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/2hp/2hp_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 [25]:
```
X = []
Y = []
```

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 [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), 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(encoder.classes_), activation='softmax'))
```

In [13]:
```
# Compile the model
cnn_model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

In [14]:
```
# Train the CNN model
history = cnn_model.fit(X_train, y_train, batch_size=128, epochs=50, verbose=1, validation_data=(X_test, y_t
```

```
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 39ms/step - loss: 1.9709 - accuracy: 0.2599 - val_loss: 1.7497
- val_accuracy: 0.3543
Epoch 2/50
80/80 [==============================] - 3s 38ms/step - loss: 1.6266 - accuracy: 0.3552 - val_loss: 1.5148
- val_accuracy: 0.3854
Epoch 3/50
80/80 [==============================] - 3s 39ms/step - loss: 1.4569 - accuracy: 0.4352 - val_loss: 1.3818
- val_accuracy: 0.5102
Epoch 4/50
80/80 [==============================] - 3s 34ms/step - loss: 1.3436 - accuracy: 0.5178 - val_loss: 1.2803
- val_accuracy: 0.5446
Epoch 5/50
80/80 [==============================] - 3s 37ms/step - loss: 1.2539 - accuracy: 0.5525 - val_loss: 1.1965
- val_accuracy: 0.5697
Epoch 6/50
80/80 [==============================] - 3s 36ms/step - loss: 1.1779 - accuracy: 0.5838 - val_loss: 1.1253
- val_accuracy: 0.5743
Epoch 7/50
80/80 [==============================] - 3s 34ms/step - loss: 1.1101 - accuracy: 0.6045 - val_loss: 1.0560
- val_accuracy: 0.6167
Epoch 8/50
80/80 [==============================] - 3s 36ms/step - loss: 1.0472 - accuracy: 0.6268 - val_loss: 1.0038
- val_accuracy: 0.6302
Epoch 9/50
80/80 [==============================] - 3s 39ms/step - loss: 0.9908 - accuracy: 0.6429 - val_loss: 0.9441
- val_accuracy: 0.6678
Epoch 10/50
80/80 [==============================] - 3s 38ms/step - loss: 0.9407 - accuracy: 0.6563 - val_loss: 0.8958
- val_accuracy: 0.6655
Epoch 11/50
80/80 [==============================] - 3s 41ms/step - loss: 0.8967 - accuracy: 0.6706 - val_loss: 0.8602
- val_accuracy: 0.6774
Epoch 12/50
80/80 [==============================] - 3s 38ms/step - loss: 0.8585 - accuracy: 0.6820 - val_loss: 0.8191
- val_accuracy: 0.6918
Epoch 13/50
80/80 [==============================] - 3s 38ms/step - loss: 0.8206 - accuracy: 0.6931 - val_loss: 0.9089
- val_accuracy: 0.6010
Epoch 14/50
80/80 [==============================] - 3s 40ms/step - loss: 0.7922 - accuracy: 0.7037 - val_loss: 0.7582
- val_accuracy: 0.7198
Epoch 15/50
80/80 [==============================] - 3s 41ms/step - loss: 0.7709 - accuracy: 0.7117 - val_loss: 0.7644
- val_accuracy: 0.7048
Epoch 16/50
80/80 [==============================] - 3s 39ms/step - loss: 0.7614 - accuracy: 0.7151 - val_loss: 0.8392
- val_accuracy: 0.6457
Epoch 17/50
80/80 [==============================] - 3s 38ms/step - loss: 0.7319 - accuracy: 0.7327 - val_loss: 0.8019
- val_accuracy: 0.6642
Epoch 18/50
80/80 [==============================] - 3s 40ms/step - loss: 0.7124 - accuracy: 0.7438 - val_loss: 0.6711
- val_accuracy: 0.7598
Epoch 19/50
80/80 [==============================] - 3s 39ms/step - loss: 0.6915 - accuracy: 0.7470 - val_loss: 0.6609
- val_accuracy: 0.7609
Epoch 20/50
80/80 [==============================] - 3s 39ms/step - loss: 0.6660 - accuracy: 0.7608 - val_loss: 0.6388
- val_accuracy: 0.7718
Epoch 21/50
80/80 [==============================] - 3s 40ms/step - loss: 0.6489 - accuracy: 0.7711 - val_loss: 0.6247
- val_accuracy: 0.7721
Epoch 22/50
80/80 [==============================] - 3s 39ms/step - loss: 0.6280 - accuracy: 0.7804 - val_loss: 0.5985
- val_accuracy: 0.7901
Epoch 23/50
80/80 [==============================] - 3s 40ms/step - loss: 0.6038 - accuracy: 0.7836 - val_loss: 0.5938
- val_accuracy: 0.7915
Epoch 24/50
80/80 [==============================] - 3s 42ms/step - loss: 0.5817 - accuracy: 0.7986 - val_loss: 0.5831
- val_accuracy: 0.7906
Epoch 25/50
```

```
80/80 [==============================] - 3s 40ms/step - loss: 0.5650 - accuracy: 0.8024 - val_loss: 0.5425
- val_accuracy: 0.8136
Epoch 26/50
80/80 [==============================] - 3s 40ms/step - loss: 0.5432 - accuracy: 0.8134 - val_loss: 0.5398
- val_accuracy: 0.8072
Epoch 27/50
80/80 [==============================] - 3s 41ms/step - loss: 0.5281 - accuracy: 0.8213 - val_loss: 0.5129
- val_accuracy: 0.8268
Epoch 28/50
80/80 [==============================] - 3s 41ms/step - loss: 0.5171 - accuracy: 0.8247 - val_loss: 0.5159
- val_accuracy: 0.8245
Epoch 29/50
80/80 [==============================] - 3s 41ms/step - loss: 0.4859 - accuracy: 0.8396 - val_loss: 0.4836
- val_accuracy: 0.8346
Epoch 30/50
80/80 [==============================] - 3s 40ms/step - loss: 0.4707 - accuracy: 0.8423 - val_loss: 0.4677
- val_accuracy: 0.8419
Epoch 31/50
80/80 [==============================] - 3s 44ms/step - loss: 0.4535 - accuracy: 0.8505 - val_loss: 0.4519
- val_accuracy: 0.8465
Epoch 32/50
80/80 [==============================] - 3s 41ms/step - loss: 0.4390 - accuracy: 0.8576 - val_loss: 0.4689
- val_accuracy: 0.8389
Epoch 33/50
80/80 [==============================] - 3s 39ms/step - loss: 0.4222 - accuracy: 0.8629 - val_loss: 0.4169
- val_accuracy: 0.8665
Epoch 34/50
80/80 [==============================] - 3s 38ms/step - loss: 0.4020 - accuracy: 0.8733 - val_loss: 0.4056
- val_accuracy: 0.8684
Epoch 35/50
80/80 [==============================] - 3s 37ms/step - loss: 0.3895 - accuracy: 0.8758 - val_loss: 0.3894
- val_accuracy: 0.8789
Epoch 36/50
80/80 [==============================] - 3s 38ms/step - loss: 0.3784 - accuracy: 0.8776 - val_loss: 0.4269
- val_accuracy: 0.8271
Epoch 37/50
80/80 [==============================] - 3s 37ms/step - loss: 0.3623 - accuracy: 0.8839 - val_loss: 0.3729
- val_accuracy: 0.8779
Epoch 38/50
80/80 [==============================] - 3s 38ms/step - loss: 0.3502 - accuracy: 0.8886 - val_loss: 0.4011
- val_accuracy: 0.8480
Epoch 39/50
80/80 [==============================] - 3s 38ms/step - loss: 0.3349 - accuracy: 0.8948 - val_loss: 0.3375
- val_accuracy: 0.8992
Epoch 40/50
80/80 [==============================] - 3s 37ms/step - loss: 0.3473 - accuracy: 0.8813 - val_loss: 0.3464
- val_accuracy: 0.8903
Epoch 41/50
80/80 [==============================] - 3s 37ms/step - loss: 0.3301 - accuracy: 0.8908 - val_loss: 0.3502
- val_accuracy: 0.8722
Epoch 42/50
80/80 [==============================] - 3s 38ms/step - loss: 0.3156 - accuracy: 0.9001 - val_loss: 0.3217
- val_accuracy: 0.8921
Epoch 43/50
80/80 [==============================] - 3s 42ms/step - loss: 0.3434 - accuracy: 0.8733 - val_loss: 0.3540
- val_accuracy: 0.8542
Epoch 44/50
80/80 [==============================] - 3s 39ms/step - loss: 0.2933 - accuracy: 0.9035 - val_loss: 0.2982
- val_accuracy: 0.9055
Epoch 45/50
80/80 [==============================] - 3s 38ms/step - loss: 0.3064 - accuracy: 0.8927 - val_loss: 0.3024
- val_accuracy: 0.8912
Epoch 46/50
80/80 [==============================] - 3s 39ms/step - loss: 0.2614 - accuracy: 0.9250 - val_loss: 0.2703
- val_accuracy: 0.9220
Epoch 47/50
80/80 [==============================] - 3s 38ms/step - loss: 0.2874 - accuracy: 0.9019 - val_loss: 0.3098
- val_accuracy: 0.8820
Epoch 48/50
80/80 [==============================] - 3s 39ms/step - loss: 0.2767 - accuracy: 0.9046 - val_loss: 0.4646
- val_accuracy: 0.7915
Epoch 49/50
80/80 [==============================] - 3s 38ms/step - loss: 0.2641 - accuracy: 0.9119 - val_loss: 0.2647
- val_accuracy: 0.9151
Epoch 50/50
80/80 [==============================] - 3s 38ms/step - loss: 0.2705 - accuracy: 0.9054 - val_loss: 0.2728
- val_accuracy: 0.9053
```

In [15]:

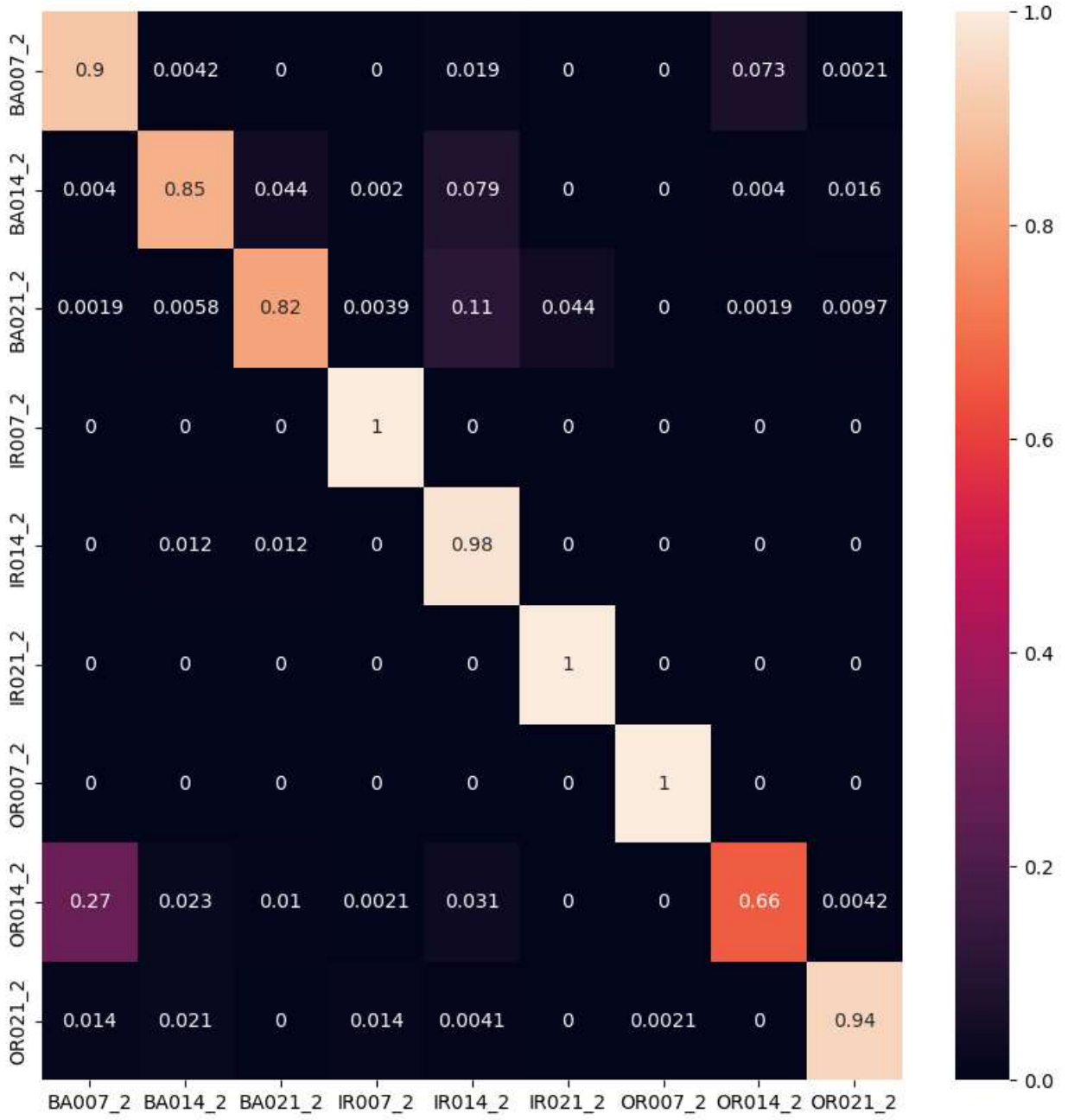
```python
# Function to inverse transform predictions
def inv_Transform_result(y_pred):
```

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

In [16]:
```
# 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 [17]:
```
# 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 [18]:
```
# Classification Report
print("Classification Report:")
print(classification_report(Y_test, Y_pred, target_names=encoder.classes_))
```

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

     BA007_2       0.75      0.90      0.82       478
     BA014_2       0.93      0.85      0.89       496
     BA021_2       0.93      0.82      0.87       517
     IR007_2       0.98      1.00      0.99       497
     IR014_2       0.77      0.98      0.86       431
     IR021_2       0.96      1.00      0.98       535
     OR007_2       1.00      1.00      1.00       464
     OR014_2       0.89      0.66      0.76       480
     OR021_2       0.97      0.94      0.96       485

    accuracy                           0.91      4383
   macro avg       0.91      0.91      0.90      4383
weighted avg       0.91      0.91      0.90      4383
```

In [19]:
```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 [20]:
```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: 0.9058
Precision per class:
BA007_2: 0.7559
BA014_2: 0.9289
BA021_2: 0.9251
IR007_2: 0.9781
IR014_2: 0.7995
IR021_2: 0.9574
OR007_2: 0.9979
OR014_2: 0.8926
OR021_2: 0.9672
Recall per class:
BA007_2: 0.9017
BA014_2: 0.8508
BA021_2: 0.8201
IR007_2: 1.0000
IR014_2: 0.9768
IR021_2: 1.0000
OR007_2: 1.0000
OR014_2: 0.6583
OR021_2: 0.9443
F1 Score per class:
BA007_2: 0.8224
BA014_2: 0.8881
BA021_2: 0.8695
IR007_2: 0.9889
IR014_2: 0.8793
IR021_2: 0.9782
OR007_2: 0.9990
OR014_2: 0.7578
OR021_2: 0.9556
```

In [21]:
```python
# Visualize Results
num_samples_to_visualize = 5
```

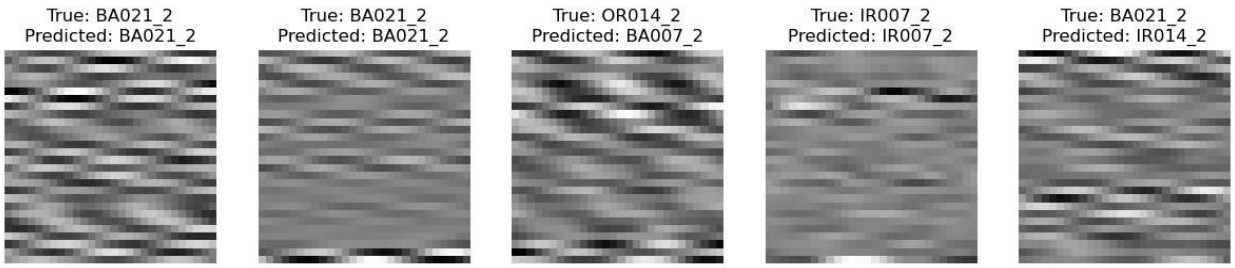
In [22]:
```python
# Randomly select some samples from the test set
random_indices = np.random.choice(len(X_test), num_samples_to_visualize, replace=False)
sample_images = X_test[random_indices]
true_labels = Y_test[random_indices]
```

In [23]:
```python
# Predict the labels for the selected samples
predicted_labels = inv_Transform_result(cnn_model.predict(sample_images))
```

```
1/1 [==============================] - 0s 45ms/step
```

In [24]:
```python
# 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 [ ]: