In [3]:

```
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 [6]:

```
df = pd.read_csv('C:/FAULT_DIAG_PROJ/CWRU_dataset/48k_drive_end/3hp/3hp_all_faults.csv')
```

In [7]:

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

In [8]:

```
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 [9]:

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

In [10]:

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

In [11]:

```
# 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 [12]:

```
# 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 [14]:

```
# 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 [15]:

```
# Set the number of epochs to 50
epochs = 50
```

In [16]:

```
# 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.

89/89 [==============================] - 4s 38ms/step - loss: 2.0922 - accuracy: 0.2391 - val_loss: 1.8773
- val_accuracy: 0.3643
Epoch 2/50
89/89 [==============================] - 3s 33ms/step - loss: 1.7401 - accuracy: 0.4010 - val_loss: 1.6264
- val_accuracy: 0.4355
Epoch 3/50
89/89 [==============================] - 3s 32ms/step - loss: 1.5429 - accuracy: 0.4652 - val_loss: 1.4667
- val_accuracy: 0.4632
Epoch 4/50
89/89 [==============================] - 3s 32ms/step - loss: 1.3919 - accuracy: 0.5222 - val_loss: 1.3349
- val_accuracy: 0.5435
Epoch 5/50
89/89 [==============================] - 3s 34ms/step - loss: 1.2697 - accuracy: 0.5721 - val_loss: 1.2281
- val_accuracy: 0.5897
Epoch 6/50
89/89 [==============================] - 3s 34ms/step - loss: 1.1674 - accuracy: 0.6102 - val_loss: 1.1389
- val_accuracy: 0.6086
Epoch 7/50
89/89 [==============================] - 3s 39ms/step - loss: 1.0811 - accuracy: 0.6437 - val_loss: 1.0641
- val_accuracy: 0.6234
Epoch 8/50
89/89 [==============================] - 3s 35ms/step - loss: 1.0066 - accuracy: 0.6653 - val_loss: 0.9947
- val_accuracy: 0.6626
Epoch 9/50
89/89 [==============================] - 4s 41ms/step - loss: 0.9402 - accuracy: 0.6950 - val_loss: 0.9297
- val_accuracy: 0.6914
Epoch 10/50
89/89 [==============================] - 3s 37ms/step - loss: 0.8798 - accuracy: 0.7157 - val_loss: 0.8755
- val_accuracy: 0.7092
Epoch 11/50
89/89 [==============================] - 3s 38ms/step - loss: 0.8247 - accuracy: 0.7338 - val_loss: 0.8262
- val_accuracy: 0.7191
Epoch 12/50
89/89 [==============================] - 3s 38ms/step - loss: 0.7748 - accuracy: 0.7532 - val_loss: 0.7789
- val_accuracy: 0.7333
Epoch 13/50
89/89 [==============================] - 3s 38ms/step - loss: 0.7292 - accuracy: 0.7657 - val_loss: 0.7313
- val_accuracy: 0.7595
Epoch 14/50
89/89 [==============================] - 3s 38ms/step - loss: 0.6870 - accuracy: 0.7814 - val_loss: 0.6926
- val_accuracy: 0.7700
Epoch 15/50
89/89 [==============================] - 3s 38ms/step - loss: 0.6484 - accuracy: 0.7949 - val_loss: 0.6586
- val_accuracy: 0.7817
Epoch 16/50
89/89 [==============================] - 3s 39ms/step - loss: 0.6160 - accuracy: 0.8071 - val_loss: 0.6315
- val_accuracy: 0.7821
Epoch 17/50
89/89 [==============================] - 3s 39ms/step - loss: 0.5840 - accuracy: 0.8181 - val_loss: 0.5961
- val_accuracy: 0.8000
Epoch 18/50
89/89 [==============================] - 4s 40ms/step - loss: 0.5568 - accuracy: 0.8242 - val_loss: 0.5642
- val_accuracy: 0.8170
Epoch 19/50
89/89 [==============================] - 3s 39ms/step - loss: 0.5301 - accuracy: 0.8311 - val_loss: 0.5363
- val_accuracy: 0.8253
Epoch 20/50
89/89 [==============================] - 4s 40ms/step - loss: 0.5042 - accuracy: 0.8412 - val_loss: 0.5268
- val_accuracy: 0.8181
Epoch 21/50
89/89 [==============================] - 3s 38ms/step - loss: 0.4804 - accuracy: 0.8483 - val_loss: 0.5058
- val_accuracy: 0.8265
Epoch 22/50
89/89 [==============================] - 3s 39ms/step - loss: 0.4612 - accuracy: 0.8529 - val_loss: 0.4713
- val_accuracy: 0.8478
Epoch 23/50
89/89 [==============================] - 4s 40ms/step - loss: 0.4423 - accuracy: 0.8571 - val_loss: 0.4530
- val_accuracy: 0.8546
Epoch 24/50
89/89 [==============================] - 4s 41ms/step - loss: 0.4244 - accuracy: 0.8611 - val_loss: 0.4577
- val_accuracy: 0.8407
Epoch 25/50
```

```
89/89 [==============================] - 4s 40ms/step - loss: 0.4173 - accuracy: 0.8612 - val_loss: 0.4452
- val_accuracy: 0.8441
Epoch 26/50
89/89 [==============================] - 4s 39ms/step - loss: 0.4019 - accuracy: 0.8654 - val_loss: 0.4382
- val_accuracy: 0.8437
Epoch 27/50
89/89 [==============================] - 3s 39ms/step - loss: 0.3844 - accuracy: 0.8728 - val_loss: 0.4184
- val_accuracy: 0.8630
Epoch 28/50
89/89 [==============================] - 3s 39ms/step - loss: 0.3878 - accuracy: 0.8649 - val_loss: 0.3899
- val_accuracy: 0.8678
Epoch 29/50
89/89 [==============================] - 3s 38ms/step - loss: 0.3701 - accuracy: 0.8734 - val_loss: 0.4052
- val_accuracy: 0.8595
Epoch 30/50
89/89 [==============================] - 3s 38ms/step - loss: 0.3615 - accuracy: 0.8784 - val_loss: 0.3632
- val_accuracy: 0.8809
Epoch 31/50
89/89 [==============================] - 4s 40ms/step - loss: 0.3576 - accuracy: 0.8793 - val_loss: 0.3837
- val_accuracy: 0.8665
Epoch 32/50
89/89 [==============================] - 3s 38ms/step - loss: 0.3400 - accuracy: 0.8850 - val_loss: 0.3546
- val_accuracy: 0.8821
Epoch 33/50
89/89 [==============================] - 4s 41ms/step - loss: 0.3282 - accuracy: 0.8905 - val_loss: 0.3376
- val_accuracy: 0.8854
Epoch 34/50
89/89 [==============================] - 3s 39ms/step - loss: 0.3400 - accuracy: 0.8795 - val_loss: 0.3973
- val_accuracy: 0.8503
Epoch 35/50
89/89 [==============================] - 4s 40ms/step - loss: 0.3240 - accuracy: 0.8886 - val_loss: 0.3767
- val_accuracy: 0.8610
Epoch 36/50
89/89 [==============================] - 4s 40ms/step - loss: 0.3168 - accuracy: 0.8880 - val_loss: 0.3307
- val_accuracy: 0.8891
Epoch 37/50
89/89 [==============================] - 4s 40ms/step - loss: 0.3159 - accuracy: 0.8884 - val_loss: 0.3201
- val_accuracy: 0.8922
Epoch 38/50
89/89 [==============================] - 4s 40ms/step - loss: 0.2994 - accuracy: 0.8968 - val_loss: 0.3181
- val_accuracy: 0.8949
Epoch 39/50
89/89 [==============================] - 3s 39ms/step - loss: 0.3019 - accuracy: 0.8921 - val_loss: 0.2989
- val_accuracy: 0.8994
Epoch 40/50
89/89 [==============================] - 3s 39ms/step - loss: 0.2772 - accuracy: 0.9075 - val_loss: 0.3104
- val_accuracy: 0.8908
Epoch 41/50
89/89 [==============================] - 3s 39ms/step - loss: 0.2781 - accuracy: 0.9036 - val_loss: 0.2904
- val_accuracy: 0.9031
Epoch 42/50
89/89 [==============================] - 3s 37ms/step - loss: 0.2784 - accuracy: 0.9040 - val_loss: 0.2927
- val_accuracy: 0.9025
Epoch 43/50
89/89 [==============================] - 3s 38ms/step - loss: 0.2710 - accuracy: 0.9074 - val_loss: 0.2839
- val_accuracy: 0.9062
Epoch 44/50
89/89 [==============================] - 4s 39ms/step - loss: 0.2590 - accuracy: 0.9119 - val_loss: 0.3020
- val_accuracy: 0.8914
Epoch 45/50
89/89 [==============================] - 3s 39ms/step - loss: 0.2721 - accuracy: 0.9041 - val_loss: 0.3126
- val_accuracy: 0.8893
Epoch 46/50
89/89 [==============================] - 3s 38ms/step - loss: 0.2625 - accuracy: 0.9083 - val_loss: 0.4307
- val_accuracy: 0.8306
Epoch 47/50
89/89 [==============================] - 3s 38ms/step - loss: 0.2538 - accuracy: 0.9120 - val_loss: 0.2689
- val_accuracy: 0.9094
Epoch 48/50
89/89 [==============================] - 4s 41ms/step - loss: 0.2481 - accuracy: 0.9133 - val_loss: 0.2663
- val_accuracy: 0.9064
Epoch 49/50
89/89 [==============================] - 4s 41ms/step - loss: 0.2459 - accuracy: 0.9144 - val_loss: 0.3107
- val_accuracy: 0.8848
Epoch 50/50
89/89 [==============================] - 3s 37ms/step - loss: 0.2320 - accuracy: 0.9223 - val_loss: 0.2555
- val_accuracy: 0.9117
```

In [22]:

```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
```

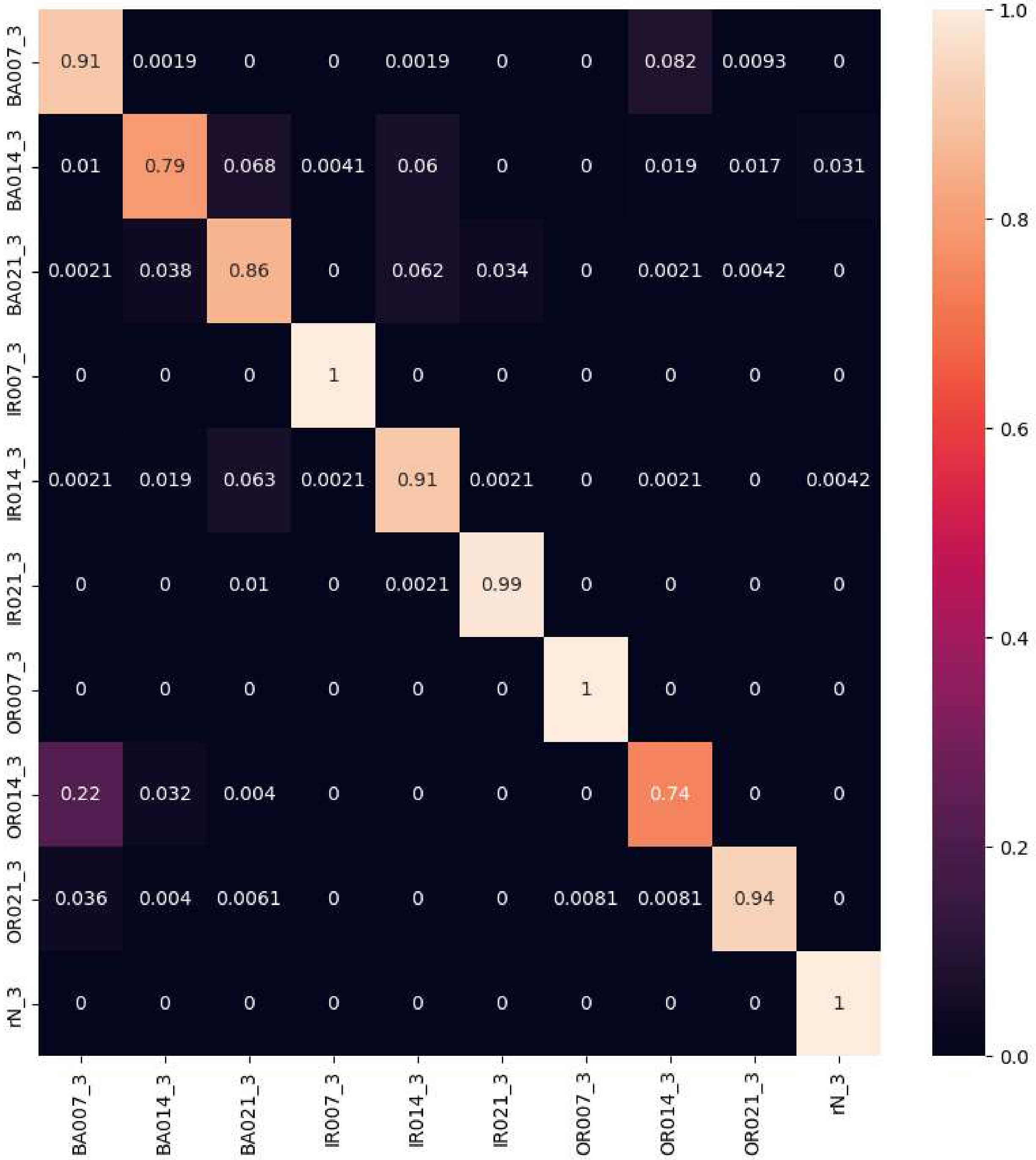
In [23]:

```
# 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)
```

```
153/153 [==============================] - 1s 4ms/step
```

In [24]:

```
# 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 [25]:

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

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

     BA007_3       0.78      0.91      0.84       538
     BA014_3       0.89      0.79      0.84       483
     BA021_3       0.85      0.86      0.85       471
     IR007_3       0.99      1.00      1.00       444
     IR014_3       0.88      0.91      0.89       476
     IR021_3       0.97      0.99      0.98       479
     OR007_3       0.99      1.00      1.00       502
     OR014_3       0.86      0.74      0.80       501
     OR021_3       0.97      0.94      0.95       494
        rN_3       0.97      1.00      0.98       482

    accuracy                           0.91      4870
   macro avg       0.91      0.91      0.91      4870
weighted avg       0.91      0.91      0.91      4870
```

In [26]:

```
# 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 [27]:

```
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: 0.9127
Precision per class:
BA007_3: 0.7686
BA014_3: 0.8928
BA021_3: 0.8496
IR007_3: 0.9938
IR014_3: 0.8782
IR021_3: 0.9648
OR007_3: 0.9920
OR014_3: 0.8682
OR021_3: 0.9689
rN_3: 0.9659
Recall per class:
BA007_3: 0.9052
BA014_3: 0.7909
BA021_3: 0.8577
IR007_3: 1.0000
IR014_3: 0.9055
IR021_3: 0.9875
OR007_3: 1.0000
OR014_3: 0.7425
OR021_3: 0.9372
rN_3: 1.0000
F1 Score per class:
BA007_3: 0.8313
BA014_3: 0.8388
BA021_3: 0.8537
IR007_3: 0.9969
IR014_3: 0.8916
IR021_3: 0.9760
OR007_3: 0.9960
OR014_3: 0.8004
OR021_3: 0.9528
rN_3: 0.9827
```

In [28]:

```
# Visualize Results
num_samples_to_visualize = 5
```

In [34]:

```
# 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 [30]:

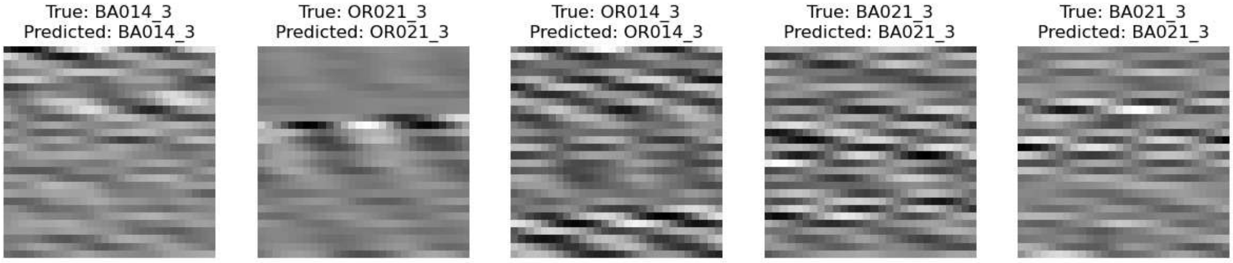
```
# Predict the labels for the selected samples
predicted_labels = inv_Transform_result(cnn_model.predict(sample_images))
```

```
1/1 [==============================] - 0s 23ms/step
```

In [33]:

```
# 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 [32]:

In [ ]: