DATA-DRIVEN DIGITAL TWIN

Abstract

Digital twin technology provides us opportunities to accurately model the process, resources such as manufacturing machines and outcomes in the manufacturing industry. They are key to our progress towards Industry 4.0 Given the importance of maintaining the health of machines involved in a manufacturing process, Artificial Intelligence and Machine Learning can be utilized to a large extent to optimise processes by using sensor data to achieve predictive maintenance. As bearings form primary components of all mechanical equipment, effective bearing health monitoring is essential for avoiding overhead costs and to efficiently run manufacturing processes. Our work focuses on understanding of bearing health through ML-based vibration monitoring. We use Deep learning models to accurately detect a fault in bearings along the type of fault(s) if any. We consider Artificial Neural Networks and LSTM based methods to enable fault detection on the provided time series data of vibration and power.

Introduction

The purpose of this model is to classify bearing health conditions in mechanical systems thereby enabling us to implement a data-driven digital twin approach. We classify bearing health broadly into 4 common categories.

1. **Inner race fault:**
   Any crack or fracture in the inner race of a ball or roller bearing causes what we call an inner race fault.
2. **Outer race fault:**
   When cracks or fractures happen in the outer raceway, we term it an outer race fault. Raceway faults can occur due to true or false brinelling, lubricant failure, corrosion etc.
3. **Cage Broken:**
   The bearing cage (also called a separator or a retainer) is the metal structure that holds the rolling elements (balls or cylindrical rollers) in proper position and orientation so that they don't group together. Cage damage includes cage deformation, fracture and wear, fracture of cage pillars etc. Possible causes include poor mounting (misalignment), large moment load, shock or large vibration.
4. **Healthy**
   When none of the aforementioned faults have been observed, we term it a healthy bearing.

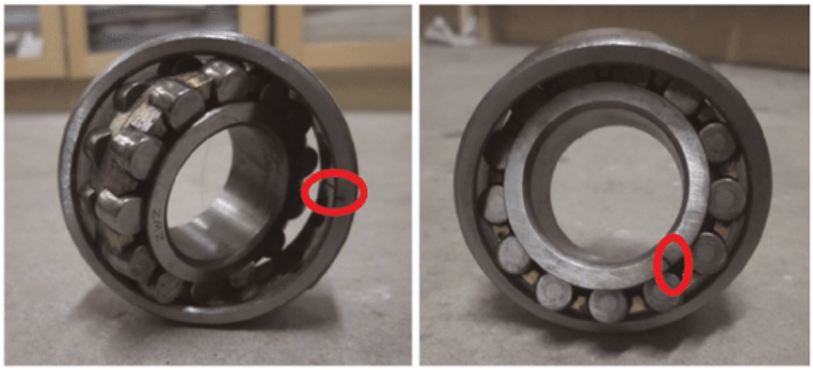


Fig 1.a: outer raceway fault (left), inner raceway fault (right)

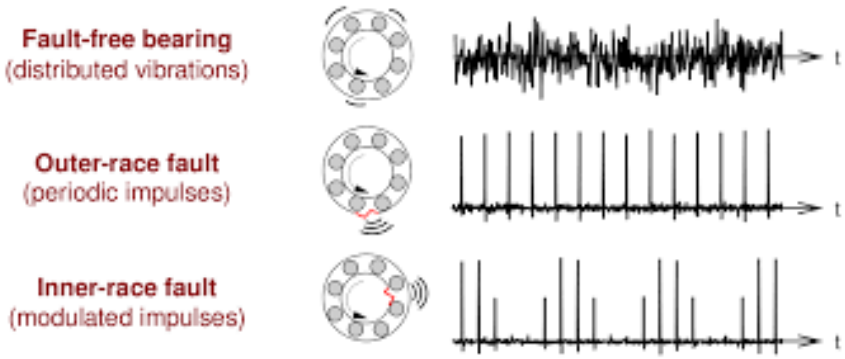


Fig 1.b: periodicity of vibration holds a lot of information about the bearing condition

From the above image (Fig 1.b), it is fairly evident that the vibration signal holds key information regarding the bearing health condition and especially in classifying whether it is an inner race or an outer race fault. Insights into our data on vibration is discussed further below.

Data

Visualization and Insights:

In this, We are going to plot our dependent variables i.e. Vibration and Power with sequence length of 400 for the 4 datasets where each one represents a different class to get an idea of its behaviour.

1) **Vibration:**

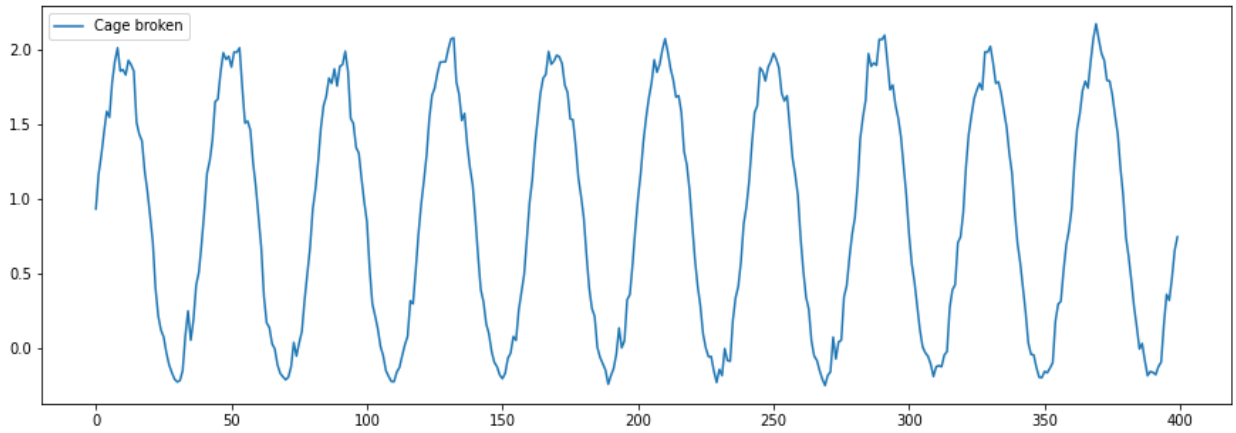


Fig 1a.Vibration vs Cage broken class (dataset)

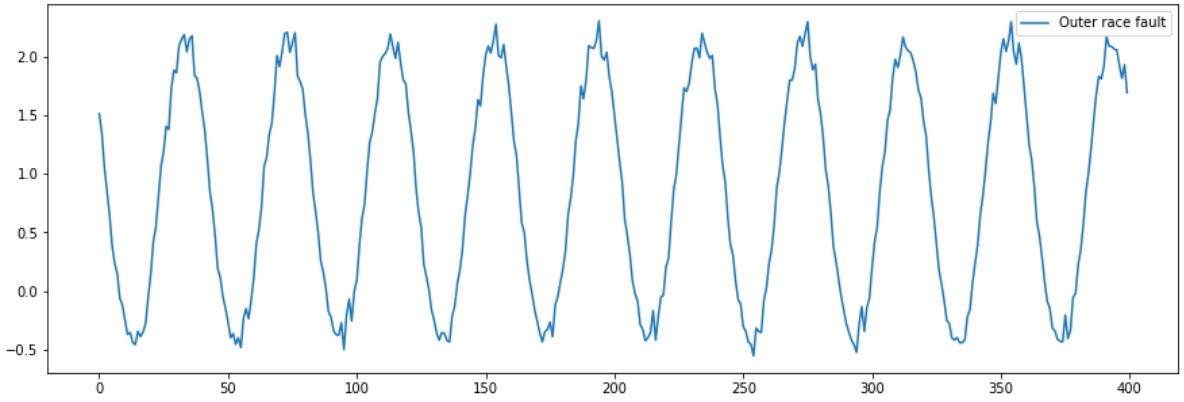


Fig 1b. Vibration vs Outer race fault class (dataset)

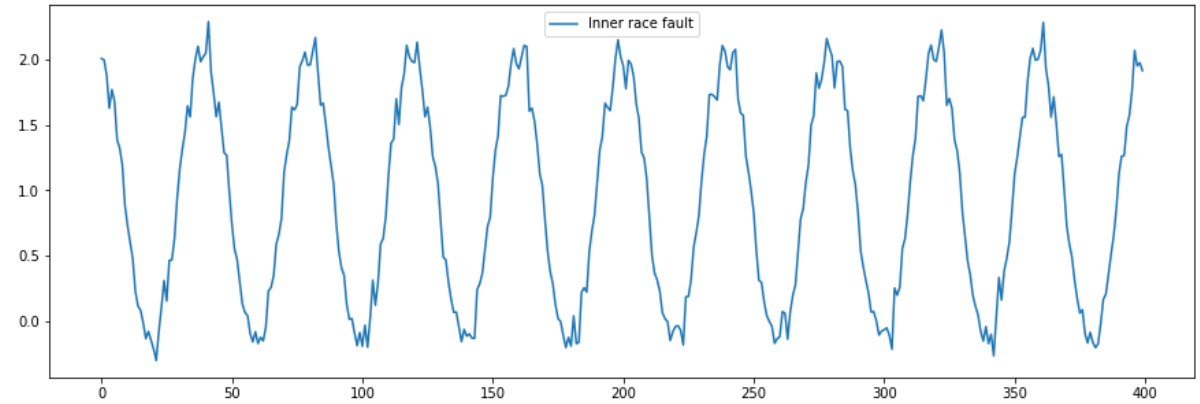


Fig 1c. Vibration vs Inner race fault class (dataset)

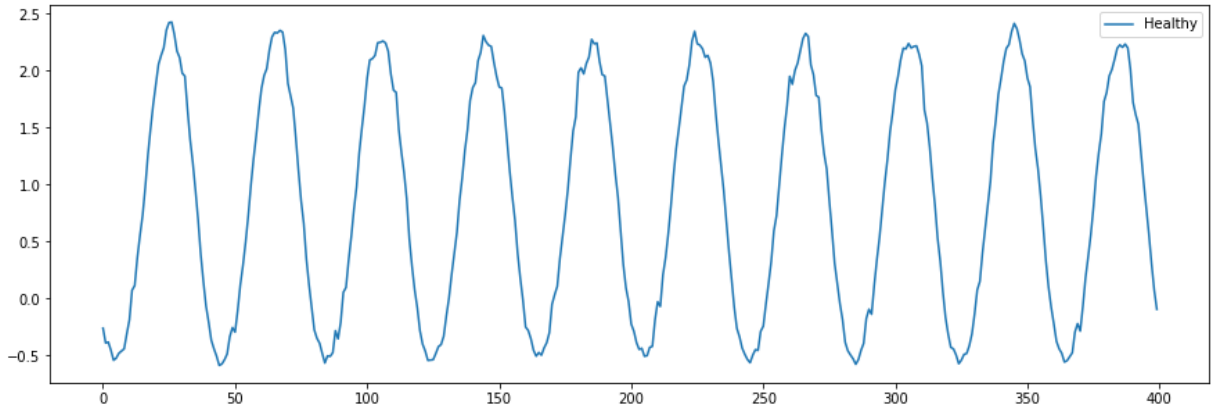


Fig 1d. Vibration vs Healthy class (dataset)

Observations:

From the above plots for this variable, we can observe that all the classes are showing periodic behavior and the periodicity is approximately 40 points. Also, The peaks aren't smooth and are instead showing zig-zag behaviour in Figures 1a,1b and 1c clearly indicating the defective nature of those classes unlike 1d where the peaks are almost perfectly smooth like that of sinusoidal. Inner and Outer Race faults are especially showing more rigidity compared to Cage broken.

2) Power:

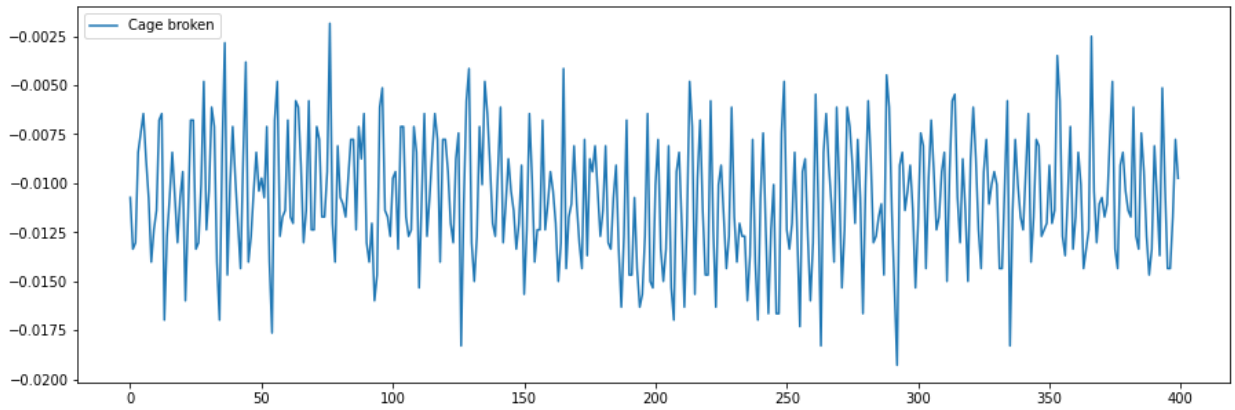


Fig 2a. Power vs Cage Broken class (dataset)

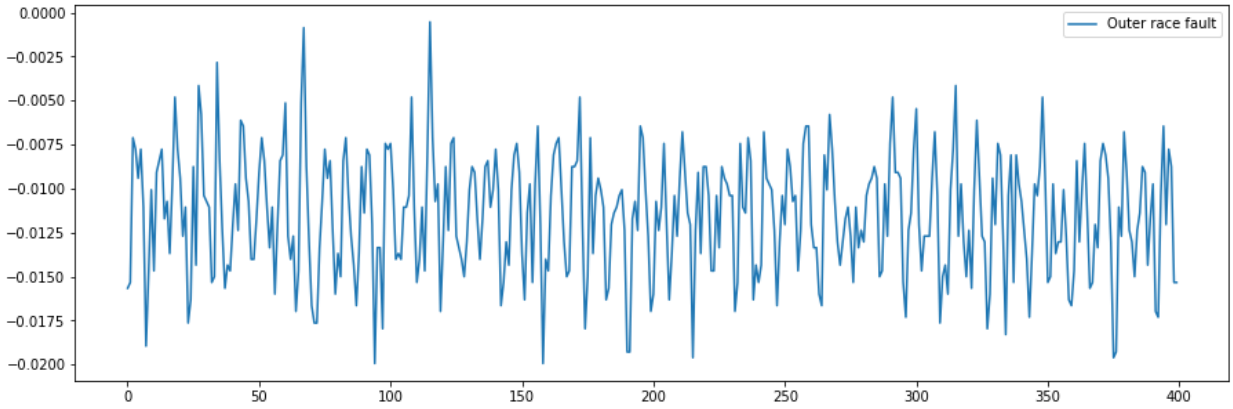


Fig 2b. Power vs Outer race fault class (dataset)

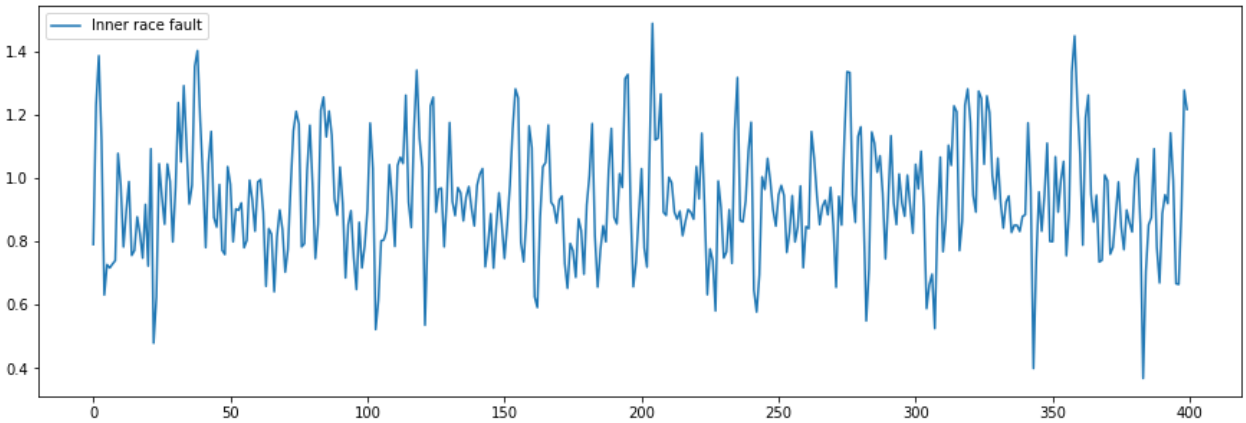


Fig 2c. Power vs Inner race fault class (dataset)

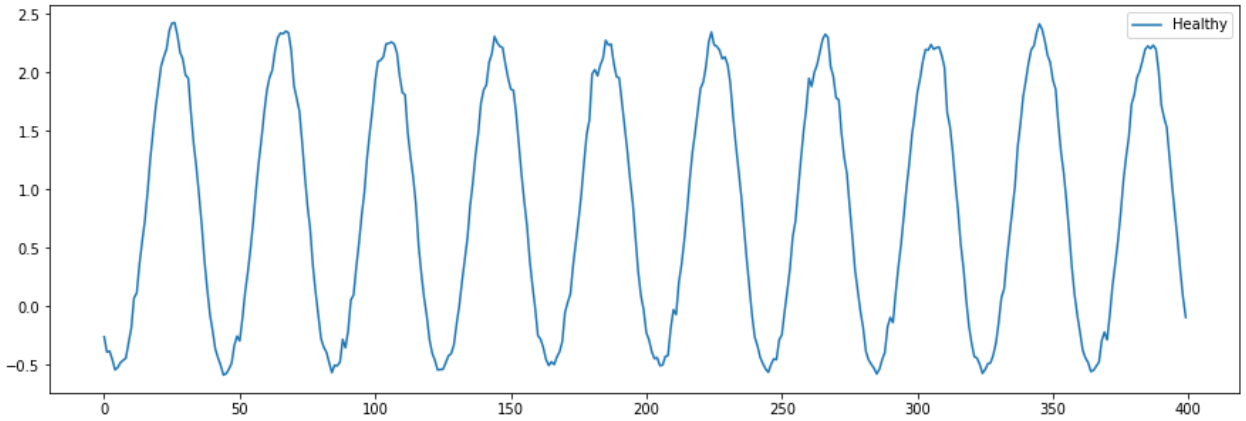


Fig 2d. Power vs Healthy class (dataset)

Observations:

From the above plots, We can observe from figures 2a, 2b and 2c the extreme zigzag behaviour of the power variable showing it's volatile and unstable nature. Also, We have found an anomaly with the data wherein for the Power data on Inner Race fault, the values are of the order of 1.4-2 while for the other three classes(healthy, Outer Race fault and cage broken), power values are in the order 0.0025-0.01. Finally, The power data corresponding to the health class shows a periodic behaviour and also the peaks are smooth similar to that of the plot given by vibration data.

Training Data Generation

Time-series based models will mostly rely on the past data which is also called a lookback period to make any prediction on the future data. A "lookback period" defines how many previous timesteps are used in order to predict the subsequent time step. We have made a training data generation package that accepts time-series data along with a look back period as input and provides training data as an output as seen in fig 3.

0.1	0.3	0.4	0.5	0.6	.....	.....	0.4	0.3	0.1

Segmenter

col1	col2	.....	col_lookbackperiod	Class
0.1	0.2	.....	0.4	A
0.6	0.36	.....	0.2	B

0.7	0.8	.....	0.5	A

Figure 3. Data processing by Segmentor

By assigning the segmenter class to any object along with a lookback period an empty list for X, y will be initialized. Add array can be utilized to add records into the X with a single type label. However, the value of the number of records that are to be generated, time-series data along with the class name has to be passed. Each record contains a random sequence with the length of the lookback period, we made sure that there will be no duplicates present in the generated data. This addarray can be utilized as many times as required according to the number of classes present. With return data, we can obtain the data frame with the training data which is properly shuffled to ensure no 5/10 continuous records have the same class. Sample code for data generation can be seen in sample code 1.

```
class segmenter:
    def __init__(self,sequence_length):
        self.X_data = [] # List for time series data.
        self.y_data = [] # Labels
        self.l = sequence_length #Lookback period
    def addarray(self, data_t, class_type, n):
        done_locations = set() # Store index number of the records
        i = 0 #For counting the size of generated data
        while i < n:
            # Randomly selecting staring location in the time series
            start = random.randint(0,len(data_t)-self.l)
            if start not in done_locations: # Check if this record is already
generated
                i+=1 # Increase the count by one.
                done_locations.add(start) # Updating the completed record
                # Appending a sequence from start to start + lookback
                # period in to the training list
                self.X_data.append(data_t[start:start+self.l])
                self.y_data.append(class_type) \\Appending the label
    def returndata(self): # Returning generated dataframe
        X_data = np.array(self.X_data) # Converting list to numpy array
        y_data = np.array(self.y_data) # Converting list to numpy array
        df = pd.DataFrame(X_data, columns = [col_names]) # making a data frame
        df['class'] = y_data # Adding classes column to the dataframe
        df = df.sample(frac=1) # Shuffling the records
        return df
```

Sample code 1. Package to generate training data.

Models

1) Classic LSTM based Model

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. The first standard and basic model in this category is Classic LSTM.

A Classic LSTM (a.k.a Vanilla) is an LSTM model that has a single hidden layer of LSTM units, and an output layer used to make a prediction.

This architecture consists of 3 main components:

- Input Gate
- Output Gate
- Forget Gate

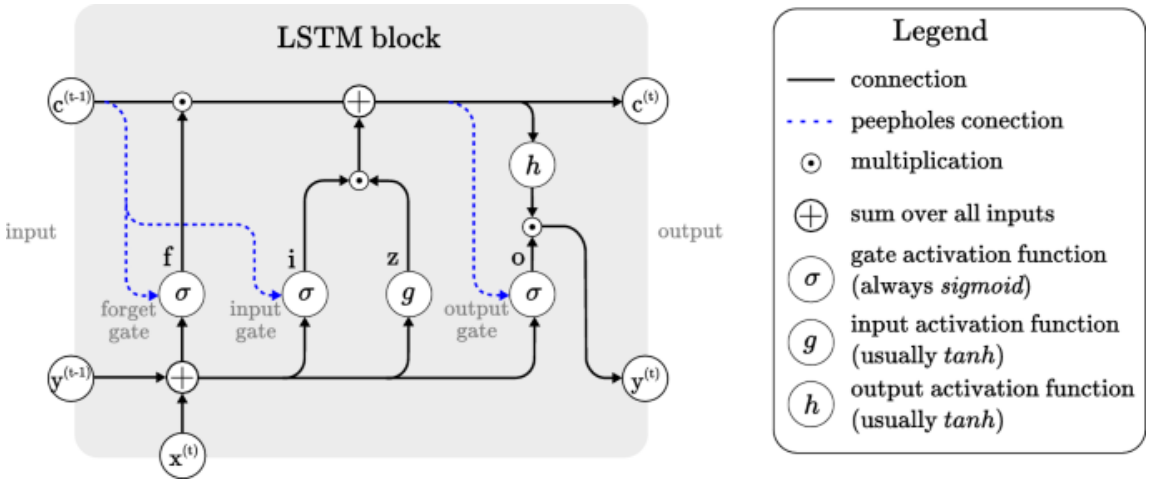


Fig 4. Detailed Functioning of LSTM architecture

The forget gate chooses what values of the old cell state to get rid of, based on the current input data. The two input gates (often denoted *i* and *j*) work together to decide what to add to the cell state depending on the input. *i* and *j* typically have different activation functions, which we intuitively expect to be used to suggest a scaling vector and candidate values to add to the cell state.

Finally, the output gate determines what parts of the cell state should be passed on to the output.

Before we even start building the model, we need to first transform the shape of training and test sets from 2d (samples, features) to 3d (samples, timesteps, features). This is necessary as the LSTM will take in the input timesteps wise not all at once like the traditional Neural Nets. Our model consists of 2 LSTM layers instead of 1 each composed of 50 neurons, a dense layer of 32 neurons. Finally, a dense layer with a soft-max activation function is utilized to find the probability of the type of failure.

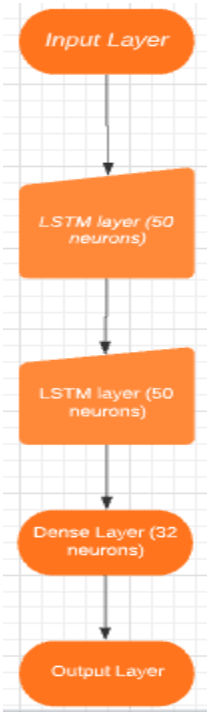


Fig 5. High-Level Representation of Classic LSTM workflow

2) Bi-directional LSTM based architecture

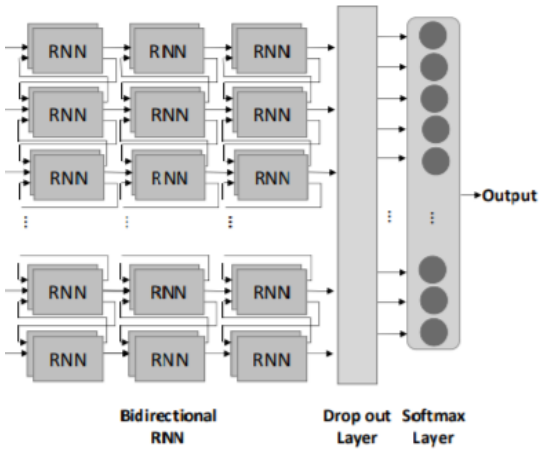


Figure 6. Bi-directional LSTM architecture

At each time step within the sequence, the simple RNN takes advantage of the information learned from past states in generating the current state that will also be propagated to future states. Learning sequential patterns in this way is important in many applications where the temporal component of the input data should not be ignored. However, for identifying the failure, accessing the entire sequence at once could enable not only learning from past states but also from future states as shown in figure **6**. This can be achieved using bidirectional RNN, which incorporates two RNNs trained to make the output decision. The first RNN operates from the beginning to the end of the sequence, while the other operates in the opposite direction.

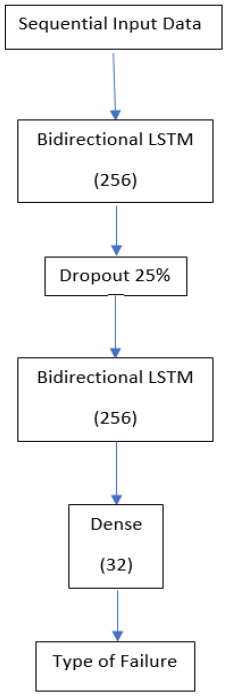


Figure 7. The high-level illustration of Neural Network architecture for failure identification.

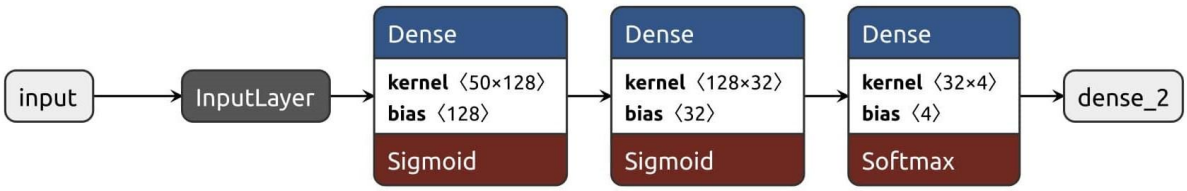
This model will be able to identify the type of failure. Sequential time-series data is passed into the network with two Bidirectional LSTM layers with 256 neurons, a dense layer with 32 neurons and the Dropout layer with a 25% dropout to minimize the chances of overfitting. Finally, a dense layer with a soft-max activation function is utilized to find the probability of the type of failure. Figure 7 shows the high-level illustration of architecture.

Using this model we were able to achieve an accuracy of 55% on vibration data which is not acceptable. The performance of the model is not up to the mark and can be understood that the model is not able to learn much.

3) Artificial Neural Network

Considering the relative failure of our initial LSTM model, we went for a primitive fully connected ANN approach. Using the same segmenter for the training data, we generated 4 million randomly sampled sequences of sequence length 50 each as our initial approach limiting our data to the vibration variable only. Considering the sensor makes measurements at intervals of 500 μs (i.e. 2000 measurements/second), this model, if successful would be able to predict machine status with just the last 50 observations (i.e. without taking into account memory latency, the system would respond to any significant changes within around 25 ms).

Architecture
Apart from the input layer the network has 3 hidden layers as shown below:



Two subsequent hidden layers with **sigmoid** activation function and one final **softmax** layer for classification into the 4 target segments. We did not use dropout regularization in the final model partly because it wasn't producing any better results and because it leads us to overestimate validation accuracy w.r.t. testing set accuracy. We used **categorical cross-entropy** as the loss measure and **Adam** as the optimizer algorithm.

This produced fairly good results both over the validation set and test set (we used a 7:2:1 test-train-validation split). After running training for **100 epochs** with a **batch_size** of **512** samples, we achieved a validation set accuracy of **~ 96.80%**. The test set accuracy turned out to be around **~ 96.88%**. Please refer to the results section for further details.

We ran another neural network for the power signal data with 800,000 samples and 250 sequence length (taking into account the last 250 sensor input). However the accuracy on both validation and test set turned out to be around **74.90%**. So we discarded the model on the grounds that perhaps the power signal data doesn't significantly indicate the status of the machine.

Results

LSTM:

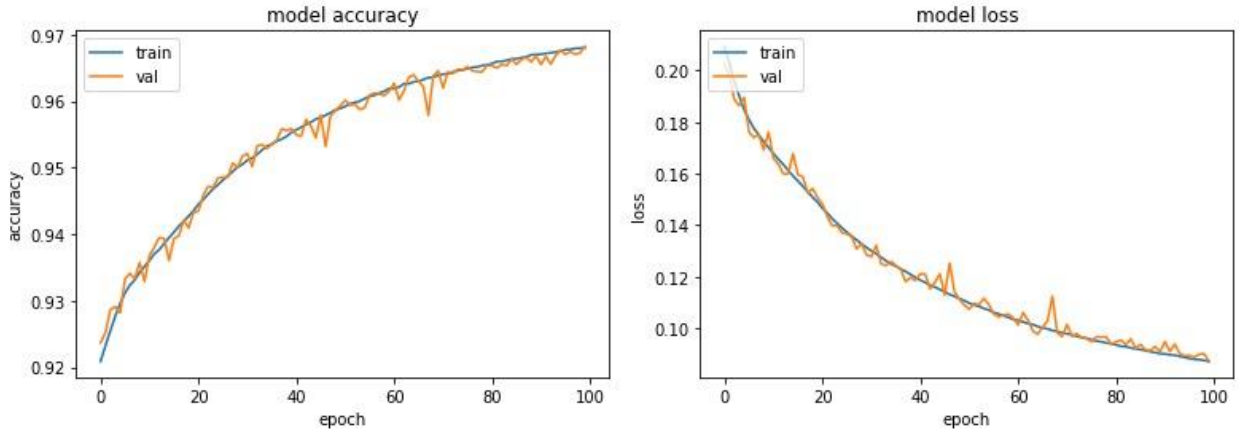
For the vibration model, The training accuracy and test accuracy are 96.5% and 95.8% respectively and their respective losses are 0.0845 and 0.1275 whereas for the power model, The training accuracy and test accuracy is 50.7% and 49.7% respectively and their respective losses were 2.34 and 2.78.

Clearly, This model performed well for the vibration whereas it dipped and isn't accurate enough in the case of power. So, there is further scope for a better model to be designed for giving accurate results on both variables.

ANN

Vibration model

Fig 8.1: the model accuracy and loss for training the ANN for vibration signal data

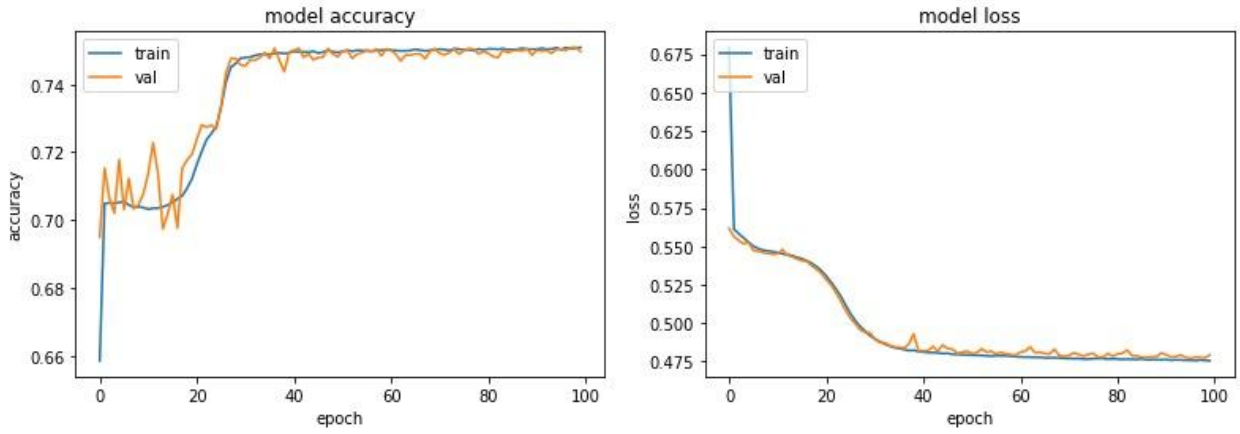


Test set accuracy = 96.88%

Test set loss = 0.0856

Power model

Fig 8.2: the model accuracy and loss for training the ANN for power signal data



Test set accuracy = 74.89%

Test set loss = 0.4786

Evidently, the accuracy is not upto the mark and we were forced to discard the model.

Conclusion

In this work, we have outlined a description of the basic types of faults in bearings. These types form the classes which are outputs of the various deep learning models we have described. Our data consisted of time series data obtained from power and vibration sensors. Data visualizations helped us find visual nuances between the various classes, especially in vibration monitoring data. Both LSTM and ANN models were able to generate good predictions using vibration data, although the same could not be reproduced on power data. Even simple ANNs could be used very effectively for understanding this data. Further work as an extension to what we have shown here can be in different manifolds depending on the technology used. We have used single sensor data here but with the advent of 5G and sensor technology, more data can be procured for analysis and complex neural networks can then provide us more insights into the working of manufacturing equipment. We would also like to investigate the results obtained when we use neural networks in combination with time-domain analysis technologies; which can prove effective in obtaining fault-features from vibration signals. Similarly, frequency domain techniques could provide us with important frequency characteristics while avoiding noise. All these methods provide new avenues to transform the manufacturing industry for the future.