FSM Online Internship Report

Project Title: Remaining Usable Life Estimation(Bearing Dataset)
Domain: Machine Learning
Phase3: Testing & Comparison

Submitted by:
Akanksha Singh
Thapar Institute Of Engineering & Technology
Patiala, Punjab - 147004

Under Mentorship of:
Devesh Tarasia

IITD-AIA Foundation for Smart Manufacturing

[1-June-2022 to 31-July-2022]
Remaining Usable Life Estimation (Bearing Dataset)

The model studied: SVM (But because of it complex nature not used), LSTM, CNN; etc. CNN is by default an expert model for correctly extracting features from unprepared data. Feature extraction is quite important in data-driven methods & regression problems because it is helpful in understanding more insights about data & it also highly affects the end result, but feature extraction techniques are a complex process. So to avoid complexity in the work, the researchers used CNN because it has the ability to extract the feature directly from the input image.

DIFFERENT LIBRARIES & FUNCTIONS USED

Code for processing data samples can get messy and hard to maintain; we ideally want our dataset code to be decoupled from our model training code for better readability and modularity. PyTorch provides two data primitives: *torch.utils.data.DataLoader* & *torch,utils.data.Dataset* that allows you to use pre-loaded datasets as well as your own data.
Dataset stores the samples and their corresponding labels
DataLoader wraps an iterable around the *Dataset*

conv1d is used when you slide your convolution kernels along 1 dimensions (i.e. you reuse the same weights, sliding them along 1 dimensions), whereas tf. layers. conv2d is used when you slide your convolution kernels along 2 dimensions (i.e. you reuse the same weights, sliding them along 2 dimensions).

Filters represents the number of filters that should be learnt by the convolutional layer. From the schematic drawing above, you should understand that each filter slides over the input image, generating a "feature map" as output.

The kernel size represents the number of pixels in height and width that should be summarized, i.e. the two-dimensional width and height of the filter.

The stride tells us how the kernel jumps over the input image. If the stride is 1, it slides pixel by pixel. If it's two, it jumps one pixel. It jumps two with a stride of 3, and so on.

The padding tells us what happens when the kernels/filters don't fit, for example because the input image has a width and height that do not match with the combination of kernel size and stride.

Depending on the backend you're using Keras with, the *channels* (each image has image channels, e.g. 3 channels with Red-Green-Blue or RGB) are in the *first* dimension or the *last*. Hence, the data format represents whether it's a channel first or channels last approach. With recent versions of Keras, which support TensorFlow only, this is no longer a concern.

If you're using dilated convolutions, the dilation rate can be specified as well.

The activation function to which the linear output of the Conv2D layer is fed to make it nonlinear can be specified too.

A bias value can be added to each layer in order to scale the learnt function vertically. This possibly improves training results. It can be configured here, especially if you *don't* want to use biases. By default, it's enabled.

The initializer for the kernels, the biases can be configured too, as well as regularizers and constraints

Batch normalization is a technique to standardize the inputs to a network, applied to ether the activations of a prior layer or inputs directly. Batch normalization accelerates training, in some cases by halving the epochs or better, and provides some regularization, reducing generalization error.

DATASET

The already processed data is divided into Training and validation data in ration 9:1. First, we preprocess the train data & using that preprocessed train data, draw out important features, & implemented algorithms to train the model to predict. Using validation data, the model validation can be performed. During data preprocessing, we divide the learning dataset into train data + validation data. Validation data is the set of data that is separated from learning dataset to validate or check the trained model.
10% of Learning data is kept for validation & remaining 90% is used for training the model. Validation data is used to validate the model whether the model is working correctly or not on the data which the model is not trained.
One of the major reasons we need validation data is to make sure that our model is not over-fitted to the data in the learning dataset.

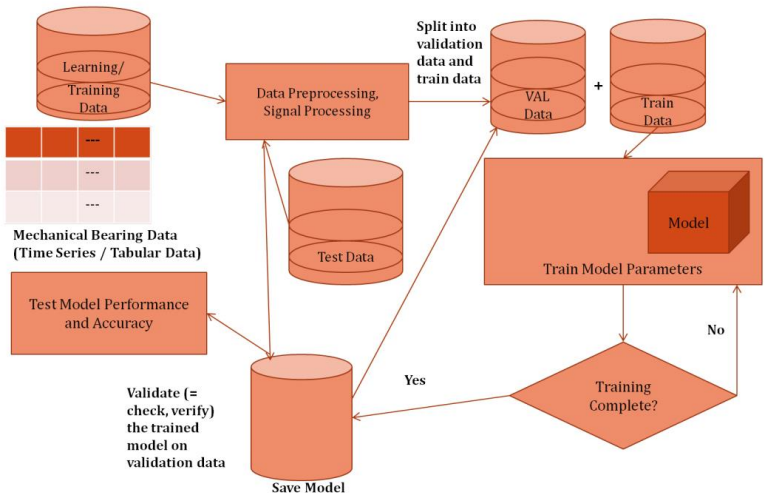

Fig1: Modules Flow Diagram

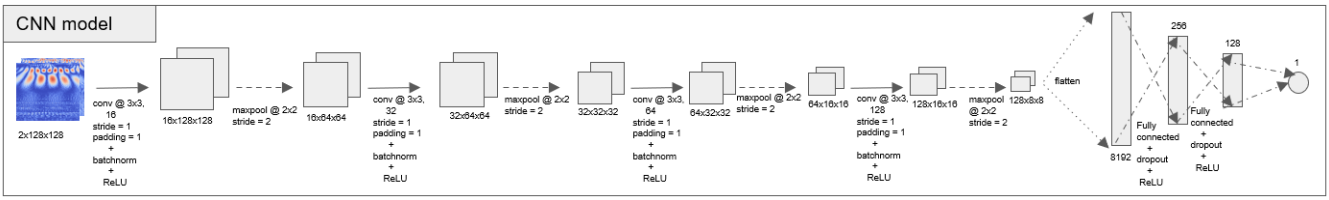

Fig2: CNN Model

CNN + LSTM Model

Input data is still represented as 2D CWT images i.e. time-frequency CWT feature images, but a window of pre-define frames es (say 5) is used as a sequence. First passed through a CNN architecture for encoding and then encoded feature vector sequence is passed through LSTM architecture, hence CNN + LSTM architecture.

IMPLEMENTATION

- Splitting the learning data to train and validation (90:10 ration). Train data to train the model & validation data to validate the model. After splitting both data were stored in separate pickle files.

- Imported torch Dataset & DataLoader for easy accessing of data. Torch provides many tools to makes the data loading into the model easy. DataLoader helps in easy feeding the data into the model, and also makes the code more readable.

- Initially we calculated the fault probability with the pure CNN model

- To train the model, we switched from CPU device to GPU device in collaboratory in order to reduce the
  training time. We utilized GPU for computation with CUDA tensor.

- During training, first we calculated the loss function to measure the difference between the actual expected value and the model predicted value. The loss function evaluates how well our algorithm performs on our dataset.

- If our model predictions are totally off i.e. wrong, our loss function will output a higher number. If our predictions are pretty good, our loss function will output a lower number.

- loss function will tell us whether our model predicting correctly or not

- Since RUL Prediction is a regression problem, MSE (Mean Squared Error) Loss is used. MSE Loss is estimated as MSE loss is the average of squared differences between actual expected values and predicted values.

- Loss helps us to understand how much the predicted value differs from the actual value. We then use this loss to train our network model such that it performs better.

- We applied Adam (Adaptive Moment Estimation) optimizer or optimization algorithm for reducing the loss and providing the most accurate results possible. Implemented a learning rate scheduler to improve the optimization process.

- The learning rate controls how much to change the model in response to the estimated error each time the model weights are updated.

- Epochs are the batches of data samples used to train the artificial neural network. Calculated the loss on train data and validation data for 30 random epochs.

- The blue dots represent predicted fault probability values of train data, the red dots represent predicted fault probability values of validation data, and the black line represent expected failure probability values. Expected values are the ideal values.

- In Fig 4 you can observe that the distance between the val loss curve & train loss curve decreases from model 1 to model 3. This is because in model1 is the curve after first training the model and model 3 is curve after training the model 3rd time. Model3 is overfitted as train curve is a straight line.

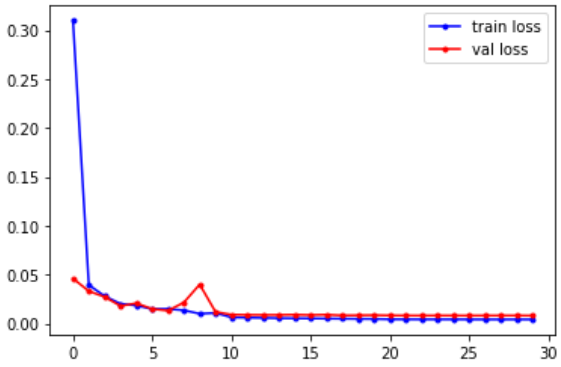

Fig3: CNN Model (Val Loss & Train Loss Curve)

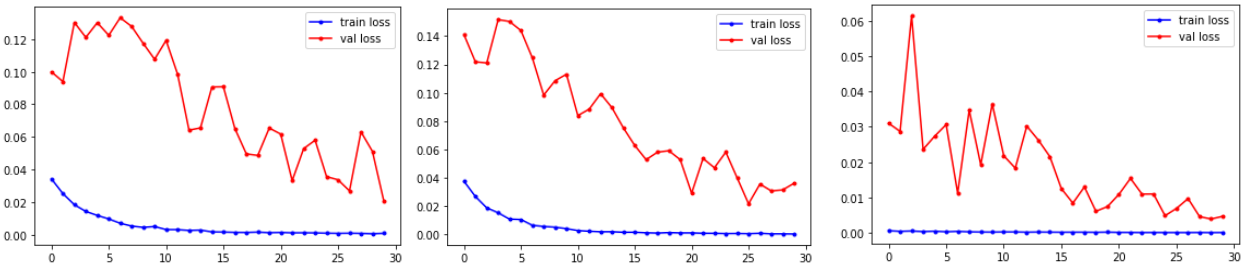

Fig4: CNN + LSTM Model (Val Loss & Train Loss Curve) (Mode11, Model2, Model3)

IMPLEMENTATION:

1. The entire exploratory analysis is performed in a Collaboratory platform using python programming

2. For easy data acSplitting the learning data to train and validation (90:10 ration). Train data to train the model & validation data to validate the model. After splitting both data were stored in separate pickle files.

3. Imported torch Dataset & DataLoader for easy accessing of data. Torch provides many tools to makes the data loading into the model easy. DataLoader helps in easy feeding the data into the model, and also makes the code more readable.

4. Initially we calculated the fault probability with the pure CNN model

5. To train the model, we switched from CPU device to GPU device in colaboratory in order to reduce the training time. We utilized GPU for computation with CUDA tensor.

6. During training, first we calculated the loss function to measure the difference between the actual expected value and the model predicted value. The loss function evaluates how well our algorithm performs on our dataset.

7. If our model predictions are totally off i.e. wrong, our loss function will output a higher number. If our predictions are pretty good, our loss function will output a lower number.

8. Loss function will tell us whether our model predicting correctly or not. Since RUL Prediction is a regression problem, MSE (Mean Squared Error) Loss is used. MSE Loss is estimated as MSE loss is the average of squared differences between actual expected values and predicted values. Loss helps us to understand how much the predicted value differs from the actual value. We then use this loss to train our network model such that it performs better.

9. We applied Adam (Adaptive Moment Estimation) optimizer or optimization algorithm for reducing the loss and to provide the accurate results possible. Implemented learning rate scheduler to improve the optimization process. The learning rate controls how much to change the model in response to the estimated error each time the model weights are updated.

10 Epochs are the batches of data samples used to train the artificial neural network. Calculated the loss on train data and validation data for 30 random epochs

MODEL TEST RESULT OF BEARING1_3

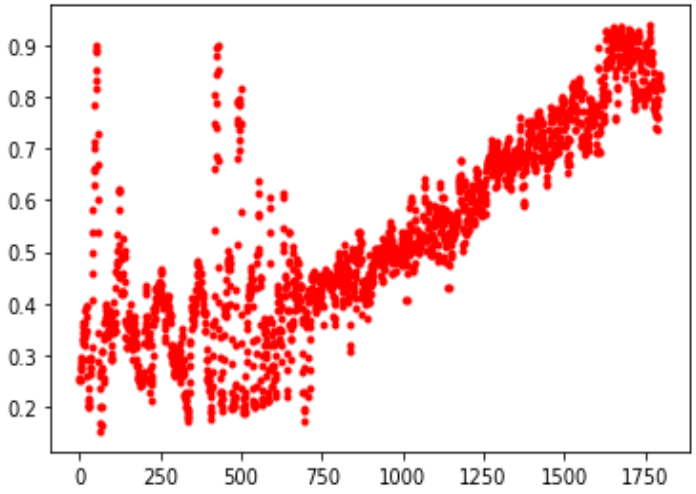

Fig 5: Scatter Plot

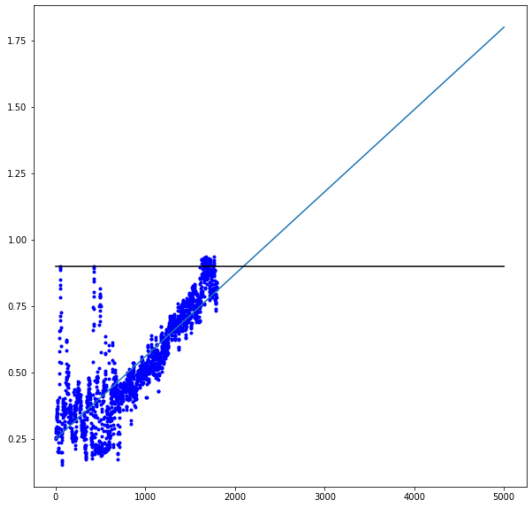

Fig 6: Result 1

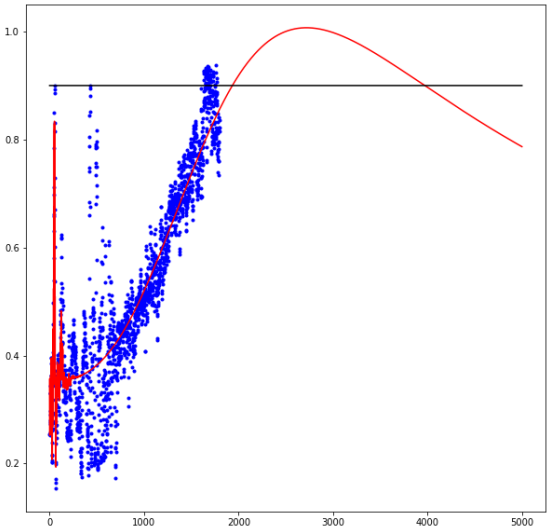

Fig 7: Result 2

MODEL RESULT CALCULATION

Pass the test data till current time index through the architecture and store the predicted fault probability values
Then fit the predicted fault probability values and corresponding time indices through a regressor to extrapolate and find the time index when the fault probability exceeds a predefined threshold to consider occurrence of failure. Here we used Linear regression and Guassian Process Regression (GPR) provided by sklearn python toolbox, and set the fault probability threshold to be 0.9

RUL is estimated from the estimated failure occurrence time and current time as follows,
RUL = predicted fault time - current time

CONCLUSION

From the result obtained the accuracy was good for training and validation dataset but on test dataset it showed up-down curve. So this model is not accurate and more changes in parameters needs to do to get better prediction

CODE IMPLEMENTATION

Data Acquisition
https://colab.research.google.com/drive/1Fn6XKVbQKvGmO9H2xZhRweTtaBH9jHm5?usp=sharing

Data Preprocessing
https://colab.research.google.com/drive/16Pw5r-WN3psy-hVIZWhDRB0NvnPcmopd?usp=sharing

Data Preparation-1 (Converting 1d to 2d feature)
https://colab.research.google.com/drive/1-nIjz5iDKyPvWZVTThP2VLvF7O7Fh-Zw?usp=sharing

Data Preparation-2 (Preparing the batches of the data)
https://colab.research.google.com/drive/1_5JQ9D4AaEnNmXILdqnrPYiLZS072Ea2?usp=sharing

Model Building/ Data Splitting/ Model Training (CNN) –
https://colab.research.google.com/drive/1KRvniMZyjsh4jWhxC65V4gwIAO7tfSka?usp=sharing

Model Building/ Data Splitting/ Model Training (LSTM) –
https://colab.research.google.com/drive/1fxSweF8YxkfpaxdHdRPygBddQwVxUKfy?usp=sharing

Data Acquisition For Test Dataset
https://colab.research.google.com/drive/1TkigsnE6SZY1uOH0x7704Ugm-gkn6kk_?usp=sharing

Test Data Preparation
https://colab.research.google.com/drive/1vyX2papHwjZtd3FWyiVueJs46aW4cZA6?usp=sharing

Model Testing (LSTM + CNN)
https://colab.research.google.com/drive/1puvqXkWrX2xCdFLSUzwtiEPwjvgqRuHa?usp=sharing

REFERENCE

Bhumireddy Naga Sai Abhiji, Levaku Tharun Sai Reddy, and S. Sharanya "Remaining Useful Life (RUL) Prediction of Mechanical Bearings Using Convolution Neural Network (CNN) and Long Short Term Memory (LSTM)" vol. 27, Issue 5, 2021 pp. 592- 607