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/1hp/1hp_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), pad
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]:
```
# 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), pad
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]:

```
# 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 [10]:

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

In [11]:

```
# 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.

80/80 [==============================] - 4s 42ms/step - loss: 1.9858 - accuracy: 0.2581 - val_loss: 1.7765
- val_accuracy: 0.4292
Epoch 2/50
80/80 [==============================] - 3s 37ms/step - loss: 1.6278 - accuracy: 0.3986 - val_loss: 1.5057
- val_accuracy: 0.4621
Epoch 3/50
80/80 [==============================] - 3s 35ms/step - loss: 1.4254 - accuracy: 0.4987 - val_loss: 1.3362
- val_accuracy: 0.5404
Epoch 4/50
80/80 [==============================] - 3s 37ms/step - loss: 1.2787 - accuracy: 0.5761 - val_loss: 1.2070
- val_accuracy: 0.6022
Epoch 5/50
80/80 [==============================] - 3s 35ms/step - loss: 1.1619 - accuracy: 0.6300 - val_loss: 1.1020
- val_accuracy: 0.6461
Epoch 6/50
80/80 [==============================] - 3s 36ms/step - loss: 1.0657 - accuracy: 0.6598 - val_loss: 1.0128
- val_accuracy: 0.6822
Epoch 7/50
80/80 [==============================] - 3s 37ms/step - loss: 0.9839 - accuracy: 0.6885 - val_loss: 0.9367
- val_accuracy: 0.7062
Epoch 8/50
80/80 [==============================] - 3s 35ms/step - loss: 0.9132 - accuracy: 0.7098 - val_loss: 0.8696
- val_accuracy: 0.7447
Epoch 9/50
80/80 [==============================] - 3s 35ms/step - loss: 0.8498 - accuracy: 0.7317 - val_loss: 0.8154
- val_accuracy: 0.7586
Epoch 10/50
80/80 [==============================] - 3s 39ms/step - loss: 0.7940 - accuracy: 0.7619 - val_loss: 0.7604
- val_accuracy: 0.7630
Epoch 11/50
80/80 [==============================] - 3s 42ms/step - loss: 0.7423 - accuracy: 0.7789 - val_loss: 0.7071
- val_accuracy: 0.7923
Epoch 12/50
80/80 [==============================] - 3s 40ms/step - loss: 0.6973 - accuracy: 0.7927 - val_loss: 0.6656
- val_accuracy: 0.8087
Epoch 13/50
80/80 [==============================] - 4s 44ms/step - loss: 0.6561 - accuracy: 0.8013 - val_loss: 0.6286
- val_accuracy: 0.8085
Epoch 14/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6216 - accuracy: 0.8097 - val_loss: 0.5937
- val_accuracy: 0.8209
Epoch 15/50
80/80 [==============================] - 3s 43ms/step - loss: 0.5896 - accuracy: 0.8214 - val_loss: 0.5703
- val_accuracy: 0.8298
Epoch 16/50
80/80 [==============================] - 3s 39ms/step - loss: 0.5612 - accuracy: 0.8236 - val_loss: 0.5405
- val_accuracy: 0.8367
Epoch 17/50
80/80 [==============================] - 3s 40ms/step - loss: 0.5399 - accuracy: 0.8282 - val_loss: 0.5164
- val_accuracy: 0.8318
Epoch 18/50
80/80 [==============================] - 3s 42ms/step - loss: 0.5157 - accuracy: 0.8355 - val_loss: 0.5041
- val_accuracy: 0.8554
Epoch 19/50
80/80 [==============================] - 3s 39ms/step - loss: 0.4992 - accuracy: 0.8419 - val_loss: 0.4797
- val_accuracy: 0.8488
Epoch 20/50
80/80 [==============================] - 3s 38ms/step - loss: 0.4812 - accuracy: 0.8453 - val_loss: 0.4711
- val_accuracy: 0.8499
Epoch 21/50
80/80 [==============================] - 3s 38ms/step - loss: 0.4669 - accuracy: 0.8513 - val_loss: 0.4552
- val_accuracy: 0.8476
Epoch 22/50
80/80 [==============================] - 3s 38ms/step - loss: 0.4540 - accuracy: 0.8515 - val_loss: 0.4536
- val_accuracy: 0.8440
Epoch 23/50
80/80 [==============================] - 3s 38ms/step - loss: 0.4452 - accuracy: 0.8518 - val_loss: 0.4246
- val_accuracy: 0.8593
Epoch 24/50
80/80 [==============================] - 3s 42ms/step - loss: 0.4257 - accuracy: 0.8627 - val_loss: 0.4137
- val_accuracy: 0.8710
Epoch 25/50
```

```
80/80 [==============================] - 4s 46ms/step - loss: 0.4178 - accuracy: 0.8642 - val_loss: 0.4053
- val_accuracy: 0.8726
Epoch 26/50
80/80 [==============================] - 3s 44ms/step - loss: 0.4054 - accuracy: 0.8692 - val_loss: 0.4420
- val_accuracy: 0.8543
Epoch 27/50
80/80 [==============================] - 3s 43ms/step - loss: 0.3998 - accuracy: 0.8708 - val_loss: 0.3833
- val_accuracy: 0.8746
Epoch 28/50
80/80 [==============================] - 3s 42ms/step - loss: 0.3834 - accuracy: 0.8781 - val_loss: 0.3734
- val_accuracy: 0.8815
Epoch 29/50
80/80 [==============================] - 3s 40ms/step - loss: 0.3769 - accuracy: 0.8771 - val_loss: 0.3801
- val_accuracy: 0.8655
Epoch 30/50
80/80 [==============================] - 3s 40ms/step - loss: 0.3668 - accuracy: 0.8814 - val_loss: 0.3759
- val_accuracy: 0.8714
Epoch 31/50
80/80 [==============================] - 3s 39ms/step - loss: 0.3611 - accuracy: 0.8853 - val_loss: 0.3604
- val_accuracy: 0.8744
Epoch 32/50
80/80 [==============================] - 3s 40ms/step - loss: 0.3441 - accuracy: 0.8902 - val_loss: 0.3357
- val_accuracy: 0.8929
Epoch 33/50
80/80 [==============================] - 3s 39ms/step - loss: 0.3380 - accuracy: 0.8897 - val_loss: 0.3244
- val_accuracy: 0.9032
Epoch 34/50
80/80 [==============================] - 3s 41ms/step - loss: 0.3257 - accuracy: 0.8969 - val_loss: 0.3200
- val_accuracy: 0.8950
Epoch 35/50
80/80 [==============================] - 4s 45ms/step - loss: 0.3179 - accuracy: 0.8984 - val_loss: 0.3131
- val_accuracy: 0.9035
Epoch 36/50
80/80 [==============================] - 3s 44ms/step - loss: 0.3074 - accuracy: 0.9029 - val_loss: 0.3474
- val_accuracy: 0.8721
Epoch 37/50
80/80 [==============================] - 3s 39ms/step - loss: 0.2993 - accuracy: 0.9084 - val_loss: 0.2887
- val_accuracy: 0.9110
Epoch 38/50
80/80 [==============================] - 3s 37ms/step - loss: 0.2866 - accuracy: 0.9124 - val_loss: 0.2951
- val_accuracy: 0.8977
Epoch 39/50
80/80 [==============================] - 3s 40ms/step - loss: 0.2784 - accuracy: 0.9136 - val_loss: 0.2705
- val_accuracy: 0.9188
Epoch 40/50
80/80 [==============================] - 4s 44ms/step - loss: 0.2684 - accuracy: 0.9158 - val_loss: 0.2756
- val_accuracy: 0.9128
Epoch 41/50
80/80 [==============================] - 3s 41ms/step - loss: 0.2602 - accuracy: 0.9208 - val_loss: 0.2560
- val_accuracy: 0.9190
Epoch 42/50
80/80 [==============================] - 3s 39ms/step - loss: 0.2493 - accuracy: 0.9242 - val_loss: 0.2472
- val_accuracy: 0.9314
Epoch 43/50
80/80 [==============================] - 3s 41ms/step - loss: 0.2420 - accuracy: 0.9288 - val_loss: 0.2430
- val_accuracy: 0.9254
Epoch 44/50
80/80 [==============================] - 3s 42ms/step - loss: 0.2347 - accuracy: 0.9310 - val_loss: 0.2378
- val_accuracy: 0.9291
Epoch 45/50
80/80 [==============================] - 3s 38ms/step - loss: 0.2248 - accuracy: 0.9327 - val_loss: 0.2315
- val_accuracy: 0.9266
Epoch 46/50
80/80 [==============================] - 3s 38ms/step - loss: 0.2170 - accuracy: 0.9341 - val_loss: 0.2168
- val_accuracy: 0.9330
Epoch 47/50
80/80 [==============================] - 3s 37ms/step - loss: 0.2096 - accuracy: 0.9369 - val_loss: 0.2090
- val_accuracy: 0.9391
Epoch 48/50
80/80 [==============================] - 3s 37ms/step - loss: 0.2050 - accuracy: 0.9391 - val_loss: 0.2114
- val_accuracy: 0.9316
Epoch 49/50
80/80 [==============================] - 3s 41ms/step - loss: 0.1991 - accuracy: 0.9398 - val_loss: 0.2258
- val_accuracy: 0.9197
Epoch 50/50
80/80 [==============================] - 3s 37ms/step - loss: 0.1919 - accuracy: 0.9422 - val_loss: 0.1909
- val_accuracy: 0.9456
```

In [12]:

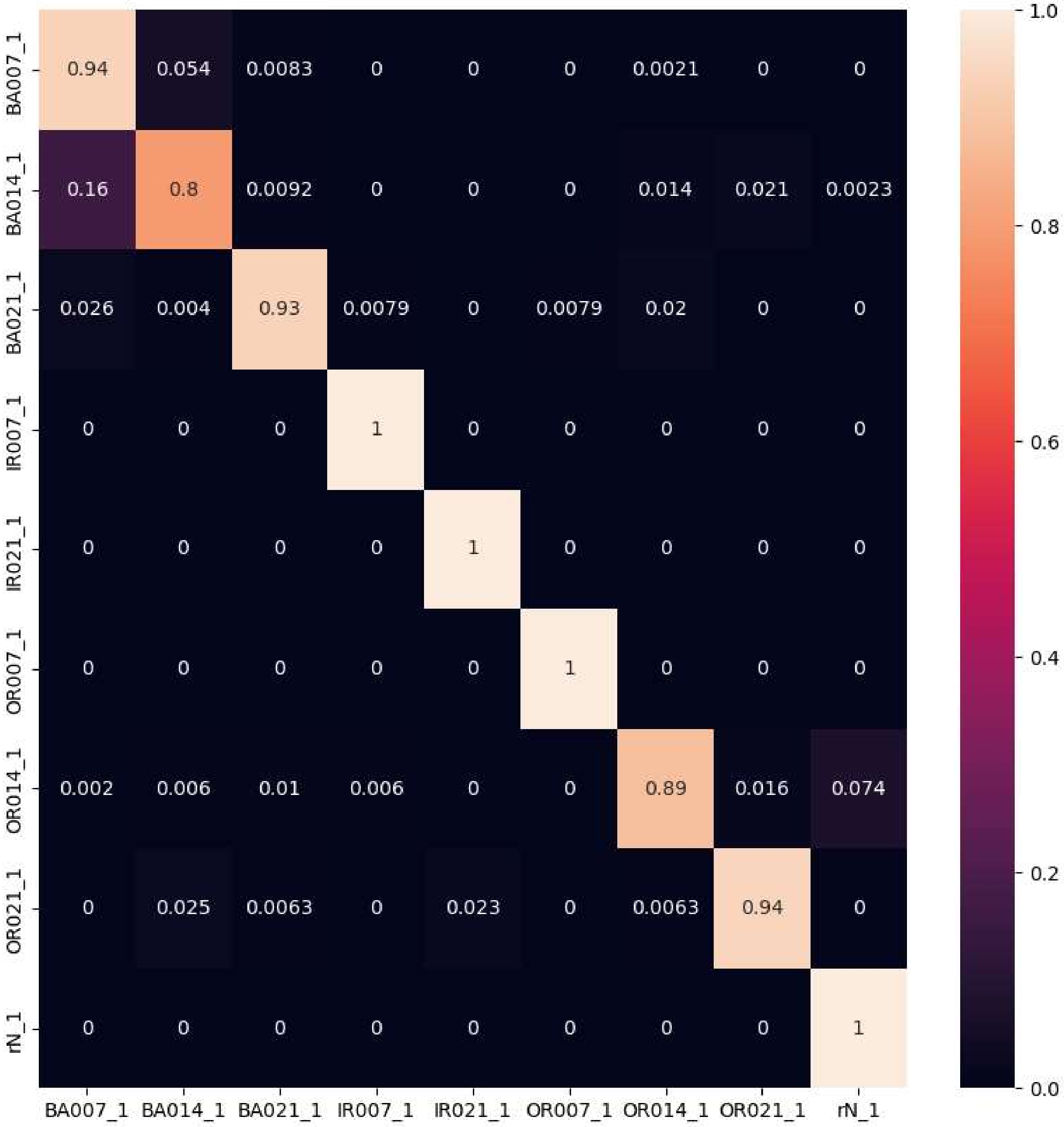
```
# Function to inverse transform predictions
def inv_Transform_result(y_pred):
```

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

In [13]:
```
# 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)
```

```
137/137 [==============================] - 1s 5ms/step
```

In [14]:
```
# 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 [15]:
```
# Classification Report
print("Classification Report:")
print(classification_report(Y_test, Y_pred, target_names=encoder.classes_))
```

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

     BA007_1       0.85      0.94      0.89       481
     BA014_1       0.89      0.80      0.84       433
     BA021_1       0.97      0.93      0.95       506
     IR007_1       0.99      1.00      0.99       487
     IR021_1       0.98      1.00      0.99       500
     OR007_1       0.99      1.00      1.00       477
     OR014_1       0.96      0.89      0.92       500
     OR021_1       0.96      0.94      0.95       479
        rN_1       0.93      1.00      0.96       508

    accuracy                           0.95      4371
   macro avg       0.95      0.94      0.94      4371
weighted avg       0.95      0.95      0.95      4371
```

In [16]:
```
# 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 [17]:
```
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: 0.9436
Precision per class:
BA007_1: 0.8351
BA014_1: 0.8995
BA021_1: 0.9651
IR007_1: 0.9863
IR021_1: 0.9776
OR007_1: 0.9922
OR014_1: 0.9548
OR021_1: 0.9623
rN_1: 0.9291
Recall per class:
BA007_1: 0.9356
BA014_1: 0.7968
BA021_1: 0.9348
IR007_1: 1.0000
IR021_1: 1.0000
OR007_1: 1.0000
OR014_1: 0.8860
OR021_1: 0.9395
rN_1: 1.0000
F1 Score per class:
BA007_1: 0.8825
BA014_1: 0.8450
BA021_1: 0.9497
IR007_1: 0.9931
IR021_1: 0.9886
OR007_1: 0.9961
OR014_1: 0.9191
OR021_1: 0.9508
rN_1: 0.9632
```

In [18]:
```
# Visualize Results
num_samples_to_visualize = 5
```

In [19]:
```
# 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 [20]:
```
# Predict the labels for the selected samples
predicted_labels = inv_Transform_result(cnn_model.predict(sample_images))
```

```
1/1 [==============================] - 0s 27ms/step
```

In [21]:

```
# 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 [ ]: