)
            for _ in range(num_augments):
                aug_img = next(aug_iter)[0].squeeze()
                data.append({'image': aug_img, 'label': label})
    return data



# Augment images in both folders
yes_data = augment_images(yes_dir, 1, datagen, num_augments=3)  # Tumor label is 1
no_data = augment_images(no_dir, 0, datagen, num_augments=3)    # No-tumor label is 0

# Combine data into a DataFrame
data_augmented = yes_data + no_data
df_augmented = pd.DataFrame(data_augmented)

# Example: Display first few rows of the DataFrame
df_augmented
```

Out[6]:

	image	label
0	[[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,...	1
1	[[6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0,...	1
2	[[6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0,...	1
3	[[6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0,...	1
4	[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...	1
...	...	...
1007	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...	0
1008	[[28, 29, 0, 2, 0, 3, 0, 1, 0, 0, 0, 0, 0, 0, ...	0
1009	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...	0
1010	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...	0
1011	[[28.0, 28.0, 28.0, 28.0, 28.487947, 19.391432...	0

1012 rows × 2 columns

In [7]:

```
from tensorflow.image import resize
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import pandas as pd
from PIL import Image
import os
import random

# Set the target size for images
target_size = (64, 64)

# Augmentation setup
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'
)

# Create a DataFrame to store image and label
augmented_images = []

# Loop through the original 'yes' and 'no' folders and augment images
def augment_images_from_folder(folder, label):
    for filename in os.listdir(folder):
        if filename.endswith('.png') or filename.endswith('.jpg'):
            # Open the image
            img_path = os.path.join(folder, filename)
            img = Image.open(img_path).convert('L')  # Convert to grayscale
            img = np.array(img)

            # Standardize shape to target size (e.g., 64x64)
            img_resized = resize(np.expand_dims(img, axis=-1), target_size).numpy()

            # Perform augmentation
            img_reshaped = img_resized.reshape((1,) + img_resized.shape)  # Reshape for
augmentation
            it = datagen.flow(img_reshaped, batch_size=1)

            # Store augmented images and their labels
            for _ in range(5):  # Augment and store 5 versions per image
                augmented_img = next(it)[0].astype(np.uint8)  # Get the augmented image
                augmented_images.append([augmented_img, label])


# Augment images from 'yes' and 'no' folders
augment_images_from_folder(yes_dir, 1)  # Tumor label = 1
augment_images_from_folder(no_dir, 0)  # No tumor label = 0

# Convert to DataFrame
df_augmented = pd.DataFrame(augmented_images, columns=['image', 'label'])

# Show a sample from the dataframe (e.g., first 3 images)
df_augmented.head(3)
```

Out[7]:

	image	label
0	[[[1], [1], [1], [1], [1], [1], [1], [1], [1],...	1
1	[[[1], [1], [1], [1], [1], [1], [1], [1], [1],...	1
2	[[[1], [1], [1], [1], [1], [1], [1], [1], [1],...	1

Baseline Model

In [8]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers.legacy import Adam
import matplotlib.pyplot as plt
import numpy as np
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import BatchNormalization, GlobalAveragePooling2D


# Assuming df_augmented is already created as shown in the previous steps
X = np.array([np.expand_dims(img, axis=-1) for img in df_augmented['image']])  # Shape:
(n_samples, 64, 64, 1)
y = np.array(df_augmented['label'])  # Labels: 0 or 1

# First split the data into training and temporary (for validation + test)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42
)

# Now split the temporary set into validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_st
ate=42)
```

In [23]:

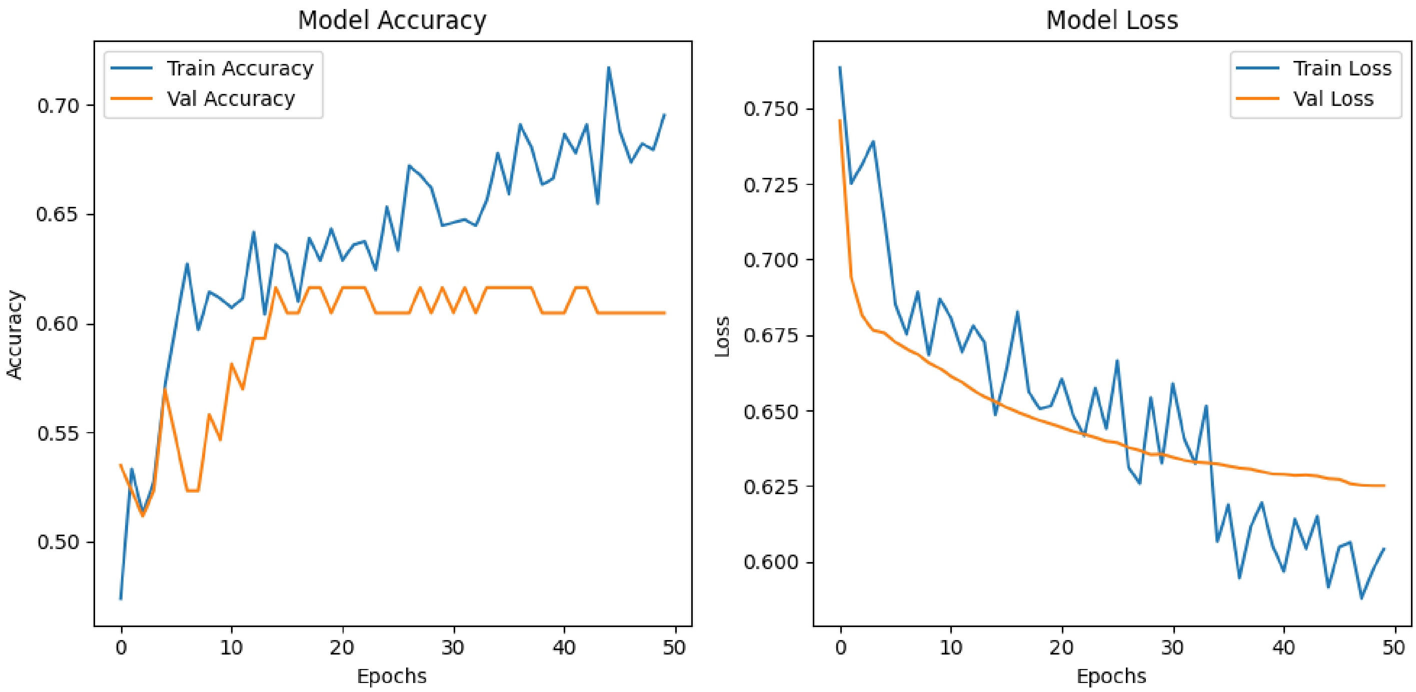
```
# Define CNN model architecture

def baseline_cnn_model(input_shape=(64, 64, 1)):
    model = Sequential()

    # First convolutional block
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_sh
ape))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    # Second convolutional block
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    # Third convolutional block
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.4))

    # Global average pooling to reduce dimensions and avoid overfitting
    model.add(GlobalAveragePooling2D())

    # Fully connected layer
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))

    # Output layer
    model.add(Dense(1, activation='sigmoid'))  # For binary classification (tumor vs non
-tumor)

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.00001), loss='binary_crossentropy', met
rics=['accuracy'])

    return model


# Create CNN model
baseline_model = baseline_cnn_model()

# Train the model
history = baseline_model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50
, batch_size=32, verbose=1)

# Plot the learning curves
plt.figure(figsize=(10, 5))
```

```
# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

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

plt.tight_layout()
plt.show()

baseline_model.save('./baseline_model.keras')
```

```
Epoch 1/50
22/22 [==============================] - 3s 102ms/step - loss: 0.7635 - accuracy: 0.4740
- val_loss: 0.7460 - val_accuracy: 0.5349
Epoch 2/50
22/22 [==============================] - 2s 83ms/step - loss: 0.7251 - accuracy: 0.5332 -
val_loss: 0.6942 - val_accuracy: 0.5233
Epoch 3/50
22/22 [==============================] - 4s 191ms/step - loss: 0.7315 - accuracy: 0.5130
- val_loss: 0.6813 - val_accuracy: 0.5116
Epoch 4/50
22/22 [==============================] - 4s 178ms/step - loss: 0.7391 - accuracy: 0.5275
- val_loss: 0.6765 - val_accuracy: 0.5233
Epoch 5/50
22/22 [==============================] - 4s 175ms/step - loss: 0.7136 - accuracy: 0.5708
- val_loss: 0.6756 - val_accuracy: 0.5698
Epoch 6/50
22/22 [==============================] - 4s 175ms/step - loss: 0.6852 - accuracy: 0.5997
- val_loss: 0.6726 - val_accuracy: 0.5465
Epoch 7/50
22/22 [==============================] - 4s 176ms/step - loss: 0.6752 - accuracy: 0.6272
- val_loss: 0.6704 - val_accuracy: 0.5233
Epoch 8/50
22/22 [==============================] - 4s 177ms/step - loss: 0.6892 - accuracy: 0.5968
- val_loss: 0.6686 - val_accuracy: 0.5233
Epoch 9/50
22/22 [==============================] - 4s 174ms/step - loss: 0.6683 - accuracy: 0.6142
- val_loss: 0.6658 - val_accuracy: 0.5581
Epoch 10/50
22/22 [==============================] - 4s 178ms/step - loss: 0.6869 - accuracy: 0.6113
- val_loss: 0.6639 - val_accuracy: 0.5465
Epoch 11/50
22/22 [==============================] - 4s 184ms/step - loss: 0.6807 - accuracy: 0.6069
- val_loss: 0.6613 - val_accuracy: 0.5814
Epoch 12/50
22/22 [==============================] - 4s 173ms/step - loss: 0.6693 - accuracy: 0.6113
- val_loss: 0.6593 - val_accuracy: 0.5698
Epoch 13/50
22/22 [==============================] - 4s 174ms/step - loss: 0.6781 - accuracy: 0.6416
- val_loss: 0.6567 - val_accuracy: 0.5930
Epoch 14/50
22/22 [==============================] - 4s 175ms/step - loss: 0.6727 - accuracy: 0.6040
- val_loss: 0.6545 - val_accuracy: 0.5930
Epoch 15/50
22/22 [==============================] - 4s 184ms/step - loss: 0.6486 - accuracy: 0.6358
- val_loss: 0.6529 - val_accuracy: 0.6163
Epoch 16/50
22/22 [==============================] - 4s 182ms/step - loss: 0.6631 - accuracy: 0.6315
- val_loss: 0.6510 - val_accuracy: 0.6047
```

```
Epoch 17/50
22/22 [==============================] - 6s 264ms/step - loss: 0.6827 - accuracy: 0.6098
- val_loss: 0.6494 - val_accuracy: 0.6047
Epoch 18/50
22/22 [==============================] - 4s 177ms/step - loss: 0.6562 - accuracy: 0.6387
- val_loss: 0.6480 - val_accuracy: 0.6163
Epoch 19/50
22/22 [==============================] - 4s 185ms/step - loss: 0.6505 - accuracy: 0.6286
- val_loss: 0.6466 - val_accuracy: 0.6163
Epoch 20/50
22/22 [==============================] - 4s 180ms/step - loss: 0.6514 - accuracy: 0.6431
- val_loss: 0.6455 - val_accuracy: 0.6047
Epoch 21/50
22/22 [==============================] - 4s 178ms/step - loss: 0.6604 - accuracy: 0.6286
- val_loss: 0.6443 - val_accuracy: 0.6163
Epoch 22/50
22/22 [==============================] - 4s 180ms/step - loss: 0.6485 - accuracy: 0.6358
- val_loss: 0.6430 - val_accuracy: 0.6163
Epoch 23/50
22/22 [==============================] - 4s 180ms/step - loss: 0.6415 - accuracy: 0.6373
- val_loss: 0.6421 - val_accuracy: 0.6163
Epoch 24/50
22/22 [==============================] - 4s 180ms/step - loss: 0.6574 - accuracy: 0.6243
- val_loss: 0.6410 - val_accuracy: 0.6047
Epoch 25/50
22/22 [==============================] - 4s 180ms/step - loss: 0.6440 - accuracy: 0.6532
- val_loss: 0.6398 - val_accuracy: 0.6047
Epoch 26/50
22/22 [==============================] - 4s 180ms/step - loss: 0.6665 - accuracy: 0.6329
- val_loss: 0.6393 - val_accuracy: 0.6047
Epoch 27/50
22/22 [==============================] - 4s 199ms/step - loss: 0.6312 - accuracy: 0.6720
- val_loss: 0.6377 - val_accuracy: 0.6047
Epoch 28/50
22/22 [==============================] - 4s 181ms/step - loss: 0.6258 - accuracy: 0.6676
- val_loss: 0.6367 - val_accuracy: 0.6163
Epoch 29/50
22/22 [==============================] - 4s 183ms/step - loss: 0.6542 - accuracy: 0.6618
- val_loss: 0.6354 - val_accuracy: 0.6047
Epoch 30/50
22/22 [==============================] - 4s 191ms/step - loss: 0.6325 - accuracy: 0.6445
- val_loss: 0.6355 - val_accuracy: 0.6163
Epoch 31/50
22/22 [==============================] - 4s 185ms/step - loss: 0.6588 - accuracy: 0.6460
- val_loss: 0.6345 - val_accuracy: 0.6047
Epoch 32/50
22/22 [==============================] - 4s 181ms/step - loss: 0.6409 - accuracy: 0.6474
- val_loss: 0.6335 - val_accuracy: 0.6163
Epoch 33/50
22/22 [==============================] - 4s 176ms/step - loss: 0.6324 - accuracy: 0.6445
- val_loss: 0.6328 - val_accuracy: 0.6047
Epoch 34/50
22/22 [==============================] - 4s 178ms/step - loss: 0.6515 - accuracy: 0.6561
- val_loss: 0.6326 - val_accuracy: 0.6163
Epoch 35/50
22/22 [==============================] - 4s 180ms/step - loss: 0.6065 - accuracy: 0.6777
- val_loss: 0.6322 - val_accuracy: 0.6163
Epoch 36/50
22/22 [==============================] - 4s 180ms/step - loss: 0.6187 - accuracy: 0.6590
- val_loss: 0.6316 - val_accuracy: 0.6163
Epoch 37/50
22/22 [==============================] - 4s 187ms/step - loss: 0.5944 - accuracy: 0.6908
- val_loss: 0.6309 - val_accuracy: 0.6163
Epoch 38/50
22/22 [==============================] - 4s 200ms/step - loss: 0.6114 - accuracy: 0.6806
- val_loss: 0.6306 - val_accuracy: 0.6163
Epoch 39/50
22/22 [==============================] - 6s 287ms/step - loss: 0.6194 - accuracy: 0.6633
- val_loss: 0.6297 - val_accuracy: 0.6047
Epoch 40/50
22/22 [==============================] - 6s 288ms/step - loss: 0.6049 - accuracy: 0.6662
- val_loss: 0.6289 - val_accuracy: 0.6047
```

```
Epoch 41/50
22/22 [==============================] - 5s 236ms/step - loss: 0.5967 - accuracy: 0.6864
- val_loss: 0.6288 - val_accuracy: 0.6047
Epoch 42/50
22/22 [==============================] - 4s 182ms/step - loss: 0.6140 - accuracy: 0.6777
- val_loss: 0.6284 - val_accuracy: 0.6163
Epoch 43/50
22/22 [==============================] - 4s 194ms/step - loss: 0.6042 - accuracy: 0.6908
- val_loss: 0.6286 - val_accuracy: 0.6163
Epoch 44/50
22/22 [==============================] - 5s 203ms/step - loss: 0.6149 - accuracy: 0.6546
- val_loss: 0.6282 - val_accuracy: 0.6047
Epoch 45/50
22/22 [==============================] - 4s 189ms/step - loss: 0.5914 - accuracy: 0.7168
- val_loss: 0.6274 - val_accuracy: 0.6047
Epoch 46/50
22/22 [==============================] - 4s 185ms/step - loss: 0.6048 - accuracy: 0.6879
- val_loss: 0.6271 - val_accuracy: 0.6047
Epoch 47/50
22/22 [==============================] - 4s 188ms/step - loss: 0.6063 - accuracy: 0.6734
- val_loss: 0.6258 - val_accuracy: 0.6047
Epoch 48/50
22/22 [==============================] - 4s 199ms/step - loss: 0.5877 - accuracy: 0.6821
- val_loss: 0.6252 - val_accuracy: 0.6047
Epoch 49/50
22/22 [==============================] - 4s 181ms/step - loss: 0.5969 - accuracy: 0.6792
- val_loss: 0.6250 - val_accuracy: 0.6047
Epoch 50/50
22/22 [==============================] - 4s 179ms/step - loss: 0.6041 - accuracy: 0.6951
- val_loss: 0.6250 - val_accuracy: 0.6047
```



Test

In [18]:

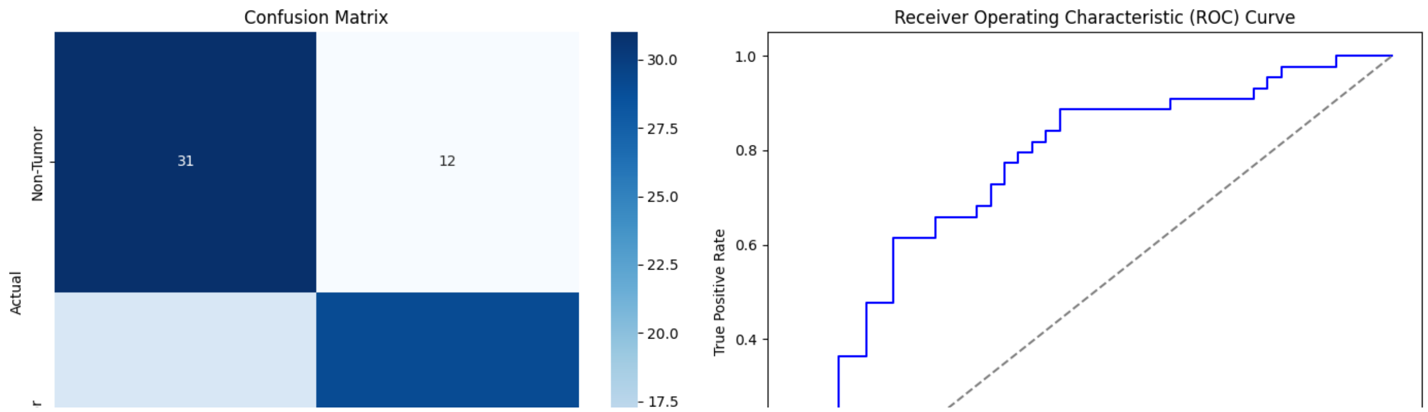
```python
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

def evaluate_model(model, X_test, y_test, threshold=0.5):

    # Predict probabilities or labels
    y_pred_prob = model.predict(X_test)
```

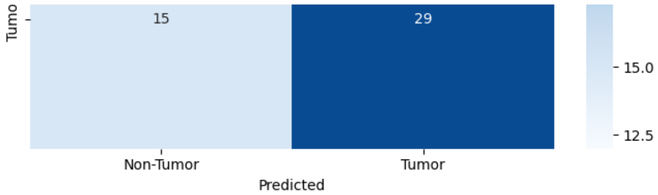
```
    # For binary classification, convert probabilities to labels
    if y_pred_prob.shape[1] == 1:  # Binary classification
        y_pred = (y_pred_prob > threshold).astype(int)
    else:  # For multi-class classification
        y_pred = np.argmax(y_pred_prob, axis=1)

    # Classification Report
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)

    # Create a 1x2 subplot layout (side-by-side)
    fig, axes = plt.subplots(1, 2, figsize=(14, 6))

    # Plot Confusion Matrix
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Tumor', 'Tumor'
], yticklabels=['Non-Tumor', 'Tumor'], ax=axes[0])
    axes[0].set_title('Confusion Matrix')
    axes[0].set_xlabel('Predicted')
    axes[0].set_ylabel('Actual')

    # ROC Curve
    if y_pred_prob.shape[1] == 1:  # Binary classification
        fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
        roc_auc = auc(fpr, tpr)

        axes[1].plot(fpr, tpr, color='b', label=f'ROC Curve (AUC = {roc_auc:.2f})')
        axes[1].plot([0, 1], [0, 1], color='gray', linestyle='--')
        axes[1].set_title('Receiver Operating Characteristic (ROC) Curve')
        axes[1].set_xlabel('False Positive Rate')
        axes[1].set_ylabel('True Positive Rate')
        axes[1].legend(loc='lower right')
    else:
        axes[1].text(0.5, 0.5, "ROC curve is only applicable for binary classification."
, horizontalalignment='center', verticalalignment='center', fontsize=12)
        axes[1].axis('off')

    # Show the plots
    plt.tight_layout()
    plt.show()
```

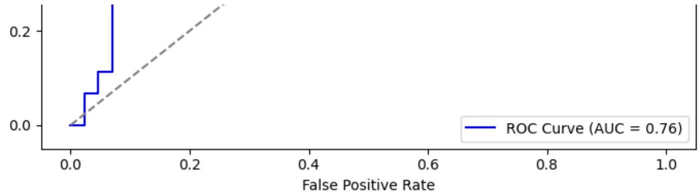
In [14]:

```
evaluate_model(baseline_model,X_test,y_test)
```

```
3/3 [==============================] - 0s 21ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.67      0.72      0.70        43
           1       0.71      0.66      0.68        44

    accuracy                           0.69        87
   macro avg       0.69      0.69      0.69        87
weighted avg       0.69      0.69      0.69        87
```



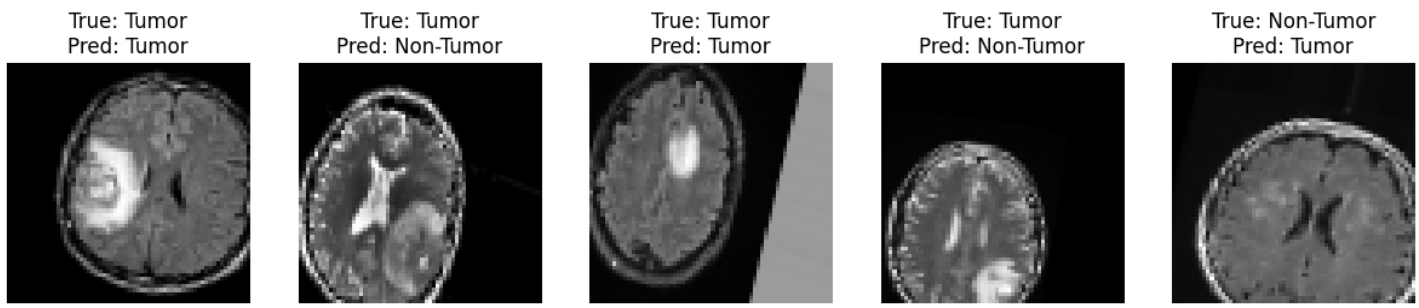




In [17]:

```
import matplotlib.pyplot as plt
import numpy as np

def show_predictions(model, X, y, num_samples=5):

    # Remove extra singleton dimensions (if any)
    X = np.squeeze(X)

    # Generate predictions
    y_pred = model.predict(X[:num_samples])

    # Plot the images with true and predicted labels
    plt.figure(figsize=(15, 5))
    for i in range(num_samples):
        plt.subplot(1, num_samples, i+1)

        # Ensure the image is in the correct format for displaying (height, width, channels)
        img = X[i]

        # Check if it's a grayscale or RGB image and process accordingly
        if img.ndim == 2:  # Grayscale image
            plt.imshow(img, cmap='gray')
        else:  # RGB image
            plt.imshow(img)

        plt.title(f"True: {'Tumor' if y[i] == 1 else 'Non-Tumor'}\nPred: {'Tumor' if y_pred[i] > 0.5 else 'Non-Tumor'}")
        plt.axis('off')

    plt.show()

# Example usage:
show_predictions(baseline_model, X_test, y_test, num_samples=5)

X_test.shape
```

```
1/1 [==============================] - 0s 103ms/step
```



Out[17]:

```
(87, 64, 64, 1, 1)
```

Hyperparameter tuning

In [18]:

```
import tensorflow as tf
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPooling2D, Dropout, G
lobalAveragePooling2D, Dense
from tensorflow.keras.optimizers import Adam
import keras_tuner as kt

# Define the CNN model with hyperparameters
def baseline_cnn_model(hp, input_shape=(64, 64, 1)):
    model = Sequential()

    # First convolutional block with hyperparameters for filters and dropout rate
    model.add(Conv2D(hp.Int('filters_1', min_value=32, max_value=128, step=32),
                     (3, 3),
                     activation='relu',
                     padding='same',
                     input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(hp.Float('dropout_1', min_value=0.2, max_value=0.5, step=0.1)))

    # Second convolutional block
    model.add(Conv2D(hp.Int('filters_2', min_value=64, max_value=256, step=64),
                     (3, 3),
                     activation='relu',
                     padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(hp.Float('dropout_2', min_value=0.2, max_value=0.5, step=0.1)))

    # Third convolutional block
    model.add(Conv2D(hp.Int('filters_3', min_value=128, max_value=512, step=128),
                     (3, 3),
                     activation='relu',
                     padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(hp.Float('dropout_3', min_value=0.3, max_value=0.6, step=0.1)))

    # Global average pooling to reduce dimensions and avoid overfitting
    model.add(GlobalAveragePooling2D())

    # Fully connected layer with hyperparameter for units and dropout rate
    model.add(Dense(hp.Int('dense_units', min_value=64, max_value=256, step=64), activat
ion='relu'))
    model.add(Dropout(hp.Float('dropout_4', min_value=0.3, max_value=0.6, step=0.1)))

    # Output layer
    model.add(Dense(1, activation='sigmoid'))  # For binary classification (tumor vs non
-tumor)

    # Compile the model with hyperparameter tuning for learning rate
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', min_value=1e-5,
max_value=1e-2, sampling='LOG')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

# Setup Keras Tuner for hyperparameter search
def hyperparameter_tuning(X_train, y_train, X_val, y_val):
    tuner = kt.RandomSearch(
        baseline_cnn_model,
        objective='val_accuracy',
        max_trials=10,  # Number of different hyperparameter combinations to try
        executions_per_trial=1,
        directory='tuner_dir',
        project_name='cnn_hyperparam_tuning'
    )

    # Perform hyperparameter search
    tuner.search(X_train, y_train, validation_data=(X_val, y_val), epochs=20, batch_size
```

```
=32)

    # Get the best model and hyperparameters
    best_model = tuner.get_best_models()[0]
    best_hyperparameters = tuner.get_best_hyperparameters()[0]

    print(f"Best Hyperparameters: {best_hyperparameters.values}")

    return best_model

# Train and evaluate the best model
def train_and_evaluate_model(X_train, y_train, X_val, y_val, X_test, y_test):
    # Run hyperparameter tuning
    best_model = hyperparameter_tuning(X_train, y_train, X_val, y_val)

    # Evaluate the best model on test data
    history = best_model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50
, batch_size=32)

    # Evaluate on test set
    test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
    print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")

    # Use your previous evaluation function to plot confusion matrix and ROC curve


    evaluate_model(best_model, X_test, y_test)

    # Return the best model and training history
    return best_model, history

# Assuming X_train, y_train, X_val, y_val, X_test, y_test are already prepared
best_model, history = train_and_evaluate_model(X_train, y_train, X_val, y_val, X_test, y
_test)
```

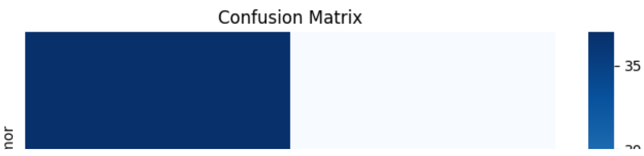
```
Trial 10 Complete [00h 11m 36s]
val_accuracy: 0.7790697813034058

Best val_accuracy So Far: 0.8720930218696594
Total elapsed time: 02h 18m 22s
```

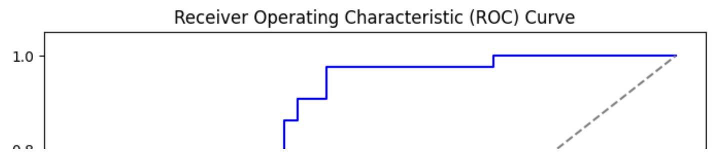
```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on
M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizer
s.legacy.Adam`.
```

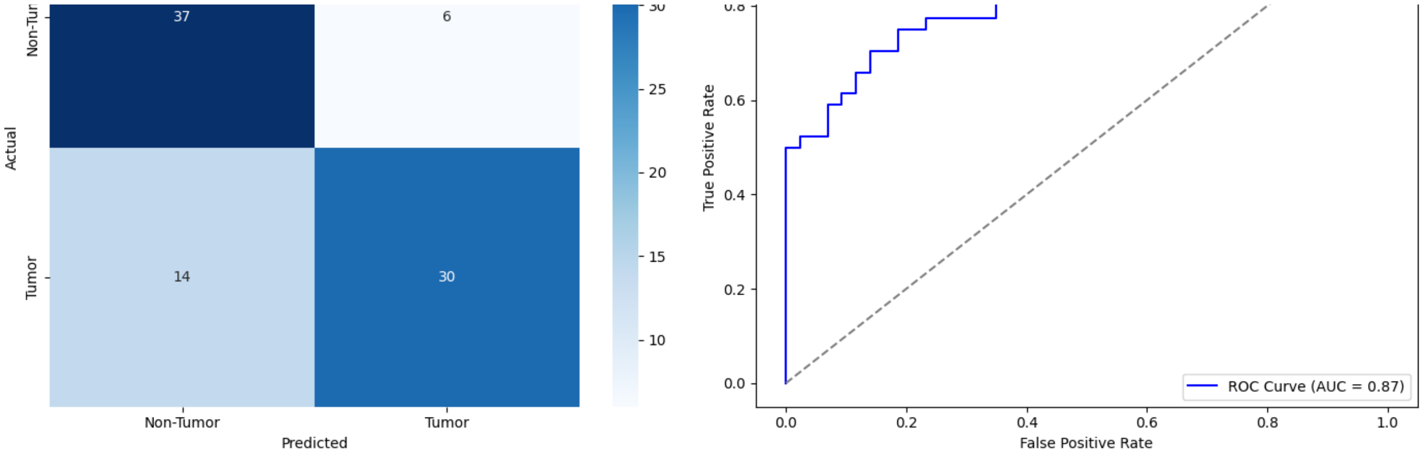
```
Best Hyperparameters: {'filters_1': 128, 'dropout_1': 0.2, 'filters_2': 64, 'dropout_2':
0.2, 'filters_3': 384, 'dropout_3': 0.3, 'dense_units': 256, 'dropout_4': 0.3, 'learning_
rate': 0.005747270907290 9165}
Epoch 1/50
22/22 [==============================] - 23s 1s/step - loss: 0.2422 - accuracy: 0.9104 -
val_loss: 0.2983 - val_accuracy: 0.8488
Epoch 2/50
22/22 [==============================] - 13s 580ms/step - loss: 0.2725 - accuracy: 0.8887
- val_loss: 0.7013 - val_accuracy: 0.7558
Epoch 3/50
22/22 [==============================] - 12s 555ms/step - loss: 0.2129 - accuracy: 0.9147
- val_loss: 0.7646 - val_accuracy: 0.7093
Epoch 4/50
22/22 [==============================] - 13s 584ms/step - loss: 0.1599 - accuracy: 0.9350
- val_loss: 0.2496 - val_accuracy: 0.8721
Epoch 5/50
22/22 [==============================] - 14s 623ms/step - loss: 0.1671 - accuracy: 0.9306
- val_loss: 0.5580 - val_accuracy: 0.8140
Epoch 6/50
22/22 [==============================] - 19s 858ms/step - loss: 0.1351 - accuracy: 0.9465
- val_loss: 0.3571 - val_accuracy: 0.8488
Epoch 7/50
22/22 [==============================] - 14s 644ms/step - loss: 0.1597 - accuracy: 0.9379
- val_loss: 0.6539 - val_accuracy: 0.8256
Epoch 8/50
22/22 [==============================] - 13s 594ms/step - loss: 0.1302 - accuracy: 0.9480
- val_loss: 0.8800 - val_accuracy: 0.7558
Epoch 9/50
```

```
Epoch 9/50
22/22 [==============================] - 11s 508ms/step - loss: 0.1791 - accuracy: 0.9321
- val_loss: 2.4594 - val_accuracy: 0.5698
Epoch 10/50
22/22 [==============================] - 11s 505ms/step - loss: 0.1099 - accuracy: 0.9639
- val_loss: 0.2711 - val_accuracy: 0.8721
Epoch 11/50
22/22 [==============================] - 11s 507ms/step - loss: 0.1270 - accuracy: 0.9566
- val_loss: 0.9521 - val_accuracy: 0.6512
Epoch 12/50
22/22 [==============================] - 11s 513ms/step - loss: 0.1258 - accuracy: 0.9509
- val_loss: 0.7154 - val_accuracy: 0.7674
Epoch 13/50
22/22 [==============================] - 12s 557ms/step - loss: 0.1471 - accuracy: 0.9509
- val_loss: 0.8103 - val_accuracy: 0.7907
Epoch 14/50
22/22 [==============================] - 11s 505ms/step - loss: 0.1795 - accuracy: 0.9263
- val_loss: 0.3388 - val_accuracy: 0.8953
Epoch 15/50
22/22 [==============================] - 11s 516ms/step - loss: 0.1176 - accuracy: 0.9552
- val_loss: 0.3194 - val_accuracy: 0.8953
Epoch 16/50
22/22 [==============================] - 11s 513ms/step - loss: 0.1053 - accuracy: 0.9566
- val_loss: 1.1399 - val_accuracy: 0.7442
Epoch 17/50
22/22 [==============================] - 12s 530ms/step - loss: 0.1479 - accuracy: 0.9465
- val_loss: 0.7110 - val_accuracy: 0.8140
Epoch 18/50
22/22 [==============================] - 12s 531ms/step - loss: 0.1805 - accuracy: 0.9292
- val_loss: 0.9333 - val_accuracy: 0.7442
Epoch 19/50
22/22 [==============================] - 11s 501ms/step - loss: 0.0911 - accuracy: 0.9697
- val_loss: 3.9153 - val_accuracy: 0.5814
Epoch 20/50
22/22 [==============================] - 19s 879ms/step - loss: 0.1144 - accuracy: 0.9538
- val_loss: 0.8031 - val_accuracy: 0.8023
Epoch 21/50
22/22 [==============================] - 23s 989ms/step - loss: 0.0584 - accuracy: 0.9812
- val_loss: 0.1374 - val_accuracy: 0.9186
Epoch 22/50
22/22 [==============================] - 12s 538ms/step - loss: 0.0528 - accuracy: 0.9855
- val_loss: 0.4214 - val_accuracy: 0.8605
Epoch 23/50
22/22 [==============================] - 11s 507ms/step - loss: 0.0666 - accuracy: 0.9725
- val_loss: 0.4535 - val_accuracy: 0.8837
Epoch 24/50
22/22 [==============================] - 11s 509ms/step - loss: 0.0767 - accuracy: 0.9740
- val_loss: 0.4066 - val_accuracy: 0.8953
Epoch 25/50
22/22 [==============================] - 11s 503ms/step - loss: 0.0926 - accuracy: 0.9653
- val_loss: 0.7657 - val_accuracy: 0.8488
Epoch 26/50
22/22 [==============================] - 11s 507ms/step - loss: 0.0634 - accuracy: 0.9841
- val_loss: 1.9023 - val_accuracy: 0.7326
Epoch 27/50
22/22 [==============================] - 11s 516ms/step - loss: 0.0508 - accuracy: 0.9827
- val_loss: 0.4786 - val_accuracy: 0.8488
Epoch 28/50
22/22 [==============================] - 11s 505ms/step - loss: 0.0360 - accuracy: 0.9870
- val_loss: 0.2758 - val_accuracy: 0.8953
Epoch 29/50
22/22 [==============================] - 11s 513ms/step - loss: 0.0430 - accuracy: 0.9841
- val_loss: 0.5912 - val_accuracy: 0.8488
Epoch 30/50
22/22 [==============================] - 13s 592ms/step - loss: 0.1093 - accuracy: 0.9624
- val_loss: 0.4285 - val_accuracy: 0.8256
Epoch 31/50
22/22 [==============================] - 11s 503ms/step - loss: 0.0768 - accuracy: 0.9740
- val_loss: 2.1050 - val_accuracy: 0.6279
Epoch 32/50
22/22 [==============================] - 11s 504ms/step - loss: 0.0628 - accuracy: 0.9798
- val_loss: 0.7480 - val_accuracy: 0.8023
Epoch 33/50
```

```
Epoch 33/50
22/22 [==============================] - 12s 553ms/step - loss: 0.0610 - accuracy: 0.9812
- val_loss: 0.3231 - val_accuracy: 0.8605
Epoch 34/50
22/22 [==============================] - 11s 505ms/step - loss: 0.0308 - accuracy: 0.9913
- val_loss: 0.3906 - val_accuracy: 0.8953
Epoch 35/50
22/22 [==============================] - 12s 526ms/step - loss: 0.0365 - accuracy: 0.9884
- val_loss: 0.5316 - val_accuracy: 0.8721
Epoch 36/50
22/22 [==============================] - 16s 756ms/step - loss: 0.0394 - accuracy: 0.9870
- val_loss: 0.1217 - val_accuracy: 0.9535
Epoch 37/50
22/22 [==============================] - 20s 913ms/step - loss: 0.0389 - accuracy: 0.9812
- val_loss: 0.2278 - val_accuracy: 0.9419
Epoch 38/50
22/22 [==============================] - 12s 559ms/step - loss: 0.0389 - accuracy: 0.9827
- val_loss: 0.2489 - val_accuracy: 0.9186
Epoch 39/50
22/22 [==============================] - 12s 556ms/step - loss: 0.0583 - accuracy: 0.9855
- val_loss: 0.2290 - val_accuracy: 0.8953
Epoch 40/50
22/22 [==============================] - 13s 596ms/step - loss: 0.1660 - accuracy: 0.9494
- val_loss: 0.3035 - val_accuracy: 0.8953
Epoch 41/50
22/22 [==============================] - 13s 605ms/step - loss: 0.1192 - accuracy: 0.9610
- val_loss: 0.3681 - val_accuracy: 0.8488
Epoch 42/50
22/22 [==============================] - 13s 570ms/step - loss: 0.0789 - accuracy: 0.9697
- val_loss: 0.2876 - val_accuracy: 0.9070
Epoch 43/50
22/22 [==============================] - 12s 553ms/step - loss: 0.0312 - accuracy: 0.9913
- val_loss: 0.2483 - val_accuracy: 0.9186
Epoch 44/50
22/22 [==============================] - 12s 551ms/step - loss: 0.0331 - accuracy: 0.9899
- val_loss: 0.4625 - val_accuracy: 0.8605
Epoch 45/50
22/22 [==============================] - 12s 550ms/step - loss: 0.0371 - accuracy: 0.9870
- val_loss: 0.5184 - val_accuracy: 0.8721
Epoch 46/50
22/22 [==============================] - 13s 578ms/step - loss: 0.0330 - accuracy: 0.9870
- val_loss: 0.4343 - val_accuracy: 0.8953
Epoch 47/50
22/22 [==============================] - 12s 550ms/step - loss: 0.0577 - accuracy: 0.9798
- val_loss: 0.1844 - val_accuracy: 0.9302
Epoch 48/50
22/22 [==============================] - 12s 562ms/step - loss: 0.0800 - accuracy: 0.9725
- val_loss: 4.8360 - val_accuracy: 0.6395
Epoch 49/50
22/22 [==============================] - 12s 547ms/step - loss: 0.1077 - accuracy: 0.9624
- val_loss: 0.5108 - val_accuracy: 0.8605
Epoch 50/50
22/22 [==============================] - 14s 658ms/step - loss: 0.0617 - accuracy: 0.9754
- val_loss: 0.4631 - val_accuracy: 0.8140
3/3 [==============================] - 1s 290ms/step - loss: 0.7695 - accuracy: 0.7701
Test Loss: 0.7695075869560242, Test Accuracy: 0.7701149582862854
3/3 [==============================] - 1s 206ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.86      0.79        43
           1       0.83      0.68      0.75        44

    accuracy                           0.77        87
   macro avg       0.78      0.77      0.77        87
weighted avg       0.78      0.77      0.77        87
```







In [23]:

```
best_model.save('./finetuned_model.keras')
```

Pre trained model Resnet 50

In [ ]:

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers.legacy import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

# Adjust input shape to match ResNet requirements
input_shape = (64, 64, 3)  # ResNet expects 3 channels

X_train = np.squeeze(X_train, axis=-1)  # Remove the last dimension
X_val = np.squeeze(X_val, axis=-1)
X_test = np.squeeze(X_test, axis=-1)

X_train_rgb = np.repeat(X_train, 3, axis=-1)  # (692, 64, 64, 1) -> (692, 64, 64, 3)
X_val_rgb = np.repeat(X_val, 3, axis=-1)
X_test_rgb = np.repeat(X_test, 3, axis=-1)

# Load pretrained ResNet50 with weights from ImageNet
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)

# Freeze the base layers
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers for fine-tuning
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

# Create the model
resnet_model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
resnet_model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', m
etrics=['accuracy'])

# Train the model
history = resnet_model.fit(
    X_train_rgb, y_train,
    validation_data=(X_val_rgb, y_val),
```

```python
    epochs=20,
    batch_size=32,
    verbose=1
)

# Unfreeze some layers for fine-tuning
for layer in base_model.layers[-20:]:  # Unfreeze last 20 layers
    layer.trainable = True

# Recompile the model with a lower learning rate for fine-tuning
resnet_model.compile(optimizer=Adam(learning_rate=0.00001), loss='binary_crossentropy',
metrics=['accuracy'])

# Fine-tune the model
history_fine_tune = resnet_model.fit(
    X_train_rgb, y_train,
    validation_data=(X_val_rgb, y_val),
    epochs=20,
    batch_size=32,
    verbose=1
)

# Evaluate on test set
results = resnet_model.evaluate(X_test_rgb, y_test, verbose=1)
print(f"Test Accuracy: {results[1] * 100:.2f}%")
```

```
Epoch 1/20
22/22 [==============================] - 11s 396ms/step - loss: 1.2011 - accuracy: 0.6098
- val_loss: 0.6500 - val_accuracy: 0.6860
Epoch 2/20
22/22 [==============================] - 7s 326ms/step - loss: 0.6505 - accuracy: 0.7298
- val_loss: 0.5923 - val_accuracy: 0.7442
Epoch 3/20
22/22 [==============================] - 7s 315ms/step - loss: 0.6261 - accuracy: 0.7587
- val_loss: 0.5500 - val_accuracy: 0.7674
Epoch 4/20
22/22 [==============================] - 7s 333ms/step - loss: 0.4495 - accuracy: 0.8165
- val_loss: 0.4989 - val_accuracy: 0.7791
Epoch 5/20
22/22 [==============================] - 7s 320ms/step - loss: 0.4423 - accuracy: 0.8121
- val_loss: 0.4968 - val_accuracy: 0.7791
Epoch 6/20
22/22 [==============================] - 7s 322ms/step - loss: 0.3553 - accuracy: 0.8353
- val_loss: 0.4577 - val_accuracy: 0.8023
Epoch 7/20
22/22 [==============================] - 7s 323ms/step - loss: 0.3372 - accuracy: 0.8584
- val_loss: 0.4457 - val_accuracy: 0.8256
Epoch 8/20
22/22 [==============================] - 7s 311ms/step - loss: 0.3126 - accuracy: 0.8555
- val_loss: 0.4280 - val_accuracy: 0.8140
Epoch 9/20
22/22 [==============================] - 7s 312ms/step - loss: 0.2613 - accuracy: 0.8786
- val_loss: 0.4361 - val_accuracy: 0.8256
Epoch 10/20
22/22 [==============================] - 7s 311ms/step - loss: 0.2708 - accuracy: 0.8931
- val_loss: 0.4066 - val_accuracy: 0.8372
Epoch 11/20
22/22 [==============================] - 7s 329ms/step - loss: 0.2640 - accuracy: 0.8858
- val_loss: 0.3822 - val_accuracy: 0.8372
Epoch 12/20
22/22 [==============================] - 7s 327ms/step - loss: 0.2391 - accuracy: 0.8945
- val_loss: 0.3842 - val_accuracy: 0.8372
Epoch 13/20
22/22 [==============================] - 7s 311ms/step - loss: 0.1894 - accuracy: 0.9277
- val_loss: 0.3802 - val_accuracy: 0.8372
Epoch 14/20
22/22 [==============================] - 7s 316ms/step - loss: 0.1948 - accuracy: 0.9234
- val_loss: 0.3915 - val_accuracy: 0.8605
Epoch 15/20
22/22 [==============================] - 7s 318ms/step - loss: 0.1836 - accuracy: 0.9292
- val_loss: 0.3808 - val_accuracy: 0.8605
Epoch 16/20
```

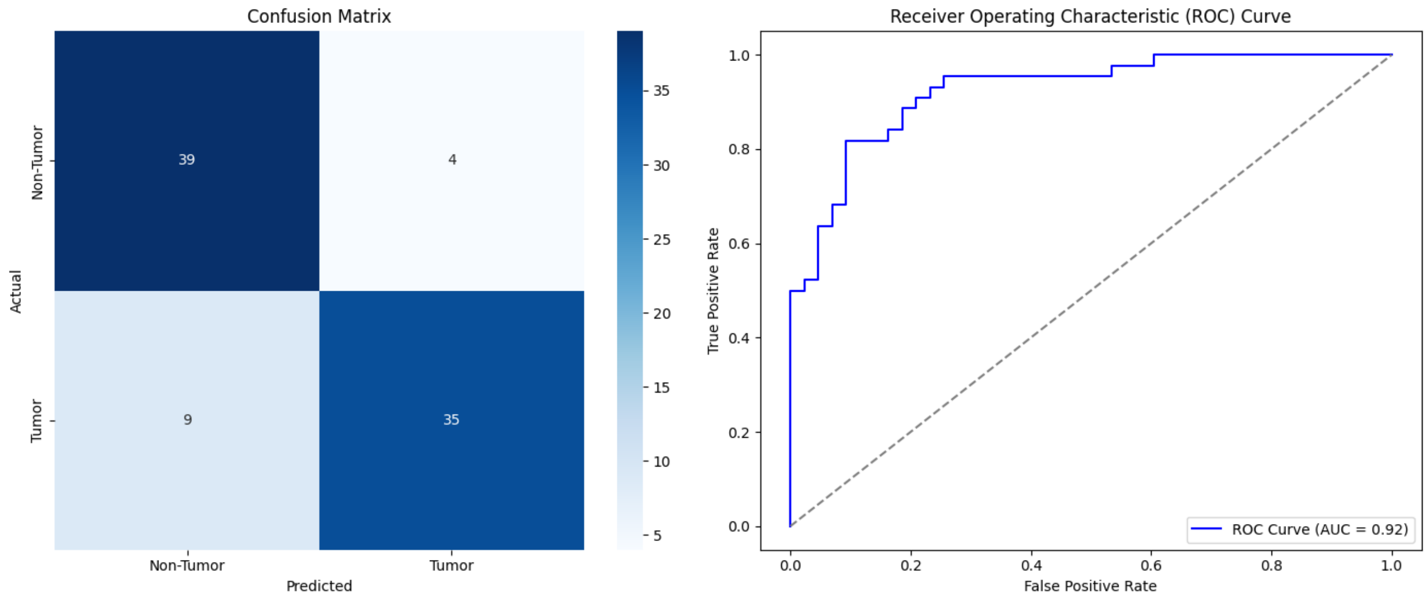
```
22/22 [==============================] - 7s 339ms/step - loss: 0.1740 - accuracy: 0.9176
- val_loss: 0.3733 - val_accuracy: 0.8605
Epoch 17/20
22/22 [==============================] - 7s 325ms/step - loss: 0.1644 - accuracy: 0.9465
- val_loss: 0.3822 - val_accuracy: 0.8605
Epoch 18/20
22/22 [==============================] - 7s 315ms/step - loss: 0.1781 - accuracy: 0.9292
- val_loss: 0.3887 - val_accuracy: 0.8605
Epoch 19/20
22/22 [==============================] - 7s 318ms/step - loss: 0.1654 - accuracy: 0.9292
- val_loss: 0.3791 - val_accuracy: 0.8372
Epoch 20/20
22/22 [==============================] - 7s 317ms/step - loss: 0.1511 - accuracy: 0.9465
- val_loss: 0.3793 - val_accuracy: 0.8605
Epoch 1/20
22/22 [==============================] - 15s 572ms/step - loss: 0.3326 - accuracy: 0.8526
- val_loss: 0.3749 - val_accuracy: 0.8372
Epoch 2/20
22/22 [==============================] - 12s 547ms/step - loss: 0.2609 - accuracy: 0.8931
- val_loss: 0.3678 - val_accuracy: 0.8372
Epoch 3/20
22/22 [==============================] - 12s 538ms/step - loss: 0.2146 - accuracy: 0.9191
- val_loss: 0.3578 - val_accuracy: 0.8605
Epoch 4/20
22/22 [==============================] - 15s 700ms/step - loss: 0.1961 - accuracy: 0.9176
- val_loss: 0.3510 - val_accuracy: 0.8488
Epoch 5/20
22/22 [==============================] - 14s 637ms/step - loss: 0.1801 - accuracy: 0.9306
- val_loss: 0.3411 - val_accuracy: 0.8488
Epoch 6/20
22/22 [==============================] - 12s 557ms/step - loss: 0.1453 - accuracy: 0.9552
- val_loss: 0.3392 - val_accuracy: 0.8372
Epoch 7/20
22/22 [==============================] - 12s 558ms/step - loss: 0.1302 - accuracy: 0.9668
- val_loss: 0.3357 - val_accuracy: 0.8372
Epoch 8/20
22/22 [==============================] - 13s 614ms/step - loss: 0.1190 - accuracy: 0.9653
- val_loss: 0.3358 - val_accuracy: 0.8372
Epoch 9/20
22/22 [==============================] - 12s 558ms/step - loss: 0.0967 - accuracy: 0.9812
- val_loss: 0.3370 - val_accuracy: 0.8256
Epoch 10/20
22/22 [==============================] - 12s 557ms/step - loss: 0.1032 - accuracy: 0.9697
- val_loss: 0.3332 - val_accuracy: 0.8256
Epoch 11/20
22/22 [==============================] - 12s 565ms/step - loss: 0.0925 - accuracy: 0.9740
- val_loss: 0.3319 - val_accuracy: 0.8256
Epoch 12/20
22/22 [==============================] - 12s 549ms/step - loss: 0.0818 - accuracy: 0.9855
- val_loss: 0.3277 - val_accuracy: 0.8256
Epoch 13/20
22/22 [==============================] - 12s 552ms/step - loss: 0.0669 - accuracy: 0.9928
- val_loss: 0.3268 - val_accuracy: 0.8256
Epoch 14/20
22/22 [==============================] - 12s 565ms/step - loss: 0.0690 - accuracy: 0.9884
- val_loss: 0.3324 - val_accuracy: 0.8256
Epoch 15/20
22/22 [==============================] - 12s 544ms/step - loss: 0.0532 - accuracy: 0.9971
- val_loss: 0.3336 - val_accuracy: 0.8372
Epoch 16/20
22/22 [==============================] - 13s 591ms/step - loss: 0.0520 - accuracy: 0.9942
- val_loss: 0.3353 - val_accuracy: 0.8488
Epoch 17/20
22/22 [==============================] - 13s 604ms/step - loss: 0.0534 - accuracy: 0.9899
- val_loss: 0.3356 - val_accuracy: 0.8372
Epoch 18/20
22/22 [==============================] - 14s 629ms/step - loss: 0.0462 - accuracy: 0.9971
- val_loss: 0.3346 - val_accuracy: 0.8372
Epoch 19/20
22/22 [==============================] - 17s 767ms/step - loss: 0.0451 - accuracy: 0.9913
- val_loss: 0.3323 - val_accuracy: 0.8488
Epoch 20/20
```

```
22/22 [==============================] - 13s 592ms/step - loss: 0.0423 - accuracy: 0.9957
- val_loss: 0.3317 - val_accuracy: 0.8605
3/3 [==============================] - 1s 254ms/step - loss: 0.3944 - accuracy: 0.8506
Test Accuracy: 85.06%

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[17], line 69
     66 print(f"Test Accuracy: {results[1] * 100:.2f}%")
     68 # Use evaluation function for metrics and visualizations
---> 69 evaluate_model(resnet_model, X_test_rgb, y_test)
     71 resnet_model.save('./resnet_model.keras')

NameError: name 'evaluate_model' is not defined
```

In [19]:

```
# Use evaluation function for metrics and visualizations
evaluate_model(resnet_model, X_test_rgb, y_test)

resnet_model.save('./resnet_model.keras')
```

```
3/3 [==============================] - 1s 127ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.86        43
           1       0.90      0.80      0.84        44

    accuracy                           0.85        87
   macro avg       0.85      0.85      0.85        87
weighted avg       0.86      0.85      0.85        87
```



VGG

In [ ]:

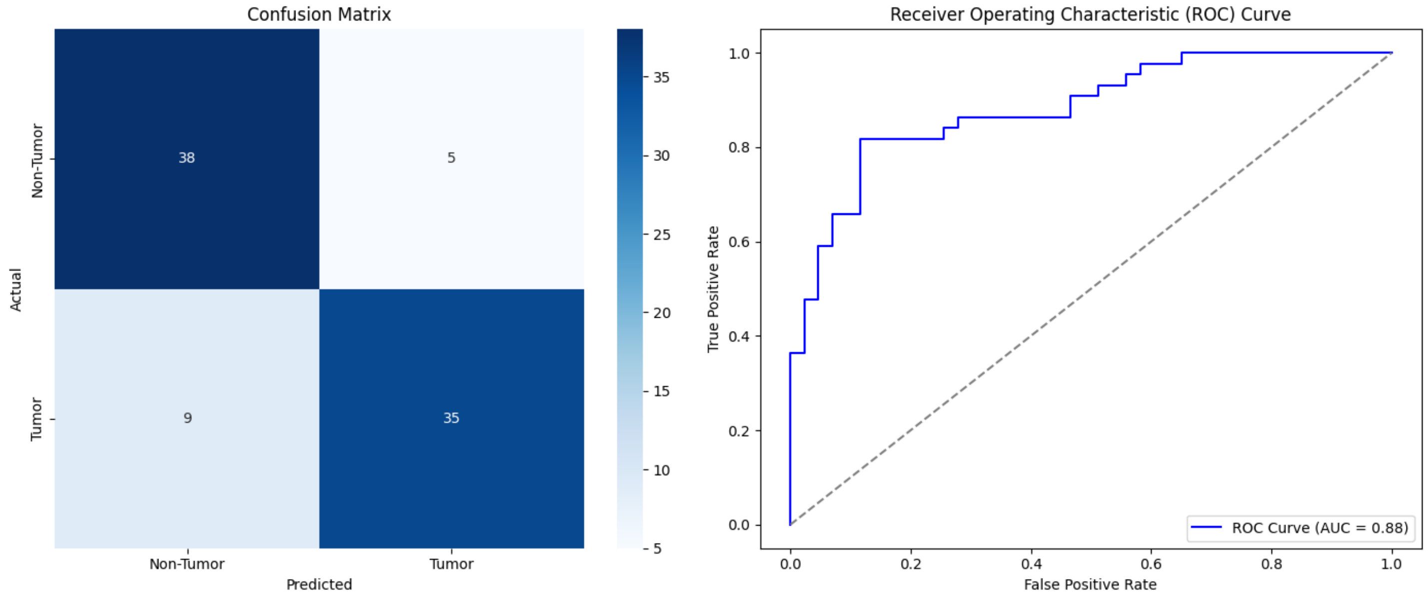
```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers.legacy import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Load the VGG16 model with pre-trained ImageNet weights
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3))
```

```python
# Freeze the base model layers for initial training
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers for binary classification
x = base_model.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(1, activation='sigmoid')(x)  # Binary classification output

# Create the final model
vgg_model = Model(inputs=base_model.input, outputs=output)

# Compile the model
vgg_model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics
=['accuracy'])

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model with the base layers frozen
history = vgg_model.fit(
    X_train_rgb, y_train,
    validation_data=(X_val_rgb, y_val),
    epochs=20,
    batch_size=32,
    callbacks=[early_stopping],
    verbose=1
)

# Fine-tuning: Unfreeze some layers
for layer in base_model.layers[-20:]:  # Unfreeze last 20 layers
    layer.trainable = True

# Recompile the model with a lower learning rate for fine-tuning
vgg_model.compile(optimizer=Adam(learning_rate=1e-5), loss='binary_crossentropy', metrics
=['accuracy'])

# Fine-tune the model
history_fine_tune = vgg_model.fit(
    X_train_rgb, y_train,
    validation_data=(X_val_rgb, y_val),
    epochs=20,
    batch_size=32,
    callbacks=[early_stopping],
    verbose=1
)

# Evaluate the model on the test set
evaluate_model(vgg_model, X_test_rgb, y_test)

# Save the fine-tuned model
```

```
Epoch 1/20
22/22 [==============================] - 18s 792ms/step - loss: 5.8054 - accuracy: 0.5246
- val_loss: 1.5412 - val_accuracy: 0.7326
Epoch 2/20
22/22 [==============================] - 19s 887ms/step - loss: 3.4765 - accuracy: 0.6691
- val_loss: 1.4337 - val_accuracy: 0.8023
Epoch 3/20
22/22 [==============================] - 20s 926ms/step - loss: 2.6054 - accuracy: 0.7052
- val_loss: 1.3322 - val_accuracy: 0.7907
Epoch 4/20
22/22 [==============================] - 20s 904ms/step - loss: 2.2957 - accuracy: 0.7500
- val_loss: 1.2653 - val_accuracy: 0.8256
Epoch 5/20
22/22 [==============================] - 20s 900ms/step - loss: 1.6504 - accuracy: 0.8150
- val_loss: 1.1863 - val_accuracy: 0.8140
Epoch 6/20
22/22 [==============================] - 26s 1s/step - loss: 1.5627 - accuracy: 0.8078 -
val_loss: 1.0699 - val_accuracy: 0.8488
```

```
Epoch 7/20
22/22 [==============================] - 20s 912ms/step - loss: 1.3226 - accuracy: 0.8223
- val_loss: 1.0869 - val_accuracy: 0.8605
Epoch 8/20
22/22 [==============================] - 20s 922ms/step - loss: 1.1878 - accuracy: 0.8410
- val_loss: 0.9755 - val_accuracy: 0.8605
Epoch 9/20
22/22 [==============================] - 20s 923ms/step - loss: 1.0183 - accuracy: 0.8497
- val_loss: 0.8841 - val_accuracy: 0.8721
Epoch 10/20
22/22 [==============================] - 21s 974ms/step - loss: 0.8677 - accuracy: 0.8699
- val_loss: 0.8913 - val_accuracy: 0.8953
Epoch 11/20
22/22 [==============================] - 20s 908ms/step - loss: 0.7541 - accuracy: 0.8656
- val_loss: 0.8624 - val_accuracy: 0.8605
Epoch 12/20
22/22 [==============================] - 19s 882ms/step - loss: 0.7545 - accuracy: 0.8685
- val_loss: 0.8503 - val_accuracy: 0.8953
Epoch 13/20
22/22 [==============================] - 20s 907ms/step - loss: 0.6807 - accuracy: 0.8714
- val_loss: 0.8667 - val_accuracy: 0.8837
Epoch 14/20
22/22 [==============================] - 20s 929ms/step - loss: 0.4753 - accuracy: 0.8945
- val_loss: 0.7868 - val_accuracy: 0.8837
Epoch 15/20
22/22 [==============================] - 29s 1s/step - loss: 0.6419 - accuracy: 0.8844 -
val_loss: 0.7864 - val_accuracy: 0.8721
Epoch 16/20
22/22 [==============================] - 21s 958ms/step - loss: 0.4385 - accuracy: 0.9046
- val_loss: 0.7767 - val_accuracy: 0.9302
Epoch 17/20
22/22 [==============================] - 21s 942ms/step - loss: 0.4149 - accuracy: 0.9017
- val_loss: 0.7326 - val_accuracy: 0.9186
Epoch 18/20
22/22 [==============================] - 22s 997ms/step - loss: 0.4338 - accuracy: 0.9090
- val_loss: 0.6983 - val_accuracy: 0.9070
Epoch 19/20
22/22 [==============================] - 21s 939ms/step - loss: 0.3239 - accuracy: 0.9090
- val_loss: 0.7412 - val_accuracy: 0.9186
Epoch 20/20
22/22 [==============================] - 20s 929ms/step - loss: 0.2113 - accuracy: 0.9379
- val_loss: 0.7597 - val_accuracy: 0.9186
Epoch 1/20
22/22 [==============================] - 73s 3s/step - loss: 0.5446 - accuracy: 0.8844 -
val_loss: 0.8754 - val_accuracy: 0.8605
Epoch 2/20
22/22 [==============================] - 71s 3s/step - loss: 0.2902 - accuracy: 0.9176 -
val_loss: 0.4839 - val_accuracy: 0.8837
Epoch 3/20
22/22 [==============================] - 72s 3s/step - loss: 0.1703 - accuracy: 0.9379 -
val_loss: 0.6514 - val_accuracy: 0.8953
Epoch 4/20
22/22 [==============================] - 76s 3s/step - loss: 0.0881 - accuracy: 0.9639 -
val_loss: 0.6236 - val_accuracy: 0.8953
Epoch 5/20
22/22 [==============================] - 69s 3s/step - loss: 0.0571 - accuracy: 0.9841 -
val_loss: 0.5643 - val_accuracy: 0.8953
Epoch 6/20
22/22 [==============================] - 82s 4s/step - loss: 0.0622 - accuracy: 0.9740 -
val_loss: 0.6040 - val_accuracy: 0.8953
Epoch 7/20
22/22 [==============================] - 75s 3s/step - loss: 0.0329 - accuracy: 0.9884 -
val_loss: 0.5331 - val_accuracy: 0.9070
3/3 [==============================] - 3s 970ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        43
           1       0.88      0.80      0.83        44

    accuracy                           0.84        87
   macro avg       0.84      0.84      0.84        87
```

```
weighted avg       0.84      0.84      0.84        87
```



In [22]:

```
vgg_model.save('./vgg_model.keras')
```

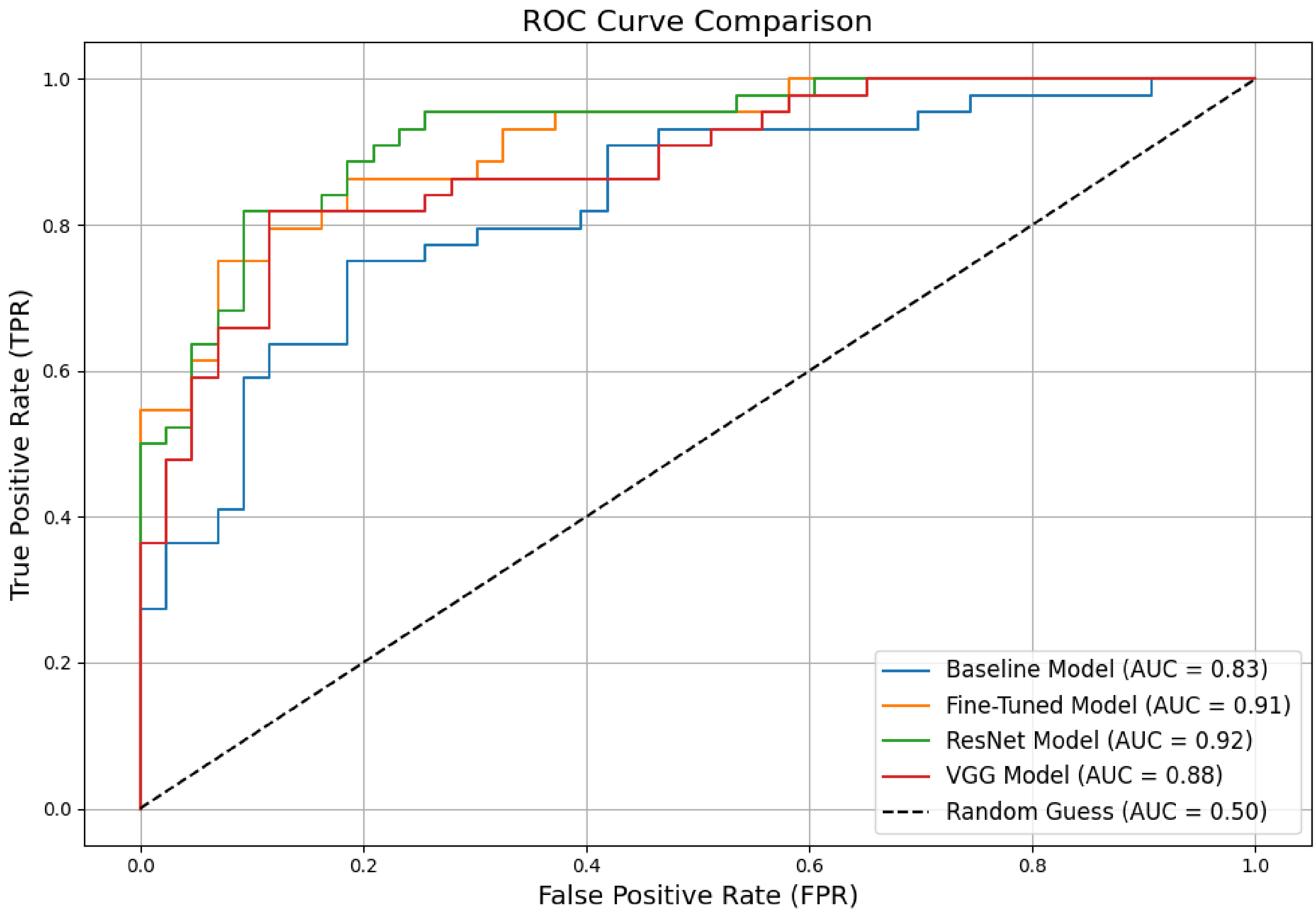
In [26]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from tensorflow.keras.models import load_model

# Function to compute ROC and AUC for a model
def compute_roc_auc(model_path, X_test, y_test, use_rgb=False):
    model = load_model(model_path)  # Load the model
    X = X_test_rgb if use_rgb else X_test  # Select the appropriate test data
    y_pred_prob = model.predict(X)  # Predict probabilities
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    return fpr, tpr, roc_auc

# Specify models and whether they require RGB input
models = {
    "Baseline Model": {"path": "./models/baseline_model.keras", "use_rgb": False},
    "Fine-Tuned Model": {"path": "./models/finetuned_model.keras", "use_rgb": False},
    "ResNet Model": {"path": "./models/resnet_model.keras", "use_rgb": True},
    "VGG Model": {"path": "./models/vgg_model.keras", "use_rgb": True},
}

plt.figure(figsize=(10, 7))
for model_name, details in models.items():
    fpr, tpr, roc_auc = compute_roc_auc(
        details["path"], X_test, y_test, use_rgb=details["use_rgb"]
    )
    plt.plot(fpr, tpr, label=f"{model_name} (AUC = {roc_auc:.2f})")

# Plot aesthetics
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess (AUC = 0.50)")
plt.title("ROC Curve Comparison", fontsize=16)
plt.xlabel("False Positive Rate (FPR)", fontsize=14)
plt.ylabel("True Positive Rate (TPR)", fontsize=14)
plt.legend(loc="lower right", fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
3/3 [==============================] - 0s 23ms/step
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on
M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizer
s.legacy.Adam`.
```

```
3/3 [==============================] - 0s 43ms/step
3/3 [==============================] - 1s 87ms/step
3/3 [==============================] - 1s 228ms/step
```



In [27]:

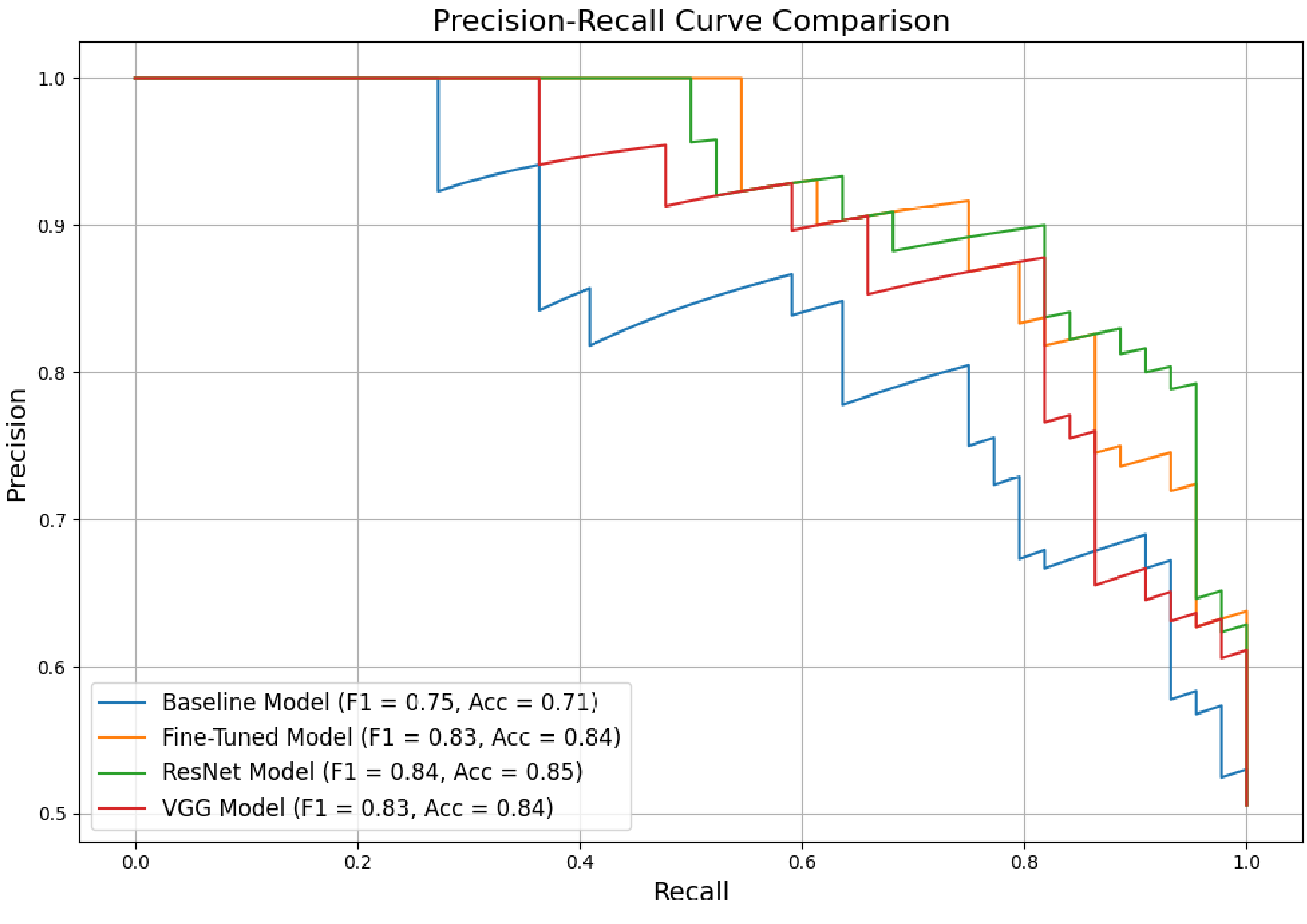
```
from sklearn.metrics import precision_recall_curve, f1_score, accuracy_score, classificat
ion_report
import matplotlib.pyplot as plt

# Function to compute metrics for a model
def compute_metrics(model_path, X_test, y_test, use_rgb=False):
    model = load_model(model_path)  # Load the model
    X = X_test_rgb if use_rgb else X_test  # Select the appropriate test data
    y_pred_prob = model.predict(X)  # Predict probabilities
    y_pred_class = (y_pred_prob > 0.5).astype(int)  # Convert probabilities to binary cl
ass predictions

    # Precision-Recall data
    precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)
    f1 = f1_score(y_test, y_pred_class)
    accuracy = accuracy_score(y_test, y_pred_class)
    report = classification_report(y_test, y_pred_class, output_dict=True)

    return precision, recall, thresholds, f1, accuracy, report

# Store models and their input format
models = {
    "Baseline Model": {"path": "./models/baseline_model.keras", "use_rgb": False},
    "Fine-Tuned Model": {"path": "./models/finetuned_model.keras", "use_rgb": False},
    "ResNet Model": {"path": "./models/resnet_model.keras", "use_rgb": True},
    "VGG Model": {"path": "./models/vgg_model.keras", "use_rgb": True},
}

# Plot Precision-Recall Curves
plt.figure(figsize=(10, 7))
metrics_summary = {}
```

```
for model_name, details in models.items():
    precision, recall, _, f1, accuracy, report = compute_metrics(
        details["path"], X_test, y_test, use_rgb=details["use_rgb"]
    )
    metrics_summary[model_name] = {"F1 Score": f1, "Accuracy": accuracy, "Report": repor
t}

    # Plot the Precision-Recall Curve
    plt.plot(recall, precision, label=f"{model_name} (F1 = {f1:.2f}, Acc = {accuracy:.2f
})")

# Add plot aesthetics
plt.title("Precision-Recall Curve Comparison", fontsize=16)
plt.xlabel("Recall", fontsize=14)
plt.ylabel("Precision", fontsize=14)
plt.legend(loc="lower left", fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()

# Print metrics for comparison
for model_name, metrics in metrics_summary.items():
    print(f"==== {model_name} ====")
    print(f"F1 Score: {metrics['F1 Score']:.2f}")
    print(f"Accuracy: {metrics['Accuracy']:.2f}")
    print("Classification Report:")
    print(metrics["Report"])
    print()
```

```
3/3 [==============================] - 0s 20ms/step
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on
M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizer
s.legacy.Adam`.
```

```
3/3 [==============================] - 0s 46ms/step
3/3 [==============================] - 1s 96ms/step
3/3 [==============================] - 1s 232ms/step
```



```
==== Baseline Model ====
F1 Score: 0.75
Accuracy: 0.71
Classification Report:
{'0': {'precision': 0.78125, 'recall': 0.5813953488372093, 'f1-score': 0.6666666666666666
, 'support': 43.0}, '1': {'precision': 0.6727272727272727, 'recall': 0.8409090909090909,
'f1-score': 0.7474747474747475, 'support': 44.0}, 'accuracy': 0.7126436781609196, 'macro
avg': {'precision': 0.7269886363636364, 'recall': 0.7111522198731501, 'f1-score': 0.70707
07070707071, 'support': 87.0}, 'weighted avg': {'precision': 0.7263649425287356, 'recall'
: 0.7126436781609196, 'f1-score': 0.7075351213282248, 'support': 87.0}}

==== Fine-Tuned Model ====
F1 Score: 0.83
Accuracy: 0.84
Classification Report:
{'0': {'precision': 0.8085106382978723, 'recall': 0.8837209302325582, 'f1-score': 0.84444
44444444444, 'support': 43.0}, '1': {'precision': 0.875, 'recall': 0.7954545454545454, 'f
1-score': 0.8333333333333334, 'support': 44.0}, 'accuracy': 0.8390804597701149, 'macro av
g': {'precision': 0.8417553191489362, 'recall': 0.8395877378435518, 'f1-score': 0.8388888
888888889, 'support': 87.0}, 'weighted avg': {'precision': 0.8421374419173391, 'recall':
0.8390804597701149, 'f1-score': 0.8388250319284803, 'support': 87.0}}

==== ResNet Model ====
F1 Score: 0.84
Accuracy: 0.85
Classification Report:
{'0': {'precision': 0.8125, 'recall': 0.9069767441860465, 'f1-score': 0.8571428571428572,
'support': 43.0}, '1': {'precision': 0.8974358974358975, 'recall': 0.7954545454545454, 'f
1-score': 0.8433734939759037, 'support': 44.0}, 'accuracy': 0.8505747126436781, 'macro av
g': {'precision': 0.8549679487179487, 'recall': 0.8512156448202959, 'f1-score': 0.8502581
755593804, 'support': 87.0}, 'weighted avg': {'precision': 0.8554560860595344, 'recall':
0.8505747126436781, 'f1-score': 0.8501790412883059, 'support': 87.0}}

==== VGG Model ====
F1 Score: 0.83
Accuracy: 0.84
Classification Report:
{'0': {'precision': 0.8085106382978723, 'recall': 0.8837209302325582, 'f1-score': 0.84444
44444444444, 'support': 43.0}, '1': {'precision': 0.875, 'recall': 0.7954545454545454, 'f
1-score': 0.8333333333333334, 'support': 44.0}, 'accuracy': 0.8390804597701149, 'macro av
g': {'precision': 0.8417553191489362, 'recall': 0.8395877378435518, 'f1-score': 0.8388888
888888889, 'support': 87.0}, 'weighted avg': {'precision': 0.8421374419173391, 'recall':
0.8390804597701149, 'f1-score': 0.8388250319284803, 'support': 87.0}}
```