In [1]:
```
import scipy.io
import seaborn as sns
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.metrics import confusion_matrix
```

```
WARNING:tensorflow:From C:\Users\vinut\anaconda3\lib\site-packages\keras\src\losses.py:2976: The name tf.lo
sses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entrop
y instead.
```

In [2]:
```
df = pd.read_csv('C:/FAULT_DIAG_PROJ/CWRU_dataset/48k_drive_end/0hp/0hp_all_faults.csv')
```

In [3]:
```
# Data preprocessing
win_len = 784
stride = 300
X = []
Y = []
```

In [4]:
```
for k in df['fault'].unique():
    df_temp_2 = df[df['fault'] == k]

    for i in np.arange(0, len(df_temp_2) - (win_len), stride):
        temp = df_temp_2.iloc[i:i + win_len, :-1].values
        temp = temp.reshape((1, -1))
        X.append(temp)
        Y.append(df_temp_2.iloc[i + win_len, -1])
```

In [5]:
```
X = np.array(X)
X = X.reshape((X.shape[0], 28, 28, 1))
Y = np.array(Y)
```

In [6]:
```
# One-hot encode the target variable
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
OHE_Y = to_categorical(encoded_Y)
```

In [7]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, OHE_Y, test_size=0.3, shuffle=True)
```

In [8]:
```
# Create the CNN model
cnn_model = Sequential()
cnn_model.add(Conv2D(32, kernel_size=(3, 3), activation='tanh', input_shape=(X.shape[1], X.shape[2], 1), pac
cnn_model.add(MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
cnn_model.add(Conv2D(64, (3, 3), activation='tanh', padding='same'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
cnn_model.add(Flatten())
cnn_model.add(Dense(128, activation='tanh'))
cnn_model.add(Dense(len(df['fault'].unique()), activation='softmax'))
```

```
WARNING:tensorflow:From C:\Users\vinut\anaconda3\lib\site-packages\keras\src\backend.py:873: The name tf.ge
t_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\vinut\anaconda3\lib\site-packages\keras\src\layers\pooling\max_pooling2d.p
y:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.
```

In [9]:
```
# Change Activation Functions
cnn_model = Sequential()
cnn_model.add(Conv2D(32, kernel_size=(3, 3), activation='tanh', input_shape=(X.shape[1], X.shape[2], 1), pac
cnn_model.add(MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
cnn_model.add(Conv2D(64, (3, 3), activation='tanh', padding='same'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
cnn_model.add(Flatten())
cnn_model.add(Dense(128, activation='tanh'))
cnn_model.add(Dense(len(df['fault'].unique()), activation='softmax'))
```

In [10]:
```
# Compile the model
cnn_model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\vinut\anaconda3\lib\site-packages\keras\src\optimizers\__init__.py:309: Th
e name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

In [11]:
```
# Set the number of epochs to 50
epochs = 50
```

In [12]:
```
# Train the CNN model
history = cnn_model.fit(X_train, y_train, batch_size=128, epochs=epochs, verbose=1, validation_data=(X_test,
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\vinut\anaconda3\lib\site-packages\keras\src\utils\tf_utils.py:492: The nam
e tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\vinut\anaconda3\lib\site-packages\keras\src\engine\base_layer_utils.py:38
4: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly
_outside_functions instead.

42/42 [==============================] - 3s 46ms/step - loss: 2.1245 - accuracy: 0.1985 - val_loss: 1.9160
- val_accuracy: 0.2978
Epoch 2/50
42/42 [==============================] - 2s 37ms/step - loss: 1.7203 - accuracy: 0.4630 - val_loss: 1.5256
- val_accuracy: 0.6293
Epoch 3/50
42/42 [==============================] - 2s 38ms/step - loss: 1.3393 - accuracy: 0.6325 - val_loss: 1.1880
- val_accuracy: 0.6611
Epoch 4/50
42/42 [==============================] - 1s 35ms/step - loss: 1.0731 - accuracy: 0.6880 - val_loss: 0.9981
- val_accuracy: 0.6836
Epoch 5/50
42/42 [==============================] - 2s 39ms/step - loss: 0.9221 - accuracy: 0.7255 - val_loss: 0.8812
- val_accuracy: 0.7919
Epoch 6/50
42/42 [==============================] - 1s 35ms/step - loss: 0.8225 - accuracy: 0.7945 - val_loss: 0.7945
- val_accuracy: 0.7477
Epoch 7/50
42/42 [==============================] - 2s 38ms/step - loss: 0.7459 - accuracy: 0.8163 - val_loss: 0.7271
- val_accuracy: 0.7901
Epoch 8/50
42/42 [==============================] - 2s 36ms/step - loss: 0.6838 - accuracy: 0.8316 - val_loss: 0.6706
- val_accuracy: 0.8489
Epoch 9/50
42/42 [==============================] - 2s 36ms/step - loss: 0.6312 - accuracy: 0.8634 - val_loss: 0.6247
- val_accuracy: 0.8246
Epoch 10/50
42/42 [==============================] - 1s 35ms/step - loss: 0.5850 - accuracy: 0.8754 - val_loss: 0.5787
- val_accuracy: 0.8524
Epoch 11/50
42/42 [==============================] - 1s 33ms/step - loss: 0.5436 - accuracy: 0.8812 - val_loss: 0.5396
- val_accuracy: 0.8767
Epoch 12/50
42/42 [==============================] - 1s 35ms/step - loss: 0.5054 - accuracy: 0.8951 - val_loss: 0.5057
- val_accuracy: 0.8719
Epoch 13/50
42/42 [==============================] - 1s 36ms/step - loss: 0.4710 - accuracy: 0.8998 - val_loss: 0.4733
- val_accuracy: 0.8922
Epoch 14/50
42/42 [==============================] - 2s 36ms/step - loss: 0.4385 - accuracy: 0.9081 - val_loss: 0.4523
- val_accuracy: 0.8683
Epoch 15/50
42/42 [==============================] - 2s 41ms/step - loss: 0.4111 - accuracy: 0.9095 - val_loss: 0.4184
- val_accuracy: 0.8962
Epoch 16/50
42/42 [==============================] - 2s 44ms/step - loss: 0.3860 - accuracy: 0.9127 - val_loss: 0.3952
- val_accuracy: 0.8975
Epoch 17/50
42/42 [==============================] - 2s 43ms/step - loss: 0.3638 - accuracy: 0.9151 - val_loss: 0.3781
- val_accuracy: 0.8904
Epoch 18/50
42/42 [==============================] - 2s 42ms/step - loss: 0.3437 - accuracy: 0.9140 - val_loss: 0.3600
- val_accuracy: 0.9010
Epoch 19/50
42/42 [==============================] - 2s 39ms/step - loss: 0.3271 - accuracy: 0.9184 - val_loss: 0.3440
- val_accuracy: 0.9103
Epoch 20/50
42/42 [==============================] - 2s 39ms/step - loss: 0.3109 - accuracy: 0.9216 - val_loss: 0.3317
- val_accuracy: 0.9152
Epoch 21/50
42/42 [==============================] - 2s 39ms/step - loss: 0.2976 - accuracy: 0.9261 - val_loss: 0.3169
- val_accuracy: 0.9134
Epoch 22/50
42/42 [==============================] - 2s 39ms/step - loss: 0.2849 - accuracy: 0.9242 - val_loss: 0.3151
- val_accuracy: 0.9156
Epoch 23/50
42/42 [==============================] - 2s 39ms/step - loss: 0.2749 - accuracy: 0.9297 - val_loss: 0.3021
- val_accuracy: 0.9134
Epoch 24/50
42/42 [==============================] - 2s 41ms/step - loss: 0.2650 - accuracy: 0.9295 - val_loss: 0.2912
- val_accuracy: 0.9191
Epoch 25/50
```

```
42/42 [==============================] - 2s 45ms/step - loss: 0.2560 - accuracy: 0.9316 - val_loss: 0.2802
- val_accuracy: 0.9174
Epoch 26/50
42/42 [==============================] - 2s 46ms/step - loss: 0.2475 - accuracy: 0.9352 - val_loss: 0.2753
- val_accuracy: 0.9129
Epoch 27/50
42/42 [==============================] - 2s 41ms/step - loss: 0.2407 - accuracy: 0.9329 - val_loss: 0.2656
- val_accuracy: 0.9218
Epoch 28/50
42/42 [==============================] - 2s 39ms/step - loss: 0.2342 - accuracy: 0.9356 - val_loss: 0.2630
- val_accuracy: 0.9152
Epoch 29/50
42/42 [==============================] - 2s 49ms/step - loss: 0.2276 - accuracy: 0.9375 - val_loss: 0.2540
- val_accuracy: 0.9218
Epoch 30/50
42/42 [==============================] - 2s 53ms/step - loss: 0.2218 - accuracy: 0.9394 - val_loss: 0.2494
- val_accuracy: 0.9249
Epoch 31/50
42/42 [==============================] - 2s 43ms/step - loss: 0.2166 - accuracy: 0.9403 - val_loss: 0.2484
- val_accuracy: 0.9160
Epoch 32/50
42/42 [==============================] - 2s 44ms/step - loss: 0.2112 - accuracy: 0.9398 - val_loss: 0.2433
- val_accuracy: 0.9236
Epoch 33/50
42/42 [==============================] - 2s 47ms/step - loss: 0.2064 - accuracy: 0.9418 - val_loss: 0.2351
- val_accuracy: 0.9266
Epoch 34/50
42/42 [==============================] - 2s 43ms/step - loss: 0.2016 - accuracy: 0.9417 - val_loss: 0.2364
- val_accuracy: 0.9284
Epoch 35/50
42/42 [==============================] - 2s 42ms/step - loss: 0.1980 - accuracy: 0.9435 - val_loss: 0.2459
- val_accuracy: 0.9236
Epoch 36/50
42/42 [==============================] - 2s 41ms/step - loss: 0.1939 - accuracy: 0.9437 - val_loss: 0.2240
- val_accuracy: 0.9275
Epoch 37/50
42/42 [==============================] - 2s 38ms/step - loss: 0.1889 - accuracy: 0.9449 - val_loss: 0.2236
- val_accuracy: 0.9319
Epoch 38/50
42/42 [==============================] - 2s 38ms/step - loss: 0.1852 - accuracy: 0.9471 - val_loss: 0.2190
- val_accuracy: 0.9346
Epoch 39/50
42/42 [==============================] - 2s 37ms/step - loss: 0.1823 - accuracy: 0.9460 - val_loss: 0.2204
- val_accuracy: 0.9289
Epoch 40/50
42/42 [==============================] - 2s 37ms/step - loss: 0.1789 - accuracy: 0.9479 - val_loss: 0.2098
- val_accuracy: 0.9293
Epoch 41/50
42/42 [==============================] - 2s 37ms/step - loss: 0.1749 - accuracy: 0.9494 - val_loss: 0.2078
- val_accuracy: 0.9306
Epoch 42/50
42/42 [==============================] - 2s 38ms/step - loss: 0.1721 - accuracy: 0.9506 - val_loss: 0.2036
- val_accuracy: 0.9337
Epoch 43/50
42/42 [==============================] - 2s 38ms/step - loss: 0.1682 - accuracy: 0.9506 - val_loss: 0.2088
- val_accuracy: 0.9315
Epoch 44/50
42/42 [==============================] - 2s 39ms/step - loss: 0.1662 - accuracy: 0.9515 - val_loss: 0.1985
- val_accuracy: 0.9377
Epoch 45/50
42/42 [==============================] - 2s 38ms/step - loss: 0.1627 - accuracy: 0.9513 - val_loss: 0.2007
- val_accuracy: 0.9373
Epoch 46/50
42/42 [==============================] - 2s 39ms/step - loss: 0.1594 - accuracy: 0.9528 - val_loss: 0.2091
- val_accuracy: 0.9284
Epoch 47/50
42/42 [==============================] - 2s 38ms/step - loss: 0.1574 - accuracy: 0.9528 - val_loss: 0.1872
- val_accuracy: 0.9412
Epoch 48/50
42/42 [==============================] - 2s 39ms/step - loss: 0.1538 - accuracy: 0.9540 - val_loss: 0.2044
- val_accuracy: 0.9373
Epoch 49/50
42/42 [==============================] - 2s 38ms/step - loss: 0.1519 - accuracy: 0.9545 - val_loss: 0.1958
- val_accuracy: 0.9395
Epoch 50/50
42/42 [==============================] - 2s 39ms/step - loss: 0.1487 - accuracy: 0.9536 - val_loss: 0.1839
- val_accuracy: 0.9377
```

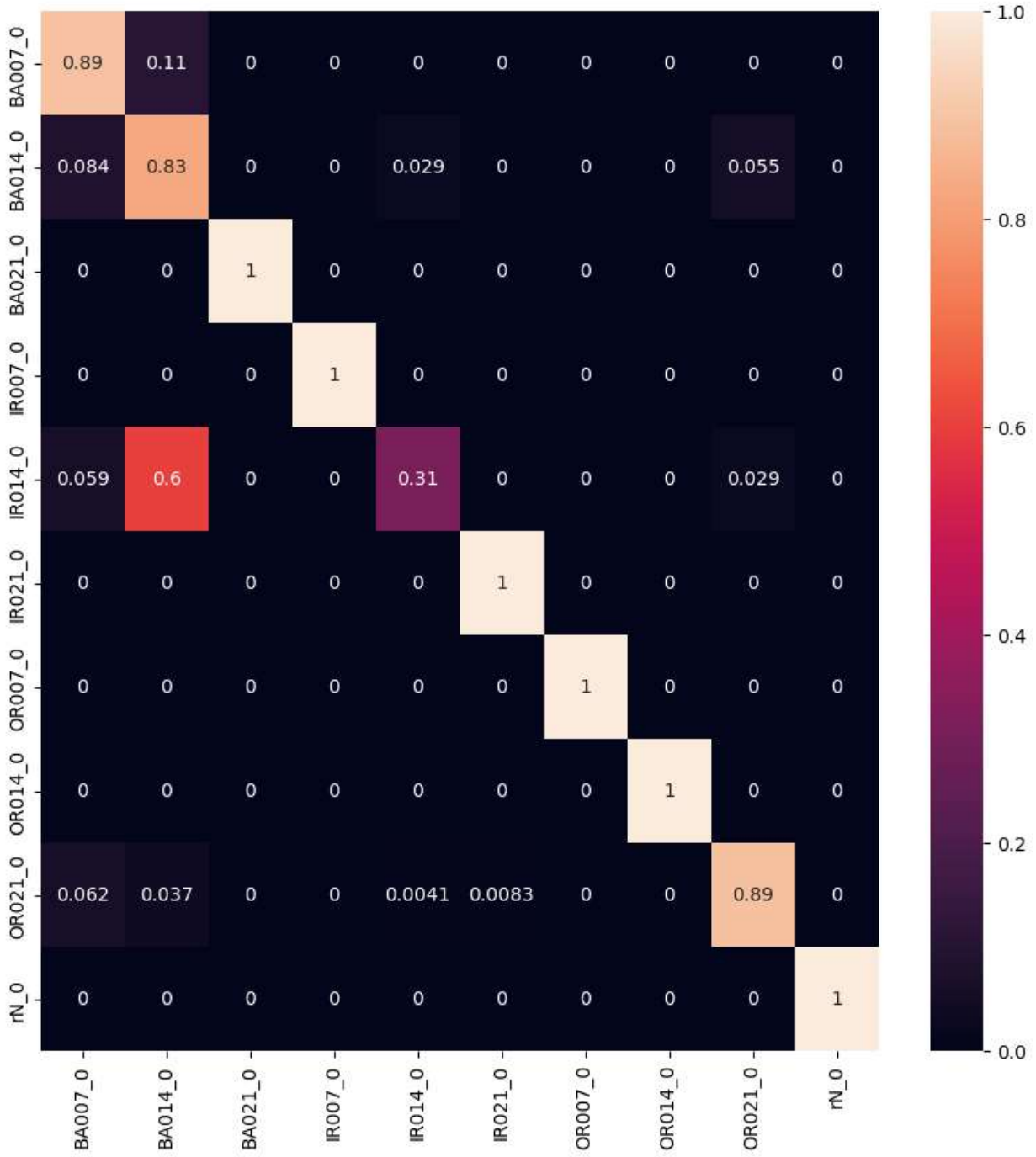
In [13]:
```python
# Function to inverse transform predictions
def inv_Transform_result(y_pred):
```

```
    y_pred = y_pred.argmax(axis=1)
    y_pred = encoder.inverse_transform(y_pred)
    return y_pred
```

In [14]:
```
# Predictions on the test set
y_pred = cnn_model.predict(X_test)
Y_pred = inv_Transform_result(y_pred)
Y_test = inv_Transform_result(y_test)
```

```
71/71 [==============================] - 1s 4ms/step
```

In [15]:
```
# Confusion Matrix
plt.figure(figsize=(10, 10))
cm = confusion_matrix(Y_test, Y_pred, normalize='true')
f = sns.heatmap(cm, annot=True, xticklabels=encoder.classes_, yticklabels=encoder.classes_)
plt.show()
```



In [16]:
```
# Classification Report
print("Classification Report:")
print(classification_report(Y_test, Y_pred, target_names=encoder.classes_))
```

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

     BA007_0       0.85      0.89      0.87       254
     BA014_0       0.72      0.83      0.77       238
     BA021_0       1.00      1.00      1.00       248
     IR007_0       1.00      1.00      1.00       249
     IR014_0       0.72      0.31      0.43        68
     IR021_0       0.99      1.00      1.00       243
     OR007_0       1.00      1.00      1.00       248
     OR014_0       1.00      1.00      1.00       241
     OR021_0       0.93      0.89      0.91       242
        rN_0       1.00      1.00      1.00       232

    accuracy                           0.94      2263
   macro avg       0.92      0.89      0.90      2263
weighted avg       0.94      0.94      0.93      2263
```

In [17]:
```python
# Additional Performance Metrics
accuracy = np.sum(np.diag(cm)) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1_score = 2 * (precision * recall) / (precision + recall)
```

In [18]:
```python
print("\nAdditional Performance Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print("Precision per class:")
for fault, prec in zip(encoder.classes_, precision):
    print(f"{fault}: {prec:.4f}")
print("Recall per class:")
for fault, rec in zip(encoder.classes_, recall):
    print(f"{fault}: {rec:.4f}")
print("F1 Score per class:")
for fault, f1 in zip(encoder.classes_, f1_score):
    print(f"{fault}: {f1:.4f}")
```

```
Additional Performance Metrics:
Accuracy: 0.8923
Precision per class:
BA007_0: 0.8135
BA014_0: 0.5271
BA021_0: 1.0000
IR007_0: 1.0000
IR014_0: 0.9020
IR021_0: 0.9918
OR007_0: 1.0000
OR014_0: 1.0000
OR021_0: 0.9136
rN_0: 1.0000
Recall per class:
BA007_0: 0.8937
BA014_0: 0.8319
BA021_0: 1.0000
IR007_0: 1.0000
IR014_0: 0.3088
IR021_0: 1.0000
OR007_0: 1.0000
OR014_0: 1.0000
OR021_0: 0.8884
rN_0: 1.0000
F1 Score per class:
BA007_0: 0.8517
BA014_0: 0.6453
BA021_0: 1.0000
IR007_0: 1.0000
IR014_0: 0.4601
IR021_0: 0.9959
OR007_0: 1.0000
OR014_0: 1.0000
OR021_0: 0.9008
rN_0: 1.0000
```

In [19]:
```python
# Visualize Results
num_samples_to_visualize = 5
```

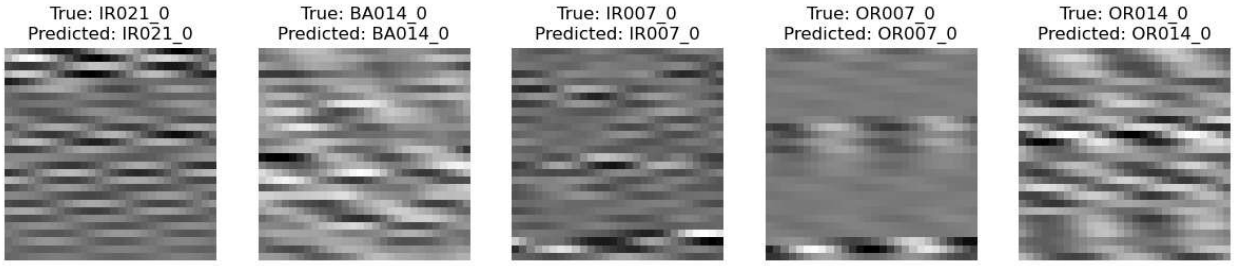
In [20]:
```python
# Randomly select some samples from the test set
random_indices = np.random.choice(len(X_test), num_samples_to_visualize, replace=False)
sample_images = X_test[random_indices]
true_labels = Y_test[random_indices]
```

In [21]:
```
# Predict the labels for the selected samples
predicted_labels = inv_Transform_result(cnn_model.predict(sample_images))
```

```
1/1 [==============================] - 0s 26ms/step
```

In [22]:
```
# Plot the selected samples along with true and predicted labels
plt.figure(figsize=(15, 8))
for i in range(num_samples_to_visualize):
    plt.subplot(1, num_samples_to_visualize, i + 1)
    plt.imshow(sample_images[i, :, :, 0], cmap='gray')
    plt.title(f'True: {true_labels[i]}\nPredicted: {predicted_labels[i]}')
    plt.axis('off')

plt.show()
```



In [ ]: