University of Alberta

Summary Report

Yongqing Liu

yo12@ualberta.ca

Jun 19th - Aug. 25th, 2023

TABLE OF CONTENTS

1. Project Overview

Over the past few decades, condition monitoring has become a crucial factor in the efficient operation of manufacturing and industrial systems. A noteworthy trend has emerged whereby machine learning (ML) is used to improve the monitoring and diagnosis of rotating machinery for optimal effectiveness. This change has significantly improved the precision and efficiency of evaluating the condition of various types of rotating machinery.

The main objective of this project is to use advanced machine learning techniques to perform fault diagnosis on the equipment located in the Applied Control & Diagnosis laboratory at the University of Alberta, utilizing a combination of simulated and actual machine-generated data.

2. Data Collection

The data utilized in all machine algorithms comes from a simulated machine model constructed in Simulink and sensors from a physical machine.

Table 1. Description of Data obtained

Data type	Essential data components employed in machine learning analyses
Simulated machine data	X and Y displacements at the positions of the left and right rotor, as well as the left and right bearings
Physical machine data	Left rotor acceleration in the x-axis

3. Methodology

Within the scope of this project, a variety of machine learning algorithms were utilized, encompassing both classification and regression methods. Specifically, I rigorously analyzed three complex machine learning algorithms, Extreme Learning Machine (ELM), Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN), within the MATLAB environment.

Furthermore, to enhance performance evaluation, I executed a CNN classification task using the PyTorch library within a Python environment. The utilization of PyTorch allowed for a superior evaluation of CNN's capabilities, consequently leading to a more accurate and dependable analytical result.

- Classification
  - Detect the specific fault type (if present) within the machine's operation
  - Three classes: no fault, static fault, dynamic fault
  - ELM
    - Categorize the operational status of the machine using simulation data. Code provided by Author of [1].
  - CNN
    - Employ wavelet transform plots of the left rotor's x-axis displacement signal to categorize the machine's operational state.

- Regression
  - Predict the value of rotor displacement in x & y directions given x & y displacement at bearing positions. Target number = 4.
  - ELM and RNN with LSTM layers are used for this task.

4. Results

4.1 Classification

- **ELM classification**

Features: Simulated machine data (8 total), fault weight is 6g.
Training and Testing data ratio = 75%: 25%
Activation function: Sigmoid (best performance)

Table 2. ELM classification results in an imbalanced data ratio for each class

Hidden layer neurons	20	35	50
No fault f1	0.8747	0.9341	0.9914
Static f1	0.7824	0.9141	0.9857
Dynamic f1	0.9162	0.9584	0.9947

Table 3. ELM classification results in a balanced data ratio for each class

Hidden layer neurons	20	35	50
Training Accuracy	84.76%	96.44%	98.70%
Testing Accuracy	84.67%	96.48%	98.57%

- **CNN classification on wavelet transforms plots**

  ***** The plots are created by extracting 2-second signal segments of simulated machine data for each fault type during a 5-minute simulation period (148 segments). In this task, the dynamic fault data is obtained from simulations with different weights on the flywheels: 8g on the left side and 4g on the right side.

  Sample wavelet plots [Figure 0]

Self-built CNN:

- Achieved a training, validation, and testing accuracy of **100%**. [Figure1]
- CNN architecture [Figure 2]

Configurations:

1. The number of samples is balanced for each class in the dataset, also when splitting into training, validation, and test datasets.
2. Batch size = 2; loss function = cross-entropy loss; optimizer = Adam; learning rate0.001
3. Data ratio: 60% training, 20% testing, and 20% validation

Transfer learning with VGG16:

- The testing and validation accuracy is **100%** and the training accuracy is around **99%**. [Figure 3]
- Extract the bottom layers of VGG16 with frozen weights, then add three fully connected layers for the classification task of 3 outputs [2] [Figure 4].

Configurations:

1. Configuration is the same compared to self-built CNN except that the learning rate is 0.0001 and the optimizer used is rmsprop.

4.2 Regression

- **ELM & RNN regression in MATLAB**

Data: Simulated machine data, fault weights = 4g or 6g

Train: Test = 75%: 25%

Activation function: Sigmoid (best performance)

Self-built RNN layout [Figure 5]

Table 4. Training and testing accuracy of RLM and RNN

Data set	RNN		ELM	
	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
Static 4g fault	3.0	1.4531	1.599	1.603
Static 6g fault	3.0	1.5213	1.617	1.613
Dynamic 4g fault	2.1	1.0777	1.307	1.306
Dynamic 6g fault	2.0	1.0074	1.262	1.259
Stat + Dyn 4g	3.2	1.6070	1.983	1.975
Stat + Dyn 6g	3.1	1.5492	2.017	2.004
Stat 4g + Stat 6g	4.0	1.9396	2.101	2.116
Dyn 4g + Dyn6g	2.05	1.0434	1.219	1.219

*ELM has much less training/testing time

Table 5. Out-of-sample performance of ELM and RNN for regression

Data set		RNN		ELM	
Train	Test	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
Stat 4g	Stat 6g	1.5	*2.6767	1.599	2.954
Dyn 4g	Dyn 6g	1.2	0.6148	1.3165	1.3567

- **RNN regression in Python**

  Optimizer: adam, learning rate = 0.001, 2 LSTM layers

Table 6. Training and testing accuracy of RNN

Dataset	Training RMSE	Testing RMSE
Static 4g	1.2464	1.2891
Static 6g	1.2429	1.3120
Dynamic 4g	1.0097	1.1216
Dynamic 6g	0.9688	1.0063

The performance of RNN is better in a Python environment.

In terms of out-of-sample performance, for both cases, the model has a more minor (improved by almost 1) training RMSE value compared to Table 5, but a slightly smaller test RMSE value (improved by 0.2).

- **ELM regression error signal**

  The model has undergone training using both the static fault 4g dataset and the dynamic fault 4g dataset. The features utilized for training encompass the x and y displacement in the bearing positions, while the target value entails the rotor's position displacement in the x and y dimensions. After this training, the model is equipped to generate predictions for the rotor's position in the forthcoming seconds, assuming the rotor remains operational.

  To evaluate the accuracy of these predictions, an error signal is derived through the subtraction of the predicted value from the actual value. This actual value is obtained

by extending the simulation timeframe beyond that of the training dataset, providing a more comprehensive basis for comparison [Figure 6].

5. Conclusion

This report provides a comprehensive overview of the undertaken work and the subsequent results obtained through the application of machine learning techniques on the rotor machine dataset.

In the realm of fault type classification, both Convolutional Neural Network (CNN) and Extreme Learning Machine (ELM) demonstrated exceptional performance, yielding nearly 100% accuracy across both the training and testing datasets.

For the regression task, which specifically focuses on predicting the x and y displacement of the rotor, the Recurrent Neural Network (RNN) utilizing Long Short-Term Memory (LSTM) layers emerged as the superior performer when compared to the ELM algorithm. Notably, this achievement was witnessed within a Python-based simulation environment.

Interestingly, ELM excelled in terms of efficiency by substantially reducing the time required for model training and testing in comparison to the other two algorithms employed for these tasks.

These findings collectively underscore the effectiveness of machine learning methodologies in the context of rotor machine analysis, with notable accomplishments in both classification and regression scenarios.

6. Future Work

The detection of the operating condition of the rotor model, applicable to both simulation and physical machine scenarios, involves the utilization of 8+ distinct features for each setting. Employing Principal Component Analysis (PCA) to visualize this data serves as a valuable technique for identifying the most informative features. Anomaly detection further enhances this process, enabling the identification of the specific fault type the machine might have encountered.

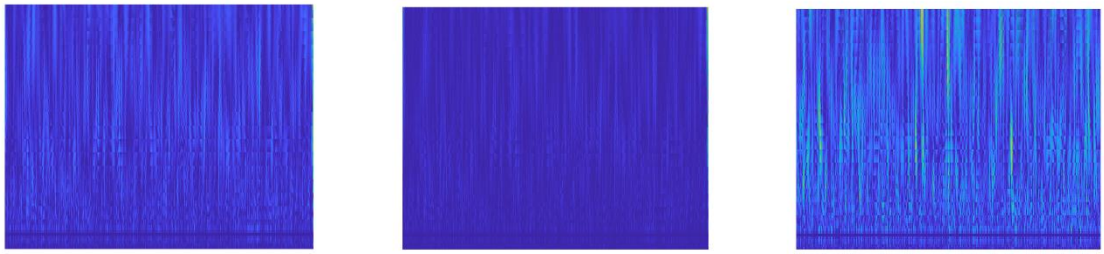
Another promising avenue lies in the fusion of Reinforcement Learning (RL) techniques with either classification or regression applications. This synergistic approach, when combined with

subspace methods, can yield substantial insights into the classification or prediction of fault occurrences within the rotor system.

These methodologies collectively represent an effective and comprehensive strategy for addressing the multifaceted task of detecting and characterizing faults in the rotor model, encompassing visualization, anomaly detection, reinforcement learning, and subspace techniques. Such a multifarious approach holds great potential for advancing the precision and sophistication of fault detection mechanisms in this domain.

Appendix

Figure 0: Wavelet transform plots for each class at same time interval



No fault | Static fault | Dynamic fault

Figure 1: Training result of self-built CNN on classification

```
Epoch [1/5], Training Loss: 29.5324, Training Accuracy: 82.50%, Validation Accuracy: 66.67%
Epoch [2/5], Training Loss: 13.7147, Training Accuracy: 87.08%, Validation Accuracy: 100.00%
Epoch [3/5], Training Loss: 1.8556, Training Accuracy: 97.92%, Validation Accuracy: 100.00%
Epoch [4/5], Training Loss: 0.0000, Training Accuracy: 100.00%, Validation Accuracy: 100.00%
Epoch [5/5], Training Loss: 0.0000, Training Accuracy: 100.00%, Validation Accuracy: 100.00%
Training finished
```

Figure 2: Self-built CNN architecture

```python
class CNN(nn.Module):
    def __init__(self, dropout_prob=0.7):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(8)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(dropout_prob)
        self.fc1 = nn.Linear(8 * 328 * 437, 3)

    def forward(self, x):
        x = self.pool(self.relu(self.bn1(self.conv1(x))))
        x = self.dropout(x)
        x = x.view(-1, 8 * 328 * 437)
        x = self.fc1(x)
        return x
```

Figure 3: Training result of transfer learning approach with VGG16

```
Epoch [1/5] - Train Loss: 0.4398 - Val Loss: 0.0052 - Train Acc: 89.58% - Val Acc: 100.00%
Epoch [2/5] - Train Loss: 0.0987 - Val Loss: 0.0008 - Train Acc: 97.92% - Val Acc: 100.00%
Epoch [3/5] - Train Loss: 0.1004 - Val Loss: 0.0038 - Train Acc: 99.17% - Val Acc: 100.00%
Epoch [4/5] - Train Loss: 0.0668 - Val Loss: 0.1597 - Train Acc: 99.17% - Val Acc: 96.30%
Epoch [5/5] - Train Loss: 0.0792 - Val Loss: 0.0007 - Train Acc: 98.75% - Val Acc: 100.00%
```

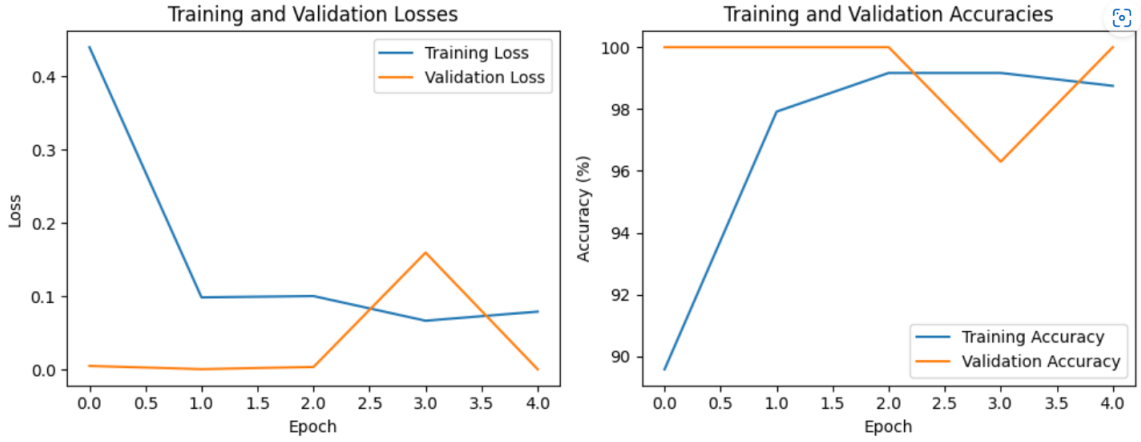


Figure 4: Layout and configuration of VGG16 transfer learning CNN model

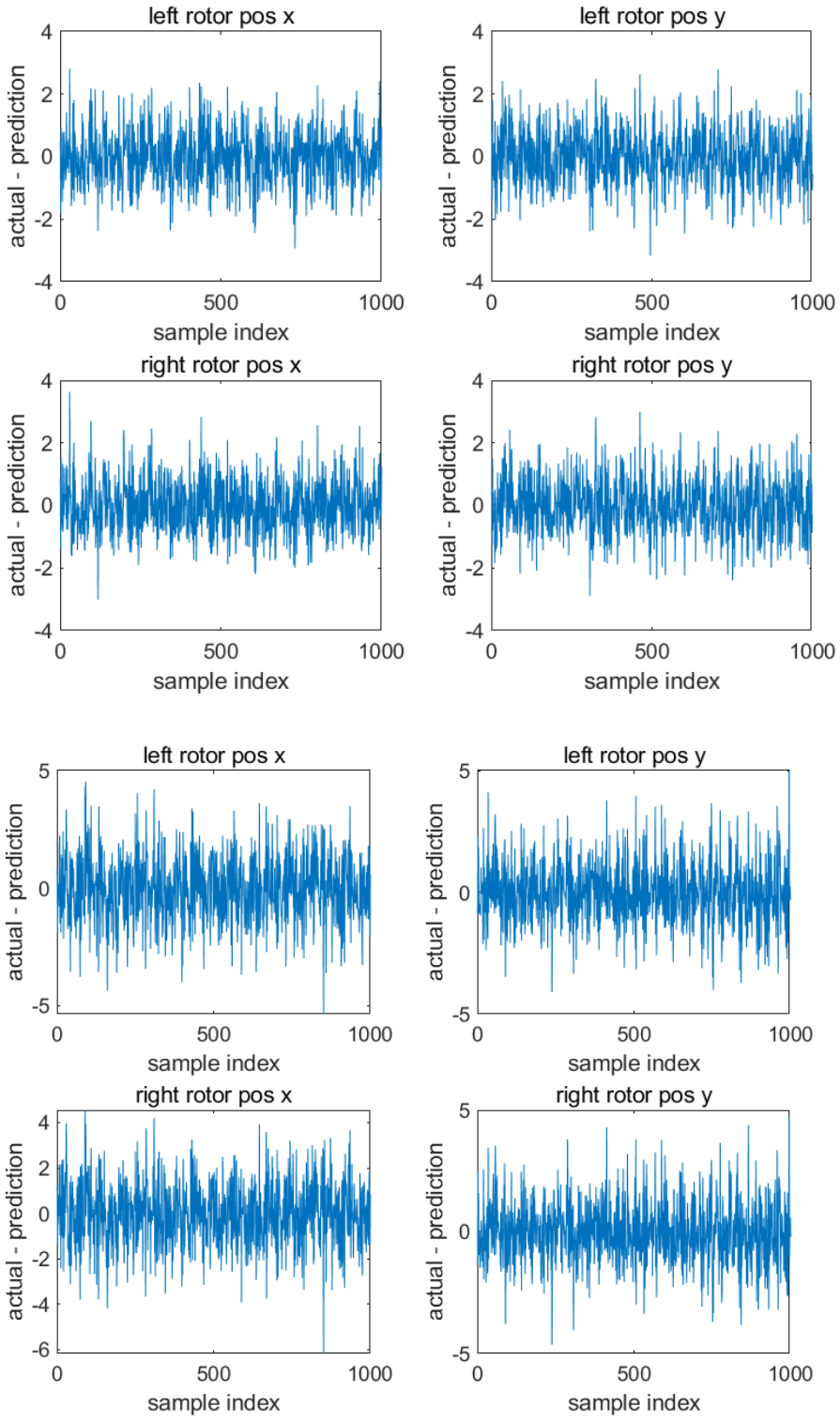
```python
for param in vgg16.features.parameters():
    param.requires_grad = False
# Extract the bottom layers
bottom_layers = vgg16.features
```

```python
# Define the additional fully connected layers
fully_connected_layers = nn.Sequential(
    nn.Linear(512 * 7 * 7, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(0.3),

    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(0.3),

    nn.Linear(4096, 3),
)

# Combine layers to create the complete classification network
classification_network = nn.Sequential(
    bottom_layers,
    nn.Flatten(),  # Flatten the output from the bottom layers
    fully_connected_layers,
)
```

Figure 5: Self-built RNN layout & and configuration for regression task

```
layers = [ ...
    sequenceInputLayer(4)
    lstmLayer(45,'OutputMode','sequence')
    fullyConnectedLayer(4)
    regressionLayer];

options = trainingOptions('adam', ...
    'MaxEpochs',50, ...
    'MiniBatchSize',5000, ...    % 5000
    'InitialLearnRate',0.1, ...
    'GradientThreshold',1, ...
    'Shuffle','never', ...
    'Plots','training-progress',...
    'Verbose',0);
```

Figure 6: Comparative error signal plots depicting different positions for the dynamic fault signal (upper) and static fault signal (lower).



References

[1] Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew, Extreme learning machine: Theory and applications, Neurocomputing, Volume 70, Issues 1–3, 2006, Pages 489-501, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2005.12.126.

[2]
Cheick Abdoul Kadir A. KOUNTA, Lionel ARNAUD, Bernard KAMSU FOGUEM, Fana TANGARA, Deep learning for the detection of machining vibration chatter