Bearing Fault Classification using Deep Q Learning: A Deep Reinforcement Learning Approach

1. Introduction:

Fault prognosis and diagnosis of Machines has become an important subject in the field of Mechanical Engineering. Early prediction and diagnosis can help industries to be more productive and economical. Prognosis and Diagnosis has become an integral part of Industry 4.0.

In this project, a deep reinforcement learning approach is used to classify bearing fault modes by using the method of Deep Q Learning. The Deep Q Network has a 2D Convolution Neural Network and Fully Connected Layer to obtain q-values for a given State. Vibration data is converted into Continous Wavelet Transform Scalograms images and are fed to the network.

2. Data:

The data is obtained from Case Western University Bearing Data. The data contains vibration data of 2 HP Reliance Motor. Vibration Data is taken from the Drive End and Fan End.

Motor bearings are installed with faults using electro-discharge machining (EDM). Faults ranging from 0.007 inches in diameter to 0.040 inches in diameter were introduced separately at the inner raceway, rolling element (i.e. ball) and outer raceway. Faulted bearings were reinstalled into the test motor and vibration data was recorded for motor loads of 0 to 3 horsepower (motor speeds of 1797 to 1720 RPM).

There are 4 condition of Operation:
1. Normal Condition without Fault.
2. Inner Race Fault.
3. Ball Fault.
4. Outer Race Fault.

The data of all 4 files in Normal Condition is used in this project. For the fault data, the data of 12k Drive End Fault Data is used. In this data link, all the files of inner race, ball fault, outer race centerd @ 6 are used. Also 4 files of Outer Race Orthogonal @ 3 with fault depth of 0.021 inches are used.

These data files are in matlab format and data containing on ly Drive End Accerometer data is used for making Scalograms.

2.1 Data Generation:

The data of all four conditions are used to create continous Wavelet Transform Scalograms. The drive end data is used to get scalograms. The images generated are 64 x 64 x 3 images. This means that the signals were cut into arrays of 64 elements. This array is used to generate CWT Scalograms. For the scales parameter to generate scalogram, scales from 1 to 65 are used. Mexican Wavelet is used as a parameter for the wave form.

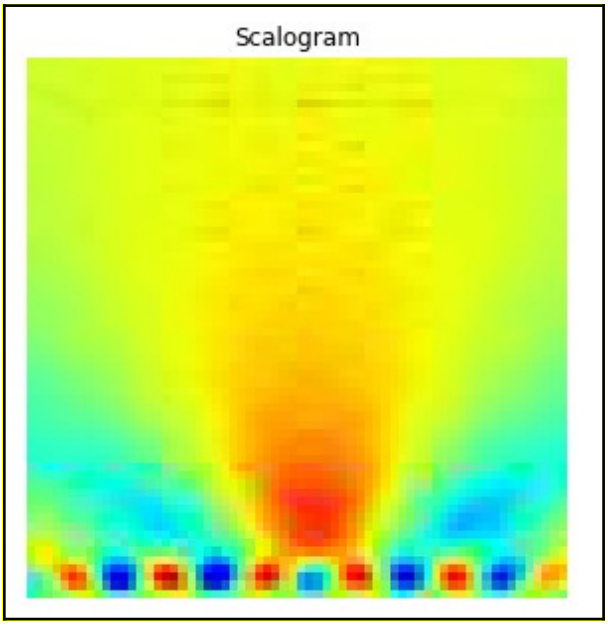


Figure 1: Scalogram example 1

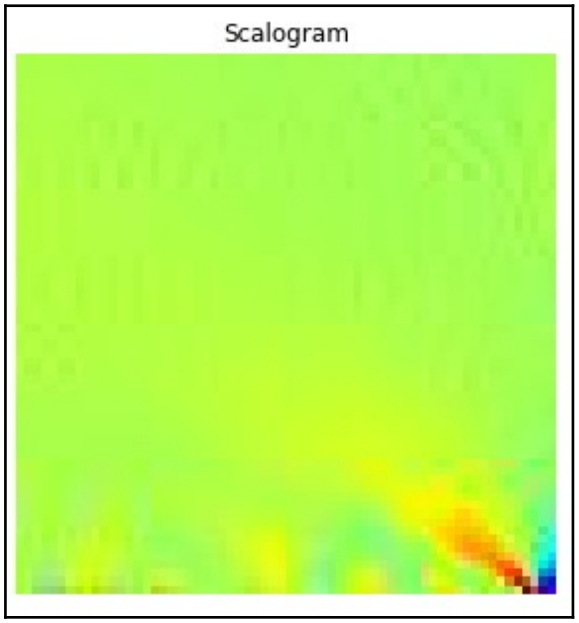


Figure 2: Scalogram example 2

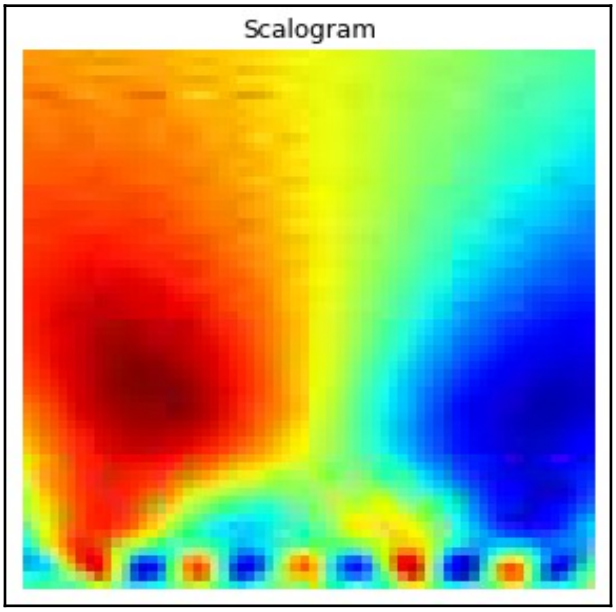


Figure 3: Scalogram example 3

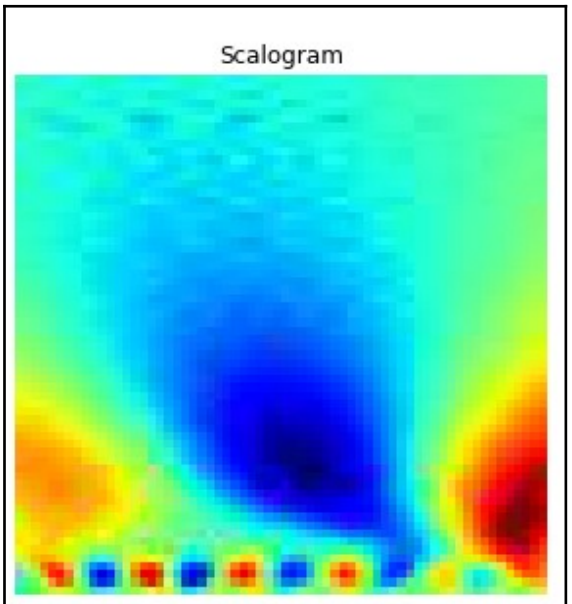


Figure 4: Scalogram example 4

2.2 Data Split:

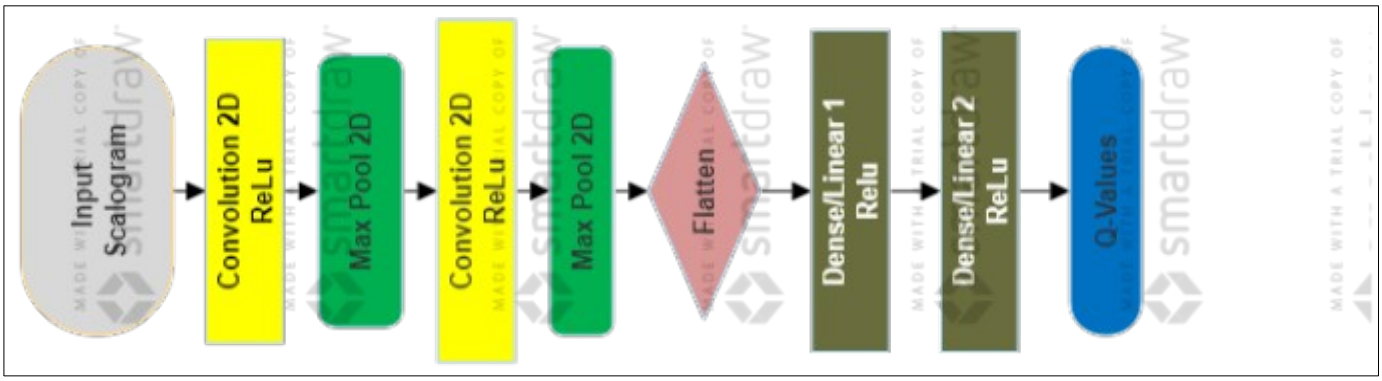
The scalograms generated are spilt into 95 % training and 5 % testing. All the training data is stored in 1700 batch files as evenly as possible. Each Batch contains images of all fault condition with almost equal propotions. All the test data are stored in a single file for testing.

3. Methodology

3.1 Deep Q Network:

The deep Q network essentially takes state as input and gives q-values as output. Here the state is an image and the output are action-values a.k.a q-values. In this project 2 layers of 2-D convolution with filters 64 and 32 respectively are used along with stride of 2 and kernel size of 3 for both layers. After convolution, ReLu activation is applied. After each convolution, the data is passed to a MaxPool Layer with kernel size of 2 and stride of 1.

The data is then flattened and sent to Two layers of Fully Connected Layer with 64 and 32 units which also get ReLu Activation. The final layer has 4 units which represent q-value for each action.




```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [-1, 64, 31, 31]          1,792
├─MaxPool2d: 1-2                         [-1, 64, 30, 30]          --
├─Conv2d: 1-3                            [-1, 32, 14, 14]          18,464
├─MaxPool2d: 1-4                         [-1, 32, 13, 13]          --
├─Linear: 1-5                            [-1, 64]                  346,176
├─Linear: 1-6                            [-1, 32]                  2,080
├─Linear: 1-7                            [-1, 4]                   132
==========================================================================================
Total params: 368,644
Trainable params: 368,644
Non-trainable params: 0
Total mult-adds (M): 5.62
==========================================================================================
Input size (MB): 0.05
Forward/backward pass size (MB): 0.52
Params size (MB): 1.41
Estimated Total Size (MB): 1.97
==========================================================================================
```

Figure 5: Deep Q Network and Network Summary

3.2 Replay Buffer/Memory:

In order to update the weights in the network, a replay memory is used. The replay memory stores parameters of (state, action, reward, next state, done). During back propagation, a sample from the memory is used to train the network. Here the size of memory is 1E06. The sample batch is 32.

3.3 Agent:

The agent interacts with the environment and takes action. Its also observes rewards and next states. Here the agent contains two DQN Networks i.e. Local Network and Target Network. Both networks are used to obtain losses during training. The agent has to observe a given state and take an action. It has to recive reward and take actions on next state. It stores these state, action, reward, next state information and learn from them during training.

3.4 Environment:

The Environment has to output states to the agent, take in actions from the agent and output rewards for that action and next state. The environment give +1 reward for correct actions and -1 reward for incorrect actions. The environment is designed to be a game/quiz like environment.

3.5 The game:

The training takes place like an episodic quiz. Here there are total of 1700 Episodes with 58-62 scalograms in each episode/batches. The goal of an agent is to observe one scalogram at a time and predict which class the scalogram belongs. A correct answer yields +1 and -1 for incorrect answer. The agent has to improve its score and train itself to maximize the score.

4. Analysis

4.1 Training

The training takes place in the following steps:

1. The number of games/episodes are intialised to 1700 and epsilon value is set to 1.

2. Each Episode has 62-58 states/images.

3. The agent observes an image and passes it the the DQN Local Network. The DQN outputs a matrix of 4 rows with q values. The maximum value index in the q value matrix becomes the action based on epsilon value.

4. The agent selets to explore or exploit based on episolon value. A Random action is taken if the agent chooses to Explore or the maximum value index in the q value martrix if the agent chooses to exploit

5. The agent passes the action to the environment. The environment outputs the reward, next state and whether the episode is over or not.

6. The agent observes the reward and next states. The agent stores these (states, actions, reward, next states, done) in the replay memory.

7. The agent will only train the network once there are enough samples in the memory. Here the sample batch is set to 32. This means that the weights in the DQN will not be updated in the first 32 observations. Once there are enough samples, the agent samples from the memory and trains the network.

8. The next states are passed to the target network and states are passed to the local network.

9. The loss between the target and local networks is calculated and then the local network undergoes backpropagation.

10. The weights from the trained local network is copied to the target network. The copied weights are tweaked a little bit(soft update).

11. The above process continues till all the episodes are over.

12. Each episode rewards are summed up to get the score the agent obtains in each episodes.

As the training takes place the agents goal is to get a score of 58-62 depending on the number of scalograms in an episode/game.

5. Results

5.1 Training Results:

During the initial part of the training, the agent score were in negative side. As the agent learns from the observations, the score increase slowly and gradually. When the agent scored an average score of 36 in previous 100 games, the training is stopped and the weights of the networks are stored.

```
Episode: 0 =====> Average Score: -30.0
Episode: 10 =====> Average Score: -30.0
Episode: 20 =====> Average Score: -29.904761904761905
Episode: 30 =====> Average Score: -28.967741935483872
Episode: 40 =====> Average Score: -27.170731707317074
Episode: 50 =====> Average Score: -25.96078431372549
Episode: 60 =====> Average Score: -24.0
Episode: 70 =====> Average Score: -22.169014084507044
Episode: 80 =====> Average Score: -20.641975308641975
Episode: 90 =====> Average Score: -19.626373626373628
Episode: 100 =====> Average Score: -18.0
Episode: 110 =====> Average Score: -15.6
Episode: 120 =====> Average Score: -12.96
Episode: 130 =====> Average Score: -10.96
Episode: 140 =====> Average Score: -9.6
Episode: 150 =====> Average Score: -7.78
Episode: 160 =====> Average Score: -6.88
Episode: 170 =====> Average Score: -6.06
Episode: 180 =====> Average Score: -5.52
Episode: 190 =====> Average Score: -4.48
Episode: 200 =====> Average Score: -4.58
Episode: 210 =====> Average Score: -4.28
Episode: 220 =====> Average Score: -3.48
Episode: 230 =====> Average Score: -1.16
Episode: 240 =====> Average Score: 1.7
Episode: 250 =====> Average Score: 3.98
Episode: 260 =====> Average Score: 6.64
Episode: 270 =====> Average Score: 9.04
Episode: 280 =====> Average Score: 11.66
```

Figure 6: Start of Training with negative scores

```
Episode: 620 =====> Average Score: 32.36
Episode: 630 =====> Average Score: 32.66
Episode: 640 =====> Average Score: 32.8
Episode: 650 =====> Average Score: 33.08
Episode: 660 =====> Average Score: 33.02
Episode: 670 =====> Average Score: 32.82
Episode: 680 =====> Average Score: 33.28
Episode: 690 =====> Average Score: 33.32
Episode: 700 =====> Average Score: 33.3
Episode: 710 =====> Average Score: 33.52
Episode: 720 =====> Average Score: 33.54
Episode: 730 =====> Average Score: 33.78
Episode: 740 =====> Average Score: 34.42
Episode: 750 =====> Average Score: 34.32
Episode: 760 =====> Average Score: 34.34
Episode: 770 =====> Average Score: 34.56
Episode: 780 =====> Average Score: 34.44
Episode: 790 =====> Average Score: 34.38
Episode: 800 =====> Average Score: 34.78
Episode: 810 =====> Average Score: 34.6
Episode: 820 =====> Average Score: 34.52
Episode: 830 =====> Average Score: 34.36
Episode: 840 =====> Average Score: 33.98
Episode: 850 =====> Average Score: 34.06
Episode: 860 =====> Average Score: 34.4
Episode: 870 =====> Average Score: 34.44
Episode: 880 =====> Average Score: 34.7
```

Figure 7: Evolution of scores as the training progresses. Here it has reached a score of 30

5.2 Testing Results

The trained weights are loaded and the DQN Network is used to test scalograms. The scalogram is passed to the network and it outputs the Q-value matrix. The maximum value index is the fault condition.

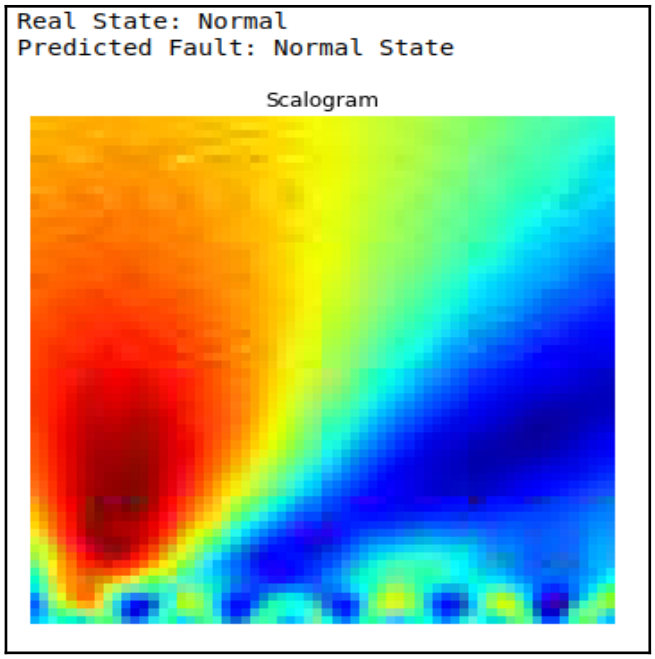


Figure 8: Normal State is classified as normal state

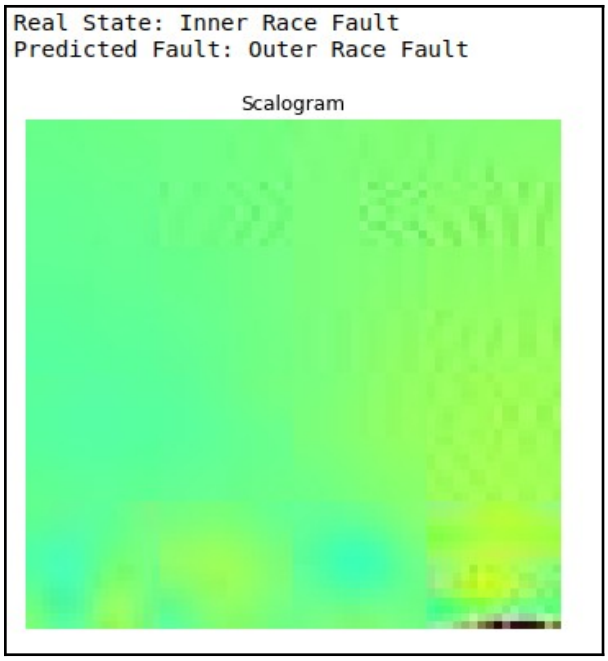


Figure 9: Inner Race Fault is misclassified as outer state

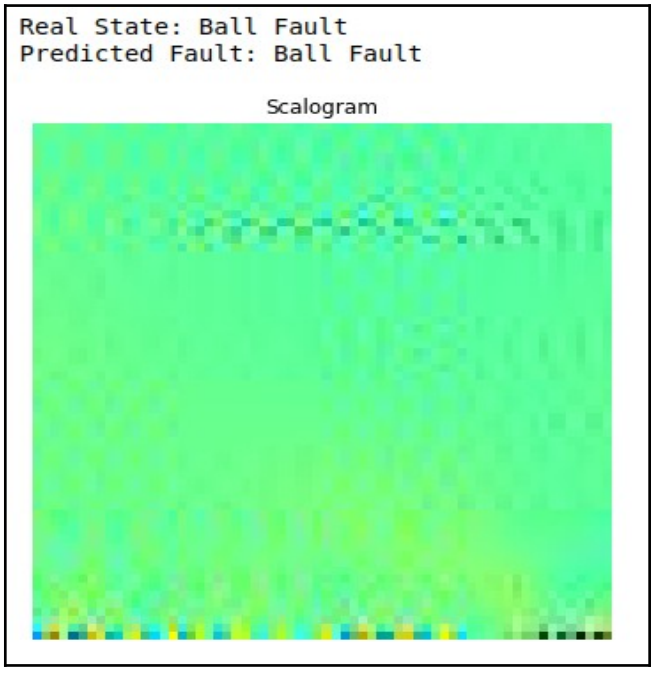


Figure 10: Ball Fault State is classified as ball fault state

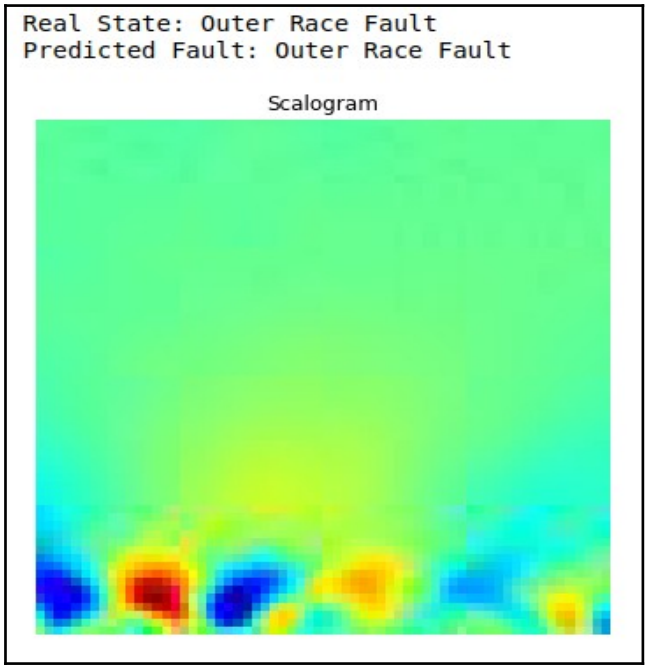


Figure 11: Outer Race Fault State is classified as outer race fault state

5.3 Accuracy:

The accuracy of the agent is calculated with the number of scalograms it can classify correctly. There are total 21657 scalograms in the test set. The agent classified 20082 scalograms correctly and 1575 scalograms miscorrectly. The total accuracy is found to be 93 %.

```
Total Images in Test Set:  21657 Images
Highest Score that can be obtained:  21657
Score Obtained with trained agent:  20082
Accuracy of Agent:  93.0  %
```

Figure 12: Testing Accuracy

6. Conclusions:

The trained agent obtained an average score of 36 during training and predicted 93 % of the testing image correctly. Further imporvements can be made by changing the network architecture of the network and including more scalogram data from the fan end.

7. References:

1. Paper Reference:

Intelligent fault diagnosis for rotating machinery using deep Q-network based health state classification: A deep reinforcement learning approach by: Yu Ding, Liang Ma, Jian Ma, Mingliang Suo, Laifa Tao, Yujie Cheng, Chen Lu.

2. Data Reference: Case Western University Bearing Data

https://engineering.case.edu/bearingdatacenter/download-data-file