In [1]:
```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
```

```
2025-01-12 14:11:37.006607: I tensorflow/core/platform/cpu_feature_guard.cc:
210] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

In [4]:
```python
# Set the directory path
base_dir =  '/Users/apple/Desktop/PROJECT FILES/photos dataset'
# Subfolders are:
me_only_dir = os.path.join(base_dir, 'me only')
me_others_dir = os.path.join(base_dir, 'me+others')
others_only_dir = os.path.join(base_dir, 'other only')
```

In [5]:
```python
def count_valid_images(folder_path):
    valid_extensions = ('.jpg', '.jpeg', '.png', '.bmp','.heic') # Common im
    return len([f for f in os.listdir(folder_path) if
f.lower().endswith(valid_extensions)])

print(f"'Me only': {count_valid_images(me_only_dir)} images")
print(f"'Me+others': {count_valid_images(me_others_dir)} images")
print(f"'Others only': {count_valid_images(others_only_dir)} images")
```

```
'Me only': 60 images
'Me+others': 60 images
'Others only': 60 images
```

In [19]:
```python
def preprocess_image(img_path, target_size=(224, 224)):
    img = cv2.imread(img_path)
    old_size = img.shape[:2]  # Original size (height, width)
    ratio = float(target_size[0]) / max(old_size)
    new_size = tuple([int(x * ratio) for x in old_size])
    img = cv2.resize(img, (new_size[1], new_size[0]))  # Resize by ratio
    delta_w = target_size[1] - new_size[1]
    delta_h = target_size[0] - new_size[0]
    color = [0, 0, 0]  # Black padding
    img = cv2.copyMakeBorder(img, delta_h // 2, delta_h - delta_h // 2,
                             delta_w // 2, delta_w - delta_w // 2,
                             cv2.BORDER_CONSTANT, value=color)
    img = img / 255.0
    return img
```

In [20]:
```python
import tensorflow as tf
from tensorflow.keras import layers, models
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split  # Import train_test_sp

def load_images(folder_paths, labels, img_size=(224, 224)):
    """
```

```
    Load and preprocess all images from given folders.
    """
    images, image_labels = [], []
    for folder, label in zip(folder_paths, labels):
        for filename in os.listdir(folder):
            img_path = os.path.join(folder, filename)
            img = cv2.imread(img_path)
            if img is not None:
                img = cv2.resize(img, img_size)  # Resize to fixed size
                images.append(img)
                image_labels.append(label)
    return np.array(images), np.array(image_labels)
```

In [21]:

```
# Define paths and labels
folders = [me_only_dir, me_others_dir, others_only_dir]
labels = [1, 1, 0]  # 1 = "me", 0 = "not me"

# Load and preprocess the data
X, y = load_images(folders, labels)
X = X / 255.0  # Normalize pixel values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Define the model with a Resizing layer to handle variable input sizes
model = models.Sequential([
    layers.Input(shape=(None, None, 3)),
    layers.Resizing(224, 224),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),  # Prevent overfitting
    layers.Dense(1, activation='sigmoid')
])
```

In [22]:

```
datagen = ImageDataGenerator(
    rotation_range=30,
    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(X_train)

X = X / 255.0
```

In [23]:

```
base_model = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3),
                                               include_top=False,
```

```
                                               weights='imagenet')
base_model.trainable = False  # Freeze the base model

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

In [24]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accura
```

In [25]:
```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['acc
```

In [26]:
```
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=50,
    validation_data=(X_test, y_test),
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience
)
```

```
Epoch 1/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.5080 - loss: 0.7719 - val
_accuracy: 0.6000 - val_loss: 0.6472
Epoch 2/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.6014 - loss: 0.7298 - val_
accuracy: 0.6667 - val_loss: 0.5874
Epoch 3/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 964ms/step - accuracy: 0.6142 - loss: 0.7448 - v
al_accuracy: 0.7000 - val_loss: 0.5467
Epoch 4/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.6640 - loss: 0.5773 - val_
accuracy: 0.7333 - val_loss: 0.5131
Epoch 5/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 977ms/step - accuracy: 0.6800 - loss: 0.5992 - v
al_accuracy: 0.8333 - val_loss: 0.4830
Epoch 6/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.7015 - loss: 0.5934 - val_
accuracy: 0.9333 - val_loss: 0.4588
Epoch 7/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.7714 - loss: 0.4870 - val_
accuracy: 0.9333 - val_loss: 0.4428
Epoch 8/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 986ms/step - accuracy: 0.8041 - loss: 0.4766 - v
al_accuracy: 0.9333 - val_loss: 0.4259
Epoch 9/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 991ms/step - accuracy: 0.7490 - loss: 0.5347 - v
al_accuracy: 0.9333 - val_loss: 0.4087
Epoch 10/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.7883 - loss: 0.4146 - val_
accuracy: 0.9333 - val_loss: 0.3895
Epoch 11/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8307 - loss: 0.4238 - val_
accuracy: 0.9333 - val_loss: 0.3746
Epoch 12/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - accuracy: 0.8631 - loss: 0.3891 - val
_accuracy: 0.9333 - val_loss: 0.3611
Epoch 13/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8501 - loss: 0.3872 - val_
accuracy: 0.9333 - val_loss: 0.3482
Epoch 14/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8085 - loss: 0.3637 - val_
accuracy: 0.9000 - val_loss: 0.3376
Epoch 15/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step - accuracy: 0.9319 - loss: 0.2746 - val_
accuracy: 0.9333 - val_loss: 0.3310
Epoch 16/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8193 - loss: 0.4813 - val_
accuracy: 0.9333 - val_loss: 0.3226
Epoch 17/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.8754 - loss: 0.3345 - val_
accuracy: 0.9333 - val_loss: 0.3110
Epoch 18/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.8527 - loss: 0.3798 - val_
accuracy: 0.9333 - val_loss: 0.3016
Epoch 19/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8306 - loss: 0.3638 - val_
```

```
accuracy: 0.9333 - val_loss: 0.2937
Epoch 20/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.8640 - loss: 0.2863 - val_
accuracy: 0.9667 - val_loss: 0.2855
Epoch 21/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9138 - loss: 0.2869 - val_
accuracy: 0.9667 - val_loss: 0.2771
Epoch 22/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8920 - loss: 0.2740 - val_
accuracy: 0.9333 - val_loss: 0.2737
Epoch 23/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8795 - loss: 0.3251 - val_
accuracy: 0.9333 - val_loss: 0.2741
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9035 - loss: 0.2863 - val_
accuracy: 0.9333 - val_loss: 0.2688
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 995ms/step - accuracy: 0.9109 - loss: 0.2625 - v
al_accuracy: 0.9333 - val_loss: 0.2588
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9571 - loss: 0.1952 - val_
accuracy: 0.9667 - val_loss: 0.2494
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9117 - loss: 0.2426 - val_
accuracy: 0.9667 - val_loss: 0.2421
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9582 - loss: 0.1934 - val_
accuracy: 0.9667 - val_loss: 0.2365
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8593 - loss: 0.2694 - val_
accuracy: 0.9667 - val_loss: 0.2309
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9324 - loss: 0.2015 - val_
accuracy: 0.9667 - val_loss: 0.2271
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9235 - loss: 0.2039 - val_
accuracy: 0.9333 - val_loss: 0.2283
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9471 - loss: 0.2357 - val_
accuracy: 0.9333 - val_loss: 0.2295
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9009 - loss: 0.2773 - val_
accuracy: 0.9333 - val_loss: 0.2266
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8844 - loss: 0.2699 - val_
accuracy: 0.9667 - val_loss: 0.2150
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9003 - loss: 0.2480 - val_
accuracy: 0.9667 - val_loss: 0.2099
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9300 - loss: 0.1842 - val_
accuracy: 0.9667 - val_loss: 0.2062
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9369 - loss: 0.1605 - val_
accuracy: 0.9667 - val_loss: 0.2026
Epoch 38/50
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9586 - loss: 0.1825 - val_
accuracy: 0.9667 - val_loss: 0.1989
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9108 - loss: 0.2013 - val_
accuracy: 0.9667 - val_loss: 0.1962
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9404 - loss: 0.1832 - val_
accuracy: 0.9667 - val_loss: 0.1932
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9521 - loss: 0.1680 - val_
accuracy: 0.9667 - val_loss: 0.1915
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8948 - loss: 0.2707 - val_
accuracy: 0.9667 - val_loss: 0.1901
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 987ms/step - accuracy: 0.8938 - loss: 0.2627 - v
al_accuracy: 0.9667 - val_loss: 0.1862
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9166 - loss: 0.2045 - val_
accuracy: 0.9667 - val_loss: 0.1844
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9615 - loss: 0.1787 - val_
accuracy: 0.9667 - val_loss: 0.1836
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9791 - loss: 0.1603 - val_
accuracy: 0.9667 - val_loss: 0.1815
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9272 - loss: 0.1989 - val_
accuracy: 0.9667 - val_loss: 0.1798
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9251 - loss: 0.1921 - val_
accuracy: 0.9667 - val_loss: 0.1765
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9341 - loss: 0.1875 - val_
accuracy: 0.9667 - val_loss: 0.1734
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9500 - loss: 0.1809 - val_
accuracy: 0.9667 - val_loss: 0.1715
```

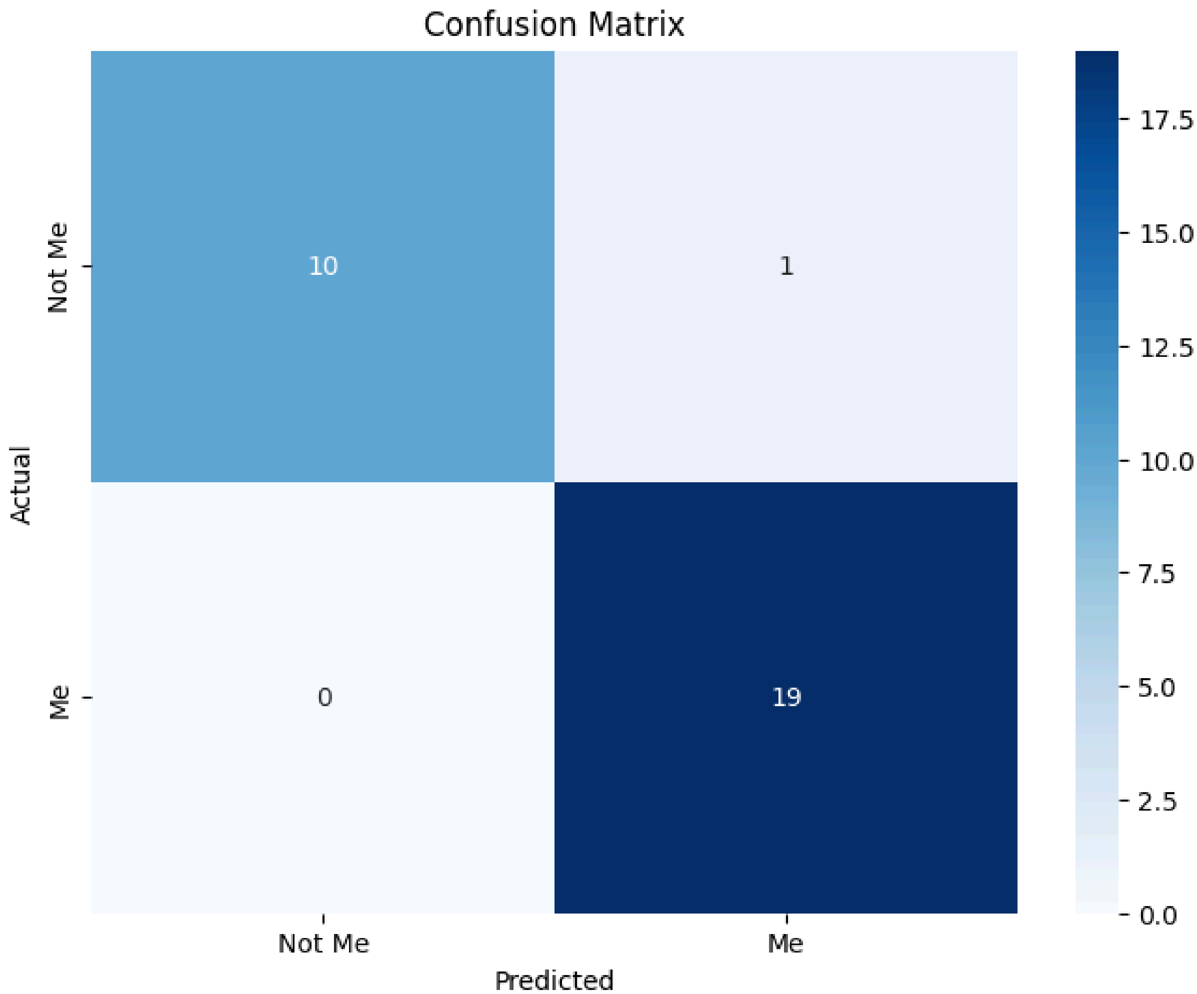
In [27]:

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

y_pred = (model.predict(X_test) > 0.5).astype("int32")
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Me', 'M
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```




```
In [28]: test_loss, test_accuracy = model.evaluate(X_test, y_test)
         print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 728ms/step - accuracy: 0.9667 - loss: 0.1715
Test Accuracy: 96.67%
```