Internship Project
Report

Classification of Grain-Oriented Electrical Steel Sheets by Magnetic Barkhausen Noise Using Time Analysis and Selected Deep Learning Algorithms

Submitted by

Romina Aalishah
Electrical Engineering Student
Amirkabir University of Technology(Tehran Polytechnic)

Under the guidance of

Prof. Grzegorz Psuj
Associate Professor

Center for Electromagnetic Fields Engineering and High-Frequency Techniques, Faculty of Electrical Engineering
West Pomeranian University of Technology
ul. Sikorskiego 37, 70-313 Szczecin

Internship 2023

Contents

Chapter 1

Introduction

1.1 Previous Works

Previous Works related to this topic that was done by Prof. Grzegorz Psuj were ”Time-frequency representation of magnetic Barkhausen noise under various measurement conditions”, ”Use of Time-Frequency Representation of Magnetic Barkhausen Noise for Evaluation of Easy Magnetization Axis of Grain-Oriented Steel”, ”Time-Frequency Analysis of Barkhausen Noise for the Needs of Anisotropy Evaluation of Grain-Oriented Steels”, ”Classification of Grain-Oriented Electrical Steel Sheets by Magnetic Barkhausen Noise Using Time-Frequency Analysis and Selected Machine Learning Algorithms”, etc.

1.2 Internship Description

During my internship, the project was to classify grain-oriented electrical steel sheets by magnetic Barkhausen noise using Selected Deep Learning Algorithms. I was in charge of the deep learning code. I worked directly with Prof. Grzegorz Psuj and did the following procedure:

I. Getting Familiar with the Functionality and Theory

II. Using CNN Algorithm and Coding with Python

III. Dataset Preparation and Adding More Features

IV. Result Evaluation

V. Switching to Matlab and LSTM Algorithm

VI. Result Evaluation

1.3 Review of internship experience

I was able to develop my experience with Matlab during my internship. At first, I had difficulties working with Matlab and that was the reason I started to work with Python. Since there happened to be some problems during the training process and it was taking so long to get fixed and also there was some former code done in Matlab because of previous works, I switched to Matlab to use my supervisor experience as well. Despite the problems I had at the beginning due to lack of experience, I found it to be beneficial in the development of my knowledge of LSTM and also Matlab coding.

Chapter 2

Work Done

2.1 Literature Review

During the first weeks, I tried to get familiar with the subject and how the data were gathered. Then I tried to ask my questions regarding the matter to clarify the process for myself.
During the review, I realized some algorithms like CNN, RNN-LSTM, CRNN, and Autoencoders would probably give better results. So, the main focus was on working with CNN and LSTM.
I also learned what hyperparameters and features can be more effective.

2.2 Experimental setup

The main dataset that was gathered during previous works, consists of 3 rows and 25001 columns. The rows are B, H, and BN signals.
Some parts of the data because of problems during gathering the data were ignored (2nd sample of class 0.14 and 3rd sample of class 0.51).

2.2.1 CNN with Python

Since for the CNN algorithm, it's better for the dimensions to be closer as possible, the dataset got converted to a 69x1087 table. The procedure of conversion was to convert each row to a 23x1087 to keep data for each signal close to each other. Then concatenate these three tables vertically. In the end, the transposition of this matrix got used to consider the time effect in the training process.
To simplify the process of reading the data in the code, the dataset got

separated in 6 folders with the names of classes(0.14, 0.26, 0.34, 0.41, 0.51, 0.65).

2.2.2 LSTM with Matlab

The shape of the dataset isn't important in this part, so the main dataset was used. A code in Matlab was written by My supervisor in which the dataset is given as input and dataset in the format of Datastore is given as the output. To be able to work with different databases and evaluation of different conditions, these options got added to the code to control it easier:

- resampling

  To compare the results and time efficiency among 1x, 2x, and 10x resampled data, this option was added.

- averaging

  To do data augmentation in a logical way and increase accuracy, this option was added. For this purpose, the average of B, H, and BN signal of the combinations of data(each two) in each class was obtained and got added to the database.

- reshaping

  Since at first there was this mindset that there need to be an option for reshaping in the CNN algorithm, this option was added. This option wasn't used during the main code.

- filtering

  The process turned into this way to first train the model on filtered data, and then on unfiltered. So this option was added to further simplify the procedure.

- plotting

  To investigate how the averaging part is working and if it works properly, this part was added so that the average of each two can be seen and evaluated.

Regarding the averaging and resampling order, the primary idea was that first getting average and then resampling would give better results, because of losing less data. So the experiment was to test both types of orders and

compare the results; First to get the average, then resample. Second to get the resample, then average.
By comparing the results, it is concluded that there is less noise with the first way. Also, by analyzing the BN signal in this way, we can see that output from the first way is more favorable to the characteristic of the original signals.

2.3 Training

2.3.1 CNN with Python

In order to there would be no need to install packages and dependencies, I used Google Colab(a platform for Jupyter Notebook).
In the first trials, a code was written to only have an evaluation of the current situation. The procedure is as followed:

- The dataset was uploaded on GoogleDrive. In order to connect to GoogleDrive, there is added some code to ask the user permission.
- Train, test, and validation datasets were specified(70% Train and the rest got halved between test and validation datasets.).
- The model was defined.

  two convolutional layers with activation function "relu" and after each one 2-dimensional MaxPool layer, one flatten layer(this layer converts the 2-dimensional matrix to a 1-dimensional one), and two Dense layers(last Dense layer has the activation function "softmax" for the classification purpose).
- Batch size and epochs were decided.
- A function to shuffle the data and then read them was defined.
- Loss function, optimizer and metrics were decided.

  "CategoricalAccuracy" for metric and "sparse_categorical_crossentropy" for loss function were used; these 2 are for classification approaches.

In the first trials, both the training and validation accuracy was 1. Different approaches were chosen to change these values and find the problem. Approaches like changing batch size, optimizer, number of epochs, number of convolutional layers and their size and dataset dimensions, adding more

data, and using early stopping weren't effective.
One problem that was found during this process was that the same target was assigned to all the data.
The experiment was also done with two classes. The accuracy was about 60% for both train and validation(classes: 0.14, 0.26). Since there is more data in 0.26, it shows that the model predicts the same target for every data.

2.3.2 LSTM with Matlab

At first, I tried to work with Deep Network Designer, but the accuracy was low. Then I was provided with a code that was based on the number of GPUs on the device.
The first test was done with the following training options:

- rmsprop
- lr=0.001
- epochs=300

And the following layers:

- one LSTM layer=256
- dropout=0.5

On the dataset with the following features:

- filtered
- resampled 10x
- augmented with the average of the signals

According to this, 50 epochs would be enough, so the optimization process went with epochs=50.
Several experiments were done to decide which batch size and learning rate would probably give better results. According to the papers and experiments, smaller batch sizes would have more accuracy and from the other side larger batch sizes cause better efficiency in the hardware accelerator process, so the largest among the small batch size span(2 to 32) was decided. Also during these experiments, it was determined that there wouldn't be that much difference among the resampling amounts. So 10x was decided to make the process faster. For the optimizers, Adam and Rmsprop were tested and Adam gave better and more logical results. So the decided parameters are as followed:

- batch size = 32
- epoch = 50
- learning rate = 0.001
- optimizer = Adam

Experiments were done according to these and the following results were achieved in the first place:

- One trial was to compare them in the aspect of time taken in predicting, but there wasn't that much difference and the ways to do this weren't that much certain (there were other things running at the same time with prediction)
- loss difference between train and validation was calculated in order to compare the results, but it was in accordance with the difference in accuracies(the more gap between train and test accuracy shows the more conversion to overfitting. Also, in loss difference is like this).
- The memory needed for the models was calculated. The most was for 2-layer LSTM 256 and was around 3 MB.